Τεχνολογικό Εκπαιδευτικό Ίδρυμα Δυτικής Ελλάδας

Σχολή Διοίκηση και Οικονομία

Τμήμα Πληροφορικής και ΜΜΕ

Πτυχιακή εργασία

Ιστορική αναδρομή στις γλώσσες προγραμματισμού και στην παραγωγή λογισμικού και η εξέλιξη τους μέχρι σήμερα.

Ονοματεπώνυμο: Καρακόιδα Ελένη

Εποπτεύων καθηγητής: Γκοτσίνας Αντώνιος

Πύργος 2016

ΥΠΕΥΘΗΝΗ ΔΗΛΩΣΗ ΠΕΡΙ ΜΗ ΛΟΓΟΚΛΟΠΗΣ

Βεβαιώνω ότι είμαι συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Ακόμα δηλώνω ότι αυτή η γραπτή εργασία

προετοιμάστηκε από εμένα προσωπικά και αποκλειστικά και ειδικά για την συγκεκριμένη πτυχιακή εργασία και ότι θα αναλάβω πλήρως τις συνέπειες εάν η εργασία αυτή αποδειχθεί ότι δεν μου ανήκει.

ΟΝΟΜΑΤΕΠΩΝΥΜΟ ΣΠΟΥΔΑΣΤΗ 1	**ΑΜ**	**ΥΠΟΓΡΑΦΗ**
ΚΑΡΑΚΟΙΔΑ ΕΛΕΝΗ	1069	

ΟΝΟΜΑΤΕΠΩΝΥΜΟ ΣΠΟΥΔΑΣΤΗ 2 *(σε περίπτωση που είναι απαραίτητο)*	**ΑΜ**	**ΥΠΟΓΡΑΦΗ**
.....	.....	.....

ΟΝΟΜΑΤΕΠΩΝΥΜΟ ΣΠΟΥΔΑΣΤΗ 3 *(σε περίπτωση που είναι απαραίτητο)*	**ΑΜ**	**ΥΠΟΓΡΑΦΗ**
.....	.....	.....

Κεφάλαιο 1
Αλγόριθμοι

1.1 Ιστορική αναδρομή στην θεωρία αλγορίθμων[12]

1.1.1 Εισαγωγή

Ένας τρόπος ώστε να αποδώσουμε στην επιστήμη των υπολογιστών κύρος είναι να δείξουμε ότι οι ρίζες της βρίσκονται βαθιά μέσα στην ιστορία και ότι δεν είναι απλά ένα βραχύβιο φαινόμενο. Είναι λογικό λοιπόν, προκειμένου να υπηρετήσουμε αυτό τον στόχο μας, να στραφούμε στα πρώτα σωζόμενα κείμενα της ανθρωπότητας τα οποία αφορούν υπολογισμούς ώστε να κατανοήσουμε το τρόπο με τον οποίο προσέγγιζαν το ζήτημα των υπολογισμών οι άνθρωποι πριν από περίπου 4000 χρόνια. Αρχαιολογικές ανασκαφές στην μέση ανατολή έφεραν στο φως ένα μεγάλο αριθμό από πήλινες πλάκες οι οποίες έχουν καταγεγραμμένους επάνω τους μαθηματικούς υπολογισμούς και οι οποίες μας δίνουν πάρα πολλά και ενδιαφέροντα στοιχεία για το πως αντιμετώπιζαν το ζήτημα των υπολογισμών οι πρώτοι επιστήμονες των υπολογισμών.
Οι πλάκες στις οποίες αναφερόμαστε ανακαλύφθηκαν στην ευρύτερη περιοχή της Μεσοποταμίας, περιοχή ανάμεσα στους ποταμούς Τίγρη και Ευφράτη, εκεί που βρίσκεται το σημερινό Ιράκ, και για να είμαστε περισσότερο συγκεκριμένοι εκεί που θεωρείται ότι η βρισκόταν η αρχαία πόλη της Βαβυλώνας, σημερινή Βαγδάτη. Είναι γραμμένες με μια σφηνοειδή μορφή γραφής η οποία χρονολογείται στο 3000 π.χ. αυτές όμως που έχουν το μεγαλύτερο μαθηματικό ενδιαφέρον χρονολογούνται στην δυναστεία των Χαμουραμπί μεταξύ 1800 και 1600 π.χ.

1.1.2 Τετραγωνικές ρίζες των Βαβυλωνίων

Από τις πλάκες αυτές έχουμε εξάγει το συμπέρασμα ότι οι Βαβυλώνιοι είχαν ανακαλύψει ένα τρόπο υπολογισμού των τετραγωνικών ριζών των αριθμών. Ωστόσο η πλήρης περιγραφή του συγκεκριμένου αλγόριθμου γίνεται για πρώτη φορά από τον Ήρωνα της Αλεξάνδρειας στον πρώτο μ.χ. αιώνα. Ο Ήρων παρατήρηση ότι εάν το χ είναι μια προσέγγιση της τετραγωνικής ρίζας του ν τότε μια καλύτερη προσέγγιση της τετραγωνικής ρίζας δίνεται από τον υπολογισμό του μέσου όρου του χ με το ν/χ και έτσι υπολογίζουμε το χ΄=(χ+ν/χ)/2. Η αρχική τιμή του χ μπορεί να είναι οποιαδήποτε από το σύνολο [1, ν] αν και η διαδικασία συγκλίνει γρηγορότερα αν το χ είναι εγγύτερα του ν. Στην έκδοση του αλγορίθμου που υλοποιούμε παρακάτω δεσμεύσαμε το ν να είναι μεταξύ του 1 και του 4 ώστε να μπορούμε να καθορίσουμε τον αριθμό των απαιτούμενων επαναλήψεων προκειμένου να πετύχουμε υπολογισμό διπλής ακρίβειας. Μια άλλη προσέγγιση θα μπορούσε να είναι η συνεχής επανάληψη έως ότου η απόκλιση μεταξύ δύο διαδοχικών υπολογισμών να βρίσκεται εντός ενός αποδεκτού εύρους ε.

```
function sqrt(n)
   if n < 1 return sqrt(n * 4) / 2
   if 4 <= n return sqrt(n / 4) * 2
   x = (n + 1) / 2
   repeat 5 times
      x = (x + n / x) / 2
   return x
```

---

1 Ancient Babylonian Algorithms By Donald P. Knuth Stanford University

2 Count Like an Egyptian: A Hands-on Introduction to Ancient Mathematics By David Reimer

1.1.3 Πολλαπλασιασμός των αγροτών

Ο αλγόριθμος αυτός είναι ένας αλγόριθμος πολλαπλασιασμού ο οποίος ανακαλύφθηκε από πολλές κοινωνίες καθώς αυτές σημείωναν πρόοδο στα μαθηματικά. Έτσι κατά περίπτωση έχει ονομαστεί[34] ως η Αιγυπτιακή μέθοδος, η Αιθιοπική μέθοδος, η Ρωσική μέθοδος, η Ουκρανική μέθοδος και η κοινή του ονομασία είναι η μέθοδος των αγροτών. Οι πρώτοι εισηγητές του ωστόσο είναι οι Αιγύπτιοι οι οποίοι είναι οι πρώτοι εισηγητές του ιστορικά είναι γνωστό ότι τον χρησιμοποίησαν στους υπολογισμούς κατά την κατασκευή των πυραμίδων. Ο αλγόριθμος ξεκινάει με τους δύο προς πολλαπλασιασμό αριθμούς στην κεφαλή δύο στηλών. Ακολούθως ο αριθμός της αριστερής στήλης διαιρείται με το δύο και το αποτέλεσμα γράφεται διαδοχικά στην επόμενη σειρά της ίδιας στήλης δίχως να λαμβάνεται υπ΄ όψιν το οποιοδήποτε υπόλοιπο έως ότου το αποτέλεσμα να είναι το 1. Ο αριθμός της δεξιάς στήλης πολλαπλασιάζεται με το 2 για όσες φορές υποδιπλασιαστεί ο αριθμός στην αριστερή στήλη. Ακολούθως διαγράφονται και από τη δεξιά στήλη όσοι αριθμοί έχουν στη ίδια σειρά της αριστερής στήλης μονούς αριθμούς. Όσοι απομείνουν στη δεξιά στήλη προστίθενται και το αποτέλεσμα ισούται με το γινόμενο των δύο αρχικών αριθμών.
Όλοι οι υπολογιστές ακόμα και σήμερα οι οποίοι κάνουν την αριθμητική τους στο δυαδικό σύστημα έχουν αυτόν τον αλγόριθμο εντυπωμένο μέσα στα ολοκληρωμένα υπολογιστικά κυκλώματα τους ολισθαίνοντας ψηφία προκειμένου να πετύχουν τους διπλασιασμούς και τουςς υποδιπλασιασμούς.

1.1.4 Τα πυθαγόρια τριπλάσια

Καθώς συνεχίζουμε την περιήγηση μας στο χρονικό των αλγορίθμων φτάνουμε στον πέμπτο π.χ. αιώνα και στον Πυθαγόρα. Ο Πυθαγόρας που ήταν μαθηματικός, φιλόσοφος και αποκρυφιστής ανακάλυψε ότι η τετραγωνική ρίζα του 2 δεν μπορεί να εκφραστεί ως αναλογία δύο ακέραιων αριθμών θεώρησε ότι εντόπιση μια ρωγμή – ατέλεια στην τελειότητα της κατασκευής του κόσμου και σαν συνέπεια όρκισε τους μαθητές και τους ακολούθους του να μην αποκαλύψουν ποτέ το μυστικό αυτό. Ο μύθος μας λέει ότι ο Ίπασσος ένας εκ των ακολούθων του και μαθητής του αποκάλυψε το μυστικό και για αυτό δολοφονήθηκε. Ο Πυθαγόρας απέδειξε ότι για οποιοδήποτε ορθογώνιο τρίγωνο το τετράγωνο της υποτείνουσας πλευράς ισούται με το άθροισμα των τετραγώνων των δύο άλλων πλευρών (Πυθαγόρειο Θεώρημα). Απέδειξε επίσης ότι για οποιοδήποτε ορθογώνιο τρίγωνο για το οποίο οι πλευρές θέλουμε να είναι πρώτοι αριθμοί μεταξύ τους ισχύει το εξής: ξεκινώντας με τους αριθμούς μ και ν οι οποίοι είναι πρώτοι μεταξύ τους μπορεί να προκύψει ορθογώνιο τρίγωνο με πρώτες μεταξύ τους πλευρές με τις παρακάτω διαστάσεις $\mu^2+\nu^2$, 2μν, $\mu^2+\nu^2$.

```
function triple(m, n)
   return m*m - n*n, 2*m*n, m*m + n*n
```

1.1.5 Μέγιστος κοινός διαιρέτης

Φτάνουμε στον τρίτο π.χ. αιώνα και στον Έλληνα μαθηματικό Ευκλείδη. Ο Ευκλείδης είναι ο συγγραφέας του βιβλίου “Στοιχεία”. Ένα βιβλίο που αφορούσε την γεωμετρία καθώς και θεωρήματα σχετικά με τους αριθμούς. Το βιβλίο έγινε διάσημε διότι στηριζόμενο σε πέντε αξιώματα μπορούσε να παράγει όλη την γνωστή γεωμετρία.

---

3 Ancient Babylonian Algorithms By Donald P. Knuth Stanford University

4 Count Like an Egyptian: A Hands-on Introduction to Ancient Mathematics By David Reimer

Ο Ευκλείδης στον 7 τόμο του Στοιχεία παραθέτει ένα αλγόριθμο υπολογισμού του μέγιστου κοινού διαιρέτη μεταξύ δύο ακεραίων. Η λειτουργία του αλγορίθμου του Ευκλείδη στηρίζεται στην επαναλαμβανόμενη αφαίρεση του μικρότερου αριθμού από τον μεγαλύτερο κάτι το οποίο έχει σαν αποτέλεσμα την συνεχή μείωση των δύο αριθμών και άρα εγγυώμενος το τέλος της υπολογιστκής διαδικασίας.

```
function gcd(m, n) # αρχαίο[56]
        if m < n return gcd(m, n-m)
        if n < m return gcd(m-n, n)
        return m
    function gcd(m, n) # σύγχρονο
        if n == 0 return m
        return gcd(n, m % n)
```

Όπως φαίνεται και παραπάνω ο Ευκλείδης χρησιμοποίησε επαναλαμβανόμενες αφαιρέσεις ενώ ο σύγχρονος αλγόριθμος χρησιμοποιεί αριθμητική υπολοίπων. Ο μεγάλος Donald Knuth αποκαλ τον αλγόριθμο αυτό τον παππού όλων των αλγορίθμων γιατί αν και δεν είναι ο παλαιότερος όλων είναι ο πρώτος ο οποίος γράφτηκε σε αυστηρά καθορισμένη γλώσσα.

1.1.6 Η προσέγγιση του π

Θα παραμείνουμε στον τρίτο π.χ. αιώνα μιας και η ιστορική μας αναδρομή θα ήταν κατά πολύ φτωχότερη αν παραλείπαμε τον Αρχιμήδη και τον υπολογισμό του π. Ο Αρχιμήδης από τις Συρακούσες και συγκεκριμένα από το νησί της Σικελίας είναι ένας διάσημος μαθηματικός και εφευρέτης. Παρά το πλήθος των εφευρέσεων του (οδόμετρο, αντλία κλπ) είναι γνωστός για τον μοχλό, μια εφεύρεση η οποία δεν είναι δική του αλλά την οποία είναι ο πρώτος που ερμήνευσε τον τρόπο λειτουργίας του και είναι γνωστή η διάσημη ρήση του *“Δῶς μοι πᾶ στῶ* καὶ *τὰν γᾶν κινάσω”*. Πασίγνωστη επίσης και η ανακάλυψη του σχετικά με το ότι τα αντικείμενα επιπλέουν όταν το βάρος του νερού που εκτοπίζουν είναι μικρότερο από το βάρος τους.

Η αναλογία της περιφέρειας ενός κύκλου ως προς την διάμετρο του είναι σταθερή και γνωστή ως η σταθερά π το οποίο ισούται με 3,141592654. Ο Αρχιμήδης κατάφερε να περιορίσει την τιμή του π στο εύρος $^{223}/_{71} < \pi < {}^{22}/_{7}$ μετρώντας την περίμετρο 2 εξαγώνων, ενός το οποίο ήταν οριακά εντός του κύκλου και ενός το οποίο ήταν οριακά εκτός του κύκλου. Κατόπιν διπλασίασε διαδοχικά τον αριθμό των πλευρών των εγγεγραμμένων και περιγεγραμμένων κανονικών πολυγώνων μέσω της σειρά 6,12,24,48,96...

```
function pi(n) # ο Αρχιμήδης χρησομοποίησε 6
   outer := 3 * sqrt(3)
   inner := outer / 2
   for i from 1 to n
      outer = 2 / (1/outer + 1/inner)
      inner = sqrt(outer * inner)
   return inner, outer
```

Η προσέγγιση του Αρχιμήδη $^{22}/_{7}$ χρησιμοποιείται ακόμα και σήμερα αρκετά συχνά. Η μέθοδος υπολογισμού όμως έχει αντικατασταθεί από μια επέκταση της σειρά κατά Srinivasa Ramanujan ή κάποια παρόμοια σειρά η οποία μας δίνει αποτέλεσμα μετά από μερικά βήματα σε αντίθεση με την

---

5 Ancient Babylonian Algorithms By Donald P. Knuth Stanford University

6 Count Like an Egyptian: A Hands-on Introduction to Ancient Mathematics By David Reimer

μέθοδο του Αρχιμήδη η οποία απαιτεί 27 υπολογιστικά βήματα για υπολογισμό διπλής ακρίβειας.

1.1.7 Πρώτοι αριθμοί

Ακολούθως θα μεταβούμε στο 200 π.χ. και στον Ερατοσθένη. Ο Ερατοσθένης ήταν Έλληνας μαθηματικός, γεωγράφος, αστρονόμος και γεωδαίτης. Υπολόγισε το μήκος της περιφέρειας της γης[78], την απόσταση γης ηλίου και την κλίση του άξονα της γης, επινόησε ένα σύστημα συντεταγμένων με γεωγραφικά μήκη και πλάτη και ήταν ο τρίτος τη τάξη βιβλιοθηκάριος της μεγάλης βιβλιοθήκης της Αλεξάνδρειας.
Επινόησε ένα σύστημα υπολογισμού των πρώτων αριθμών για το οποίο γνωρίζουμε από τα γραπτά του Νικόμαχου μιας και τα γραπτά του Ερατοσθένη έχουν χαθεί. Το κόσκινο του Ερατοσθένη όπως ονομάζεται η μέθοδος αναγνωρίζει τους πρώτους αριθμούς απορρίπτοντας τα πολλαπλάσια και κρατώντας ότι μένει. Το πνεύμα της μεθόδου είναι ότι οι σύνθετοι αριθμοί περνάνε μέσα από το κόσκινο ενώ οι πρώτοι παγιδεύονται σε αυτό. Ας θεωρήσουμε μια λίστα αριθμών από το 2 έως το επιθυμητό όριο ν. Αρχικά θεωρούμε το λ ίσο με το 2 τον πρώτο αριθμό της λίστας και διαγράφουμε από τη λίστα όλα τα πολλαπλάσια του λ που είναι μικρότερα ή ίσα με το ν. Ακολούθως βρίσκουμε τον πρώτο αριθμό στη λίστα μετά τον λ και το αντικαθιστούμε με αυτό. Και επαναλαμβάνουμε έως ότου το λ^2 να είναι μεγαλύτερο του ν. Όσοι αριθμοί παραμένουν στη λίστα είναι πρώτοι αριθμοί.

```
function primes(n)
   sieve := makeArray(2..n, True)
   for p from 2 to n
      if sieve[p]
         output p
         for i from p*p to n step p
            sieve[i] = False
```

Οι μοντέρνοι μαθηματικοί βελτίωσαν τον αλγόριθμο του Ερατοσθένη εξουδετερώνοντας όλους τους μικρούς πρώτους αριθμούς οι οποίοι λόγω των πολλών πολλαπλασίων τους καταναλώνουν τον περισσότερο υπολογιστικό χρόνο. Έτσι το να βγάλουμε εκτός τα πολλαπλάσια του 2 είναι πολλοί απλό. Ακολούθως μπορούμε να φτιάξουμε κύκλους ώστε να πετάμε εκτός τα πολλαπλάσια του 3, 4, 5 και 7 αν και όσο μεγαλύτεροι γίνονται οι κύκλοι αυτοί τόσο δυσκολότερη γίνεται η χρήση τους. Παρόλα αυτά ένα βελτιστοποιημένο κόσκινο του Ερατοσθένη παραμένει γρηγορότερη μέθοδος από πολλές μεθόδους που έχουν προταθεί κατά την σύγχρονη εποχή.

1.1.8 Νεότερη ιστορία

Ερχόμενοι στον 19ο αιώνα έχουμε πυκνά βήματα τα οποία έρχονται σιγά σιγά να φέρουν πιο κοντά τους αλγόριθμους με τις μηχανές. Ουσιαστικά πρόκειται για ένα αγώνα ανακαλύψεων από τους σχετικούς επιστήμονες οι πολλές φορές συμπληρώνουν ο ένας τον άλλον και μας φέρνουν όλο και πιο κοντά στον ηλεκτρονικό υπολογιστή. Έτσι το 1847 ο Gerge Boole εφευρίσκει την δυαδική άλγεβρα την βάση των ηλεκτρονικών υπολογιστών. Στην πραγματικότητα ο Boole με την εισήγηση της δυαδικής άλγεβρας ενώνει τον κόσμο της λογικής με τον κόσμο των υπολογισμών με ένα κοινό συμβολισμό. Το 1879 ο Γερμανός Gottlob Frege εισηγείται την formula language's η οποία είναι μα γλώσσα εξ' ολοκλήρου γραμμένη με ειδικά σύμβολα, φτιαγμένη μόνο για σκέψη και απαλλαγμένη από ρητορικούς καλλωπισμούς. Το 1888 ο Glussepe Peano παρουσιάζει το Αρχές της αριθμητικής με ένα νέο τρόπο παρουσίασης που ουσιαστικά είναι η πρώτη προσπάθεια να παρουσιαστούν τα

---

7 Ancient Babylonian Algorithms By Donald P. Knuth Stanford University

8 Count Like an Egyptian: A Hands-on Introduction to Ancient Mathematics By David Reimer

μαθηματικά αξιώματα σε μια συμβολική γλώσσα. Ακολουθούν το 1913 οι Alfred North Whitehead και Bertrand Russel οι οποίοι απλοποιούν την δουλεία του Frege στο έργο τους Principia Mathematica. Το 1931 ο Kurt Goëdel εισηγείται το παράδοξο του ψεύτη το οποίο μειώνει απολύτως τους κανόνες αναδρομής στους αριθμούς.
Το 1936 ο Emil Post παρουσιάζει τις δράσεις βάση των οποίων κινείται ο υπολογιστής. Ακολουθεί η περιγραφή του "...δύο έννοιες εμπεριέχονται. Αυτή ενός συμβολικού χώρου, στον οποίο η δουλειά η οποία οδηγεί από το πρόβλημα στην απάντηση, πρόκειται να διεκπεραιωθεί και αυτή ενός συνόλου οδηγιών το οποίο δεν δύναται να μετατραπεί."[910]
Το 1937 ακολουθεί ο μεγάλος Alan Turing ο οποίος εισηγείται ένα νέο υπολογιστικό μοντέλο το οποίο λέγεται μηχανή Turing. Είναι άγνωστο το αν ο Turing ήταν ενήμερος για την δουλειά του Post όμως η μηχανή του ξεκινάει ακριβώς με το ίδιο τρόπο που ξεκινάει και του Post, με μια ανάλυση ενός ανθρώπινου υπολογιστή τον οποίο παρακάτω απλοποιεί σε ένα απλό σετ κινήσεων και νοητικών καταστάσεων. Πηγαίνει όμως ένα βήμα παραπέρα από τον Post και ορίζει την μηχανή του σαν ένα μοντέλο υπολογισμού των αριθμών.
Το 1939 ακολουθεί ο j Barkley Rosser με την αποτελεσματική του μέθοδο. Η κατά Rosser αποτελεσματική μαθηματική μέθοδος "...είναι αυτή κάθε βήμα της οποίας καθορίζεται ακριβώς και που θα παράγει την απάντηση σε ένα πεπερασμένο αριθμό βημάτων. Με την ειδική αυτή έννοια έχουν δοθεί μέχρι σήμερα τρεις διαφορετικοί ορισμοί. Η απλούστερη από αυτές δηλώνει (κατά Post και Turing) ουσιαστικά ότι μια αποτελεσματική μέθοδος επιλύσης συγκεκριμένων συνόλων προβλημάτων υπάρχει μόνο εάν κάποιος μπορεί να κατασκευάσει μια μηχανή η οποία ακολούθως θα μπορέσει να λύσει όλα τα προβλήματα του συνόλου χωρίς καμία ανθρώπινη παρέμβαση πέραν της διατύπωσης της ερώτησης και της λήψης της απάντησης. Και οι τρεις ορισμοί είναι ισότιμοι οπότε δεν έχει καμιά σημασία το ποιον χρησιμοποιείς. Το γεγονός επίσης της ισοτιμίας τους είναι από μόνο του ένα πολύ δυνατό επιχείρημα της ορθότητας τους."
Θα κλείσουμε την ιστορική μας αναδρομή στην θεωρία των αλγορίθμων αναφέροντας τον Stephen C. Kleene οποίος το 1943 διατύπωσε την πλέον διάσημη θεωρία του γνωστή και ως θεωρία Turing – Churh."...Προκειμένου να δημιουργήσουμε μια πλήρη αλγοριθμική θεωρία πρέπει να περιγράψουμε μια διαδικασία η οποία να είναι εκτελέσιμη για οποιοδήποτε σύνολο τιμών των ανεξάρτητων μεταβλητών, η οποία διαδικασία φυσικά φτάνει στο τέλος της με τέτοιο τρόπο ώστε από το αποτέλεσμα να μπορούμε να πάρουμε μια ξεκάθαρη τύπου ναι ή όχι απάντηση στην ερώτηση είναι αληθής η αρχική τιμή ή όχι."
Κλείνοντας το ιστορικό μας ταξίδι στη θεωρία των αλγορίθμων συμπεραίνουμε με ευκολία ότι η πληροφορική με την έννοια της επιστήμης των υπολογισμών είναι μάλλον συνώνυμη του ανθρώπου και τον ακολουθεί πιστά από την αρχή της ιστορίας του έως και σήμερα. Οι αλγόριθμοι μάλλον αρχικά για την λύση των προβλημάτων της καθημερινότητας και εν συνεχεία για την τέλεση των περίπλοκων αλλά αναγκαίων υπολογισμών συντροφεύουν την ανθρωπότητα από το λυκαυγές της. Είδαμε ότι η επιστήμη των υπολογισμών και των αλγορίθμων γνώρισε μεγάλη άνθιση στην αρχαιότητα και δεν έμεινε ανεπηρέαστη από τον σκοταδισμό του μεσαίωνα μια περίοδο οπού ιστορικά δεν υπάρχουν ευρήματα τα οποία να αφορούν τους αλγόριθμους και την πρόοδο τους. Και βέβαια βλέπουμε την ραγδαία εξέλιξη της επιστήμης αυτής κατά των 19 και 20 αιώνα καθώς και το πως έγινε η διασύνδεση των υπολογισμών και των υπολογιστών, υπό την καθοδήγηση των πρωτοπόρων της πληροφορικής.

1.2 Αλγόριθμος

1.2.1 Ορισμός

9 Ancient Babylonian Algorithms By Donald P. Knuth Stanford University
10 Count Like an Egyptian: A Hands-on Introduction to Ancient Mathematics By David Reimer

Ως αλγόριθμος ορίζεται μια πεπερασμένη σειρά ενεργειών, αυστηρά καθορισμένων και εκτελέσιμων σε πεπερασμένο χρόνο, που στοχεύουν στην επίλυση ενός προβλήματος. Πιο απλά αλγόριθμο ονομάζουμε μία σειρά από εντολές που έχουν αρχή και τέλος, είναι σαφείς και εκτελέσιμες που σκοπό έχουν την επίλυση κάποιου προβλήματος.

Η λέξη *αλγόριθμος* προέρχεται από μία μελέτη του Πέρση μαθηματικού του 8ου μ.χ. αιώνα Αλ Χουαρίζμι (Abu Ja'far Mohammed ibn Musa Al-Khwarismi), η οποία περιείχε συστηματικές τυποποιημένες λύσεις αλγεβρικών προβλημάτων και αποτελεί ίσως την πρώτη πλήρη πραγματεία άλγεβρας. Πέντε αιώνες αργότερα η μελέτη μεταφράστηκε στα Λατινικά και άρχιζε με τη φράση "Algorithmus dixit ...." (ο Αλγόριθμος είπε...). Έτσι η λέξη αλγόριθμος καθιερώθηκε αργά τα επόμενα χίλια χρόνια με την έννοια «συστηματική διαδικασία αριθμητικών χειρισμών». Τη σημερινή της σημασία την οφείλει στη γρήγορη ανάπτυξη των ηλεκτρονικών υπολογιστών στα μέσα του 20ου αιώνα[11].

Εικόνα 1 Ένα απλό διάγραμμα ροής, το οποίο απεικονίζει τον αλγόριθμο ελέγχου και επισκευής μιας λάμπας η οποία δεν δουλεύει.

Η έννοια του αλγορίθμου γίνεται ευκολότερα αντιληπτή με το παρακάτω παράδειγμα. Αν κάποιος επιθυμεί να γευματίσει θα πρέπει να εκτελέσει κάποια συγκεκριμένα βήματα: να συγκεντρώσει τα υλικά, να προετοιμάσει τα σκεύη μαγειρικής, να παρασκευάσει το φαγητό, να στρώσει το τραπέζι, να ετοιμάσει τη σαλάτα, να γευματίσει, να καθαρίσει το τραπέζι και να πλύνει τα πιάτα. Προφανώς, η προηγούμενη αλληλουχία οδηγεί στο επιθυμητό αποτέλεσμα. Δεν είναι όμως η μοναδική για την επίτευξη του σκοπού, αφού μπορεί να αλλάξει η σειρά των βημάτων (π.χ. πρώτα να ετοιμάσει τη σαλάτα και μετά να στρώσει το τραπέζι). Ωστόσο το νόημα είναι πως η κατάτμηση μιας σύνθετης εργασίας σε διακριτά βήματα που εκτελούνται διαδοχικά, είναι ο πιο πρακτικός τρόπος επίλυσης πολλών προβλημάτων[12].

1.2.2 Δημιουργία αλγορίθμου

Οι αλγόριθμοι θα πρέπει να πληρούν κάποια πρότυπα και να διατυπώνονται με συγκεκριμένο τρόπο.

Έτσι ένας αλγόριθμος πρέπει να ικανοποιεί τα επόμενα κριτήρια[13]:

---

11 Α. Βακάλη, Η. Γιαννόπουλος, Ν. Ιωαννίδης, Χ. Κοίλιας, Κ. Μαλάμας, Ι. Μανωλόπουλος, Π. Πολίτης (2010 - Έκδοση ΙΑ). *Ανάπτυξη Εφαρμογών σε Προγραμματιστικό Περιβάλλον*. Υπουργείο Εθνικής Παιδείας και Θρησκευμάτων - Παιδαγωγικό Ινστιτούτο, σελ. 25-27. ISBN 960-06-1408-3.

12 Α. Βακάλη, Η. Γιαννόπουλος, Ν. Ιωαννίδης, Χ. Κοίλιας, Κ. Μαλάμας, Ι. Μανωλόπουλος, Π. Πολίτης (2010 - Έκδοση ΙΑ). *Ανάπτυξη Εφαρμογών σε Προγραμματιστικό Περιβάλλον*. Υπουργείο Εθνικής Παιδείας και Θρησκευμάτων - Παιδαγωγικό Ινστιτούτο, σελ. 25-27. ISBN 960-06-1408-3.

13 Α. Βακάλη, Η. Γιαννόπουλος, Ν. Ιωαννίδης, Χ. Κοίλιας, Κ. Μαλάμας, Ι. Μανωλόπουλος, Π. Πολίτης (2010 - Έκδοση ΙΑ). *Ανάπτυξη Εφαρμογών σε Προγραμματιστικό Περιβάλλον*. Υπουργείο Εθνικής Παιδείας και Θρησκευμάτων

- Καθοριστικότητα
- Περατότητα
- Αποτελεσματικότητα
- Επεκτασιμότητα
- Να έχει είσοδο δεδομένων, επεξεργασία και έξοδο αποτελεσμάτων

Καθοριστικότητα - Definiteness

Κάθε κανόνας του ορίζεται επακριβώς και η αντίστοιχη διεργασία είναι συγκεκριμένη. Κάθε εντολή πρέπει να καθορίζεται χωρίς καμία αμφιβολία για τον τρόπο εκτέλεσής της. Π.χ. Σε μία διαίρεση να λαμβάνεται υπόψη και η περίπτωση όπου ο διαιρετέος λαμβάνει μηδενική τιμή. Τυπικές περιπτώσεις η διαίρεση με το μηδέν, υπόριζος ποσότητα αρνητική, κλπ. Προβλήματα καθοριστικότητας αντιμετωπίζονται συχνά με τη λογική της επιλογής, δηλ. Αν α>0 τότε ..... αλλιώς .....

Περατότητα - Finiteness

Κάθε εκτέλεση είναι *πεπερασμένη*, δηλαδή τελειώνει ύστερα από έναν πεπερασμένο αριθμό διεργασιών ή βημάτων. Μία διαδικασία που δεν τελειώνει μετά από συγκεκριμένο/πεπερασμένο αριθμό βημάτων λέγεται απλά υπολογιστική διαδικασία.

Αποτελεσματικότητα - Effectiveness

Είναι *μηχανιστικά* αποτελεσματικός, δηλαδή όλες οι διαδικασίες που περιλαμβάνει μπορούν να πραγματοποιηθούν με ακρίβεια και σε πεπερασμένο χρόνο "με μολύβι και χαρτί". Κάθε μεμονωμένη εντολή του αλγορίθμου να είναι απλή (και όχι σύνθετη). Δηλαδή μία εντολή δεν αρκεί να έχει ορισθεί αλλά πρέπει να είναι και εκτελέσιμη.

Είσοδος δεδομένων - Input

Κατά την εκκίνηση εκτέλεσης του αλγορίθμου καμία, μία ή περισσότερες τιμές δεδομένων πρέπει να δίνονται ως είσοδοι στον αλγόριθμο. Η περίπτωση που δε δίνονται τιμές δεδομένων εμφανίζεται όταν ο αλγόριθμος δημιουργεί και επεξεργάζεται κάποιες πρωτογενείς τιμές με τη βοήθεια συναρτήσεων παραγωγής τυχαίων αριθμών ή με τη βοήθεια άλλων απλών εντολών.

Έξοδος αποτελεσμάτων - Output

Δίδει τουλάχιστον ένα μέγεθος ως αποτέλεσμα που εξαρτάται κατά κάποιο τρόπο από τις αρχικές εισόδους.Ο αλγόριθμος πρέπει να δημιουργεί τουλάχιστον μία τιμή (δεδομένων) ως αποτέλεσμα προς το χρήστη ή προς ένα άλλο αλγόριθμο.

Περιγραφή και αναπαράσταση

Τέσσερις είναι οι βασικοί τρόποι αναπαράστασης ενός αλγορίθμου[14]:

1. Ελεύθερο κείμενο, που αποτελεί τον πιο αδόμητο τρόπο παρουσίασης αλγορίθμου. Ελλοχεύει η δημιουργία μιας μη εκτελέσιμης κατάστασης παραβιάζοντας έτσι το κριτήριο της αποτελεσματικότητας.
2. Διάγραμμα ροής, που συνιστά έναν πιο γραφικό τρόπο παρουσίασης του αλγορίθμου. Η

- Παιδαγωγικό Ινστιτούτο, σελ. 25-27. ISBN 960-06-1408-3.

14 Α. Βακάλη, Η. Γιαννόπουλος, Ν. Ιωαννίδης, Χ. Κοίλιας, Κ. Μαλάμας, Ι. Μανωλόπουλος, Π. Πολίτης (2010 - Έκδοση ΙΑ). *Ανάπτυξη Εφαρμογών σε Προγραμματιστικό Περιβάλλον*. Υπουργείο Εθνικής Παιδείας και Θρησκευμάτων - Παιδαγωγικό Ινστιτούτο, σελ. 28-29. ISBN 960-06-1408-3.

χρήση διαγραμμάτων ροής δεν είναι η πλέον ενδεδειγμένη λύση για ένα πρόβλημα και για αυτό χρησιμοποιούνται σπάνια στη βιβλιογραφία.

3. Φυσική γλώσσα που εκτελείται κατά βήματα. Σε αυτή την περίπτωση μπορεί να παραβιαστεί το κριτήριο του καθορισμού μεταξύ των βημάτων.
4. Κωδικοποίηση του αλγορίθμου σε ψευδογλώσσα ή γλώσσα προγραμματισμού. Έτσι ο αλγόριθμος παρουσιάζεται πιο συνοπτικός, συμπαγής ενώ πληρεί και τις προϋποθέσεις του Δομημένου προγραμματισμού.

1.2.3 Βασικές εντολές

Δομή ακολουθίας

Η δόμηση των διαδικασιών σε τέτοια μορφή, έτσι ώστε οι διαδικασίες να εκτελούνται με τη σειρά από τον υπολογιστή[15].

Δομή επιλογής

Η προγραμματιστική δομή που περικλείει τον έλεγχο μιας συνθήκης και μία ή δύο ομάδες εντολών. Από τις ομάδες των εντολών εκτελείται η πρώτη, αν ισχύει η συνθήκη, ή αν υπάρχει και δεύτερη ομάδα εντολών εκτελείται η δεύτερη αν δεν ισχύει η συνθήκη. Με τον όρο συνθήκη εννοούμε δυο όρους ίδιου τύπου και ανάμεσα τους ένας τελεστής σύγκρισης. Με τον όρο τελεστή σύγκρισης εννοούμε ένα από τα σύμβολα < , > , <> , <= , >= , =. Το αποτέλεσμα της σύγκρισης των όρων (νοείται εφόσον οι όροι έχουν κάποια τιμή , δηλαδή δεν περιέχουν την τιμή null) είναι η αλγεβρική τιμή Αληθής (**True**) ή Ψευδής (**False**). Οι όροι μπορεί να είναι μεταβλητές ή σταθερές[16].

Δομή επανάληψης

Η προγραμματιστική δομή που περικλείει τον συνεχή έλεγχο μίας συνθήκης και μία ομάδα εντολών. Οι εντολές εκτελούνται ανάλογα με την δομή της επανάληψης. Υπάρχουν τριών ειδών επαναλήψεις[17].

- **Επανάλαβε εφόσον**. Σε αυτή την δομή επανάληψης ελέγχεται πρώτα η συνθήκη και εφόσον ισχύει (δίνει τιμή αληθή) εκτελείται η ομάδα εντολών.
- **Επανάλαβε μέχρι**. Σε αυτή την δομή επανάληψης εκτελείται η ομάδα εντολών , στην συνέχεια ελέγχεται αν ισχύει η συνθήκη και εφόσον **ΔΕΝ ισχύει** (δίνει τιμή ψευδής) εκτελείται ξανά η ομάδα εντολών.
- **Για Ν φορές επανάλαβε.** Σε αυτή την δομή επανάληψης εκτελείται η ομάδα εντολών Ν φορές όπου Ν είναι αριθμός θετικός ακέραιος.

---

15 Α. Βακάλη, Η. Γιαννόπουλος, Ν. Ιωαννίδης, Χ. Κοίλιας, Κ. Μαλάμας, Ι. Μανωλόπουλος, Π. Πολίτης (2010 - Έκδοση ΙΑ). *Ανάπτυξη Εφαρμογών σε Προγραμματιστικό Περιβάλλον*. Υπουργείο Εθνικής Παιδείας και Θρησκευμάτων - Παιδαγωγικό Ινστιτούτο, σελ. 30-32. ISBN 960-06-1408-3.

16 Α. Βακάλη, Η. Γιαννόπουλος, Ν. Ιωαννίδης, Χ. Κοίλιας, Κ. Μαλάμας, Ι. Μανωλόπουλος, Π. Πολίτης (2010 - Έκδοση ΙΑ). *Ανάπτυξη Εφαρμογών σε Προγραμματιστικό Περιβάλλον*. Υπουργείο Εθνικής Παιδείας και Θρησκευμάτων - Παιδαγωγικό Ινστιτούτο, σελ. 32-35. ISBN 960-06-1408-3.

17 Α. Βακάλη, Η. Γιαννόπουλος, Ν. Ιωαννίδης, Χ. Κοίλιας, Κ. Μαλάμας, Ι. Μανωλόπουλος, Π. Πολίτης (2010 - Έκδοση ΙΑ). *Ανάπτυξη Εφαρμογών σε Προγραμματιστικό Περιβάλλον*. Υπουργείο Εθνικής Παιδείας και Θρησκευμάτων - Παιδαγωγικό Ινστιτούτο, σελ. 39-48. ISBN 960-06-1408-3.

1.2.4 Τυποποιημένοι αλγόριθμοι

Οι αλγόριθμοι είναι σημαντικοί γιατί σχετίζονται άμεσα με τον τρόπο με τον οποίο οι υπολογιστές επεξεργάζονται δεδομένα και παράγουν πληροφορίες. Ένα πρόγραμμα υπολογιστών είναι ουσιαστικά ένας αλγόριθμος που λέει στον υπολογιστή ποια συγκεκριμένα βήματα να εκτελέσει (σε ποια συγκεκριμένη σειρά) προκειμένου να επιτευχθεί ένας συγκεκριμένος στόχος, όπως π.χ. ο υπολογισμός των μισθών των υπαλ18λήλων ή η εκτύπωση των έλεγχων των μαθητών. Κατά συνέπεια, ένας αλγόριθμος μπορεί να θεωρηθεί οποιαδήποτε ακολουθία εντολών που μπορεί να εκτελεσθεί από μια υπολογιστική μηχανή (Turing)[1819].
Χαρακτηριστικά, όταν ένας αλγόριθμος συνδέεται με την επεξεργασία πληροφοριών, τα δεδομένα διαβάζονται από μια συσκευή εισόδου, γράφονται σε μια συσκευή εξόδου, και / ή αποθηκεύονται για την περαιτέρω χρήση. Τα αποθηκευμένα στοιχεία θεωρούνται ως τμήμα της εσωτερικής κατάστασης του συστήματος που εκτελεί τον αλγόριθμο.
Για οποιαδήποτε τέτοια υπολογιστική διαδικασία, ο αλγόριθμος πρέπει να οριστεί αυστηρά: να είναι ορισμένος για όλες τις πιθανές περιστάσεις που θα μπορούσαν να προκύψουν. Δηλαδή οποιαδήποτε υπό όρους βήματα πρέπει να εξεταστούν συστηματικά, και σε κάθε περίπτωση τα κριτήρια πρέπει να είναι σαφή (και υπολογίσιμα).
Επειδή ένας αλγόριθμος είναι ένας ακριβής κατάλογος βημάτων ακριβείας, η σειρά του υπολογισμού θα είναι σχεδόν πάντα κρίσιμη για τη λειτουργία του αλγόριθμου. Οι εντολές συνήθως απαριθμούνται ρητά, και περιγράφονται σαν να ξεκινούν "από την κορυφή" και πηγαίνοντας "προς στο κατώτατο σημείο", μια ιδέα που περιγράφεται τυπικά με τον όρο της "ροής ελέγχου".
Μέχρι τώρα, σε αυτήν η συζήτηση για την τυποποίηση του αλγορίθμου, έχουμε δεχθεί ως βάση τον διαδικασιακό προγραμματισμό. Αυτή είναι και η πιο κοινή αντίληψη, η οποία προσπαθεί να περιγράψει ένα έργο με διακεκριμένα, "μηχανικά" μέσα. Μοναδικός σε αυτήν την αντίληψη των αλγόριθμων είναι ο ρόλος της λειτουργίας ανάθεσης (ο καθορισμός της τιμής μιας μεταβλητής) ο οποίος προέρχεται από τη ιδέα "της μνήμης" σαν πρόχειρο τετράδιο. Δείτε ακόμα το λειτουργικό προγραμματισμό και τον λογικό προγραμματισμό για εναλλακτικές αντιλήψεις για το τι αποτελεί έναν αλγόριθμο.

1.2.5 Εφαρμογή των αλγορίθμων

Οι αλγόριθμοι μπορούν να υλοποιηθούν από προγράμματα ηλεκτρονικών υπολογιστών, μολονότι συχνά σε περιορισμένες μορφές. Ένα λάθος στον σχεδιασμό ενός αλγόριθμου για την λύση ενός

---

18 Σ. Ζάχος, Δ. Φωτάκης. «Μηχανές Turing και Υπολογισιμότητα». Διαφάνειες μαθήματος στην Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών, Εθνικό Μετσόβιο Πολυτεχνείο. Ανακτήθηκε στις 2011-10-06.

19 Παπαδοπούλου Βίκη (2007). «Θεωρία Υπολογισμού και Πολυπλοκότητα». European University of Cyprus. Ανακτήθηκε στις 2011-10-06.

προβλήματος μπορεί να οδηγήσει σε αποτυχίες/βλάβες στο εφαρμοσμένο πρόγραμμα.
Οι αλγόριθμοι δεν υλοποιούνται μόνο ως προγράμματα υπολογιστών, αλλά συχνά επίσης και με άλλα μέσα, όπως π.χ. σε ένα βιολογικό νευρικό δίκτυο, ή σε ένα ηλεκτρονικό κύκλωμα, ή σε μια μηχανική συσκευή.
Η ανάλυση και η μελέτη των αλγορίθμων είναι ένας τομέας τής επιστήμης της πληροφορικής, και ασκείται συχνά αφαιρετικά (χωρίς τη χρήση μιας συγκεκριμένης γλώσσας προγραμματισμού ή άλλη εφαρμογή). Από αυτή την άποψη, μοιάζει με άλλους μαθηματικούς τομείς, συγκεκριμένα στο ότι η εστίαση της ανάλυσης είναι πάνω στις βασικές αρχές του αλγορίθμου, και όχι σε οποιαδήποτε ιδιαίτερη εφαρμογή του. Ένας τρόπος απεικόνισης ένας αλγόριθμου είναι το γράψιμο του ψευδοκώδικα. Άλλοι τρόποι είναι: με ελεύθερο κείμενο , με φυσική γλώσσα περιγράφοντας τα βήματα και με λογικό διάγραμμα .

1.3 Θεωρία Αλγορίθμων[20][21][22]

1.3.1 Επισκόπηση

Η θεωρία των αλγορίθμων μελετά κατά βάση τα μέτρα πολυπλοκότητας των συμβολοσειρών (ή άλλων δομών δεδομένων). Επειδή οι περισσότεροι τύποι μαθηματικών αντικειμένων μπορούν να περιγραφούν με όρους συμβολοσειρών ή σαν το όριο μιας ακολουθίας συμβολοσειρών η θεωρία των αλγορίθμων μπορεί να χρησιμοποιηθεί για την μελέτη μιας ευρύτατης ποικιλίας μαθηματικών αντικειμένων των ακεραίων μη εξαιρουμένων.
Ανεπίσημα, από την σκοπιά της θεωρίας των αλγορίθμων, το πληροφοριακό περιεχόμενο μιας συμβολοσειράς ισούται με το μήκος της όσο το δυνατόν περισσότερο συμπιεσμένης αυτοδύναμης αναπαράστασης της συμβολοσειράς αυτής. Αυτοδύναμη αναπαράσταση συμβολοσειράς αποκαλούμε ουσιαστικά ένα πρόγραμμα, γραμμένο σε κάποια καθορισμένη γλώσσα προγραμματισμού (η οποία δεν έχει σημασία για την θεωρία των αλγορίθμων), το οποίο όταν εκτελεστεί μας δίνει ως εξαγόμενο την αρχική συμβολοσειρά.
Από αυτή την οπτική γωνία, μια εγκυκλοπαίδεια 3000 σελίδων περιέχει ουσιαστικά μέσα της πολύ λιγότερη πληροφορία από ότι θα περιείχαν 3000 σελίδες γεμάτες με τυχαία γράμματα παρά το γεγονός ότι η εγκυκλοπαίδεια έχει πολύ μεγαλύτερη χρησιμότητα. Αυτό συμβαίνει γιατί προκειμένου να ανακατασκευάσει κάποιος μια πλήρη πρόταση από τυχαία γράμματα θα πρέπει να γνωρίζει επακριβώς το τί είναι και τί αναπαριστά το κάθε γράμμα. Στις δομημένες προτάσεις όμως της εγκυκλοπαίδειας κάτι τέτοιο δεν είναι απαραίτητο. Έτσι αν για παράδειγμα από κάθε λέξη της εγκυκλοπαίδειας αφαιρούσαμε όλα τα φωνήεντα κάποιος με βασικές γνώσεις της Ελληνικής γλώσσας θα ήταν σε θέση να αναπαράγει πλήρως τις προτάσεις π.χ. Πς θ πμ στ θτρ.
Σε αντίθεση με την κλασσική θεωρία της πληροφορίας η θεωρία των αλγορίθμων επισήμως αποδίδει αυστηρούς ορισμούς, μιας τυχαίας συμβολοσειράς και μιας τυχαίας άπειρης ακολουθίας, οι οποίοι δεν εξαρτώνται από φυσικές ή φιλοσοφικές διαισθήσεις περί μη αιτιατότητας ή πιθανότητας ύπαρξης.

1.3.2 Ιστορική αναδρομή

Η θεωρία των αλγορίθμων θεμελιώθηκε από τον Ray Solomonoff (25/07/1926 – 07/12/2009) ο

20 Vitanyi, P. "Obituary: Ray Solomonoff, Founding Father of Algorithmic Information Theory"
21 Solomonoff, R., "A Preliminary Report on a General Theory of Inductive Inference
22 Yongge Wang: Randomness and Complexity. PhD Thesis, 1996

οποίος είναι ο εισηγητής της έννοιας της αλγοριθμικής πιθανότητας καθώς επίσης και ο πρωτοπόρος του κλάδου της τεχνητής νοημοσύνης που βασίστηκε στην δυνατότητα της μηχανής να μαθαίνει, να προβλέπει και να πιθανολογεί.

Ο Solomonoff πρωτοεισηγήθηκε την έννοια της αλγοριθμικής πιθανότητας το 1960 δημοσιεύοντας εκείνο το θεώρημα στο οποίο στηρίχθηκε η πολυπλοκότητα του Kolmogorov και η θεωρία των αλγορίθμων. Οι βασικές ιδέες στις οποίες στηρίχθηκε η θεωρία των αλγορίθμων αποτελούσαν κομμάτι της εφεύρεσης της αλγοριθμικής πιθανότητας την οποία και εισήγαγε για να αντιμετωπίσει μια σειρά από προβλήματα με την εφαρμογή των κανόνων του Bayer στην στατιστική. Πρωτοπεριέγραψε τα αποτελέσματα της έρευνας του σε ένα συνέδριο στο Caltech το 1960. Η θεωρία των αλγορίθμων αναπτύχθηκε στη συνέχεια αυτόνομα από τους Andrey Kolmogorov το 1965 και Gregory Chaitin το 1966.

Υπάρχουν πολλές παραλλαγές της πολυπλοκότητας του kolmogorov ή της αλγοριθμικής πληροφορίας. Η πλέον ευρέως χρησιμοποιούμενη στηρίζεται στα αυτοοριοθετούμενα προγράμματα και την οφείλουμε κυρίως στον Leonid Levin οποίος την εισηγήθηκε το 1974. Ο Martin Lof επίσης συνέβαλε σημαντικά στην θεωρία της πληροφορίας των άπειρων ακολουθιών. Μια αξιωματική προσέγγιση της θεωρίας των αλγορίθμων με βάση τα αξιώματα του Blum έγινε από τον Mark Burgin σε έγγραφο το οποίο έφερε προς δημοσίευση ο Kolmogorov το 1982. Η αξιωματική[23][24] [25]προσέγγιση περιλαμβάνει άλλες προσεγγίσεις στην θεωρία των αλγορίθμων. Εισάγει την δυνατότητα θεώρησης διαφόρων μέτρων αλγοριθμικής πληροφορίας ως ξεχωριστές περιπτώσεις αξιωματικά ορισμένων μέτρων αλγοριθμικής πληροφορίας. Έτσι αντί να αποδεικνύει παρόμοια θεωρήματα, όπως το βασικό αναλλοίωτο θεώρημα, για κάθε συγκεκριμένο μέτρο είναι δυνατόν να συμπεραίνουμε εύκολα όλα τα αποτελέσματα βασισμένοι σε αντίστοιχο θεώρημα που αποδείχθηκε στο σύνολο των αξιωμάτων. Αυτό είναι ένα γενικό πλεονέκτημα της αξιωματικής προσέγγισης στα μαθηματικά. Η αξιωματική προσέγγιση της θεωρίας των αλγορίθμων προήχθη περεταίρω από τις δημοσιεύσεις του Burgin το 2005

23 Vitanyi, P. "Obituary: Ray Solomonoff, Founding Father of Algorithmic Information Theory"

24 Solomonoff, R., "A Preliminary Report on a General Theory of Inductive Inference

25 Yongge Wang: Randomness and Complexity. PhD Thesis, 1996

Κεφάλαιο 2
Γλώσσες Χαμηλού Επιπέδου

2.1 Γλώσσα Μηχανής

2.1.1 Εισαγωγή

Γλώσσα μηχανής ή κώδικα μηχανής ονομάζουμε ένα σύνολο εντολών οι οποίες εκτελούνται απευθείας από την κεντρική μονάδα επεξεργασίας (CPU) ενός υπολογιστή. Κάθε εντολή εκτελεί μια πολύ συγκεκριμένη εργασία όπως μια φόρτωση, ένα άλμα ή μια αριθμητική ή λογική πράξη σε μια μονάδα δεδομένων που βρίσκονται αποθηκευμένα σε ένα καταχωρητή ή σε μια θέση μνήμης. Κάθε πρόγραμμα το οποίο εκτελείται απευθείας από την κεντρική μονάδα επεξεργασίας είναι φτιαγμένο από μια σειρά τέτοιου είδους εντολών.
Ο αριθμητικός κώδικας μηχανής (όχι ο κώδικας assembly) θεωρείται ως το χαμηλότερο επίπεδο αναπαράστασης ενός μεταγλωττισμένου προγράμματος ή σαν μια πρωτόγονη και εξαρτημένη από το υλικό γλώσσα προγραμματισμού. Αν και είναι δυνατόν να γράψουμε προγράμματα απευθείας σε μορφή αριθμητικού κώδικα μηχανής, διαχείριση κάθε bit ξεχωριστά και ο συνεχής υπολογισμός αριθμητικών διευθύνσεων και σταθερών είναι μια διαδικασία ιδιαιτέρως βαρετή, χρονοβόρα κατά την οποία τα λάθη είναι πολύ εύκολο να συμβούν.
Σχεδόν όλα τα προγράμματα σήμερα είναι γραμμένα σε γλώσσες υψηλού επιπέδου ή σε γλώσσα assembly και μεταφράζονται σε κώδικα μηχανής από εφαρμογές όπως είναι οι μεταφραστές και οι συμβολομεταφραστές. Τα προγράμματα που είναι γραμμένα σε γλώσσες διερμηνείς[26] δεν είναι μεταφρασμένα σε γλώσσα μηχανής αν και οι διερμηνείς τους συνήθως αποτελούνται από απευθείας εκτελέσιμο κώδικα μηχανής ο οποίος έχει παραχθεί από κάποια άλλη γλώσσα υψηλού επιπέδου ή από την γλώσσα assembly.

2.1.2 Εντολές κώδικα μηχανής

Κάθε επεξεργαστής ή γενιά επεξεργαστών έχει το δικό του σύνολο εντολών μηχανής. Οι εντολές είναι πρότυπα από bits τα οποία είναι σχεδιασμένα με τέτοιο τρόπο ώστε να ενεργοποιούν διαφορετικές λειτουργίες της μηχανής. Έτσι το σύνολο των εντολών μηχανής είναι συγκεκριμένο για μια κλάση επεξεργαστών οι οποίοι χρησιμοποιούν την ίδια αρχιτεκτονική. Οι επεξεργαστές οι

26 Such as many versions of BASIC, especially early ones, as well as Smalltalk, MATLAB, Perl, Python, Ruby and other special purpose or scripting languages.

οποίοι είναι διάδοχοι ή έχουν παραχθεί στηριζόμενοι στην αρχιτεκτονική άλλων επεξεργαστών συχνά περιέχουν όλες τις εντολές του συνόλου του κώδικα μηχανής που είχε ο επεξεργαστής πρόγονος και μπορεί να προσθέτουν και κάποιες επιπλέον. Ενίοτε ένας διάδοχος σχεδιασμός θα καταργήσει ή θα αλλάξει το νόημα ορισμένων εντολών μηχανής της αρχικής αρχιτεκτονικής (συνήθως επειδή είναι απαραίτητες για κάποιο άλλο σκοπό) επηρεάζοντας έτσι την συμβατότητα του κώδικα μηχανής σε κάποιο βαθμό. Ακόμα και σχεδόν συμβατοί μεταξύ τους επεξεργατές είναι δυνατόν να παρουσιάσουν έστω και ελαφρώς διαφορετική συμπεριφορά σε κάποιες εντολές ωστόσο αυτό το γεγονός σπάνια αποτελεί πρόβλημα. Τα συστήματα μπορεί επειπλέον να παρουσιάζουν διαφορές και σε άλλα ζητήματα όπως είναι ας πούμε η διάταξη της μνήμης, το λειτουργικό σύστημα ή οι περιφερειακές συσκευές. Μιας και ένα πρόγραμμα και η εκτέλεση του βασίζονται σε αυτούς τους παράγοντες διαφορετικά συστήματα είναι πολύ πιθανόν να μην μπορούν να τρέξουν τον ίδιο κώδικα μηχανής ακόμα και αν χρησιμοποιούν τον ίδιο τύπο επεξεργατεί.
Ένα σύνολο εντολών κώδικα μηχανής μπορεί να έχει όλες τις εντολές με ίδιο μήκος ή με διαφορετικό. Ο τρόπος με τον οποίο τα πρότυπα οργανώνονται ποικίλει έντονα μεταξύ διαφορετικών αρχιτεκτονικών καθώς επίσης και μεταξύ του είδους των εντολών. Οι περισσότερες εντολές μηχανής έχουν ένα ή περισσότερα πεδία τα οποία καθορίζουν τον βασικό τύπο της εντολής (αν για παράδειγμα είναι αριθμητική εντολή, λογική εντολή ή εντολή ολίσθησης κλπ) την πραγματική λειτουργία της εντολής (αν για παράδειγμα πρόκειται για εντολή αθροίσματος ή σύγκρισης) καθώς επίσης και επιπλέον πεδία τα οποία προσδιορίζουν των τύπο των ορισμάτων της εντολής, το τρόπο και το μέγεθος της μνήμης που απαιτούνται για την εκτέλεση της εντολής[27].
Δεν έχουν όλες οι μηχανές ή οι ξεχωριστές εντολές σαφή ορίσματα. Μια μηχανή συσσωρευτής έχει ένα συνδυαστικό αριστερό όρισμα και παράγει ως αποτέλεσμα έναν έμμεσο συσσωρευτή για τις περισσότερες αριθμητικές εντολές. Άλλες αρχιτεκτονικές (όπως η 8086 και η γενιά επεξεργαστών x86) συσσωρευτές κοινών εντολών, με τον συσσωρευτή να θεωρείται σαν ένα από τους γενικούς καταχωριστές από τις εντολές μεγαλύτερου μεγέθους. Μια μηχανή στοίβα έχει τις περισσότερες ή όλες τις μεταβλητές σε μια σιωπηρή στοίβα. Οι εντολές ειδικού σκοπού συχνά δεν έχουν σαφή ορίσματα (για παράδειγμα η CPUID στην x86 αρχιτεκτονική γράφει τις τιμές σε τέσσερις καταχωριστές). Η διαφοροποίηση μεταξύ άμεσων και έμμεσων ορισμάτων είναι πολύ σημαντική για τις γεννήτριες κώδικα μηχανής ιδιαίτερα στην κατανομή του χώρου του μητρώου. Ένας ποιοτικός βελτιστοποιητής κώδικα μπορεί να παρακολουθήσει τόσο τα άμεσα όσο και τα έμμεσα ορίσματα γεγονός το οποίο μπορεί να μας προσφέρει αντικατάσταση και αναδίπλωση σταθερών στους καταχωριστές (ένας καταχωριστής στον οποίο έχει ανατεθεί το αποτέλεσμα μιας σταθερής έκφρασης απελευθερώνεται αντικαθιστώντας την έκφραση με την σταθερή τιμή την οποία έχει), καθώς και άλλες βελτιστοποιήσεις κώδικα.

2.1.3 Προγράμματα

Ως πρόγραμμα υπολογιστή ορίζουμε μια σειρά εντολών οι οποίες εκτελούνται από μια Κεντρική

27 *Bradley Kjell; kjell at ieee dot org. "Immediate Operand".*

Μονάδα Επεξεργασίας (CPU – επεξεργαστής). Ενώ οι απλοί επεξεργαστές μπορούν να εκτελούν εντολές την μία μετά την άλλη, οι επεξεργαστές υπερκλίμακας μπορούν ταυτόχρονα να εκτελέσουν πολλές εντολές.
Η ροή του προγράμματος μπορεί να χειραγωγηθεί από ειδικές εντολές άλματος οι οποίες μεταφέρουν τον έλεγχο προγράμματος σε εντολή η οποία δεν είναι η αμέσως επόμενη από την εντολή που μόλις εκτελέστηκε. Τα υπό προϋπόθεση άλματα εκτελούνται (και άρα ο έλεγχος προγράμματος συνεχίζει σε κάποια άλλη διεύθυνση) ή δεν εκτελούνται (και άρα ο έλεγχος προγράμματος συνεχίζει στην επόμενη εντολή) ανάλογα με το αν ικανοποιείται κάποια συνθήκη ή όχι.

2.1.4 Γλώσσες Assembly

Μια πολύ πιο αναγνώσιμη απόδοση της γλώσσας μηχανής, την οποία αποκαλούμε γλώσσα assembly, χρησιμοποιεί μνημονικούς κώδικες αναφοράς στις εντολές μηχανής αντί να χρησιμοποιεί την αριθμητική τους τιμή. Για παράδειγμα στον επεξεργαστή Zilog Z80, η ετολή μηχανής 00000101, η οποία προκαλεί την μείωση του B καταχωριστή του επεξεργαστή, στην γλώσσα assembly θα γραφόταν ως DEC B.
Η σχέση με τον μικροκώδικα
Σε κάποιες αρχιτεκτονικές υπολογιστών ο κώδικας μηχανής υλοποιείται από ένα περισσότερο θεμελιώδες υποκείμενο στρώμα προγραμμάτων τα οποία ονομάζουμε μικροπρογράμματα τα οποία μας παρέχουν ένα περισσότερο κοινότοπο σύστημα διεπαφής αναφορικά με την γλώσσα μηχανής, κατά μήκος μιας ολόκληρης οικογένειας διαφορετικών μοντέλων υπολογιστών ακόμα και αν υπάρχουν μεταξύ τους μεγάλες διαφορές στις υποκείμενες ροές δεδομένων. Αυτό γίνεται για να διευκολυνθεί η μεταφορά των προγραμμάτων σε γλώσσα μηχανής μεταξύ διαφορετικών μοντέλων. Ένα παράδειγμα τέτοιας χρήσης έχουμε στην οικογένεια υπολογιστών System/360 της IBM και στους διαδόχους τους. Με εύρη ροών δεδομένων από 8 έως 64 bits και παραπέρα, καταφέρνουν να παρουσιάσουν μια κοινή αρχιτεκτονική στο επίπεδο της γλώσσας μηχανής για όλους τους υπολογιστές της οικογένειας.
Χρησιμοποιώντας ένα στρώμα μικροκώδικα για την υλοποίηση ενός εξομοιωτή καταφέρνουμε ένας υπολογιστής να παρουσιάζει την αρχιτεκτονική ενός εντελώς διαφορετικού υπολογιστή. Η γενιά υπολογιστών System/360 χρησιμοποίησε αυτή την δυνατότητα ώστε να καταφέρει να εισαγάγει προγράμματα από παλαιότερες γενιές υπολογιστών της IBM στις νεότερες, για παράδειγμα ένας εξομοιωτής IBM 1401/1440/1460 στον IBM S/360 μοντέλο 40.

2.1.5 Η σχέση με τον ενδιάμεσο κώδικα

Ο κώδικας μηχανής δεν θα πρέπει να συσχετίζεται με αυτό το οποίο αποκαλούμε ενδιάμεσο κώδικα (Bytecode ή p-code) ο οποίος εκτελείται είτε από ένα διερμηνέα είτε μετατρέπεται σε κώδικα μηχανής για γρηγορότερη (άμεση) εκτέλεση. Ο κώδικας μηχανής και ο κώδικας assembly αποκαλούνται κάποιες φορές ως πρωτογενής κώδικας όταν αναφέρονται σε τμήματα κώδικα τα των

οποίων η λειτουργία εξαρτάται από την πλατφόρμα στην οποία πρόκειται να εκτελεστούν, σε χαρακτηριστικά γλωσσών ή σε βιβλιοθήκες προγραμμάτων[28].

2.1.6 Αποθήκευση στην μνήμη

Η αρχιτεκτονική του Harvard είναι μια αρχιτεκτονική υπολογιστών η οποία διαχωρίζει φυσικά την αποθήκευση και τα σημασμένα μονοπάτια για τον κώδικα (οδηγίες) και για τα δεδομένα. Στις μέρες μας πλέον οι περισσότεροι επεξεργαστές υλοποιούν τέτοια ξεχωριστά σημασμένα μονοπάτια για λόγους απόδοσης αλλά στην ουσία δεν υλοποιούν τίποτα περισσότερο από μια τροποποιημένη αρχιτεκτονική του Harvard, ώστε να είναι σε θέση να υποστηρίξουν εργασίες όπως την φόρτωση ενός εκτελέσιμου προγράμματος από τον αποθηκευτικό χώρο ως δεδομένα και ακολούθως να προβούν στην εκτέλεση του. Η αρχιτεκτονική του Harvard έρχεται σε αντίθεση με την αρχιτεκτονική που εισηγήθηκε ο Von Neuman στην οποία τα δεδομένα και ο κώδικας αποθηκεύονται στην ίδια μνήμη η οποία προσπελαύνεται από την κεντρική μονάδα επεξεργασίας επιτρέποντας στο υπολογιστικό σύστημα να εκτελέσει εντολές.
Από την οπτική γωνία μιας διεργασίας ο χώρος του κώδικα είναι το κομμάτι εκείνο του χώρου διευθύνσεων όπου ο προς εκτέλεση κώδικας αποθηκεύεται. Στα πολυεπεξεργαστικά συστήματα αυτό περιλαμβάνει την περιοχή κώδικα του προγράμματος και τις διαμοιραζόμενες βιβλιοθήκες. Σε πολυνηματικά περιβάλλοντα, διαφορετικά νήματα μιας διαδικασίας μοιράζονται μεταξύ τους των χώρο του κώδικα με τον χώρο των δεδομένων, γεγονός το οποίο μειώνει σημαντικά την επιβάρυνση της μεταγωγής σε σύγκριση με μια κατάσταση αλλαγής διαδικασίας.

2.1.7 Αναγνωσιμότητα από τον άνθρωπο

Έχει ειπωθεί ότι ο κώδικας μηχανής είναι τόσο δύσκολο να διαβαστεί ώστε το γραφείο πνευματικής ιδιοκτησίας των Ηνωμένων Πολιτειών της Αμερικής να μην μπορεί να αναγνωρίσει αν ένα πρόγραμμα γραμμένο σε κώδικα μηχανής είναι αυθεντική δουλειά ενός συγκεκριμένου προγραμματιστή ή όχι[29]. Ωστόσο το εν λόγω γραφείο επιτρέπει την κατοχύρωση πνευματικών δικαιωμάτων για το λογισμικό. Ο [30]Douglas Hofstadter συγκρίνει τον κώδικα μηχανής με τον γενετικό κώδικα λέγοντας χαρακτηριστικά:<<...Το να κοιτάζεις ένα πρόγραμμα γραμμένο σε κώδικα μηχανής προσομοιάζει σε πολύ μεγάλο βαθμό με το να κοιτάς ένα μόριο DNA άτομο προς άτομο.[31]>>.

---

28 *"Managed, Unmanaged, Native: What Kind of Code Is This?". developer.com. Retrieved 2008-09-02.*

29 *Pamela Samuelson (Sep 1984). "CONTU Revisited: The Case against Copyright Protection for Computer Programs in Machine-Readable Form" **1984** (4). Duke Law Journal: 663–769. JSTOR 1372418.*

30 *"Copyright Registration for Computer Programs" (PDF). US Copyright Office. August 2008. Retrieved February 23, 2014.*

31 *D. Hofstadter (1980). "Gödel, Escher, Bach: An Eternal Golden Braid": 290.*

2.1.8 Γλώσσες μηχανής

Οι γλώσσες μηχανής είναι κατευθείαν εκτελέσιμες από την κεντρική μονάδα επεξεργασίας. Σχηματίζονται τυπικά σαν πρότυπα από bit τα οποία αναπαριστώνται συχνά στο οκταδικό ή δεκαεξαδικό σύστημα αρίθμησης. Κάθε πρότυπο από bit αναγκάζει τα κυκλώματα της κεντρικής μονάδας επεξεργασίας να εκτελέσουν μια από τις θεμελιώδεις εργασίες του υλικού. Η ενεργοποίηση συγκεκριμένων ηλεκτρικών εισροών (για παράδειγμα το πακέτο ακίδων της κεντρικής μονάδας επεξεργασίας για μικροεπεξεργαστές) και λογικών ρυθμίσεων για τις τιμές κατάστασης της κεντρικής μονάδας επεξεργασίας, ελέγχουν τους υπολογισμούς που κάνει ο επεξεργαστής. Κάθε ξεχωριστή γλώσσα μηχανής απευθύνεται συγκεκριμένα σε μια οικογένεια επεξεργαστών. Κώδικας σε γλώσσα μηχανής γραμμένος για μια οικογένεια επεξεργαστών δεν μπορεί να τρέξει απευθείας σε επεξεργαστές που ανήκουν σε άλλη οικογένεια εκτός και αν οι επεξεργαστές αυτοί έχουν επιπλέον υλικό υλοποιημένο ώστε να υποστηρίξουν την διαφορετική γλώσσα μηχανής (για παράδειγμα οι επεξεργαστές DEC VAX είχαν υλοποιημένο ένα σύστημα συμβατότητας με τους PDP-11 επεξεργαστές). Οι γλώσσες μηχανής καθορίζονται πάντα από τους κατασκευαστές των κεντρικών μονάδων επεξεργασίας και όχι από τρίτους. Η συμβολική έκδοση, η assembly δηλαδή γλώσσα του κάθε επεξεργαστή επίσης στις περισσότερες περιπτώσεις καθορίζεται από τον κατασκευαστή. Παρακάτω ακολουθεί μια λίστα από τα ευρύτερα χρησιμοποιούμενα σύνολα εντολών κώδικα μηχανής.

- ARM
  - Αρχικά 32-bit
  - 16-bit Οδηγίες αντίχειρα (υποσύνολο καταχωρητών που χρησιμοποιούνται)
  - 64-bit (Σημαντική αλλαγή αρχιτεκτονικής, επιπλέον καταχωρητες)
- DEC PDP-6/PDP-10/DECSYSTEM-20
- DEC PDP-11 (επηρέασε τους VAX και M68000)
- DEC VAX
- DEC Alpha
- Intel 8008, 8080 and 8085
  - Zilog Z80
- x86:
  - 16-bit x86, πρώτη φορά χρησιμοποιήθηκε στον Intel 8086
    - Intel 8086 και 8088 (το δεύτερο χρησιμοποιήθηκε και στα πρώτα IBM PC)
    - Intel 80186
    - Intel 80286 (ο πρώτος x86 επεξεργαστής με κατάσταση ασφαλείας, χρησιμοποιήθηκε και στο IBM AT)
  - IA-32, το πρωτοείδαμε στο 80386
  - x86-64 Ο αρχικός προσδιορισμός έγινε από την AMD. Υπάρχουν και παραλλαγές ανάλογα τον προμηθευτεί οι οποίες όμως είναι κατ' ουσίαν ίδιες μεταξύ τους:
    - AMD's AMD64

Intel's Intel 64

IBM System/360 και διάδοχοι, συμπεριλαμβάνεται και η z/Architecture

MIPS

Motorola 6800

Motorola 68000 family (κεντρικές μονάδες επεξεργασίας που χρησιμοποιήθηκαν στα πρώτα Apple Macintosh και στους πρώτους υπολογιστές Sun)

MOS Technology 65xx

6502 (CPU για VIC-20, Apple II, και Atari 800)

6510 (CPU για Commodore 64)

Western Design Center 65816/65802 (CPU για Apple IIGS και (παραλλαγή) Super Nintendo Entertainment System)

National NS320xx

Power Architecture

POWER, πρωτοχρησιμοποιήθηκε στο IBM RS/6000

PowerPC – χρησιμοποιήθηκε στο Power Macintosh και η τεχνολογία επίσης χρησιμοποιήθηκε σε πολλές παλαιότερες παιχνιδομηχανές)

Sun SPARC .

2.1.9 Μόνιτορ κώδικα μηχανής

Ως μόνιτορ κώδικα μηχανής (επίσης γνωστό και σαν μόνιτορ γλώσσας μηχανής) ορίζουμε ένα λογισμικό το οποίο επιτρέπει στον χρήστη να εισάγει εντολές και να αλλάξει τις τοποθεσίες της μνήμης σε έναν υπολογιστή με την δυνατότητα να φορτώσει και να αποθηκεύσει τα περιεχόμενα της μνήμης από και προς την δευτερεύουσα μνήμη. Κάποια πλήρων προδιαγραφών μόνιτορ κώδικα μηχανής παρέχουν και λεπτομερή έλεγχο (έλεγχος βήμα προς βήμα) της εκτέλεσης του προγράμματος σε γλώσσα μηχανής (ένα χαρακτηριστικό το οποίο σε μεγάλο βαθμό μοιάζει με τον απασφαλματωτή των μεταφραστών) καθώς επίσης παρέχουν και δυνατότητες συναρμολόγησης και αποσυναρμολόγησης κώδικα σε επίπεδο απόλυτης μνήμης.

Τα μόνιτορ της γλώσσας μηχανής έγιναν ιδιαιτέρως δημοφιλή κατά την πρώτη περίοδο των οικιακών υπολογιστών κατά τις δεκαετίες 1970 και 1980 σε τέτοιο βαθμό μάλιστα ώστε να αποτελούν υλοποιημένο λογισμικό σε κάποιους υπολογιστές (για παράδειγμα το ενσωματωμένο μόνιτορ κώδικα μηχανής του Commodore 128). Πολλές φορές ολόκληρος ο προγραμμτισμός ενός προγραμματιστή μπορούσε να γίνει σε μια οθόνη συμβολικού συναρμολογητή (assembler). Ακόμη και μετά την έλευση των ολοκληρομένων συναρμολογητών το μόνιτορ κώδικα μηχανής παρέμεινε απαραίτητο ως εργαλείο αποσφαλμάτωσης του κώδικα. Η συνήθης τεχνική ήταν ο ορισμός σημείων παύσης (break points) σε διάφορα μέρη του υπό έλεγχο κώδικα και εκκίνηση της

εκτέλεσης του κώδικα[323334]. Όταν η κεντρικλή μονάδα επεξεργασίας συναντούσε ένα σημείο παύσης η εκτέλεση του υπό έλεγχο προγράμματος πάγωνε και ο έλεγχος μεταφερόταν στο μόνιτορ του κώδικα μηχανής . Υπό κανονικές συνθήκες κάτι τέτοιο θα πυροδοτούσε ένα άδειασμα των καταχωρητών και το μόνιτορ θα περίμενε από τον προγραμματιστή να εισάγει τα δεδομένα που ήθελε αυτός προκειμένου να εκτελέσει τον έλεγχο του προγράμματος. Σε αυτό το σημείο επίσης ο προγραμματιστής θα εξέταζε και το περιεχόμενο της μνήμης και εφ' όσον το επιθυμούσε μπορούσε να αλλάξει και τους καταχωρητές του επεξεργαστή πριν επανεκινήσει το υπό έλεγχο πρόγραμμα.
Καθώς η απόριψη συγγραφής λογισμικού εξ' αρχής και εξ' ολοκλήρου σε γλώσσα assembly έγινε καθολική, η χρήση των μόνιτορ κώδικα μηχανής πέρασε στη λήθη και αποτελεί πλέον κάτι σαν ξεχασμένη τέχνη. Στα περισσότερα συστήματα όπου γίνεται χρήση γλωσσών προγραμματισμού υψηλότερου επιπέδου οι αποσφαλματωτές χρησδιμοποιούνται για να μας δώσουν μια πιο περιληπτική και φιλική εικόνα των διεργασιών στο εσωτερικό ενός προγράμματος. Ωστόσο η χρήση του μόνιτορ του κώδικα μηχανής απαντάται ακόμη ιδιώς στον τομέα της κατασκευής πειραματικών υπολογιστών.

2.1.10 Αρχιτεκτονική μειωμένου συνόλου εντολών

Reduced Instruction Set Computing ή απλώς RISC είναι μια στρατηγική προσέγγιση σχεδιασμού κεντρικών μονάδων επεξεργασίας η οποία βασίζεται στην αντίληψη ότι ένα όσο το δυνατόν πιο απλοποιημένο σύνολο εντολών γλώσσας μηχανής (εν αντιθέση με τα πολύπλοκα σύνολα) είναι δυνατόν να οδηγήσει σε υψηλότερες αποδόσεις όταν συνδυαστεί με αρχιτεκτονικές μικροεπεξεργαστών ικανών να εκτελέσουν αυτές τις λίγες και απλές εντολές κάνοντας χρήση λιγότερων κύκλων επεξεργαστή ανά εντολή[35]. Ο υπολογιστής ο οποίος είναι σχεδιασμένος με βάση αυτή τη προσέγγιση είναι ένας υπολογιστής μειωμένου συνόλου εντολών (Reduced Instruction Set), τον οποίο αποκαλούμε και RISC. Η αντίθετης προσέγγιση αρχιτεκτονική ονομάζεται Complex Instruction Set Computing ή απλά CISC.

Εικόνα 2 Ένας Sun UltraSPARC, ένας μικροεπεξεργαστής RISC

Έχουν γίνει διάφορες εισηγήσεις αναφορικά με τον ακριβή ορισμό του τί είναι RISC, ωστόσο ο γενικός ορισμός συνίσταται στο ότι πρόκειται για ένα σύστημα το οποίο χρησιμοποιεί ένα μικρό και ιδιαιτέρως βελτιστοποιήμενο σετ εντολών μηχανής αντί για ένα πολύπλευρο σετ σαν και αυτά που συναντάμε σε άλου τύπου αρχιτεκτονικές. Ένας άλλο κοινό

32 *R.J. Tocci & L.P. Laskowski (1979). Microprocessors and Microcomputers: Hardware and Software. Prentice-Hall. p. 379. ISBN 9780135813225.*

33 *L.A. Leventhal (1986). 6502 Assembly Language Programming. Osborne/McGraw-Hill. ISBN 9780078812163.*

34 *A.F. Kuckes & B.G. Thompson (1987). Apple II in the Laboratory. UP Archive. p. 93. ISBN 9780521321983.*

35 *Northern Illinois University, Department of Computer Science, "RISC - Reduced instruction set computer"*

χαρακτηριστικό είναι ότι η RISC αρχιτεκτονική χρησιμοποιεί την αρχιτεκτονική φόρτωσης αποθήκευσης[36] όπου η μνήμη είναι προσπελάσιμη μόνο μέσω συγκεκριμένων εντολών και όχι προσβάσιμη μέσα από τμήματα και άλλων εντολών όπως ας πούμε είναι η εντολή add.
Αν και αρκετά συστήματα από τις δεκαετίες του 60 και 70 έχουν αναγνωριστεί ως οι πρόδρομοι της αρχιτεκτονικής RISC οι σύγχρονες προσεγγίσεις της χρονολογούνται στην δεκαετία του 1980. Πιο συγκεκριμένα δύο εργασίες η μία στο πανεπιστήμιο Stanford και η άλλη στο πανεπιστήμιο της California το Stanford είναι οι πλέον σχετιζόμενες με την αύξηση της αποδοχής της αρχιτεκτονικής αυτής. Το σχέδιο που προτάθηκε από την εργασία του πανεπιστημίου του Stanford θα εκφραζόταν εμπορικά σαν η επιτυχημένη αρχιτεκτονική MIPS ενώ η προτεινόμενη από το Berkley RISC αρχιτεκτονική έδωσε το όνομα της στο έργο με το όνομα SPARC. Άλλη μια επιτυχία στον τομέα των RISC αρχιτεκτονικών της εποχής προήλθε από τις προσπάθειες της IBM και οδήγησε τελικά στην πολλή επιτυχημένη αρχιτεκτονική Power. Καθώς τα παραπάνω σχέδια ωρίμασαν μια ευρύτατη γκάμα αρχιτεκτονικών παρόμοιας σύλληψης άνθισε στα τέλη της δεκαετίας του 1980 και κυρίως στις αρχές του 1990, αποτελώντας ένα κυρίαρχο ρεύμα στην αγορά των Unix σταθμών εργασίας όπως επίσης σε ενσωματωμένους επεξεργαστές, λέιζερ εκτυπωτές και άλλα παρόμοια προϊόντα.
Γνωστές RISC σχεδίασης οικογένειες περικλείουν τα DEC Alpha, AMD Am29000, ARC, ARM, Atmel D. Hofstadter (1980). "Gödel, Escher, Bach: An Eternal Golden Braid"AVR, Blackfin, Intel i860 και i960, MIPS, Motorola 88000, PA-RISC, Power (μαζί και το PowerPc), RISC-V, SuperH και SPARC. Στον 21ο αιώνα η χρήση επεξεργαστών βασισμένων στην αρχιτεκτονική ARM βρήκε χρήση σε smart phones, tablets, τόσο με λειτουργικό Android όσο και με λειτουργικό της Apple παρέχοντας έτσι μια ευρύτερη από κάθε άλλη φορά βάση χρήσης επεξεργαστών RISC αρχιτεκτονικής. Οι βασισμένοι στην αρχιτεκτονική RISC επεξεργαστές χρησιμοποιούνται επίσης σε υπερυπολογιστές όπως ο K Computer, ο οποίος είναι ο γρηγορότερος της λίστας TOP500 το 2011, δεύτερος στη λίστα αυτή το 2012 και τέταρτος το 2013, καθώς επίσης και στον Seqoia, γρηγορότερος στη λίστα το 2012 και τρίτος το 2013[37][38], επιδόσεις οι οποίες αποδεικνύουν πέραν πάσης αμφιβολίας την υπεροχή της αρχιτεκτονικής RISC στην σχεδίαση υπολογιστικών συστημάτων.

2.1.11 Αιωρούμενος κώδικας

Ως αιωρούμενο (overhead) κώδικα εννοούμε τον επιπλέον (πλεονασματικό) αντικειμενικό κώδικα ο οποίος παράγεται από ένα μεταγλωττιστή προκειμένου να παράξει κώδικα μηχανής ο οποίος πρόκειται να εκτελεστεί από μια συγκεκριμένη κεντρική μονάδα επεξεργασίας. Αυτός ο κώδικας περιλαμβάνει τις μεταγλωττίσεις γενικών διαπλατφορμικών εντολών και είναι φτιαγμένος για εκτέλεση σε μια συγκεκριμένη πλατφόρμα ή αρχιτεκτονική. Σαν ένα παράδειγμα αιωρούμενου διαπλατφορμικού κώδικα μπορούμε να σκεφτούμε τον κώδικα που θα παραγόταν για να

36 *Flynn, Michael J. (1995). Computer architecture: pipelined and parallel processor design. pp. 54–56. ISBN 0867202041.*
37 *"Japanese 'K' Computer Is Ranked Most Powerful". The New York Times. 20 June 2011. Retrieved 20 June 2011.*
38 *"Supercomputer "K computer" Takes First Place in World". Fujitsu. Retrieved 20 June 2011.*

διαχειριστεί την καταμέτρηση αναφοράς την στιγμή που ο πηγαίος κώδικας θα ήταν γραμμένος σε μια γλώσσα προγραμματισμού υψηλού επιπέδου

2.1.12 Αναλογίες

Η περιοχή η οποία καλύπτεται από ένα σπίτι είναι περισσότερη από την περιοχή που καλύπτουν τα περιβλήματα. Παρομοίως ο αιωρούμενος κώδικας είναι το μέρος του προγράμματος το οποίο δεν απαριθμείται στον πηγαίο κώδικα. Ωστόσο είναι απαραίτητος ώστε το τελικό λογισμικό να δουλέψει απροβλημάτιστα.

2.1.13 Ενδιάμεσος κώδικας μηχανής

Στον προγραμματισμό ηλεκτρονικών υπολογιστών ο ενδιάμεσος κώδικας μηχανής (p-code machine ή αλλιώς portable code machine) είναι μια εικονική μηχανή σχεδιασμένη έτσι ώστε να εκτελεί τον ενδιάμεσο κώδικα (την γλώσσα assembly μιας υποθετική κεντρικής μονάδας επεξεργασίας). Αυτός ο όρος χρησιμοποιείται τόσο γενικά προκειμένου να περιγράψει όλες τις μηχανές αυτού του είδους (όπως είναι η Java Virtual Machine), όσο και για να περιγράψει εξειδικευμένες υλοποιήσεις με την πλέον γνωστή να είναι η p-μηχανή του Pascal -P συστήματος και ειδικότερα η UCSD Pascal υλοποίηση.

2.1.14 Πολύ μεγάλες λέξεις εντολών

Η αρχιτεκτονική πολύ μεγάλων λέξεων εντολών (Very long instruction Word ή VLIW) αναφέρεται σε αρχιτεκτονικές επεξεργαστών οι οποίες προσπαθούν να εκμεταλλευτούν τα οφέλη από τον παραλληλισμό των επιπέδων των εντολών (instruction level parallelism ή αλλιως ILP). Ενώ οι κοινοί επεξεργαστές επιτρέπουν κυρίως στα προγράμματα μόνο να προσδιορίσουν τις εντολές οι οποίες θα εκτελεστούν στη σειρά ένας VLIW επεξεργαστής επιτρέπει στα προγράμματα να προσδιορίσουν ρητά τις εντολές οι οποίες πρόκειται να εκτελεστούν την ίδια χρονική στιγμή (δηλαδή παραλλήλως). Αυτού του είδους η αρχιτεκτονική έχει σκοπό να καταφέρει να επιτύχει μεγαλύτερες επιδόσεις δίχως να έχει να αντιμετωπίσει την πολυπλοκότητα άλλων προσεγγίσεων με κοντινές επιδόσεις.

```
MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER  PAGE    2

 C000                   ORG    ROM+$0000 BEGIN MONITOR
 C000 8E 00 70  START   LDS    #STACK

                ****************************************
                * FUNCTION: INITA - Initialize ACIA
                * INPUT: none
                * OUTPUT: none
                * CALLS: none
                * DESTROYS: acc A

 0013           RESETA  EQU    %00010011
 0011           CTLREG  EQU    %00010001

 C003 86 13     INITA   LDA A  #RESETA   RESET ACIA
 C005 B7 80 04          STA A  ACIA
 C008 86 11             LDA A  #CTLREG   SET 8 BITS AND 2 STOP
 C00A B7 80 04          STA A  ACIA

 C00D 7E C0 F1          JMP    SIGNON    GO TO START OF MONITOR

                ****************************************
                * FUNCTION: INCH - Input character
                * INPUT: none
                * OUTPUT: char in acc A
                * DESTROYS: acc A
                * CALLS: none
                * DESCRIPTION: Gets 1 character from terminal

 C010 B6 80 04  INCH    LDA A  ACIA      GET STATUS
 C013 47                ASR A            SHIFT RDRF FLAG INTO CAR
 C014 24 FA             BCC    INCH      RECIEVE NOT READY
 C016 B6 80 05          LDA A  ACIA+1    GET CHAR
 C019 84 7F             AND A  #$7F      MASK PARITY
 C01B 7E C0 79          JMP    OUTCH     ECHO & RTS

                ****************************************
                * FUNCTION: INHEX - INPUT HEX DIGIT
                * INPUT: none
                * OUTPUT: Digit in acc A
                * CALLS: INCH
                * DESTROYS: acc A
                * Returns to monitor if not HEX input

 C01E 8D F0     INHEX   BSR    INCH      GET A CHAR
 C020 81 30             CMP A  #'0       ZERO
 C022 2B 11             BMI    HEXERR    NOT HEX
 C024 81 39             CMP A  #'9       NINE
 C026 2F 0A             BLE    HEXRTS    GOOD HEX
 C028 81 41             CMP A  #'A
 C02A 2B 09             BMI    HEXERR    NOT HEX
 C02C 81 46             CMP A  #'F
 C02E 2E 05             BGT    HEXERR
 C030 80 07             SUB A  #7        FIX A-F
 C032 84 0F     HEXRTS  AND A  #$0F      CONVERT ASCII TO DIGIT
 C034 39                RTS

 C035 7E C0 AF  HEXERR  JMP    CTRL      RETURN TO CONTROL LOOP
```

2.2 Η γλώσσα assembly

2.2.1 Εισαγωγή

Μία γλώσσα συναρμολογητής (assebly ή assembler συχνά αναγραφόμενη ως asm) είναι μια γλώσσα προγραμματισμού[39] χαμηλού επιπέδου ηλεκτρονικών υπολογιστών ή άλλων προγραμματιζόμενων συσκευώ για την οποία υπάρχει πολύ ισχυρή συσχέτιση (γενικά μια

39 Assembler language, IBM Knowledge center

σχέση ένα προς ένα) ανάμεσα στην γλώσσα και στην αρχιτεκτονική του συνόλου εντολών της γλώσσας μηχανής. Κάθε γλώσσα assembly είναι φτιαγμένη για μια συγκεκριμένη αρχιτεκτονική υπολογιστών, εν αντιθέσι με τις γλώσσες προγραμματισμού υψηλού επιπέδου οι οποίες είναι μεταφέρσιμες δίχως προβλήματα συμβατότητας μεταξύ των διαφόρων αρχιτεκτονικών αλλά απαιτούν την θυσία της μεταγλώτισης ή της μετάφρασης κάθε φορά. Η γλώσσα assembly αποκαλείται επίσης και ως συμβολικός κώδικας μηχανής[40].
Ο κώδικας που είναι γραμμένος σε γλώσσα assembly μετατρέπεται σε εκτελέσιμο κώδικα μηχανής από ένα βοηθητικό πρόγραμμα το οποίο ονομάζουμε συναρμολογητή (assembler). Η διαδικασία της μετατροπής ονομάζεται συναρμολόγηση (assembling) του πηγαίου κώδικα.
Η γλώσσα assembly χρησιμοποιεί μνημονικά για να αναπαραστήσει κάθε εντολή ή λειτουργία κώδικα μηχανής όπως επίσης και για κάθε καταχωρητή, σημαφόρο κλπ της εν λόγω αρχιτεκτονικής. Πολλές από τις λειτουργίες απαιτούν ένα ή περισσότερους τελεστές προκειμένου να αναπαραστήσουν μια πλήρη εντολή και οι περισσότεροι συναρμολογητές μπορούν να χρησιμοποιήσουν εκφράσεις αριθμών και σταθερές με ονόματα όπως επίσης καταχωρητές και ετικέτες σαν τελεστές των εκφράσεων αφαιρώντας από τον προγραμματιστή την ανάγκη να προβαίνει κάθε τόσο σε βαρετούς και επαναλαμβανόμενους υπολογισμούς. Ανάλογα με την αρχιτεκτονική τα παραπάνω στοιχεία που αναφέραμε μπορούν επίσης να συνσυαστούν για τον σχηματισμό συγκεκριμένων εντολών ή μετατοπίσεις χρησιμοποιώντας μετατοπίσεις ή άλλα δεδομένα όπως επίσης και δεσμευμένες διευθύνσεις μνήμης. Πολλοί συναρμολογητές προσφέρουν επιπλέον μηχανισμούς προκειμένου να επικουρήσουν την ανάπτυξη προγραμματων, να ελέξγουν την διαδικασία της συναρμολόγησης και για να βοηθήσουν στην αποσφαλατωση.

2.2.2 Ορολογία

Ένας μακροσυναρμολογητής περιλαμβάνει μια εγκατάσταση μακροεντολών έτσι ώστε ο (παραμετροποιημένος) κείμενο γραμμένο σε γλώσσα assembly να μπορεί να αποκτήσει όνομα ώστε να χρησιμοπιοήται το όνομα αυτό ώστε να μπορεί να εισαχθεί αυτό το κωμμάτι κώδικα assembly μέσα άλλο κώδικα. Από το 1970 και έπειται σχεδόν όλοι οι συναρμολογητές ήταν στην ουσία μακροσυναρμολογητές.
Ένας διαπλατφορμικός συναρμολογητής (cross assembler) είναι ένας συναρμολογητής ο οποίος εκτελείται σε έναν υπολογιστή ή λειτουργικό σύστημα (σύστημα φιλοξενίας) διαφορετικού τύπου ή αρχιτεκτονικής από το σύστημα στο οποίο ο παραγόμενος κώδικας πρόκειται να εκτελεστεί (σύστημα στόχος). Η χρήση των διαπλατφορμικών συναρμολογητών μας παρέχει την δυνατότητα προγραμματισμού συστημάτων τα οποία δεν είναι υπολογιστές γενικής χρήσης όπως ενσωματωμένα συστήματα. Ανάλογα με την χρονική περίοδο ο κώδικας μηχανής που παράγεται από την διαδικασία συναρμολόγησης και σύνδεσης (assembling and linking processes) προωθείται στην συσκευή που πρόκειται να τον εκτελέι (για παράδειγμα ROM, PROM, ή μνήμη φλαση) μέσω

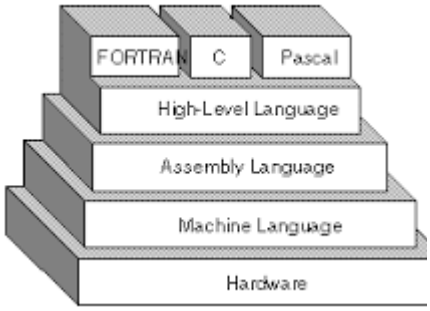
40 *Saxon, James; Plette, William (1962). "Programming the IBM 1401". Prentice-Hall. LoC 62-20615. [use of the term assembly program]*

ενός σειριακού διαύλου, μια καλωδιοταινίας, ενός φορητού μέσου αποθήκευσης ή με όποιον τρόπο διεπαφής προσφέρει το σύστημα στόχο. Η μορφή του παραγόμενου κώδικας μπορεί να είναι είτε ένα ακριβές αντίγραφο (bit by bit copy) του κώδικα μηχανής, ή κάποια αθροιστική μορφή κειμένου του κώδικα αυτού (όπως είναι η SREC ή η Hex της Intel) ή κάποια άλλη αναπαράσταση ανάλογα με τις προδιαγραφές του συστήματος στόχου.

Ένας συναρμολογητής υψηλού επιπέδου είναι ένα πρόγραμμα το οποίο παρέχει τις αφαιρέσεις της γλώσσας και το οποίο έχει συνήθως σχέση με γλώσσες υψηλού επιπέδου, παρέχοντας προχωρημένες δομές ελέγχου (όπως οι IF/THEN/ELSE, DO CASE, κλπ.), και υψηλού επιπέδου αφαιρετικούς τύπους δεδομένων όπως είναι τα structures/records, unions, classes και sets.

Ο μικροσυναρμολογητής είναι ένα πρόγραμμα το οποίο μας βοηθάει να φτιάξουμε ένα μικροπρόγραμμα (τα οποία αποκαλούμε firmware) με σκοπό να ελέγξουμε την χαμηλού επιπέδου λειτουργία ενός υπολογιστή.

Ένας μετάσυναρμολογητής είναι ο όρος που χρησιμοποιείται από κάποιους κύκλους για να περιγράψει ένα πρόγραμμα το οποίο δέχεται την συντακτική και σημασιολογική περιγραφή μιας γλώσσας συναρμολογητή και εν συνεχεία παράγει ένα συναρμολογητή για την γλώσσα αυτή[41].



Εικόνα 4: Η θέση της Assembly σε σχέση με τις άλλες γλώσσες προγραμματισμού

2.2.3 Έννοιες κλειδιά

Συναρμολογητής

Συναρμολογητής είναι ένα πρόγραμμα το οποίο παράγει αντικειμενικό κώδικα μεταφράζοντας, συνδυασμούς από μνημονικά και συντακτικά που αφορούν λειτουργίες και διευθυνσιοδοτήσεις, στις αντίστοιχες αριθμητικές τους τιμές. Αυτή η αναπαράσταση αποτελείται τυπικά από έναν κώδικα λειτουργίας (opcode) όπως επίσης και από άλλα bits ελέγχου. Ο συναρμολογητής επίσης υπολογίζει τις τιμές των σταθερών εκφράσεων και αντιλαμβάνεται τα συμβολικά ονόματα για τις τοποθεσίες μνήμης και για άλλες οντότητες[42]. Η χρήση συμβολικών αναφορών είναι ένα σημείο κλειδί για τους συναρμολογητές χάρη στο οποίο ο προγραμματιστής δεν χρειάζεται να προβεί σε βαρετούς και επαναλαμβανόμενους υπολογισμούς και χειροκίνητες ενημερώσεις διευθύνσεων μετά από αλλαγές στο πρόγραμμα. Οι περισσότεροι συναρμολογητές επίσης περικλείουν μακροχαρακτηριστικά ώστε να είναι σε θέση να κάνουν αντικαταστάσεις κειμένου, για παράδειγμα να μπορούν να παράξουν σύντομες ακολουθίες εντολών εσωτερικά αντί να είναι αναγκασμένοι να προβούν στην κλήση υποπρογραμμάτων.

Κάποιοι συναρμολογητές μπορεί επιπλέον να είναι σε θέση να μπορούν να εκτελέσουν κάποιες μορφές βελτίωσης προσανατολισμένες σε συγκεκριμένο σετ εντολών γλώσσας μηχανής. Ένα χαρακτηριστικό παράδειγμα της περίπτωσης αυτής είναι οι πανταχού παρόντες x86

41 (John Daintith, ed.) A Dictionary of Computing: "meta-assembler"

42 David Salomon (1993). *Assemblers and Loaders*

συναρμολογητές από διάφορους προμηθευτές. Οι περισσότεροι από αυτούς είναι σε θέση να εκτελέσουν αντικαταστάσεις άλματος (μεγάλα άλματα αντικαθίστανται από μικρά ή σχετικά άλματα) σε οποιοδήποτε αριθμό περασμάτων κατ' εντολή. Άλλοι μπορούν να εκτελέσουν ακόμη και εκ νέου διευθετήσεις ή και εισαγωγή εντολών, με χαρακτηριστικότερο παράδειγμα κάποιους συναρμολογητές αρχιτεκτονικών RISC οι οποίοι μπορούν (εντός ορίων) να βοηθήσουν στην βελτιστοποίηση της εκμετάλλευσης της ροής δεδομένων προς την κεντρική μονάδα επεξεργασίας, όσο περισσότερο αποδοτικά είναι κάτι τέτοι δυνατόν.

Όπως και οι αρχικές γλώσσες προγραμματισμού (βλέπε Fortran, Algol, Cobol, Lisp κλπ) οι συναρμολογητές είναι διαθέσιμοι από την δεκαετία του 1950 από τις πρώτες γενιές διεπαφών υπολογιστών βασισμένες σε κείμενο. Ωστόσο ουσιαστικά οι συναρμολογητές προηγήθηκαν μιας και είναι κατά πολύ απλούστερη η συγγραφή τους σε σχέση με τους μεταγλωττιστές των γλωσσών υψηλού επιπέδου. Αυτό συμβαίνει διότι κάθε μνημονικό μαζί με τους τρόπους αντιμετώπισης και τους τελεστές κάθε εντολής μεταφράζεται μάλλον άμεσα στην αντίστοιχη αριθμητική παράσταση γλώσσας μηχανής χωρίς περεταίρω περιεχόμενο ή ανάλυση. Έχουν υπάρξει επίσης και διάφορες τάξεις μεταφραστών και ημιαυτόματων γεννητριών κώδικα παρόμοιοι τόσο με την assembly όσο και με τις γλώσσες υψηλού επιπέδου με την speedcode να αποτελεί ίσως το περισσότερο γνωστό και χαρακτηριστικό παράδειγμα.

Είναι πιθανό να υπάρχουν υπάρχουν αρκετοί συναρμολογητές με διαφορετικό συντακτικό για μια συγκεκριμένη μονάδα κεντρικής επεξεργασίας ή για ένα συγκεκριμένο σύνολο εντολών γλώσσας μηχανής. Για παράδειγμα, μια εντολή προσθήκης δεδομένων της μνήμης σε καταχωριστή για έναν επεξεργαστή της οικογένειας των x86 θα μπορούσε να είναι “add eax, [ebx]”, στο αρχικό συντακτικό της Intel, ενώ η ίδια εντολή, για το συντακτικό της AT&T το οποίο χρησιμοποιείται από τον συναρμολογητή GNU, θα ήταν ως εξής “add1 (%ebx), %eax”. Παρά την διαφορετική εμφάνιση και το διαφορετικό συντακτικό οι διαφορετικοί συναρμολογητές παράγουν σε γενικές γραμμές τον ίδιο αριθμητικό κώδικα γλώσσας μηχανής. Ένας απλός συναρμολογητής μπορεί επίσης να έχει διαφορετικές καταστάσεις λειτουργίας έτσι ώστε να είναι σε θέση να υποστηρίξει διαφοροποιήσεις στο συντακτικό όπως επίσης και στην ακριβή μετάφραση των εντολών (όπως βλέπουμε μεταξύ των συντακτικών FASM και TASM, καθώς επίσης και σε ειδικές περιπτώσεις προγραμματισμού σε assembly για τον x86).

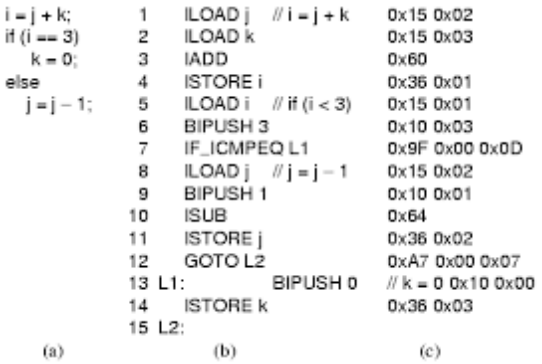
2.2.4 Πλήθος περασμάτων

Υπάρχουν δύο τύποι συναρμολογητών ανάλογα με το πόσα περάσματα από των πηγαίο κώδικα απαιτούνται ώστε να παραχθεί το εκτελέσιμο παράγωγο.

- Οι συναρμολογητές μονού περάσματος οι οποίοι όπως υποδηλώνει και το όνομα περνούν μία φορά από τον κώδικα. Κάθε σύμβολο που θα χρησιμοποιηθεί πριν να δηλωθεί θα απαιτήσει διορθώσεις στο τέλος του αντικειμενικού κώδικα αναγκάζοντας το πρόγραμμα σύνδεσης ή το πρόγραμμα φόρτωσης να πάει πίσω και να γράψει πάνω σε μια θέση κράτησης η οποία έχει μείνει κενή πριν γίνει χρήση του συμβόλου.
- Συναρμολογητές πολλών περασμάτων οι οποίοι δημιουργούν ένα πίνακα με τα σύμβολα και

τις τιμές τους στα πρώτα περάσματα και στα επόμενα περάσματα χρησιμοποιούν τον πίνακα στα επόμενα περάσματα προκειμένου να παράξουν τον εκτελέσιμο κώδικα.

Σε κάθε περίπτωση ο συναρμολογητής θα πρέπει να είναι σε θέση να ορίσει το μέγεθος κάθε εντολής στο αρχικό του πέρασμα έτσι ώστε να μπορεί να τις διευθύνσεις των υπόλοιπων συμβόλων. Αυτό σημαίνει ότι, αν το μέγεθος μιας διαδικασίας, που αναφέρεται σε ένα τελεστή που ορίζεται αργότερα εξαρτάται από τον τύπο ή την απόσταση του τελεστή και ο συναρμολογητής θα κάνει μια συγκρατημένη εκτίμηση του μεγέθους όταν θα συναντήσει για πρώτη φορά την διαδικασία και αν κριθεί αναγκαίο κατά τα περάσματα διορθώσεως θα αποδώσει επιπλέον χώρο. Σε έναν συναρμολογητή με βελτιστοποίηση γέφυρας οι διευθύνσεις μπορεί να επανυπολογίζονται μεταξύ των περασμάτων ώστε να είναι δυνατή η αλλαγή και ο επαναπροσδιορισμός των συγκρατημένων εκτιμήσεων με υπολογισμούς ακριβείας αναφορικά με το μέγεθος και την απόσταση από τον στόχο.



Ο αρχικός λόγος για τον οποίο στραφήκαμε στους συναρμολογητές μονού περάσματος ήταν η ταχύτητα συναρμολόγησης μιας και συχνά ένα δεύτερο πέρασμα μπορεί να απαιτούσε το γύρισμα στην αρχή μιας μπομπίνας με μαγνητική ταινία ή το διάβασμα από την αρχή ενός συνόλου καρτών που φιλοξενούσαν επάνω τους το πρόγραμμα. Με τα σύγχρονα μέσα αποθήκευσης και τις υπολογιστικές ταχύτητες των σύγχρονων μηχανημάτων όλα τα προηγούμενα έπαψαν να αποτελούν λόγους αποτροπής από την χρήση συναρμολογητών πολλών περασμάτων. Το πλεονέκτημα με τους συναρμολογητές πολλαπλών περασμάτων είναι ότι εξ' αιτίας της απουσίας διορθώσεων η διαδικασία σύνδεσης ή φορτώματος του προγράμματος είναι κατά πολύ συντομότερη[43].

2.2.5 Παράδειγμα

Στο παρακάτω παράδειγμα κώδικα, ένας συναρμολογητής μονού περάσματος, θα ήταν σε θέση να υπολογίσει την διεύθυνση της εντολής οπισθοδρόμησης BKWD κατά την συναρμολόγηση της εντολής S2, αλλά δεν θα ήταν σε θέση να υπολογίσει την διεύθυνση της εντολής προώθησης FWD κατά την συναρμολόγηση του σώματος κώδικα S12. Πράγματι η FWD μπορεί να είναι απροσδιόριστη. Ένας συναρμολογητής διπλού περάσματος θα προδιόριζε και τις δύο διευθύνσεις κατα το πρώτο πέρασμα ώστε να του είναι γνωστές και την παραγωγή του κώδικα στο πέρασμα 2.

```
S1   B    FWD
  ...
FWD   EQU *
  ...
BKWD  EQU *
  ...
```

43 *Beck, Leland L. (1996). "2". System Software: An Introduction to Systems Programming. Addison Wesley.*

S2 B BKWD

2.2.6 Συναρμολογητές υψηλού επιπέδου

Οι περισσότερο εκλεπτυσμένοι συναρμολογητές προσφέρουν και την δυνατότητα γλωσσικών αφαιρέσεων όπως:

5. Υψηλού επιπέδου δήλωση και επίκληση μεθόδων και διαδικασιών.
6. Προχωρημένες δομές ελέγχου.
7. Υψηλού επιπέδου τύπους δεδομένων, όπως είναι οι structures, records, unions, classes και sets.
8. Προχωρημένες δυνατότητες μάκρο επεξεργασίας (αν και πρόκειται για χαρακτηριστικό το οποίο είναι διαθέσιμο και στους απλούς συναρμολογητές από τα τέλη της δεκαετίας του 50 στην σειρά 700 της IBMκαι από την δεκαετία του 60 για τους IMB/360, αλλά και σε άλλα μηχανήματα).
9. Δυνατότητα χαρακτηριστικών αντικειμενοστραφούς προγραμματισμού όπως είναι οι κλάσεις, τα αντικείμενα, η αφαίρεση, ο πολυμορφισμός και η κληρονομικότητα44.

2.2.7 Η γλώσσα Assembly

Ένα πρόγραμμα γραμμένο σε γλώσσα Assembly αποτελείται από ένα σύνολο εντολών (γραμμένων με μνημονικά) και μέτα δηλώσεων (γνωστών και ως οδηγιών ή ψεύδο εντολών), σχολίων και δεδομένων. Οι εντολές τις γλώσσας Assembly αποτελούνται συνήθως από το μνημονικό κομμάτι με το οποίο είναι γνωστές ακολουθούμενο από μια λίστα δεδομένων, παραμέτρων45 και τελεστών. Αυτά όλα μεταφράζονται από ένα συναρμολογητή σε εντολές γλώσσας μηχανής οι οποίες μπορουν να φορτωθούν στην μνήμη και να εκτελεστούν.
Για παράδειγμα η παρακάτω εντολή λέει σε ένα επεξεργαστή x86/IA-32 να μετακινήσει μια 8μπιτη τιμή σε ένα καταχωριστή. Ο δυαδικός κώδικας για την εντολή αυτή είναι 10110 ακολουθούμενο από μια 3μπιτη τιμή η οποία προσδιορίζει τον καταχωριστή που θέλουμε να χρησιμοποιήσουμε. Το αναγνωριστικό για τον καταχωριστή AL είναι το 000 και έτσι το παρακάτω κομμάτι κώδικα μηχανής φορτώνεται στον καταχωριστή AL με δεδομένα τα 0110000146.
10110000 01100001
Ο παραπάνω κώδικας μηχανής μπορεί να γίνει περισσότερο αναγνώσιμος από τον άνθρωπο αν τον εκφράσουμε σε δεκαεξαδικό αριθμητικό σύστημα ως ακολούθως.
B0 61

---

44 Hyde, Randall. "Chapter 12 – Classes and Objects". The Art of Assembly Language, 2nd Edition. No Starch Press. © 2010.

45 *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference (PDF). Intel Corporation. 1999. Retrieved 18 November 2010.*

46 *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference (PDF). Intel Corporation. 1999. pp. 442 and 35. Retrieved 18 November 2010.*

Εδώ, B0 σημαίνει “μετακίνησε ένα αντίγραφο της ακόλουθης τιμής στον AL” και 61 είναι η δεκαεξαδική αναπαράσταση της τιμής 01100001 η οποία στο δεκαδικό θα ήταν 97. Η γλώσσα Assembly για την οικογένεια επεξεργαστών 8086 παρέχει το μνημονικό MOV (μια σύντμηση του move) για εντολές σαν και αυτή και έτσι ο παραπάνω δυσανάγνωστος κώδικας μπορεί να γραφτεί όπως φαίνεται ακολούθως σε γλώσσα Assembly πλήρως και με ένα επεξηγηματικό σχόλιο να τον συνοδεύει εάν κάτι τέτοιο απαιτείται μετά από ένα ερωτηματικό.

```
MOV AL, 61h       ; Load AL with 97 decimal (61 hex)
```

Σε κάποιες εκδόσεις των γλωσσών assembly το ίδιο μνημονικό μιας εντολής όπως είναι ας πούμε το MOV μπορεί να χρησιμοποιηθεί για μια ολόκληρη οικογένεια σχετικών εντολών, για παράδειγμα για φόρτωμα, αντιγραφή και μετακίνηση δεδομένων, είτε πρόκειται για άμεσες τιμές είτε για τιμές αποθηκευμένες σε καταχωριστές είτε για τοποθεσίες μνήμης που υποδεικνύονται από τιμές καταχωριστών. Άλλοι συναρμολογητές είναι δυνατόν να χρησιμοποιούν διαφορετικά μνημονικά για κάθε είδους εντολή όπως L για την μετακίνηση από την μνήμη στον καταχωριστή, ST για μετακίνηση από τον καταχωριστή στην μνήμη, LR για μετακίνηση μεταξύ καταχωριστών και MVI για άμεση μετακίνηση τελεστή στη μνήμη.

Ο κώδικας λειτουργίας (opcode) 10110000 (B0) για την x86 αρχιτεκτονική αντιγράφει μια 8μπιτη τιμή στον καταχωριστή AL ενώ ο κώδικας 10110001 (B1) μεταφέρει την τιμή αυτή στον CL καταχωριστή και ο κώδικας 10110010 (B2) κάνει τον ίδιο με προορισμό τον DL. Ακολουθούν κάποια παραδείγματα[47]:

```
MOV AL, 1h        ; Φόρτωσε AL με την άμεση τιμή 1
MOV CL, 2h        ; Φόρτωσε CL με την άμεση τιμή 2
MOV DL, 3h        ; Φόρτωσε DL με την άμεση τιμή 3
```

Η σύνταξη της εντολής MOV μπορεί να είναι και περισσότερο πολύπλοκη όπως φαίνεται στα παρακάτω παραδείγματα[48]

```
MOV EAX, [EBX]    ; Μετακίνησε 4 bytes της θέσης μνήμης που περιέχεται στον EBX στον EAX
MOV [ESI+EAX], CL ; Μετακίνησε τα περιεχόμενα τουCL στα byte της διεύθυνσης ESI+EAX
```

Σε κάθε περίπτωση το μνημονικό MOV μεταφράζεται άμεσα σε κώδικα λειτουργίας στο εύρος αριθμών 88-8E, A0-A3, B0-B8, C6 ή C7 από τον συναρμολογητή και ο προγραμματιστής δεν είναι ανάγκη να θυμάται σε ποιο[49].

Η μετατροπή του κώδικα assembly σε κώδικα μηχανής είναι δουλειά του συναρμολογητή και το αντίστροφο μπορεί, αν και κάποιες φορές τμηματικά, να το κάνει ένας αποσυναρμολογητής. Αντίθετα με ότι συμβαίνει με τις γλώσσες υψηλού επιπέδου συνήθως υπάρχει μια σχέση ένα προς ένα ανάμεσα στις απλές εντολές της γλώσσας assembly και των εντολών του κώδικα μηχανής. Ωστόσο σε κάποιες περιπτώσεις ένας συναρμολογητής μπορεί να παρέχει την δυνατότητα χρήσης ψευδοεντολών (ουσιαστικά μακροεντολών), οι οποίες αναλύονται σε περισσότερες εντολές κώδικα μηχανής, έτσι ώστε να παρέχει στον προγραμματιστή ένα επιθυμητό επίπεδο χρηστικότητας και ευκολίας. Για παράδειγμα για μια μηχανή με έλλειψη της δομής IF μεγαλύτερο ή ίσο,

---

47 *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference (PDF). Intel Corporation. 1999. pp. 442 and 35. Retrieved 18 November 2010.*

48 *Evans, David (2006). "x86 Assembly Guide". University of Virginia. Retrieved 18 November 2010.*

49 *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference (PDF). Intel Corporation. 1999. pp. 442 and 35. Retrieved 18 November 2010.*

συναρμολογητής μπορεί να παρέχει την δυνατότητα χρήσης της ψευδοεντολής η οποία θα αναλύεται στις εντολές όρισε αν είναι μικρότερο από και στη δομή εάν είναι ίσο με το μηδέν. Οι περισσότεροι πλήρων χαρακτηριστικών συναρμολογητές επίσης παρέχουν μια αρκετά πλούσια γκάμα μακροεντολών οι οποίες χρησιμοποιούνται από προμηθευτές και προγραμματιστές για την δημιουργία των περισσότερο σύνθετων τμημάτων κώδικα και αλληλουχιών δεδομένων.
Κάθε υπολογιστική αρχιτεκτονική έχει τη δική της γλώσσα μηχανής. Οι υπολογιστές ουσιαστικά διαφέρουν στον πλήθος και στο είδος των λειτουργιών που μπορούν να υποστηρίξουν, στο μέγεθος και στο πλήθος των καταχωριστών και στην αναπαράσταση των δεδομένων σε αποθήκευση. Παρόλο που οι γενικού σκοπού υπολογιστές ουσιαστικά καταφέρνουν να υλοποιούν τις ίδιες εργασίας ο τρόπος με τον οποίο τις φέρουν εις πέρας διαφέρει μεταξύ τους. Οι γλώσσα assembly που είναι φτιαγμένη για τον κάθε ένα ουσιαστικά αντανακλά τις διαφορές αυτές.
Είναι δυνατόν πολλά σύνολα μνημονικών ή συντακτικών γλώσσας assembly να υπάρχουν για ένα σύνολο εντολών μηχανής. Τα σύνολα αυτά αρχικοποιούνται σε διαφορετικά προγράμματα συναρμολόγησης. Σε αυτές τις περιπτώσεις το πλέον δημοφιλές είναι αυτό με το οποίο προμηθεύει ο κατασκευαστής τους πελάτες του και είναι και αυτό το οποίο χρησιμοποιείται και στην τεκμηρίωση του.

2.2.8 Ο σχεδιασμός της γλώσσας
2.2.8.1 Βασικά χαρακτηριστικά

Υπάρχει ένας μεγάλος βαθμός διαφοροποίησης στον τρόπο με τον οποίο οι δημιουργοί των συναρμολογητών κατηγοριοποιούν τις δηλώσεις και στην ονοματολογία που χρησιμοποιούν. Πιο συγκεκριμένα μερικοί περιγράφουν κάθε τι πέρα από ένα μνημονικό εντολής κώδικα μηχανής ή έστω διευρυμένο μνημονικό σαν μια ψευδολειτουργία (pseudo-op). Μια τυπική υλοποίηση γλώσσας assembly περιέχει 3 τύπους εντολών οι οποίες χρησιμοποιούνται για να περιγράψουν οτιδήποτε άλλο πέραν των λειτουργιών του προγράμματος:

- Μνημονικά κώδικα λειτουργίας (opcode mnemonics)
- Ορισμούς δεδομένων
- Οδδηγίες assembly

2.2.8.2 Μνημονικά κώδικα λειτουργίας και εκτεταμένα μνημονικά

Οι εντολές (δηλώσεις) στην γλώσσα assembly γενικά είναι πολύ απλές σε αντίθεση με τις εντολές που βλέπουμε στις γλώσσες υψηλού επιπέδου. Γενικά ως μνημονικό ορίζουμε ένα συμβολικό όνομα για μια απλή εκτελέσιμη εντολή του κώδικα μηχανής (μια εντολή του κώδικα λειτουργίας) και υπάρχει τουλάχιστον ένα μνημονικό κώδικα λειτουργίας ορισμένο για κάθε εντολή κώδικα μηχανής. Κάθε εντολή τυπικά αποτελείται από μια λειτουργία (opcode) και κανένα ή περισσότερα ορίσματα ή τελεστές. Οι περισσότερες εντολές αναφέρονται σε μια μόνο τιμή ή έστω ένα ζεύγος τιμών. Οι τελεστές μπορεί να είναι άμεσοι (τιμές οι οποίες εμφανίζονται αριθμητικά μέσα στην ίδια την εντολή), καταχωριστές οι οποίοι προσδιορίζονται μέσα στην εντολή ή υπονοούνται ή οι διευθύνσεις δεδομένων που βρίσκονται σε άλλες θέσεις του αποθηκευτικού χώρου. Αυτό καθορίζεται από την αρχιτεκτονική επεξεργαστή την οποία προσπαθεί να εκμεταλλευτεί ο υπό ανάπτυξη συναρμολογητής. Ουσιαστικά ο συναρμολογητής θα αντανακλά τον τρόπο λειτουργίας αυτής της αρχιτεκτονικής. Εκτεταμένα μνημονικά συχνά χρησιμοποιούνται για προσδιορίσουν ένα συνδυασμό μια εντολής κώδικα λειτουργίας με μια ένα συγκεκριμένο τελεστή. Για παράδειγμα οι συναρμολογητές του System/360 χρησιμοποιούν το B σαν ένα εκτεταμένο μνημονικό για το BC με μια μάσκα του 15 και NOP (“No OPeration – μην κάνεις τίποτα για ένα βήμα”) για το BC με μάσκα

του 0.
Τα εκτεταμένα μνημονικά συχνά χρησιμοποιούνται και για να υποστηρίξουν εξειδικευμένες χρήσεις των εντολών οι οποίες δεν είναι προφανείς από το όνομα της εντολής. Για παράδειγμα πολλές κεντρικές μονάδες επεξεργασίας δεν διαθέτουν μια αποκλειστική NOP εντολή, αλλά έχουν εντολές οι οποίες μπορούν να χρησιμοποιηθούν για τον σκοπό αυτό. Στις κεντρικές μονάδες επεξεργασίας της οικογένειας 8086 η εντολή xchg ax, ax χρησιμοποιείται ως nop με την nop να είναι μια ψευδοεντολή κώδικα λειτουργίας για την κωδικοποίηση της εντολής xchg ax, ax. Μερικοί αποσυναρμολογητές το αναγνωρίζουν αυτό και θα αποκωδικοποιήσουν την εντολή xchg ax, ax ως nop. Παρόμοια οι συναρμολογητές της IBM για τα System/360 και System/370 χρησιμοποιούν το εκτεταμένο μνημονικό NOP και NORP για τις εντολές BC και BCR με μάσκα του 0. Για την αρχιτεκτονική SPARC αυτές οι περιπτώσεις είναι γνωστές ως σύνθετες εντολές[50].
Κάποιοι συναρμολογητές επίσης υποστηρίζουν τις απλές ενσωματωμένες μακροεντολές οι οποίες αναλύονται σε δύο ή περισσότερες εντολές γλώσσας μηχανής. Για παράδειγμα, με μερικούς Z80 συναρμολογητές η εντολή ld hl, bc ξέρουμε ότι παράγει την εντολή ld l, c ακολουθούμενη από την εντολή ld h, b[51]. Είναι από τις περιπτώσεις που η αρχική εντολή αποκαλείται και ψευδοεντολή κώδικα λειτουργίας.
Τα μνημονικά είναι ουσιαστικά αυθαίρετα σύμβολα. Το 1985 το IEEE δημοσίευσε το στάνταρ 694 για μια ενιαία μορφή συνόλου εντολών οι οποία θα έπρεπε να υιοθετείται από κάθε συναρμολογητή. Η προσπάθεια δεν απέδωσε καρπούς και έκτοτε αποσύρθηκε.

2.2.8.3 Οδηγίες δεδομένων

Υπάρχουν εντολές οι οποίες χρησιμοποιούνται για να προσδιοριστούν δεδομένα, για να κρατηθούν δεδομένα και μεταβλητές. Καθορίζουν τον τύπο των δεδομένων και το μέγεθος τους. Αυτές οι εντολές μπορούν επίσης να προσδιορίσουν αν τα εν λόγω δεδομένα θα είναι διαθέσιμα και σε εξωτερικά προγράμματα (προγράμματα που συναρμολογήθηκαν ξεχωριστά δηλαδή) ή αν θα είναι διαθέσιμα μόνο στο πρόγραμμα μέσα στο οποίο ορίζονται. Κάποιοι συναρμολογητές κατηγοριοποιούν αυτές τις εντολές ως ψευδοεντολές κώδικα λειτουργίας.

2.2.8.4 Εντολές συναρμολόγησης

Οι εντολές συναρμολόγησης, επίσης γνωστές και ως ψευδοεντολές κώδικα λειτουργίας ή ψευδολειτουργίες είναι εντολές οι οποίες δίνονται στον συναρμολογητή οδηγώντας τον στην εκτέλεση λειτουργιών άλλων από την συναρμολόγηση εντολών[52]. Οι εντολές αυτές επηρεάζουν το πως ο συναρμολογητής λειτουργεί και μπορεί να επηρεάσουν τον αντικειμενικό κώδικα, τον πίνακα συμβόλων, την λίστα των αρχείων και τις τιμές των εσωτερικών παραμέτρων του συναρμολογητή. Μερικές φορές ο όρος ψευδοεντολή κώδικα λειτουργίας χρησιμοποιείται για εκείνες τις εντολές οι οποίες παράγουν αντικειμενικό κώδικα όπως επίσης και για εκείνες οι οποίες παράγουν δεδομένα[53].
Τα ονόματα των ψευδολειτουργιών συχνά ξεκιProvinciano, Brian. "NESHLA: The High Level, Open Source, 6502 Assembler for the Nintendo Entertainment System".νούν με μια τελεία ώστε να είναι δυνατή η διάκριση τους από τις εντολές της μηχανής. Οι ψευδολειτουργίες μπορούν να κάνουν την συναρμολόγηση του προγράμματος να εξαρτάται από τα δεδομένα εισόδου των

---

50 *"The SPARC Architecture Manual, Version 8" (PDF). SPARC, International. 1992.*

51 Z80 Op Codes for ZINT. Z80.de. Retrieved on 2013-07-21.

52 David Salomon (1993). *Assemblers and Loaders*

53 *Microsoft Corporation. "MASM: Directives & Pseudo-Opcodes" (PDF). Retrieved March 19, 2011.*

παραμέτρων από τον προγραμματιστή έτσι ώστε το πρόγραμμα να μπορεί να συναρμολογείται με πολλούς διαφορετικούς τρόπους ενδεχομένως για διαφορετικές εφαρμογές. Ή μια ψευδολειτουργία μπορεί να χρησιμοποιηθεί για να χειραγωγήσει την παρουσίαση ενός προγράμματος έτσι ώστε να γίνει περισσότερο αναγνώσιμο και πιο εύκολο να συντηρηθεί. Μια άλλη κοινή χρήση των ψευδολειτουργιών είναι η δέσμευση τμημάτων του αποθηκευτικού χώρου για δεδομένα που αφορούν τον χρόνο εκτέλεσης του προγράμματος και την κατά συνθήκη αρχικοποίηση τους σε γνωστές τιμές.
Οι συμβολικοί συναρμολογητές επιτρέπουν στους προγραμματιστές την συσχέτιση αυθαίρετων ονομάτων (σύμβολα ή ετικέτες) με τοποθεσίες της μνήμης και μεταβλητές ή σταθερές. Συνήθως, σε κάθε σταθερά και μεταβλητή αποδίδεται ένα όνομα έτσι ώστε οι εντολές να μπορούν να αναφέρονται στις τοποθεσίες αυτές μέσω του ονόματος, προάγοντας έτσι τον αυτοτεκμηριούμερνο κώδικα. Στον εκτελέσιμο κώδικα, το όνομα κάθε υπορουτίνας συσχετίζεται με το σημείο εισόδου έτσι ώστε κάθε κλήση της υπορουτίνας αυτής να μπορεί να χρησιμοποιήσει το όνομα αυτό. Στο εσωτερικό των υπορουτίνων στους προορισμούς GOTO αποδίδονται ετικέτες. Κάποιοι συμβολομεταφραστές υποστηρίζουν τοπικά σύμβολα τα οποία λεκτικά διαχωρίζονται από τα κανονικά σύμβολα (για παράδειγμα η χρήση του 10$ σαν ένας GOTO προορισμός).
Άλλοι συναρμολογητές όπως ο NASM παρέχουν δυνατότητα ευέλικτης διαχείρισης των συμβόλων, παρέχοντας στους προγραμματιστές την δυνατότητα να διαχειριστούν διαφορετικές περιοχές ονοματοδοσίας, την δυνατότητα αυτόματου υπολογισμού μετατοπίσεων εντός των δομών δεδομένων και την ανάθεση ετικετών σε πραγματικές τιμές ή στα αποτελέσματα από απλούς υπολογισμούς που υλοποιούνται από τον συναρμολογητή. Οι ετικέτες μπορούν επίσης να χρησιμοποιηθούν για την αρχικοποίηση σταθερών και μεταβλητών με δυνατότητα αλλαγής της τοποθεσίας τους στη μνήμη.
Οι γλώσσες assembly όπως οι περισσότερες άλλε γλώσσες προγραμματισμού επιτρέπουν την προσθήκη σχολίων στον πηγαίο κώδικα του προγράμματος, τα οποία θα αγνοηθούν κατά την διαδικασία συναρμολόγησης. Τα έξυπνα και περιεκτικά σχόλια είναι πολύ σημαντικά στα προγράμματα σε γλώσσα assembly μια και το νόημα και ο σκοπός μιας ακολουθίας δυαδικών εντολών μηχανής, μπορεί να είναι εξαιρετικά δύσκολο να προσδιοριστούν. Ο ωμός (ασχολίαστος) κώδικας assembly που παράγεται από μεταφραστές ή από αποσυναρμολογητές είναι ιδιαιτέρως δύσκολο να διαβαστεί και να διορθωθεί ή τροποποιηθεί.

2.2.8.5 Μακροεντολές

Πολλού συναρμολογητές υποστηρίζουν τη χρήση εξ αρχής προσδιορισμένων μακροεντολών και άλλοι υποστηρίζουν μακροεντολές οι οποίες είναι ορισμένες από τον προγραμματιστή (και με δυνατότητα διαρκούς επαναπροσδιορισμού), οι οποίες είναι ακολουθίες γραμμών κειμένου στις οποίες οι σταθερές και οι μεταβλητές είναι ενσωματωμένες. Αυτές οι ακολουθίες από γραμμές κειμένου μπορεί να περιέχουν εντολές κώδικα λειτουργίας ή οδηγίες. Από τη νστιγμή που μια μακροεντολή θα οριστεί το όνομα της μπορεί να χρησιμοποιηθεί στην θέση ενός μνημονικού. Όταν ο συναρμολογητής επεξεργάζεται μια τέτοια δήλωση, αντικαθιστά την δήλωση με τις γραμμές κειμένου που έχουν συσχετισθεί με την μακροεντολή αυτή και στην συνέχεια τις επεξεργάζεται σαν να υπήρχαν οι ίδιες μέσα στο αρχείο του κώδικα. Οι μακροεντολές με αυτή την μορφή και νόημα πρωτοεμφανίστηκαν στους autocoders της IBM κατά την δεκαετία του 1950.
Αυτός ο ορισμός των μακροεντολών είναι θα λέγαμε περισσότερο περιεκτικός από ότι είναι η χρήση του όρου σε άλλα πλαίσια όπως στην γλώσσα προγραμματισμού C. Οι μακροεντολές της C οι οποίες δημιουργούνται με την χρήση της οδηγίας #define τυπικά αποτελούνται από μια μόνο γραμμή κώδικα ή σε κάποιες εξαιρετικές περιπτώσεις από λίγες γραμμές κώδικα. Οι μακροεντολές συναρμολογητών σαν τον PL/1 μπορεί να είναι ολόκληρα μακροσκελή προγράμματα οι ίδιες και να

εκτελούνται μετά από μετάφραση από τον συναρμολογητή κατά την συναρμολόγηση.
Μιας και οι μακροεντολές μπορούν να έχουν σύντομα ονόματα αλλά ουσιαστικά να εκτείνονται σε αρκετές και πολλές φορές πολλές γραμμές κώδικα μπορούν να χρησιμοποιηθούν για να κάνουν τα προγράμματα σε γλώσσα assembly να φαίνονται πολύ μικρότερα από ότι είναι στην πραγματικότητα, απαιτώντας πολύ λιγότερες γραμμές κώδικα από ότι τα προγράμματα γραμμένα σε γλώσσες υψηλού επιπέδου. Μπορούν επίσης να χρησιμοποιηθούν για να προσθέσουν υψηλότερα επίπεδα δόμησης στα προγράμματα assembly και προαιρετικά να εισαγάγουν την ενσωματωμένη αποσφαλμάτωση του κώδικα μέσω παραμέτρων και άλλων παρόμοιων χαρακτηριστικών.
Οι μακροσυναρμολογητές συχνά επιτρέπουν οι μακροεντολές να δέχονται παραμέτρους. Κάποιοι συναρμολογητές περιέχουν αρκετά εκλεπτυσμένες δυνατότητες μακροεντολών, ενσωματώνοντας πολύ υψηλού επιπέδου στοιχεία γλώσσας όπως προαιρετικές παραμέτρους, συμβολικές μεταβλητές, υποθετικά, διαχείριση συμβολοσειρών και αριθμητικές λειτουργίες, όλα με την δυνατότητα να χρησιμοποιηθούν κατά την εκτέλεση μιας δεδομένης μακροεντολής κια επιτρέποντας στις μακροεντολές να αποθηκεύσουν περιεχόμενο ή να ανταλλάξουν πληροφορίες. Έτσι μια μακροεντολή μπορεί να παράξει πολλές εντολές γλώσσας assembly ή ορισμούς δεδομένων βασισμένα στα στοιχεία της μακροεντολής. Αυτό το χαρακτηριστικό μπορεί να χρησιμοποιηθεί για την παραγωγή δομών δεδομένων τύπου record ή ξετυλιγμένους βρόχους ή θα μπορούσε για παράδειγμα να παράξει ολόκληρους αλγόριθμους βασισμένους σε πολύπλοκες παραμέτρους. Ένας οργανισμός ο οποίος χρησιμοποιεί μια γλώσσα assembly η οποία έχει επεκταθεί σημαντικά χρησιμοποιώντας τέτοια γκάμα μακροχαρακτηριστικών μπορεί να θεωρηθεί ότι χρησιμοποιεί ουσιαστικά μια γλώσσα υψηλού επιπέδου, μιας και οι προγραμμτιστές του οργανισμού δεν δουλεύουν με τα χαρακτηριστικά χαμηλού επιπέδου της γλώσσας. Υπογραμμίζοντας αυτό το σημείο θα πρέπει να αναφέρουμε ότι οι μακροεντολές χρησιμοποιήθηκαν αρχικά για την υλοποίηση μιας νοητής μηχανής στο SNOBOL4 το 1967 η οποία είχε γραφεί στη γλώσσα υλοποίησης SNOBOL, μιας γλώσσας assembly ιδανικής για νοητές μηχανές, η οποία ακολούθως στράφηκε στις φυσικές μηχανές με την ενσωμάτωση της σε έναν φυσικό συναρμολογητή μέσω ενός μακροσυναρμολογητή[54]. Αυτό παρείχε έναν πολύ υ ψυλό βαθμό μεταφερσιμότητας ειδικά για τα δεδομένα της εποχής.
Οι μακροεντολές χρησιμοποιήθηκαν για προσαρμογές σε συστήματα λογισμικού μεγάλης κλίμακας , για συγκεκριμένους πελάτες στην περιοχή των mainframe και επίσης χρησιμοποιήθηκαν γαι την ικανοποίηση των αναγκών των υπαλλήλων κατασκευάζοντας συγκεκριμένες εκδόσεις κατασκευαστικών λειτουργικών συστημάτων. Αυτό για παράδειγμα έγινε από τους προγραμματιστές συστημάτων που δούλευαν με τη νοητή μηχανή του Συστήματος Διαλογικού Μόνιτορ της IBM VM/CMS και με τα επιπλέον χαρακτηριστικά του συστήματος επεξεργασίας συναλλαγών πραγματικού χρόνου της IBM, το Σύστημα Διαχείρισης Πληροφοριών Πελατών CICS και ACP/TPF σύστημα , το σύστημα διαχείρισης των οικονομικών των αερογραμμών το οποίο ξεκίνησε το 1970 και το οποίο τρέχει ακόμη πολλά μεγάλα υπολογιστικά συστήματα κρατήσεων και πιστωτικών καρτών.
Είναι επίσης δυνατόν να χρησιμοποιηθούν μόνο τα μακροχαρακτηριστικά στοιχεία ενός συναρμολογητή έτσι ώστε να παράξει κώδικα γραμμένο σε εντελώς διαφορετικές γλώσσες. Για παράδειγμα για να δημιουργήσουμε μια έκδοση ενός προγράμματος σε COBOL χρησιμοποιώντας

54 Griswold, Ralph E. *The Macro Implementation of SNOBOL4*. San Francisco, CA: W. H. Freeman and Company, 1972 (ISBN 0-7167-0447-1), Chapter 1.

έναν πλήρη μακροσυναρμολογητή ο οποίος θα περιέχει γραμμές κώδικα COBOL ανάμεσα σε assembly λειτουργίες χρόνου οδηγώντας τον συναρμολογητή να παράξει αυθαίρετο κώδικα. Το λειτουργικό της IBM OS/360 χρησιμοποιεί μακροεντολές για να καταφέρει να εκτελέσει την παραγωγή συστήματος. Ο χρήστης κάνει επιλέγει κωδικοποιώντας μια σειρά από μακροεντολές συναρμολογητή. Η συναρμολόγηση αυτών των μακροεντολών παράγει ένα ρεύμα εργασίας που χτίζει το σύστημα περιέχοντας και την γλώσσα ελέγχου των εργασιών και τις εντολές ελέγχου εφαρμογών.

Αυτό συμβαίνει διότι όπως έγινε κατανοητό την δεκαετία του 60 η έννοια της επεξεργασίας μακροεντολών είναι ανεξάρτητη από την έννοια της συναρμολόγησης με την πρώτη να είναι περισσότερο επεξεργασία κειμένου και λέξεων παρά παραγωγή αντικειμενικού κώδικα. Η έννοια της μακροεπεξεργασίας εμφανίστηκε και από τότε την βλέπουμε και στον προγραμματισμό με γλώσσα C, η οποία υποστηρίζει οι προεπεξεργαστικές εντολές να μπορούν να ορίζουν μεταβλητές και να μπορούν να πραγματοποιούν ελέγχους κατάστασης στις τιμές τους. Να σημειωθεί εδώ ότι σε αντίθεση με τους προηγούμενους μακροεπεξεργαστές που συναντούσαμε στο εσωτερικό συναρμολογητών ο προεπεξεργαστής της C δεν είναι ολοκληρωμένος κατά Turing διότι δεν έχει την ιδιότητα του βρόχου ή του άλματος.

Παρά την μεγάλη δύναμη των μακροεντολών, η χρήση των τελευταίων μάλλον ατόνησε στις περισσότερες γλώσσες υψηλού επιπέδου (με μεγάλες εξαιρέσεις τις C/C++ και PL/I) ενώ παρμένει αιωνόβιο χαρακτηριστικό στους συναρμολογητές.

Η αντικατάσταση των μακροπαραμέτρων γίνεται αποκλειστικά μέσω ονόματος: στον μακροεπεξεργαστικό χρόνο η τιμή μιας παραμέτρου αντικαθίσταται ως κείμενο με το όνομα της. Η πλέον γνωστή κλάση σφαλμάτων που προέκυψαν από αυτές τις πρακτικές ήταν η χρήση μιας παραμέτρου η οποία ήταν ήδη μια έκφραση και όχι ένα απλό όνομα την στιγμή που ο μακροεπεξεεργαστής περίμενε ένα όνομα. Στην μακροεντολή:

foo: macro a
load a*b

η πρόθεση ήταν το πρόγραμμα που καλεί την μακροεντολή να προμηθεύσει το όνομα της μεταβλητής a και η καθολική μεταβλητή ή σταθερά b να χρησιμοποιηθεί για πολλαπλασιαστεί με το a. Εαν η foo κληθεί με την παράμετρο a-c, η μακροεντολή επεκτείνεται με την load a-c*b. Προκειμένου να αποφύγουν ασάφειες οι χρήστες μακροεπεξεργαστών μπορούν να βάζουν εντός παρενθέσεων τις κανονικές παραμέτρους που βρίσκονται εντός του ορισμού των μακροεντολών[55].

2.2.9 Υποστήριξη για τον δομημένο προγραμματισμό

Ορισμένοι συναρμολογητές έχουν ενσωματώσει στοιχεία δομημένου προγραμματισμού προκειμένου να κωδικοποιήσουν την εκτέλεση της ροής. Το πρώτο παράδειγμα αυτής της προσέγγισης ήταν το Concept-14 macro set, το οποίο προτάθηκε αρχικά από τον Δρ. H.D. Mills τον Μάρτιο του 1970 και υλοποιήθηκε από τον marvin Kessler στο IBM Federal System Division ο οποίος επέκτεινε τον μακροσυναρμολογητή του S/360 με την προσθήκη IF/ELSE/ENSIF και άλλες

55 *"Macros (C/C++), MSDN Library for Visual Studio 2008". Microsoft Corp. Retrieved 2010-06-22.*

παρόμοιες δομές ελέγχου της ροής εκτέλεσης του προγράμματος[56]. Αυτός ήταν ένας τρόπος για να μειωθεί ή να απαλειφθεί πλήρως η χρήση της GOTO εντολής στον κώδικα assembly, έναν από τους κύριους παράγοντες παραγωγής κώδικα spaghetti στα προγράμματα γλώσσας assembly. Η εν λόγω προσέγγιση έγινε ευρέως αποδεκτή στις αρχές της δεκαετίας του 80.
Μια περίεργη σχεδίαση ήταν αυτή της A-natural, ενός συναρμολογητή με stream προσανατολισμό, για 8080/Z80 επεξεργαστές από την Whitesmiths Ltd (προγραμματιστές του Idris ενός λειτουργικού παρόμοιου με το Unix, και αυτού που χαρακτηρίστικε ως ο πρώτος εμπορικός μεταφραστής C).Η γλώσσα κατηγοριοποιήθηκε ως συναρμολογητής γιατί δούλευε με χαμηλού επιπέδου στοιχεία μηχανής όπως καταχωριστές και εντολές κώδικα λειτουργίας και αναφορές στη μνήμη. Ωστόσο ενσωμάτωνε μια έκφραση συντακτικού τέτοια ώστε να καταδεικνύει την σειρά εκτέλεσης. Παρενθέσεις και άλλα ειδικά σύμβολα μαζί με κατασκευές προγραμμάτων με προσανατολισμό δομημένου προγραμματισμού, έλεγχαν την ακολουθία των παραγωγή των οδηγιών. Η A-natural φτιάχθηκε ως η αντικειμενική γλώσσα ενός C μεταφραστή παρά σαν ένα σύστημα παραγωγής κώδικα με το χέρι, ωστόσο το λογικό συντακτικό της κατέληξε να κερδίσει αρκετούς οπαδούς από τον χώρο των συναρμολογητών.
Η αλήθεια είναι ότι υπήρξε φαινομενικά μικρή ζήτηση για περισσότερο εκλεπτυσμένους συναρμολογητές από την στιγμή που άρχισε η αποκλιμάκωση ανάπτυξης λογισμικού μεγάλης κλίμακας[57]. Παρόλα αυτά ακόμα και σήμερα οι συναρμολογητές εξακολουθούν να αναπτύσσονται και να βρίσκουν εφαρμογή ιδιαιτέρως σε περιπτώσεις όπου, οι περιορισμοί στους πόρους ή οι ιδιαιτερότητες της αρχιτεκτονικής του συστήματος για το οποίο γίνεται η ανάπτυξη, αποτελούν εμπόδιο για την αποτελεσματική χρήση γλωσσών προγραμματισμού υψηλού επιπέδου[58].

2.2.10 Χρήση της γλώσσας assembly
2.2.10.1 Ιστορική προοπτική

Οι γλώσσες assembly και η χρήση της λέξης assembly εκκινούν ιστορικά από την στιγμή που εισήχθη η έννοια των αποθηκευμένων προγραμμάτων. Το Electronic Delay Storage Automatic Calculator (EDSAC) περιείχε ένα συναρμολογητή με το όνομα inital orders με κύριο χαρακτηριστικό του τα μνημονικά ενός γράμματος το 1949[59]. Το SOAP (Symbolic Optimal Assembly Program) ήταν μια γλώσσα assembly γαι τον IBM 650 γραμμένη από τον Stan Poley το 1955[60].
Οι γλώσσες assembly εξουδετέρωσαν σε πολύ μεγάλο βαθμό τον επιρρεπή σε λάθη, βαρετό και χρονοβόρο τρόπο προγραμματισμού πρώτης γενιάς ο οποίος ήταν απαραίτητος για τους πρώτους υπολογιστές, απαλλάσσοντας τους προγραμματιστές από την ανάγκη απομνημόνευσης

56 *"Concept 14 Macros". MVS Software. Retrieved May 25, 2009.*

57 *Answers.com. "assembly language: Definition and Much More from Answers.com". Retrieved 2008-06-19.*

58 *Provinciano, Brian. "NESHLA: The High Level, Open Source, 6502 Assembler for the Nintendo Entertainment System".*

59 *Salomon. Assemblers and Loaders (PDF). p. 7. Retrieved 2012-01-17.*

60 *"The IBM 650 Magnetic Drum Calculator". Retrieved 2012-01-17.*

αριθμητικών κωδικών και τον υπολογισμό διευθύνσεων. Για ένα πολύ μεγάλο χρονικό διάστημα οι γλώσσες assembly υπήρξαν ευρύτατα χρησιμοποιούμενες. Ωστόσο κατά την δεκαετία του 80 (δεκαετία του 90 για τους μικροϋπολογιστές) η χρήση τους αντικαταστάθηκε σε μεγάλο βαθμό από την χρήση γλωσσών προγραμματισμού υψηλού επιπέδου θυσία στον βωμό της επίτευξης βελτιωμένης παραγωγής κώδικα. Οι σημερινές γλώσσες assembly χρησιμοποιούνται ακόμη για άμεση χειραγώγηση του υλικού του μηχανήματος, πρόσβαση σε εξειδικευμένες εντολές επεξεργαστή ή την προσπάθεια αντιμετώπισης κρίσιμων θεμάτων υπολογιστικής απόδοσης. Τυπικές εφαρμογές της assembly στις μέρες μας είναι η ανάπτυξη προγραμμάτων οδηγών, χαμηλού επιπέδου ενσωματωμένα συστήματα και συστήματα πραγματικού χρόνου.

Ιστορικά πάρα πολλά προγράμματα έχουν γραφεί αποκλειστικά στην γλώσσα assembly. Ολόκληρα λειτουργικά συστήματα γράφτηκαν εξ' ολοκλήρου σε assembly έως και την εισαγωγή του Burroughs MCP το 1961 το οποίο ήταν γραμμένο σε Executive Systems Problem Oriented Language (ESPOL) μια διαλέκτου της Algol. Πολλές εμπορικές εφαρμογές γράφτηκαν σε assembly όπως ένα μεγάλο μέρος του λογισμικού mainframe της IBM. Ωστόσο ένα μεγάλο μέρος από τα λογισμικά αυτά ξαναγράφτηκε στη συνέχεια σε COBOL, FORTRAN και PL/I αν και ένας αριθμός μεγάλων οργανισμών διατήρησε τον κώδικα σε assembly έως και τα μέσα της δεκαετίας του 90.

Οι πρώτοι μικροϋπολογιστές βασίστηκαν σε κώδικα assembly γραμμένο στο χέρι, συμπεριλαμβανομένων και των περισσότερων λειτουργικών συστημάτων και μεγάλων εφαρμογών. Αυτό συνέβη κυρίως λόγω των ιδιαίτερα μεγάλων περιορισμών σε πόρους οι οποίο επέβαλαν ιδιοσυγκρασιακή χρήση της μνήμης και λόγω ιδιαίτερων αρχιτεκτονικών παρουσίασης και είχε ως αποτέλεσμα περιορισμένων και προβληματικών υπηρεσιών συστήματος. Ενδεχομένως η βασικότερη αιτία χρήσης της assembly να ήταν απλά η έλλειψη πρωτοκλασάτων μεταφραστών γλωσσών υψηλού επιπέδου κατάλληλων για χρήση στους μικροϋπολογιστές. Ένας ψυχολογικός παράγοντας σίγουρα επίσης διαδραμάτισε κάποιο ρόλο στην αρχική στροφή στην assembly. Η πρώτη γενιά προγραμματιστών μικροϋπολογιστών διατήρησε ένα ύφος χομπίστα το οποίο δεν απέβαλε ποτέ.

Από άποψη εμπορικότητας οι σοβαρότεροι λόγοι χρήσης της assembly ήταν οι, μινιμαλιστικό αποτέλεσμα (μέγεθος), μεγαλύτερη ταχύτητα και αξιοπιστία.

Χαρακτηριστικά παραδείγματα μεγάλων προγραμμάτων γραμμένων σε assembly από εκείνη την εποχή είναι το DOS της IBM ένα λειτουργικό για προσωπικούς υπολογιστές καθώς και οι πρώτες εκδόσεις εφαρμογών όπως το spreadsheet και τα Lotus 1-2-3. Ακόμα και μέσα στην δεκαετία του 1990 τα περισσότερα παιχνίδια των παιχνιδομηχανών ήταν γραμμένα σε assembly συμπεριλαμβανομένων των παιχνιδιών για το Mega Drive/Genesis και το Super Nintendo Entertainment System. Σύμφωνα με εσωτερικές πληροφορίες η γλώσσα assembly ήταν η γλώσσα η οποία εκμεταλλευόταν με τον καλύτερο τρόπο και απόδοση τα χαρακτηριστικά του Sega Saturn, μιας κονσόλας η οποία ήταν διάσημη για το πόσο προκλητική ήταν για την ανάπτυξη προγραμμάτων[61]. Το παιχνίδι NBA Jam είναι ένα χαρακτηριστικό παράδειγμα.

Η γλώσσα assembly αποτέλεσε για πολλά χρόνια την κύρια γλώσσα ανάπτυξης εφαρμογών για

61 Eidolon's Inn: SegaBase Saturn

πολλούς διάσημους προσωπικούς υπολογιστές των δεκαετιών του 80 και 90 (όπως ήταν οι Sinclair, ZX Spectrum, Commodore 64, Commodore Amiga και Atari ST). Αυτό συνέβη κυρίως λόγω των μεταφρασμένων διαλέκτων BASIC που υπήρχαν σε αυτά τα συστήματα και οι οποίες πρόσφεραν αναποτελεσματική ταχύτητα εκτέλεσης καθώς και ανεπαρκείς δομές ανήμπορες να εκμεταλευτούν πλήρως τις δυνατότητες του υλικού στα συστήματα αυτά. Κάποια από τα συστήματα αυτά διέθεταν και ένα ολοκληρωμένο περιβάλλον ανάπτυξης λογισμικού με υψηλές δυνατότητες αποσφαλμάτωσης και μακροχαρακτηριστικών.
Ο συναρμολογητής για τον VIC-20 που γράφτηκς από τον Don French και δημοσιοποιήθηκε από τον French Silk. Με μήκος 1639 bytes ο συγγραφέας του θεωρεί είναι ο μικρότερος συναρμολογητής που γράφτηκε ποτέ. Ο συναρμολογητής υποστήριζε την συνιθισμένη συμβολική διευθυνσιοδότηση και τον ορισμό των συμβολοσειρών χαρακτήρων ή των δεκαεξαδικών συμβολοσειρών. Επίσης επιτρέπει εξαιρέσεις εκφράσεων οι οποίες μπορούν να συνδυαστούν με την πρόσθεση, αφαίρεση, πολλαπλασιασμό, διαίρεση, λογικό και, λογικό Ή και τελεστές ύψωσης σε δύναμη[62].

2.2.10.2 Χρήση στις μέρες μας

Ανέκαθεν υπήρχαν διαμάχες σχετικά με την χρησιμότητα και την απόδοση της assembly σε σύγκριση με τις σημερινές γλώσσες υψηλού επιπέδου[63]. Η assembly έχει πολύ συγκεκριμένες χρήσεις για τις οποίες είναι πολύ σημαντική[646566]. Ο συναρμολογητής μπορεί να χρησιμοποιηθεί για την βελτιστοποίηση της ταχύτητας εκτέλεσης και για βελτιστοποίηση στο μέγεθος του παραγόμενου λογισμικού[6768]. Στην περίπτωση της βελτιστοποίησης της ταχύτητας εκτέλεσης οι σύγχρονοι βελτιστοποιητές μεταφραστές ισχυρίζονται ότι μεταφράζουν τις γλώσσες υψηλού επιπέδου σε κώδικα ο οποίος μπορεί να τρέξει εξίσου γρήγορα με τα γραμμένα στο χέρι προγράμματα assembly παρά τα αντιπαραδείγματα τα οποία υπάρχουν[69]. Η πολυπλοκότητα των σύγχρονων κεντρικών μονάδων επεξεργασίας και τα υποσυστήματα διαχείρισης μνήμης κάνει την διαδικασία βελτιστοποίησης αυξανόμενα δύσκολη τόσο για τους μεταφραστές όσο και για τους

---

62 *Jim Lawless (2004-05-21). "Speaking with Don French : The Man Behind the French Silk Assembler Tools". Archived from the original on 21 August 2008. Retrieved 2008-07-25.*

63 *"TIOBE Index for May 2016". Retrieved 2016-05-17.*

64 *"Writing the Fastest Code, by Hand, for Fun: A Human Computer Keeps Speeding Up Chips". New York Times, John Markoff. 2005-11-28. Retrieved 2010-03-04.*

65 *"Bit-field-badness". hardwarebug.org. 2010-01-30. Archived from the original on 5 February 2010. Retrieved 2010-03-04.*

66 *"GCC makes a mess". HardwareBug.org. 2009-05-13. Archived from the original on 16 March 2010. Retrieved 2010-03-04.*

67 *Randall Hyde. "The Great Debate". Archived from the original on 16 June 2008. Retrieved 2008-07-03.*

68 *"Code sourcery fails again". hardwarebug.org. 2010-01-30. Archived from the original on 2 April 2010. Retrieved 2010-03-04.*

69 *Click, Cliff. "A Crash Course in Modern Hardware". Retrieved May 1, 2014.*

προγραμματιστές που γράφουν assembly. Επιπλέον η ολοένα και αυξανόμενη ταχύτητα των επεξεργαστών συνεπάγεται οι περισσότερες μονάδες επεξεργασίας να δουλεύουν την περισσότερη ώρα σε χαμηλές ταχύτητες με καθυστερήσεις να προκύπτουν μόνο από προβλέψιμα μποτιλιαρίσματα στην κίνηση των δεδομένων, σε αστοχίες της μνήμης cache, διαδικασίες εισόδου εξόδου και την σελιδοποίηση της μνήμης. Όλα τα παραπάνω έχουν καταστήσει τους περισσότερους προγραμματιστές αδιάφορους για την ταχύτητα εκτέλεσης των προγραμμάτων.
Υπάρχουν όμως ακόμα κάποιες περιπτώσεις στις οποίες ο προγραμματιστής μπορεί να επιλέξει την χρήση της assembly:

- Στην περίπτωση που είναι απαραίτητη η δημιουργία ενός αυτόνομου και μικρού μεγέθους εκτελέσιμου το οποίο να πρέπει να εκτελείται δίχως να ανατρέχει στους πόρους του χρόνου εκτέλεσης ή σε βιβλιοθήκες που σχετίζονται με γλώσσες υψηλού επιπέδου. Ίσως αυτή να είναι και η χαρακτηριστικότερη περίπτωση. Για παράδειγμα το firmware των τηλεφώνων ή των αυτοκινητιστικών συστημάτων διαχείρισης καυσίμου και ανάφλεξης, τα συστήματα διαχείρισης των κλιματιστικών, τα συστήματα ασφαλείας και τα συστήματα των αισθητήρων.
- Ο κώδικας ο οποίος πρέπει να αλληλεπιδρά άμεσα με το υλικό, όπως είναι για παράδειγμα τα προγράμματα οδήγησης των συσκευών και οι χειριστές διακοπών.
- Σε έναν ενσωματωμένο επεξεργαστή ή DSP, οι υψηλής επαναληπτικότητας διακοπές απαιτούν τον μικρότερο δυνατό αριθμό κύκλων ανά διακοπή, όπως μια διακοπή ή οποία παρουσιάζεται 1000 ή 10000 φορές το δευτερόλεπτο.
- Προγράμματα τα οποία χρειάζεται να κάνουν χρήση συγκεκριμένων εντολών επεξεργαστή οι οποίες δεν βρίσκονται υλοποιημένες από κάποιον μεταφραστή. Ένα κοινό παράδειγμα είναι η εντολή κύλισης bit στον πυρήνα πολλών αλγορίθμων κρυπτογράφησης, όπως επίσης και η αναζήτηση της ισοτιμίας του ενός byte ή την μεταφορά των 4 bit μιας πρόσθεσης.
- Πρόγραμματα τα οποία παράγουν διανυσματικές μεθόδους για προγράμματα γλωσσών υψηλού επιπέδου όπως η C. Στην γλώσσα υψηλότερου επιπέδου αυτό το χαρακτηριστικό κάποιες φορές προάγεται από εγγενείς μεθόδους του μεταφραστή οι οποίες οδηγούν απευθείας σε μνημονικά SIMD και τα οποία παρόλα αυτά οδηγούν σε ένα αποτέλεσμα μετατροπής σε assembly ένα προς ένα σχετικό με τον δεδομένο επεξεργαστή διανύσματος.
- Πρόγραμματα τα οποία απαιτούν ακραία βελτιστοποίηση, όπως για παράδειγμα ένας εσωτερικός βρόχος σε έναν αλγόριθμο που είναι ιδιαίτερα απαιτητικός από τον επεξεργαστή. Η προγραμματιστές παιχνιδιών εκμεταλλεύονται τις δυνατότητες που τους παρέχουν τα χαρακτηριστικά του υλικού καταφέρνοντας να κάνουν τα παιχνίδια να τρέξουν γρηγορότερα. Επίσης μεγάλες επιστημονικές προσομοιώσεις απαιτούν ιδιαιτέρως βελτιστοποιημένους αλγορίθμους, για παράδειγμα η γραμμική άλγεβρα με την BLAS[7071] ή ο

70 *"Writing the Fastest Code, by Hand, for Fun: A Human Computer Keeps Speeding Up Chips". New York Times, John Markoff. 2005-11-28. Retrieved 2010-03-04.*

71 *"BLAS Benchmark-August2008". eigen.tuxfamily.org. 2008-08-01. Retrieved 2010-03-04.*

διακριτός μετασχηματισμός συνημιτόνου[72].

- Περιπτώσεις τις οποίες καμιά γλώσσα υψηλού επιπέδου δεν μπορεί να καλύψει, για παράδειγμα σε ένα καινούριο επεξεργαστή.
- Προγράμματα τα οποία απαιτούν ακριβή συγχρονισμό όπως:
    - Προγράμματα πραγματικού χρόνου όπως είναι οι προσομοιωτές, συστήματα πλοήγησης πτήσης, και ιατρικός εξοπλισμος. Για παράδειγμα σε ένα fly-by-wire σύστημα τα δεδομένα της τηλεμετρίας πρέπει να λαμβάνονται, να ερμηνεύονται και να λαμβάνονται δράσεις εντός πολύ αυστηρών χρονικών ορίων. Είναι ζωτικής σημασίας για τέτοια συστήματα να εξαλειφθούν πλήρως όλες οι πιθανές πηγές απρόβλεπτης καθυστέρησης οι οποίες μπορεί να προκύψουν από ορισμένες γλώσσες μεταφρασμένες για διαφόρους λόγους όπως ας πούμε το garbage collection, τις διαδικασίες σελιδοποίησης κλπ. Όμως κάποιες από τις γλώσσες υψηλού επιπέδου ενσωματώνουν χαρακτηριστικά πραγματικού χρόνου τα οποία είναι πιθανό να προκαλέσουν τέτοιες καθυστερήσεις. Επιλέγοντας την assembly ή τις γλώσσες χαμηλότερου επιπέδου για εφαρμογές σαν τις παραπάνω οι προγραμματιστές έχουν αυξημένη ορατότητα και έλεγχο πάνω στις ενδεχόμενες καθυστερήσεις που μπορεί να προκύψει.
    - Κρυπτογραφικοί αλγόριθμοι οι οποίοι θα πρέπει πάντα να να απαιτούν τον ίδιο χρόνο εκτέλεσης αποτρέποντας έτσι επιθέσεις συγχρονισμού.
- Καταστάσεις στις οποίες απαιτείται απόλυτος έλεγχος πάνω στο περιβάλλον, περιπτώσεις με απαιτήσεις υψηλής ασφάλειας στις οποίες τίποτα δεν μπορεί να θεωρηθεί δεδομένο.
- Ιοί υπολογιστών, προγράμματα φόρτωσης λειτουργικών και συγκεκριμένα προγράμματα οδηγοί ή άλλες συσκευές που είναι πολύ κοντα στο υλικό ή χαμηλού επιπέδου λειτουργικά συστήματα.
- Προσομοιωτές συνόλου εντολών για μόνιτορς, παρακολούθηση κώδικα και αποσφαλμάτωση όπου η πρόσθετη επιβάρυνση πρέπει να κρατείται στο ελάχιστο.
- Αντίστροφη μηχανική και τροποποίηση αρχείων προγραμμάτων σαν
    - δυαδικά αρχεία τα οποία μπορεί ή όχι να έχουν γραφεί αρχικά σε μια γλώσσα υψηλού επιπέδου, για παράδειγμα όταν γίνεται απόπειρα αναπαραγωγής προγραμμάτων για τα οποία ο πηγαίος κώδικας δεν είναι διαθέσιμος ή έχει χαθεί ή όταν προσπαθούμε να αντιπαρέλθουμε την προστασία λογισμικού ή την προστασία αντιγραφής.
    - Παιχνίδια (μέθοδος γνωστή και ως χακάρισμα ROM) η οποία είναι εφικτή με διάφορες μεθόδους. Η πλέον ευρέως χρησιμοποιούμενη είναι η μετατροπή του κώδικα του προγράμματος σε επίπεδο γλώσσας assembly.
- Αυτότροποποιούμενος κώδικας, μια διαδικασία στην οποία η assembly τα πηγαίνει αρκετά καλά.
- Παιχνίδια και άλλο λογισμικό για υπολογιστές γραφικών[73].

---

72 *"x264.git/common/x86/dct-32.asm". git.videolan.org. 2010-09-29. Retrieved 2010-09-29.*

Η γλώσσα assembly διδάσκεται ακόμα στα περισσότερα τμήματα επιστήμης υπολογιστών και ηλεκτρονικής μηχανικής. Αν και ελάχιστοι προγραμματιστές σήμερα δουλεύουν συστηματικά με τη γλώσσα assembly ως εργαλείο τα βασικά θέματα της assembly παραμένουν πολύ σημαντικά. Τέτοια θεμελιώδη ζητήματα όπως είναι η δυαδική αριθμητική, κατανομή μνήμης, επεξεργασία στοίβας, προγραμματισμός συνόλου χαρακτήρων, επεξεργασία διακοπών και σχεδιασμός μεταφραστών θα ήταν θέματα πολύ δύσκολο να μελετηθούν με λεπτομέρεια δίχως διεξοδική γνώση του τρόπου λειτουργίας του υπολογιστή σε επίπεδο υλικού. Μιας και η συμπεριφορά του υπολογιστή θεμελιωδώς προσδιορίζεται από το σύνολο των εντολών μηχανής που υλοποιεί ο λογικότερος δρόμος μελέτης των θεμάτων αυτών είναι μέσω της μελέτης μια γλώσσας assembly. Οι περισσότεροι σύγχρονοι υπολογιστές έχουν παραπλήσια σύνολα εντολών μηχανής. Για αυτό η μελέτη μιας γλώσσας assembly είναι πολύ αποτελεσματική στην εκμάθηση ι) των βασικών εννοιών, ιι)στην αναγνώριση καταστάσεων όπου η χρήση της γλώσσας assembly ενδείκνυται, ιιι) στο πόσο αποτελεσματικός εκτελέσιμος κώδικας μπορεί να παραχθεί από τις γλώσσες υψηλού επιπέδου. [74]Η κατάσταση είναι απολύτως ανάλογη με την ανάγκη να μάθουν τα μικρά παιδιά τις βασικές αριθμητικές πράξεις παρά το γεγονός ότι σήμερα οι υπολογιστές χρησιμοποιούνται ευρέως για όλους τους υπολογισμούς πέραν των πολύ απλών.

2.2.11 Τυπικές εφαρμογές

- Η γλώσσα assembly τυπικά χρησιμοποιείται στο κώδικα εκκίνησης ενός συστήματος, ο χαμηλού επιπέδου κώδικας ο οποίος αρχικοποιεί και δοκιμάζει το υλικό του συστήματος πριν την διαδικασία φόρτωσης του λειτουργικού και συχνά αποθηκεύεται σε κάποια ROM (το BIOS στους συμβατούς με IBM προσωπικούς υπολογιστές και συστήματα CP/M είναι χαρακτηριστικά παραδείγματα).
- Ορισμένοι μεταφραστές μεταφράζουν τις γλώσσες υψηλού επιπέδου πρώτα σε εντολές assembly πριν παράξουν τον τελικό εκτελέσιμο κώδικα, δίνοντας την δυνατότητα ο κώδικας assembly να εξεταστεί πριν την παραγωγή του τελικού εκτελέσιμο κώδικα.
- Σχετικά χαμηλού επιπέδου γλώσσες προγραμματισμού όπως η C επιτρέπουν στον προγραμματιστή να ενσωματώσει εντολές της γλώσσας assembly απευθείας στον πηγαίο κώδικα. Προγράμματα που κάνουν χρήση αυτών των υποδομών, όπως ας πούμε ο πυρήνας του Linux μπορούν εν συνεχεία να κατασκευάσουν αφαιρέσεις χρησιμοποιώντας διαφορετικές γλώσσες assembly σε διαφορετικές πλατφόρμες υλικού. Ο φορητός κώδικας του συστήματος μπορεί εν συνεχεία να χρησιμοποιήσει αυτά τα συστατικά που είναι εξαρτώμενα από τον εκάστοτε επεξεργαστή μέσω ενός συστήματος διεπαφής.
- Η γλώσσα assembly είναι όπως έχουμε ξαναπεί ιδιαιτέρως χρήσιμη στην αντίστροφη μηχανική. Πολλά προγράμματα διανέμονται αποκλειστικά και μόνο σε μορφή κώδικα

---

73 *"68K Programming in Fargo II". Archived from the original on 2 July 2008. Retrieved 2008-07-03.*

74 *Hyde, Randall (1996-09-30). "Foreword ("Why would anyone learn this stuff?"), op. cit.". Archived from the original on 25 March 2010. Retrieved 2010-03-05.*

μηχανής η οποία μπορεί να μεταφραστεί αρκετά εύκολα σε κώδικα assembly αλλά μάλλον δύσκολα σε κώδικα γλώσσας υψηλού επιπέδου. Τέτοια εργαλεία όπως είναι το Interactive Disassampler κάνουν εκτεταμένη χρήση της αποσυναρμολόγησης για τον σκοπό αυτό.

- Οι συναρμολογητές μπορούν να χρησιμοποιηθούν για την δημιουργία τμημάτων δεδομένων δίχως την επίβλεψη κάποιας γλώσσας υψηλού επιπέδου από μορφοποιημένο και σχολιασμένο πηγαίο κώδικα .

2.3 Η γλώσσα προγραμματισμού C

2.3.1 Εισαγωγή

Εικόνα 5 Ken Thompson (αριστερά) με τον Dennis Ritchie (δεξιά, ο εφευρέτης της γλώσσας C)

Η C είναι γενικού σκοπού επιτακτική γλώσσα προγραμματισμού η οποία υποστηρίζει τον δομημένο προγραμματισμό, λεξιλόγιο μεταβλητού πεδίου και αναδρομή ενώ ένα στατικό σύστημα γραφής εμποδίζει την πολλές ακούσιες εργασίες. Βάση σχεδιασμού η C παρέχει κατασκευές οι οποίες μετατρέπονται αποτελεσματικά σε τυπικές εντολές μηχανής και για αυτό τον λόγο βρήκε και διαρκή εφαρμογή σε εφαρμογές οι οποίες είχαν κωδικοποιηθεί αρχικά σε γλώσσα assembly, όπως είναι τα λειτουργικά συστήματα όπως επίσης και σε διάφορες εφαρμογές λογισμικού για υπολογιστές σε όλο το εύρος από υπερυπολογιστές έως και ενσωματωμένα συστήματα.
Η C αρχικά αναπτύχθηκε από τον Dennis Ritchie μεταξύ 1969 και 1973 στα AT&T Bell Labs[75] και χρησιμοποιήθηκε για την επανυλοποίηση του λειτουργικού συστήματος Unnix. [76]Εκτοτε είναι μια από τις ευρύτερα χρησιμοποιημένες γλώσσες προγραμματισμού ό.λων των εποχών[7778] με τους μεταφραστές της C από διάφορους προμηθευτές να είναι διαθέσιμοι για την πλειονότητα των υπαρκτών υπολιγστικών αρχιτεκτονικών και λειτουργικών συστημάτων. Η C πτουποποιήθηκε από το Amerivan National Standards Institute (ANSI) από το 1989 (βλέπε ANSI C) ακολούθως από τον International Organization fot Standarization (ISO).

2.3.2 Σχεδιασμός

Η C είναι μια επιτακτική (διαδικασιακή) γλώσσα. Σχεδιάστηκε ώστε να μεταγλωττίζεται από ένα αρκετά ευθύ μεταγλωττιστή, για να παρέχει πρόσβαση χαμηλού επιπέδου στην μνήμη, για να παρέχει γλωσσικές δομές οι οποίες χαρτογραφούνται αποτελεσματικά σε εντολές μηχανής και για

---

75 Ritchie (1993)

76 *Lawlis, Patricia K. (August 1997). "Guidelines for Choosing a Computer Language: Support for the Visionary Organization". Ada Information Clearinghouse. Retrieved 18 July 2006.*

77 *"Programming Language Popularity". 2009. Retrieved 16 January 2009.*

78 *"TIOBE Programming Community Index". 2009. Retrieved 6 May 2009.*

να απαιτεί την ελάχιστη δυνατή υποστήριξη χρόνου εκτέλεσης. Χάρη σε αυτά τα χαρακτηριστικά η C είναι κατάλληλη για πολλά διαφορετικά είδη εφαρμογών οι οποίες στο παρελθόν κωδικοποιούσαμε σε γλώσσα assembly, όπως είναι ο προγραμματισμός συστημάτων.
Παρά τις δυνατότητες χαμηλού επιπέδου όμως η γλώσσα ενθάρρυνε και τον διαπλοτφορμικό προγραμματισμό. Ένα πρόγραμμα C γραμμένο σύμφωνα με τα πρότυπα μπορεί να μεταγλωττιστή για μια μεγάλη γκάμα υπολογιστικών πλατφορμών και λειτουργικών συστημάτων με ελάχιστες αλλαγές στον πηγαίο του κώδικα. Η γλώσσα έγινε διαθέσιμη σε μια ευρύτατη γκάμα υπολογιστικών πλατφορμών από μικροελεγκτές έως υπερυπολογιστές.

2.3.3 Επισκόπηση

Στην C όλος ο εκτελέσιμος κώδικας περιέχεται μέσα σε υπορουτίνες οι οποίες αποκαλούνται functions (αν και δεν πρόκειται για functions με την αυστηρή έννοια του διαδικασιακού προγραμματισμού). Οι παράμετροι των functions πάντοτε περνάνε μέσω τιμής . Το πέρασμα μέσω αναφοράς προσομοιώνεται στην C αποκλειστικά με το πέρασμα τιμών δεικτών (pointers). Το κείμενο του πηγαίου κώδικα της C είναι ελεύθερης μορφοποίησης χρησιμοποιώντας το ερωτηματικό για τη σήμανση του τέλους και τις αγκύλες για τον προσδιορισμό των σωμάτων κώδικα.
Η γλώσσα C επίδεικνύει επίσης και τα παρακάτψ χαρακτηριστικά:
Υπάρχει ένας μικρός αριθμός δεσμευμένων λέξεων ο οποίος περιέχει και ένα ένα υποσύνολο εντολών ελέγχου της ροής: for, if/else, while, switch, και do/while. Υπάρχει μόνο ένας χώρος ονομάτων και τα ονόματα τα οποία ορίζονται από τον χρήστη δεν διαχωρίζονται από τις δεσμευμένες λέξεις με κανενός είδους σήμανση.
Υπάρχει ένας μεγάλος αριθμός αριθμητικών και λογικών τελεστών όπως +, +=, ++, &, ~ κλπ.
Περισσότερες από μια αναθέσεις μπορούν να εκτελεστούν σε μια δήλωση.
Οι τιμές που επιστρέφονται από τις συναρτήσεις μπορούν να αγνοηθούν αν δεν είναι χρήσιμες.
Το γράψιμο είναι στατικό. Όλα τα δεδομένα έχουν τύπο αλλά είναι δυνατές έμμεσες μετατροπές, για παράδειγμα οι χαρακτήρες μπορούν να χρησιμοποιηθούν και ως ακέραιοι.
Η δήλωση του συντακτικού μιμείται το πλαίσιο χρήσης, η C δεν διαθέτει εντολή define, αντί αυτής κάθε δήλωση η οποία ξεκινάει με το όνομα ενός τύπου λογίζεται ως δήλωση. Δεν υπάρχει η δεσμευμένη λέξη function, αντί αυτού οι functions υπονοούνται από τις παρενθέσεις μιας λίστας παραμέτρων.
Είναι δυνατοί τύποι ορισμένοι από τον χρήστη (typedef) όπως επίσης και οι σύνθετοι τύποι:
Ετερογενή σύνολα τύπων δεδομένων (structs) επιτρέπουν συνδεδεμένα στοιχεία δεδομένων να είναι προσβάσιμα και να αναθέτονται σαν μονάδες.
Η δημιουργία ευρετηρίων πινάκων είναι επίσης μια δυνατότητα οριζόμενη με όρους αριθμητικής δεικτών. Σε αντίθεση με τα structs οι πίνακες δεν είναι πρώτης κλάσεως αντικείμενα και δεν μπορούν να ανατεθούν ή να συγκριθούν χρησιμοποιώντας ενσωματωμένους τελεστές. Δεν υπάρχει η δεσμευμένη λέξη array για χρήση κατά τον ορισμό. Αντί αυτής αγκύλες υποδεικνύουν συντακτικά τους πίνακες, για παράαδειγμα month[11].

Τύποι απαρίθμησης είναι δυνατοί με την χρήση της δεσμευμένης λέξης enum. Δεν είναι χαρακτηρισμένοι και είναι ελεύθερα ενδομετατρέψιμοι σε ακεραίους.
Οι συμβολοσειρές δεν είναι ξεχωριστός τύπος δεδομένων, αλλά είναι συμβατικά υλοποιημένες σαν κενού περιεχομένου πίνακες χαρακτήρων.
Η χαμηλού επιπέδου πρόσβαση στην μνήμη είναι δυνατή με την μετατροπή διευθύνσεων μηχανής σε δείκτες.
Οι διαδικασίες (υπορουτίνες δίχως τιμές επιστροφής) είναι ειδικές περιπτώσεις function στις οποίες ως τύπος δεδομένων επιστροφής δεν ορίζεται το void.
Οι functions δεν μπορούν να οριστούν μέσα στο λεξικό εύρος άλλων function.
Οι functions και οι δετες δεδομένων επιτρέπουν πολυμορφισμό χρόνου εκτέλεσης.
Ένας προεπεξεργαστής εκτελεί ορισμό μακροεντολών, περίληψη αρχείων του πηγαίου κώδικα και μεταγλώτιση συνθηκών.
Υπάρχει μια βασική μορφή σπονδύλωσης. Τα αρχεία μπορού να να μεταγλωττίζονται ξεχωριστά και να ενώνονται μαζί με έλεγχο πάνω στο ποιες functions και αντικείμενα δεδομένων θα είναι ορατά σε άλλα αρχεία μέσω των χαρακτηριστικών static και extern.
Η σύνθετη χρηστικότητα όπως η είσοδος/έξοδος, διαχείριση των συμβολοσειρών και οι μαθηματικές συναρτήσεις αναθέτονται σε εξωτερικές βιβλιοθήκες.
Η C δεν περιλαμβάνει κάποιες από τις δυνατότητες των πιο σύγχρονων υψηλού επιπέδου γλωσσών προγραμματισμού όπως είναι είναι η αντικειμενοστρέφεια και η αποκομιδή σκουπιδιών.

2.3.4 Σχέσεις με άλλες γλώσσες

Πολλές μεταγενέστερες γλώσσες δανείστηκαν από την C είτα άμεσα είτε έμμεσα μεταξύ των οποίων και οι C++, D, Go, Rust, Java, JavaScript, Limbo, LPC, C#, Objective-C, Perl, PHP, Python, Verilog (γλώσσα περιγραφής υλικού)[79] και η C shell του Unix. Αυτές οι γλώσσες τράβηξαν πολλές από τις δομές ελέγχους τους και άλλα βασικά τους χαρακτηριστικά από την C. Οι περισσότερες από αυτές (με την Python να είναι το πιο χαρακτηριστικό παράδειγμα) μοιάζουν πολύ με την C και συντακτικά και τείνουν να να συνδυάζουν τις αναγνωρίσιμες εκφράσεις του συντακτικού της C με τους τύπους και τα μοντέλα δεδομένων να είναι αρκετά διαφορετικά.

2.3.5 Ιστορία

2.3.5.1 Αρχικές εξελίξεις

Η απαρχή της C είναι πολύ στενά δεμένη με την ανάπτυξη του λειτουργικού συστήματος Unix, το οποίο αρχικά είχε υλοποιηθεί σε assembly σε ένα PDP-7 από τον Ritchie και τον Thompson ενσωματώνοντας πολλές από τις ιδέες συναδέλφων τους. Τελικά αποφάσισαν να μεταφέρουν το

79 *"Verilog HDL (and C)" (PDF). The Research School of Computer Science at the Australian National University. 2010-06-03. Retrieved 2013-08-19.* 1980s: ; Verilog first introduced ; Verilog inspired by the C programming language

λειτουργικό σύστημα σε ένα PDP-11. Η αρχική υλοποίηση του unix ήταν σε γλώσσα assembly. Οι προγραμματιστές σκέφτηκαν να το ξαναγράψουν σε Β γλώσσα. Την απλοποιημένη έκδοση της BCPL[80] του Thompson. Ωστόσο η ανικανότητα της Β να χρησιμοποιήσει κάποια από τα χαρακτηριστικά του PDP-11 κυρίως την διευθυσνιοδότηση των byte οδήγησε στην χρήση της C.
Η ανάπτυξη ξεκίνησε το 1972 στο σύστημα Unix[81] του PDP-11 και πρωτοεμφανίστηκε στην δεύτερη έκδοση του Unix[82]. Η γλώσσα δεν σχεδιάστηκε αρχικά με το στόχο της μεταφερσιμότητας, ωστόσο πολύ σύντομα βρέθηκε νε εκτελείται σε πολλές διαφορετικές πλατφόρμες. Ένας μεταγλωττιστής για το Honeywell 6000 γράφτηκε τον πρώτο χρόνο της ιστορίας της C ενώ σύντομα ακολούθησε και μια υλοποίηση για το IBM System/370[8384]. Το γράμμα C επιλέχθηκε ως απλή συνέχεια της αλφαβητικής τάξης και μιας και υπήρχε ήση η B[85].
Το 1972 επίσης ένας μεγάλο κομμάτι του Unix ξαναγράφτηκε σε C[86]. Μέχρι το 1973 με την προσθήκη των structs η C ήταν τόσο ισχυρή που το μεγαλύτερο κομμάτι του πυρήνα του Unix ήταν γραμμένο πλέον σε C.
Το Unix ήταν ένα από τα πρώτα λειτουργικά συστήματα των οποίων ο πυρήνας ήταν γραμμένος σε γλώσσα άλλη από την assembly (προηγήθηκαν τα Multics γραμμένο σε PL/I και MCP (Master Control Program) για το Burroughs B5000 γραμμένο σε ALGOL το 1961). Το 1977 οι Circa, Richie kai Stephen C. Johnson έκαναν περαιτέρω αλλαγές στη γλώσσα για να προσδώσουν υποδομές μεταφερσιμότητας στο Unix. Ο φορητός μεταγλωττιστής C του johnson υπήρξε η βάση για τις διάφορες υλοποιήσεις της C σε νέες πλατφόρμες[87].

2.3.5.2 K&RC

Το 1978 οι Brian Kernighan και Dennis Ritchie εξέδωσαν την πρώτη έκδοση του The C

---

80 *Ritchie, Dennis M. (March 1993). "The Development of the C Language". ACM SIGPLAN Notices* ***28*** *(3): 201–208. doi:10.1145/155360.155580.*

81 *Johnson, S. C.; Ritchie, D. M. (1978). "Portability of C Programs and the UNIX System". Bell System Tech. J.* ***57*** *(6): 2021–2048. doi:10.1002/j.1538-7305.1978.tb02141.x. Retrieved 16 December 2012.* (Note: this reference is an OCR scan of the original, and contains an OCR glitch rendering "IBM 370" as "IBM 310".)

82 *McIlroy, M. D. (1987). A Research Unix reader: annotated excerpts from the Programmer's Manual, 1971–1986 (PDF) (Technical report). CSTR. Bell Labs. p. 10. 139.*

83 *Kernighan, Brian W.; Ritchie, Dennis M. (February 1978). The C Programming Language (1st ed.). Englewood Cliffs, NJ: Prentice Hall. ISBN 0-13-110163-3.* Regarded by many to be the authoritative reference on C.

84 *Johnson, S. C.; Ritchie, D. M. (1978). "Portability of C Programs and the UNIX System". Bell System Tech. J.* ***57*** *(6): 2021–2048. doi:10.1002/j.1538-7305.1978.tb02141.x. Retrieved 16 December 2012.* (Note: this

85 Ulf Bilting & Jan Skansholm "Vägen till C" (Swedish) meaning "The Road to C", third edition, Studentlitteratur, year 2000, page 3. ISBN 91-44-01468-6.

86 Stallings, William. "Operating Systems: Internals and Design Principles" 5th ed, page 91. Pearson Education, Inc. 2005.

87 *Johnson, S. C.; Ritchie, D. M. (1978). "Portability of C Programs and the UNIX System". Bell System Tech. J.* ***57*** *(6): 2021–2048. doi:10.1002/j.1538-7305.1978.tb02141.x. Retrieved 16 December 2012.* (Note: this

Programming Language[88]. Αυτό το βιβλίο γνωστό στους προγραμματιστές της C ως “K&R” υπηρέτησε για πολλά χρόνια σαν η ανεπίσημη τεκμηρίωση της γλώσσας. Η έκδοση της γλώσσας την οποία περιγράφει είναι γνωστή ως K&R C. Η δεύτερη έκδοση του βιβλίου[89] περιγράφει το μεταγενέστερο πρότυπο ANSI της C που περιγράφεται παρακάτω.

Το K&R εισήγαγε διάφορα χαρακτηριστικά της γλώσσας:

- standard I/O library
- long int data type
- unsigned int data type
- σύνθετοι τελεστές ανάθεσης της μορφής =op όπως =- άλλαξαν σε μορφή op= δηλαδή -= για να εξουδετερώσουν την ασάφεια την οποία δημιουργούσαν εντολές όπως i=-10 η οποία μεταφραζόταν σαν μείωση του i κατά 10 αντί για ανάθεσε στο i το -10.

Ακόμα και μετά την έκδοση του 1980 προτύπου ANSI για πολλά χρόνια το K&R C πολλοί προγραμματιστές το θεωρούσαν τον ελάχιστο κοινό παρανομαστή στον οποίο οι προγραμματιστές περιόριζαν τους εαυτούς τους όταν το ζητούμενο ήταν η μέγιστη μεταφερσιμότητα μιας και πολλοί παλαιότεροι μεταγλωττιστές ήταν ακόμη ενεργοί και μιας και ο προσεκτικά γραμμένος K&R C κώδικας αποτελούσε ένα C πρότυπο.

Στις πρώιμες εκδόσεις της C μόνο οι συναρτήσεις οι οποίες επέστρεφαν κάτι άλλο πέραν του int έπρεπε να δηλωθούν αν χρησιμοποιούνταν πριν τον ορισμό της συνάρτησης. Κάθε συνάρτηση η οποία χρησιμοποιούνταν πριν τον ορισμό της θεωρείτο ότι επέστρεφε τύπο int αν η τιμή της έπρεπε να χρησιμοποιηθεί.

Για παράδειγμα

```
long some_function();
/* int */ other_function();

/* int */ calling_function()
{
   long test1;
   register /* int */ test2;

   test1 = some_function();
   if (test1 > 0)
        test2 = 0;
   else
        test2 = other_function();
   return test2;
}
```

Οι ορισμοί τύπου int οι οποίοι βρίσκονται εδώ ανενεργοί με την μορφή σχολίων μπορούσαν να παραλειφθούν στην γραφή K&R C, ωστόσο απαιτούνται στα μεταγενέστερα πρότυπα.

---

88 *Kernighan, Brian W.; Ritchie, Dennis M. (February 1978). The C Programming Language (1st ed.). Englewood Cliffs, NJ: Prentice Hall. ISBN 0-13-110163-3.* Regarded by many to be the authoritative reference on C.

89 *Kernighan, Brian W.; Ritchie, Dennis M. (March 1988). The C Programming Language (2nd ed.). Englewood Cliffs, NJ: Prentice Hall. ISBN 0-13-110362-8.*

Μιας και στην K&R C οι δηλώσεις των συναρτήσεων δεν περιέχουν καμιά πληροφορία για τα ορίσματα των συναρτήσεων, ο έλεγχος τύπων των παραμέτρων των συναρτήσεων δεν γίνεται αν και κάποιοι μεταγλωττιστές θα εκδώσουν ένα μήνυμα προειδοποίησης εάν η κλήση μιας τοπικής συνάρτησης γίνει με τον λάθος αριθμό ορισμάτων. Ξεχωριστά εργαλεία όπως ήταν το Lint του unix αναπτύχθηκαν προκειμένου να εκτελούν έλεγχο ορθότητας μεταξύ των συναρτήσεων σε διαφορετικά αρχεία πηγαίου κώδικα.
Στα χρόνια που ακολούθησαν την έκδοση του K&R C, πολλά χαρακτηριστικά προστέθηκαν στην γλώσσα τα οποία υποστηρίζονται από του μεταγλωττιστές της AT&T (κυρίως τον PCC[90]) και από μεταγλωττιστές άλλων κατασκευαστών. Τέτοια χαρακτηριστικά είναι:

10. void συναρτήσεις
11. συναρτήσεις με επιστρεφόμενο τύπο struct ή union.
12. Αναθέσεις για τύπους δεδομένων struct
13. enumerated τύπους

Ο μεγάλος αριθμός επεκτάσεων και η μη αποδοχή χρήσης μια βιβλιοθήκες μαζί με την δημοτικότητα της γλώσσας και το γεγονός ότι ούτε οι μεταγλωττιστές του UNIX δεν υλοποιούσαν αποκλειστικά την K&R τυποποίηση οδήγησαν στην ανάγκη προτυποποίησης.

2.3.5.3 ANSI C και ISO C

Κατά το τέλος της δεκαετίας του 70 και στις αρχές της δεκαετίας του 80 εκδόσεις της C είχαν υλοποιηθεί για μια μεγάλη ποικιλία mainframe υπολογιστών. Minicomputers και microcomputers μαζί και το IBM PC, καθώς η δημοτικότητα της άαρχιζει να αυξάνεται σημαντικά.
Το 1983 το American National Standards Institute (ANSI) σχημάτισε μια επιτροπή την X3J11 για να καθιερώσει μια πρότυπη τεκμηρίωση της C. Η X3J11 βασίστηκε στο πρότυπο της C το οποίο χρησιμοποιήθηκε στην υλοποίηση του Unix. Ωστόσο το μη μεταφέρσιμο κομμάτι της βιβλιοθήκης του Unix δόθηκε στην IEEE eorking group1003 για να γίνει η βάση για το 1988 POSIX πρότυπο.
Το 1989 το πρότυπο της C κυρώθηκε σαν ANSI X3.159-1989 “Γλώσσα προγραμματισμού C”. Αυτή η έκδοση της γλώσσας συχνά καλείται και ως ANSI C, Standard C ή μερικές φορές C89.
Το 1990 πρότυπο ANSI της C (με αλλαγές μορφών) υιοθετήθηκε από το International Organization for Standarization (ISO) σαν ISO/IEC 9899:1990, το οποίο πολλές φορές αποκαλέιται και C90/ Έτσι οι όροι C89 και C90 αναφέρονται ουσιαστικά στην ίδια γλώσσα προγραμματισμού.
Το ANSI και άλλα εθνικά συμβούλια προτυποποίησης δεν αναπτύσουν πλέον το πρότυπο της C ανεξάρτητα αλλά με αναφορές στο διεθνές πρότυπο της C το οποίο διατηρείται από την ομάδα ISO/IEC.
Ένας από τους στόχους της διαδικασίας προτυποποίησης της C ήταν η παραγωγή ενός υπερσυνόλου της K&R C ενσωματώνοντας πολλά από τα ανεπίσημα χαρακτηριστικά που ακολούθησαν. Η επιτροπή προτύπων επίσης συμπεριέλαβε αρκετά επιπλέον χαρακτηριστικά όπως πρότυπα συναρτήσεων (από την C++) , κενούς δείκτες, υποστήριξη για διεθνή σύνολα χαρακτήρων καθώς και βελτιώσεις προεπεξεργαστή. Αν και το συντακτικό για την δήλωση παραμέτρων

90 *Stroustrup, Bjarne (2002). Sibling rivalry: C and C++ (PDF) (Report). AT&T Labs.*

επαυξήθηκε για να συμπεριλάβει και το στυλ που χρησιμοποιεί η C++ το συντακτικό που προτείνεται από το K&R C συνεχίστηκε να επιτρέπεται προκειμένου να εξυπηρετηθεί η συμβατότητα με τον ήδη υπάρχοντα κώδικα.
Η C 89 υποστηρίζεται σήμερα από τους σημερινούς C μεταγλωττιστές και περισσότεροες κώδικας που γράφεται πλέον στηρίζεται σε αυτή. Κάθε πρόγραμμα που γράφεται σε Standar C και δίχως υποθέσεις εξάρτησης από το υλικό θα τρέξει απροβλημάτιστα σε οποιαδήποτε σύμφωνη υλοποίηση της C εντός βέβαια των ορίων των πόρων. Δίχως τέτοιες προφυλάξεις τα προγράμματα θα μεταγλωττιστούν μόνο σε ορισμένες πλατφόρμες ή μόνο με ορισμένο μεταγλωττιστή με το πρόβλημα να οφείλεται για παράδειγμα στην χρήση μη τυποποιημένων βιβλιοθηκών, όπως είνα οι βιβλιοθήκες γραφικών περιβαλλόντων διεπαφής ή σε εξαρτήσεις από χαρακτηριστικά εξαρτημένα από τον μεταγλωττιστή ή την πλατφόρμα όπως είναι το ακριβές μέγεθος των τύπων δεδομένων. Σε περιπτώσεις όπου ο κώδικας πρέπει να είναι μεταγλωττίσιμος είτε από σύμφωνους με πρότυπο μεταγλωττιστές είτε από μεταγλωττιστές που έχουν ως βάση την K&R C η μακροεντολή _STDC_ μπορεί να χρησιμοποιηθεί για να διαχωρίσει των κώδικα σε τυποποιημένα και K&R τμήματα για αποτρέψει την χρήση σε ένα μεταγλωττιστή K&R C χαρακτηριστικών τα οποία είναι διαθέσιμα μόνο στην Standard C.
Μετά τις διαδικασίες τυποποίησης ANSI/ISO οι προδιαγραφές της γλώσσας C παρέμειναν σχετικά δίχως επεκτάσεις για αρκετά χρόνια. Μόλις το 1995 προέκυψε η πρώτη κανονιστική τροπολογία (ISO/IEC 9899/AMD1:1995) με στόχο να διορθώσει κάποιες λεπτομέρειες και για να προσθέσει περαιτέρω υποστήριξη διεθνών συνόλων χαρακτήρων.

2.3.5.4 C99

Η τυποποιημένη C αναθεωρήθηκε επιπλέον στα τέλη της δεκαετίας του 1990 οδηγώντας στην έκδοση το ISO/IEC9899:1999 το 1999 στο οποίο αναφερόμαστε σαν C99. Έκτοτε έχουν προκύψει τρεις τροπολογίες με τεχνικές διορθώσεις[91].
Η C99 εισήγαγε αρκετά νέα χαρακτηριστικά συμπεριλαμβανομένων και των γραμμικών συναρτήσεων, πολλούς νέους τύπους δεδομένων (όπως οι long long in και ένα σύνθετο τύπο για την αναπαράσταση σύνθετων αριθμών), πίνακες με μεταβλητό μήκος και ευέλικτα μέλη πινάκων, βελτίωσε την υποστήριξη για τους IEEE 754 αριθμούς κινητής υποδιαστολής, υποστήριξη για μεταβλητές μακροεντολές και υποστήριξη για σχόλια μιας γραμμής τα οποία ξεκινούν με //, όπως και την BCPL ή στην C++. Τα περισσότερα από αυτά τα χαρακτηριστικά είχαν ήδη υλοποιηθεί δε διάφορους μεταγλωττιστές σαν επεκτάσεις.
Η C99 είναι στο μεγαλύτερο κομμάτι της συμβατή προς τα πίσω με την C90 αλλά είναι αυστηρότερη κατά κάποιους τρόπους. Ποιο συγκεκριμένα μια δήλωση η οποία δεν έχει έναν προσδιοριζόμενο τύπο στην C99 δεν αποδίδει αυτόματα τον τύπο int. Μια τυποποιημένη έκδοση μακροεντολών _STDC_VERSION_ ορίζεται με την τιμή 199901L για να υποδηλώσει ότι η υποστήριξη της C99 είναι διαθέσιμη και πλέον. Οι GCC, Solaris Studio και άλλοι μεταγλωττιστές

91 *"JTC1/SC22/WG14 – C". Home page. ISO/IEC. Retrieved 2 June 2011.*

της C υποστηρίζουν τα περισσότερα από αυτά τα κομμάτια της C99 για συμβατότητα με την C++11[92].

2.3.5.5 C11

Το 1997 ξεκίνησαν εργασίες για μια νέα αναθεώρηση του προτύπου της C το οποίο είχε το όνομα C1X έως και την επίσημη ημερομηνία κυκλοφορίας του στις 08-12-2011. Η επιτροπή προτύπων της C υιοθέτησε γραμμές για να περιορίσει την ενσωμάτωση νέων χαρακτηριστικών τα οποία δεν είχαν ελεγχθεί από τις υπάρχουσες υλοποιήσεις.
Το πρότυπο C11 προσθέτει ένα μεγάλο αριθμό νέων χαρακτηριστικών στην C και στην βιβλιοθήκη της, όπως είναι οι γενικού τύπου μακροεντολές, οι ανώνυμες δομές, η βελτιωμένη υποστήριξη του Unicode, πολυνηματικότητα και συναρτήσεις με ελέγχους ορίων. Επιπλέον κάνει κάποια τμήματα της βιβλιοθήκης της C99 προαιρετικά και βελτιώνει την συμβατότητα με την C++. Το πρότυπο των μακροεντολών _STDC_VERSION_ ορίζεται ως 201112Λ για να δείξει ότι η υποστήριξη της C11 είναι διαθέσιμη.

2.3.5.6 Ενσωματωμένη C

Ιστορικά ο ενσωματωμένος προγραμματισμός C απαιτεί επεκτάσεις της C εκτός των προτύπων ώστε να είναι σε θέση να υποστηρίξει εξωτικά χαρακτηριστικά όπως είναι η αριθμητική σταθερών σημείων, πολλαπλές ξεχωριστές τράπεζες μνήμης και βασικές I/O λειτουργίες.
Το 2008 η επιτροπή προτύπων της C εξέδωσε μια τεχνική αναφορά η οποία επέκτεινε την C για να επιλύσει αυτά τα ζητήματα παρέχοντας ένα κοινό πρότυπο προς υιοθέτηση[93]. Περιείχε ένα σύνολο χαρακτηριστικών τα οποία δεν ήταν διαθέσιμα στην κανονική C.

2.3.6 Συντακτικό

Η C διαθέτει γραμματική η οποία καθορίζεται από το πρότυπο της[94]. Σε αντίθεση με γλώσσες σαν την FORTRAN77 ο πηγαίος κώδικας της C είναι ελεύθερης μορφής γεγονός το οποίο επιτρέπει την αυθαίρετη χρήση του κενού χώρου για την μορφοποίηση του κώδικα δίχως να έχει περιορισμούς στηλών ή γραμμών κειμένου. Ωστόσο τα όρια των γραμμών παραμένουν σημαντικά κατά την φάση της προεπεξεργασίας. Τα σχόλια μπορούν να εμφανίζονται είτε μεταξύ των περιοριστών /* */ είτε (από την C99 και έπειτα) ακολουθώντας το // και μέχρι το τέλος της γραμμής. Τα σχόλια που περιορίζονται με τα /* */ δεν μπορούν να είναι εμφωλευμένα και αυτές οι ακολουθίες χαρακτήρων δεν μεταφράζονται σαν σχόλια όταν εμφανίζονται στο εσωτερικό συμβολοσειρών ή ακολουθιών

92 *Andrew Binstock (October 12, 2011). "Interview with Herb Sutter". Dr. Dobbs. Retrieved September 7, 2013.*

93 *"TR 18037: Embedded C" (PDF). ISO / IEC. Retrieved 26 July 2011.*

94 *Harbison, Samuel P.; Steele, Guy L. (2002). C: A Reference Manual (5th ed.). Englewood Cliffs, NJ: Prentice Hall. ISBN 0-13-089592-X.* Contains a BNF grammar for C.

χαρακτήρων[95].

Τα πηγαία αρχεία της C περιέχουν δηλώσεις και ορισμούς συναρτήσεων. Οι ορισμοί των συναρτήσεων περιέχουν δηλώσεις. Οι δηλώσεις είτε ορίζουν νέους τύπους χρησιμοποιώντας δεσμευμένες λέξεις όπως το struct, το union και το enum, ή αναθέτουν τύπους και πιθανώς δεσμεύουν αποθηκευτικό χώρο για νέες μεταβλητές, συνήθως γράφοντας τον τύπο της μεταβλητής ακολουθούμενο από το όνομα της. Δεσμευμένες λέξεις σαν τις char και int ορίζουν ενσωματωμένους τύπους μεταβλητών. Τα τμήματα κώδικα περικλείονται σε αγκύλες για να περιορίσουν το εύρος των δηλώσεων και για να δουλέψουν ως όριο για τις δομές ελεχου.

Μια επιτακτική γλώσσα σαν την C χρησιμοποιεί δηλώσεις για να ορίσει δράσεις. Η ποιο κοινή δήλωση είναι μια δήλωση έκφρασης, η οποία περιέχει μια έκφραση προς αξιολόγηση ακολουθούμενη από ένα ερωτηματικό. Σαν παρενέργεια της αξιολόγησης οι συναρτήσεις μπορούν να κληθούν και οι μεταβλητές μπορεί να λάβουν καινούριες τιμές. Για να τροποποιήσουμε την κανονική σειρά εκτέλεσης των εντολών η C παρέχει διάφορες εντολές ελέγχου της ροής οι οποίες είναι αναγνωρίσιμες με κάποιες δεσμευμένες λέξεις. Ο δομημένος προγραμματισμός υποστηρίζεται από την υπό συνθήκη εκτέλεση (if-else) και από τους βρόχους επανάληψης (for, while, do while). Η δήλωση for εκτελεί ξεχωριστά την αρχικοποίηση, τον έλεγχο και την έκφραση επαναρχικοποίησης κάθε ένα από τα οποία είναι δυνατόν να παραληφθεί. Εκφράσεις σαν την break και την continue δίνουν την δυνατότητα να μείνει χωρίς εκτέλεση τμήμα του επαναλαμβανόμενου κώδικα ή να εκτελεστεί επαναρχικοποίση του βρόχου. Υπάρχει επίσης και η goto, μια εντολή η οποία δεν αποτελεί δομή του δομημένου προγραμματισμού η οποία μεταφέρει κατευθείαν τον έλεγχο προγράμματος κατευθείαν σε κάποιο σημασμένο σημείο του κώδικα. Η switch επιλέγει την εκτέλεση κάποιας εντολής κατά συνθήκη βάση μια έκφρασης ακεραίων.

Οι εκφράσεις μπορούν να χρησιμοποιούν μια μεγάλη ποικιλία ενσωματωμένων τελεστών και μπορούν να περιέχουν και κλήσεις συναρτήσεων. Η σειρά με την οποία τα ορίσματα με τις συναρτήσεις και τους τελεστές αξιολογούνται είναι ακαθόριστη. Οι αξιολογήσεις μπορεί να είναι ένθετες. Ωστόσο όλες οι “παρενέργειες” (συμπεριλαμβανομένης της αποθήκευσης των μεταβλητών) θα συμβούν πριν το επόμενο σημείο ακολουθίας. Σημείο ακολουθίας λογίζεται και το τέλος της κάθε έκφρασης εντολής καθώς και τα σημεία εισόδου και επιστροφής από κάθε συνάρτηση. Σημεία ακολουθίας επίσης εμφανίζονται κατά την αξιολόγηση εκφράσεων που περιέχουν συγκεκριμένους τελεστές (&&, ||, ?: και ο τελεστής κόμμα). Αυτό επιτρέπει ένα υψηλό βαθμό βελτιστοποίησης του αντικειμενικού κώδικα από τον μεταγλωττιστή αλλά απαιτεί από τους προγραμματιστές C να δείχνουν περισσότερη προσοχή ώστε να παίρνουν αξιόπιστα από ότι απαιτείται από άλλες γλώσσες προγραμματισμού.

Οι Krnighan και Ritchie λένε στην εισαγωγή του The C Programming Language: “Η C όπως κάθε άλλη γλώσσα έχει τις αδυναμίες της. Μερικοί από τους λειτουργούς έχουν λανθασμένο προβάδισμα και κάποια μέρη του συντακτικού θα μπορούσαν να είναι καλύτερα[96]. Η τυποποίηση της C δεν προσπάθησε να διορθώσει κάποια από τα προβλήματα αυτά εξ' αιτίας του αντίκτυπου των αλλαγών

95 *Kernighan, Brian W.; Ritchie, Dennis M. (1996). The C Programming Language (2nd ed.). Prentice Hall. p. 192. ISBN 7 302 02412 X.*

96 Page 3 of the original K&R[1]

αυτών στον ήδη υπάρχον λογισμικό.

2.3.7 Σύνολο χαρακτήρων

Τα βασικό σύνολο χαρακτήρων της C περιλαμβάνει τους κάτωθι χαρακτήρες:

- Κεφαλαία και μικρά γράμματα: a-z, A-Z
- Δεκαδικά ψηφία: 0-9
- Γραφικούς χαρακτήρες: ! " # % & ' ( ) * + , - . / : ; < = > ? [\ ] ^ _ { | } ~
- Κενούς χαρακτήρες: κενό, tab οριζόντιο, tab κάθετο, χαρακτήρας νέας γραμμής

Ο χαρακτήρας νέας γραμμής υποδηλώνει το τέλος μια γραμμής κειμένου. Δεν είναι απαραίτητο να αντιδρά σαν ένας μόνο χαρακτήρας αλλά ή C του συμπεριφέρεται έτσι.
Επιπλέον multibyte κωδικοποιημένοι χαρακτήρες μπορεί να χρησιμοποιηθούν σε συμβολοσειρές αλλά δεν είναι απολύτως μεταφέρσιμοι. Το τελευταίο πρότυπο C (C11) επιτρέπει πολυεθνικούς unicode χαρακτήρες να ενσωματωθούν και να είναι μεταφέρσιμοι μέσα στον πηγαίο κώδικα της C χρησιμοποιώντας \uXXXX ή \UXXXXXXXX κωδικοποίηση (όπου X υπονοείται ένας δεκαεξαδικός χαρακτήρας), αν και αυτό το χαρακτηριστικό δεν έχει ακόμη ευρέως χρησιμοποιηθεί.
Το βασικό σύνολο χαρακτήρων εκτέλεσης της C περιέχει τους ίδιους χαρακτήρες μαζί με εκφράσεις για τα alert, bacspace και το carriage return. Η υποστήριξη εκτενούς συνόλου χαρακτήρων σε χρόνο εκτέλεσης μεγάλωσε σε κάθε αναθεώρηση του πρωτύπου της C.

2.3.8 Δεσμευμένες λέξεις

Η C89 έχει 32 δεσμευμένες λέξεις

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Η C99 πρόσθεσε πέντε επιπλέον:

_Bool	_Imaginary	restrict
_Complex	inline	

Η C11 πρόσθεσε επτά ακόμη97

_Alignas	_Atomic	_Noreturn	_Thread_
_Alignof	_Generic	_Static_assert	local

Οι πιο πρόσφατες προσθήκες δεσμευμένων λέξεων ξεκινούν με μια κάτω παύλα ακολουθούμενη

97 ISO/IEC 9899:201x (ISO C11) Committee Draft

από ένα κεφαλαίο γράμμα, γιατί αναγνωριστικά σαν αυτής της μορφής είχαν κρατηθεί από τα πρώτυπα της C παλαιότερα για χρήση μόνο σε υλοποιήσεις. Αφού ο υπάρχον πηγαίος κώδικας δεν θα έπρεπε να χρησιμοποιεί αυτά τα αναγνωριστικά δεν θα υπήρχαν προβλήματα όταν οι υλοποιήσεις της C θα ξεκινούσαν να υποστηρίζουν αυτές τις επεκτάσεις της γλώσσας. Κάποιες τυποποιημένες κεφαλίδες ορίζουν περισσότερο βολικά για αναγνωριστικά με κάτω παύλα. Η γλώσσα στο παρελθόν περιείχε την δεσμευμένη λέξη entry η οποία όμως σπανίως υλοποιείτο και έτσι εγκαταλείφθηκε[98].

2.3.9 Τελεστές

Η C διαθέτει ένα πολύ πλούσιο σύνολο τελεστών οι οποίοι είναι σύμβολα τα οποία χρησιμοποιούνται στο εσωτερικό εκφράσεων για τον ορισμό των χειρισμών που πρέπει να συμβούν κατά την αξιολόγηση της έκφρασης. Η C έχει τους παρακάτω τελεστές:

- αριθμητικοι: +, -, *, /, %
- ανάθεσης: =
- επαξυημένης ανάθεσης: +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=
- λογικής bit: ~, &, |, ^
- μετάθεσης bit: <<, >>
- τελεστές boolean: !, &&, ||
- αξιολόγησης καταστάσεων: ? :
- ελεγκτές ισότητας: ==, !=
- κλήσης συναρτήσεων: ( )
- increment and decrement: ++, --
- member selection: ., ->
- object size: sizeof
- order relations: <, <=, >, >=
- αύξησης και μείωσης: &, *, [ ]
- σειριοποίησης: ,
- ομαδοποίηση υποεκφράσεων: ( )
- αλλαγή τύπου: (*typename*)

Η C χρησιμοποιεί τον τελεστή = στις μαθηματικές εκφράσεις για να εκφράσει ισότητα, για να υποδηλώσει ανάθεση ακολουθώντας το παράδειγμα της Fortran και της PL/I αλλά όζι της ALGOL και των απογόνων της. Η ομοιότητα μεταξύ των τελεστών της C για ανάθεση και έλεγχο ισότητας (==) έγινε αντικείμενο κρητικής καθώς ελοχεύει ο κίνδυνος της χρήσης του ενός στη θέση του άλλου. Σε πολλές περιπτώσεις είναι δυνατόν ο ένας να χρησιμοποιηθεί αντί του άλλου δίχως να παραχθεί λάθος μεταγλώττισης (αν και κάποιοι μεταγλωττιστές μπορεί να πετάξουν προειδοποιήσεις). Για παράδειγμα η έκφραση ελέγχου στην if(a=b+1) είναι αληθής αν ο a δεν είναι

98 *Kernighan, Brian W.; Ritchie, Dennis M. (1996). The C Programming Language (2nd ed.). Prentice Hall. pp. 192, 259. ISBN 7 302 02412 X.*

μηδέν μετά την ανάθεση[99]. Επιπλέον οι προκάτοχοι των τελεστών της C δεν είναι ενστικτώδεις όπως το == οδηγώντας σε ισχυρότερες δεσμεύσεις από ότι οι τελεστές & και | σε εκφράσεις σαν x&1==0 η οποία θα έπρεπε να γραφεί (x&1)==0 για να αξιολογηθεί σωστά[100].

2.3.10 Τύποι δεδομένων

Το σύστημα τύπων στην C είναι στατικό και αδύναμης δακτυλογράφησης και άρα παρόμοιο με αυτό των απογόνων της ALGOL όπως και η Pascal[101]. Υπάρχουν ενσωματωμένοι τύποι για ακεραίους διαφόρων μεγεθών, με πρόσημο και χωρίς, αριθμούς κινητής υποδιαστολής, χαρακτήρες και enums. Η C99 πρόσθεσε των τύπο των boolean. Υπάρχουν επίσης και συμπληρωματικοί τύποι όπως είναι οι πίνακες, οι δείκτες, τα records και τα structs καθώς επίσης και μη ταυτοποιημένοι όπως είναι τα unions.
Η C συχνά χρησιμοποιείται σε προγραμματισμό συστημάτων χαμηλού επιπέδου όπου χρειάζονται λύσεις πέραν των λύσεων που χρησιμοποιούνται στον προγραμματισμό συστημάτων, αλλά ο προγραμματιστής μπορεί να υπερκεράσει τους ελέγχους με πολλούς τρόπους είτε χρησιμοποιώντας μια αλλαγή τύπου για να μετατρέψει μια τιμή από τον ένα τύπο σε έναν άλλο ή με την χρήση δεικτών ή unions ώστε να πετύχει την επανερμηνεία των υπογραμμισμένων bits των δεδομένων των αντικειμένων με κάποιον άλλο τρόπο[102].
Μερικοί θεωρούν το συντακτικό δήλωσης της C δυσνόητο, ειδικά στον τομέα των δεικτών των συναρτήσεων. (Η εισήγηση του Ritchie ήταν να δηλώνονται αναγνωριστικά στα περιεχόμενα ανάλογα με την χρήση τους.)
Οι αριθμητικές μετατροπές της C επιτρέπουν την δημιουργία αποτελεσματικού κώδικα αλλά δεν είναι λίγες φορές κατά τις οποίες τα αποτελέσματα είναι απρόσμενα. Για παράδειγμα η σύγκριση ισότητας προσημασμένων και μη ακεραίων ίσου εύρους πολλές φορές απαιτεί τροποποίηση της προσημασμένης τιμής σε μη προσημασμένη. Αυτό μπορεί να οδηγήσει σε απροσδόκητα αποτελέσματα εάν η προσημασμένη τιμή είναι αρνητική.

2.3.11 Δείκτες

Η C υποστηρίζει την χρήση δεικτών, ένα τύπο αναφοράς ο οποίος καταγράφει την διεύθυνση ή την τοποθεσία ενός αντικειμένου ή μιας συνάρτησης στην μνήμη. Οι δείκτες μπορεί να αναχθούν στο να έχουν πρόσβαση σε δεδομένα τα οποία είναι αποθηκευμένα στην διεύθυνση στην οποία δείχνουν ή να επικαλούνται μια συνάρτηση στην οποία δείχνουν. Οι ο χειρισμός των δεικτών

---

99 *"10 Common Programming Mistakes in C++". Cs.ucr.edu. Retrieved 26 June 2009.*

100 *Schultz, Thomas (2004). C and the 8051 (3rd ed.). Otsego, MI: PageFree Publishing Inc. p. 20. ISBN 1-58961-237-X. Retrieved 10 February 2012.*

101 *Feuer, Alan R.; Gehani, Narain H. (March 1982). "Comparison of the Programming Languages C and Pascal". ACM Computing Surveys **14** (1): 73–92. doi:10.1145/356869.356872.*

102 Page 122 of K&R2[14]

γίνεται χρησιμοποιώντας αναθέσεις ή αριθμητική δεικτών. Η παρουσίαση ενός δείκτη σε χρόνο εκτέλεσης είναι συνήθως μια διεύθυνση μνήμης συνήθως επαυξημένης η μετατοπισμένης κατά το μέγεθος του τύπου δεδομένων στο οποίο δείχνουν, όμως αφού ο κάθε δείκτης περιέχει τον τύπο του αντικειμένου στο οποίο δείχνει οι εκφράσεις οι οποίες περιέχουν δείκτες μπορούν να αξιολογηθούν και κατά τον χρόνο μεταγλώττισης. Η αριθμητική δεικτών κλιμακώνεται αυτόματα από το μέγεθος του τύπου δεδομένων στον οποίο δείχνουν. Οι δείκτες έχουν πάρα πολλές χρήσεις στην C. Οι συμβολοσειρές κειμένου συνήθως διαχειρίζονται με την χρήση δεικτών οι οποίοι δείχνουν σε πίνακες χαρακτήρων,. Η δυναμική δέσμευση μνήμης γίνεται με τη χρήση δεικτών. Πολύ τύποι δεδομένων όπως είναι τα δέντρα συνήθως υλοποιούνται σαν δυναμικά αντικείμενα τύπου struct τα οποία συνδέονται μεταξύ τους μέσω δεικτών. Οι δείκτες σε συναρτήσεις είναι χρήσιμοι για το πέρασμα συναρτήσεων σαν ορίσματα σε συναρτήσεις υψηλότερου επιπέδου (όπως είναι οι qsort ή η bsearch) ή σαν επανκλήσεις οι οποίες πυροδοτούνται από τους χειριστές συμβάντων[103].

Ένας δείκτης με κενή τιμή δεν δείχνει σε καμιά δόκιμη τοποθεσία. Η αποανεφοροποίηση ενός κενού δείκτη δεν είναι ορισμένη και συχνά οδηγεί σε σφάλμα κατάτμησης. Οι τιμές των κενών δεικτών είναι χρήσιμες για να υποδεικνύουν ειδικές περιπτώσεις όπως ένας μη επόμενος δείκτης στον τελευταίο κόμβο μιας συνδεδεμένης λίστας ή σας μια ένδειξη λάθους από συναρτήσεις οι οποίες επιστρέφουν δείκτες. Σε κατάλληλα πλαίσια στον πηγαίο κώδικα όπως ας πούμε στην ανάθεση μιας μεταβλητής δείκτη μια σταθερά κενού δείκτη μπορεί να γραφεί ως μηδέν με ή χωρίς σαφής μετατροπή σε τύπο δείκτη ή όπως η κενή μακροεντολή ορίστηκε από πολλές κεφαλίδες προτύπων. Στα περιεχόμενα υποθετικών οι δείκτες κενής τιμής αξιολογούνται ως false, ενώ κάθε άλλη τιμή ως true.

Οι κενοί δείκτες (void *) δείχνουν σε αντικείμενα ακαθόριστου τύπου και άρα μπορούν να χρησιμοποιηθούν σαν γενικής φύσεως δείκτες δεδομένων. Αφού του μέγεθος και ο τύπος, του αντικειμένου στο οποίο δείχνουν δεν είναι γνωστός, οι κενοί δείκτες αυτής της περίπτωσης δεν είναι δυνατόν να αποαναφεροπιηθούν ούτε και είναι εφικτή η αριθμητική δεικτών για αυτούς αν και μπορούν αρκετά εύκολα να μετατραπούν σε ή από οποιουδήποτε τύπου αντικειμένου δείκτες[104].

Η απρόσεκτη χρήση των δεικτών είναι δυνητικά επικίνδυνη. Επειδή γενικά δεν ελέγχεται μια μεταβλητή δείκτη μπορεί να δείχνει σε κάποια αυθαίρετη τοποθεσία, κατάσταση η οποία μπορεί να οδηγήσει στην εμφάνιση μη επιθυμητών αποτελεσμάτων. Αν και οι σωστά χρησιμοποιούμενοι δείκτες δείχνουν σε ασφαλείς τοποθεσίες, μπορεί να στραφούν προς μη ασφαλείς τοποθεσίες απλά και μόνο λόγω χρήσης άτοπης αριθμητικής δεικτών. Τα αντικείμενα στα οποία δείχνουν μπορεί να χάσουν το τμήμα μνήμης στο οποίο είναι αποθηκευμένα και να χρησιμοποιηθούν (dangling δείκτες), μπορεί να χρησιμοποιηθούν δίχως να αρχικοποιηθούν (wild δείκτες) ή μπορεί να ανατεθούν απευθείας σε μια μη ασφαλή τιμή χρησιμοποιώντας μια cast ένα union ή μέσω ενός άλλου κατεστραμμένου δείκτη. Γενικά η C επιτρέπει την διαχείριση των και την μετατροπή μεταξύ των τύπων των δεικτών, αν και συνήθως οι μεταγλωττιστές παρέχουν δυνατότητες για πολλά διαφορετικά επίπεδα ελέγχου. Άλλες γλώσσες προγραμματισμού λύνουν αυτού του είδους τα

103 *Klemens, Ben (2013). 21st Century C. O'Reilly Media. ISBN 1-4493-2714-1.*

104 *Klemens, Ben (2013). 21st Century C. O'Reilly Media. ISBN 1-4493-2714-1.*

προβλήματα κάνοντας χρήση πιο περιοστικών τύπων αναφοράς.

2.3.12 Πίνακες

Οι τύποι των πινάκων στη C είναι παραδοσιακά σταθερού και προκαθορισμένου μεγέθους το οποίο καθορίζεται κατά τον χρόνο μεταγλώττισης (το τελευταίο πρότυπο της C η C99 παρέχει την δυνατότητα μιας μορφής πινάκων μεταβλητού μήκους). Ωστόσο είναι δυνατόν να γίνει δέσμευση ενός τμήματος μνήμης (αυθαίρετου μεγέθους) σε χρόνο εκτέλεσης χρησιμοποιώντας την συνάρτηση της βιβλιοθήκης του προτύπου malloc και να χειριστούμε από εκεί και πέρα το τμήμα αυτό σαν πίνακα. Η ένωση της C των πινάκων και των δεικτών σημαίνει ότι οι δηλωμένοι πίνακες και οι πίνακες δυναμικής δέσμευσης μνήμης είναι τύποι μεταξύ τους όμοιοι.
Αφού η πρόσβαση στους πίνακες γίνεται πάντα μέσω δεικτών οι προσβάσεις δεν υπάρχει έλεγχος για το αν οι προσβάσεις προσπαθούν να έχουν πρόσβαση πέραν των ορίων του πίνακα[105]. Αν και μερικοί μεταγλωττιστές μπορεί να παρέχουν έλεγχο ορίων σαν επιλογή. Οι παραβιάσεις των ορίων των πινάκων είναι λοιπόν πιθανές και μάλλον συχνές σε κώδικα ο οποίος έχει γραφεί επιπόλαια και μπορεί να έχουν διάφορες επιπτώσεις μεταξύ των οποίων μυ επιτρεπτές προσβάσεις μνήμης, απώλεια και καταστροφή δεδομένων, υπερφόρτωση buffers και σφάλματα χρόνου εκτέλεσης. Εάν είναι επιθυμητός ο έλεγχος ορίων θα πρέπει να γίνει χειροκίνητα.
Η C δεν έχει ειδική πρόβλεψη για πολυδιάστατους πίνακες αλλά βασίζεται κυρίως στην αναδρομικότητα μεταξύ των τύπων του συστήματος για να δηλώσει πίνακες πινάκων οι οποίοι αποτελεσματικά έχουν το ίδιο αποτέλεσμα. Ο κατάλογος τιμών του πολυδιάστατου πίνακα μπορεί να θεωρηθεί σαν μια αύξηση στην σειρά κατά παραγγελία.
Οι πολυδιάστατοι πίνακες είναι ευρέως χρησιμοποιούμενοι σε αριθμητικούς αλγόριθμους (κυρίως αλγόριθμους της γραμμικής άλγεβρας) για την αποθήκευση μητρών. Η δομή του πίνακα της C είναι επαρκής για την εκτέλεση αυτής της αποστολής. Ωστόσο αφού οι πίνακες ουσιαστικά περνάνε ως δείκτες τα όρια του πίνακα πρέπει να είναι γνωστές σταθερές τιμές ρητώς διαβιβαζόμενες σε κάθε υπορουτίνα που θέλει να τους χρησιμοποιήσει και οι δυναμικού μεγέθους πίνακες πινάκων δεν μπορούν να είναι προσβάσιμες μέσω διπλών ευρετηρίων (ένας τρόπος για να ξεπεραστεί το πρόβλημα αυτό είναι να διανείμουμε τον πίνακα σαν ένα διάνυσμα δεικτών στις στήλες του πίνακα).
Η C 99 εισήγαγε τους πίνακες μεταβλητού μήκους οι οποίοι έλυσαν πολλά αλλά όχι όλα από τα ζητήματα των απλών πινάκων της C.

2.3.13 Εναλλαξιμότητα πινάκων δεικτών

Η σημειογραφία x[i] (όπου ο είναι ένας δείκτης) είναι μια συντακτική ωραιοποίηση το *(x+i)[106].

---

105 For example, gcc provides _FORTIFY_SOURCE. *"Security Features: Compile Time Buffer Checks (FORTIFY_SOURCE)". fedoraproject.org. Retrieved 2012-08-05.*

106 *Raymond, Eric S. (11 October 1996). The New Hacker's Dictionary (3rd ed.). MIT Press. p. 432. ISBN 978-0-*

Εκμεταλευόμενοι την γνώση του μεταγλωττιστή για τον τύπο του δείκτη η θέση στην οποία δείχνει ο x+i δεν είναι η βασική διεύθυνση στην οποία δείχνε ο x επαυξημένη κατά i bytes αλλά μάλλον ορίζεται σαν η βασική διεύθυνση επαυξημένη κατά i και πολλαπλασιασμένη επί το μέγεθος ενός στοιχείου στο οποίο δείχνει ο δείκτης. Έτσι το x[i] δειχνει το i+1 στοιχείο του πίνακα.
Επιπλέον στα περιεχόμενα των περισσότερων εκφράσεων (μια εξαίρεση που πρέπει να αναφέρουμε είναι ο τελεστής του sizeof) το όνομα ενός πίνακα αυτομάτως μετατρέπεται σε ένα δείκτη στο πρώτο στοιχείο του πίνακα Αυτό υπονοεί ότι ένας πίνακας ποτέ δεν αντιγράφεται πλήρως όταν ονοματίζεται σαν όρισμα σε μια συνάρτηση αλλά μόνο σαν η διεύθυνση του πρώτου στοιχείου περνιέται. Έτσι αν και οι κλήσεις συναρτήσεων στη C χρησιμοποιούν το πέρασμα μέσω τιμών οι πίνακες τελικά περνούν μέσω αναφοράς.
Το μέγεθος ενός αντικειμένου μπορεί να προσδιοριστεί με τη χρήση του sizeof, έτσι για οποιοδήποτε αντικείμενο x n=sizeof*x ή n=sizeofx[0] και ο αριθμός των αντικειμένων σε έναν πίνακα A μπορεί να προσδιοριστεί ως εξής sizeof A/sizeof A[0]. Λόγω της σημασιολογίας της C δεν είναι δυνατόν να προσδιοριστεί ολόκληρο το μέγεθος πινάκων μέσω δεικτών ή πινάκων οι οποίοι έχουν κατασκευαστεί με δυναμική δέσμευση μνήμης (malloc). Κώδικας σαν τον ακόλουθο sizeof arr /sizeof arr[0] (όπου ο arr υποδηλώνει έναν δείκτη) δεν θα δουλέψει αφού ο μεταγλωττιστής θεωρεί ότι ζητείται το πραγματικό μέγεθος του δείκτη και όχι της θέσης στην οποία δείχνε[107][108]. Μιας και τα ονόματα των πινάκων που είναι ορίσματα στην sizeof δεν μετατρέπονται σε δείκτες δεν υπάρχει ζήτημα ασάφειας κατά τον υπολογισμό. Ωστόσο οι πίνακες οι οποίοι δημιουργούνται με δυναμική δέσμευση μνήμης είναι προσβάσιμοι μόνο μέσω δεικτών αντιμετωπίζουν όλα τα προβλήματα που αντιμετωπίζει η sizeof σαν δείκτες σε πίνακες.
Έτσι παρά την φαινομενική ισορροπία μεταξύ πινάκων και δεικτών υπάρχει ακόμα μία διάκριση να γίνει μεταξύ τους. Αν και το όνομα ενός πίνακα στις περισσότερες εκφράσεις μετατρέπεται σε ένα δείκτη (που δείχνει το πρώτο στοιχείο του), ο δείκτης αυτός δεν καταλαμβάνει κάποιο χώρο στη μνήμη. Το όνομα του πίνακα δεν είναι μια l-value και η διεύθυνση του είναι σταθερή σε αντίθεση με μια μεταβλητή δείκτη. Κατά συνέπεια αυτό στο οποίο δείχνει ένας πίνακας δεν είναι δυνατό να μεταβληθεί αν και είναι πιθανό να αναθέσουμε μια καινούρια διεύθυνση σε ένα όνομα πίνακα. Τα περιεχόμενα του πίνακα μπορεί να καταληφθούν ωστόσο με χρήση της συνάρτησης memcpy ή αλλάζοντας τα περιεχόμενα κάθε θέσης τους πίνακα ξεχωριστά

2.3.14 Διαχείριση της μνήμης

Μια από τις πλέον βασικές λειτουργίες μιας γλώσσας προγραμματισμού είναι η παραοχή υποδομών διαχείρισης της μνήμης και των αντικειμένων τα οποία είναι αποθηκευμένα εκεί. Η C παρέχει τρεις διαφορετικού τρόπους δέσμευσης μνήμης για αντικείμενα[109]:

262-68092-9. Retrieved 5 August 2012.

107 *Summit, Steve. "comp.lang.c Frequently Asked Questions 6.23". Retrieved March 6, 2013.*

108 *Summit, Steve. "comp.lang.c Frequently Asked Questions 7.28". Retrieved March 6, 2013.*

109 *Klemens, Ben (2013). 21st Century C. O'Reilly Media. ISBN 1-4493-2714-1.*

- Στατική δέσμευση μνήμης: όπου ο χώρος για το αντικείμενο παρέχεται κατά το χρόνο μεταγλώττισης. Αυτά τα αντικείμενα έχουν μια επέκταση (ζωής) για όσο χρονικό διάστημα παραμένουν φορτωμένα στην μνήμη.
- Αυτόματη δέσμευση μνήμης: όπου προσωρινά αντικείμενα μπορούν να αποθηκευτούν σε μια στοίβα και αυτός ο χώρος ελευθερώνεται αυτόματα και μπορεί να επαναχρησιμοποιηθεί μετά την έξοδο από το τμήμα τμήματος στο οποίο δηλώθηκαν.
- Δυναμική δέσμευση μνήμης: όπου τμήματα μνήμης, αυθαίρετου μεγέθους ζητούνται κατά τον χρόνο εκτέλεσης χρησιμοποιώντας συναρτήσεις της βιβλιοθήκης όπως είναι η malloc, από ένα τμήμα της μνήμης το οποίο ονομάζεται σωρός. Αυτά τα τμήματα επιμένουν έως ότου ελευθερωθούν για να επαναχρησιμοποιηθούν από τις συναρτήσεις βιβλιοθήκης realloc ή free.

Αυτές οι τρεις προσεγγίσεις είναι κατάλληλες σε διαφορετικές περιπτώσεις και συνεπάγονται η κάθε μια διάφορους συμβιβασμούς. Για παράδειγμα, η επίμονη φύση των στατικών αντικειμένων είναι χρήσιμη για την διατήρηση δεδομένων κατά την κλήση συναρτήσεων, η αυτόματη δέσμευση μνήμης είναι εύκολη στη χρήση αλλά γενικά ο χώρος της στοίβας είναι κατά πολύ περισσότερο περιορισμένος και παροδικός σε σχέση με την στατική μνήμη ή τον χώρο του σωρού και η δυναμική δέσμευση μνήμης επιτρέπει βολικές δεσμεύσεις μνήμης αλλά μόνο για αντικείμενα το μέγεθος των οποίων είναι γνωστό κατά το χρόνο εκτέλεσης. Τα περισσότερα προγράμματα γραμμένα σε C χρησιμοποιούν εκτεταμένα και τους τρεις τρόπους.

Όπου είναι δυνατόν η αυτόματη ή η στατική δέσμευση μνήμης είναι συνήθως ευκολότερες στη χρήση γιατί τον αποθηκευτικό χώρο τον διαχειρίζεται ο μεταγλωττιστής, ελευθερώνοντας τον προγραμματιστή από την επιρρεπή σε λάθη αγγαρεία της χειροκίνητης δέσμευσης μνήμης και απελευθέρωσης του χώρου. Ωστόσο πολλές δομές δεδομένων μπορεί να μεταβάλλουν το μέγεθος τους κατά τον χρόνο εκτέλεσης και έτσι αφού η στατική δέσμευση μνήμης (αλλά και η αυτόματη μέχρι την έκδοση C99) πρέπει να έχει ένα δεδομένο και σταθερό μέγεθος γνωστό κατά το χρόνο μεταγλώττισης, είμαστε αναγκασμένοι να καταφύγουμε στην δυναμική δέσμευση μνήμης. Πριν το πρότυπο της C99[110] οι πίνακες μεταβλητού μεγέθους ήταν ένα χαρακτηριστικό παράδειγμα της περίπτωσης που μόλις περιγράψαμε. Σε αντίθεση με την αυτόματη δέσμευση μνήμης η οποία μπορεί να αποτύχει κατά τον χρόνο εκτέλεσης με ανεξέλεγκτες για το πρόγραμμα συνέπειες, οι συναρτήσεις δυναμικής δέσμευσης μνήμης επιστρέφουν μια ένδειξη (με την μορφή ενός δείκτη με κενή τιμή) όταν ο απαιτούμενος χώρος δεν είναι δυνατόν να δεσμευτεί. Στην περίπτωση της αδυναμίας στατικής δέσμευσης μνήμης (λόγω υπέρβασης του διαθέσιμου χώρου) η αδυναμία ανιχνεύεται από τον συνδετήρα ή τον φορτωτή (linker or loader) πριν καν αρχίσει η εκτέλεση του προγράμματος.

Εκτός και αν έχει οριστεί διαφορετικά, τα στατικά αντικείμενα περιέχουν μηδενικές τιμές είναι κενά κατά την έναρξη εκτέλεσης του προγράμματος. Τα αυτόματα και τα δυναμικά δεσμευμένα αντικείμενα αρχικοποιούνται μόνο αν μια αρχική τιμή είναι σαφώς ορισμένη. Διαφορετικά κατά την αρχικοποίηση τους έχουν ακαθόριστες τιμές. Αν το πρόγραμμα επιχειρήσει να ανακτήσει τις μη

110 *Klemens, Ben (2013). 21st Century C. O'Reilly Media. ISBN 1-4493-2714-1.*

αρχικοποιημένες τιμές τα αποτελέσματα θα είναι επίσης ακαθόριστα. Πολλοί σύγχρονοι μεταγλωττιστές προσπαθούν να ανιχνεύσουν και να προειδοποιήσουν για αυτά τα προβλήματα χωρίς ωστόσο να έχει καταστεί δυνατή η πλήρης εξάλειψη του φαινομένου.
Ένα άλλο ζήτημα είναι ότι η δέσμευση μνήμης του σωρού πρέπει να είναι συγχρονισμένη με την πραγματική χρήση της σε κάθε πρόγραμμα ώστε να είναι όσο το δυνατόν επαναχρησιμοποιήσιμη. Για παράδειγμα, αν ο μόνος δείκτης στη μνήμη σωρού βγει εκτός εύρους ή αν η τιμή του αντικατασταθεί πριν την κλήση της free τότε τα περιεχόμενα αυτού του κομματιού της μνήμης δεν μπορούν να ανακληθούν για χρήση αργότερα και είναι πολύ βασική απώλεια για το πρόγραμμα,μια κατάσταση γνωστή και ως διαρροή μνήμης. Αντιστρόφως είναι πιθανό ένα τμήμα μνήμης να ελευθερωθεί αλλά να εξακολουθήσει να αναφέρεται με απρόβλεπτα αποτελέσματα. Τυπικά τα συμπτώματα θα εμφανιστούν σε ένα τμήμα του προγράμματος αρκετά μακρινό από το τμήμα στο οποίο παρουσιάστηκε το πραγματικό λάθος κάνοντας ιδιαίτερα δύσκολο τον εντοπισμό του. Τέτοια ζητήματα αντιμετωπίζονται αρκετά αποτελεσματικά σε γλώσσες που παρέχουν αυτόματη αποκομιδή σκουπιδιών.

2.3.15 Βιβλιοθήκες

Η γλώσσα C χρησιμοποιεί βιβλιοθήκες σαν την κύρια μέθοδο επέκτασης της. Στην C μια βιβλιοθήκη είναι ένα σύνολο συναρτήσεων μέσα σε ένα αρχείο. Κάθε βιβλιοθήκη έχει ένα αρχείο κεφαλίδα το οποίο το οποίο περιέχει τα πρότυπα των συναρτήσεων που περιέχονται στην βιβλιοθήκη και οι οποίες μπορεί να χρησιμοποιηθούν από κάποιο πρόγραμμα, καθώς επίσης και δηλώσεις τυχόν ειδικών τύπων δεδομένων και σύμβολα μακροεντολών τα οποία μπορεί να χρησιμοποιούνται με την βιβλιοθήκη αυτή. Προκειμένου να μπορέσει να χρησιμοποιήσει μια βιβλιοθήκη πρέπει αυτή να συμπεριληφθεί (με χρήση της include) στην κεφαλίδα του προγράμματος και πρέπει η βιβλιοθήκη να συνδεθεί με το πρόγραμμα. Μια διαδικασία που σε ορισμένες περιπτώσεις απαιτεί σημαφόρους μεταγλωττιστή[111].
Η πλέον κοινή βιβλιοθήκη της C είναι η C standard βιβλιοθήκη η οποία ορίζεται στα πρότυπα ANSI και ISO και ακολουθεί κάθε υλοποίηση της C. Υλοποιήσεις της γλώσσας που έχουν σαν στόχο περιορισμένα περιβάλλοντα όπως είναι τα ενσωματωμένα συστήματα μπορεί να παρέχουν κάποιο υποσύνολο της standard βιβλιοθήκης. Αυτή η βιβλιοθήκη υποστηρίζει είσοδο και έξοδο ρεύματος (δεδομένων), δέσμευση μνήμης, μαθηματικές συναρτήσεις, χαρακτήρες, συμβολοσειρές και χρονικές τιμές. Διάφορες ξεχωριστές κεφαλίδες της standar (για παράδειγμα η stdio.h) καθορίζουν τις διεπαφές για αυτές και άλλες δομές της βιβλιοθήκης.
Ένα άλλο πολύ κοινό σύνολο συναρτήσεων της C είναι εκείκνες που χρησιμοποιούνται από τις εφαρμογές και απευθύνονται αποκλειστικά σε συστήματα Unix και στους κλώνους του, ιδίως οι συναρτήσεις που παρέχουν διεπαφή με τον πυρήνα του λειτουργικού. Αυτές οι λειτουργίες περιγράφονται σε διάφορα πρότυπα όπως είναι το POSIX και η Single UNIX τεκμηρίωση.
Αφού πολλά προγράμματα έχουν γραφεί σε C υπάρχει διαθέσιμη μια μεγάλη ποικιλία άλλων

111 *Klemens, Ben (2013). 21st Century C. O'Reilly Media. ISBN 1-4493-2714-1.*

βιβλιοθηκών οι οποίες είναι συχνά γραμμένες σε C μιας και οι μεταγλωττιστές C είναι γνωστή για την παραγωγή αποδοτικού αντικειμενικού κώδικα. Οι προγραμματιστές ακολούθως δημιουργούν διεπαφές για την βιβλιοθήκη έτσι οι συναρτήσεις της να μπορούν να χρησιμοποιηθούν από γλώσσες υψηλότερου επιπέδου όπως είναι οι java, Perl και Python[112].

2.3.16 Εργαλεία της γλώσσας

Διάφορα εργαλεία έχουν δημιουργηθεί για να βοηθήσουν τους προγραμματιστές της C να αποφύγουν τα διάφορα εγγενή προβλήματα της γλώσσας όπως είναι δηλώσεις με απρόβλεπτη συμπεριφορά ή κακής πρακτικής δηλώσεις με την έννοια ότι μπορεί να οδηγήσουν σε μη επιθυμητή συμπεριφορά κατά την εκτέλεση του προγράμματος.
Ο αυτόματος έλεγχος του πηγαίου κώδικα είναι προσοδοφόρος σε κάθε γλώσσα και για την C υπάρχουν πολλά εργαλεία που το κάνουν όπως το Lint. Μια κοινή πρακτική είναι η χρήση του Lint για την ανίχνευση αμφισβητήσιμου κώδικα όταν ένα πρόγραμμα πρωτογράφεται. Εφόσον ένα πρόγραμμα περάσει από το Lint μια φορά μετά μεταγλωττίζεται από τον μεταγλωττιστή. Επίσης πολύ μεταγλωττιστές παρέχουν την δυνατότητα ειδοποίησης για κατασκευές οι οποίες παρόλο που είναι αποδεκτές συντακτικά μάλλον θα οδηγήσουν σε πρόβλημα. Το MISRA C είναι ένα σύνολο κατευθύνσεων για την αποφυγή τέτοιου αμφισβητήσιμου κώδικα το οποίο φτιάχθηκε για ενσωματωμένα συστήματα[113].
Υπάρχουν επίσης μεταγλωττιστές, βιβλιοθήκες και μηχανισμοί επιπέδου λειτουργικού συστήματος για την εκτέλεση δράσεων οι οποίες δεν αποτελούν κανονικό κομμάτι της C όπως είναι ο έλεγχος τον ορίων πινάκων, η ανίχνευση υπερχείλισης του buffer και αυτόματη αποκομιδή σκουπιδιών.
Εργαλεία σαν το Purify ή το Valgrind και η σύνδεση με βιβλιοθήκες οι οποίες περιέχουν ειδικές εκδόσεις των συναρτήσεων δέσμευσης μνήμης μπορούν να αποκαλύψουν σφάλματα χρόνου εκτέλεσης που αφορούν την χρήση της μνήμης.

2.3.17 Χρήσεις

Η C χρησιμοποιείται ευρέως για τον προγραμματισμό συστημάτων, για την υλοποίηση λειτουργικών συστημάτων και για εφαρμογές ενσωματωμένων συστημάτων λόγω των επιθυμητών χαρακτηριστικών που διαθέτει όπως η μεταφερσιμότητα του παραγόμενου κώδικα, η αποτελεσματικότητα αυτού, η δυνατότητα να έχει πρόσβαση σε συγκεκριμένες θέσεις της μνήμης και οι χαμηλές απαιτήσεις σε πόρους για την εκτέλεση. [114]Η C μπορεί επίσης να χρησιμοποιηθεί για την δημιουργία ιστοσελίδων με την χρήση των CGI σαν δίοδο για την πληροφορία μεταξύ της εφαρμογής ιστού, του server και του browser[115].

---

112 *Klemens, Ben (2013). 21st Century C. O'Reilly Media. ISBN 1-4493-2714-1.*

113 *"Man Page for lint (freebsd Section 1)". unix.com. 2001-05-24. Retrieved 2014-07-15.*

114 *Dr. Dobb's Sourcebook. U.S.A.: Miller Freeman, Inc. November–December 1995.*

115 *"Using C for CGI Programming". linuxjournal.com. 1 March 2005. Retrieved 4 January 2010.*

Μια συνέπεια του μεγάλου εύρους διαθεσιμότητας της C και της αποτελεσματικότητας της είναι ότι οι μεταγλωττιστές, οι βιβλιοθήκες και οι μεταφραστές άλλων γλωσσών προγραμματισμού είναι συχνά υλοποιημένοι σε C.Οι αρχικε υλοποιήσεις των Python, Perl 5 και PHP για παράδειγμα είναι όλες γραμμένες σε C.
Λόγω του λεπτού επιπέδου αφαίρεσης η C επειτρέπει την αποτελεσματική υλοποίηση αλγορίθμων και δομών δεδομένων τα οποίοι είναι πολύ χρήσιμα για προγράμματα τα οποία εκτελούν πολλούς υπολογισμούς. Για παράδειγμα η GNU Precision Arithmetic library, το GNU Scientific Library, Mathematica και το MATLAB είναι πλήρως ή μερικώς γραμμένα σε C.
Η C κάποιες φορές χρησιμοποιείται σαν ενδιάμεση γλώσσα από υλοποιήσεις άλλων γλωσσών. Αυτή η προσέγγιση μπορεί να επιλεγεί για λόγους μεταφερσιμότητας ή και ευκολίας. Χρησιμοποιώντας την C σαν ενδιάμεση γλώσσα δεν είναι απαραίτητο να αναπτύξουμε γεννήτριες κώδικα προσανατολισμένες στη μηχανή. Η C έχει κάποια χαρακτηριστικά όπως ο εντολές προεπεξεργαστή γραμμής αριθμών τα οποία υποστηρίζουν την μεταγλώττιση του παραγόμενου κώδικα. Ωστόσο μερικά από τα μειονέκτημα της C οδήγησαν στην ανάπτυξη άλλων γλωσσών με βάση τη C ειδικά σχεδιασμένων για χρήση ως ενδιάμεσες γλώσσες, όπως είναι η C--.
Η C επίσης χρησιμοποιείται ευρύτατα για την υλοποίηση εφαρμογών τελικού χρήστη ωσστόσο τα τελευταία χρόνια η ανάπτυξη τέτοιων εφαρμογών γίνεται με άλλες γλώσσες προγραμματισμού υψηλού επιπέδου.

2.3.18 Σχετιζόμενες γλώσσες

Η C άμεσα ή έμμεσα έχει επηρεάσει πολλές μεταγενέστερες γλώσσες όπως οι C#, D, Go, Java, JavaScript, Limbo, LPC, Perl, PHP, Python, και η C shell του Unix. Περισσότερο επηρέασε το συντακτικό τους, όλες οι γλώσσες που αναφέραμε συνδυάζουν τις δηλώσεις (περισσότερο ή λιγότερο φανερά) και το συντακτικό της C μαζί με τους τύπους δεδομένων τα μοντέλα δεδομένων και τις μεγάλης κλίμακας δομές των προγραμμάτων οι οποίες διαφέρουν από της C ορισμένες ριζικά.
Υπάρχουν Αρκετοί μεταφραστές C ή παρόμοιοι με της C μεταξύ των οποίων και οι Ch και CINT οι οποίοι μπορούν επιπλέον να χρησιμοποιηθούν για scripting.
Όταν οι αντικειμενοστραφείς γλώσσες έγιναν δημοφιλείς η C++ και η Objective-C αποτέλεσαν δύο επεκτάσεις της C οι οποίες παρείχαν αντικειμενοστραφείς δυνατότητες. Και οι δύο γλώσσες αρχικά υλοποιήθηκαν σαν source-to-source μεταγλωττιστές. Δηλαδή πρώτα ο κώδικας τους μεταφράστηκε σε C κώδικα και μετά μεταγλωττίστηκε με ένα μεταγλωττιστή C.
Η γλώσσα προγραμματισμού C++ επινοήθηκε από τον Bjarne Stroustrup σαν μια προσέγγιση που παρείχε αντικειμενοστραφή λειτουργικότητα με ένα συντακτικό παρόμοιο με της C[116]. Η C++ προσθέτει μεγαλύτερη δύναμη στη γραφή του κώδικα, στο εύρος των μεταβλητών και έχει και άλλα εργαλεία χρήσιμα στον αντικειμενοστραφή προγραμματισμό και επιτρέπει generic programming μέσω προτύπων. Σχεδόν ένα υπερσύνολο της C η C++ τώρα ποια υποστηρίζει το

116 *Stroustrup, Bjarne (1993). "A History of C++: 1979−1991" (PDF). Retrieved 9 June 2011.*

μεγαλύτερο κομμάτι της Cμε κάποιες εξαιρέσεις.
Η Objective-C αρχικά αποτελούσε ένα λεπτό επίπεδο πάνω από την C και παραμένει ένα στενό υπερσύνολο της C το οποίο επιτρέπει αντικειμενοστραφή προγραμματισμό χρησιμοποιώντας ένα υβρίδιο στατικού/δυναμικού παραδείγματος γραφής. Η Objective-C αντλεί το συντακτικό της τόσο από την C όσο και από την Smaltalk, συντακτικό το οποίο εμπεριέχει προεπεξεργασία, εκφράσεις, δηλώσεις συναρτήσεων, και κλήσεις συναρτήσεων κληρονομιά από την C ενώ το συντακτικό της για τα χαρακτηριστικά του αντικειμενοστραφούς προγραμματισμού το έχει πάρει από την Smaltalk.
Πέραν της C++ και της Objective-C ουσιασικά και οι γλώσσες Ch, Cilk και η Unified Parallel C αποτελούν ουσιαστικά υπερσύνολα της C.

Κεφάλαιο 3
Γλώσσες Υψηλού Επιπέδου

3.1 Java

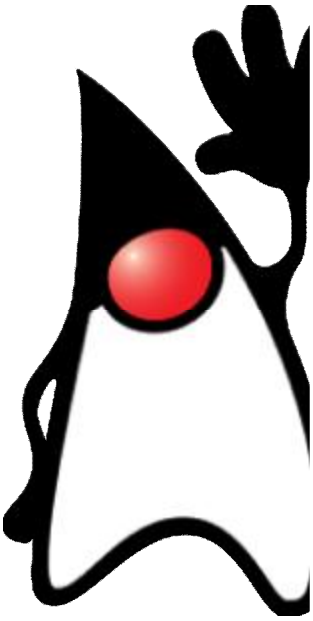
3.1.1 Εισαγωγή

Η Java είναι μια αντικειμενοστρεφής γλώσσα προγραμματισμού που σχεδιάστηκε από την εταιρεία πληροφορικής Sun Microsystems.

3.1.2 Ιστορία

Στις αρχές του 1991[117], η ***Sun*** αναζητούσε το κατάλληλο εργαλείο για να αποτελέσει την πλατφόρμα ανάπτυξης λογισμικού σε μικρο-συσκευές (έξυπνες οικιακές συσκευές έως πολύπλοκα συστήματα παραγωγής γραφικών). Τα εργαλεία της εποχής ήταν γλώσσες όπως η C++ και η C. Μετά από διάφορους πειραματισμούς προέκυψε το συμπέρασμα ότι οι υπάρχουσες γλώσσες δεν μπορούσαν να καλύψουν τις ανάγκες τους. Ο "πατέρας" της Java, James Gosling, που εργαζόταν εκείνη την εποχή για την Sun, έκανε ήδη πειραματισμούς πάνω στη C++ και είχε παρουσιάσει κατά καιρούς κάποιες πειραματικές γλώσσες (C++ ++,που μετέπειτα ονομαστικέ C#) ως πρότυπα για το νέο εργαλείο που αναζητούσαν στην ***Sun***. Τελικά μετά από λίγο καιρό κατέληξαν με μια πρόταση για το επιτελείο της εταιρίας, η οποία ήταν η γλωσσα ***Oak***. Το όνομά της το πήρε από το ομώνυμο δένδρο (βελανιδιά) το οποίο ο Gosling είχε έξω από το γραφείο του και έβλεπε κάθε μέρα.

3.1.3 Από την Oak στη Java



Εικόνα 7: Duke η μασκότ της Java

Η ***Oak*** ήταν μία γλώσσα που διατηρούσε μεγάλη συγγένεια με την C++. Παρόλα αυτά είχε πολύ πιο έντονο αντικειμενοστρεφή (***object oriented***) χαρακτήρα σε σχέση με την C++ και χαρακτηριζόταν για την απλότητα της. Σύντομα οι υπεύθυνοι ανάπτυξης της νέας γλώσσας ανακάλυψαν ότι το όνομα ***Oak*** ήταν ήδη κατοχυρωμένο οπότε κατά την διάρκεια μιας εκ των πολλών συναντήσεων σε κάποιο τοπικό καφέ αποφάσισαν να μετονομάσουν το νέο τους δημιούργημα σε Java που εκτός των άλλων ήταν το όνομα της αγαπημενης ποικιλιας καφέ για τους δημιουργούς της. Η επίσημη εμφάνιση της ***Java*** αλλα και του ***HotJava*** (πλοηγός με υποστήριξη ***Java***) στη βιομηχανία της πληροφορικής έγινε το Μάρτιο του 1995 όταν η ***Sun*** την ανακοίνωσε στο συνέδριο ***Sun World*** 1995. Ο πρώτος μεταγλωττιστής (***compiler***) της ήταν γραμμένος στη γλώσσα C από τον James Gosling. Το 1994, ο A.Van Hoff ξαναγράφει τον μεταγλωττιστή της γλώσσας σε ***Java***, ενώ το Δεκέμβριο του 1995 πρώτες οι IBM, Borland, Mitsubishi Electronics, Sybase και Symantec ανακοινώνουν σχέδια να χρησιμοποιήσουν[118] τη ***Java*** για την δημιουργία λογισμικού. Από εκεί και πέρα η Java ακολουθεί μία ανοδική πορεία και είναι πλέον μία από τις πιο δημοφιλείς γλώσσες στον χώρο της πληροφορικής. Στις 13 Νοεμβρίου του 2006 η Java έγινε πλέον μια γλώσσα ανοιχτού κώδικα (GPL) όσον αφορά το μεταγλωττιστή (javac) και το πακέτο ανάπτυξης (JDK, Java Development Kit).

---

117 *Gosling, James; Joy, Bill; Steele, Guy; Bracha, Gilad; Buckley, Alex (2014). The Java® Language Specification (PDF) (Java SE 8 ed.).*

118 *Gosling, James; Joy, Bill; Steele, Guy L., Jr.; Bracha, Gilad (2005). The Java Language Specification (3rd ed.). Addison-Wesley. ISBN 0-321-24678-0.*

3.1.4 Η εξαγορά από την Oracle και το μέλλον της Java

Στις 27 Απριλίου 2010 η εταιρία λογισμικού[119] Oracle Corporation ανακοίνωσε ότι μετά από πολύμηνες συζητήσεις ήρθε σε συμφωνία για την εξαγορά της Sun Microsystems και των τεχνολογιών (πνευματικά δικαιώματα/ πατέντες) που η δεύτερη είχε στην κατοχή της ή δημιουργήσει. Η συγκεκριμένη συμφωνία θεωρείται σημαντική για το μέλλον της Java και του γενικότερου οικοσυστήματος τεχνολογιών γύρω από αυτή μιας και ο έμμεσος έλεγχος της τεχνολογίας και η εξέλιξη της περνάει σε άλλα χέρια.

3.1.5 Τα χαρακτηριστικά της Java

Εικόνα 8: James Gossling ο δημιουργός της Java

Ένα από τα βασικά πλεονεκτήματα της Java έναντι των περισσότερων άλλων γλωσσών είναι η ανεξαρτησία του λειτουργικού συστήματος και πλατφόρμας. Τα προγράμματα που είναι γραμμένα σε *Java* τρέχουνε ακριβώς το ίδιο σε Windows, Linux, Unix και Macintosh (σύντομα θα τρέχουν και σε Playstation καθώς και σε άλλες κονσόλες παιχνιδιών) χωρίς να χρειαστεί να ξαναγίνει μεταγλώττιση (compiling) ή να αλλάξει ο πηγαίος κώδικας για κάθε διαφορετικό λειτουργικό σύστημα. Για να επιτευχθεί όμως αυτό χρειαζόταν κάποιος τρόπος έτσι ώστε τα προγράμματα γραμμένα σε Java να μπορούν να είναι «κατανοητά» από κάθε υπολογιστή ανεξάρτητα του είδους επεξεργαστή (Intel x86, IBM, Sun SPARC, Motorola) αλλά και λειτουργικού συστήματος (Windows, Unix, Linux, BSD, MacOS). Ο λόγος είναι ότι κάθε κεντρική μονάδα επεξεργασίας κατανοεί διαφορετικό κώδικα μηχανής. Ο συμβολικός κώδικας (***assembly***) που μεταφράζεται και εκτελείται σε Windows είναι διαφορετικός από αυτόν που μεταφράζεται και εκτελείται σε έναν υπολογιστή Macintosh. Η λύση δόθηκε με την ανάπτυξη της *Εικονικής Μηχανής* (***Virtual Machine*** ή VM ή ΕΜ στα ελληνικά).

3.1.5.1 Η εικονική μηχανή της Java

Αφού γραφεί κάποιο πρόγραμμα σε Java, στη συνέχεια[120] μεταγλωτίζετται μέσω του μεταγλωττιστή javac, ο οποίος παράγει έναν αριθμό από αρχεία .class (κώδικας byte ή bytecode). Ο κώδικας byte είναι η μορφή που παίρνει ο πηγαίος κώδικας της Java όταν μεταγλωττιστεί. Όταν πρόκειται να εκτελεστεί η εφαρμογή σε ένα μηχάνημα, το Java Virtual Machine που πρέπει να είναι εγκατεστημένο σε αυτό θα αναλάβει να διαβάσει τα αρχεία .class. Στη συνέχεια τα μεταφράζει σε γλώσσα μηχανής που να υποστηρίζεται από το λειτουργικό σύστημα και τον επεξεργαστή, έτσι

119 *Gosling, James; Joy, Bill; Steele, Guy L., Jr.; Bracha, Gilad (2005). The Java Language Specification (3rd ed.). Addison-Wesley. ISBN 0-321-24678-0.*

120 *Lindholm, Tim; Yellin, Frank (1999). The Java Virtual Machine Specification (2nd ed.). Addison-Wesley. ISBN 0-201-43294-3.*

ώστε να εκτελεστεί (αυτό συμβαίνει με την παραδοσιακή Εικονική Μηχανή (Virtual Machine). Πιο σύγχρονες εφαρμογές της εικονικής Μηχανής μπορούν και μεταγλωττίζουν εκ των προτέρων τμήματα bytecode απευθείας σε κώδικα μηχανής (εγγενή κώδικα ή native code) με αποτέλεσμα να βελτιώνεται η ταχύτητα). Χωρίς αυτό δε θα ήταν δυνατή η εκτέλεση λογισμικού γραμμένου σε Java. Η JVM είναι λογισμικό που εξαρτάται από την πλατφόρμα, δηλαδή για κάθε είδος λειτουργικού συστήματος και αρχιτεκτονικής επεξεργαστή υπάρχει διαφορετική έκδοση του. Έτσι υπάρχουν διαφορετικές JVM για Windows, Linux, Unix, Macintosh, κινητά τηλέφωνα, παιχνιδομηχανές κλπ.

Οτιδήποτε θέλει να κάνει ο προγραμματιστής (ή ο χρήστης) γίνεται μέσω της εικονικής μηχανής. Αυτό βοηθάει στο να υπάρχει μεγαλύτερη ασφάλεια στο σύστημα γιατί η εικονική μηχανή είναι υπεύθυνη για την επικοινωνία χρήστη - υπολογιστή. Ο προγραμματιστής δεν μπορεί να γράψει κώδικα ο οποίος θα έχει καταστροφικά αποτελέσματα για τον υπολογιστή γιατί η εικονική μηχανή θα τον ανιχνεύσει και δε θα επιτρέψει να εκτελεστεί. Από την άλλη μεριά ούτε ο χρήστης μπορεί να κατεβάσει «κακό» κώδικα από το δίκτυο και να τον εκτελέσει. Αυτό είναι ιδιαίτερα χρήσιμο για μεγάλα κατανεμημένα συστήματα όπου πολλοί χρήστες χρησιμοποιούν το ίδιο πρόγραμμα συγχρόνως.

Εικόνα 9: Το λειτουργικό σύστημα Android κάνει εκτεταμένη χρήση των

3.1.5.2 Ο συλλέκτης απορριμμάτων (Garbage Collector)

Ακόμα μία ιδέα που βρίσκεται πίσω από τη *Java*[121] είναι η ύπαρξη του συλλέκτη απορριμμάτων (*Garbage Collector*). Συλλογή απορριμμάτων είναι μία κοινή ονομασία που χρησιμοποιείται στον τομέα της πληροφορικής για να δηλώσει την ελευθέρωση τμημάτων μνήμης από δεδομένα που δε χρειάζονται και δε χρησιμοποιούνται άλλο. Αυτή η απελευθέρωση μνήμης στη Java είναι αυτόματη και γίνεται μέσω του συλλέκτη απορριμμάτων. Υπεύθυνη για αυτό είναι και πάλι η εικονική μηχανή η οποία μόλις «καταλάβει» ότι ο σωρός (heap) της μνήμης (στη Java η συντριπτική πλειοψηφία των αντικειμένων αποθηκεύονται στο σωρό σε αντίθεση με τη C++ όπου αποθηκεύονται κυρίως στη στοίβα) κοντεύει να γεμίσει ενεργοποιεί το συλλέκτη απορριμμάτων. Έτσι ο προγραμματιστής δε χρειάζεται να ανησυχεί για το πότε και αν θα ελευθερώσει ένα συγκεκριμένο τμήμα της μνήμης, ούτε και για σφάλματα δεικτών. Αυτό είναι ιδιαίτερα σημαντικό γιατί είναι κοινά τα σφάλματα προγραμμάτων που οφείλονται σε λανθασμένο χειρισμό της μνήμης.

3.1.5.3 Επιδόσεις

121 *Gosling, James; Joy, Bill; Steele, Guy; Bracha, Gilad; Buckley, Alex (2014). The Java® Language Specification (PDF) (Java SE 8 ed.).*

Παρόλο που η εικονική μηχανή προσφέρει[122] όλα αυτά (και όχι μόνο) τα πλεονεκτήματα, η *Java* αρχικά ήταν πιο αργή σε σχέση με άλλες προγραμματιστικές γλώσσες υψηλού επιπέδου (high-level) όπως η C και η C++. Εμπειρικές μετρήσεις στο παρελθόν είχαν δείξει ότι η C++ μπορούσε να είναι αρκετές φορές γρηγορότερη από την Java. Ωστόσο γίνονται προσπάθειες από τη Sun για τη βελτιστοποίηση της εικονικής μηχανής, ενώ υπάρχουν και άλλες υλοποιήσεις της εικονικής μηχανής από διάφορες εταιρίες (όπως της IBM), οι οποίες μπορεί σε κάποια σημεία να προσφέρουν καλύτερα και σε κάποια άλλα χειρότερα αποτελέσματα. Επιπλέον με την καθιέρωση των μεταγλωττιστών JIT (Just In Time), οι οποίοι μετατρέπουν τον κώδικα byte απευθείας σε γλώσσα μηχανής, η διαφορά ταχύτητας από τη C++ έχει μικρύνει κατά πολύ.
Οι τελευταίες εκδόσεις του javac με τη χρήση της τεχνολογίας Hot Spot έχουν καταφέρει αξιόλογες επιδόσεις που πλησιάζουν ή και ξεπερνούν σε μερικές περιπτώσεις τον εγγενή κώδικα.

3.1.6 Εργαλεία ανάπτυξης

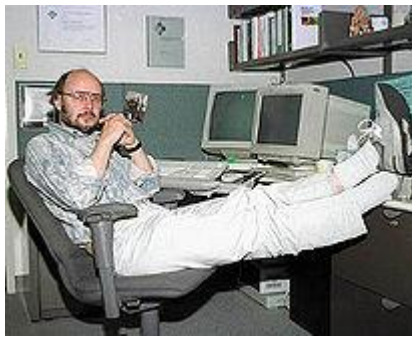
Όλα τα εργαλεία που χρειάζεται κάποιος για να γράψει Java προγράμματα[123] έρχονται δωρεάν, από το περιβάλλον ανάπτυξης μέχρι εργαλεία ***build*** όπως το Apache Ant και βιβλιοθήκες, ενώ υπάρχουν πολλές διαφορετικές υλοποιήσεις της *Εικονικής Μηχανής* και του *μεταγλωττιστή* (πχ the GNU Compiler for Java) της *Java*.
Πολλά εργαλεία και τεχνολογίες σε Java μπορούν να βρεθούν στο Apache Software Foundation αλλά και στο Jakarta Project.

3.1.7 Ολοκληρωμένο περιβάλλον ανάπτυξης (IDE)

Για να να γράψει κάποιος κώδικα Java δε χρειάζεται [124]τίποτα άλλο παρά έναν επεξεργαστή κειμένου, όπως το Σημειωματάριο (Notepad) των Windows ή ο vi (γνωστός στο χώρο του Unix). Παρ'όλ'αυτά, ένα ολοκληρωμένο περιβάλλον ανάπτυξης (*IDE*) βοηθάει πολύ, ιδιαίτερα στον εντοπισμό σφαλμάτων (debugging). Υπάρχουν αρκετά διαθέσιμα, ενώ πολλά από αυτά έρχονται δωρεάν.

3.2 Γλώσσα C++

3.2.1 Εισαγωγή



Εικόνα 10: Bjarna Stroustrup
Ο δημιουργός της C++

122 *Lindholm, Tim; Yellin, Frank (1999). The Java Virtual Ma*
Addison-Wesley. ISBN 0-201-43294-3.

123 *Gosling, James; Joy, Bill; Steele, Guy L., Jr.; Bracha, Gila*
Specification (3rd ed.). Addison-Wesley. ISBN 0-321-24678-0.

124 *Lindholm, Tim; Yellin, Frank (1999). The Java Virtual Ma*
Addison-Wesley. ISBN 0-201-43294-3.

Η C++ είναι μια γλώσσα προγραμματισμού γενικού σκοπού. Διαθέτει επιτακτικότητα, αντικειμενοστρέφεια και γενικά προγραμματιστικά χαρακτηριστικά ενώ παράλληλα παρέχει και τις απαραίτητες δομές για διαχείριση μνήμης χαμηλού επιπέδου.
Σχεδιάστηκε με κύριο σκοπό τον προγραμματισμό συστημάτων, ενσωματωμένων συστημάτων, συστημάτων περιορισμένων πόρων και μεγάλων συστημάτων, με τις επιδόσεις, την αποδοτικότητα και την ευελιξία να αποτελούν τα κύρια χαρακτηριστικά της[125]. Η C++ αποδείχτηκε χρήσιμη και σε μια πλειάδα άλλες χρήσεις πέραν των αρχικών με βασικότερες την υποδομή του λογισμικού και τις εφαρμογές περιορισμένων πόρων, μεταξύ των οποίων εφαρμογές επιφάνειας εργασίας, servers (για εμπόριο, αναζήτηση ιστού και SQL servers) καθώς και εφαρμογές υψηλών επιδόσεων (όπως είναι τα switches της τηλεφωνίας ή τους ανιχνευτές διαστήματος)[126]. Η C++ είναι μια μεταγλωττιζόμενη γλώσσα με υλοποιήσεις της διαθέσιμες σε πάρα πολλές πλατφόρμες που παρέχονται από πολλούς οργανισμούς όπως οι FSF, LLVM, Microsoft, Intel και IBM.
Η C++ έχει προ τυποποιηθεί από τον International Organization for Standarization (ISO) με την τελευταία έκδοση να εκδίδεται τον Δεκέμβριο του 2014 σαν ISO/IEC 14882:2014 (ανεπίσημα γνωστή και ως C++14)[127]. Η C++ αρχικά προτυποποιήθηκε το 1998 σαν ISO/IEC 14882:1998 έκδοση η οποία αντικαταστάθηκε από την C++03, ISO/IEC 14882:2003 πρότυπο. Το τρέχον C++14 πρότυπο αποτελεί υπερσύνολο του C++11 ενσωματώνοντας νέα χαρακτηριστικά και με μια ευρύτερη στάνταρ βιβλιοθήκη. Πριν την αρχική προτυποποίηση του 1998 η C++ αναπτύχθηκε από τον Bjarne Stroustrup στα Bell Labs από το 1979 σαν μια επέκταση της C η οποία επίσης παρείχε χαρακτηριστικά υψηλού επιπέδου για προγραμματισμό οργανισμών.
Πολλές άλλες γλώσσες προγραμματισμού επηρεάστηκαν από την C++ μεταξύ των οποίων η C#, η Java, και οι νεότερες εκδόσεις της C.

3.2.2 Ιστορία

Ο Bjarne Stroustrup, ένας Δανός επιστήμονας υπολογιστών ξεκίνησε να εργάζεται πάνω στην C με κλάσεις, [128]τον πρόγονο της C++ το 1979. Το κίνητρο για μια νέα γλώσσα προγραμματισμού προήλθε για τον Stroustrup από την εμπειρία του στον προγραμματισμό κατά την διατριβή του διδακτορικού του. Ο Stroustrup είδε ότι η Simula διέθετε χαρακτηριστικά πολύ χρήσιμα για την ανάπτυξη λογισμικού μεγάλου μεγέθους με την γλώσσα όμως να είναι ιδιαιτέρως αργή και άρα ακατάλληλη για πρακτική χρήση την στιγμή που η BCPL ενώ ήταν πολύ γρήγορη ήταν ταυτόχρονα ιδιαιτέρως χαμηλού επιπέδου για την ανάπτυξη λογισμικού μεγάλης κλίμακας. Όταν ο Stroustrup ξεκίνησε να εργάζεται στην AT&T Bell Labs αντιμετώπιζε το πρόβλημα της ανάλυσης του πυρήνα του UNIX σε σχέση με τα κατανεμημένα υπολογιστικά συστήματα. Χρησιμοποιώντας την εμπειρία

125 *Stroustrup, B. (6 May 2014). "Lecture:The essence of C++. University of Edinburgh.". Retrieved 12 June 2015.*

126 *Stroustrup, Bjarne (17 February 2014). "C++ Applications". stroustrup.com. Retrieved 5 May 2014.*

127 *"ISO/IEC 14882:2011". International Organization for Standardization.*

128 *Stroustrup, Bjarne (7 March 2010). "Bjarne Stroustrup's FAQ: When was C++ invented?". stroustrup.com. Retrieved 16 September 2010.*

του από το διδακτορικό του προσπάθησε να εμπλουτίσει την C με χαρακτηριστικά αντίστοιχα της Simula[129]. Η C επελέγη διότι ήταν μια γλώσσα γενικού σκοπού, γρήγορη, μεταφέρσιμη και ευρέως χρησιμοποιούμενη. Όπως η C και η Simula και άλλες γλώσσες επηρέασαν την C++ όπως οι ALGOL 68, η Ada, η CLU και η ML.

Αρχικά η C με κλάσεις του Stroustrup πρόσθεσε χαρακτηριστικά στον μεταγλωττιστή της C μεταξύ των οποίων ήταν οι κλάσεις, οι συμπληρωματικές κλάσεις, η ισχυρή γραφή, οι αντιστοιχίες και τα προεπιλεγμένα επιχειρήματα[130].

Το 1983 η C με κλάσεις μετονομάζεται σε C++ (με το ++ να είναι ο τελεστής αύξησης στην C) προσθέτοντας νέα χαρακτηριστικά όπως οι εικονικές συναρτήσεις, η υπερφόρτωση τελεστών και συναρτήσεων, οι αναφορές, οι σταθερές, η απελευθέρωση και η κατάληψη μνήμης, τα σχόλια μονής γραμμής τύπου BCPL τα οποία αρχίζουν με // καθώς και με ένα μεταγλωττιστή της C++ τον Cfront.

Το 1985 η πρότη έκδοση του The C++ Programming Language κυκλοφόρησε και αποτέλεσε τον ορισμό αναφοράς για την γλώσσα μιας και δεν υπήρχε ακόμη κάποιο επίσημο πρότυπο. [131]Η πρώτη εμπορική υλοποίηση της C++ κυκλοφόρησε τον Οκτώβριο του ίδιου έτους[132].

Το 1989 η C++2 κυκλοφόρησε ακολουθούμενη από μια ενημερωμένη δεύτερη έκδοση του The C++ Programming Language το 1991. [133]Τα νέα χαρακτηριστικά της δεύτερη έκδοσης περιείχαν την πολλαπλή κληρονομικότητα, τις abstract κλάσεις, τα static, const και protected μέλη συναρτήσεων. Το 1990 εκδίδεται το Annotated C++ Reference manual. Αυτή η εργασία έγινε η βάση του μελλοντικού προτύπου. Νεώτερες προσθήκες χαρακτηριστικών αποτελούν τα templates, τις εξαιρέσεις, τα namespaces, new casts και έναν τύπο boolean.

Μετά την 2.0 αναβάθμιση η C++ εξελίχθηκε σχετικά αργά μέχρι το 20112 οπότε και το C++11 πρότυπο κυκλοφόρησε προσθέτοντας πολλά νέα χαρακτηριστικά επεκτείνοντας περαιτέρω την στάνταρ βιβλιοθήκη και παρέχοντας περισσότερες δομές της C++ στους προγραμματιστές. Μετά την μικρή αναβάθμιση στο πρότυπο C++14 το 2014 σχεδιάζονται αρκετές αλλαγές για το 2017.

3.2.3 Ετυμολογία

Σύμφωνα με τον Stroustrup “το όνομα υποδηλώνει την επαναστατική φύση των αλλαγών από την C”[134]. Αυτή η ονομασία αποδίδεται στον Rick Mascriti και πρωτοχρησιμοποιήθηκε τον Δεκέμβριο του 1983[135]. Το όνομα εκπηγάζει από τον τελεστή αύξησης ++ της C (ο οποίος αυξάνει την τιμή μιας μεταβλητής) και από μια κοινά χρησιμοποιούμενη μετατροπή ονόματος με την χρήση του +

---

129 *Stroustrup, Bjarne. "Evolving a language in and for the real world: C++ 1991-2006" (PDF).*

130 *Stroustrup, Bjarne. "A History of C ++ : 1979– 1991" (PDF)*

131 *Stroustrup, Bjarne. "The C++ Programming Language" (First ed.). Retrieved 16 September 2010.*

132 *Stroustrup, Bjarne (7 March 2010). "Bjarne Stroustrup's FAQ: When was C++ invented?". stroustrup.com. Retrieved 16 September 2010.*

133 *Stroustrup, Bjarne. "The C++ Programming Language" (Second ed.). Retrieved 16 September 2010.*

134 *"Bjarne Stroustrup's FAQ – Where did the name "C++" come from?". Retrieved 16 January 2008.*

135 *Stroustrup, Bjarne. "A History of C ++ : 1979– 1991" (PDF).*

για την υποδήλωση ενός αναβαθμισμένου προγράμματος υπολογιστή.
Κατά την διάρκεια ανάπτυξης της γλώσσας το όνομα το οποίο χρησιμοποιείτο ήταν το new C ή το C with classes[136][137] εως ότου κατέληξαν στο C++.

3.2.4 Φιλοσοφία

Σε όλη τη διάρκεια της ύπαρξης της η C++ η ανάπτυξη της και η εξέλιξη της διέπονται ατύπως από το σύνολο κανόνων που ακολουθούν[138]:

- Πρέπει να οδηγείται από υπαρκτά προβλήματα και τα χαρακτηριστικά της θα πρέπει άμεσα να είναι χρήσιμα σε πραγματικά προγράμματα.
- Κάθε χαρακτηριστικό της θα πρέπει να είναι υλοποιήσιμο (με κάποιο προφανή τρόπο).
- Οι προγραμματιστές θα πρέπει να αφήνονται ελεύθεροι να επιλέξουν οι ίδιοι το προγραμματιστικό στυλ που επιθυμούν και το στυλ αυτό θα πρέπει να είναι απολύτως υποστηριζόμενο από την C++.
- Το να επιτρέψουμε ένα χρήσιμο χαρακτηριστικό είναι σημαντικότερο από το να αποτρέψουμε κάθε είδους κακή χρήση της γλώσσας.
- Η γλώσσα θα πρέπει να παρέχει υποδομές για την οργάνωση των προγραμμάτων σε καλά χωρισμένα μέρη καθώς και υποδομές για τον συνδυασμό ξεχωριστά αναπτυγμένων τμημάτων κώδικα.
- Δεν θα επιτρέπονται σιωπηρές παραβιάσεις του συστήματος τύπων (θα επιτρέπονται όμως σαφής, αυτές δηλαδή που σαφώς θα αιτούνται από τον προγραμματιστή).
- Οι τύποι που δημιουργούνται από τον χρήστη θα πρέπει να έχουν την ίδια υποστήριξη και απόδοση με τους ενσωματωμένους τύπους.
- Οι μη χρησιμοποιούμενοι τύποι δεν θα πρέπει να επηρεάζουν αρνητικά τα παραγόμενα εκτελέσιμα (μειώνοντας ας πούμε την απόδοση).
- Δεν θα πρέπει να υπάρχει άλλη γλώσσα κάτω από την C++ (πέραν της assembly).
- Η γλώσσα θα πρέπει να μπορεί να συνεργαστεί με τις ήδη υπάρχουσες γλώσσες προγραμματισμού αντί να δημιουργεί το δικό της ξεχωριστό και ασύμβατο προγραμματιστικό περιβάλλον.
- Αν οι προθέσεις του προγραμματιστή είναι άγνωστες να του δίνεται η δυνατότητα να τις προσδιορίσει με χειροκίνητο έλεγχο.

3.2.5 Προτυποποίηση

Η C++ προτυποποιήθηκε από μια ομάδα εργασίας ISO γνωστή και ως JTC1/SC22/WG21. Έως και σήμερα έχει εκδώσει τέσσερις αναθεωρήσεις του προτύπου τους γλώσσας και αυτήν την στιγμή

136 *Stroustrup, Bjarne. "A History of C ++ : 1979– 1991" (PDF).*

137 *"C For C++ Programmers". Northeastern University. Retrieved 7 September 2015.*

138 *Stroustrup, Bjarne. "Evolving a language in and for the real world: C++ 1991-2006" (PDF).*

εργάζεται στην νέα αναθεώρηση την C++17.
Το 1998 η ομάδα εργασίας ISO προτυποποίησε την C++ για πρώτη φόρα σαν ISO/IEC 14882:1998 το οποίο είναι ατύπως γνωστό και ως C++98. Το 2003 εξέδωσε μια νέα έκδοση του προτύπου της C++ γνστή ως ISO/IEC 14882:2003 η οποία έφτιαχνε προβλήματα τα οποία αναγνωρίστηκαν στην C++98.
Η νέα μεγάλη αναθεώρηση του προτύπου στην αναφερόμαστε ατύπως ως C++0x δόθηκε το 2011[139] και περιείχε πολλές προσθήκες τόσο στον πυρήνα της γλώσσας όσο και στην στάνταρ βιβλιοθήκη[140].
Το 2014 δόθηκε το C++14 πρότυπο (γνωστό επίσης και ως C++1y) σαν μια μικρή επέκταση του C++11 η οποία είχε κυρίως διορθώσεις προβλημάτων και μικρές βελτιώσεις[141]. Οι διαδικασίες του Draft International Standard ballot διεκπαιρεόθηκαν τον Αύγουστο του 2014[142].
Μετά την C++14 μια μεγάλη αναθεώρηση, ατύπως γνωστή και ως C++17 ή C++1z σχεδιάζεται για το 2017[143] η οποία θα είναι σχεδόν πλήρης από άποψη χαρακτηριστικών[144].

3.2.6 Γλώσσα

Η γλώσσα C++ έχει δύο κύρια συστατικά: την άμεση χαρτογράφηση των χαρακτηριστικών του υλικού η οποία παρέχεται κυρίως από το υποσύνολο της C και οι μηδενικές πέραν αυτών των χαρακτηριστικών αφαιρέσεις οι οποίες βασίζονται στην χαρτογράφηση αυτή. Ο Stroustrup περιγράφει την C++ σαν μια ελαφράς αφαίρεσης γλώσσα προγραμματισμού η οποία σχεδιάστηκε για την κατασκευή και χρήση αποτελεσματικών και κομψών αφαιρέσεων. Η παροχή πρόσβασης στο υλικό και αφαιρετικότητα είναι βάση της C++[145].Το ότι το κάνει αποτελεσματικά είναι το χαρακτηριστικό που την διαχωρίζει από τις άλλες γλώσσες[146].
Η C++ κληρονομεί το περισσότερο συντακτικό της από το συντακτικό της C. Ακολουθεί η πρόταση του Stroustrup για το πρόγραμμα Hello World με χρήση της στάνταρ βιβλιοθήκης της C++ για την δημιουργία του ρεύματος που είναι απαραίτητο για το γράψιμο του μηνύματος στη στάνταρ έξοδο[147][148]:

---

139 *"We have an international standard: C++0x is unanimously approved". Sutter's Mill.*

140 *"ISO/IEC 14882:2011". International Organization for Standardization.*

141 *"The Future of C++".*

142 *"We have C++14! : Standard C++".*

143 *"The Future of C++".*

144 Recent milestones: C++17 nearly feature-complete, second round of TSes now under development

145 *Stroustrup, B. (6 May 2014). "Lecture:The essence of C++. University of Edinburgh.". Retrieved 12 June 2015.*

146 *B. Stroustrup (interviewed by Sergio De Simone) (30 April 2015). "Stroustrup: Thoughts on C++17 - An Interview". Retrieved 8 July 2015.*

147 *B. Stroustrup (interviewed by Sergio De Simone) (30 April 2015). "Stroustrup: Thoughts on C++17 - An Interview". Retrieved 8 July 2015.*

148 *Stroustrup, Bjarne. "Open issues for The C++ Programming Language (3rd Edition)". This code is copied*

```
#include <iostream>

int main()
     {
     std::cout << "Hello, world!\n";
     }
```

Στο εσωτερικό συναρτήσεων μη void τύπου, η αποτυχία επιστροφής τιμής πριν ο έλεγχος να φτάσει στο τέλος της συνάρτησης, οδηγεί σε μη προσδιορισμένη συμπεριφορά (οι μεταγλωττιστές παρέχουν τα μέσα για την διεξαγωγή ελέγχου αυτών των περιπτώσεων). [149]Η μόνη εξαίρεση στον κανόνα αυτό είναι η συνάρτηση main η οποία τυπικά επιστρέφει την τιμή 0[150].

3.2.7 Αποθήκευση αντικειμένων

Όπως η C η C++ υποστηρίζει τέσσερις τύπους διαχείρισης μνήμης. Αντικείμενα προς αποθήκευση διάρκειας Static, αντικείμενα προς αποθήκευση διάρκειας thread, αντικείμενα προς αποθήκευση με αυτόματη διάρκεια και αντικείμενα προς αποθήκευση δυναμική διάρκεια[151].

3.2.7.1 Αντικείμενα με διάρκεια αποθήκευσης static

Τα αντικείμενα αυτού του τύπου δημιουργούνται πριν την είσοδο στη main και καταστρέφονται με σειρά αντίστροφη από τη σειρά δημιουργίας μετά την έξοδο από την main. Η ακριβής σειρά δημιουργίας δεν ορίζεται από το πρότυπο ώστε να υπάρχει κάποια σχετική ελευθερία στους οργανισμούς που υλοποιούν τους μεταγλωττιστές στο τρόπο οργάνωσης της έκδοσης τους. Περισσότερο τυπικά αντικείμενα του τύπου this έχουν μήκος ζωής το οποίο θα διαρκέσει για όλη την διάρκεια του προγράμματος[152].
Τα static διάρκειας αντικείμενα αρχικοποιούνται σε δύο φάσεις. Πρώτα εκτελείται η static αρχικοποίηση και μόνο αφού ολοκληρωθεί πλήρως εκκινείται η δυναμική αρχικοποίηση. Στην static αρχικοποίηση όλα τα αντικείμενα αρχικοποιούνται με την τιμή μηδέν, ακολούθως όλα τα αντικείμενα τα οποία έχουν μια σταθερή φάση αρχικοποίησης αρχικοποιούνται με την σταθερή έκφραση. Αν και δεν προσδιορίζεται στο πρότυπο η static φάση αρχικοποίησης μπορεί να ολοκληρώνεται σε χρόνο μεταγλώττισης και να αποθηκεύεται στο τμήμα δεδομένων του εκτελέσιμου. Η δυναμική αρχικοποίηση συμπεριλαμβάνει όλη την αρχικοποίηση αντικειμένων η οποία γίνεται μέσω ενός κατασκευαστή ή της κλήσης μιας συνάρτησης. Η σειρά της δυναμικής αρχικοποίησης ορίζεται σαν η σειρά δήλωσης μέσα στην μονάδα μεταγλώττισης. Δεν παρέχονται εγγυήσεις για την σειρά αρχικοποίησης μεταξύ μονάδων μεταγλώττισης.

3.2.7.2 Αντικείμενα με διάρκεια αποθήκευσης thread

---

directly from Bjarne Stroustrup's errata page (p. 633). He addresses the use of '\n' *rather than* std::endl. *Also see Can I write "void main()"? for an explanation of the implicit* return 0; *in the* main *function. This implicit return is not available in other functions.*

149 ISO/IEC (2003). *ISO/IEC 14882:2003(E): Programming Languages – C++ §6.6.3 The return statement [stmt.return]* para. 2

150 ISO/IEC (2003). *ISO/IEC 14882:2003(E): Programming Languages – C++ §3.6.1 Main function [basic.start.main]* para. 5

151 ISO/IEC. *Programming Languages – C++11 Draft (n3797) §3.7 Storage duration [basic.stc]*

152 ISO/IEC. *Programming Languages – C++11 Draft (n3797) §3.7.1 Static Storage duration [basic.stc.static]*

Οι μεταβλητές αυτού του τύπου μοιάζουν πολύ με τα αντικείμενα με διάρκεια Static. Η κύρια διαφορά τους είναι ο χρόνος δημιουργίας τους που είναι ακριβώς πριν από την δημιουργία του αντίστοιχου thread και η καταστροφή τους γίνεται μετά την ένταξη του εκάστοτε thread[153].

3.2.7.3 Αντικείμενα με διάρκεια αποθήκευσης automatic

Πρόκειται για τις πλέον κοινές μεταβλητές[154] στην C++, είναι τοπικές μεταβλητές στο εσωτερικό μιας συνάρτησης ή ενός τμήματος κώδικα. Το κοινό χαρακτηριστικό των αυτόματων μεταβλητών είναι ότι έχουν διάρκεια ζωής η οποία περιορίζεται στο εύρος της μεταβλητής. Δημιουργούνται και πιθανώς αρχικοποιούνται στο σημείο δήλωσης και καταστρέφονται με την αντίθετη σειρά από αυτή που δημιουργήθηκαν όταν το εύρος τους τελειώνει.
Οι τοπικές μεταβλητές δημιουργούνται τη στιγμή που ο έλεγχος προγράμματος περνάει από το σημείο δήλωσης τους. Οι τοπικές μεταβλητές καταστρέφονται ότα το τοπικό τμήμα κώδικα ή η συνάρτηση, μέσα στα οποία δηλώνονται, κλείνει. Οι καταστροφείς της C++ για τις τοπικές μεταβλητές καλούνται στο τέλος του εύρους ζωής του αντικειμένου επιτρέποντας μια πειθαρχία για τον τερματισμό των αυτόματων πόρων RAII η οποία χρησιμοποιείται ευρύτατα στην C++.
Οι μεταβλητές μέλη δημιουργούνται κατά την δημιουργία του αντικειμένου γονέα. Οι πίνακες μέλη αρχικοποιούνται από το 0 έως το τελευταίο στοιχείο του πίνακα σε σειρά. Οι μεταβλητές μέλη καταστρέφονται όταν το αντικείμενο γονέας καταστρέφεται με την αντίστροφη σειρά από την σειρά της δημιουργίας τους. Για παράδειγμα αν ο γονιός είναι ένα αυτόματα αντικείμενο τότε θα καταστραφεί όταν βρεθεί εκτός εύρους ζωής οπότε και ενεργοποιείται η καταστροφή όλων των μελών του.
Οι προσωρινές μεταβλητές δημιουργούνται ως αποτέλεσμα αξιολόγησης εκφράσεων και καταστρέφονται όταν η δήλωση έχει πλήρως αξιολογηθεί.

3.2.7.4 Αντικείμενα με διάρκεια αποθήκευσης dynamic

Τα αντικείμενα αυτά έχουν δυναμικά προσδιοριζόμενο όριο ζωής, δημιουργούνται με την κλήση της συνάρτησης new και καταστρέφονται με μια μοναδική κλήση της delete[155].

3.2.8 Πρότυπα

Τα πρότυπα της C++ καθιστούν δυνατό τον γενικό προγραμματισμό. Η C++ υποστηρίζει πρότυπα συναρτήσεων, κλάσεων, ψευδώνυμων και μεταβλητών. Τα πρότυπα μπορεί να παραμετροποιούνται με τη χρήση τύπων, σταθερών χρόνου μεταγλώττισης, και μέσω άλλων προτύπων. Για την αρχικοποίηση ενός προτύπου οι μεταγλωττιστές υποκαθιστούν συγκεκριμένα ορίσματα για ώστε οι παράμετροι του προτύπου να παράξουν ένα στιγμιότυπο μιας απτής συνάρτησης ή κλάσης. Μερικές υποκαταστάσεις δεν είναι δυνατές. Αυτές εξουδετερώνονται από μια υπερφορτωμένη πολιτική ανάλυσης η οποία περιγράφεται από την φράση “η αποτυχία της υποκατάστασης δεν είναι λάθος ” (Substitution failure is not an erro – SFINAE). Τα πρότυπα είναι δυνατά εργαλεία τα οποία μπορούν να χρησιμοποιηθούν για γενικό προγραμματισμό, προγραμματισμό προτύπων και για την βελτίωση κώδικα, όμως αυτή η δύναμη συνεπάγεται κόστος. Η χρήση προτύπων μπορεί να οδηγήσει στην αύξηση του μεγέθους του κώδικα γιατί η αρχικοποίηση κάθε προτύπου παράγει ένα

153 ISO/IEC. *Programming Languages – C++11 Draft (n3797) §3.7.2 Thread Storage duration [basic.stc.thread]*

154 ISO/IEC. *Programming Languages – C++11 Draft (n3797) §3.7.3 Automatic Storage duration [basic.stc.auto]*

155 ISO/IEC. *Programming Languages – C++11 Draft (n3797) §3.7.4 Dynamic Storage duration basic.stc.dynamic*

αντίγραφο του κώδικα του προτύπου. Ένα για κάθε πρότυπο ορισμάτων, ωστόσο αυτή είναι η ίδια ή και μικρότερη σε μέγεθος ποσότητα κώδικα που θα είχε παραχθεί αν ο παραγόμενος κώδικας γραφόταν με το χέρι[156]. Αυτό έρχεται σε αντίθεση με τις γενικότητες χρόνου εκτέλεσης που βλέπουμε σε άλλες γλώσσες στις οποίες κατά την διάρκεια του χρόνου εκτέλεσης ο τύπος μπορεί να διαγραφεί ενώ διατηρείται το σώμα του προτύπου.
Τα πρότυπα διαφέρουν από τις μακροεντολές. Ενώ και για τα δύο τα χαρακτηριστικά της γλώσσας χρόνου μεταγλώττισης καθιστούν δυνατή την υπό όρους μεταγλώττιση, τα πρότυπα δεν περιορίζονται στην λεκτική υποκατάσταση. Τα πρότυπα γνωρίζουν τη σημασιολογία του συστήματος τύπων της γλώσσας τους όπως επίσης και του ορισμούς τύπων του χρόνου μεταγλώττισης και μπορούν να εκτελέσουν λειτουργίες υψηλού επιπέδου όπως προγραμματιστικό έλεγχο ροής βασισμένο στην αξιολόγηση παραμέτρων αυστηρών τύπων. Οι μακροεντολές είναι ικανές για κατά συνθήκη έλεγχο της μεταγλώττισης βασισμένο σε προκαθορισμένα κριτήρια αλλά δεν μπορούν να αρχικοποιήσουν νέους τύπους, να είναι αναδρομικές ή να εκτελέσουν αξιολόγηση τύπων και σαν αποτέλεσμα είναι περιορισμένες στην προ της μεταγλώττισης υποκατάστασης κειμένου και στον αποκλεισμό ή όχι κειμένου. Με άλλα λόγια οι μακροεντολές μπορούν να ελέγξουν την ροή της μεταγλώττισης βασισμένες σε προκαθορισμένα σύμβολα αλλά δε μπορούν, σε αντίθεση με τα πρότυπα, ανεξάρτητα να αρχικοποιήσουν νέα σύμβολα. Τα πρότυπα είναι ένα αργαλείο στατικού πολυμορφισμού και γενικού προγραμματισμού.
Επιπλέον τα πρότυπα είναι ένας μηχανισμός χρόνου μεταγλώττισης στην C++ ο οποίος είναι ολοκληρωμένος κατά Turing με την έννοια ότι κάθε υπολογισμός που μπορεί να εκφραστεί από ένα πρόγραμμα υπολογιστή μπορεί να υπολογιστεί σε κάποια μορφή από ένα μεταπρόγραμμα προτύπου πριν τον χρόνο εκτέλεσης.
Συνοψίζοντας, πρότυπο είναι μια παραμετροποιημένη συνάρτηση ή κλάση χρόνου μεταγλώττισης η οποία είναι γραμμένη δίχως αντίληψη των ορισμάτων που χρησιμοποιούνται για την αρχικοποίηση του. Μετά την αρχικοποίηση ο παραγόμενος κώδικας είναι ίδιος με τον κώδικα που έχει γραφεί για τα περασμένα ορίσματα. Με αυτόν τον τρόπο τα πρότυπα παρέχουν ένα τρόπο αποσύνδεσης γενικές και ευρέως εφαρμόσιμες πτυχές των συναρτήσεων και των κλάσεων (που έχουν κωδικοποιηθεί εντός του προτύπου) από συγκεκριμένες πτυχές (κωδικοποιημένες στις παραμέτρους του προτύπου) δίχως να θυσιάζουν την απόδοση στο βωμό της αφαίρεσης.

3.2.9 Αντικείμενα

Η C++ εισαγάγει τον αντικειμενοστραφή προγραμματισμό στην C. Παρέχει κλάσεις οι οποίες παρέχουν τα τέσσερα χαρακτηριστικά που βλέπουμε στις αντικειμενοστραφείς γλώσσες προγραμματισμού: την αφαίρεση, την ενθυλάκωση, την κληρονομικότητα και τον πολυμορφισμό. Ένα χαρακτηριστικό που διαφοροποιεί τις κλάσεις της C++ από τις κλάσεις άλλων γλωσσών προγραμματισμού είναι η υποστήριξη ντετερμινιστικών καταστροφέων οι οποίοι με την σειρά τους παρέχουν υποστήριξη για την έννοια του Resource Acquisition is Initialization (RAII).

3.2.10 Ενθυλάκωση

156 *"Nobody Understands C++: Part 5: Template Code Bloat". http://blog.emptycrate.com/: EmptyCrate Software. Travel. Stuff. 6 May 2008. Retrieved 8 March 2010.* On occasion you will read or hear someone talking about C++ templates causing code bloat. I was thinking about it the other day and thought to myself, "self, if the code does exactly the same thing then the compiled code cannot really be any bigger, can it?" [...] And what about compiled code size? Each were compiled with the command g++ <filename>.cpp -O3. Non-template version: 8140 bytes, template version: 8028 bytes!

Η ενθυλάκωση είναι η απόκρυψη πληροφορίας με σκοπό την διασφάλιση ότι οι δομές δεδομένων και οι τελεστές χρησιμοποιούνται με τον τρόπο που πρέπει και για να γίνει περισσότερο προφανές το μοντέλο χρήσης τους στον προγραμματιστή. Η C++ παρέχει την δυνατότητα του ορισμού κλάσεων και συναρτήσεων σαν τον πρωτεύοντα μηχανισμό ενθυλάκωσης. Μέσα σε μια κλάση, τα μέλη μπορούν να δηλωθούν ως public, ως protected είτε ως private ώστε να επιβάλλουν ρητώς την ενθυλάκωση. Ένα μέλος κλάσης ορισμένο ως public είναι προσβάσιμο από οποιαδήποτε συνάρτηση. Ένα μέλος κλάσης ορισμένο ως private είναι προσβάσιμο μόνο από συναρτήσεις που είναι μέλη της κλάσης και από κλάσεις οι οποίες έχουν ρητώς το δικαίωμα πρόσβασης από την ίδια την κλάση (φίλοι). Ένα μέλος κλάσης ορισμένο ως protected είναι προσβάσιμο μόνο από μέλη των κλάσεων τα οποία κληρονομούν από την κλάση καθώς και από τα μέλη της ίδιας κλάσης και "φίλων" αν υπάρχουν.
Η αρχή της αντικειμενοστρέφειας είναι ότι όλες οι συναρτήσεις (και μόνον οι συναρτήσεις) οι οποίες έχουν δικαίωμα πρόσβασης της εσωτερικής παρουσίασης ενός τύπου θα πρέπει να ενθυλακωθούν μέσα στον ορισμό του τύπου. Η C++ υποστηρίζει αυτή την αρχή (μέσω συναρτήσεων μελών και φίλων) αλλά δεν το επιβάλλει. Ο προγραμματιστής μπορεί να δηλώσει μέλη ή και όλοι την παρουσίαση ενός τύπου να είναι public και επιτρέπεται να φτάξει public οντότητες οι οποίες δεν αποτελούν τμήμα της παρουσίασης του τύπου. Έτσι η C++ υποστηρίζει περάν του αντικειμενοστραφούς προγραμματισμού και άλλα όχι το ίδιο ισχυρά παραδείγματα όπως είναι ο αρθρωτός προγραμματισμός.
Γενικά θεωρείτε καλή προγραμματιστική τακτική η δήλωση όλων των δεδομένων ως private ή protected και να ορίζονται ως public μόνο αυτές οι συναρτήσεις οι οποίες αποτελούν μέρος ενός μικρού συστήματος διεπαφής των χρηστών με την κλάση. Έτσι μπορεί να παραμείνουν κρυφές λεπτομέρειες της υλοποίησης των δεδομένων επιτρέποντας στον σχεδιαστή αργότερα να μπορεί να αλλάξει θεμελιωδώς την υλοποίηση δίχως να πρέπει να αλλάξει έστω και ελάχιστα τον τρόπο διεπαφής[157][158].

3.2.11 Κληρονομικότητα

Η κληρονομικότητα επιτρέπει σε ένα τύπο δεδομένων να αποκτήσει ιδιότητες άλλων τύπων. Η κληρονομιά από μια κλάση βάση μπορεί να δηλωθεί public, protected ή και private. Αυτός ο προσδιοριστής πρόσβασης καθορίζει το αν άσχετες μαζί τους και συμπληρωματικές κλάσεις μπορούν να έχουν πρόσβαση στα κληρονομηθέντα public και protected μέλη της κλάσης βάσης. Μόνο τα public κληρονομηθέντα αντιστοιχούν σε αυτό που θέλουμε να εννοούμε ως κληρονομιά. Οι άλλοι δύο τύποι είναι πολύ σπανιότερα χρησιμοποιούμενοι. Εάν ο προσδιοριστής πρόσβασης παραλείπεται μια κλάση κληρονομεί prvately ενώ μια δομή κληρονομεί publicly. Οι κλάσεις βάσεις μπορεί να δηλωθούν ως virtual. Αυτό αποκαλείται virtual κληρονομικότητα. Η virtual

157 *Sutter, Herb; Alexandrescu, Andrei (2004). C++ Coding Standards: 101 Rules, Guidelines, and Best Practices. Addison-Wesley.*

158 *Henricson, Mats; Nyquist, Erik (1997). Industrial Strength C++. Prentice Hall. ISBN 0-13-120965-5.*

κληρονομικότητα διασφαλίσει ότι μόνο ένα στιγμιότυπο της κλάσης βάσης υπάρχει στο διάγραμμα κληρονομικότητας αποφεύγοντας με τον τρόπο αυτό πολλά από τα προβλήματα ασάφειας της έννοιας της πολλαπλής κληρονομικότητας.
Η πολλαπλά κληρονομικότητα στην C++ είναι ένα χαρακτηριστικό το οποίο δεν απαντάται στις περισσότερες άλλες γλώσσες και επιτρέπει σε μια κλάση να συμπληρώνεται από περισσότερες από μια κλάσεις. Αυτό επιτρέπει περισσότερο επεξεργασμένες σχέσεις κληρονομικότητας. Για παράδειγμα μια “Flying Cat” κλάση μπορεί να κληρονομεί τόσο από την κλάση “Cat” οσο και από την κλάση “Flying Mammal”. Κάποιες άλλες γλώσσες όπως η C# και η Java κατεφέρνουν κάτι αντίστοιχο (αν και πιο περιορισμένο) επιτρέποντας την κληρονομικότητα από πολλαπλά interfaces ενώ περιορίζουν τον αριθμό της κλάσης βάσης σε μία (τα interfaces σε αντίθεση με τις κλάσεις παρέχουν μόνο δηλώσεις συναρτήσεων μελών και όχι υλοποιήσεις των δεδομένων μελών). Το interface όπως στην C# και στην Java μπορεί να οριστεί στην C++ σαν μια κλάση η οποία περιέχει μόνο εικονικές συναρτήσεις συχνά αποκαλούμενη και ως abstract κλάση ή ABC. Οι συναρτήσεις μέλη μιας τέτοιας κλάσης υπο κανονικές συνθήκες ορίζονται ρητώς στην συμπληρωματική κλάση και δεν κληρονομούνται σιωπηλά. Η virtual κληρονομικότητα της C++ παρουσιάζει μια ασαφή ανάλυση η οποία καλείται κυριαρχία.

3.2.12 Τελεστές και υπερφόρτωση τελεστών

Τελεστές οι οποίοι δεν μπορεί να υπερφορτωθούν

Τελεστής	**Σύμβολο**
Scope resolution Τελεστής	::
Conditional Τελεστής	?:
dot Τελεστής	.
Member selection Τελεστής	.*
"sizeof" Τελεστής	sizeof
"typeid" Τελεστής	typeid

Η C++ παρέχει περισσότερους από 35 τελεστές μαζί με τους τελεστές βασικής αριθμητικής, την διαχείριση των bit, τα πλάγια μέσα, τους τελεστές σύγκρισης, τους λογικούς τελεστές και άλλους. Σχεδόν όλοι οι τελεστές μπορούν να υπερφορτωθούν για τύπους που ορίζονται από τον χρήστη με μερικές εξαιρέσεις οι οποίες φαίνονται στον παραπάνω πίνακα. Το πλούσιο σύνολο από τελεστές που μπορούν να υπερφορτωθούν είναι κεφαλαιώδες ώστε να είναι δυνατόν οι τύποι που ορίζει ο χρήστης να μοιάζουν με τους ενσωματωμένους τύπους.
Οι υπερφορτώσιμοι τελεστές είναι επίσης μεγάλης σημασίας για πολλές από τις προχωρημένες προγραμματιστικές τεχνικές της C++ όπως είναι ας πούμε οι έξυπνοι δείκτες. Η υπερφόρτωση ενός τελεστή δεν αλλάζει το προβάδισμα του κατά τους υπολογισμούς, ούτε τροποποιεί το πλήθος των τελεσταίων που χρησιμοποιεί ο τελεστής (κάθε τελεσταίος ωστόσο μπορεί να αγνοηθει από τον τελεστή αν και θα αξιολογηθεί πριν την εκτέλεση). Οι υπερφορτωμένοι && και || τελεστές χάνουν την ιδιότητα βραχυκυκλωμένης αξιολόγησης τους.

3.2.13 Πολυμορφισμός

Ο πολυμορφισμός επιτρέπει τη χρήση ένας κοινού προτύπου για πολλές υλοποιήσεις και για τα αντικείμενα να αντιδρούν διαφορετικά κάτω από διαφορετικές καταστάσεις.
Η C++ υποστηρίζει διάφορα είδη staic (χρόνου μεταγλώττισης) και dynamic (χρόνου εκτέλεσης) πολυμορφισμού, τα οποία υποστηρίζονται από τα χαρακτηριστικά της γλώσσας που περιγράψαμε πιο πανω. Ο πολυμορφισμός χρόνου μεταγλώττισης δεν επιτρέπει ορισμένες αποφάσεις χρόνου εκτέλεσης, ενώ ο πολυμορφισμός χρόνου εκτέλεσης συνήθως συνεπάγεται κόστος στην απόδοση.

3.2.13.1 Static πολυμορφισμός

Η υπερφόρτωση συναρτήσεων επιτρέπει σε ένα πρόγραμμα να ορίσει πολλαπλές συναρτήσεις με το ίδιο όνομα αλλά με διαφορετικά ορίσματα (ad hoc πολυμορφισμός). Οι συναρτήσεις ξεχωρίζουν από το πλήθος ή τον τύπο των τυπικών τους παραμέτρων. Έτσι το ίδιο όνομα συνάρτησης μπορεί να συνεπάγεται την κλήση διαφορετικών συναρτήσεων ανάλογα με το περιεχόμενο με το οποίο χρησιμοποιείται. Ο τύπος επιστροφής από την συνάρτηση δεν χρησιμοποιείται για την διάκριση υπερφορτωμένων συναρτήσεων και μια τέτοια προσπάθεια θα είχε σαν αποτέλεσμα ένα σφάλμα χρόνο μεταγλώττισης.
Όταν δηλώνει μια συνάρτηση ο προγραμματιστής μπορεί να ορίσει για μια ή περισσότερες παραμέτρους μια αρχική τιμή. Έτσι επιτρέπει στις παραμέτρους κατά συνθήκη να παραλείπονται κατά την κλήση της συνάρτησης περίπτωση κατά την οποία η ορισμένες αρχικές τιμές θα χρησιμοποιηθούν. Όταν μια συνάρτηση καλείται με λιγότερα ορίσματα από όσα ορίζεται να έχει, σαφή ορίσματα ανατίθενται σε παραμέτρους από αριστερά προς τα δεξιά με κάθε παράμετρο που δεν ανατίθεται να αναγράφεται στο τέλος της λίστας ορισμάτων και να παίρνει τις τιμές που έχει ορίσει ο προγραμματιστής. Σε πολλές περιπτώσεις ο ορισμός αρχικών τιμών σε μια απλή κλήση συνάρτησης είναι προτιμότερο από την υπερφόρτωση της συνάρτησης με διαφορετικό αριθμό παραμέτρων.
Τα πρότυπα στην C++ παρέχουν ένα εκλεπτυσμένο τρόπο γραφής γενικού προγραμματισμού, πολυμορφικού κώδικα. Συγκεκριμένα μέσω του Curiously Recurring Template Pattern είναι δυνατή η υλοποίηση static πολυμορφισμού ο οποίος μιμείται το συντακτικό παράκαμψης virtual συναρτήσεων. Επειδή τα πρότυπα της C++ έχουν επίγνωση των τύπων και είναι ολοκληρωμένα κατά Turing μπορούν επίσης να χρησιμοποιηθούν για να επιτρέψουν στον μεταγλωττιστή να λύσει καταστάσεις αναδρομικότητας και να παράγει αναπληρωματικά προγράμματα μέσω του μεταπρογραμματισμού προτύπων. Παρά την πεποίθηση ορισμένων ο προγραμματισμός προτύπων δεν παράγει ογκώδη κώδικα μετά την μεταγλώττιση με τις ρυθμίσεις του μεταγλωττιστή[159].

---

159 *"Nobody Understands C++: Part 5: Template Code Bloat". http://blog.emptycrate.com/: EmptyCrate Software. Travel. Stuff. 6 May 2008. Retrieved 8 March 2010.* On occasion you will read or hear someone talking about C++ templates causing code bloat. I was thinking about it the other day and thought to myself, "self, if the code does exactly the same thing then the compiled code cannot really be any bigger, can it?" [...] And what about compiled code size? Each were compiled with the command g++ <filename>.cpp -O3. Non-template version: 8140 bytes, template version: 8028 bytes!

3.2.13.2 Dynamic πολυμορφισμός

Οι μεταβλητές δείκτες (και οι αναφορές) σε μια κλάση βάση στην C++ μπορεί να αναφέρονται σε αντικείμενα οποιασδήποτε συμπληρωματικής κλάσης του τύπου, επιπλέον των αντικειμένων που με πλήρη αντιστοίχηση στον τύπο μεταβλητής. Αυτό επιτρέπει στους πίνακες και σε κάθε άλλο είδος δοχείου να κρατά δείκτες σε αντικείμενα διαφορετικών τύπων. Επειδή η ανάθεση των τιμών σε μεταβλητές συνήθως συμβαίνει κατά των χρόνο εκτέλεσης αυτό είναι απαραίτητα ένα φαινόμενο του χρόνου εκτέλεσης.
Η C++ επίσης παρέχει ένα τελεστή dynamic_cast ο οποίος επιτρέπει στο πρόγραμμα να προσπαθήσει με ασφάλεια την μετατροπή ενός αντικειμένου σε ένα άλλο αντικείμενο περισσότερο συγκεκριμένο (σε αντίθεση με την μετατροπή σε ένα πιο γενικό τύπο η οποία πάντα επιτρέπεται). Αυτό το χαρακτηριστικό βασίζεται σε πληροφορίες τύπων χρόνου εκτέλεσης (RTTI). Τα αντικείμενα τα οποία ανήκουν σε ένα συγκεκριμένο τύπο μπορούν επίσης να μετατραπούν σε αυτόν τον τύπο με την static_cast, ένα κατασκεύασμα αμιγώς χρόνου μεταγλώττισης το οποίο διόλου δεν έχει ανάγκη τον χρόνο εκτέλεσης και δεν χρειάζεται το RTTI.

3.2.14 Virtual συναρτήσεις μέλη

Κανονικά όταν μια συνάρτηση σε μια συμπληρωματική κλάση παρακάμπτει μια συνάρτηση κλάσης βάσης η συνάρτηση στην κλήση καθορίζεται από τον τύπο του αντικειμένου. Μια δεδομένη συνάρτηση έχει παρακαμφθεί όταν δεν υπάρχει καμιά διαφορά στον αριθμό ή στον τύπο των ορισμάτων μεταξύ δύο ή περισσότερων ορισμών της συνάρτησης αυτής. Έτσι, κατά τον χρόνο μεταγλώττισης, μπορεί να μην είναι δυνατόν να καθοριστεί ο τύπος του αντικειμένου και άρα η σωστή προς κλήση συνάρτηση, με δεδομένο μόνο ένα δείκτη σε κλάση βάση. Η απόφαση λοιπόν αναβάλλεται έως τον χρόνο εκτέλεσης. Αυτό καλείται δυναμική αποστολή. Οι virtual συναρτήσεις μέλη ή μέθοδοι[160] επιτρέπουν την πλέον συγκεκριμένη υλοποίηση της συνάρτησης που πρόκειται να κληθεί σύμφωνα με τον πραγματικό τύπο του αντικειμένου κατά τον χρόνο εκτέλεσης. Στις υλοποιήσεις της C++ αυτό γίνεται συνήθως χρησιμοποιώντας πίνακες virtual συναρτήσεων. Αν ο τύπος του αντικειμένου είναι γνωστός αυτό μπορεί να παρακαμφθεί προσθέτοντας στην αρχή το όνομα μιας κλάσης που έχει πλήρως τα προσόντα πριν την κλήση της συνάρτησης. Αλλά σαν γενικός κανόνας οι κλήσεις στις virtual συναρτήσεις λύνονται κατά τον χρόνο εκτέλεσης.
Επιπροσθέτως των στάνταρ μελών συναρτήσεων, της υπερφόρτωσης τελεστών και οι καταστροφείς μπορούν να είναι virtual. Σαν κανόνας ορθότητας αν οποιαδήποτε συνάρτηση μέσα στην κλάση είναι virtual το ίδιο θα πρεπει να είναι και ο καταστροφέας. Καθώς ο τύπος ενός αντικειμένου κατά την δημιουργία του είναι γνωστός στον χρόνο εκτέλεσης, οι κατασκευαστές και κατ' επέκταση τα αντίγραφα των κατασκευαστών δεν μπορούν να είναι virtual. Παρόλα αυτά μπορεί να παρατηρηθεί μια κατάσταση κατά την οποία το αντίγραφο ενός αντικειμένου πρέπει να δημιουργηθεί όταν ένας

160 *Stroustrup, Bjarne (2000). The C++ Programming Language (Special ed.). Addison-Wesley. p. 310. ISBN 0-201-70073-5.* A virtual member function is sometimes called a method.

δείκτης σε ένα συμπληρωματικό αντικείμενο περνάει σε ένα δείκτη σε αντικείμενο βάσης. Σε αυτήν την περίπτωση μια κοινά χρησιμοποιούμενη λύση είναι να δημιουργηθεί ένας κλώνος (clone()) virtual συνάρτηση η οποία δημιουργεί και επιστρέφει ένα αντίγραφο της συμπληρωματικής κλάσης όταν κληθεί.
Μια συνάρτηση μέλος μπορεί επίσης να φτιαχτεί pure virtual προσαρτώντας με το =0 μετά το κλείσιμο της παρένθεσης και πριν το ερωτηματικό. Μια κλάση που περιέχει μια pure virtual συνάρτηση καλείται abstract τύπος δεδομένων. Δεν είναι δυνατή η δημιουργία αντικειμένων από abstract τύπους δεδομένων. Μπορούν μόνο να κληρονομηθούν τα χαρακτηριστικά τους. Κάθε συμπληρωματική κλάση κληρονομεί την virtual συνάρτηση σαν pure και πρέπει να παρέχει έναν μη pure ορισμό πριν τα αντικείμενα των συμπληρωματικών κλάσεων δημιουργηθούν. Ένα πρόγραμμα το οποίο προσπαθεί να δημιουργήσει ένα αντικείμενο μιας κλάσης, με pure virtual συνάρτηση μέλος ή το οποίο κληρονόμησε μια pure virtual συνάρτηση μέλος, είναι ill-formed.

3.2.15 Λάμδα εκφράσεις

Η C++ παρέχει υποστήριξη για ανώνυμες συναρτήσεις οι οποίες είναι επίσης γνωστές ως λάμδα εκφράσεις και έχουν την ακόλουθη μορφή:
[σύλληψη](ορίσματα) -> επιστρεφόμενος τύπος {σώμα συνάρτησης}
Η λίστα [σύλληψη] υποστηρίζει τον ορισμό των κλεισιμάτων. Τέτοιες λάμδα εκφράσεις ορίζονται στο πρότυπο σαν συντακτική ζάχαρη για μια ανώνυμη συνάρτηση αντικειμένου. Ένα παράδειγμα λάμδα συνάρτησης μπορεί να οριστεί ως ακολούθως:
[](int x, int y) -> int {return x+y;}

3.2.16 Διαχείριση εξαιρέσεων

Η διαχείριση των εξαιρέσεων χρησιμοποιείται για να καταστεί δυνατή η επικοινωνία της ύπαρξης ενός λάθους χρόνου εκτέλεσης ή λάθους από το σημείο που εντοπίστηκε στο σημείο που το ζήτημα μπορεί να διευθετηθεί[161]. Επιτρέπει αυτό να γίνει με ένα τυπικό τρόπο και έξω από το κυρίως σώμα κώδικα κατά την ανίχνευση όλων των προβλημάτων[162]. Αν ένα λάθος προκύψει μια εξαίρεση πετιέται (raised) η οποία στη συνέχεια πιάνεται από τον κοντινότερο κατάλληλο διαχειριστή εξαίρεσης. Η εξαίρεση προκαλεί έξοδο από το τρέχον σώμα κώδικα καθώς και από οποιοδήποτε εξωτερικό σώμα έως ότου να βρεθεί ο κατάλληλος διαχειριστής εξαιρέσεων, καλώντας με τη σειρά του τους καταστροφείς κάθε σώματος κώδικα από το οποίο εξήλθε[163]. Την ίδια στιγμή η εξαίρεση παρουσιάζεται σαν ένα αντικείμενο το οποίο μεταφέρει δεδομένα σχετικά με το πρόγραμμα[164].

---

161 *Mycroft, Alan (2013). "C and C++ Exceptions | Templates" (PDF). Cambridge Computer Laboratory - Course Materials 2013-14. Retrieved July 2014.*

162 *Stroustrup, Bjarne (2013). The C++ Programming Language. Addison Wesley. p. 345. ISBN 9780321563842.*

163 *Stroustrup, Bjarne (2013). The C++ Programming Language. Addison Wesley. pp. 363–365. ISBN 9780321563842.*

164 *Stroustrup, Bjarne (2013). The C++ Programming Language. Addison Wesley. pp. 345, 363.*

Ο κώδικας που παράγει την εξαίρεση τοποθετείται μέσα σε ένα try σώμα κώδικα. Οι εξαιρέσεις διαχειρίζονται σε διαφορετικά catch σώματα. Κάθε σώμα try μπορεί να έχει πολλαπλούς διαχειριστές[165].
Είναι επίσης δυνατή η δημιουργία εξαιρέσεων εσκεμμένα με την χρήση της εντολής throw. Ο χειρισμός των εξαιρέσεων αυτών γίνεται με τον συνήθη τρόπο. Σε μερικές περιπτώσεις οι εξαιρέσεις δεν είναι δυνατόν να χρησιμοποιηθούν για τεχνικούς λόγους. Ένα τέτοιο παράδειγμα αποτελούν τα βασικά συστατικά ενός ενσωματωμένου συστήματος, όπου κάθε εργασία πρέπει να είναι εγγυημένο ότι θα τελειώσει μέσα σε ένα συγκεκριμένο χρονικό όριο. Αυτό δεν είναι δυνατόν να καθοριστεί μέσω εξαιρέσεων καθώς δεν υπάρχουν εργαλεία για τον προσδιορισμό του ελάχιστου απαιτούμενου χρόνου για τον χειρισμό μιας εξαίρεσης[166].

3.2.17 Η στάνταρ βιβλιοθήκη

Το πρότυπο της C++ αποτελείται από δύο μέρη: τον πυρήνα της γλώσσας και την στάνταρ βιβλιοθήκη. Η προγραμματιστές της C++ περιμένουν τις τελευταίες εξελίξεις από κάθε μεγάλη υλοποίηση της C++. Περιλαμβάνει vectors, lists, maps, algorithms, sets, queues, stacks, arrays, tupples, imput/output δομές, smart δείκτες, υποστήριξη συνήθων εκφράσεων, βιλβιοθήκη πολυνηματικού προγραμματισμού, υποστήριξη atomics, υπηρεσίες που αφορούν τον χρόνο, ένα σύστημα μετατροπής της αναφοράς των λαθών το οποίο δεν χρησιμοποιεί την διαχείριση εξαιρέσεων της C++ εντός των εξαιρέσεων της C++, γεννήτρια τυχαίων αριθμών και μια ελαφρώς τροποποιημένη έκδοση της στάνταρ βιβλιοθήκης της C.
Ένα μεγάλο κομμάτι της βιβλιοθήκης της C++ βασίζεται στην Standard Template Library (STL). Χρήσιμα εργαλεία που παρέχονται από την STL είναι τα containers των συλλογών αντικειμένων (όπως είναι οι vectors και οι lists) εκφράσεις που παρέχουν πρόσβαση στα container που μοιάζει με την πρόσβαση σε πίνακες και αλγόριθμους για την εκτέλεση εργασιών όπως αναζήτηση και ταξινόμηση.
Επιπλέον (πολυ) maps και (πολυ) sets παρέχονται χαρακτηριστικά για τα οποία εξάγουν συμβατά interfaces. Έτσι με την χρήση των προτύπων είναι δυνατή η συγραφή γενικών αλγορίθμων οι οποίοι είναι συμβατοί με κάθε είδους container. Όπως και στην C τα χαρακτηριστικά της βιβλιοθήκης είναι προσβάσιμα με την χρήση της include για να συμπεριληφθεί μιας στάνταρ κεφαλίδα. Η C++ παρέχει 105 στάνταρ κεφαλίδες από τις οποίες οι 27 έχουν αποδοκιμαστεί.
Το πρότυπο ενσωματώνει την STL η οποία αρχικά σχεδιάστηκε από τον Alexander Stepanov ο οποίος πειραματίστηκε με γενικούς αλγορίθμους και containers για πολλά χρόνια. Όταν ξεκίνησε με την C++ ανακάλυψε τελικά μια γλώσσα με την οποία ήταν δυνατή η δημιουργία γενικών αλγορίθμων οι οποίοι απέδιδαν καλύτερα ακόμα και από τους αλγορίθμους της στάνταρ βιβλιοθήκης της C χάρη στα χαρακτηριστικά της C++ που χρησιμοποιούσαν inlining και δεσμούς

ISBN 9780321563842.

165 *Stroustrup, Bjarne (2013). The C++ Programming Language. Addison Wesley. pp. 344, 370. ISBN 9780321563842.*

166 *Stroustrup, Bjarne (2013). The C++ Programming Language. Addison Wesley. p. 349. ISBN 9780321563842.*

χρόνου μεταγλώττισης αντί για δείκτες σε συναρτήσεις. Το πρότυπο δεν αναφέρεται σε αυτό ως STL μιας και πρόκειται απλώς για ένα τμήμα της στάνταρ βιβλιοθήκης, αλλά ο όρος βρίσκεται ακόμη ευρέως σε χρήση για να το διαχωρίσει από το υπόλοιπο τμήμα της στάνταρ βιβλιοθήκης[167].
Οι περισσότεροι μεταγλωττιστές C++ και οι περισσότεροι μεγάλοι παρέχουν υλοποιήσεις της στάνταρ βιβλιοθήκης οι οποίες είναι συμμορφωμένες με την βιβλιοθήκη του προτύπου.

3.2.18 Συμβατότητα

Για να δώσε στους υλοποιητές των μεταγλωττιστών μεγαλύτερη ελευθερία η επιτροπή προτύπου της C++ αποφάσισε να μην υποδείξει την υλοποίηση κατακρεούργησης ονομάτων, την διαχείριση των εξαιρέσεων και άλλα χαρακτηριστικά. Το πρόβλημα που προκύπτει βέβαια από αυτή την απόφαση είναι ότι ο αντικειμενικός κώδικας που παράγεται από διαφορετικούς μεταγλωττιστές αναμένεται να μην είναι συμβατός. Υπήρξαν ωστόσο απόπειρες τυποποίησης των μεταγλωττιστών για συγκεκριμένα μηχανήματα ή λειτουργικά συστήματα, μια προσπάθεια η οποία πλέον έχει εγκαταλειφθεί[168].

3.2.18.1 Με την C

Η C++ συχνά λογίζεται ως υπερσύνολο της C. Αυτό δεν είναι απολύτως ακριβές. Ο περισσότερος κώδικας σε C μπορεί εύκολα να μετατραπεί ώστε να μεταγλωττίζεται σωστά σε C++. Υπάρχουν ωστόσο κάποιες διαφορές οι οποίες μπορεί να οδηγήσουν σωστό C κώδικα να μην είναι σωστός ή να συμπεριφέρεται διαφορετικά από το ανεμενόμενο στην C++. Για παράδειγμα η C επιτρέπει την σιωπηρή μετατροπή από void* σε άλλα είδη δεικτών ενώ η C++ δεν το επιτρέπει. Επίσης η C++ πρίζει πολλές νέες λέξεις κλειδιά όπως είναι οι new και class οι οποίες μπορεί να χρησιμοποιούνται ως αναγνωριστικά σε ένα πρόγραμμα C.
Μερικές από τις ασυμβατότητες απομακρύνθηκαν κατά την αναθεώρηση του προτύπου το 1999 του προτύπου της C η οποία πλέον υποστηρίζει χαρακτηριστικά της C++ όπως είναι τα σχόλια γραμμής και οι δηλώσεις που είναι ανάμικτες με τον κώδικα. Από την άλλη η C99 εισήγαγε ένα σύνολο νέων χαρακτηριστικών τα οποία η C++ δεν υποστήριζε και τα οποία ήταν ασύμβατα ή περιττά στην C++ όπως είναι οι πίνακες *"C++0x Support in GCC". Retrieved 12 October 2010.* μεταβλητού μήκους, οι φυσικά πολύπλοκοι τύποι αριθμών και άλλα. Μερικά από τα νεοεισαχθέντα χαρακτηριστικά της C99 εισήχθησαν στο πρότυπο C++11.

3.2.19 Κριτικές

Παρά την ευρύτατη υιοθέτηση της πολλοί προγραμματιστές άσκησαν κριτική στην C++. Μερικοί

167 *Graziano Lo Russo (2008). "An Interview with A. Stepanov". stlport.org. Retrieved 2015-10-08.*

168 *"C++ ABI Summary". 20 March 2001. Retrieved 30 May 2006.*

από αυτούς ήταν οι Linus Torvalds[169], Richard Stallman[170] και Krn Thompson[171]. Τα θέμετα που έθεσαν περιλαμβάνουν την έλλειψη αντανάκλασης ή της αποκομιδής απορριμάτων ή τους αργούς χρόνους μεταγλώττισης και τα πολύλογα μηνύματα λάθους ιδίως κατά τον προγραμματισμό προτύπων[172]. Επειδή η C++ εισάγει ένα επιπλέον αποτύπωμα μνήμης στα προγράμματα εξ' αιτίας των εσωτερικά δημιουργούμενων vtables και constructors προγραμματιστές σαν τον Torvards προτιμούν την C από την C++ για επιδόσεις χαμηλού επιπέδου και για μεταφερσιμότητα του λογισμικού μεταξύ συστημάτων[173].

3.3 Η γλώσσα προγραμματισμού C Sharp (C#)

3.3.1 Εισαγωγή

Η C#[174] είναι μια γλώσσα προγραμματισμού πολλαπλού παραδείγματος η οποία ενσωματώνει την ισχυρή γραφή, την επιτακτικότητα, την δηλωτικότητα, τον διαδικασιακό προγραμματισμό, τον γενικευμένο προγραμματισμό, τον αντικειμενοστραφή προγραμματισμό και τις αντικειμενοστραφείς αρχές με προσανατολισμό συστατικών. Αναπτύχθηκε από την Microsoft μέσα στα πλαίσια της πρωτοβουλίας του .NET και αργότερα εγκρίθηκε ως πρότυπο από την Ecma (Ecma-334) και σαν ISO(ISO/IEC 23270:2006). Η C# είναι μια από τις γλώσσες που σχεδιάστηκαν για την κοινή γλωσσική υποδομή (Comon Language Infrastracture).

Εικόνα 11: Το λογότυπο της C#

Η C# είναι μια γενικού σκοπού, αντικειμενοστραφής γλώσσα προγραμματισμού[175]. Η ομάδα ανάπτυξης είχε επικεφαλής τον Anders Hejlsberg. Η πλέον πρόσφατη έκδοση της είναι η C# 6 και κυκλοφόρησε στις 20 Ιουλίου του 2015[176].

3.3.2 Στόχοι του σχεδιασμού

---

169 *"Re: [RFC] Convert builin-mailinfo.c to use The Better String Library" (Mailing list). 6 September 2007. Retrieved 31 March 2015.*

170 *"Re: Efforts to attract more users?" (Mailing list). 12 July 2010. Retrieved 31 March 2015.*

171 *Andrew Binstock (18 May 2011). "Dr. Dobb's: Interview with Ken Thompson". Retrieved 7 February 2014.*

172 *Kreinin, Yossi (October 13, 2009). "Defective C++". Retrieved February 3, 2016.*

173 *"Re: [RFC] Convert builin-mailinfo.c to use The Better String Library" (Mailing list). 6 September 2007. Retrieved 31 March 2015.*

174 *Naugler, David (May 2007). "C# 2.0 for C++ and Java programmer: conference workshop". Journal of Computing Sciences in Colleges **22** (5).* Although C# has been strongly influenced by Java it has also been strongly influenced by C++ and is best viewed as a descendant of both C++ and Java.

175 *C# Language Specification (PDF) (4th ed.). Ecma International. June 2006. Retrieved January 26, 2012.*

176 *Lander, Rich (20 July 2015). "Announcing .NET Framework 4.6". .NET Blog. Microsoft.*

Το πρότυπο της ECMA παραθέτει τους παρακάτω σχεδιαστικούς στόχους:[177]

- Η γλώσσα C# έχει σκοπό να είναι μια απλή, μοντέρνα, γενικού σκοπού, αντικειμενοστραφείς γλώσσα προγραμματισμού.
- Η γλώσσα και άρα οι υλοποιήσεις της θα πρέπει να παρέχει υποστήριξη στις αρχές της μηχανικής λογισμικού, όπως είναι ο έλεγχος ισχυρής γραφής, έλεγχο ορίων πινάκων, ανίχνευση απόπειρας χρήσης μη αρχικοποιημένων μεταβλητών και αυτόματη αποκομιδή απορριμμάτων. Η στιβαρότητα του λογισμικού, η αντοχή και η παραγωγικότητα των προγραμματιστών είναι πολύ σημαντικά.
- Η γλώσσα προορίζεται για χρήση στην ανάπτυξη τμημάτων λογισμικού κατάλληλων για διανομή σε κατανεμημένα περιβάλλοντα.
- Η φορητότητα είναι ιδιαιτέρως σημαντική για τον πηγαίο κώδικα και τους προγραμματιστές, ιδιαιτέρως για όσους είναι εξοικειωμένοι με τις C και C++.
- Η υποστήριξη της διεθνοποίησης είναι πολύ σημαντική.
- Η C# προορίζεται να είναι κατάλληλη για το γράψιμο εφαρμογών τόσο για φιλοξενούμενα όσο και για ενσωματωμένα συστήματα, με διακύμανση από τα πολύ μεγάλης κλίμακας τα οποία χρησιμοποιούν εξεζητημένα λειτουργικά συστήματα έως τα πιο μικρά τα οποία έχουν αφοσιωμένες λειτουργίες.
- Αν καοι οι εφαρμογές σε C# προορίζεται να είναι οικονομική όσον αφορά τις απαιτήσεις σε μνήμη και υπολογιστική ισχύ η γλώσσα δεν προοριζόταν να συγκριθεί άμεσα στους τομείς της απόδοσης και του μεγέθους με την C ή την assembly.

3.3.3 Ιστορία

Κατά την ανάπτυξη του .NET Framework οι βιβλιοθήκες των κλάσεων αρχικά γράφτηκαν χρησιμοποιώντας ένα σύστημα μεταγλωττιστών διαχείρισης κώδικα το οποίο αποκαλέιται Simple Managed C (SMC)[178][179]. Τον Ιανουάριο του 1999 ο Anders Hejlsberg σχημάτισε μια ομάδα για την δημιουργία μιας νέας γλώσσας η οποία τον καιρό εκείνο ονομαζόταν Cool αρκτικόλεξο για το C-like Object Orientd Language[180]. Η αρχική σκέψη ήταν να παραμείνει αυτό ως όνομα της γλώσσας αλλά στην πορεία εγκαταλείφθηκε για λόγους πνευματικών δικαιωμάτων. Μέχρι την στιγμή που το .NET ανακοινώθηκε δημόσια τον Ιούλιο του 200 στο Professional Develoopers Conference η γλώσσα είχε μετονομαστεί σε C# και οι βιβλιοθήκες των κλάσεων και το ASP.NET είχαν μεταφερθεί στην C#.

---

177 *C# Language Specification (PDF) (4th ed.). Ecma International. June 2006. Retrieved January 26, 2012.*

178 *Zander, Jason (November 24, 2008). "Couple of Historical Facts". Retrieved February 23, 2009.*

179 *Guthrie, Scott (November 28, 2006). "What language was ASP.Net originally written in?". Retrieved February 21, 2008.*

180 *Hamilton, Naomi (October 1, 2008). "The A-Z of Programming Languages: C#". Computerworld. Retrieved October 1, 2008.*

Ο επικεφαλής σχεδιαστή και αρχιτέκτονας της C# στην Microsoft είναι ο Anders Hejlsberg ο οποίος προηγουμένως είχε εμπλακεί στην ανάπτυξη της Turbo Pascal, της Embarcadero Delphi και της Visual J++. Σε συνεντεύξεις και τεχνικές δημοσιεύσεις έχει δηλώσει ότι οι ατέλειες στις κυριότερες γλώσσες προγραμματισμού (C++, Java, Delphi και Smaltalk) οδήγησαν στην θεμελίωση της Common Language Runtime (CLR) η οποία με τη σειρά της οδήγησε στον σχεδιασμό της C#.

Ο James Gosling ο οποίος δημιούργησε τηνς Java το 1994 και ο Bill Joy ένας συνιδρυτής της Sun Microsystems, απεκάλεσαν την C# μια απομίμηση της Java. Ο Gosling επιπλέον δήλωσε ότι η C# είναι ένα είδος Java με τα χαρακτηριστικά της αξιοπιστίας, της παραγωγικότητας και της ασφάλειας διαγραμμένα. [181][182]Ο klaus Kreft και η Angelika Langer (συγγραφείς bβιβλίων για την C++) δήλωσαν ότι οι Java και C# είναι σχεδόν πανομοιότυπες γλώσσες προγραμματισμού και συνέχισε “βαρετή επανάληψη η οποία στερείται καινοτομίας[183], σχεδόν κανείς δεν θα ισχυριστεί ότι η Java και η C# είναι επαναστατικές γλώσσες οι οποίες άλλαξαν τον τρόπο που γράφουμε προγράμματα και η C# δανείστηκε αρκετά από την Java και αντίστροφα. Τώρα που η C# υποστηρίζει boxing και unboxing υπάρχει μια παρόμοια ιδιότητα στην Java”. [184]Τον Ιούλιο του 2000 ο Hejlsberg δήλωσε ότι η C# δεν αποτελεί κλώνο της Java και είναι σχεδιαστηκά περισσότερο κοντά στην C++[185].

Από την στιγμή της κυκλοφορίας της C# 2.0 τον Νοέμβριο του 2005 ηη C# και η Java αναπτύχθηκαν σε εντελώς διαφορετικές τροχιές με αποτέλεσμα να γίνονται όλο και πιο διαφορετικές. Μια από τις πρώτες διαφοροποιήσεις ήρθε με την προσθήκη του γενικευμένου και στις δύο γλώσσες με εντελώς διαφορετικές υλοποιήσεις. Η C# κάνει χρήση της πραγμοποίησης για να παρέχει γενικευμένα αντικείμενα πρώτης κλάσης τα οποία μπορούν να χρησιμοποιηθούν σαν οποιαδήποτε άλλη κλάση με την παραγωγή κώδικα να λαμβάνει χώρα την στιγμή φορτόματος της κλάσης[186]. Επιπλέον η C# πρόσθεσε αρκετά επιπλέον βασικά χαρακτηριστικά για την διευκόλυνση του διαδικασιακού προγραμματισμού με αποκορύφωμα την LINQ επέκταση που κυκλοφόρησε με την C# 3.0 έκδοση και το υποστηρικτικό της πλαίσιο εργασίας εκφράσεων λάμδα, επέκτασης μεθόδων, και ανώνυμων τύπων[187]. Αυτά τα χαρακτηριστικά καθιστούν ικανούς τους προγραμματιστές να χρησιμοποιούν τεχνικές διαδικασιακού προγραμματισμού όπως είναι τα closures όταν αυτό ευνοεί την εργασία τους. Οι LINQ επεκτάσεις και οι διαδικασιακές εισαγωγές βοηθούν τους προγραμματιστές στην μείωση του στερεοτυπικού κώδικα ο οποίος περιλαμβάνεται

---

181 *Wylie Wong (2002). "Why Microsoft's C# isn't". CNET: CBS Interactive. Retrieved May 28, 2014.*

182 *Bill Joy (February 7, 2002). "Microsoft's blind spot". cnet.com. Retrieved January 12, 2010.*

183 *Klaus Kreft and Angelika Langer (2003). "After Java and C# - what is next?". Retrieved June 18, 2013.*

184 *Klaus Kreft and Angelika Langer (July 3, 2003). "After Java and C# - what is next?". artima.com. Retrieved January 12, 2010.*

185 *Osborn, John (August 1, 2000). "Deep Inside C#: An Interview with Microsoft Chief Architect Anders Hejlsberg". O'Reilly Media. Retrieved November 14, 2009*

186 *"Generics (C# Programming Guide)". Microsoft. Retrieved March 21, 2011.*

187 *Don Box and Anders Hejlsberg (February 2007). "LINQ: .NET Language-Integrated Query". Microsoft. Retrieved March 21, 2011.*

στις κοινές εργασίες όπως είναι τα ερωτήματα στις βάσεις, στην ανάλυση xml αρχείων ή στην αναζήτηση μέσα σε δομές δεδομένων, δίνοντας έμφαση στην πραγματική προγραμματιστική λογική ώστε να υποστηρίξουν την αναγνωσιμότητα και την συντηρισημότητα[188].
Η C# είχε μια μασκότ με τον όνομα Andy. Εγκαταλείφθηκε στις 29 Ιανουαρίου 2004[189].
Η C# αρχικά υποβλήθηκε στην υποεπιτροπή ISO JTC 1/SC 22 για έλεγχο υπό το[190] ISO/IEC 23270:2003[191], αποσύρθηκε και ακολούθως εγκρίθηκε υπό το ISO/IEC 23270:2006[192]

3.3.4 Όνομα

Το όνομα C Sharp εμπνεύστηκε από μια μουσική σημειογραφία όπου ένα sharp υποδηλώνει ότι η γραμμένη νότα πρέπει να παιχτεί ένα ημίτονο υψηλότερα από τις υπόλοιπες[193]. Αυτό είναι παρόμοι με την ονομασία της C++ όπου το ++ υποδηλώνει η τιμή μια μεταβλητής θα πρέπει να αυξηθεί κατά ένα. Το sharp σύμβολο επίσης μοιάζει με τέσσερα + σύμβολα υπονοώντας επιπλέον ότι η γλώσσα είναι μια αύξηση της C++[194].

3.3.5 Χαρακτηριστικά διαφοροποίησης

3.3.5.1 Φορητότητα

Λόγω σχεδιασμού η C# είναι η γλώσσα προγραμματισμού οι οποία περισσότερο αντανακλά την έννοια CLI (Common Language Infrastracture). Οι περισσότεροι από τους εσωτερικούς τύπους ανταποκρίνονται σε τύπους δεδομένων υλοποιημένων από το πλαίσιο εργασίας του CLI[195]. Ωστόσο η τεκμηρίωση της γλώσσας δεν δηλώνει τις απαιτήσεις σε επεξεργαστική ισχύ του μεταγλωττιστή. Δεν δηλώνεται δηλαδή ότι ένας μεταγλωττιστής C# θα πρέπει να στοχεύει την Common Language Runtime ή να παράξει Common Intermediate Languafe (CIL) ή να παράξει άλλη συγκεκριμένη μορφή. Θεωρητικά ένας μεταγλωττιστής C# θα μπορούσε να παράξει κώδικα μηχανής όπως οι παραδοσιακοί μεταγλωττιστές C++ και Fortran.

3.3.5.2 Γραφή

---

188 *Mercer, Ian (April 15, 2010). "Why functional programming and LINQ is often better than procedural code". abodit.com. Retrieved March 21, 2011.*

189 *"Andy Retires". Dan Fernandez's Blog. Blogs.msdn.com. January 29, 2004. Retrieved October 4, 2012.*

190 *"Technical committees - JTC 1/SC 22 - Programming languages, their environments and system software interfaces". ISO. Retrieved October 4, 2012.*

191 *"ISO/IEC 23270:2003 - Information technology - C# Language Specification". Iso.org. August 23, 2006. Retrieved October 4, 2012.*

192 *"ISO/IEC 23270:2006 - Information technology - Programming languages - C#". Iso.org. January 26, 2012. Retrieved October 4, 2012.*

193 *Kovacs, James (September 7, 2007). "C#/.NET History Lesson". Retrieved June 18, 2009.*

194 *Hejlsberg, Anders (October 1, 2008). "The A-Z of Programming Languages: C#". Computerworld.*

195 *Visual Studio 2010 and .NET 4 Six-in-One. Wrox Press. 2010. ISBN 0470499486.*

Η C# υποστηρίζει ισχυρής γραφής σιωπηλές δηλώσεις μεταβλητών με την λέξη κλειδί var καθώς και σιωπηλούς τύπους πινάκων με την λέξη κλειδί new[] ακολουθούμενη από έναν αρχικοποιητή συλλογής.
Η C# υποστηρίζει ένα αυστηρό Boolean τύπο δεδομένων bool. Οι δηλώσεις οι οποίες παίρνουν καταστάσεις όπως είναι η while και η if απαιτούν μια έκφραση ενός τύπου που υλοποιεί τον true τελεστή όπως είναι ο τύπος Boolean. Ενώ η C++ διαθέτει ένα τύπο Boolean μπορεί ελεύθερα να μετατραπεί σε και από ακεραίους και εκφράσεις όπως if(a) απαιτώντας μόνο ο a να είναι μετατρέψιμος σε bool και άρα επιτρέποντας ο a να είναι int ή δείκτης. Η C# δεν επιτρέπει αυτή την έκφραση μέσω ακεραίων του αληθούς ή ψευδούς αναγκάζοντας τους προγραμματιστές να χρησιμοποιούν εκφράσεις οι οποίες επιτρέπουν ακριβώς bool τιμές οι οποίες μπορούν να αποτρέψουν από συγκεκριμένους τύπους προγραμματιστικών σφαλμάτων.
Η C# διαθέτει μεγαλύτερη ασφάλεια τύπων από την C++. Οι μόνες σιωπηρές μετατροπές εξ' ορισμού είναι αυτές οι οποίες θεωρούνται ασφαλείς, όπως είναι η διεύρυνση των ακεραίων. Αυτό επιβάλλεται κατά τον χρόνο μεταγλώττισης, κατά το JIT και σε κάποιες περιπτώσεις κατά την διπάρκεια του χρόνου εκτέλεσης. Δεν υπάρχουν σιωπηλές μετατροπές μεταξύ Booleans και ακεραίων ούτε μεταξύ μέλη enumeration και ακεραίων. Κάθε μετατροπή καθορισμένη από τον χρήστη θα πρέπει ρητά να δηλωθεί ως ρητή ή κρυφή σε αντίθεση με τις αντιγραφές κατασκευαστών και τελεστών μετατροπής της C++ οι οποίοι είναι και οι δύο κρυφή εξ' ορισμού.
Η C# έχει ρητή υποστήριξη για covariance και contravariance σε γενικούς τύπους σε αντίθεση με την C++ η οποία σε κάποιο βαθμό υποστηρίζει την contravariance μέσω σημασιολογίας ή τύπων επιστροφής virtual μεθόδων.
Τα μέλη των τύπων enumeration τοποθετούνται σε δικό τους εύρος.
Η C# δεν επιτρέπει global μεταβλητές ή συναρτήσεις. Κάθε μέθοδος και τα μέλη της πρέπει να δηλώνονται μέσα σε κλάσεις. Τα static μέλη public κλάσεων μπορούν να υποκαταστήσουν τις global μεταβλητές και συναρτήσεις.
Οι τοπικές μεταβλητές δεν μπορούν να επισκιάσουν μεταβλητές του σώματος που περικλείουν σε αντίθεση με τις C και C++.

3.3.6 Μεταπρογραμματισμός

Ο μεταπρογραμματισμός μέσω της C# είναι ένα κομμάτι της γλώσσας. Πολλά από τα σχετικά της χαρακτηριστικά αναπαράγουν την χρηστικότητα που συναντούμε στις εντολές προεπεξεργατσή των GCC' s και VisualC++.

3.3.7 Μέθοδοι και συναρτήσεις

Όπως στην C++ και σε αντίθεση με την Java οι προγραμματιστές της C# πρέπει να χρησιμοποιήσουν την λέξη κλειδί virtual για να επιτρέψουν την παράκαμψη μεθόδων από τις υποκλάσεις.

Οι extension μέθοδοι στην C# επιτρέπουν στους προγραμματιστές να χρησιμοποιήσουν static μεθόδους σαν να επρόκειτο για μεθόδους από τον πίνακα μεθόδων μιας κλάσης και επιτρέποντας με αυτόν τον τρόπο να προσθέσουν μεθόδους σε ένα αντικείμενο που θεωρούν ότι θα έπρεπε να υπάρχει σε ένα αντικείμενο και τους απογόνους του.
Ο τύπος dynamic επιτρέπει για απόλυτη δέσμευση μεθόδων κατά τον χρόνο εκτέλεσης, επιτρέποντας έτσι κλήσεις μεθόδων που μοιάζουν με την JavaScript και την σύνθεση αντικειμένων κατά τον χρόνο εκτέλεσης.
Η C# υποστηρίζει τις συναρτήσεις δεικτών ισχυρής γραφής μέσω της δεσμευμένης λέξης delegate. Όπως και στο Qt πλαίσιο εργασίας οι ψευδοεντολές της C++ signal και slot, η C# έχει σημασιολογία συγκεκριμένη ώστε να περιλαμβάνει publish-subscribe τύπου περιστατικά αν και η C# χρησιμοποιεί αντιπροσώπους για να το καταφέρει.
Η C# παρέχει συγχρονισμένες κλήσεις συναρτήσεων με το στυλ της Java μέσω της δήλωσης [MethodImpl](MethodImplOptions, Synchronized), και διαθέτει υποστήριξη για αμοιβαία αποκλειόμενα κλειδώματα μέσω της λέξης κλειδί lock.

3.3.8 Ιδιότητες

Η C# παρέχει ιδιότητες σαν συντακτική ζάχαρη για ένα κοινό μοτίβο στο οποίο ένα ζευγάρι μεθόδων, ο accessor (getter) και ο mutator (setter) ενθυλακώνουν λειτουργίες σε ένα μόνο χαρακτηριστικό της κλάσης. Δεν απαιτείται το γράψιμο περιττών υπογραφών των μεθόδων για την υλοποίηση του setter/getter και η ιδιότητα μπορεί να είναι προσβάσιμη μέσω ιδιωτικής σύνταξης παρά μέσω πολύλογων κλήσεων μεθόδου.

3.3.8.1 Namespace

Στην C# ένα namespace παρέχει το ίδιο επίπεδο απομόνωσης κώδικα όπως τα πακέτα στην Java ή το namespace στην C++ με παρόμοιες ιδιότητες και χαρακτηριστικά με τα πακέτα.

3.3.8.2 Πρόσβαση στη μνήμη

Στην C# οι δείκτες διευθύνσεων μνήμης μπορούν να χρησιμοποιηθούν μόνο εντός τμημάτων που έχουν σημανθεί ως μη ασφαλή και τα προγράμματα με μη ασφαλή κώδικα απαιτούν σχετικές άδειες για να εκτελεστούν. Η περισσότερη πρόσβαση στα αντικείμενα επιτυγχάνεται μέσω αφαλών αναφορών σε αντικείμενα οι οποίες (αναφορές) πάντα δείχνουν είτε σε ένα εν ζωή αντικείμενο ή έχουν την καλά ορισμένη null τιμή. Είναι πιθανό να παρατηρηθούν αναφορές σε νεκρά ανικείμενα (αντικείμενα τα οποία τα έχει καταστρέψει η αποκομιδή απορριμάτων) ή σε ένα τυχαίο τμήμα μνήμης. Ένας μη ασφαλής δείκτης μπορεί να δείχνει σε στιγμιότυπο ενός τύπου τιμής, πίνακα, συμβολοσειρά ή τμήμα μνήμης που έχει διατεθεί σε μια στοίβα. Ο κώδικας που δεν έχει σημανθεί ως μη ασφαλής μπορεί επίσης να αποθηκεύσει και να διαχειριστεί δείκτες μέσω της System.IntPtr type, αλλά δεν μπορεί να τους απελευθερώσει.

Η μνήμη την οποία έχουμε χρησιμοποιήσει δεν είναι δυνατόν να ελευθερωθεί ρητά. Αντιθέτως καθαρίζεται αυτόματα από την αποκομιδή απορριμάτων. Η αποκομιδή απορριμάτων αντιμετωπίζει το πρόβλημα της διαρροής μνήμης ελευθερώνοντας τον προγραμματιστή από την ευθύνη απελευθέρωσης της μνήμης η οποία πλέον δεν είναι απαραίτητη.

3.3.8.3 Εξαιρέσεις

Οι εξαιρέσεις δεν υπάρχουν στην C# (σε αντίθεση με την Java). Πρόκειται για μια συνειδητή επιλογή λόγω ζητημάτων επεκτασιμότητας και ικανότητας παραγωγής εκδόσεων[196].

3.3.8.4 Πολυμορφισμός

Σε αντίθεση με την C++ η C# δεν υποστηρίζει την πολλαπλή κληρονομικότητα αν και μια κλάση μπορεί να υλοποιήσει οποιονδήποτε αριθμό interfaces. Πρόκειται για μια σχεδιαστική επιλογή του αρχιτέκτονα της γλώσσας για την αποφυγή πολυπλοκότητας και απλοποίησης των αρχιτεκτονικών απαιτήσεων μέσω του CLI. Όταν έχουμε την υλοποίηση πολλαπλών interfaces τα οποία περιέχουν μια μέθοδο με το ιδιο αποτύπωμα, η C# επιτρέπει την υλοποίηση της μεθόδου ανάλογα με το interface από το οποίο η μέθοδος καλείται ή όπως η Java επιτρέπει την υλοοίηση της μεθόδου μια φορά και καθορίζει η φορά αυτή να είναι η μόνη επίκληση σε μια κλήση συνάρτησης μέσω οποιουδήποτε inteerface της κλάσης.
Ωστόσο σε αντίθεση με την Java η C# υποστηρίζει την υπερφόρτωση τελεστών. Μόνο οι πιο κοινοί τελεστές της C++ μπορούν να υπερφορτωθούν στην C#.

3.3.8.5 Διαδιακασιακός προγραμματισμός

Αν και αρχικά μια επιτακτική γλώσσα η C#2.0 πρόσφερε περιορισμένη υποστήριξη για τον διαδικασιακό προγραμματισμό μέσω των συναρτήσεων πρώτης κλάσης και κλεισίματα από ανώνυμους αντιπροσώπους. Η C#3.0 επέκτεινε περαιτέρω την υποστήριξη του διαδικασιακού προγραμματισμού με την εισαγωγή ενός ελαφρού συντακτικού λάμδα εκφράσεων, επέκτασης μεθόδων και μια λίστα κανοητού συντακτικού με την μορφή κατανόησης ερωτήσεων.

3.3.9 Το κοινό σύστημα τύπων

Η C# διαθέτει ένα ενοποιημένο σύστημα τύπων. Το ενοποιημένο αυτό σύστημα αποκαλείται Common Type System (CTS)[197]. Ένα ενοποιημένο σύστημα τύπων υποδηλώνει ότι όλοι οι τύποι

196 *Venners, Bill; Eckel, Bruce (August 18, 2003). "The Trouble with Checked Exceptions". Retrieved March 30, 2010.*

197 *Archer, Tom (2001). "Part 2, Chapter 4: The Type System". Inside C#. Redmond, Washington: Microsoft Press. ISBN 0-7356-1288-9.*

και οι πρωτόγονοι όπως είναι οι integers είναι υποκλάσεις της κλάσης System.Object. Για παράδειγμα κάθε τύπος κληρονομεί μια toString() μέθοδο.

3.3.10 Κατηγορίες τύπων δεδομένων

Το CTS χωρίζει τους τύπους δεδομένων σε δύο κατηγορίες[198]:

1. Τύποι αναφοράς
2. Τύποι τιμών

Στιγμιότυπα των τύπων τιμών δεν έχουν αναφορική ταυτότητα ούτε μη αναφορική σημασιολογία σύγκρισης (οι συγκρίσεις ισότητας και μη ισότητας για τους τύπους τιμών γίνεται με σύγκριση των πραγματικών τιμών μέσα στα στιγμιότυπα εκτός και αν οι σχετικοί τελεστές είναι υπερφορτωμένοι). Οι τύποι τιμών συμπληρώνονται από το System.ValueType, πάντα έχουν μια αρχική τιμή και μπορούν πάντα να δημιουργηθούν και να αντιγραφούν. Μερικοί άλλοι περιορισμοί των τύπων τιμών είναι ότι δεν μπορούν να δημιουργηθούν ο ένας από τον άλλο (αλλά μπορούν να υλοποιήσουν interfaces) και δεν μπορούν να έχουν ένα σαφή προεπιλεγμένο (με ορίσματα) κατασκευαστή. Παραδείγματα τύπων τιμών είναι όλοι οι πρωτόγονοι τύποι όπως οι Int, float, char, και ο System.DateTime.

Σε αντίθεση με τους παραπάνω οι τύποι αναφοράς έχουν έννοια της αναφορικής ταυτότητας (κάθε στιγμιότυπο ενός τύπου αναφοράς κληρονομεί χωριστά από κάθε άλλο στιγμιότυπο ακόμα και ανα τα δεδομένα στο εσωτερικό των δύο στιγμιότυπων είναι τα ίδια). Αυτό αντανακλάται στην εξ' ορισμού σύγκριση ισότητας ή μη ισότητας για τους τύπους αναφοράς, οπότε και εξετάζεται η αναφορική και όχι η δομική ισότητα, εκτός και αν οι ανάλογοι τελεστές είναι υπερφορτωμένοι (παράδειγμα η περίπτωση για το System.String). Γενικά δεν είναι πάντα εφικτό να δημιουργήσουμε ένα στιγμιότυπο ενός τύπου αναφοράς ούτε να αντιγράψουμε ένα υπάρχουν στιγμιότυπο ή να εκτελέσουμε μια σύγκριση τιμών σε δύο υπαρκτά στιγμιότυπα αν και συγκεκριμένοι τύποι αναφοράς μπορεί να προσφέρουν τέτοιες λειτουργίες εκθέτοντας έναν public κατασκευαστή ή υλοποιώντας ένα αντίστοιχο interface (όπως είναι το Icloneable ή το Icomparable). Παραδείγματα τύπων αναφοράς είναι το object (η απόλυτη κλάση βάσης για όλες τις άλλες κλάσεις της C#), το System.String και το System.Array.

Και οι δύο κατηγορίες είναι επεκτάσιμες με τύπους ορισμένους από τον χρήστη.

3.3.11 Προτυποποίηση

Τον Αύγουστο του 2000 η Microsoft Corporation, η Hewlett-Packard και η Intel Corporation συνυποστήριξαν την υποβολή των προδιαγραφών της C# σαν την Common Language Infrastructure (CLI) στον οργανισμό προτύπων Ecma International. Το Δεέμβριο του 2001 η ECMA κυκλοφόρησε το ECMA-334 C# Language Specification. Η C# έγινε πρότυπο ISO το 2003

198 *Archer, Tom (2001). "Part 2, Chapter 4: The Type System". Inside C#. Redmond, Washington: Microsoft Press. ISBN 0-7356-1288-9.*

(ISO/IEC 23270:2003 Information Technology – Programming Languages – C#). Η ECMA προηγουμένως είχε υιοθετήσει τις αντίστοιχες προδιαγραφές σαν την δεύτερη έκδοση της C# τον Δεκέμβριο του 2002.
Τον Ιούνιο του 2005 η ECMA ενέκρινε την τρίτη έκδοση προδιαγραφών της C# και ενημέρωσε το ECMA – 334 οι προσθήκες αφορούσαν τις partial κλάσεις, τις ανώνυμες μεθόδους, του nullable τύπους και τα generics (κάτι αντίστοιχο με τα πρότυπα της C++).
Τον Ιούλιο του 2005 η ECMA υπέβαλλε το ISO/IEC JTC 1 μέσω του Fast-Track process, the standards and related Trs. Αυτή η διαδικασία συνήθως απαιτεί 6-9 μήνες.
Ο ορισμός της C# και το CLI έχουν προτυποποιηθεί και σαν ISO και σαν ECMA πρότυπα τα οποία παρέχουν λογική και μη μεροληπτική αδειοδότηση και προστασία από διεκδικήσεις πατέντας.
Η Microsoft συμφώνησε να μην μηνύσει τους προγραμματιστές ανοιχτού κώδικα για παραβιάσηη πατέντας σε μη κερδοσκοπικά έργα για το κομμάτι του πλαισίου εργασίας το οποίο καλύπτεται από το OSP[199]. Η Microsoft επίσης συμφώνησε να μην επιβάλλει πατέντες σχετικές με τα προϊόντα της Novell εναντίον πελατών[200] της Novell με εξαίρεση μια λίστας προϊόντων τα οποία δεν αναφέρουν ρητά την C#, το .NET ή την υλοποίηση της Novell του .NET (The Mono Project)[201]. Η Novell ωστόσο ισχυρίζεται ότι το Mono δεν παραβιάζει καμιά πατέντα της Microsoft[202]. Η Microsoft επίσης συμφώνησε να μην εγείρει ζητήματα παραβίασης πατέντας για τα δικαιώματα που σχετίζονται με το Moonlight plugin για τα προγράμματα περιήγησης στο Internet[203].

3.3.12 Υλοποίησεις

Ο μεταγλωττιστής αναφοράς της C# είναι ο Microsoft Visual C# ο οποίος είναι ανοιχτού κώδικα[204]. Η Microsoft ηγείται της ανάπτυξης ενός νέου C# μεταγλωττιστή ανοιχτού κώδικα και ενός

199 *"Patent Pledge for Open Source Developers"*.

200 *"Patent Cooperation Agreement - Microsoft & Novell Interoperability Collaboration". Microsoft. November 2, 2006. Retrieved July 5, 2009.* Microsoft, on behalf of itself and its Subsidiaries (collectively "Microsoft"), hereby covenants not to sue Novell's Customers and Novell's Subsidiaries' Customers for infringement under Covered Patents of Microsoft on account of such a Customer's use of specific copies of a Covered Product as distributed by Novell or its Subsidiaries (collectively "Novell") for which Novell has received Revenue (directly or indirectly) for such specific copies; provided the foregoing covenant is limited to use by such Customer (i) of such specific copies that are authorized by Novell in consideration for such Revenue, and (ii) within the scope authorized by Novell in consideration for such Revenue.

201 *"Definitions". Microsoft. November 2, 2006. Retrieved July 5, 2009.*

202 *Steinman, Justin (November 7, 2006). "Novell Answers Questions from the Community". Retrieved July 5, 2009.* We maintain that Mono does not infringe any Microsoft patents.

203 *"Covenant to Downstream Recipients of Moonlight - Microsoft & Novell Interoperability Collaboration". Microsoft. September 28, 2007. Retrieved March 8, 2008.* "Downstream Recipient" means an entity or individual that uses for its intended purpose a Moonlight Implementation obtained directly from Novell or through an Intermediate Recipient... Microsoft reserves the right to update (including discontinue) the foregoing covenant... "Moonlight Implementation" means only those specific portions of Moonlight 1.0 or Moonlight 1.1 that run only as a plug-in to a browser on a Personal Computer and are not licensed under GPLv3 or a Similar License.

204 https://github.com/dotnet/roslyn

συνολου εργαλείων τα οποία έως τώρα ήταν γνωστά ως Roslyn. Ο μεταγλωττιστής ο οποίος είναι αποκλειστικά γραμμένος σε managed code (C#) έχει ανοίξει και αναδύθηκε λειτουργικά ως APIs. Εμπλέκει με τον τρόπο αυτό τους προγραμματιστές να φτιάξουν διαγνωστικά εργαλεία και εργαλεία επαναπαραγοντοποίησης.
Άλλοι C# μεταγλωττιστές υπάρχουν και συχνά περιλαμβάνουν μια υλοποίηση της Common Language Infrastructure και των βιβλιοθηκών κλάσεων του .NET έως και το .NET2.0:

- Το Mono σχέδιο παρέχει ένα μεταγλωττιστή C# ανοιχτού κώδικα, μια πλήρη υλοποίηση ανοιχτού κώδικα της Common language Infrastructure που περιέχει τις απαιτούμενες βιβλιοθήκες πλαισίου εργασίας όπως αυτές εμφανίζονται στην τεκμηρίωση της ECMA και μια σχεδόν πλήρη υλοποίηση των βιβλιοθηκών κλάσεων του Microsoft .NET έως την έκδοση 3.5. Έως το Mono 2.6 δεν υπάρχουν σχέδια υλοποίησης του WPF. Το WF σχεδιάζεται να συμπεριληφθεί σε μεταγενέστερη έκδοση και υπάρχουν μόνο τμηματικές υλοποιήσεις των LINQ to SQL και του WCF[205].
- Το Rotor σχέδιο της Microsoft (προς το παρόν γνωστό ως Shared Source Common Language Infrastructure) παρέχει μια διαμοιραζόμενου κώδικα υλοποίηση του CLR και του C# μεταγλωττιστή και ένα υποσύνολο από τις βιβλιοθήκες του απαιτούμενου πλαισίου εργασίας της CLI όπως φαίνονται στην τεκμηρίωση της ECMA.

3.4 LabVIEW

3.4.1 Εισαγωγή

Το LabVIEW (σύντμηση του Laboratory Virtual Instrument Engineering Workbench) είναι μια πλατφόρμα σχεδιασμού συστημάτων και ένα περιβάλλον ανάπτυξης για μια visual γλώσσα προγραμματισμού από την National Instrumrents.
Η γραφική γλώσσα ονομάζεται G (δεν πρόκειται για G-code) αρχικά κυκλοφόρησε για τα Macintosh της Apple το 1986. Το LabVIEW χρησιμοποιείται συχνά για απόκτηση δεδομένων, έλεγχο οργάνων και βιομηχανικό αυτοματισμό σε μια πλειάδα συστημάτων μεταξύ των οποίων τα Windows της Microsoft, διάφορες εκδόσεις του UNIX, του Linux και του OS X. Η τελευταία έκδοση είναι η LabVIEW 2015 και κυκλοφορησε τον Αύγουστο του 2015.

3.4.2 Προγραμματισμός ροής δεδομένων

Η γλώσσα προγραμματισμού του LabVIEW γνωστή και ως G μια γλώσσα προγραμματισμού ροής δεδομένων. Η εκτέλεση καθορίζεται από την δομή ενός γραφικού διαγράμματος (ο πηγαίος κώδικας του LabVIEW) πάνω στο οποίο ο προγραμματιστής συνδέει διάφορους κόμβους συναρτήσεων ζωγραφίζοντας γραμμές. Αυτές οι γραμμές διαδίδουν μεταβλητές και κάθε κόμβος μπορεί να εκτελεστεί μόλις όλα τα δεδομένα εισόδου γίνουν διαθέσιμα. Μιας και κάτι τέτοιο

205 *"Compatibility - Mono". Mono-project.com. December 19, 2011. Retrieved October 4, 2012.*

μπορεί να απαιτήσει την ταυτόχρονη εκτέλεση πολλών κόμβων η G είναι κληρονομικά ικανή για παράλληλη εκτέλεση. Το πολυεπεξεργαστικό και το πολυνηματικό υλικό είναι αυτομάτων εκμεταλεύσιμο από τον ενσωματωμένο σχεδιαστή ο οποίος πολυπλέκει πολλαπλά νήματα του λειτουργικού συστήματος πάνω από τους κόμβους που είναι έτοιμοι για εκτέλεση.

3.4.3 Γραφικός προγραμματισμός

Το LabVIEW δεσμεύει την δημιουργία των διεπαφών των χρηστών (τα οποία λέγονται front panels) μέσα στον κύκλο ανάπτυξης του λογισμικού. Τα προγράμματα και οι υπορουτίνες του LabVIEW λέγονται εικονικά όργανα (Virtual Instruments – Vis). Κάθε VI έχει τρία συστατικά, ένα τμήμα σχηματικού διαγράμματος, ένα front panel κια ένα panel σύνδεσης. Το τελευταίο χρησιμοποιείται για την αναπαράσταση των VI στο διάγραμμα άλλων κληθέντων VIs. Το front panel ενσωματώνεται χρησιμοποιώντας ελέγχους και δείκτες. Οι έλεγχοι είναι είσοδοι, επιτρέπουν στον χρήστη την παροχή πληροφοριών στο VI. Οι δείκτες είναι έξοδοι οι οποίοι αναπαριστούν ή απεικονίζουν τα αποτελέσματα βασισμένοι στα δεδομένα εισόδου στο VI. Το back panel το οποίο είναι ένα σχηματικό διάγραμμα το οποίο περιέχει τον γραφικό πηγαίο κώδικα. Όλα τα αντικείμενα που τοποθετούνται στο front panel θα εμφανιστούν στο back panel ως τερματικά. Το back panel επίσης περιέχει δομές και συναρτήσεις οι οποίες εκτελούν λειτουργίες στους ελέγχους και παρέχουν δεδομένα στους δείκτες. Οι δομές και οι συναρτήσεις βρίσκονται στην παλέτα των συναρτήσεων και μπορούν να τοποθετηθούν στο back panel. Συγκεντρωτικά οι έλεγχοι, οι δείκτες, οι δομές και οι συναρτήσεις θα αναφέρονται ως κόμβοι. Οι κόμβοι συνδέονται μεταξύ τους με την χρήση γραμμών. Για παράδειγμα δύο έλεγχοι και ένας δείκτης μπορούν να συνδεθούν σε μια επιπλέον συνάρτηση έτσι ώστε ο δείκτης να απεικονίζει το άθροισμα των δύο ελέγχων. Έτσι ένα εικονικό όργανο μπορεί να εκτελεστεί είτε ως πρόγραμμα, με το front panel να υπηρετεί ως διεπαφή χρήστη, ή όταν πέφτει σαν κόμβος πάνω στο σχηματικό διάγραμμα, το front panel καθορίζει τις εισόδους και τις εξόδους για τον κόμβο μέσω του panel σύνδεσης. Αυτό υποδηλώνει ότι κάθε εικονικό όργανο μπορεί εύκολα να ελεγχθεί πριν την ενσωμάτωση του σαν υπορουτίνα σε ένα μεγαλύτερο πρόγραμμα.
Η γραφική προσέγγιση επίσης επιτρέπει στους μη προγραμματιστές να χτίσουν προγράμματα σέρνοντας και αφήνοντας εικονικές αναπαραστάσεις εξοπλισμού εργαστηρίου με τον οποίο είναι εξοικειωμένοι. Το περιβάλλον προγραμματισμού του LabVIEW με τα περιεχόμενα παραδείγματα και την τεκμηρίωση του κάνει την δημιουργία μικρών εφαρμογών μια σχετικλά εύκολη διαδικασία. Αυτό βεβαίως από μια μεριά αποτελεί κέρδος υπάρχει όμως ένας συγκεκριμένος κίνδυνος υποτίμησης της δεξιότητας που απαιτείται για τον προγραμματισμό υψηλής ποιότητας στην G. Για πολύπλοκους αλγορίθμους ή για κώδικα μεγάλης κλίμακας είναι απαραίτητο ο προγραμματιστής να έχει μια εκτενή επίγνωση του ειδικού συντακτικού του LabVIEW καθώς και τις ιδιαιτερότητες της διαχείρισης μνήμης. Τα περισσότερο εξελιγμένα LabVIEW συστήματα ανάπτυξης προσφέρουν την δυνατότητα δημιουργίας αυτόνομων εφαρμογών. Επιπλέον είναι δυνατή η δημιουργία κατανεμημένων εφαρμογών οι οποίες επικοινωνούν μέσω ενός σχήματος client/server και άρα είναι ευκολότερο να υλοποιηθούν λόγω της κληρονομικά παράλληλης φύσης της G.

3.4.4 Κέρδη

3.4.4.1 Διεπαφή με τις συσκευές

Το LabVIEW περιέχει εκτεταμένη υποστήριξη για διεπαφή με συσκευές, όργανα, κάμερες και άλλες συσκευές. Η διεπαφή των χρηστών με το υλικό είτε με την συγγραφή εντολών διαύλου (USB, GRIB, Serial) ή με την χρήση υψηλού επιπέδου και στοχευμένων για κάθε συσκευή προγραμμάτων οδηγών οι παρέχουν έμφυτους LabVIEW κόμβους συναρτήσεων για τον έλεγχο της συσκευής.
Το LabVIEW περιλαμβάνει ενσωματωμένη υποστήριξη για πλατφόρμες υλικού NI όπως είναι το CompactDAQ και Compact RIO με ένα μεγάλο αριθμό τμημάτων στοχευμένων σε συγκεκριμένες συσκευές για το υλικό αυτό[206].

3.4.4.2 Μεταγλώττιση κώδικα

Σε όρους απόδοσης το LabVIEW περιέχει ένα μεταγλωττιστή ο οποίος παράγει φυσικό κώδικα για τον επεξεργαστή της κάθε πλατφόρμας. Ο γραφικός κώδικας μεταφράζεται σε εκτελέσιμο κώδικα μηχανής μεταφράζοντας τη σύνταξη και με μεταγλώττιση. Το συνατκτικό του labVIEW είναι αυστηρά καθορισμένο κατά την διαδικασία συγγραφής του και μεταγλωττίζεται σε εκτελέσιμο κώδικα μηχανής όταν απαιτηθεί ή κατά την αποθήκευση.Στην τελευταία περίπτωση ο εκτελέσιμος και ο πηγαίος κώδικας συγχωνεύονται σε ένα μόνο αρχείο. Το εκτελέσιμο τρέχει με την βοήθεια της μηχανής χρόνου εκτέλεσης του LabVIEW η οποία περιέχει προμεταγλωττισμένο κώδικα για την εκτέλεση εργασιών που ορίζονται από την γλώσσα G. Η μηχανή χρόνου εκτέλεσης μειώνει τον χρόνο μεταγλώττισης και επιπλέον παρέχει ένα συνεπές σύστημα διεπαφής για διάφορα λειτουργικά συστήματα, γραφικά συστήματα και συστατικών του υλικού. Το περιβάλλον του χρόνου εκτέλεσης κάνει τον κώδικα μεταφέρσιμο από πλατφόρμα σε πλατφόρμα. Γενικά ο κώδικας του LabVIEW μπορεί να είναι αργότερος από τον αντίστοιχο κώδικα σε C.

3.4.4.3 Μεγάλες βιβλιοθήκες

Πολλές βιβλιοθήκες με μεγάλο αριθμό συναρτήσεων για ανάκτηση δεδομένων, δημιουργία σημάτων, μαθηματικά, στατιστικά, κατάστασης σημάτων, ανάλυσης και άλλα μαζί με πολλά συστατικά γραφικών συστημάτων διεπαφής παρέχονται από αρκετές επιλογές πακέτων του LabVIEW. Ο αριθμός των ανώτερων μαθηματικών τμημάτων για λειτουργίες όπως η ενσωμάτωση, τα φίλτρα και άλλες εξειδικευμένες ικανότητες οι οποίες συνήθως σχετίζονται με την συλλογή δεδομένων από αισθητήρες του υλικού είναι τεράστιος. Επιπροσθέτως το LabVIEW περιλαμβάνει ένα συστατικό προγραμματισμού σε κείμενο το οποίο λέγεται MathScript με επιπρόσθετη λειτουργικότητα για την ανάλυση και επεξεργασία σημάτων και για μαθηματικά. Το MathScript

206 NI Instrument Driver Network

μπορεί να ενσωματωθεί στο γραφικό προγραμματισμό με την χρήση των script κομβων κια χρησιμοποιεί συντακτικό το οποίο είναι σε γενικές γραμμές συμβατό με το MATLAB.

3.4.4.4 Παράλληλος προγραμματισμός

Το labVIEW είναι μια εκ φύσεως ταυτόχρονη γλώσσα και έτσι είναι πολύ απλός ο προγραμματισμός πολλαπλών εργασιών οι οποίες εκτελούνται παράλληλα μέσω πολυνηματικότητας. Αυτό επιτυγχάνεται εύκολα τραβώντας δύο ή περισσότερους παράλληλους βρόχους while. Αυτό είναι ένα μεγάλο κέρδος για τον έλεγχο του αυτοματισμού του συστήματος, όπου είναι κοινή πρακτική η εκτέλεση διαδικασιών σαν έλεγχος προτεραιότητας, καταγραφή δεδομένων και διεπαφή υλικού παράλληλα.

3.4.4.5 Οικοσύστημα

Λόγω της μακροζωίας και της αποδοχής του LabVIEW και της ικανότητας των χρηστών να επεκτείνουν την λειτουργικότητα ένα μεγάλο οικοσύστημα επιπροσθέτων από τρίτους παρόχους αναπτύχθηκε από την κοινότητα των χρηστών. Αυτό το οικοσύστημα είναι διαθέσιμο στο δίκτυο εργαλείων του LabVIEW το οποίο είναι μια αγορά τόσο για ελεύθερα όσο και για επί πληρωμή πρόσθετα του LabVIEW.

3.4.4.6 Κοινότητα χρηστών

Υπάρχει μια χαμηλού κόστους έκδοση του LabVIEW για μαθητές που στοχεύει τα εκπαιδευτικά ιδρύματα για μαθησιακούς σκοπούς. Υπάρχει επίσης μια ενεργή κοινότητα χρηστών του LabVIEW οι οποίοι επικοινωνούν μέσω ομάδων email και φόρουμς.

3.4.5 Κριτική

Το LabVIEW είναι ένα προϊόν της National Instruments. Σε αντίθεση με τις κοινές γλώσσες προγραμματισμού όπως η C ή η FORTRAN το LabVIEW δεν το διαχειρίζεται μια τρίτη επιτροπή προτύπων όπως είναι οι ANSI, IEEE, ISO και άλλες.

3.4.5.1 Προγραμματιστικό μοντέλο ροής δεδομένων

Λόγω της ευρείας υιοθέτησης του προγραμματιστικού μοντέλου ροής δεδομένων σε αντίθεση με την σειριακή παράθεση αυθαίρετων εντολών όπως οι περισσότερες άλλες γλώσσες υπάρχει ένα πραγματικό όριο για πολλούς ανθρώπους που επιχειρούν να εφαρμόσουν τις γνωστές τους πρακτικές και αρχές από τις άλλες προγραμματιστικές προσεγγίσεις στο LabVIEW. Η εκ φύσεως παράλληλη φύση εκτέλεσης του κώδικα του LabVIEW είναι μια αέναη πηγή σύγχυσης για όσους

είναι συνηθισμένοι σε άλλες προσεγγίσεις. Λόγω αυτού οι περισσότερες απόψεις τείνουν να είναι ιδιαιτέρως πολωμένες με τους χρήστες είτε να αποδέχονται φανατικά το στυλ είτε να το απορρίπτουν δίχως συζήτηση.

3.4.5.2 Αδειοδότηση

Το χτίσιμο μιας αυτόνομης εφαρμογής με το LabVIEW απαιτεί το συστατικό Application Builder το οποίο περιλαμβάνεται στην επαγγελματική έκδοση ανάπτυξης του LabVIEW αλλά απαιτείται ξεχωριστή αγορά αν έχει αγοραστεί μόνο το βασικό πακέτο[207].

3.4.5.3 Περιβάλλον χρόνου εκτέλεσης

Τα μεταγλωττισμένα εκτελέσιμα που παράγονται από την έκδοση 6.0 και μετά του Application Builder δεν είναι κυριολεκτικά αυτόνομα με την έννοια ότι απαιτούν τηνεγκατάσταση της μηχανής εκτελέσιμου χρόνου του LabVIEW στο μηχάνημα που πρόκειται να εκτελέσει την εφαρμογή[208]. Η χρήση στάνταρ ελέγχων απαιτεί μια βιβλιοθήκη χρόνου εκτέλεσης για κάθε γλώσσα. Όλα τα μεγάλα λειτουργικά συστήματα παρέχουν τις απαραίτητες βιβλιοθήκες για τις κοινές γλώσσες όπως είναι η C. Ωστόσο η βιβλιοθήκη χρόνου εκτέλεσης που απαιτείται για το LabVIEW δεν διατίθεται από κανένα λειτουργικό σύστημα και πρέπει επί τούτου να εγκατασταθεί από τον διαχειριστή ή τον χρήστη. Αυτό μπορεί να προκαλέσει προβλήματα εάν μια εφαρμογή διανεμηθεί σε χρήστη ο οποίος μπορεί να είναι έτοιμος για την εκτέλεση της αλλά δεν διαθέτει την απαραίτητη άδεια για την εγκατάσταση των επιπλέον αρχείων πριν την εκτέλεση του εκτελέσιμου.

3.4.5.4 Παράλληλη εκτέλεση και καταστάσεις ανταγωνισμού.

Η G γλώσσα περιέχει κατασκευές για την δημιουργία πολλαπλών νημάτων εκτέλεσης. Όπως με κάθε γλώσσα η οποία στοχεύει τα μη ντετερμινιστικά λειτουργικά συστήματα όπως είναι τα Windows, το Mac OS, και το Linux, η παράλληλη εκτέλεση πολλαπλών νημάτων μπορεί να οδηγήσεις σε καταστάσεις ανταγωνισμού. Αν και η G γλώσσα σε μεγάλο βαθμό απλοποιεί τόσο τον προγραμματισμό όσο και την διαχείριση των νημάτων σε πολυπύρηνα και πολυεπεξεργαστικά συστήματα ο προγραμματιστής της G πρέπει να προσέχει επιπλέον για την αποφυγή των καταστάσεων ανταγωνισμού, για τις οποίες υπάρχουν πολλές συναρτήσεις και τεχνικές ώστε να επιτευχθεί αυτό. Ο προγραμματισμός με το συστατικό του LabVIEW FPGA έχει ως αποτέλεσμα την πραγματική παράλληλη υλοποίηση των στόχων του FPGA.

3.4.5.5 Απόδοση

207 *"Building a Stand-Alone Application". National Instruments.*

208 *"Using the LabVIEW Run-Time Engine". National Instruments.*

Το LabVIEW παράγει σε γενικές γραμμές εφαρμογές οι οποίες είναι αργότερες από τις αντίστοιχες εφαρμογές των κλασικών γλωσσών όπως η C αν και η υψηλή απόδοση μπορεί να επιτευχθεί εάν χρησιμοποιηθούν πολυπύρηνα συστήματα ή αφοσιωμένες εργαλειοθήκες για συγκεκριμένες εργασίες. Το LabVIEW κάνει τον πολυπήρηνο προγραμματισμό πολύ απλούστερο από τον αντίστοιχο προγραμματισμό σε γλώσσες κειμένου λόγω της ρητής απλότητας και της αυτοματοποιημένης διαχείρισης των νημάτων[209].

3.4.5.6 Ελαφρές εφαρμογές

Πολύ μικρές εφαρμογές εξακολουθούν να χρειάζονται το περιβάλλον χρόνου εκτέλεσης το οποίο είναι μεγάλη και αργή εργασία. Αυτό τείνει να περιορίζει το LabVIEW σε μονολιθικές εφαρμογές. Παραδείγματα αυτού είναι μικρά προγράμματα τα οποία σκοπό έχουν να ανακτήσουν μια μόνο τιμή από κάποιο αισθητήρα υλικού τα οποία μπορεί να χρησιμοποιηθούν σε κάποια scripting γλώσσα η οποία προπορεύεται του περιβάλλοντος χρόνου εκτέλεσης κάνοντας έτσι την προσέγγιση του LabVIEW μη πρακτική.

3.4.5.7 Σύστημα συγχρονισμού

Το LabVIEW χρησιμοποιεί την 1 Ιανουαρίου του 1904 σαν τον χρόνο μηδέν. Άλλα προγράμματα που χρησιμοποιούν τον ίδιο χρόνο είναι το MacOs έως την έκδοση 9, το PalmOs και το Microsoft Excel[210].

3.4.5.8 Χωρίς κείμενο

Με την G γλώσσα να είναι ένα μια γλώσσα στην οποία ο προγραμματισμός δεν γίνεται με την πληκτρολόγηση κειμένου τα λογισμικά εργαλεία όπως αυτό των εκδόσεων που στηρίζεται στην σύγκριση των διαφορών και η παρακολούθηση των αλλαγών μεταξύ των εκδόσεων του κώδικα δεν μπορεί να γίνει με τον ίδιο τρόπο που γίνεται με τις κλασσικές γλώσσες.

3.4.5.9 Έλλειψη προς τα πίσω συμβατότητας

Ένα εικονικό εργαλείο σε μια καινούρια έκδοση του LabVIEW δεν μπορεί να ανοιχτεί από μια παλαιότερη έκδοση ούτε καν για επισκόπηση[211]. Το χαρακτηριστικό Save for Previous Version μπορεί εν μέρη να αποτελέσει λύση εφόσον ο προγραμματιστής γνωρίζει τις μελλοντικές του απαιτήσεις.

---

209 Why Dataflow Programming Languages are Ideal for Programming Parallel Hardware

210 *Spolsky, Joel. "Why are the Microsoft Office file formats so complicated? (And some workarounds)". Retrieved March 8, 2009.*

211 *"Can I Save VIs in My Current LabVIEW Version for Use in a Previous Version?". Retrieved February 1, 2016.*

3.4.5.10 Έλλειψη λειτουργίας μεγέθυνσης

Δεν υπάρχει η δυνατότητα της μεγέθυνσης ενός εικονικού εργαλείου το οποίο θα είναι δύσκολα ορατό σε μια οθόνη υψηλής ανάλυσης αν και το εν λόγω χαρακτηριστικό είναι υπό εξέλιξη[212][213].

3.4.6 Ιστορικό εκδόσεων

Ξεκινώντας με το LabVIEW 8.0 οι κύριες εκδόσεις κυκλοφορούν την πρώτη εβδομάδα του Αυγούστου για να συμπέσουν με το ετήσιο συμπόσιο της National Instruments NI Week και ακολουθείται από μια έκδοση διόρθωσης των προβλημάτων τον επόμενο Φεβρουάριο. Το 2009 η National Instruments ξεκίνησε να ονομάζει τις εκδόσεις με βάση το έτος κυκλοφορίας. Η έκδοση διόρθωσης προβλημάτων καλείται Service Pack.

3.4.7 Repositories και βιβλιοθήκες

Η OpenG όπως και το LAVA Code Repository (LAVAcr) χρησιμοποιούνται σαν repositories για μια μεγάλη γκάμα εφαρμογών ανοιχτού κώδικα και βιβλιοθηκών για το LabVIEW. Το SourceForge έχει το LabVIEW καταχωρημένο σαν μια από τις πιθανές γλώσσες στις οποίες μπορεί να γραφεί κώδικας.
Ο διαχειριστής πακέτων των εικονικών οργάνων έχει προαχθεί στον στάνταρ διαχειριστή πακέτων για τις βιβλιοθήκες του LabVIEW. Είναι παρόμοιος σε σκοπό με το RubyGem της Ruby και το CPAN της Perl, αν και παρέχει ένα γραφικό περιβάλλον διεπαφής παρόμοιο με τον διαχειριστή πακέτων του Synaptic. Ο διαχειριστής πακέτων εικονικών οργάνων παρέχει πρόσβαση σε έαν repository των βιβλιοθηκών του OpenG για το LabVIEW.
Επίσης υπάρχουν εργαλεία για την μετατροπή της MathML σε G code[214].

3.4.8 Σχετικό λογισμικό

Η National Instruments επίσης παρέχει ένα προϊόν με το όνομα Measurement Studio το οποίο παρέχει πολλούς από τους ελέγχους, τις μετρήσεις και τις δυνατότητες ελέγχου του labVIEW σαν ένα σύνολο κλάσεων για χρήση με το Microsoft Visual Studio. Αυτό επιτρέπει στους προγραμματιστές να εκμεταλλευτούν ένα κομμάτι της δύναμης του LabVIEW μέσα στο κειμενοκεντρικό .NET Framework. Η National Instruments επίσης προσφέρει το LabWindows/CVI σαν μια εναλλακτική επιλογή για τους προγραμματιστές C.

---

212 *"Can I Zoom In or Out on a LabVIEW Diagram (for Wiring or Viewing Purposes)?". Retrieved February 1, 2016.*

213 *"Add a zoom function (yes, I said zoom. So sue me)". forums.ni.com. Retrieved 2016-03-31.*

214 https://decibel.ni.com/content/docs/DOC-13859

Όταν οι εφαρμογές απαιτούν αλληλουχία οι χρήστες συχνά χρησιμοποιούν το LabVIEW με το TestStand επίσης από την National Instruments.
Ο μεταφαστής Ch είναι ένας μεταφραστής C/C++ ο οποίος μπορεί να ενσωματωθεί στο labVIEW για scripting[215].
Η πλατφόρμα TRIL (Centre Ireland BioMobius) και το FlowStone της DSP Robotics επίσης χρησιμοποιούν μια μορφή γραφικού προγραμματισμού παρόμοια με αυτή του LabVIEW αλλά είναι εργαλεία περιορισμένα στην βιομηχανία της βιοιατρικής και στην ρομποτικής.
Το labVIEW διαθέτει ένα άμεσο κόμβο με modeFRONTIER, πρόκειτε για ένα πολυπειθαρχικό και πουαντικειμενικό περιβάλλον βελτιστοποίησης και σχεδιασμού γραμμένο ώστε να επιτρέπει το ταίριασμα σχεδόν οποιονδήποτε υποβοηθούμενων από υπολογιστή εργαλείων μηχανικής. Και τα δύο μπορούν να είναι μέρη της ίδιας διαδικασίας περιγραφής της ροής της εργασίας και μπορεί να είναι εικονικά κατευθυνόμενα από τις τεχνολογίες βελτιστοποίησης που είναι διαθέσιμες στο modeFRONTIER.

3.5 Η γλώσσα προγραμματισμού Pascal

3.5.1 Εισαγωγή

Εικόνα 12: Niklaus Wirth Ο δημιουργός της Pascal

Η Pascal είναι μια επιτακτική και διαδικασιακή γλώσσα προγραμματισμού. Η οποία σχεδιάστηκε κατά τα έτη 1968-69 και δόθηκε στην δημοσιότητα το 1970 από τον Niklasus Wirth σαν μια μικρή και αποτελεσματική γλώσσα με σκοπό την ενθάρρυνση της χρήσης σωστών προγραμματιστικών πρακτικών, του διαδικασιακού προγραμματισμού και των δομών δεδομένων.
Ένα παράγωγο της Pascal γνωστό με το όνομα Object Pascal σχεδιάστηκε για τον αντικειμενοστραφή προγραμματισμό το 1985.

3.5.2 Ιστορία

Η Pascal πήρε το όνομα της προς τιμήν του Γάλλου μαθηματικού και φιλοσόφου Blaise Pascal και αναπτύχθηκε από τον Nik;laus Wirth.
Πριν από την εργασία του πάνω στην Pascal ο Wirth ανέπτυξε την Euler και την ALGOL W και αργότερα ανέπτυξε τις όμοιες με την Pascal γλώσσες Modula-2 και Oberon.
Αρχικά η Pascal αποσκοπούσε κυρίως αλλά όχι αποκλειστικά στην διδασκαλία του δομημένου προγραμματισμού στους φοιτητές[216]. Μια ολόκληρη γενιά φοιτητών χρησιμοποίησε την Pascal σαν γλώσσα εισαγωγής στον προγραμματισμό στα προπτυχιακά μαθήματα. Εναλλακτικές εκδόσεις της Pascal αρκετά συχνά χρησιμοποιήθηκαν σχεδόν για τα πάντα, από ερευνητικά έργα έως παιχνίδια

215 Embedding a C/C++ Interpreter Ch into LabVIEW for Scripting

216 Essential Pascal by Marco Cantù

υπολογιστών και προγραμματισμό ενσωματωμένων συστημάτων. Οι χρήση νεότερων μεταγλωττιστών της Pascal είναι αρκετά διαδεδομένη[217].

Η Pascal ήταν η κύρια γλώσσα ανάπτυξης του Apple Lisa καθώς και για τα πρώτα χρόνια των macintosh. Μέρη του αρχικού λειτουργικού συστήματος των Macintosh μεταφράστηκαν σε γλώσσα assembly Morola 6800 από τα πηγαί αρχεία σε Pascal[218]. Το σύστημα στοιχειοθεσίας Tex από τον Donald E. Knuth γράφτηκε σε WEB και το αρχικό literate programming σύστημα βασίστηκε στην DECPDP-10 Pascal, ενώ εφαρμογές όπως το Total Commander το Skype, και το Macromedia Captivate γράφτηκαν σε Delphi (Object Pascal).

Η Object Pascal (Embarcadero Delphi) χρησιμοποιήθηκε μέχρι πρόσφατα στην ανάπτυξη εφαρμογών για Windows αλλά έχει επίσης την δυνατότητα της διαπλατφορμικής μεταγλώττισης σε Mac, iOS και Android. Μια άλλη διαπλατφορμική έκδοση η Free Pascal μαζί με το IDE Lazarus είναι αρκετά δημοφιλής στους χρήστες του Linux μιας και προσφέρει την δυνατότητα γράψε μια φορά μεταγλώττισε οπουδήποτε. Το Code Typhon είναι μια έκδοση του Lazarus με περισσότερα προ εγκατεστημένα τμήματα και διαμεταγλωττιστές.

3.5.3 Σύντομη Περιγραφή

Η πρόθεση του Wirth ήταν η δημιουργία μιας αποτελεσματικής γλώσσας (αναφορικά τόσο με το χρόνο μεταγλώττισης όσο και με τον παραγόμενο κώδικα) βασισμένη στο πρότυπο του δομημένου προγραμμτισμού, μιας έννοιας η οποία στα τέλη του 60 ήταν ακόμη καινούρια και την εισηγούταν (ο Wirth) στο βιβλίο του Algorithms + Data Structures = Programms. Η Pascal έχει τις ρίζες της στην ALGOL 60 και εισηγήθηκε έννοιες και μηχανισμούς οι οποίοι (βασισμένοι στις κλίμακες και στους πίνακες της ALGOL) επέτρεπαν στου προγραμματιστές να ορίσουν τους δικούς τους σύνθετους (δομημένους) τύπους δεδομένων και διευκόλυνε την υλοποίηση δυναμικών και αναδρομικών δομών όπως οι λίστες, τα δέντρα και τα γραφήματα. Σημαντικά χαρακτηριστικά για την υλοποίηση όλων αυτών ήταν τα records, enumerations, subranges, dynamically allocated variables με δείκτες και τα sets. Για να κάνει όλα αυτά η Pascal εφικτά και με νόημα, είχε ένα δυνατό σύστημα γραφής για όλα τα αντικείμενα, το οποίο σημαίνει ότι ένας τύπος δεδομένων δεν μπορεί να μετατραπεί ή να ερμηνευτεί σαν κάποιος άλλος παρά μόνο αν εκτελεστούν συγκεκριμένες κινήσεις μετατροπής του. Παρόμοιοι μηχανισμοί είναι κοινός τόπος σε πολλές γλώσσες προγραμματισμού σήμερα. Άλλες γλώσσες οι οποίες επηρέασαν την ανάπτυξη της Pascal ήταν οι Simula 67 και έκδοση της ALGOL W του Wirth.

Η Pascal όπως πολλές σύγχρονες γλώσσες προγραμματισμού (όχι όμως όπως οι γλώσσες της οικογένειας της C) επιτρέπει ένθετους ορισμούς συναρτήσεων σε κάθε βαθμό βάθους και επίσης επιτρέπει σχεδόν όλες τις δηλώσεις στο εσωτερικό των συναρτήσεων. Αυτό το χαρακτηριστικό επιτρέπει τη χρήση ενός απλού και συναφούς συντακτικού στο οποίο ένα ολόκληρο πρόγραμμα είναι συντακτικά σχεδόν ίδιο με μια απλά διεργασία ή συνάρτηση (εξαιρουμένης της κεφαλίδας η

217 tiobe.com, Programming Community Index for January 2011.

218 Hertzfeld, Andy. "Hungarian folklore.org: Macintosh Stories. Retrieved 2012-03-06.

οποία περιέχει μια από τις δεσμευμένες λέξεις program, function, procedure).

3.5.4 Υλοποιήσεις

3.5.4.1 Πρώιμοι μεταγλωττιστές της Pascal

Ο πρώτος μεταγλωττιστής Pascal σχεδιάστηκε στην Ζυρίχη για την σειρά mainframe CDC 6000. Ο Wirth αναφέρει ότι η πρώτη του απόπειρα να το υλοποιήσει σε Fortran το 1969 δεν ήταν επιτυχής λόγω της αδυναμίας της Fortran να εκφράσει σύνθετες δομές δεδομένων. Η δεύτερη προσπάθεια απέδωσε την ίδια την Pascal και ήταν επιχειρησιακή στα μισά του 1970. Πολλοί μεταγλωττιστές Pascal έκτοτε είναι αυτόφιλοξενούμενοι δηλαδή ο ίδιος ο μεταγλωττιστής είναι γραμμένος σε Pascal και ο μεταγλωττιστής είναι σε θέση να επαναμεταγλωττίσει τον εαυτό του όταν νέες δυνατότητες προστίθονται στην γλώσσα ή όταν ο μεταγλωττιστής μεταφέρεται σε ένα νέο περιβάλλον. Ο GNU Pascal μεταγλωττιστής είναι μια χαρακτηριστική εξαίρεση μιας και είναι γραμμένος σε C.
Η πρώτη επιτυχής μεταφορά του μεταγλωττιστή CDC Pascal σε άλλο mainframe ολοκληρώθηκε από τους Welsh και Quimm στο Quins University of Belfast (QUB) το 1972. Ο στόχος ήταν η σειρά ICL 1900. Αυτός ο μεταγλωττιστής με την σειρά του αποτέλεσε τον πρόγονο του μεταγλωττιστή Pascal για το Information Computer Systems (ICS) Multum Minicomputer. Το σύστημα του Multum αναπτύχθηκε θεωρώντας την Pascal ως γλώσσα ανάπτυξης συστημάτων από τους Findlay, Cupples, Cavouras και Davis οι οποίοι εργάζονταν στον Department of Computing Science στο Glasgow University. Θεωρείται ότι η Multum Pascal η οποία ολοκληρώθηκε το καλοκαίρι του 1973 ίσως να είναι η πρώτη 16bit υλοποίηση.
Ένας εντελώς νέος μεταγλωττιστής ολοκληρώθηκε από τον Welsh στο QUB το 1977. Προσέφερε ένα διαγνωστικό πηγαίου κώδικα το οποίο είχε αναπτυχθεί από τους Findlay και Watt στο Glasgow University. Αυτή η υλοποίηση είσηχθηκε στο ICL 2900 το 1980 από μια ομάδα με βάση τα πανεπιστήμια του Southampton και Glasgow. Η Standard Pascal Model Implementation επίσης βασίστηκε σε αυτόν τον μεταγλωττιστή και υιοθετήθηκε από τους Welsh και Hay στο πανεπιστήμιο Manchester το 1984 για τον έλεγχο της αυστηρότητας της συμμόρφωσης στο BSI 6195/ISO 71856 πρότυπο και για να παράξει κώδικα για μια φορητή αφαιρετική μηχανή.
Ο πρώτος μεταγλωττιστής Pascal που γράφτηκε στην Βόρεια Αμερική κατασκευάστηκε στο πανεπιστήμιο του Illinois υπό την επίβλεψη του Donald B. Gillies για το PDP-11 και παρήγαγε φυσικό κώδικα μηχανής.

3.5.4.2 Το Pascal-P σύστημα

Για την γρήγορη διάδοση της γλώσσας δημιουργήθηκε ένα κιτ εισαγωγής, στη Ζυρίχη, το οποίο περιελάμβανε έναν μεταγλωττιστή ο οποίος παρήγαγε κώδικα για μια εικονική μηχανή στοίβα, κώδικα ο οποίος δάνειζε τον εαυτό του για μια αποτελεσματική ερμηνεία μαζί με έναν διερμηνέα για τον κώδικα αυτό. Το σύστημα αυτό ονομάστηκε Pascal – P σύστημα. Οι μεταγλωττιστές του P συστήματος ονομάστηκαν Pascal-P1, Pascal-P2, Pascal-P3 και Pascal-P4. Με τον Pascal-P1 να

είναι η πρώτη έκδοση και η Pascal-P4 να είναι η τελευταία προερχόμενη από την Ζυρίχη. Η έκδοση Pascal-P1 επινοήθηκε λόγω του γεγονότος ότι υπήρχαν πολλές διαφορετικές πηγές για το Pascal-P. Ο μεταγλωττιστής επανασχεδιάστηκε για να ενισχύσει την μεταφερσιμότητα και εκδόθηκε με την ονομασία Pascal-P2. Αυτός ο κώδικας αργότερα ενισχύθηκε και προέκυψε το Pascal-P3 με ενδιάμεσο κώδικα συμβατό προς τα πίσω με την Pascal-P2 και η Pascal-P4 η οποία δεν ήταν συμβατή προς τα πίσω.
Το Pascal-P4 σύστημα μεταγλωττιστή/μεταφραστή μπορεί ακόμη να μεταγλωττιστεί σε συστήματα συμβατά με την πρωτότυπη Pascal. Ωστόσο αποδέχεται ένα μόνο υποσύνολο της γλώσσας Pascal.
Το Pascal-P5 σχεδιάστηκε εκτός της ομάδας της Ζυρίχης και δέχεται το πλήρες πακέτο της γλώσσας Pascal και είναι συμβατός με το πρότυπο ISO 7185.
Η UCSD Pascal είναι ένα παρακλάδι του Pascal-P2, το οποίο ο Kenneth Bowles χρησιμοποίησε για να δημιουργήσει το μεταφραστικό UCSD p-System. Στις αρχές της δεκαετίας του 1980 η UCSD Pascal εισήχθη στα Apple II και Apple III συστήματα για να παράσχει μια δομημένη εναλλακτική στους μεταφραστές BASIC οι οποίοι έρχονταν ενσωματωμένοι με τον Apple II και οι οποίοι ήταν αρχικά τα εξ' ορισμού προγραμματιστικά περιβάλλοντα για τον Apple III. Το UCSD p-System ήταν ένα από τα τρία λειτουργικά συστήματα που ήταν διαθέσιμα κατά το ξεκίνημα του IBM PC[219]. Η UCSD Pascsal χρησιμοποιούσε έναν ενδιάμεσο κώδικα βασισμένο σε byte values και έτσι έναν από τους πρώτους byte code μεταγλωττιστές.
Ένας μεταγλωττιστής βασισμένος στον μεταγλωττιστή Pascal-P4 ο οποίος δημιουργούσε μητρικά εκτελέσιμα, δόθηκε στην κατανάλωση για το IBM System/370 mainframe computer από την Australian Atomic Energy Commission, το έλεγαν AAEC Pascal Compiler από το όνομα της επιτροπής[220].
Στης αρχές της δεκαετίας του 1980 η Watcom Pascal αναπτύχθηκε για τον IBM System 370.

3.5.4.3 Object και Turbo Pascal

Η Apple Computers δημιούργησε το δικό της Lisa Pascal για το Lisa Workshop το 1982 και μετέφερε τον μεταγλωττιστή στο Apple Macintosh και MPW το 1985. Το 1985 ο Larry Tesler κατόπιν διαβούλευσης με τον Niklaus Wirth όρισε την Object Pascal και οι επεκτάσεις αυτές ενσωματώθηκαν τόσο στους Lisa Pascal όσο και στους Mac Pascal μεταγλωττιστές.
Στη δεκαετία του 1980 ο Anders Hejlsberg έγραψε τον Blue Label Pascal μεταγλωττιστή για το Nascom-2. Μια επαναυλοποίηση του εν λόγω μεταγλωττιστή για το IBM PC ονομάστηκε Compas Pascal και PolyPascal πριν αποκτηθεί από την Borland, οπότε και ονομάστηκε Turbo Pascal. Έγινε ιδιαιτέρως δημοφιλής άχρη σε μια επιθετική στρατηγική τιμολόγησης αφενός και αφετέρου λόγω του πρώτου περιβάλλοντος ανάπτυξης πλήρους οθόνης καθώς επίσης και λόγω του ελάχιστου χρόνου (λίγα μόλις δευτερόλεπτα) μεταγλώττισης, σύνδεσης και εκτέλεσης. Επιπλέον ήταν

219 "An Interview with JOHN BRACKETT AND DOUG ROSS", p15, Charles Babbage Institute, 2004

220 "AUSTRALIAN ATOMIC ENERGY COMMISSION RESEARCH ESTABLISHMENT, LUCAS HEIGHTS, NUCLEAR SCIENCE AND TECHNOLOGY BRANCH REPORT 1977, DIVISIONAL RESEARCH", p.22, International Atomic Energy Agency (IEAE)

γραμμένος σε assembly και ιδιαιτέρως βελτιστοποιημένος. Έτσι ήταν μικρότερος και γρηγορότερος από τα ανταγωνιστικά προϊόντα. Το 1986 ο Anders εισήγαγε την Turbo Pascal στους Macintosh και ενσωμάτωσε τις επεκτάσεις της Object Pascal της Apple στην Turbo Pascal. Αυτές οι επεκτάσεις εν συνεχεία προστέθηκαν στην PC έκδοση της Turbo Pascal στην έκδοση 5.5. Την ίδια περίοδο η Microsoft επίσης υλοποίησε τον μεταγλωττιστή της Object Pascal[221][222]. Ο Turbo Pascal 5.5 είχε τεράστια επιρροή στην κοινότητα της Pascal η οποία κατά τα τέλη της δεκαετίας του 1980 άρχισε να στρέφεται κυρίως προς τον IBM PC. Πολλοί χομπίστες του PC στην αναζήτηση τους για μια γλώσσα δομημένη σε αντικατάσταση της BASIC χρησιμοποίησαν αυτό το προϊόν. Επίσης άρχισε να υιοθετείται και από επαγγελματίες προγραμματιστές. Την ίδια περίοδο ένα σύνολο εννοιών εισήχθη από την C ώστε να επιτρέψει στους προγραμματιστές Pascal να χρησιμοποιήσουν το βασισμένο στη C API των Microsoft Windows απευθείας. Αυτές οι επεκτάσεις συμπεριλάμβαναν συμβολοσειρές που τερματίζονται με null αριθμητικούς δείκτες, δείκτες συναρτήσεων και μη ασφαλείς μετατροπές τύπων.
Ωστόσο η Borland αργότερα αποφάσισε ότι ήθελε περισσότερα αντικειμενοστραφή χαρακτηριστικά και ξεκίνησε από την αρχή με την Delphi χρησιμοποιώντας το σχέδιο προτύπου της Object Pascal που προτάθηκε από την Apple ως βάση. (Το συγκεκριμένο σχέδιο ακόμη δεν έχει επισήμως προτυποποιηθεί). Η Delphi ήταν ένα πλήρες IDE και ένα περιεκτικό σύνολο από προ-κωδικοποιημένα προσαρμόσιμα αντικείμενα (τα περισσότερα από τα οποία ήταν χαρακτηριστικά της οθόνης των Windows) γνωστό και ως Βιβλιοθήκη Visual Συστατικών (Visual Component Library – VCL). Η γλώσσα του υποβάθρου ήταν η Object Pascal. Οι κύριες προσθήκες σε σχέση με τις παλαιότερες OOP επεκτάσεις, ήταν ένα αντικειμενικό μοντέλο βασισμένο στην αναφορά, στους εικονικούς κατασκευαστές και καταστροφείς και στις ιδιότητες. Πολλοί άλλοι μεταγλωττιστές επίσης υλοποίησαν αυτή την διάλεκτο.
Η Turbo Pascal και τα παράγωγα της με θεωρούνται ως αρθρωτές γλώσσες. Ωστόσο δεν παρέχουν την δυνατότητα της ένθετης έννοιας ή την εισαγωγή και εξαγωγή συγκεκριμένων συμβόλων.

3.5.4.4 Άλλες παραλλαγές

Η Super Pascal ήταν μια παραλλαγή η οποία πρόσθεσε μη αριθμητικές ετικέτες, την δήλωση return και εκφράσεις σαν ονόματα τύπων.
Τα πανεπιστήμια του Wisconsin-Masison, της Zurich, της karlsruhe και του Wupetal ανέπτυξαν τους Pascal-SC[223] και Pascal-XSC[224][225][226] μεταγλωττιστές στοχεύοντας στον προγραμματισμό αριθμητικών υπολογισμών. Η TMT Pascal ήταν ο πρώτος Borland-συμβατός μεταγλωττιστής για

221 Jon Udell, Crash of the Object-Oriented Pascals, BYTE, July, 1989.

222 M.I.Trofimov, The End of Pascal?, BYTE, March, 1990, p.36.

223 doi:10.1016/0898-1221(87)90181-7

224 PASCAL-XSC: PASCAL for Extended Scientific Computing

225 *"XSC Software". Retrieved 11 August 2015.*

226 *"Universitaet Wuppertal: Wissenschaftliches Rechnen / Softwaretechnologie". Retrieved 11 August 2015.*

32-μπιτη MS-DOS προστατευμένη κατάσταση, OS/2, και Win32 λειτουργικά συστήματα. Επίσης ο TMT μεταγλωττιστής ήταν ο πρώτος που επέτρεψε την υπερφόρτωση μεταβλητών και συναρτήσεων. Ο Pascal-SC αρχικά είχε ως στόχο τον επεξεργαστή Z80 αλλά αργότερα ξαναγράφτηκε για το DOS(x86) και το 68000. Ο Pascal-XSC έχει κατά καιρούς εισαχθεί στα Unix (Linux, SunOS, HP-UX, AIX) και στο Microsoft/IBM (MS-DOS με EMX, OS/2, Windows) λειτουργικά συστήματα. Λειτουργεί παράγοντας ενδιάμεσο κώδικα C ο οποίος στην συνέχεια μεταγλωττίζεται σε φυσικό εκτελέσιμο κώδικα. Μερικές από τις επεκτάσεις της Pascal-SC υιοθετήθηκαν από την GNU Pascal.
Η Pascal Sol σχεδιάστηκε το 1983 από μια Γαλλική ομάδα για την υλοποίηση ενός βασισμένου στο Unix σύστημα με το όνομα Sol. Επρόκειτο για στάνταρ Pascal επιπέδου-1 (με παραμετροποιημένα όρια πινάκων) αλλά ο ορισμός επέτρεπε εναλλακτικές δεσμευμένες λέξεις και από πριν ορισμένα αναγνωριστικά στα Γαλλικά καθώς και κάποιες επεκτάσεις για την διευκόλυνση του προγραμματισμού συστημάτων (για παράδειγμα ένα ισότιμο του lseek)[227]. Η ομάδα Sol αργότερα προχώρησε στο ChorusOS έργο για τον σχεδιασμό ενός κατανεμειμένου λειτουργικού συστήματος[228].
Η IP Pascal ήταν μια υλοποίηση της Pascal χρησιμοποιώντας το Micropolis DOS αλλά ταχέως μετακινήθηκε στο CP/M τρέχοντας στον Z80. Το 1994 προχώρησε στα μηχανήματα τύπου 80386 και υπάρχει ακόμα και σήμερα στις υλοποιήσεις των Windows/XP και Linux. Το 2008 το σύστημα αναβαθμίστηκε σε ένα εντελώς νέο επίπεδο με αποτέλεσμα την γλώσσα με το όνομα Pascaline (όπως ο υπολογιστής του Pascal). Περιέχει αντικείμενα, ελέγχους ονομάτων, δυναμικούς πίνακες μαζί με πολλές άλλες επεκτάσεις και θα λέγαμε γενικά ότι την χαρακτηρίζει η ίδια λειτουργικότητα και προστασία τύπων με την C#. Είναι η μοναδική τέτοιου είδους υλοποίηση η οποία είναι επίσης συμβατή με τις αρχικές υλοποιήσεις της Pascal, η οποίες είναι γνωστές ως πρότυπο ISO 7185.
Το Smart Mobile Studio δημιουργήθηκε από τον Jon Aasenden και μεταγλωττίζει την διάλεκτο του της Object Pascal σε HTML5/Javascript.
Το Smart Mobile Studio περιλαμβάνει ένα IDE το οποίο περιέχει ένα σύνολο γραφικών χαρακτηριστικών και η γλώσσα του είναι ασυνήθιστη με την έννοια ότι εμπερικλείει επεκτάσεις για την Javascript.

3.5.5 Κατασκευές της γλώσσας

Η Pascal στην αρχική της μορφή είναι μια αμιγώς διαδικαστική γλώσσα και περιέχει τον παραδοσιακό πίνακα των δομών ελέγχου με βάση την ALGOL με δεσμευμένες λέξεις όπως if, then, else, while, for κτλ. Ωστόσο η Pascal διαθέται πολλές εγκαταστάσεις δόμησης δεδομένων και άλλες αφαιρέσεις η οποίες δεν εμπεριέχονταν στην αρχική ALGOL 60 όπως οι ορισμοί τύπων, τα records, οι δείκτες, τα enumerations και τα sets. Τέτοιες κατασκευές εν μέρη κληρονομήθηκαν από την Simula 67, ALGOL 68, ALGOL W του Niklaus Wirth και κάποιες προτάσεις από το C.A.R. Hoare.

227 Michel Gien, "The SOL Operating System", in Usenix Summer '83 Conference, Toronto, ON, (July 1983), pp. 75-78

228 cs.berkeley.edu

Τα προγράμματα σε Pascal ξεκινούν με την δεσμευμένη λέξη program και με μια λίστα εξωτερικών περιγραφών αρχείων ως παραμέτρους[229] (δεν απαιτείται στην Turbo Pascal). Έπεται το κυρίως σώμα κώδικα το οποίο περικλείεται μεταξύ των λέξεων begin και end. Τα ερωτηματικά χωρίζουν τις δηλώσεις και η τελεία τερματίζει όλο το πρόγραμμα. Πεζά και κεφαλαία δεν λαμβάνονται υπόψιν.

Ακολουθεί ένα παράδειγμα πηγαίου κώδικα για το HelloWorld πρόγραμμα.

```
Program HelloWorld(output);
Begin
  Write('Hello, world!')
  {no ";" is required after the last statement of a block -
   adding one adds a "null statement" to the program;}
End.
```

3.5.6 Τύποι δεδομένων

Ένας τύπος στην Pascal και σε άλλες γλώσσες προγραμματισμού άλλωστε ορίζει μια μεταβλητή με τέτοιο τρόπο ώστε να ορίζεται πλήρως το εύρος των τιμών που μπορεί να αποθηκεύσει η μεταβλητή αυτή και επίσης προσδιορίζει και τις επιτρεπόμενες διαδικασίες που μπορούν να εφαρμοστούν στις μεταβλητές αυτού του τύπου. Οι προορισμένοι τύποι είναι:

integer: ακέραιοι αριθμοί

real: αριθμοί κινητής υποδιαστολής

boolean: με τιμές true ή false

char: ένας μόνο χαρακτήρας από ένα συγκεκριμένο σύνολο χαρακτήρων

string: ένα σύνολο χαρακτήρων

Το εύρος των επιτρεπτών τιμών για κάθε τύπο (εκτός του boolean) είναι ορισμένο από την υλοποίηση του μεταγλωττιστή. Παρέχονται συναρτήσεις για κάποιες μετατροπές τύπων. Για την μετατροπή real σε integer οι κάτωθι συναρτήσεις είναι διαθέσιμες: round (η οποία στρογγυλοποιεί τον αριθμό στον ακέραι κατά banker) και trunc (η οποία στρογγυλοποιεί προς το μηδέν).

Ο προγραμματιστής έχει την ελευθερία να ορίσει και άλλους τύπους συχνά χρησιμοποιούμενους (όπως byte, string κλπ) με όρους των προορισμένων τύπων χρησιμοποιώντας την δομή δήλωσης τύπων της Pascal. Για παράδειγμα:

```
type
  byte       = 0..255;
  signed_byte = -128..127;
  string     = packed array[1..255] of char;
```

Συχνά οι τύποι όπως τα byte και τα string είναι εξαρχής ορισμένοι σε κάποιες υλοποιήσεις.

3.5.7 Τύποι αρχείων

229 Pascal ISO 7185:1990 6.10

Τα αρχεία στην Pascal είναι ακολουθίες συστατικών. Κάθε αρχείο έχει μια μεταβλητή ρυθμιστή η οποία υποδηλώνεται με f^. Οι διαδικασίες get (για ανάγνωση) και put (για γράψιμο) μετακινούν την μεταβλητή ρυθμιστή στο επόμενο στοιχείο. Η read εισάγεται με τέτοιο τρόπο ώστε η read(f, x) να είναι το ίδιο με το x := f^; get(f);. Η write εισάγεται με τέτοιο τρόπο ώστε η write(f, x) να είναι το ίδιο με το f^ := x; put(f); Ο τύπος text είναι προορισμένος ως αρχείο χαρακτήρων. Αν και η μεταβλητή ρυθμιστής θα μπορούσε να χρησιμοποιείται για την επισκόπηση του επόμενου χαρακτήρα που πρόκειται να χρησιμοποιηθεί (ελέγχει για ένα ψηφίο πριν την ανάγνωση ενός integer), κάτι τέτοιο οδηγεί σε μια σειρά προβλημάτων με διαδραστικά προγράμματα στις πρώιμες υλοποιήσεις αλλά λύθηκε αργότερα με την έννοια του lazy I/O.
Στην Jensen & Wirth Pascal τα strings αναπαρίστανται σαν πίνακες χαρακτήρων και για αυτό έχουν συγκεκριμένο μέγεθος.

3.5.8 Τύποι δεικτών

Η Pascal υποστηρίζει την χρήση δεικτών:

```
type
  pNode = ^Node;
  Node  = record
          a : integer;
          b : char;
          c : pNode  {το επιπλέον ερωτηματικό δεν απαιτείται}
          end;
var
  NodePtr : pNode;
  IntPtr  : ^integer;
```

Εδώ η μεταβλητή NodePtr είναι ένας δείκτης στον τύπο δεδομένων Node ένα record. Οι δείκτες μπορούν να χρησιμοποιηθούν πριν να δηλωθούν. Αυτή είναι μια προς τα εμπρός δήλωση, μια εξαίρεση στον κανόνα ότι τα πράγματα θα πρέπει να δηλώνονται πριν χρησιμοποιηθούν.
Για να δημιουργήσουμε ένα νέο record και για να του αναθέσουμε την τιμή 10 και τον χαρακτήρα A στα πεδία a και b του record και για την αρχικοποίηση του δείκτη c στο NIL οι εντολές θα ήταν οι ακόλουθες:

```
New(NodePtr);
...
NodePtr^.a := 10;
NodePtr^.b := 'A';
NodePtr^.c := NIL;
...
```

Αυτό θα μπορούσε να γίνει και με χρήση της with δήλωσης ως ακολούθως:

```
New(NodePtr);
...
```

```
with NodePtr^ do
begin
  a := 10;
  b := 'A';
  c := NIL
end;
```

Μέσα στην έκταση της with δήλωσης οι a και b αναφέρονται στα πεδία του δείκτη σε record NodePtr και όχι στο record Node λη στον δείκτη τύπου pNode.

Συνδεδεμένες λίστες, στοίβες και ουρές μπορούν να δημιουργηθούν συμπεριλαμβάνοντας ένα πεδίο τύπου δείκτη στο record.

Σε αντίθεση με άλλες γλώσσες οι οποίες χρησιμοποιούν δείκτες, η Pascal επιτρέπει στους δείκτες να αναφερθούν μόνο σε δυναμικά δημιουργημένες μεταβλητές οι οποίες είναι ανώνυμες και δεν επιτρέπει την αναφορά σε στάνταρ στατικές ή τοπικές μεταβλητές. Οι δείκτες πρέπει επίσης υποχρεωτικά να έχουν ένα τύπο αναφοράς και ένας δείκτης σε ενός είδους τύπο δεν είναι συμβατός με ένα δείκτη σε άλλο τύπο (για παράδειγμα ένας δείκτης σε char δεν είναι συμβατός με ένα δείκτη σε integer). Αυτός ο κανόνας μας επιτρέπει να εξολοθρεύσουμε τα προβλήματα ασφάλειας τύπων τα οποία έχουν κληροδοτηθεί από άλλες υλοποιήσεις δεικτών, ιδίως αυτές που έχουμε στην C και στην PL/I. Επίσης αποτρεπει προβλήματα που οφείλονται σε κινούμενους δείκτες, αλλά η δυνατότητα του δυναμικά αποδιανεμημένου χώρου με χρήση της συνάρτησης dispose (η οποία έχει το ίδιο αποτέλεσμα όπως και η free συνάρτηση βιβλιοθήκης της C) σημαίνει ότι ο κίνδυνος των κινούμενων δεικτών δεν έχει απαλειφθεί πλήρως[230] όπως έχει γίνει σε γλώσσες όπως η Java και η C# οι οποίες παρέχουν αυτοματοποιημένη αποκομιδή σκουπιδιών (οι οποίες ωστόσο δεν έχουν εξαλείψει πλήρως το πρόβλημα των διαρροών μνήμης).

Κάποιοι από αυτούς τους περιορισμούς αίρονται σε νεώτερες διαλέκτους.

3.5.9 Δομές ελέγχου

Η Pascal είναι μια γλώσσα δομημένου προγραμματισμού με την έννοια ότι η ροή του ελέγχου δομείται μέσα σε συγκεκριμένες δηλώσεις συνήθως δίχως την παρουσία της goto

```
while a <> b do  WriteLn('Waiting');

if a > b then WriteLn('Αν  η συνθήκη είναι true')   {δεν επιτρέπεται ερωτηματικό!}
        else WriteLn('Αν η συνθήκη είναι false');

for i := 1 to 10 do  {Δεν επιτρέπεται ερωτηματικό για μόνες δηλώσεις!}
  WriteLn('Iteration: ', i);
```

230 J. Welsh, W. J. Sneeringer, and C. A. R. Hoare, "Ambiguities and Insecurities in Pascal", *Software Practice and Experience 7*, pp. 685–696 (1977)

```
repeat
  a := a + 1
until a = 10;

case i of
  0 : Write('zero');
  1 : Write('one');
  2 : Write('two');
  3,4,5,6,7,8,9,10: Write('?')
end;
```

3.5.10 Διαδικασίες και συναρτήσεις

Στην Pascal τα προγράμματα δομούνται σε διαδικασίες και συναρτήσεις.

```
program Mine(output);

var i : integer;

procedure Print(var j : integer);
begin
...
end;

begin
...
Print(i);
end.
```

Οι διαδικασίες και οι συναρτήσεις μπορούν να είναι ένθετες σε οποιδήποτε επίπεδο βάθους και η κατασκευή του προγράμματος είναι το απόλυτο λογικό τμήμα.

Κάθε διαδικασία ή συνάρτηση μπορεί να έχει τις δικές της δηλώσεις ή goto ετικέτες, τύπους, σταθερές, μεταβλητές και άλλες διαδικασίες ή συναρτήσεις οι οποίες θα πρέπει όλες να είναι σε αυτή τη σειρά. Η απαιτήσεις στην σειρά των χαρακτηριστικών αρχικά σκόπευε να πετύχει αποτελεσματική μεταγλωττίση μονού περάσματος. Ωστόσο σε κάποιες διαλέκτους (όπως η Embarcadero Delphi) η αυστηρή σειρά των τμημάτων δήλωσης είναι περισσότερο χαλαρή.

3.5.11 Ερωτηματικά και διαχωριστές δηλώσεων

Η Pascal υιοθέτησε πολλά συντακτικά χαρακτηριστικά από την γλώσσα ALGOL συμπεριλαμβανομένης και της χρήσης του ερωτηματικού σαν διαχωριστή δηλώσεων. Αυτό, σε αντίθεση με άλλες γλώσσες, όπως η PL/I, η C κλπ, στις οποίες το ερωτηματικό αποτελεί την

ένδειξη τερματισμού της δήλωσης και όχι την ένδειξη διαχωρισμού. Όπως φαίνεται στα παραπάνω παραδείγματα δεν χρειάζεται ερωτηματικό πριν το end της δήλωσης ενός τύπου record, ενός block ή μιας δήλωσης υπόθεσης, πριν το until μιας εντολής repeat και πριν το else μιας δήλωσης if.
Η παρουσία ενός επιπλέον ερωτηματικού δεν ήταν επιτρεπτή στις αρχικές εκδόσεις της Pascal. Ωστόσο με την προσθήκη των παρόμοιων με της ALGOL κενών δηλώσεων το 1973 στην Revised Report καθώς και σε μεταγενέστερες αλλαγές στη γλώσσα στο ISO 7185:1983 επιτρέπεται πλέον η χρήση προαιρετικών ερωτηματικών στις περισσότερες από αυτές τις περιπτώσεις. Η χρήση του ερωτηματικού δεν επιτρέπεται ακόμη πριν το else μιας if δήλωσης μιας και το else ακολουθεί μια μονή δήλωση και όχι μια ακολουθία δηλώσεων. Στην περίπτωση των ένθετων if, η χρήση του ερωτηματικού δεν μπορεί να γίνει προκειμένου να αποφευχθεί το πρόβλημα του κινούμενου else (το αν δηλαδή το else ανήκει στο εσωτερικό ή στο εξωτερικό if), τερματίζοντας υποθετικά το ένθετο if με ένα ερωτηματικό. Κάτι τέτοιο αντιθέτως τερματίζει και τα δύο τμήματα if. Αντιθέτως ένα ξεχωριστό begin...end τμήμα είναι αναγκαίο να χρησιμοποιηθεί[231].
Οι προγραμματιστές συνήθως συμπεριλαμβάνουν αυτά τα επιπλέον ερωτηματικά λόγω συνήθειας και για να αποφύγουν την αλλαγή της τελευταίας γραμμής μιας ακολουθίας δηλώσεων όταν ο νέος κώδικας επεκτείνεται.

3.5.12 Πόροι

3.5.12.1 Μεταγλωττιστές και μεταφραστές

Διάφοροι μεταγλωττιστές και μεταφραστές της Pascal είναι διαθέσιμοι για γενική χρήση.

- Η Delphi είναι ιδιοκτησία της Embarcadero (πρώην Borland/CodeGear) και αποτελεί τη ναυαρχίδα των προϊόντων της εταιρίας στον τομέα προϊόντων γρήγορης ανάπτυξης λογισμικού. Χρησιμοποιεί την Object Pascal (την οποία ονόμασε Delphi η Borland) η οποία προέκυψε από την Pascal για την δημιουργία εφαρμογών για Windows, OS X, iOS και Android. Η υποστήριξη του .NET η οποία υπήρχε από την έκδοση D8, μέσω των D2005. D2006 και D2007, έχει πλέον τερματιστεί και αντικαταστάθηκε από μια νέα γλώσσα (την Prism, η οποία επανονομάστηκε σε Oxygene) η οποία δεν είναι απολύτως προς τα πίσω συμβατή. Στις πιο σύγχρονες εκδόσεις η υπσοτήριξη του unicode προστέθηκε (D2009, D2010, Delphi XE).
- Η Free Pascal είναι ένας διαπλατφορμικός μεταγλωττιστής γραμμένος σε Object Pascal (ο οποίος είναι αυτοφιλοξενούμενος). Σκοπεύει στην παροχή ενός βολικού και ισχυρού μεταγλωττιστή ο οποίος να είναι σε θέση να μεταγλωττίση τις υπάρχουσες εφαρμογές αλλά και να παρέχει τα μέσα ανάπτυξης νέων. Διανέμεται υπό το GNU GPL, ενώ τα πακέτα και οι βιβλιοθήκες τρέχοντος χρόνου έρχονται υπό την αιγίδα ενός τροποποοιημένου GNU LGPL. Εκτός από τις καταστάσεις συμβατότητας για τις Turbo Pascal, Delphi και Mac Pascal, διαθέτει επιπλέον καταστάσεις σύνταξης διαδικασιακού και αντικειμενοστραφούς προγραμματισμού με την επέκταση για αναβαθμισμένα χαρακτηριστικά όπως είναι η

231 *Pascal*, Nell Dale and Chip Weems, "Dangling Else", p. 160–161

υπερφόρτωση χειριστών. Υποστηρίζει πολλές υπολογιστικές πλατφόρμες και λειτουργικά συστήματα. Οι τρέχουσες εκδόσεις διαθέτουν επίσης και μια κατάσταση προτύπου ISO.

- Η Turbo51 είναι ένας ελεύθερος Pascal μεταγλωττιστής για την 8051 οικογένεια μικροϋπολογιστών με το συντακτικό της Pascal 7.
- Η Oxygene (μέχρι πρόσφατα γνωστή ως Chrome) είναι ένας μεταγλωττιστής Object Pascal για τα .NET και MONO. Δημιουργήθηκε και διατίθεται από την RemObjects Software και πρόσφατα από την Embarcadero σαν ο παρασκηνιακός μεταγλωττιστής του Prism.
- Το Kylix αποτελεί ένα απογονο της Delphi με υποστήριξη για το λειτουργικό σύστημα Linux καθώς και μια βελτιωμένη βιβλιοθήκη αντικειμένων. Πλέον δεν υποστηρίζεται. Ο μεταγλωττιστής και το IDE είναι πλέον διαθέσιμα για μη εμπορική χρήση.
- Ο μεταγλωττιστής της GNU (GPC – GNU Pascal Compiler) είναι ο μεταγλωττιστής του GNU Compiler Collection (GCC). Ο ίδιος ο μεταγλωττιστής είναι γραμμένος σε C και η βιβλιοθήκη χρόνου εκτέλεσης κυρίως σε Pascal. Διανέμεται υπό την GNU General Public License και τρέχει σε πολλές υπολογιστικές πλατφόρμες και λειτουργικά συστήματα. Υποστηρίζει τα ANSI/ISO πρότυπα της γλώσσας και διαθέτει μερική υποστήριξη της διαλέκτου της Turbo Pascal. Μια απο τις πλέον οδυνηρές ελλείψεις του είναι η πλήρης απουσία συμβατών συμβολοσειρών με την διάλεκτο της Turbo Pascal. Η υποστήριξη για την Borland Delphi και για άλλες εκδόσεις της γλώσσας είναι περιορισμένες και υπάρχει μικρή υποστήριξη για την Mac-Pascal.
- Ο DWScript επίσης γνωστή και ως DelphiWebScript είναι ένας μεταφραστής που δημιουργήθηκε από τους Matthiaw Ackermann και Hannes Hermler το 2000. Η τρέχουσα έκδοση χρησιμοποιεί μια διάλεκτο της Object Pascal η οποία είναι σε πολύ μεγάλο βαθμό συμβατή με την Delphi αλλά υποστηρίζει και δομές και στοιχεία της γλώσσες που συναντάμε στην Prism. Ο κώδικας του DWScript μπορεί να ενσωματωθεί στις εφαρμογές της Delphi, με τον ίδιο τρόπο που μπορεί και η PascalScript, μεταγλωττισμένος σε αυτοτελής εφαρμογές χρησιμοποιώντας το SimpleMobileStudio ή που έχουν μεταγλωττιστεί σε κώδικα JavaScript και έχει τοποθετηθεί μέσα σε μια ιστοσελίδα[232].
- Ο Dr, Pascal είναι ένας μεταφραστής ο οποίος εκτελέι Standard Pascal. Θα πρέπει να σημειώσουμε την κατάσταση ορατή εκτέλεσης οι οποία παρουσιάζει το εκτελούμενο πρόγραμμα και τις μεταβλητές τους καθώς και τον εκτεταμένο έλεγχο για σφάλματα χρόνου εκτέλεσης. Τρέχει τα προγράμματα αλλά δεν δημιουργεί ένα ξεχωριστό εκτελέσιμο δυαδικό αρχείο και τρέχει σε DOS, Windows σε παράθυρο DOS και σε παλαιότερα Macintosh.
- Ο Extended Pascal μεταγλωττιστής του Dr. Pascal. Ο οποίος τρέχεδι σε DOS, Windows 3,1, 95, 98, NT.
- Η Virtual Pascal δημιουργήθηκε από τον Vitaly Miryanov το 1995 σαν ένας ενσωματωμένος μεταγλωττιστής του OS/2 συμβατός με το συντακτικό της Borland Pascal. Ακολούθως αναπτύχθηκε εμπορικά από την fPrint προσθέτοντας υποστήριξη για το Win32

232 *"Flock-JSCodeGenDemo.7z - dwscript - "Flock" DWScript / JavaScript CodeGen demo - Delphi Web Script general purpose scripting engine - Google Project Hosting". Retrieved 11 August 2015.*

και το 200 έγινε ελεύθερο λογισμικό (freeware). Σήμερα μπορεί να εκτελεστέι σε Win32. OS/2, Linux και είναι σε πολύ μεγάλο βαθμό συμβατός με την Borland Pascal και με την Delphi. Η επιπλέον ανάπτυξη του σταμάτησε στις 4 Απριλίου το 2005.

- Ο μεταγλωττιστής P4 είναι η βάση για πολλούς ακόλουθους μεταγλωττιστές Pascal υλοποιημένους σε Pascal. Υλοποιεί ένα υποσύνολο της πλήρους Pascal.
- Ο μεταγλωττιστής P5 είναι ένα πρότυπο ISO 7185 (της πληρους Pascal) για τον P4 μεταγλωττιστή.
- Ο μεταγλωττιστής Smart Mobile Studio είναι ένας μεταγλωττιστής από Pascal σε HTML5/Javascript.
- Ο Turbo Pascal υπήρξη ο κυρίαρχος μεταγλωττιστής Pascal για τους προσωπικούς υπολογιστές κατά την δεκαετία του 80 και στις αρχές του 90 ιδιαιτέρως δημοφιλείς τόσο λόγω των ισχυρών του επεκτάσεων όσο και λόγω των ιδιαίτερα σύντομων χρόνων μεταγλώττισης. Η Turbo Pascal ήταν γραμμένη συμπαγώς γραμμένη και μπορούσε να μεταγλωττίσει, να εκτελέσει και να αποσφαλματώσει απευθείας από την μνήμη δίχως να απαιτείται η πρόσβαση στον αργό σκληρό δίσκο. Την εποχή που εμφανίστηκε τα οι αργοί οδηγοί δισκέτας αποτελούσαν κοινοτοπία για τους προγραμματιστές, χαρακτηριστικό που υπερτόνισε επιπλέον την ταχύτητα της Turbo Pascal. Σήμερα οι παλαιότερες εκδόσεις της Turbo Pascal είναι διαθέσιμες δωρεάν από το site της Borland.
- Ο IP Pascal υλοποιεί την γλώσσα Pascaline η οποία είναι μια ιδιαιτέρως εκτεταμένη Pascal συμβατή με την αρχική Pascal κατά το πρότυπο ISO 7185. Περιέχει έννοιες αναφορικά με την ονοματοσδοσία του χώρου, μαζί με έννοιες πολυεπεξεργασίας με σημαφόρους, αντικείμενα, δυναμικούς πίνακες κάθε μεγέθους το οποίο προσδιορίζεται κατά τον χρόνο εκτέλεσης, υπερφορτώσεις, παρακάμψεις και πολλές άλλες επεκτάσεις. Η IP Pascal έχει ενσωματωμένη βιβλιοθήκη φορητότητας η οποία είναι σχεδιασμένη αποκλειστικά για την γλώσσα Pascal. Για παράδειγμα, μια εφαρμογή με στάνταρ έξοδο κειμένου από την αρχική Pascal του 1970 μπορεί να επαναμεταγλωττιστή ώστε να δουλέψει σε περιβάλλον Windows και μάλιστα με γραφικά επιπρόσθετα.
- Η Pascal-XT δημιουργήθηκε από την Siemens για το λειτουργικό των mainframe της εταιρίας BS2000 και SINIX.
- Το PocketStudio είναι ένα υποσύνολο του Compiler της Pascal και των RAD εργαλείων για το Palm OS και τους MC68xxx επεξεργαστές με μερικές επιπλέον επεκτάσεις για να συνδράμουν την αλληλεπίδραση με το Palm OS API. Προσομοιώνει την Delphi και το Lazarus με ένα γραφικό σχεδιαστή φορμών, έναν επιθεωρητή αντικειμένων και ένα συντάκτη πηγαίου κώδικα.
- Ο MIDletPascal είναι ένα ένας μεταγλωττιστής Pascal και IDE ο οποίος παράγει μικρό και γρήγορο Java ενδιάμεσο κώδικα ειδικά σχεδιασμένο για την παραγωγή κώδικα για κινητά τηλέφωνα.
- Ο VectorPascal είναι ένας μεταγλωττιστής που στοχεύει το SIMD σύνολο εντολών όπως δηλαδή επεξεργαστές όπως οι MMX και οι AMD 3d Now, υποστηρίζοντας όλους τους επεξεργαστές της Intel και της AMD, καθώς επίσης και την Emotion Engine του Sony

PlayStation 2.

- Ο Morfik Pascal επιτρέπει την ανάπτυξη εφαρμογών ιστού γραμμένων εξολοκλήρου σε Object Pascal (τόσο για το μέρος του server όσο και για το μέρος του client).
- Το WBSibyl είναι ένα γραφικό περιβάλλον ανάπτυξης εφαρμογών και ένας μεταγλωττιστής Pascal για Win32 και OS/2.
- Ο PP μεταγλωτιστής είναι ένας μεταγλωττιστής για το Palm OS ο οποίος τρέχει κατευθείαν στον φορητό υπολογιστή.
- Ο CDC 6000 μεταγλωττιστής Pascal είναι ο πηγαίος κώδικας για τον πρώτο (CDC 6000) μεταγλωττιστή Pascal.
- Pascal-S[233]
- Ο AmigaPascal είναι ένας ελεύθερος μεταγλωττιστής Pascal για τον υπολογιστή της Amiga.
-

3.5.12.2 Ολοκληρωμένα περιβάλλοντα ανάπτυξης (IDEs)

- Το Dev-Pascal είναι ένα Pascal IDE το οποίο σχεδιάστηκε σε Borlans Delphi και το όποίο υποστηρίζει την FreePascal και την GNU Pascal σαν μεταγλωττιστές παρασκηνίου.
- Το Lazarus είναι ένα ελεύθερο παρόμοιο με την Delphi γραφικό και διαπλατφορμικό IDE για την γρήγορη ανάπτυξη εφαρμογών (RAD). Βασίζεται στην Free Pascal και είναι διαθέσιμο για πολλές πλατφόρμες συμπεριλαμβανομένων του Linux, FreeBSD, Mac OS X και Microsoft Windows.
- Το Code Typhon είναι ένα Pascal IDE φτιαγμένο σαν μια εκτεταμένη έκδοση του Lazarus με πληθώρα πακέτων και scripts για την κατασκευή FPC. Το Code Typhon σχεδιάστηκε σε Free Pascal και υποστηρίζει την Object Pascal, Turbo Pascal και Delphi. Οι προγραμματιστές θα μπορούσαν να παράξουν κώδικα για κάθε συσκευή.

3.5.12.3 Βιλιοθήκες

Η βιβλιοθήκη WOL για την δημιουργία εφαρμογών γραφικών περιβαλλόντων διεπαφής GUI με χρήση του μεταγλωττιστή Free Pascal.

3.5.13 Πρότυπα

3.5.13.1 ISO/IEC 7185:1990 Pascal

Το 1983 η γλώσσα προτυποποήθηκε με το διεθνές πρότυπο IEC/ISO 7185[234] και διάφορα τοπικά ανά χώρα συγκεκριμένα πρότυπα συμπεριλαμβανομένου και του American ANSI/IEEE770X3.97-

233 "Pascal-S: A Subset and Its Implementation", N. Wirth in Pascal – The Language and Its Implementation, by D.W. Barron, Wiley 1979.

234 *ISO/IEC 7185:1990 Pascal (PDF). Retrieved 16 September 2014.*

1983 και το ISO7185:1983. Αυτά τα δύο πρότυπα διέφεραν μόνο στο ότι το ISO πρότυπο συμπεριελάμβανε μία επιπέδου 1 επέκταση για σύμφωνους πίνακες (πίνακες για τους οποίους τα όρια δεν είναι γνωστά παρά κατά τον χρόνο εκτέλεσης) την στιγμή που το ANSI πρότυπο δεν επέτρεπε τη χρήση της συγκεκριμένης επέκτασης στην αρχική έκδοση (Wirth) της γλώσσας. Το 1989 το ISO 7185 αναθεωρήθηκε (σε ISO 7185:1990) για να διορθώσει λάθη και ασάφειες του αρχικού εγγράφου.
Το ISO 7185 αποτέλεσε μια διευκρίνηση της γλώσσας σε σχέση με την κατά Wirth περιγραφή της γλώσσας το 1974, το ίδιο λεπτομερές με τον οδηγό χρήσης και αναφοράς (Jensen and Wirth) και το οποίο διέθετε επίσης και την προσθήκη των σύμφωνων πινάκων σαν επίπεδο 1 του προτύπου με το επίπεδο 0 να είναι η Pascal δίχως τους σύμφωνους πίνακες. Η προσθήκη αυτή έγινε κατόπιν αίτησης του C.A.R. Hoare και με την έγκριση του Niklaus Wirth. Ο λόγος ήταν ότι ο Hoare ήθελε την δημιουργία μιας έκδοσης της Pascal για το (NAG) Numerical Algorithms Library, έκδοση η οποία αρχικά ήταν γραμμένη σε FORTRAN και ανακάλυψε ότι κάτι τέτοιο δεν θα ήταν δυνατόν να γίνει δίχως μια επέκταση η οποία θα επέτρεπε παραμέυτρους πινάκων μεταβλητού μεγέθους. Παρόμοιες ανησυχίες οδήγησαν στο να συμπεριληφθούν στο ISO 7185 πρότυπο οι δομές για τον προσδιορισμό των τύπων των παραμέτρων των διαδικασιών και των συναρτήσεων.
Να σημειωθεί ότι ο ίδιος ο Nikalus Wirth αναφέρθηκε στη γλώσσα του 1974 σαν το πρότυπο, για παράδειγμα για να το διαχωρίσει από τον CDC 6000 μεταγλωττιστή. Αυτή η έκδοση της γλώσσας ήταν τεκμηριωμένη στο The Pascal Report[235] το δεύτερο μέρος του Pascal users manual ans report.
Στις μεγάλες μηχανές (mainframes και minicomputers) στις οποίες ξεκίνησε την ζωή της η Pascal τα πρότυπα σε γενικές γραμμές ακολουθήθηκαν. Στον IBM PC δεν έγινε κάτι τέτοιο. Στους IBM PC ο μεγαλύτερος αριθμός χρηστών ακολούθησε το πρότυπο της Borland για την Turbo Pascal και την Delphi. Έτσι είναι πολύ σημαντικό να ξεχωρίζουμε το αν μια συγκεκριμένη υλοποίηση ανταποκρίνεται στο αρχικό πρότυπο της γλώσσας ή στο πρότυπο της διαλέκτου της Borland.
Οι εκδόσεις της γλώσσας για το IBM PC ξεκίνησαν να διαφέρουν με την άνοδο της UCSD Pascal μιας μεταφρασμένης υλοποίησης η οποία υλοποίησε τις διάφορες εκδόσεις της γλώσσας μαζι με διάφορες παραλείψεις και αλλαγές. Πολλά από τα χαρακτηριστικά της USCD έκδοσης της γλώσσας επιβιώνουν σήμερα μαζί με την διάλεκτο της Borland.

3.5.13.2 ISO/IEC 10206:1990 Extented Pascal

Το 1990 μια επέκταση του προτύπου της Pascal δημιουργήθηκε και ονομάστηκε ISO/IEC 10206 το οποίο όσον αφορά το περιεχόμενο είναι ίδιο με το IEEE/ANSI 770X3.160-1989 πρότυπο[236].

3.5.13.3 Παραλλαγές

235 *Wirth, Niklaus (July 1973). The Programming Language Pascal (Revised Report) (PDF). ETH Zürich. Retrieved 16 September 2014.*

236 *770X3.160-1989 - IEEE/ANSI Standard for the Programming Language Extended Pascal. Retrieved 16 September 2014.*

Η έκδοση της Ζυρίχης του Niklaus Wirth, της Pascal εκδόθηκε εκτός ETH σε δύο βασικές μορφές, τον πηγαίο κώδικα του μεταγλωττιστή για το CDC 6000 και ένα σύστημα εισαγωγής το οποίο ονομάστηκε Pascal P-system. Ο μεταγλωττιστής Pascal-P άφησε εκτός διάφορα χαρακτηριστικά της πλήρους γλώσσας. Για παράδειγμα, η χρήση των διαδικασιών και των συναρτήσεων ως παραμέτρους, μη διαχωρίσημες μεταβλητές εγγραφών, η συσκευασία , η απόρριψη, οι διαδικαστικές goto και άλλα χαρακτηριστικά της πλήρους γλώσσας παραλήφθηκαν.
Η UCSD Pascal υπό τον καθηγητή Kenneth Bowles βασίστηκε στην Pascal-P2 και κατά συνέπεια κληρονόμησε πολλούς από τους περιορισμούς της Pascal-P. Η UCSD Pascal αργότερα υιοθετήθηκε σαν η Apple Pascal και απέκτησε πολλές εκδόσεις εκεί. Αν και η UCSD Pascal ουσιαστικά επέκτεινε το υπόσυνολο της Pascal στην Pascal-P προσθέτοντας αρκετές από τις δομές της στάνταρ Pascal, παρέμεινε μια μη πλήρης έκδοση της Pascal.
Η Turbo Pascal της Borland γραμμένη από τον Anders Hejlsberg, ήταν γραμμένη σε assembly ανεξάρτητα από τον UCSD ή τους μεταγλωττιστές της Ζυρίχης. Ωστόσο υιοθέτησε το ίδιο υποσύνολο και επεκτάσεις όπως και ο UCSD μεταγλωττιστής. Αυτό συνέβη πιθανότατα μιας και το UCSD σύστημα ήταν το πιο κοινά χρησιμοποιούμενο σύστημα Pascal κατάλληλο για την ανάπτυξη εφαρμογών στους περιορισμένων πόρων μικροεπεξεργαστικά συστήματα της εποχής.
Στις αρχές της δεκαετίας του 1990 οι Alan Burns και οι Geoff Davies ανέπτυξαν την Pascal-FC, μια επέκταση του Pl/0 (από το βιβλίο του Niklaus Wirth Algorithms + Data Structures = Programs). Διάφορες δομές προστέθηκαν για να χρησιμοποιηθεί η Pascal-FC σαν ένα εργαλείο μάθησης για τον παράλληλο προγραμματισμό (με χαρακτηριστικά όπως οι σημαφόροι, τα monitors, τα channels, τα resources και τα remote-invocation). Για να είναι δυνατή η επίδειξη του παραλληλισμού η έξοδος του μεταγλωττιστή (ένας είδος P-κώδικα) μπορούσε να εκτελεστεί σε μια εικονική μηχανή/ Αυτή η μηχανή όχι μόνο προσομοίωνε ένα περιβάλλον εκτέλεσης υπό κανονικές συνθήκες (fair mode) αλλά μπορούσε επίσης να προσομοιώσει και ακραίες καταστάσεις (unfair mode).

3.5.13.4 Συγγενικά πρότυπα

To ISO 8651-2:1988 Information processing systems – Computer graphics – Graphical Kernel System (GKS) language bindings – Part 2: Pascal

3.5.14 Αποδοχή

Η Pascal δημιούργησε μια πλειάδα αντιδράσεων από την κοινότητα της επιστήμης των υπολογιστών κατά την εμφάνιση της, τόσο θετικές όσο και αρνητικές

3.5.15 Όψιμη κριτική

Παρόλο που ήταν ιδιαίτερα δημοφιλής κατά την δεκαετία του 80 και στις αρχές του 90, οι υλοποιήσεις της Pascal που ακολούθησαν τον αρχικό ορισμό του Wirtth για την γλώσσα δέχθηκαν έντονη κριτική ως μη κατάλληλες για διδασκαλεία. Ο Brian Kernighan ο οποίος έκανε διάσημη την

C γλώσσα, περιέγραψε την κριτική του για την Pascal ήδη από το 1981, στην εργασία του Why Pascal Is Not My Favorite Programming Language[237]. Το σοβαρότερο πρόβλημα που περιέγραφε στο άρθρο αυτό ήταν ότι τα μεγέθη των πινάκων και τα μήκη των συμβολοσειρών αποτελούσαν μέρος του τύπου και άρα δεν ήταν δυνατή η συγγραφή μιας συνάρτησης η οποία θα δεχόταν μεταβλητά μήκη πινάκων ή συμβολοσειρών ως παραμέτρους. Κάτι τέτοιο καθιστούσε ανέφικτη την συγγραφή για παράδειγμα μιας βιβλιοθήκης ταξινόμησης. Ο συγγραφέας επίσης άσκησε κριτική στην απρόβλεπτη σειρά αξιολόγησης των λογικών εκφράσεων, την φτωχή υποστήριξη βιβλιοθήκης και την έλλειψη static μεταβλητών και έθεσε και ένα σύνολο επιπλέον μικρότερης σημασίας ζητημάτων. Επίσης δήλωσε ότι η γλώσσα δεν παρείχε απλές κατασκευές για την απόδραση περιορισμών και απαγορεύσεων. (Ωστόσο υπάρχει ένα χαρακτηριστικό το οποίο επιτρέπει τέτοιου είδους υπεκφυγές αν και αναμφισβήτητα είναι δύσχρηστο). Άλλα γενικότερα παράπονα από άλλες πλευρές[238][239] σημείωναν ότι το εύρος των δηλώσεων δεν δηλώνεται ξεκάθαρα στους ορισμούς της αρχικής γλώσσας, γεγονός το οποίο κάποιες φορές είχε πολύ σοβαρές συνέπειες κατά την δήλωση δηλώσεων προς τα εμπρός για την δήλωση δεικτών ή όταν η δήλωση εγγραφών οδηγούσε σε αμοιβαία αναδρομή ή όταν ένα αναγνωριστικό χρησιμοποιείτο ή όχι σε μια λίστα enumeration. Μια άλλη δυσκολία ήταν ότι όπως και στην ALGOL 60 η γλώσσα δεν επέτρεπε διαδικασίες ή συναρτήσεις να περάσουν ως παράμετροι για την εκ των προτέρων δήλωση των αναμενώμενων τύπων των παραμέτρων τους.

Πάρα την κριτική όμως η Pascal συνέχισε να εξελίσσεται και τα περισσότερα από τα ζητήματα που έθιξε ο Kernighan λύθηκαν στις μεταγενέστερες εκδόσεις της γλώσσας οι οποίες εμπλουτίστηκαν για να είναι κατάλληλες για την ανάπτυξη εμπορικών εφαρμογών όπως ήταν η Turbo pascal. Όπως προέβλεψε ο Kernighan στο άρθρο του οι περισσότερες επεκτάσεις που φτιάχτηκαν για την επίλυση αυτών των προβλημάτων ήταν ασύμβατες από μεταγλωττιστή σε μεταγλωττιστή. Από τις αρχές της δεκαετίας του 90 ωστόσο οι διαφοροποιήσεις φάνηκαν να συγκεντρώνονται σε δύο μόνο κατηγορίες στην κατηγορία των ISO και στην κατηγορία της Borland, μια εξέλιξη καλύτερη από αυτή που προέβλεψε ο Kernighan.

Αν και ο Kernighan κατέκρινε την αδυναμία της Pascal για την απόδραση από τους τύπους οι δείκτες και οι απόποπειρες απόδρασης από τους τύπους οδήγησαν στο είδος το προβλημάτων τα οποία μπορούσαν και έλυναν γλώσσες σαν την Java και την C#.

Το μεγαλύτερο μέρος της κριτικής του Kernighan αφορούσε την εγασία The Pascal Programming language[240] και ιδιαίτερα το κομμάτι του Myth 6: Pascal is not For Serious Programmers[241].

Η κριτική της εργασίας του kernighan κατέληξε αναχρονιστική και άσχετη με την υλοποίηση της

---

237 Brian W. Kernighan (1981). Why Pascal is Not My Favorite Programming Language

238 J. Welsh, W. J. Sneeringer, and C. A. R. Hoare, "Ambiguities and Insecurities in Pascal", *Software Practice and Experience 7*, pp. 685–696 (1977)

239 O. Lecarme, P. Desjardins, "More Comments on the Programming Language Pascal," *Acta Informatica 4*, pp. 231–243 (1975)

240 The Pascal Programming Language

241 Pascal Myths

εκτεταμένης Pascal[242].

Ακολουθεί ένα άθροισμα των κριτικών του kernighan καθώς και των επεκτάσεων της Pascal που τις αντιμετωπίζουν.

1. Το μέγεθος όλων των πινάκων είναι τμήμα του τύπου και άρα δεν είναι δυνατή η συγγραφή διαδικασιών γενικού σκοπού για την διαχείριση συμβολοσειρών με διαφορετικά μεγέθη. Με την εισαγωγή της εκτεταμένης Pascal τα αλφαριθμητικά μεταβλητού μεγέθους υλοποιήθηκαν μαζί με αρκετές άλλες δυναμικές δυνατότητες των αλφαριθμητικών[243]
2. Η έλλειψη static μεταβλητών και αρχικοποίησης των μεταβλητών καταστρέφει την εντοπιότητα ενός προγράμματος.
   Η αρχικοποίηση μεταβλητών περιλαμβάνεται στην εκτεταμένο πρότυπο της Pascal[244].
3. Η έλλειψη ξεχωριστή ς μεταγλώττισης παρεμποδίζει την ανάπτυξη μεγάλων προγραμμάτων και κάνει αδύνατη τη χρήση βιβλιοθηκών.
   Η επεκτασιμότητα και η ξεχωριστή μεταγλώττιση είναι κομμάτι του εκτεταμένου προτύπου της Pascal[245].
4. Η σειρά αξιολόγησης των λογικών εκφράσεων δεν μπορεί να ελεγχθεί γεγονός που οδηγεί στην παραγωγή μπερδεμένου κώδικα και εξωτερικών μεταβλητών.
   Οι βραχυκύκλωση λογικών τελεστών είναι τμήμα του εκτεταμένου προτύπου της Pascal
5. Δεν υπάρχει έλεγχος της ροής του προγράμματος λόγω της έλλειψης των εντολών return και break.
   Αν και δεν περιλαμβάνονται στα εκτεταμένα πρότυπα, οι πλέον σύγχρονες υλοποιήσεις της Pascal υποστηρίζουν έλεγχο συναρτήσεων και βρόχων. Τόσο στην Compq όσο και στην CodeWarrior Pascal η return θα έχει ως αποτέλεσμα την έξοδο από μια συνάρτηση . Στην Compaq Pascal οι continue και break υποστηρίζονται για τον έλεγχο των βρόχων και στην Code Warrior υπάρχουν οι αντίστοιχες cycle και leave. Προσφάτως η Free Pascal υποστηρίζει πλέον τις break[246], continue[247] και exit[248].
6. Η δήλωση case έχει αποδυναμωθεί γιατί δεν υπάρχει προεπιλεγμένη πρόταση.
   Η Extended Pascal τώρα ποια υποστηρίζει μια otherwise πρόταση στην δήλωση της case[249].
7. Η γλώσσα στερείται τα περισσότερα εργαλεία τα οποία είναι απαραίτητα για την συναρμολόγηση μεγάλων προγραμμάτων με περισσότερο σημαντικό την συμπερίληψη αρχείων.
   Οι έννοιες της Extended Pascal παρέχουν πολύ πιο προηγμένη υποστήριξη για την διαχείριση και συναρμολόγηση προγραμμάτων από την πρωτόγονη συμπερίληψη αρχείων.

---

242 Extended Pascal

243 Extended Pascal - Chapter 3

244 Extended Pascal - Chapter 3

245 Extended Pascal - Chapter 3

246 Free Pascal BREAK

247 Free Pascal CONTINUE

248 Free Pascal EXIT

249 Extended Pascal - Chapter 3

Παρόλα αυτά οι νέες εκδόσεις της Pascal εμπεριέχουν και την συμπερίληψη αρχείων.

8. Δεν υπάρχει διαφυγή από τους δυνατούς ελέγχους γραφής των τύπων. Αν και δεν δηλώνεται στα πρότυπα η μετατροπή τύπων υποστηρίζεται στις μοντέρνες Pascal. Στην Compaq Pascal ο :: είναι ο τελεστής μετατροπής τύπων. Στην Code Warrior Pascal η μετατροπή υλοποιείται με την χρήση παρενθέσεων. Η γλώσσα κατόρθωσε να εξελιχθεί δίχως να θυσιάσει τα οφέλη που απορρέουν από την ισχυρή δήλωση τύπων[250].

250 Pascal Architecture

Κεφάλαιο 4
Μεταγλώσσες

4 Μεταγλώσσες

4.1 Ορισμός

Γενικά ως μεταγλώσσα ορίζουμε κάθε γλώσσα ή σύμβολα τα οποία χρησιμοποιούνται όταν το θέμα συζήτησης ή εξέτασης είναι η ίδια η γλώσσα[251]. Στην λογική και στην γλωσσολογία, η με ταγλώσσα χρησιμοποιείται για να γίνουν δηλώσεις που αφορούν δηλώσεις άλλων γλωσσών (την γλώσσα αντικείμενο).Οι εκφράσεις μια μεταγλώσσας, διαχωρίζονται συχνά, από τις εκφράσεις της αντικειμενικής γλώσσας, από την χρήση της πλάγιας γραφής, των εισαγωγικών, ή της γραφής σε διαφορετική σειρά.

4.2 Τύποι μεταγλωσσών

Υπάρχει μια ποικιλία αναγνωρισμένων μεταγλωσσών μεταξύ των οποίων είναι και οι ενσωματωμένες, οι προκαθορισμένες και οι ένθετες (ή ιεραρχικές).

4.2.1 Ενσωματωμένες μεταγλώσσες

Ως ενσωματωμένη ορίζεται μια μεταγλώσσα η οποία είναι τυπικά, φυσικά και σφιχτά προσδεδεμένη σε μια αντικειμενική γλώσσα. Η ιδέα απαντάται στο βιβλίο του Douglas Hofstadter Godel, Escher, Bach, σε μια συζήτηση που αφορά τη σχέση μεταξύ δύο τυπικών γλωσσών και μιας αριθμητικής θεωρίας: “...είναι στην φύση κάθε κανονικοποιημένης θεωρίας αριθμών η μεταγλώσσα της να είναι ενσωματωμένη μέσα σε αυτή”. Παρατηρείται σε τυπικές ή μη τυπικές γλώσσες, όπως η Ελληνική, όπου λέξεις όπως ουσιαστικό, ρήμα ή ακόμα και λέξη, περιγράφουν χαρακτηριστικά και έννοιες που αφορούν την ίδια την Ελληνική γλώσσα.

4.2.2 Ενσωματωμένες οπτικές μεταγλώσσες

Μια ενσωματωμένη οπτική μεταγλώσσα είναι μια γλώσσα εικόνων βασισμένη σε κοινούς ανθρώπινους οπτικούς συσχετισμούς. Με την σειρά τους οι διάφορες γλώσσες ανήκουν στην υπερδομή των μεταγλωσσών και είναι απαραίτητες μόνο για την περιγραφή των εικόνων. Σύμφωνα με την θεωρία των αισθήσεων (Komogorov, 2015), μέσω των αισθήσεων ένα άτομο προσλαμβάνει όλες όλες τις βασικές πληροφορίες από τον αντικειμενικό κόσμο. Ως συναίσθημα ορίζουμε μια αισθησιακή εικόνα των επιμέρους πτυχών, διαδικασιών και φαινομένων του αντικειμενικού κόσμου. Λόγω της σθεναρής δραστηριότητας του ανθρώπινου μυαλού οι εικόνες που αισθάνεται

251

2010. *Cambridge Advanced Learner‘s Dictionary*. Cambridge: Cambridge University Press. Dictionary online. Available from http://dictionary.cambridge.org/dictionary/british/metalanguage Internet. Retrieved 20 November 2010

εισερχόμενες στο ανθρώπινο μυαλό υφίστανται μια ενεργή διαχείριση και μετασχηματίζονται σε εικόνες της αντίληψης παρέχοντας τα προαπαιτούμενα για την χορήγηση του νοήματος στις εικόνες αυτές[252]. Για αυτούς τους λόγους μπορούν οι εικόνες να αποτελέσουν κοινή πλατφόρμα επεξηγήσεων σε κάθε γλώσσα.

4.2.3 Προκαθορισμένες μεταγλώσσες

Η προκαθορισμένες μεταγλώσσες είναι το ανάλογο της προκαθορισμένης λογικής. Ένα παράδειγμα προκαθορισμένης μεταγλώσσας είναι η κατασκευή μιας μεταγλώσσας με σκοπό την συζήτηση μιας αντικειμενικής γλώσσας η οποία ακολουθείται από την δημιουργία μιας άλλης μεταγλώσσας για την συζήτηση της πρώτης κλπ.

4.2.4 Ένθετες μεταγλώσσες

Οι ένθετες (ή ιεραρχημένες) μεταγλώσσες είναι παρόμοιες με τις προκαθορισμένες μεταγλώσσες έτσι ώστε κάθε επίπεδο να αναπαριστά ένα μεγαλύτερο βαθμό αφαίρεσης. Ωστόσο η ένθετη μεταγλώσσα διαφέρει από μια προκαθορισμένη στο ότι κάθε επίπεδο εμπεριέχει το προηγούμενο επίπεδο του. Ένα παράδειγμα ένθετης μεταγλώσσας μας έρχεται από το ταξινομικό σύστημα του Linnaeus στη βιολογία. Κάθε επίπεδο του συστήματος ενσωματώνει το προηγούμενο επίπεδο του. Η γλώσσα η οποία χρησιμοποιείται για την συζήτηση τα γένη επίσης χρησιμοποιείται για την συζήτηση των ειδών, η γλώσσα που χρησιμοποιείται για την συζήτηση των τάξεων επίσης χρησιμοποιείται για τα γένη κλπ έως το επίπεδο των βασιλείων.

4.3 Οι μεταγλώσσες στις φυσικές γλώσσες

Οι φυσικές γλώσσες συνδυάζουν ένθετες και προκαθορισμένες μεταγλώσσες. Σε μια φυσική γλώσσα υπάρχουν άπειρες οπισθοδρομήσεις μεταγλωσσών, κάθε μια με περισσότερο εξειδικευμένο λεξιλόγιο και απλούστερο συντακτικό. Σημαίνοντας την γλώσσα ως L_0 η γραμματική της γλώσσας είναι μια συνδιαλλαγή στην μεταγλώσσα L_1 η οποία είναι μια υπογλώσσα253 ένθετη μέσα στην γλώσσα L_0. Η γραμματική της L_1 η οποία έχει την μορφή πραγματικής περιγραφής είναι μια συνδιαλλαγή μέσα στην μεταγλώσσα L_2 η οποία είναι επίσης μια υπογλώσσα της L_0.. Η γραμματική της L_2, η οποία έχει την μορφή μιας θεωρίας η οποία περιγράφει την συντακτική δομή αυτών των πραγματικών περιγραφών, δηλώνεται στην μέταμεταγλώσσα L_3 η οποία παρομοίως είναι μια υπογλώσσα της L_0. Η γραμματική της L_3 έχει την μορφή μιας μεταθεωρίας που περιγράφει την συντακτική δομή των θεωριών που δηλώνονται στην L_2. Η L_4 και οι απόγονοι της έχουν όλοι την ίδια γραμματική με την L_3 με διαφορές μόνο στις αναφορές. Μιας και όλες αυτές οι μεταγλώσσες είναι υπογλώσσες της L_0 η L_1 είναι μια ένθετη μεταγλώσσα αλλά η L_2 και οι επόμενες στη σειρά είναι προκαθορισμένες μεταγλώσσες254. Αφού όλες αυτές οι μεταγλώσσες

252 Hofstadter, Douglas. 1980. *Gödel, Escher, Bach: An Eternal Golden Braid*. New York: Vintage Books ISBN 0-14-017997-6

253 *Harris, Zellig S. (1991). A theory of language and information: A mathematical approach. Oxford: Clarendon Press. pp. 272–318. ISBN 0-19-824224-7.*

254 *Ibid*. p. 277.

είναι υπογλώσσες της L_0 είναι όλες ενσωματωμένες γλώσσες με αποδοχή της γλώσσας συνολικά.
Οι μεταγλώσσες των τυπικών συστημάτων όλες καταλήγουν τελικά στην φυσική γλώσσα, “την κοινή διάλεκτο” για την οποία μαθηματικοί και λογικολόγοι συζητούν για τον ορισμό των όρων και των λειτουργιών για την ανάγνωση των τύπων255.

4.4 Είδη εκφράσεων στις μεταγλώσσες

Υπάρχουν διάφορες οντότητες οι οποίες εκφράζονται συχνά σε μια μεταγλώσσα. Στην λογική συνήθως η αντικειμενική γλώσσα την οποία συζητά η μεταγλώσσα είναι μια τυπική γλώσσα όπως πολύ συχνά συμβαίνει και με την ίδια την μεταγλώσσα.

4.4.1 Επαγωγικά συστήματα

Ένα επαγωγικό σύστημα αποτελείται από τα αξιώματα (ή τα σχήματα των αξιωμάτων) και τους κανόνες συμπερασμάτων που μπορούν να χρησιμοποιηθούν για την εξαγωγή των θεωρημάτων του συστήματος256.

4.4.2 Μεταμεταβλητές

Μια μεταμεταβλητή (ή μεταγλωσσική μεταβλητή) είναι ένα σύμβολο ή σύνολο συμβόλων σε μια μεταγλώσσα το οποίο έχει την σημασία ενός συμβόλου ή συνόλου συμβόλων στην αντικειμενική γλώσσα την οποία περιγράφει η μεταγλώσσα. Για παράδειγμα στην πρόραση:
Έστω ότι A και B είναι αυθαίρετοι τύποι μια τυπικής γλώσσας L. Τα σύμβολα A και B δεν είναι σύμβολα της αντικειμενικής γλώσσας L, είναι μεταμεταβλητές στην μεταγλώσσα η οποία συζητά την αντικειμενική γλώσσα L.

4.4.3 Μεταθεωρίες και μεταθεωρήματα

Μια μεταθεωρία είναι μια θεωρία της οποίας το αντικειμενικό ζήτημα είναι μια άλλη θεωρία (μια θεωρία για μια άλλη θεωρία). Οι δηλώσεις που γίνονται στην μεταθεωρία και αφορούν την αντικειμενική θεωρία ονομάζονται μεταθεωρήματα. Ένα μεταθεώρημα είναι μια αληθής δήλωση για ένα τυπικό σύστημα η οποία εκφράζεται σε μια μεταγλώσσα. Σε αντίθεση με τα θεωρήματα τα οποία έχουν αποδειχτεί μέσα σε ένα δεδομένο τυπικό σύστημα, τα μεταθεωρήματα αποδεικνύονται μέσα σε μια μεταθεωρία και μπορεί να αναφέρονται σε έννοιες οι οποίες είναι υπαρκτές στην μεταθεωρία αλλά όχι στην αντικειμενική θεωρία257.

4.5 Μεταφράσεις

---

255 *Borel, Félix Édouard Justin Émile (1928). Leçons sur la theorie des fonctions (in French) (3 ed.). Paris: Gauthier-Villars & Cie. p. 160.*

256 Hunter, Geoffrey. 1971. *Metalogic: An Introduction to the Metatheory of Standard First-Order Logic*. Berkeley:University of California Press ISBN 978-0-520-01822-8

257 Ritzer, George. 1991. *Metatheorizing in Sociology*. New York: Simon Schuster ISBN 0-669-25008-2

Μετάφραση είναι η ανάθεση νοημάτων στα σύμβολα και στις λέξεις μιας γλώσσας.

4.6 Ρόλος στην μεταφορά

Ο Michael J. Reddy το 1979 ανακάλυψε και επέδειξε ότι ένα μεγάλο τμήμα της γλώσσας που χρησιμοποιούμε για να μιλήσουμε για την γλώσσα γίνεται αντιληπτό και δομείται από το αυτό που ο ίδιος αναφέρει ως αγωγός μεταφοράς258.
Το παρακάτω παράδειγμα γίνεται μεταξύ δύο σχετιζόμενων πλαισίων.
Το κύριο πλαίσιο θεωρεί την γλώσσα σαν ένα σφραγισμένο αγωγό μεταξύ ανθρώπων:

- Η γλώσσα μεταφέρει τις σκέψεις και τα συναισθήματα των ανθρώπων (διανοητικό περιεχόμενο) σε άλλους ανθρώπους
  παράδειγμα: *Προσπάθησε να μεταφέρεις τις σκέψεις σου καλύτερα*
- Οι ομιλητές και οι συντάκτες εισάγουν το διανοητικό περιεχόμενο τους σε λέξεις
  παράδειγμα: *Πρέπει να βάζεις κάθε έννοια σε λέξεις περισσότερο προσεκτικά*
- Οι λέξεις είναι δοχεία
  παράδειγμα: *Αυτή η πρόταση γέμισε με συναίσθημα*
- Οι ακροατές και οι συντάκτες εξάγουν το διανοητικό περιεχόμενο από τις λέξεις
  παράδειγμα: *Επέτρεψε μου να ξέρω αν βρεις νέες αισθήσεις στο ποίημα.*

Το ελάσσων πλαίσιο θεωρεί τις γλώσσες σαν έναν ανοιχτό αγωγό ο οποίος διαρρέει διανοητικό περιεχόμενο στο κενό:

- Οι ομιλητές και οι συντάκτες εκβάλουν διανοητικό περιεχόμενο σε ένα εξωτερικό χώρο
  παράδειγμα: *Δώσε αυτές τις ιδέες εκεί που μπορούν να προσφέρουν κάτι καλό.*
- Το διανοητικό περιεχόμενο παίρνει σάρκα και οστά στον χώρο αυτό
  παράδειγμα: *Αυτή η έννοια αιωρείται για δεκαετίες*
- Οι ακροατές και οι συντάκτες εξάγουν το διανοητικό περιεχόμενο από αυτόν τον χώρο
  παράδειγμα: *Ενημέρωσε αν βρεις κάποιες καλές έννοιες σε αυτήν την εργασία.*

4.7 Μεταπρογραμματισμός

Οι υπολογιστές ακολουθούν προγράμματα, σύνολα εντολών σε μια τυπική γλώσσα. Η ανάπτυξη μιας γλώσσας προγραμματισμού ουσιαστικά περιλαμβάνει την χρήση της μεταγλώσσας. Η πράξη της εργασίας με μεταγλώσσες στον προγραμματισμό είναι γνωστή ως μεταπρογραμματισμός. Η Backus – Naur Form αναπτύχθηκε το 1960 από τους John Backus και Peter Naur και είναι μια από τις πρώτες μεταγλώσσες που χρησιμοποιήθηκε στους υπολογιστές. Παραδείγματα σύγχρονων γλωσσών προγραμματισμού οι οποίες συχνά χρησιμοποιούνται στον μεταπρογραμματισμό είναι οι Lisp, m4, και Yacc.
Ουσιαστικά μεταπρογραμματισμός είναι η συγγραφή προγραμμάτων υπολογιστών τα οποία έχουν την ικανότητα να αντιμετωπίζουν τα προγράμματα σαν δεδομένα. Σημαίνει ότι ένα πρόγραμμα μπορεί να σχεδιαστεί ώστε να διαβάζει, παράγει, αναλύει ή μετασχηματίζει άλλα προγράμματα και ακόμα και να τροποποιεί τον εαυτό του κατά την εκτέλεση του. Σε κάποιες περιπτώσεις αυτό

258 Reddy, Michael J. 1979. The conduit metaphor: A case of frame conflict in our language about language. In Andrew Ortony (ed.), ***Metaphor and Thought.*** Cambridge: Cambridge University Press

επιτρέπει στους προγραμματιστές να ελαχιστοποιούν τον αριθμό των γραμμών κώδικα που χρειάζονται για να εκφράσουν μια λύση (και άρα μειώνοντας τον χρόνο ανάπτυξης) ή παρέχει στους προγραμματιστές περισσότερη ευελιξία ώστε να χειριστούν περισσότερο αποτελεσματικά νέες περιπτώσεις και καταστάσεις δίχως την ανάγκη επαναμεταγλώττισης.
Η γλώσσα στην οποία γράφεται το μεταπρόγραμμα ονομάζεται μεταγλώσσα. Οι γλώσσα των προγραμμάτων τα οποία αποτελούν δεδομένα του μεταπρογράμματος αποκαλείται αντικειμενική γλώσσα. Η ικανότητα των γλωσσών προγραμματισμού να είναι οι μεταγλώσσες του εαυτού τους αποκαλείται αντανάκλαση ή αντανακλαστικότητα.
Η αντανάκλαση είναι ένα πολύτιμο χαρακτηριστικό των γλωσσών ώστε να μπορέσουν να υποστηρίξουν τον μεταπρογραμματισμό. Το να έχουμε την ίδια την γλώσσα σαν τύπο δεδομένων πρώτης κλάσης (όπως γίνεται στην Lisp, στην Prolog, την SNOBOL ή την Rebol) είναι επίσης πολύ χρήσιμο, αυτό το χαρακτηριστικό είναι γνωστό ως ομοεικονικότητα (homoiconicity). Ο γενικός προγραμματισμός περιέχει χαρακτηριστικά του μεταπρογραμματισμού μέσα σε μια γλώσσα, στις γλώσσες που το υποστηρίζουν.
Ο μεταπρογραμματισμός συνήθως δουλεύει με έναν από τους τρεις παρακάτω τρόπους. Ο πρώτος τρόπος, είναι η έκθεση των εσωτερικών της μηχανής χρόνου εκτέλεσης, στον κώδικα του προγράμματος μέσω προγραμματιστικών εφαρμογών διεπαφής (APIs). Η Δεύτερη προσέγγιση είναι η δυναμική εκτέλεση εκφράσεων οι οποίες περιέχουν προγραμματιστικές εντολές, οι οποίες συχνά συνθέτονται από συμβολοσειρές, και οι οποίες μπορούν να προέρχονται από άλλες μεθόδους που χρησιμοποιούν ορίσματα ή περιεχόμενα. Έτσι είναι δυνατόν προγράμματα να γράφουν προγράμματα (χαρακτηριστική περίπτωση η συγγραφή δυναμικού κώδικα HTML από server side applications). Αν και οι δύο προσεγγίσεις μπορούν να χρησιμοποιούνται στην ίδια γλώσσα οι περισσότερες γλώσσες τείνουν να χρησιμοποιοιούν είτε την μία είτε την άλλη.
Ο τρίτος τρόπος είναι η ολική έξοδος από την γλώσσα. Συστήματα μετασχηματισμού προγραμμάτων γενικού σκοπού όπως είναι οι μεταγλωττιστές, οι οποίοι δέχονται περιγραφές γλώσσας και μπορούν να εκτελέσουν αυθαίρετους μετασχηματισμούς είναι άμεσες υλοποιήσεις γενικού μεταπρογραμματισμού. Αυτό επιτρέπει στον μεταπρογραμματισμό να βρίσκει εφαρμογή σε κάθε γλώσσα στόχο δίχως να είναι προαπαιτούμενο η γλώσσα να διαθέτει δυνατοτητες μεταπρογραμμστισμού από μόνη της.

Κεφάλαιο 5
Αρχές Αντικειμενοστραφούς Προγραμματισμού

5 Αντικειμενοστραφής προγραμματισμός

5.1 Ορισμός

Ο αντικειμενοστραφής προγραμματισμός (Object Oriented Programming – OOP) είναι προγραμματιστικό υπόδειγμα το οποίο είναι βασισμένο στην έννοια των αντικειμένων τα οποία μπορεί να περιέχουν δεδομένα, στην μορφή πεδίων, συχνά χαρακτηριζόμενα και ως χαρακτηριστικά και κώδικα με τη μορφή διαδικασιών τις οποίες συχνά αποκαλούμε και μεθόδους. Ένα διακριτό γνώρισμα των αντικειμένων είναι ότι οι διαδικασίες ενός αντικειμένου μπορούν να έχουν πρόσβαση και συχνά να τροποποιούν τα δεδομένα των πεδίων του αντικειμένου με το οποίο συνδέονται (τα αντικείμενα έχουν μια έννοια του this ή αλλιώς του εαυτού). Στον αντικειμενοστραφή προγραμματισμό, τα προγράμματα σχεδιάζονται και υλοποιούνται από αντικείμενα τα οποία αλληλεπιδρούν μεταξύ τους προκειμένου να λύσουν ένα πρόβλημα[259][260]. Υπάρχει σημαντική ποικιλομορφία στον αντικειμενοστραφή προγραμματισμό ωστόσο οι πιο συχνά χρησιμοποιούμενες γλώσσες βασίζουν την λειτουργία τους στην έννοια των κλάσεων, με την έννοια ότι τα αντικείμενα δεν είναι τίποτα άλλο από στιγμιότυπα κλάσεων οι οποίες καθορίζουν και τον τύπο τους.

Πολλές από τις πιο συχνά χρησιμοποιούμενες γλώσσες αντικειμενοστραφούς προγραμματισμού είναι ουσιαστικά γλώσσες πολλαπλών προτύπων οι οποίες υποστηρίζουν και τον αντικειμενοστραφή προγραμματισμό σε μεγαλύτερο ή μικρότερο βαθμό, σε συνδυασμό με τον προστακτικό και διαδικασιακό προγραμματισμό. Μερικές σημαντικές γλώσσες αντικειμενοστραφούς προγραμματισμού είναι οι Common Lisp, Python, C++, Objective-C, Smalltalk, Delphi, Java, Swift, C#, Perl, Ruby, και PHP.

5.2 Χαρακτηριστικά

Ο αντικειμενοστραφής προγραμματισμός εξ' ορισμού χρησιμοποιεί αντικείμενα, όμως δεν υποστηρίζονται όλες οι σχετικές τεχνικές και υποδομές της αντικειμενοστρέφειας από όλες τις γλώσσες που ισχυρίζονται ότι υποστηρίζουν των αντικειμενοστραφή προγραμματισμό. Τα χαρακτηριστικά ωστόσο που παραθέτουμε παρακάτω είναι κοινά μεταξύ των γλωσσών οι οποίες θεωρούνται ισχυρά αντικειμενοστραφείς (η πολλαπλών προγραμματιστικών προτύπων με υποστήριξη της αντικειμενοστρέφειας) με τις αξιοπρόσεκτες εξαιρέσεις να αναφέρονται[261][262][263][264].

---

259 *Kindler, E.; Krivy, I. (2011). "Object-Oriented Simulation of systems with sophisticated control". International Journal of General Systems: 313–343.*

260 *Lewis, John; Loftus, William (2008). Java Software Solutions Foundations of Programming Design 6th ed. Pearson Education Inc. ISBN 0-321-53205-8.*, section 1.6 "Object-Oriented Programming"

261 Deborah J. Armstrong. *The Quarks of Object-Oriented Development*. A survey of nearly 40 years of computing literature which identified a number of fundamental concepts found in the large majority of definitions of OOP, in descending order of popularity: Inheritance, Object, Class, Encapsulation, Method, Message Passing, Polymorphism,

5.2.1 Από κοινού χρήση χαρακτηριστικών με τις γλώσσες προκατόχους

Οι γλώσσες αντικειμενοστραφούς προγραμματισμού μοιράζονται χαρακτηριστικά χαμηλού επιπέδου με τις γλώσσες διαδικασιακού προγραμματισμού υψηλού επιπέδου. Τα θεμελιώδη εργαλεία τα οποία χρησιμοποιούνται για την δημιουργία ενός προγράμματος είναι:

- Μεταβλητές οι οποίες αποθηκεύουν πληροφορία μορφοποιημένη σε ένα μικρό αριθμό ενσωματωμένων τύπων δεδομένων όπως είναι οι ακέραιοι, οι χαρακτήρες. Εδώ εντάσσονται και κάποιες βασικές δομές δεδομένων όπως είναι τα αλφαριθμητικά, οι μήτρες, οι λίστες, δομές οι οποίες είτε ενσωματωμένες είτε γίνονται εφικτές χάρη στον συνδυασμό μεταβλητών και δεικτών στην μνήμη
- Διαδικασίες οι οποίες είναι γνωστές και ως συναρτήσεις, μέθοδοι, ρουτίνες ή υπορουτίνες, οι οποίες λαμβάνουν δεδομένα εισόδου, παράγουν δεδομένα εξόδου και χειρίζονται δεδομένα. Οι σύγχρονες γλώσσες βέβαια υποστηρίζουν και τα χαρακτηριστικά του δομημένου προγραμματισμού όπως είναι οι βρόχοι και η σύγκριση καταστάσεων

Η υποστήριξη του αρθρωτού προγραμματισμού παρέχει την δυνατότητα ομαδοποίησης των διαδικασιών σε αρχεία και ενότητες για την εξυπηρέτηση των αναγκών ολόκληρων λειτουργικών μονάδων. Οι ενότητες ονοματοδοτούνται έτσι ώστε ο κώδικας μιας ενότητας να μην συγχέεται με τον κώδικα μιας άλλης ενότητας ή αρχείου.

5.2.2 Αντικείμενα και κλάσεις

Οι γλώσσες που υποστηρίζουν των αντικειμενοστραφή προγραμματισμό χρησιμοποιούν το χαρακτηριστικό της κληρονομικότητας για να πετύχουν την επαναχρησιμοποίηση του κώδικα και για την επεκτασιμότητα του με την μορφή είτε άλλων κλάσεων είτε άλλων προτύπων. Αυτές οι οποίες υποστηρίζουν κλάσεις υποστηρίζουν δύο βασικές έννοιες:

- Κλάσεις – είναι ο ορισμός για την μορφή των δεδομένων και των διαθέσιμων διαδικασιών για ένα δεδομένο τύπος κλάσεις ή αντικειμένου. Μπορεί επιπλέον να περιέχει δεδομένα και διαδικασίες.
- Αντικείμενα – είναι τα στιγμιότυπα των κλάσεων.

Τα αντικείμενα συχνά αντιστοιχούν σε αντικείμενα που συναντούμε στην πραγματικότητα. Για παράδειγμα ένα πρόγραμμα γραφικών μπορεί να έχει αντικείμενα όπως είναι ο κύκλος και το τετράγωνο. Ένα σύστημα online αγορών μπορεί να έχει αντικείμενα όπως είναι το καρότσι αγορών, ο πελάτης και το προϊόν[265]. Μερικές φορές τα αντικείμενα αναπαριστούν περισσότερο αφηρημένες

---

and Abstraction.

262 John C. Mitchell, *Concepts in programming languages*, Cambridge University Press, 2003, ISBN 0-521-78098-5, p.278. Lists: Dynamic dispatch, abstraction, subtype polymorphism, and inheritance.

263 Michael Lee Scott, *Programming language pragmatics*, Edition 2, Morgan Kaufmann, 2006, ISBN 0-12-633951-1, p. 470. Lists encapsulation, inheritance, and dynamic dispatch.

264 *Pierce, Benjamin (2002). Types and Programming Languages. MIT Press. ISBN 0-262-16209-1.*, section 18.1 "What is Object-Oriented Programming?" Lists: Dynamic dispatch, encapsulation or multi-methods (multiple dispatch), subtype polymorphism, inheritance or delegation, open recursion ("this"/"self")

1. 265 *Booch, Grady (1986). Software Engineering with Ada. Addison Wesley. p. 220. ISBN 978-0805306088.* Perhaps the greatest strength of an object-oriented approach to development is that it offers a mechanism

έννοιες όπως ένα αντικείμενο το οποίο αναπαριστά ένα ανοιχτό αρχείο ή το αντικείμενο το οποίο παρέχει την υπηρεσία μετατροπής των μετρήσεων από το ένα μετρικό σύστημα στο άλλο.

Κάθε αντικείμενο όπως είπαμε εκλαμβάνεται ως στιγμιότυπο μιας συγκεκριμένης κλάσης (για παράδειγμα ένα αντικείμενο του οποίου το πεδίο Όνομα έχει την τιμή “Μαρία” μπορεί να είναι ένα στιγμιότυπο της κλάσης “Υπάλληλος”). Οι διαδικασίες στον αντικειμενοστραφή προγραμματισμό είναι γνωστές ως μέθοδοι και οι μεταβλητές ως πεδία, μέλη, χαρακτηριστικά ή ιδιότητες. Ακολουθούν οι ορισμοί:

1. Μεταβλητές κλάσεων – ανήκουν στην κλάση εξ' ολοκλήρου και υπάρχει μόνο ένα αντίγραφο της κάθε μιας.
2. Μεταβλητές περιπτώσεων ή χαρακτηριστικά – πρόκειται για δεδομένα τα οποία ανήκουν σε κάθε ξεχωριστό αντικείμενο κάθε αντικείμενο έχει το δικό του αντίγραφο.
3. Μεταβλητές μέλη – εδώ μιλάμε για μεταβλητές οι οποίες ανήκουν και στην κλάση και στην περίπτωση.
4. Μέθοδοι της κλάσης – πρόκειται για μεθόδους οι οποίες ανήκουν εξ' ολοκλήρου στην κλάση και οι οποίες έχουν πρόσβαση μόνο στα δεδομένα των μεταβλητών της κλάσης
5. Μέθοδοι περιπτώσεων – οι οποίες είναι μέθοδοι οι οποίες ανήκουν σε κάθε αντικείμενο ξεχωριστά και έχουν πρόσβαση στις μεταβλητές περιπτώσης του συγκεκριμένου αντικειμένου από το οποίο καλούνται.

Η πρόσβαση στα αντικείμενα γίνεται περίπου με τον τρόπο που γίνεται και στις μεταβλητές με πολύπλοκη εσωτερική δομή και σε πολλές γλώσσες αντικειμενοστραφείς γλώσσες υπάρχουν αποτελεσματικοί δείκτες οι οποίοι ουσιαστικά παίζουν τον ρόλο της αναφοράς σε ένα συγκεκριμένο στιγμιότυπο μέσα σε μια στοίβα ή ένα σωρό. Παρέχουν ένα επίπεδο αφαίρεσης το οποίο μπορεί να χρησιμοποιηθεί για να διαχωρίσει τον εσωτερικό από τον εξωτερικό κώδικα. Ο εξωτερικός κώδικας μπορεί να χρησιμοποιήσει ένα αντικείμενο, καλώντας ένα συγκεκριμένο στιγμιότυπο μεθόδου με συγκεκριμένο σύνολο παραμέτρων εισόδου, διαβάζοντας τις μεταβλητές ενός στιγμιότυπου ή γράφοντας δεδομένα σε μια μεταβλητή του στιγμιότυπου. Τα αντικείμενα δημιουργούνται με την κλήσε μιας ειδικού τύπου μεθόδου μέσα στην κλάση γνωστής και ως κατασκευαστή. Ένα πρόγραμμα μπορεί να δημιουργήσει πολλά στιγμιότυπα της ίδιας κλάσης καθώς εκτελείται, τα οποία και λειτουργούν ανεξάρτητα μεταξύ τους. Αυτός είναι και ένας εύκολος τρόπος ώστε οι ίδιες διαδικασίες να χρησιμοποιούνται για διαφορετικά σύνολα δεδομένων.

Ο αντικειμενοστραφής προγραμματισμός στον οποίο χρησιμοποιούνται κλάσεις κάποιες φορές καλείται προγραμματισμός με βάση τις κλάσεις ενώ ο προγραμματισμός με βάση τα πρότυπα δεν χρησιμοποιεί κλάσεις. Σαν αποτέλεσμα μια σημαντικά διαφορετική αλλά με αναλογίες ορολογία χρησιμοποιείται για τον ορισμό των εννοιών των αντικειμένων και των στιγμιοτύπων.

Σε μερικές γλώσσες οι κλάσεις και τα αντικείμενα μπορούν να συντεθούν χρησιμοποιώντας και άλλα συστατικά όπως είναι τα traits και τα mixins.

5.2.3 Δυναμική αποστολή/διέλευση μηνυμάτων

Εξ' ορισμού είναι δουλειά του αντικειμένου και όχι του εξωτερικού κώδικα να επιλέξει, με το που θα υπάρξει ανάγκη, την μέθοδο που θα εκτελεστεί σε πραγματικό χρόνο ως αντίδραση σε μια κλήση μεθόδου, ψάχνοντας για την μέθοδο σε πραγματικό χρόνο σε ένα πίνακα συσχετισμένο με το

---

that captures a model of the real world.

αντικείμενο. Αυτό το χαρακτηριστικό είναι γνωστό και ως δυναμική αποστολή και διαχωρίζει ένα αντικείμενο από ένα αφαιρετικό τύπο δεδομένων (ή έννοια), ο οποίος έχει μια δεδομένη (static) υλοποίηση των διαδικασιών για όλα τα στιγμιότυπα. Αν υπάρχουν πολλές μέθοδοι οι οποίες μπορεί να εκτελούνται με την κλήση ενός ονόματος (μηχανισμός ο οποίος απαιτεί και υποστήριξη από την γλώσσα) η δυνατότητα χαρακτηρίζεται ως πολλαπλή αποστολή.
Η κλήση μιας μεθόδου ονομάζεται επίσης και ως διέλευση μηνύματος. Εννοιολογικά ορίζεται σαν ένα μήνυμα (το όνομα της μεθόδου και τις παραμέτρους εισόδου) το οποίο περνάει στο αντικείμενο για αποστολή.

5.2.4 Ενθυλάκωση

Εάν μια κλάση δεν επιτρέπει στον κληθέντα κώδικα από το να έχει πρόσβαση στα εσωτερικά δεδομένα ενός αντικειμένου και επιβάλει την πρόσβαση μόνο μέσω μεθόδων, αυτή η κατάσταση είναι χαρακτηρίζεται ως μια ισχυρή μορφή αφαιρετικότητας ή ως απόκρυψη πληροφορίας γνωστή και ως ενθυλάκωση. Κάποιες γλώσσες (για παράδειγμα η Java), επιτρέπουν στις κλάσεις να επιβάλλουν περιορισμούς πρόσβασης ρητώς, για παράδειγμα η υποδήλωση εσωτερικών δεδομένων με την δεσμευμένη λέξη κλειδί private και η υποδήλωση μεθόδων προορισμένων για χρήση από κώδικα εκτός της κλάσης με την δεσμευμένη λέξη public. Οι μέθοδοι μπορούν επιπλέον να σχεδιάζονται ως public, private ή σε ενδιάμεσα επίπεδα όπως είναι η κατάσταση protected (μια κατάσταση επιτρέπει την πρόσβαση από αντικείμενα της ίδιας κλάσης αλλά όχι από αντικείμενα άλλης κλάσης). Σε άλλες γλώσσες (όπως η Python) κάτι τέτοιο επιβάλλεται μέσω σύμβασης (για παράδειγμα χαρακτηρίζοντας ως private οι οποίες ξεκινούν το όνομα τους με μια κάτω παύλα). Αυτό είναι χρήσιμο διότι εμποδίζει των εξωτερικό κώδικα να ασχολείται με με τις εσωτερικές λειτουργίες ενός αντικειμένου. Κάτι τέτοιο διευκολύνει την επαναπαραγοντοποίηση του κώδικα, επιτρέποντας παράδειγμα στο συγγραφέα μιας κλάσης να αλλάξει το πως τα αντικείμενα αυτής της κλάσης παρουσιάζουν τα εσωτερικά τους δεδομένα δίχως να προβεί σε τροποποιήσεις του εξωτερικού κώδικα (εφόσον οι κλήσεις των public μεθόδων δουλεύουν με τον ίδιο τρόπο). Επίσης ενθαρρύνει τους προγραμματιστές να ενσωματώσουν όλο τον κώδικα, ο οποίος ασχολείται με συγκεκριμένα δεδομένα στο εσωτερικό μιας κλάσης, κάτι το οποίο έχει ως αποτέλεσμα την καλύτερη οργάνωση και άρα κατανόηση από άλλους προγραμματιστές. Η ενθυλάκωση επίσης χρησιμοποιείται ως μια τεχνική η οποία προάγει την αποσύνδεση.

5.2.5 Σύνθεση, κληρονομικότητα και αντιπροσώπευση

Τα αντικείμενα μπορούν να περιέχουν άλλα αντικείμενα στις μεταβλητές στιγμιότυπου. Αυτό είναι γνωστό και σαν σύνθεση αντικειμένων (object composition). Για παράδειγμα ένα αντικείμενο της κλάσης Employee μπορεί να περιέχει (ή να δείχνει) ένα αντικείμενο της κλάσης Address, πέραν των υπολοίπων μεταβλητών του όπως first_name, και position. Η σύνθεση αντικειμένων χρησιμοποιείται για να αναπαραστήσει σχέσεις, κάθε υπάλληλος έχει μια διεύθυνση και άρα κάθε αντικείμενο Employee πρέπει να έχει χώρο για να αποθηκεύσει ένα αντικείμενο Address.
Οι γλώσσες οι οποίες υποστηρίζουν τις κλάσεις σχεδόν πάντα υποστηρίζουν και την κληρονομικότητα. Αυτό επιτρέπει στις κλάσεις να τακτοποιούνται με μια ιεραρχία η οποία παρουσιάζει και είναι ένας τύπος συσχετισμού. Για παράδειγμα η κλάση Employee μπορεί να

κληρονομεί από την κλάση Person. Όλα τα δεδομένα και οι μέθοδοι που είναι διαθέσιμα για την κλάση γονιό εμφανίζονται και στην κλάση παιδί με τα ίδια ονόματα. Για παράδειγμα η κλάση Person μπορεί να ορίζει τις παρακάτω μεταβλητές, first_name και last_name και την μέθοδο make_full_name(). Αυτά τα χαρακτηριστικά θα είναι επίσης διαθέσιμα στην κλάση Employee η οποία μπορεί επίσης να προσθέσει και τις μεταβλητές postition και salary. Αυτή η τεχνική επιτρέπει την εύκολη επαναχρησιμοποίηση των ίδιων διαδικασιών και ορισμών ενώ επιπρόσθετα αντικατοπτρίζει και σχέσεις του πραγματικού κόσμου με ένα ενστικτώδη τρόπο. Αυτές οι κλάσεις και οι υποκλάσεις εντάσσονται και στην λογική των συνόλων και των υποσυνόλων της άλγεβρας. Αντί να αξιοποιεί πίνακες βάσεων δεδομένων και υπορουτίνες προγραμμάτων, ο προγραμματιστής αξιοποιεί αντικείμενα με τα οποία ο χρήστης μπορεί να έχει περισσότερη εξοικείωση, αντικείμενα από το πεδίο ορισμού της εφαρμογής[266].

Οι υποκλάσεις μπορούν να παρακάμψουν τις μεθόδους που έχουν οριστεί στην υπερκλάση τους. Η πολλαπλή κληρονομικότητα επιτρέπεται σε κάποιες γλώσσες αν και κάτι τέτοιο μπορεί να κάνει την διαδικασία παράκαμψης του γονικού κώδικα αρκετά περίπλοκη. Μερικές γλώσσες έχουν ειδική μνεία για μίξεις αν και σε κάθε γλώσσα με υποστήριξη πολλαπλής κληρονομικότητας μια μίξη είναι απλά μια κλάση η οποία δεν αντιπροσωπεύει και δεν είναι ένας τύπος συσχετισμού. Για παράδειγμα, η κλάση UnicodeConversionMixin, θα μπορούσε να παρέχει μια μέθοδο unicode_to_ascii() όταν συμπεριλαμβάνεται στην κλάση FileReader και στην κλάση WebPageScraper, η οποίες δεν μοιράζονται ένα κοινό γονέα.

Οι κλάσεις αφαίρεσης (abstract classes) δεν μπορούν να αρχικοποιηθούν σε αντικείμενα. Υπάρχουν μόνο για να εξυπηρετούν τους σκοπούς της κληρονομικότητας σε άλλες συμπαγείς κλάσεις οι οποίες μπορούν να αρχικοποιηθούν. Στην Java η δεσμευμένη λέξη final μπορεί να χρησιμοποιηθεί για να αποτρέψει μια κλάση από το να έχει υποκλάσεις.

Το δόγμα της σύνθεσης αντί της κληρονομικότητας μάχεται υπέρ της υλοποίησης ενός τύπου σχέσεως χρησιμοποιώντας την σύνθεση αντί για την κληρονομικότητα. Για παράδειγμα η κλάση Employee αντί να κληρονομήσει από την κλάση Person, θα μπορούσε να δώσει σε κάθε αντικείμενο τύπου Employee ένα εσωτερικό αντικείμενο Person το οποίο θα είχει τότε την ευκαιρία να κρυφτεί από τον εξωτερικό κώδικα ακόμη και αν μια κλάση Person έχει πολλαπλά χαρακτηριστικά ή μεθόδους. Κάποιες γλώσσες, όπως η Go δεν υποστηρίζουν καθόλου την έννοια της κληρονομικότητας.

Η αρχή του ανοίγματος/κλεισίματος ισχυρίζεται ότι οι κλάσεις και οι μέθοδοι θα πρέπει να είναι ανοιχτές για επέκταση αλλά κλειστές για τροποποίηση.

Η αντιπροσώπευση είναι ένα άλλο χαρακτηριστικό της γλώσσας το όποίο μπορεί να χρησιμοποιηθεί ως μια εναλλακτική μορφή κληρονομικότητας.

5.2.6 Πολυμορφισμός

Η υποτυποποίηση, μια μορφή πολυμορφισμού, είναι όταν κατά την κλήση ενός αντικειμένου ο έλεγχος προγράμματος δεν έχει αίσθηση για το αν το αντικείμενο αυτό ανήκει σε μια κλάση γονέα ή στους απογόνους της. Για παράδειγμα, μια μέθοδος καλεί την make_full_name() σε ένα αντικείμενο πάνω στο οποίο και οι δύο θα δουλέψουν είτε το αντικείμενο ανήκει στην κλάση

266 *Jacobsen, Ivar; Magnus Christerson; Patrik Jonsson; Gunnar Overgaard (1992). Object Oriented Software Engineering. Addison-Wesley ACM Press. pp. 43–69. ISBN 0-201-54435-0.*

Person είτε στην κλάση Employee. Πρόκειται ουσιαστικά για άλλη μια μορφή αφαίρεσης η οποία απλοποιεί τον εξωτερικό στην ιεραρχία της κλάσης κώδικα και υποστηρίζει τον ισχυρό διαχωρισμό των μελημάτων.

5.2.7 Ανοιχτή αναδρομή

Στις γλώσσες οι οποίες υποστηρίζουν την ανοιχτή αναδρομή (open recursion) οι μέθοδοι των αντικειμένων μπορούν να καλέσουν άλλες μεθόδους στο ίδιο αντικείμενο (συμπεριλαμβνομένων και του εαυτού τους), συνήθως όταν χρησιμοποιούν μια μεταβλητή ή μια δεσμευμένη λέξη την this ή την self ανάλογα με την γλώσσα. Αυτή μεταβλητή είναι καθυστερημένα δεσμευμένη. Επιτρέπει σε μια μέθοδο που ορίζεται σε μια κλάση να καλέσει μια μέθοδο η οποία ορίζεται αργότερα σε κάποια υποκλάση αυτής.

5.3 Ιστορική αναδρομή

Ο επίκληση των όρων αντικείμενο και προσανατολισμός με την σύγχρονη έννοια του αντικειμενοστραφούς προγραμματισμού έκανε την πρώτη της εμφάνιση στο MIT στα τέλη της δεκαετίας του 50 και στις αρχές του 60. Στο περιβάλλον της ομάδας της τεχνητής νοημοσύνης στις αρχές του 60 ως αντικείμενα θεωρούσαν τα αναγνωρισμένα αντικείμενα (atoms στην ορολογία της LISP) με ιδιότητες (χαρακτηριστικά)[267][268]. Ο Alan Kay αργότερα παρέθεσε το πως η λεπτομερής κατανόηση των εσωτερικών διαδικασιών της LISP επιρέασε τον τρόπο σκέψης του το 1966[269]. Ένα άλλο πρώιμο παράδειγμα από το MIT ήταν η Sketchpad, δημιούργημα του Ivan Sutherland το 1960-61. Στο γλωσσάρι της τεχνικής αναφοράς του 1963 η οποία βασίστηκε στην διατριβή του γύρω από την Sketchpad, ο Sutherland όρισε της έννοιες του αντικειμένου και του στιγμιότυπου (με την έννοια της κλάσης να καλύπτεται από τον ορισμό master)[270]. Επίσης μια έκδοση του MIT για την ALGOL η AED-0 καθιέρωσε ένα άμεσο σύνδεσμο ανάμεσα στις δομές δεδομένων (plexes στην διάλεκτο αυτή) και των διαδικασιών σκιαγραφώντας ουσιαστικά αυτό που αργότερα ονομάστηκε μηνύματα, μέθοδοι και μέθοδοι μέλη[271][272].

Η επίσημη έννοια των προγραμματιστικών αντικειμένων εισήχθη το 1961 στην Simula 67, μια εκ βάθρων αναθεώρηση της Simula 1, μια γλώσσα προγραμματισμού σχεδιασμένη για προσομοίωση διακριτών συμβάντων, η οποία φτιάχθηκε από τους Ole-Johan Dahl και Kristen Nygaard του

---

267 *McCarthy, J.; Brayton, R.; Edwards, D.; Fox, P.; Hodes, L.; Luckham, D.; Maling, K.; Park, D.; Russell, S. (March 1960). "LISP I Programmers Manual" (PDF). Boston, Massachusetts: Artificial Intelligence Group, M.I.T. Computation Center and Research Laboratory: 88f.* In the local M.I.T. patois, association lists [of atomic symbols] are also referred to as "property lists", and atomic symbols are sometimes called "objects".

268 *McCarthy, John; Abrahams, Paul W.; Edwards, Daniel J.; Hart, swapnil d.; Levin, Michael I. (1962). LISP 1.5 Programmer's Manual (PDF). MIT Press. p. 105. ISBN 0-262-13011-4.* Object — a synonym for atomic symbol

269 *"Dr. Alan Kay on the Meaning of "Object-Oriented Programming"". 2003. Retrieved 11 February 2010.*

270 *Sutherland, I. E. (30 January 1963). "Sketchpad: A Man-Machine Graphical Communication System" (PDF). Technical Report No. 296, Lincoln Laboratory, Massachusetts Institute of Technology via Defense Technical Information Center (stinet.dtic.mil). Retrieved 3 November 2007.*

271 The Development of the Simula Languages, Kristen Nygaard, Ole-Johan Dahl, p.254 Uni-kl.ac.at

272 *Ross, Doug. "The first software engineering language". LCS/AI Lab Timeline:. MIT Computer Science and Artificial Intelligence Laboratory. Retrieved 13 May 2010.*

Norwegian Computing Center στο Όσλο[273]. [274][275]Η Simula 67 επηρεάστηκε από τη SIMSCRIPT και την C.A.R. του Tony Hoares και πρότεινε τις εγγραφές και τις κλάσεις. Η Simula λοιπόν εισήγαγε τις έννοιες της κλάσης, του στιγμιότυπου ή αντικειμένου (όπως επίσης και της υποκλάσης, της εικονικής διαδικασίας, της συνδιεργασίας, και την προσομοίωση διακριτών συμβάντων) σαν ένα παράδειγμα του σαφούς προγραμματισμού. Η γλώσσα χρησιμοποιούσε επίσης και αυτόματο καθαρισμό της μνήμης μια δυνατότητα η οποία είχε εφευρεθεί και εφαρμοσθεί νωρίτερα για την γλώσσα λειτουργικού προγραμματισμού Lisp. Η Simula χρησιμοποιήθηκε για αναπαράσταση φυσικών μοντέλων, για παράδειγμα μοντέλα προς μελέτη με σκοπό την βελτίωση του τρόπου της κίνησης των πλοίων και του φορτίου τους μεταξύ μέσω των λιμανιών. Η ιδέες και οι καινοτομίες της Simula 67 επηρέσαν πολλές μεταγενέστερες γλώσσες κάποιες από τις οποίες είναι οι Smaltalk, διαμορφώσεις της Lisp (CLOS), Object Pascal και C++.

Η γλώσσα Smaltalk, η οποία αναπτύχθηκε στο Xerex PARC από τον Alan Kay (και άλλους) στην δεκαετία του 70, εισήγαγε την έννοια του αντικειμενοστραφούς προγραμματισμού για να χαρακτηρίσει την διάχυτη χρήση των αντικειμένων και των μηνυμάτων σαν υπολογιστική βάση. Οι δημιουργοί της Smaltalk επηρεάστηκαν από τις ιδέες που εισηγήθηκε η Simula 67 αλλά η Smaltalk σχεδιάστηκε ως ένα πλήρως δυναμικό σύστημα στο οποίο οι κλάσεις μπορούσαν να δημιουργηθούν και να τροποποιηθούν δυναμικά και όχι στατικά όπως γινόταν στην Simula 67[276]. Η Smalttalk και μαζί της ο αντικειμενοστραφής προγραμματισμός παρουσιάστηκαν στο ευρύτερο κοινό των Αύγουστο του 1981 από το περιοδικό Byte.

Στην δεκαετία του 70 η Smaltalk του Kay επηρέασε την κοινότητα των προγραμματιστών της Lisp και την ώθησε στην υιοθέτηση των αντικειμενοστραφών τεχνικών οι οποίες παρουσιάστηκαν στους προγραμματιστές μέσω της Lisp machine. Πειραματισμοί με διάφορες προεκτάσεις της Lisp (όπως ήταν η LOOPS και η Favors οι οποίες εισηγήθηκαν την πολλαπλή κληρονομικότητα και τα mixins) οδήγησαν τελικά στο Common Lisp Object System, το οποίο ενσωματώνει των διαδικασιακό προγραμματισμό και τον αντικειμενοστραφή προγραμματισμό και επιτρέπει την επέκταση μέσω του πρωτοκόλλου Meta-object. Στη δεκαετία του 80 υπήρξαν κάποιες προσπάθειες για τον σχεδιασμό αρχιτεκτονικών επεξεργαστών οι οποίες θα περιελάμβαναν την υποστήριξη του υλικού για αντικείμενα στην μνήμη δίχως να στεφθούν από επιτυχία.Τέτοια παραδείγματα είναι οι Intel iAPX 432 και η Linn Smart Rekursiv.

Το 1985 ο Bertrand Meyer παρήγαγε το πρώτο σχέδιο της γλώσσας Eiffe. Εστιασμένη στην ποιότητα του λογισμικού η Eiffe είναι μια από τις απολύτως αντικειμενοστραφείς γλώσσες αλλά διαφέρει με την έννοια ότι η ίδια η γλώσσα δεν είναι απλώς μια γλώσσα προγραμματισμού αλλά ένας συμβολισμός υποστήριξης ολόκληρου του κύκλου ζωής του λογισμικού. Ο Meyer περιγράφει την Eifel σαν μια μέθοδο ανάπτυξης λογισμικού βασισμένη σε ένα σύνολο καινοτόμων ιδεών από τις επιστήμες της μηχανικής λογισμικού και της επιστήμης υπολογιστών για τον αντικειμενοστραφή προγραμματισμό. Θεμελιώδης για την εστίαση στην ποιότητα της Eifel είναι ο μηχανισμός αξιοπιστίας του Meyer ο οποίος ονομάζεται Design by Contract ο οποίος είναι ένα εσωτερικό κομμάτι τόσο της μεθόδου όσο και της γλώσσας.

---

273 *Holmevik, Jan Rune (1994). "Compiling Simula: A historical study of technological genesis" (PDF). IEEE Annals of the History of Computing 16 (4): 25–37. doi:10.1109/85.329756. Retrieved 12 May 2010.*

274 The Development of the Simula Languages, Kristen Nygaard, Ole-Johan Dahl, p.254 Uni-kl.ac.at

275 *Hoare, C. A. (Nov 1965). "Record Handling". ALGOL Bulletin (21): 39–69. doi:10.1145/1061032.1061041.*

276 *Kay, Alan. "The Early History of Smalltalk". Retrieved 13 September 2007.*

Ο αντικειμενοστραφής προγραμματισμός εξελίχτηκε στην κυρίαρχη προγραμματιστική τάση στις αρχές και τα μέσα της δεκαετίας του 1990, όταν οι γλώσσες που υποστήριζαν τις τεχνικές του έγιναν διαθέσιμες ευρέως. Τέτοιες γλώσσες ήταν οι Visual FoxPro 3.0[277][278][279] η C++[280] και η Delphi. Η κυριαρχία του στην ανάπτυξη λογισμικού αυξήθηκε ακόμη περισσότερο με την αύξηση της δημοτικότητας των γραφικών περιβαλλόντων διεπαφής τα οποία βασίζονται σε πολύ μεγάλο βαθμό στις τεχνικές του αντικειμενοστραφούς προγραμματισμού. Ένα παράδειγμα στενής συσχέτισης μεταξύ μιας δυναμικής βιβλιοθήκης γραφικού περιβάλλοντος διεπαφής και αντικειμενοστραφούς προγραμματισμού μπορούμε να δούμε στα πλαίσια εργασίας Cocoa του Mac OS X, που είναι γραμμένα σε Objective-C, μια αντικειμενοστραφής επέκτεση μυνημάτων της C βασισμένη στην Smaltalk. Οι εργαλειοθήκες του αντικειμενοστραφούς προγραμματισμού επίσης αύξησαν την δημοτικότητα των προγραμμάτων αντίδρασης σε συμβάντα (αν και η έννοια αυτή δεν περιορίζεται απλά στην αντικειμενοστρέφεια).

Στο ETH της Ζυρίχης ο Niklaus Wirth και οι συνάδελφοι του ερευνούσαν επίσης ζητήματα σαν την αφαίρεση και τον τμηματικό προγραμματισμό (αν και αυτός θεωρείτο κοινοτοπία από το 1960 ή ίσως και νωρίτερα). Η Moduls-2 (1978) περιέκλειε και τις δύο έννοιες και το σχέδιο το οποίο προέκυψε από αυτήν την έρευνα το Oberon περιέκλειε μια ενστικτώδη προσέγγιση τόσο της αντικειμενοστρέφειας όσο και των κλάσεων.

Τα χαρακτηριστικά του αντικειμοναστραφούς προγραμματισμού προστέθηκαν σε πολλές από τις γλώσσες που προϋπήρχαν όπως η Ada, Basic, Fortran, Pascal, και COBOL. Η προσθήκη αυτών των χαρακτηριστικών σε γλώσσες που δεν είχαν εξ' αρχής σχεδιαστεί ως αντικειμενοστραφείς συχνά οδήγησε σε προβλήματα συμβατότητας και συντήρησης του κώδικα.

Πιο πρόσφατα έκανε την εμφάνιση του ένας αριθμός από γλώσσες οι οποίες ήταν σχεδιασμένες κυρίως για να υποστηρίζουν τον αντικειμενοστραφή προγραμματισμό και τις τεχνικές του. Δύο τέτοιες γλώσσες είναι οι Python και Ruby. Πιθανόν η πλέον σημαντικές εμπορικές αντικειμενοστραφείς γλώσσες από τις πρόσφατες είναι οι Java ,που φτιάχθηκε από τη Sun Microsystems όπως επίσης η C# και η Visual Basic, .NET (VB, .NET) φτιαγμένες και οι δύο για την πλατφόρμα .NET της Microsoft. Κάθε ένα από αυτά τα δύο πλαίσια εργασίας δείχνει με τον τρόπο του τα οφέλη της χρήσης του αντικειμενοστραφούς προγραμματισμού με την δημιουργία μιας αφαίρεσης κατά την υλοποίηση. Η VB.NET και η C# υποστηρίζουν διαγλωσσική κληρονομικότητα επιτρέποντας σε κλάσεις οι οποίες ορίστηκαν στη μια γλώσσα να κληρονομήσουν κλάσεις οι οποίες ορίστηκαν στην άλλη.

5.4 Γλώσσες αντικειμενοστραφούς προγραμματισμού

Simula (1967) είναι γενικά αποδεκτή ως η πρώτη γλώσσα με τα βασικά χαρακτηριστικά αντικειμενοστρέφειας. Δημιουργήθηκε αρχικά για την δημιουργία προγραμμάτων προσομοίωσης στα οποία αποκαλούσαμε ως αντικείμενα τις πλέον σημαντικές πληροφορίες παρουσίασης. Η

277 1995 (June) Visual FoxPro 3.0, FoxPro evolves from a procedural language to an object-oriented language. Visual FoxPro 3.0 introduces a database container, seamless client/server capabilities, support for ActiveX technologies, and OLE Automation and null support. Summary of Fox releases

278 FoxPro History web site: Foxprohistory.org

279 1995 Reviewers Guide to Visual FoxPro 3.0: DFpug.de

280 https://books.google.co.uk/books?id=MHmqfSBTXsAC&pg=PA16&lpg=PA16

Smaltalk (1972 έως 1980) είναι ένα πρώιμο παράδειγμα και αυτό με το οποίο το μεγαλύτερο μέρος της θεωρίας του αντικειμενοστραφούς προγραμματισμού αναπτύχθηκε. Λαμβάνοντας υπ' όψιν τον βαθμό της αντικειμενοστρέφειας μπορούν να γίνουν οι παρακάτω διαχωρισμοί:

- Οι γλώσσες οι οποίες αποκαλούνται αμιγώς αντικειμενοστραφείς γλώσσες γιατί τα πάντα μέσα σε αυτές αντιμετωπίζονται ως αντικείμενα, από βασικού τύπους όπως οι χαρακτήρες και τα σημεία της στίξης μέχρι τις κλάσεις, τα πρότυπα, τα block, οι έννοιες κλπ. Σχεδιάστηκαν με σκοπό να διευκολύνουν ή ακόμα και να υποστηρίξουν τις μεθόδους του αντικειμονοστραφούς προγραμματισμού. Παραδείγματα: Eiffel, Emerald[281], JADE, Obix, Ruby, Scala, Smaltalk, Self.
- Γλώσσες οι οποίες σχεδιάστηκαν με γνώμονα τον αντικειμενοστραφή προγραμματισμό αλλά εμπεριέχουν και αρκετά στοιχεία του διαδικασιακού. Παραδείγματα: Delphi/Object Pascal, C++, Java, C#, VB.NET.
- Γλώσσες οι οποίες είναι ιστορικά γλώσσες διαδικασιακού προγραμματισμού οι οποίες όμως επεκτάθηκαν με ορισμένα στοιχεία αντικειμενοστρέφειας. Παραδείγματα: Pascal, Visual Basic, MATLAB, Fortran, Perl, COBOL 2002, PHP, ABAP, Ada 95.
- Γλώσσες με τα περισσότερα χαρακτηριστικά της αντικειμενοστρέφειας αλλά σε αρχική μορφή. Παραδείγματα: Oberon (Oberon-1, Oberon-2).
- Γλώσσες με υποστήριξη αφαίρεσης τύπων οι οποίες μπορούν να χρησιμοποιηθούν στον αντικειμενοστραφή προγραμματισμό αλλά χωρίς να έχουν υλοποιημένα όλα τα στοιχεία της αντικειμενοστρέφειας. Στην κατηγορία αυτή περικλείονται και οι βασισμένες στο αντικείμενο και οι βασισμένες στο πρωτότυπο γλώσσες. Παραδείγματα: Modula-2, Pliant, CLU, JavaScript, Lua.
- Γλώσσες χαμαιλέοντες οι οποίες υποστηρίζουν διάφορες τεχνοτροπίες μεταξύ των οποίων και η αντικειμενοστρέφεια. Η Tcl ξεχωρίζει από αυτές για την TclOO, ένα υβρίδικό αντικειμενοστραφές σύστημα το οποίο υποστηρίζει τόσο προγραμματισμό βασισμένο στο πρώτυπο όσο και στην κλάση.

5.5 Η αντικειμενοστρέφεια στις δυναμικές γλώσσες

Στα πρόσφατα χρόνια ο αντικειμενοστραφής προγραμματισμός έγινε ιδιαίτερα δημοφιλής στις δυναμικές γλώσσες προγραμματισμού όπως είναι οι Python, PowerShell, Ruby και η Groovy οι οποίες είναι δυναμικές γλώσσες βασισμένες στις αρχές της αντικειμενοστρέφειας, ενώ η Perl και η PHP απλά προσθέτουν αντικειμενοστραφή στοιχεία (η Perl από την έκδοση 5, η PHP από την έκδοση 4 και η ColdFusion από την έκδοση 6).

Το μοντέλο αντικειμένου στις HTML, XHTML, XML έχει δεσμούς με την δημοφιλή JavaScript/ECMAScript, η JavaScript είναι ίσως η πλέον γνωστή γλώσσα προγραμματισμού βασισμένου σε πρωτότυπα, μιας τεχνικής που περιλαμβάνει την κλωνοποίηση από πρωτότυπα αντί της κληρονομικότητας από μια κλάση (αντίθετα με τον προγραμματισμό που βασίζεται σε κλάσεις). Πριν την ECMAScript 6, μόνο ένα μοντέλο προγραμματισμού βασισμένο σε πρωτότυπα υποστηριζόταν. Μια άλλη Script γλώσσα που υποστηρίζει αυτή την προσέγγιση είναι η Lua.

281 *"The Emerald Programming Language". 2011-02-26.*

5.6 Ο αντικειμενοστραφής προγραμματισμός σε ένα πρωτόκολλο δικτύου

Τα μηνύματα τα οποία ρέουν μεταξύ υπολογιστών για να αιτηθούν υπηρεσίες σε ένα περιβάλλον πελάτη – εξυπηρετητή μπορούν να σχεδιαστούν σαν τις γραμμικοποιήσεις των αντικειμένων που ορίζονται από τις κλάσεις αντικείμενα γνωστά τόσο στον πελάτη όσο και στον εξυπηρετητή. Για παράδειγμα ένα απλό γραμμικό αντικείμενο θα αποτελείτο από ένα πεδίο μήκους, ένα σημείο κωδικού που θα αναγνωρίζει την κλάση και μια τιμή δεδομένων. Ένα περισσότερο πολύπλοκο παράδειγμα θα ήταν μια εντολή το οποίο θα περιείχε το μήκος, το σημείο του κωδικού της εντολής και τιμές αποτελούμενες από γραμμικοποιημένα αντικείμενα τα οποία θα αναπαριστούν τις παραμέτρους της εντολής. Κάθε τέτοια εντολή πρέπει να κατευθυνθεί από τον εξυπηρετητή σε ένα αντικείμενο του οποίου η κλάση (ή υπερκλάση) αναγνωρίζει την εντολή και είναι ικανό να παρέχει την υπηρεσία που ζητήθηκε. Οι πελάτες και οι εξυπηρετητές μοντελοποιούνται καλύτερα σαν πολύπλοκες αντικειμενοστραφείς δομές. Η κατανεμημένη αρχιτεκτονική διαχείρισης δεδομένων (Distributed Data Managment Architecture – DDM) πήρε αυτή την προσέγγιση και χρησιμοποίησε αντικείμενα κλάσεων για να ορίσει αντικείμενα σε τέσσερα επίπεδα μορφοποιημένης ιεραρχίας:

- Πεδία που ορίζουν τις τιμές των δεδομένων που σχηματίζουν τα μηνύματα, όπως είναι το μήκος τους, το σημείο κωδικού και οι τιμές των δεδομένων.
- Αντικείμενα και συλλογές αντικειμένων παρόμοιες με αυτά που θα βρίσκαμε σε ένα πρόγραμμα Smaltalk για μηνύματα και παραμέτρους.
- Διαχειριστές παρόμοιοι με τα αντικείμενα AS/400 όπως είναι οι κατάλογοι για τα αρχεία και τα αρχεία τα οποία περιέχουν metadata και εγγραφές. Οι διαχειριστές εννοιολογικά παρέχουν τους πόρους μνήμης και επεξεργαστικής ισχύος για τα αντικείμενα που περιέχουν.
- Ένας πελάτης η εξυπηρετητής ο οποίος περιέχει όλους τους απαραίτητους διαχειριστές για την υλοποίηση ενός πλήρους επεξεργαστικού περιβάλλοντος, υποστηρίζοντας θέματα όπως υπηρεσίες καταλόγου, ασφάλειας και ελέγχου συγχρονισμού.

Η αρχική έκδοση του DDM ορισμένων κατανεμημένων υπηρεσιών αρχείων αργότερα επεκτάθηκε στο θεμέλιο της κατανεμημένης σχεσιακής αρχιτεκτονικής βάσεων δεδομένων (Distributed Relational Database Architecture DRDA).

5.7 Πρότυπα σχεδίασης

Οι προκλήσεις του αντικειμενοστραφούς σχεδίου αντιμετωπίζονται με διάφορες μεθοδολογίες. Η περισσότερο κοινή είναι γνωστή σαν design patterns codified by Gamma et al. Ακόμα ευρύτερα ο όρος πρότυπα σχεδίασης μπορεί να χρησιμοποιηθεί για να αναφερθούμε σε οποιαδήποτε γενικά επαναλαμβανόμενη λύση σε ένα συχνά παρατηρούμενο πρόβλημα στον σχεδιασμό λογισμικού. Μερικά από αυτά τα συχνά παρατηρούμενα προβλήματα έχουν επιπτώσεις και λύσεις ιδιαιτέρως στην αντικειμενοστραφή ανάπτυξη.

5.8 Κληρονομικότητα και συμπεριφορά υποτύπων

Μπορούμε ενστικτωδώς να υποθέσουμε ότι η κληρονομικότητα παράγει ένα ή είναι ένας σημασιολογικός συσχετισμός και άρα να συμπαιράνουμε ότι τα αντικείμενα τα οποία αρχικοποιούνται από υποκλάσεις μπορούν πάντα ασφαλώς να χρησιμοποιηθούν σε αντίθεση με

αυτά τα οποία αρχικοποιούνται από μια υπερκλάση. Αυτή η διαίσθηση δυστυχώς είναι εσφαλμένη στις περισσότερες αντικειμενοστραφείς γλώσσες και ιδιαιτέρως σε όλες αυτές οι οποίες επιτρέπουν ευμετάβλητα αντικείμενα. Ο πολυμορφισμός υποτύπων όπως επιβάλλεται από τον ελεγκτή τύπων στις αντικειμενοστραφείς γλώσσες (με ευμετάβλητα αντικείμενα) δεν μπορεί να εγγυηθεί την υποτυποποίηση συμπεριφοράς σε κάθε περιεχόμενο. Η υποτυποποίηση συμπεριφοράς είναι δύσκολο να αποφασιστεί γενικά και έτσι δεν μπορεί να υλοποιηθεί από ένα πρόγραμμα (compiler). Η ιεραρχία αντικειμένων ή κλάσεων πρέπει να σχεδιάζεται πολύ προσεκτικά λαμβάνοντας υπόψιν πιθανές εσφαλμένες χρήσεις οι οποίες δεν θα γίνουν αντιληπτές συντακτικά. Το ζήτημα είναι γνωστό ως η αρχή αντικατάστασης του Liskov.

5.9 Σχεδιαστικά πρότυπα

Το βιβλίο με τίτλο “Πρότυπα σχεδίασης: Στοιχεία επαναχρησιμοποιούμενου αντικειμοναστραφούς κώδικα” είναι ένα βιβλίο με μεγάλη επιρροή το οποίο εκδόθηκε το 1995 γραμμένο από τους Rich Gamma, Richard Helm, Ralph Johnson και John Vlissides οι οποίοι συχνά αναφέρονται χιουμοριστικά ως η συμμορία των τεσσάρων. Μαζί με την εξερεύνηση των δυνατοτήτων αλλά και των παγίδων του αντικειμενοστραφούς προγραμματισμού, περιγράφονται σε αυτό το βιβλίο και 23 κοινά προγραμματιστικά προβλήματα και πρότυπα επίλυσης του. Από τον Απρίλιο του 2007 το βιβλίο είχε εκδοθεί 36 φορές.
Το βιβλίο περιγράφει τα ακόλουθα πρότυπα:

- Creational patterns: Factory method pattern, Abstract factory pattern, Singleton pattern, Builder pattern, Prototype pattern
- Structural patterns : Adapter pattern, Bridge pattern, Composite pattern, Decorator pattern, Facade pattern, Flyweight pattern, Proxy pattern
- Behavioral patterns: Chain-of-responsibility pattern, Command pattern, Interpreter pattern, Iterator pattern, Mediator pattern, Memento pattern, Observer pattern, State pattern, Strategy pattern, Template method pattern, Visitor pattern

5.10 Αντικειμενοστρέφεια και βάσεις δεδομένων

Από κοινού και ο αντικειμενοστραφείς προγραμματισμός και τα σχεσιακά συστήματα βάσεων δεδομένων (RDBMSs) είναι ιδιαιτέρως κοινά στην σύγχρονη παραγωγή λογισμικού. Μιας και οι σχεσιακές βάσεις δεδομένων δεν αποθηκεύουν απευθείας αντικείμενα (αν κάποια από τα σύγχρονα συστήματα διαθέτουν πλέον αντικειμενοστραφή χαρακτηριστικά προκειμένου να προσεγγίσουν το θέμα) υπάρχει μια γενική ανάγκη οι δύο κόσμοι να γεφυρωθούν. Το μεγάλο πρόβλημα που προκύπτει από το γεφύρωμα, των προσβάσεων και των προτύπων δεδομένων του αντικειμενοστραφούς προγραμματισμού με τις σχεσιακές βάσεις δεδομένων είναι γνωστό ως αντίσταση συμβατότητας αντικειμένου σχέσης (object-relational impedance mismatch). Υπάρχουν διάφορες προτεινόμενες προσεγγίσεις για την επίλυση αυτού του προβλήματος αλλά καμιά γενική λύση δίχως μειονεκτήματα[282]. Μια από τις πιο κοινές προσεγγίσεις είναι η χαρτογράφηση αντικειμένου σχέσης (object-relational mapping) όπως την βρίσκουμε σε IDE γλώσσες όπως η

282 *Neward, Ted (26 June 2006). "The Vietnam of Computer Science". Interoperability Happens. Retrieved 2 June 2010.*

VisualFoxPro και σε βιβλιοθήκες όπως η Java Data Objects και Runy on Rail' s Active Rcord.
Υπάρχουν επίσης βάσεις αντικειμένων οι οποίες μπορούν να χρησιμοποιηθούν για να αντικαταστήσουν το RDBMSs αλλά αυτές δεν είναι τεχνικά αλλά και εμπορικά το επιτυχημένες όπως τα RDBMSs.

5.11 Μοντελοποίηση στον πραγματικό κόσμο και σχέσεις

Ο αντικειμενοστραφής προγραμματισμός μπορεί να χρησιμοποιηθεί ώστε να συγκεράσει τα αντικείμενα και τις διαδικασίες του πραγματικού κόσμου με τα ψηφιακά τους ομόλογα. Ωστόσο δεν συμφωνούν όλοι ότι ο αντικειμενοστραφείς προγραμματισμός διευκολύνει την άμεση χαρτογράφηση του πραγματικού κόσμου ή ότι η χαρτογράφηση του πραγματικού κόσμου είναι ένας στόχος που αξίζει τον κόπο. Ο Bertrand Meyer επιχειρηματολογεί στο “Object Oriented Software Construction”[283] υπέρ του ότι ένα πρόγραμμα δεν είναι ένα μοντέλου του πραγματικού κόσμου αλλά μοντέλο ένός τμήματος του κόσμου: “Reality is a cousin twice removed”. Την ίδια στιγμή κάποιοι περιορισμοί αρχής του αντικειμενοστραφούς προγραμματισμού έχουν γίνει αντιληπτοί[284]. Για παράδειγμα το πρόβλημα κύκλου έλλειψης είναι δύσκολο να το διαχειριστεί κάποιος χρησιμοποιώντας την έννοια της κληρονομικότητας του αντικειμενοστραφούς προγραμματισμού.
Ωστόσο ο Niklaus Wirth (ο οποίος διέδωσε το γνωμικό που πλέον είναι γνωστό σαν ο νόμος του Wirth: “Το λογισμικό γίνεται αργότερο γρηγορότερα από ότι το υλικό γίνεται γρογορότερο”) είπε για τον αντικειμενοστραφή προγραμματισμό σε μια δημοσιευσή του; “Καλές ιδέες μέσα από τον καθρέφτη”. Αυτό το παράδειγμα αντικατοπτρίζει την δοομή των συστημάτων στον πραγματικό κόσμο και άρα είναι κατάλληλο για να μοντελοποιήσει συσήματα με πεείπλοκη συμποεριφορά (σε αντίθεση με την αρχή του KISS).
Ο Steve Yegge και άλλοι σημείωσαν ότι οι φυσικές γλώσσες δεν έχουν την προσέγγιση του αντικειμενοστραφούς προγραμματισμού της αυστηρής ιεράρχησης των πραγμάτων (αντικείμενα/ουσιαστικά) πριν την δράση (μέθοδοι/ρήματα)[285]. Αυτό το πρόβλημα μπορεί να οδηγήσει μέσω του αντικειμενοστραφούς προγραμματισμού σε περισσότερο σύνεθετες λύσεις σε σχέση με τον διαδικασιακό[286].

5.12 Αντικειμενοστραφής προγραμματισμός και έλεγχος ροής

Ο αντικειμενοστραφής προγραμματισμός συνέβαλε καθοριστικά στην επαναχρησιμοποίηση και στην εύκολη συντήρηση του πηγαίου κώδικα[287]. Η διαφανής παρουσίαση του ελέγχου ροής έπαψε να αποτελεί προτεραιότητα και αφέθηκε για χειρισμό στον μεταφραστή. Με την αυξανόμενη συνάφεια του παράλληλου υλικού και του πολυνηματικού κώδικα η ανάπτυξη διαφανούς ελέγχου ροής έγινε περισσότερο απαραίτητη, στόχος δύσκολα επιτεύξιμος μέσω του αντικειμενοστραφούς προγραμματισμού[288][289][290][291].

---

283 Meyer, Second Edition, p. 230

284 M.Trofimov, *OOOP - The Third "O" Solution: Open OOP.* First Class, OMG, 1993, Vol. 3, issue 3, p.14.

285 *Yegge, Steve (30 March 2006). "Execution in the Kingdom of Nouns". steve-yegge.blogspot.com. Retrieved 3 July 2010.*

286 *Boronczyk, Timothy (11 June 2009). "What's Wrong with OOP". zaemis.blogspot.com. Retrieved 3 July 2010.*

287 *Ambler, Scott (1 January 1998). "A Realistic Look at Object-Oriented Reuse". www.drdobbs.com. Retrieved 4 July 2010.*

288 *Shelly, Asaf (22 August 2008). "Flaws of Object Oriented Modeling". Intel Software Network. Retrieved 4 July 2010.*

5.13 Ευθύνη εναντίον σχεδιασμού οδηγούμενου από τα δεδομένα

Ο σχεδιασμός ο οποίος οδηγείται από την ευθύνη ορίζει κλάσεις με όρους συμβολαίου, με την έννοια ότι μια κλάση θα πρέπει να οριστεί γύρω από μια ευθύνη και της πληροφορίας που μοιράζεται. Αυτό το σκεπτικό είναι αντίθετο με το σκεπτικό των Wiefis-Brock και Wilkerson η οποίοι εισηγούνται τον σχεδιασμό που οδηγείται από τα δεδομένα, όπου οι κλάσεις ορίζονται γύρω από τις δομές δεδομένων τα οποία πρέπει να κρατηθούν.

5.14 Οι κατευθυντήριες γραμμές SOLID και GRASP

Το SOLID είναι ένα μνημονικό που εφευρέθηκε από τον Michael Feathers ο οποίος υποστηρίζει πέντε προγραμματιστικές πρακτικές:

- Η αρχή της μονής ευθύνης
- Η αρχή ανοιχτή/κλειστή
- Η αρχή αντικατάστασης του Liskov
- Η αρχή του διαχωρισμού του συστήματος διεπαφής
- Η αρχή της αναστροφής της εξάρτησης

Το GRASP (General Responsibility Assigment Software Patterns) είναι άλλο ένα σετ από οδηγίες τις οποίες υποστηρίζει ο Craig Larman.

5.15 Κριτική

Το παράδειγμα του αντικειμενοστραφούς προγραμματισμού έγινε αντικείμενο κριτικής για αρκετούς λόγους μεταξύ των οποίων ήταν η μη επίτευξη των στόχων που είχε θέσει για επαναχρησιμοποίηση και σπονδυλωτής κατασκευής[292][293] καθώς και για τον υπερτονισμό μιας μόνο διάστασης του σχεδιασμού λογισμικού και της μοντελοποίησης (δεδομένα/αντικείμενο) σε κόστος άλλων εξίσου σημαντικών διαστάσεων (υπολογιστικότητα/αλγόριθμος)[294][295].

Ο Luca Cardelli υποστήριξε ότι οι αντικειμενοστραφής προγραμματισμός είναι εγγενώς λιγότερο αποτελεσματικός από τον διαδικασιακό, ότι ο αντικειμενοστραφής προγραμματισμός μπορεί να χρειαστεί περισσότερο χρόνο μετάφρασης και ότι οι αντικειμενοστραφείς γλώσσες έχουν ιδιαιτέρως φτωχές ιδιότητες σπονδυλωτής κατασκευής και έχουν την τάση να είναι ιδιαιτέρως

---

289 *James, Justin (1 October 2007). "Multithreading is a verb not a noun". techrepublic.com. Retrieved 4 July 2010.*

290 *Shelly, Asaf (22 August 2008). "HOW TO: Multicore Programming (Multiprocessing) Visual C++ Class Design Guidelines, Member Functions". support.microsoft.com. Retrieved 4 July 2010.*

291 *Robert Harper (17 April 2011). "Some thoughts on teaching FP". Existential Type Blog. Retrieved 5 December 2011.*

292 *Cardelli, Luca (1996). "Bad Engineering Properties of Object-Oriented Languages". ACM Comput. Surv. (ACM) 28 (4es): 150. doi:10.1145/242224.242415. ISSN 0360-0300. Retrieved 21 April 2010.*

293 Armstrong, Joe. In *Coders at Work: Reflections on the Craft of Programming.* Peter Seibel, ed. Codersatwork.com, Accessed 13 November 2009.

294 *Stepanov, Alexander. "STLport: An Interview with A. Stepanov". Retrieved 21 April 2010.*

295 Rich Hickey, JVM Languages Summit 2009 keynote, Are We There Yet? November 2009.

πολύπλοκες[296]. Το τελευταίο χαρακτηριστικό επαναλαμβάνεται από τον Joe Armstrong τον κύριο εφευρέτη του Erlang ο οποίος είπε[297]: “το πρόβλημα με τις αντικειμενοστραφείς γλώσσες προγραμματισμού είναι αυτό το έμμεσο περιβάλλον το οποίο κουβαλούν μαζί τους. Ήθελες μια μπανάνα και αντί για αυτό πήρες ένα γορίλα που κρατάει μια μπανάνα και ολόκληρη την ζούγκλα”. Μια μελέτη από το Potok et al εν έδειξε καμιά αξιοπρόσεκτη διαφορά στην παραγωγικότητα μεταξύ των αντικειμενοστραφών και των διαδικασιακών προσεγγίσεων[298].

Ο Christopher J. Date δήλωσε ότι η ουσιαστική σύγκειση του αντικειμενοστραφούς προγραμματισμού με τις άλλες τεχνολογίες και κυρίως τις σχεσιακές, είναι δύσκολη εξαιτίας της έλλειψης ενός αυστηρού ορισμού του αντικειμενοστραφούς προγραμματισμού[299]. Ωστόσο οι Date και Darwen πρότειναν ένα θεωρητικό θεμέλιο για τον αντικειμενοστραφή προγραμματισμό το οποίο χρησιμοποιεί τον αντικειμενοστραφή προγραμματισμό σαν ένα προσαρμοζόμενο σύστημα τυπων για την υποστήριξη των RDBMS[300].

Σε κάποιο άρθρο ο Lawrence Krubner ισχυρίστηκε ότι σε σύγκριση με τις άλλες γλώσσες (τις διαλέκτους της LISP, λειτουργικές γλώσσες κλπ) οι αντικειμενοστραφείς γλώσσες δεν έχουν κάποιο μοναδικό προτέρημα και είναι αναγκασμένες να κουβαλούν ένα βαρύ φορτίου αχρείαστης πολυπλοκότητας[301].

Ο Alexander Stepanov συγκρίνει την αντικειμενοστρέφεια δυσμενώς προς τον γενικό προγραμματισμό[302]: “Βρίσκω τον αντικειμενοστραφή προγραμματισμό τεχνικώς επισφαλή. Επιχειρεί να αποσυνθέσει τον κόσμο με όρους διεπαφών τα οποία διαφέρουν σε ένα μόνο τύπο. Για να αντιμετωπίσεις πραγματικά προβλήματα χρειάζεσαι άλγεβρες πολλαπλών ειδών, οικογένειες από interfaces διαφορετικών τύπων”. Βρίσκω τον αντικειμενοστραφή προγραμματισμό φιλοσοφικά επισφαλή. Ισχυρίζεται ότι τα πάντα είναι ένα αντικείμεν. Ακόμα και αν κάτι τέτοιο είναι αλήθεια, δεν έχει κανένα απολύτως ενδιαφέρον. Το να λες ότι τα πάντα είναι ένα αντικείμενο είναι σαν να μην λες τίποτα απολύτως.

Ο Paul Graham πρότεινε ότι η ευρεία αποδοχή του αντικειμενοστραφούς προγραμματισμού μέσα στις μεγάλες εταιρίες στις μεγάλες (και συχνά εναλλασσόμενες) ομάδες μέτριων προγραμματιστών. Σύμφωνα με τον Graham η αρχή που επιβάλετε από τον αντικειμενοστραφή προγραμματισμό εμποδίζει κάποιον προγραμματιστή από το να κάνει μεγάλη ζημιά[303].

Ο Steve Yegge σημείωσε ότι σε αντίθεση με τον διαδικασιακό προγραμματισμό[304]: “Ο αντικειμενοστραφής προγραμματισμός βάζει ως πρώτα και σημαντικά τα ουσιαστικά. Γιατί να

---

296 *Cardelli, Luca (1996). "Bad Engineering Properties of Object-Oriented Languages". ACM Comput. Surv. (ACM) 28 (4es): 150. doi:10.1145/242224.242415. ISSN 0360-0300. Retrieved 21 April 2010.*

297 Armstrong, Joe. In *Coders at Work: Reflections on the Craft of Programming.* Peter Seibel, ed. Codersatwork.com, Accessed 13 November 2009.

298 *Potok, Thomas; Mladen Vouk; Andy Rindos (1999). "Productivity Analysis of Object-Oriented Software Developed in a Commercial Environment" (PDF). Software – Practice and Experience 29 (10): 833–847. doi:10.1002/(SICI)1097-024X(199908)29:10<833::AID-SPE258>3.0.CO;2-P. Retrieved 21 April 2010.*

299 C. J. Date, Introduction to Database Systems, 6th-ed., Page 650

300 C. J. Date, Hugh Darwen. *Foundation for Future Database Systems: The Third Manifesto* (2nd Edition)

301 *Krubner, Lawrence. "Object Oriented Programming is an expensive disaster which must end". smashcompany.com. Retrieved 14 October 2014.*

302 *Stepanov, Alexander. "STLport: An Interview with A. Stepanov". Retrieved 21 April 2010.*

303 *Graham, Paul. "Why ARC isn't especially Object-Oriented.". PaulGraham.com. Retrieved 13 November 2009.*

304 Stevey's Blog Rants

φτάσουμε σε τέτοια μήκη απλά και μόνο για να βάλουμε ένα μέρος του λόγου σε ένα βάθρο; Γιατί θα πρέπει ένα είδος έννοιας να υπερισχύει έναντι ενός άλλου; Είναι σαν ο αντικειμενοστραφής προγραμματισμός να έκανε ξαφνικά τα ρήματα λιγότερο σημαντικά στον τρόπο με τον οποίο σκεφτόμαστε. Είναι μια περιέργως λοξή άποψη".

Ο Rich Hickey δημιουργός του Clojure περιέγραψε τα συστήματα των αντικειμένων σαν εν πολλοίς απλουστευμένα μοντέλα του πραγματικού κόσμου. Έδωσε ιδιαίτερη έμφαση στη ανικανότητα του αντικειμενοστραφούς προγραμματισμού να μοντελοποίησει επαρκώς την ένοια του χρόνου γεγονός το οποίο γίνεται ολοένα και μεγαλύτερο πρόβλημα καθώς τα συστήματα λογισμικού γίνονται όλο και πιο συγχρονισμένα[305].

Ο Eric S. Raymond ένας από τους προγραμματιστές του Unix και συνήγορος του ανοιχτού λογισμικού κριτίκαρε ισχυρισμούς οι οποίοι παρουσιάζουν τον αντικειμενοστραφή προγραμματισμό σαν την μοναδική αλήθεια και έγραψε ότι οι αντικειμενοστραφείς γλώσσες έχουν την τάση να ενθαρύνουν την κατασκευή προγραμμάτων με πολύ χοντρά επίπεδα τα οποία καταστρέφουν την διαφάνεια[306]. Ο Raymond συγκρίνει αυτό δυσμενώς σε σχέση με την προσέγγιση του Unix και της C[307].

5.16 Σημασιολογία τύπων

Τα αντικείμενα είναι οντότητες χρόνου εκτέλεσης σε ένα αντικειμενοστραφές σύστημα. Μπορούν να αναπαριστούν ένα άτομο, ένα μέρος, ένα τραπεζικό λογαριασμό, ένα πίνακα δεδομένων ή οποιοδήποτε αντικείμενο το οποίο πρέπει να χειριστεί το πρόγραμμα.

Έχουν γίνει αρκετές προσπάθειες για την τυποποίηση των εννοιών στον αντικειμενοστραφή προγραμματισμό. Οι παρακάτω έννοιες και κατασκευές έχουν χρησιμοποιηθεί ως ερμηνείες των εννοιών του αντικειμενοστραφούς προγραμματισμού:

- Συν αλγεβρικοί τύποι δεδομένων.
- Αφαιρετικοί τύποι δεδομένων οι οποίοι επιτρέπουν τον ορισμό εννοιών αλλά δεν επιτρέπουν την δυναμική αποστολή
- αναδρομικοί τύποι
- κατάσταση ενθυλάκωσης
- κληρονομικότητα
- Εγγραφές. Αποτελούν την βάση της κατανόησης των αντικειμένων και μπορούν να αποθηκευθούν σε πεδία (όπως και στον διαδικασιακό προγραμματισμό) μα ο πραγματικός λογισμός πρέπει να είναι σημαντικά πιο περίπλοκος για να ενσωματώσει τα βασικά χαρακτηριστικά του αντικειμενοστραφούς προγραμματισμού. Διάφορες επεκτάσεις του System F<: οι οποίες διαχειρίζονται μεταβλητά αντικείμενα έχουν μελετηθεί. Αυτές επιτρέπουν τόσο των πολυμορφισμό υποτύπων όσο και τον πολυμορφισμό παραμέτρων.

Οι προσπάθειες για την ανεύρεση ενός επικρατούντος ορισμού ή θεωρίας πίσω από τα αντικείμενα δεν αποδείχτηκαν ιδιαιτέρως επιτυχημένες και συχνά διαφέρουν εκτενώς. Για παράδειγμα μερικοί

---

305 Rich Hickey, JVM Languages Summit 2009 keynote, Are We There Yet? November 2009.

306 *Eric S. Raymond (2003). "The Art of Unix Programming: Unix and Object-Oriented Languages". Retrieved 2014-08-06.*

307 *Eric S. Raymond (2003). "The Art of Unix Programming: Unix and Object-Oriented Languages". Retrieved 2014-08-06.*

ορισμοί επικεντρώνονται στις νοητικές δραστηριότητες και άλλοι στην δομή των προγραμμάτων. Ένας από τους απλούστερους ορισμούς είναι ότι αντικειμενοστραφής προγραμματισμός είναι η πράξη της χρήσης χάρτη δομών δεδομένων ή πινάκων οι οποίοι μπορούν να περιέχουν διαδικασίες και δείκτες σε άλλους χάρτες και όλα αυτά με λίγη ζάχαρη συντακτικού και εμβέλειας. Η κληρονομικότητα μπορεί να εκτελεστεί από την κλωνοποίηση των χαρτών (διαδικασία που κάποιες φορές καλείται προτυποποίηση).

Κεφάλαιο 6 Αρχιτεκτονική Υπολογιστών

6 Αρχιτεκτονική υπολογιστών[308][309]

308 David A. Patterson, John L. Hennessy (2010). *Οργάνωση και σχεδίαση υπολογιστών η διασύνδεση υλικού και*

6.1 Ορισμός

Η **αρχιτεκτονική υπολογιστών**, ή **οργάνωση υπολογιστών**, είναι το γνωστικό πεδίο της μηχανικής υπολογιστών το οποίο πραγματεύεται τον λογικό σχεδιασμό, τη δομή και τη λειτουργία του υλικού ενός υπολογιστικού συστήματος, συνήθως ηλεκτρονικού και ψηφιακού. Ως επιστημονικός τομέας εστιάζει στη συστηματική έρευνα και σχεδίαση των τεχνολογικών δομών υλικού που επιτρέπουν την αποδοτική εκτέλεση αλγορίθμων και υπολογισμών, με βάση τις διαθέσιμες τεχνολογίες κατασκευής ολοκληρωμένων κυκλωμάτων. Συνήθως, η αρχιτεκτονική υπολογιστών δίνει έμφαση στη δομή και λειτουργία του επεξεργαστή και στους τρόπους προσπέλασής του στη μνήμη.
Ένας υπολογιστής δομείται σε μία ιεραρχία αφηρημένων επιπέδων οργάνωσης τα οποία οικοδομούνται το ένα πάνω στο άλλο: κάθε υπερκείμενο επίπεδο αξιοποιεί το υποκείμενό του. Η τακτική αυτή ονομάζεται «*δομημένη οργάνωση υπολογιστών*» και επιτρέπει τη συστηματική και εύκολη ανάλυση, σχεδίαση και κατανόηση των υπολογιστικών συστημάτων. Το σύνολο των εννοιών, λειτουργιών και λεπτομερειών ενός επιπέδου ονομάζεται «*αρχιτεκτονική*» αυτού του επιπέδου.

6.2 Επισκόπηση της οργάνωσης υπολογιστικών συστημάτων
6.2.1 Μητρική Κάρτα

Μια μητρική κάρτα, επίσης γνωστή και σαν μητρική ή κάρτα συστήματος είναι το κεντρικό και βασικό τυπωμένο ηλεκτρικό κύκλωμα ενός σημερινού υπολογιστή. Ένας τυπικός υπολογιστής αποτελείται από τον μικροεπεξεργαστή, την κεντρική μνήμη και άλλα βασικά υποσυστήματα που βρίσκονται και αυτά στην μητρική. Άλλα μέρη του υπολογιστή, όπως εξωτερικά μέσα αποθήκευσης, κάρτες επέκτασης γραφικών, ήχου κτλ και διάφορα περιφερειακά όπως εκτυπωτής, πληκτρολόγια κτλ, είναι όλα τμήματα που ενσωματώνονται στην μητρική μέσω καλωδίων και υποδοχών διάφορων τύπων. Συνήθως, όλα τα κύρια εξαρτήματα του υπολογιστή - ο επεξεργαστής, η μνήμη ROM, η μνήμη RAM, ο δίαυλος (Bus), το ρολόι - είναι τοποθετημένα πάνω στη μητρική κάρτα. Κάθε τέτοια μητρική κάρτα έχει κατασκευαστεί για ένα συγκεκριμένο τύπο επεξεργαστή που λειτουργεί σε καθορισμένη συχνότητα.
Κεντρική Μονάδα Επεξεργασίας (CPU)
Όταν στον υπολογιστή εισάγονται δεδομένα (γράμματα, αριθμοί, εικόνες), η μορφή τους είναι τέτοια ώστε να γίνεται κατανοητή από το χρήστη. Ο υπολογιστής δεν καταλαβαίνει τις μορφές αυτές και πρέπει πρώτα να τα μετατρέψει σε μορφή που να τα αντιλαμβάνεται, ώστε να μπορεί να τα χειριστεί. Αφού γίνει αυτό, τα αποθηκεύει προσωρινά και στη συνέχεια εκτελεί την αριθμητική ή λογική επεξεργασία τους. Για να γίνουν αυτά πράξη, η Κ.Μ.Ε. απαρτίζεται από τις εξής επιμέρους μονάδες:

- **Μονάδα αποκωδικοποίησης** [310][311](Decoding Unit): Μετατρέπει τα "φυσικά" δεδομένα από

λογισμικού. Εκδόσεις Κλειδάριθμος.

309 *Behrouz A. Forouzan, Firouz Mosharraf (2010). Εισαγωγή στην επιστήμη των υπολογιστών. Εκδόσεις Κλειδάριθμος.*

310 David A. Patterson, John L. Hennessy (2010). *Οργάνωση και σχεδίαση υπολογιστών η διασύνδεση υλικού και*

τη μορφή υπό την οποία εισάγονται στον υπολογιστή στη "γλώσσα" που η Κ.Μ.Ε. μπορεί να "καταλάβει" και ονομάζεται "κώδικας μηχανής" (machine code).

- **Μονάδα Αριθμητικής και Λογικής**(Arithmetic and Logical Unit, ALU): Η υπομονάδα στην οποία εκτελούνται μία προς μία οι αριθμητικές ή λογικές πράξεις, όπως υπαγορεύονται από τις εντολές που έχουν δοθεί στον υπολογιστή.
- **Καταχωρητές** (Registers): Μικρά στοιχεία μνήμης, που χρησιμοποιούνται για την προσωρινή αποθήκευση (καταχώρηση) των δεδομένων, καθώς αυτά υφίστανται επεξεργασία. Οι καταχωρητές διαφέρουν ανάλογα με τον τύπο της Κ.Μ.Ε. και τον κατασκευαστή της, τόσο ως προς την οργάνωση όσο και ως προς τη χωρητικότητά τους.
- **Μονάδα ελέγχου** (Control Unit): Ελέγχει τη ροή δεδομένων από και προς την ALU, τους καταχωρητές, τη μνήμη και τις περιφερειακές μονάδες εισόδου/εξόδου.
- **Μονάδα προσκόμισης** (Fetch Unit): Μεταφέρει τις εντολές από τη μνήμη στην Κ.Μ.Ε. πριν αυτές χρειαστούν, ώστε να είναι άμεσα διαθέσιμες προς χρήση.
- **Μονάδα προστασίας** (Protection Unit): Εξασφαλίζει το αποδεκτό της κάθε διεργασίας που εκτελεί η Κ.Μ.Ε., ώστε να μη τροποποιούνται δεδομένα που δεν πρέπει ή να μην εκτελούνται μη αποδεκτές εντολές, όπως, π.χ., διαίρεση αριθμού με το μηδέν.

Τα πιο πάνω επιμέρους στοιχεία μιας Κ.Μ.Ε. αποτελούν τον πυρήνα της.

6.2.2 Μνήμη

Οι σύγχρονοι υπολογιστές σχεδιάζονται με βάση της αρχές που αναπτύχθηκαν από τον Τζον φον Νόιμαν στο ινστιτούτο προηγμένων επιστημών στο Princeton. Αυτές οι θεμελιώδεις αρχές που αναφέρονται παρακάτω συνιστούν την Αρχιτεκτονική φον Νόιμαν. Τα δεδομένα και οι εντολές αποθηκεύονται σε μια μοναδική μνήμη εγγραφής-ανάγνωσης. Τα περιεχόμενα της μνήμης αυτής μπορούν να διευθυνσιοδοτηθούν κατά θέση, χωρίς να μας ενδιαφέρει ο τύπος των δεδομένων που περιέχεται εκεί. Η εκτέλεση εντολών πραγματοποιείται ως ακολουθία (εκτός και αν υπάρχει ρητή τροποποίηση) από μια εντολή στην επόμενη.

6.2.3 Καταχωρητής

Στην επιστήμη της αρχιτεκτονικής υπολογιστών, ο καταχωρητής είναι ένας τύπος μικρής αλλά πολύ γρήγορης μνήμης που βρίσκεται μέσα στο τσιπ του επεξεργαστή. Η μνήμη αυτή χρησιμοποιείται για την βελτίωση της ταχύτητας εκτέλεσης των διαφόρων προγραμμάτων, αφού σε αυτήν συνήθως αποθηκεύονται δεδομένα που χρησιμοποιούνται συνέχεια από τα προγράμματα. Στην περίπτωση αυτή ο καταχωρητής παρέχει πολύ γρήγορη πρόσβαση σε αυτά τα δεδομένα και έτσι το πρόγραμμα εκτελείται πιο γρήγορα. Οι περισσότεροι από τους σύγχρονους ηλεκτρονικούς υπολογιστές λειτουργούν σύμφωνα με την εξής λογική: μεταφέρουν δεδομένα από την κεντρική μνήμη στους καταχωρητές, κάνουν τις διάφορες πράξεις πάνω στα δεδομένα και στην συνέχεια μεταφέρουν το αποτέλεσμα από τους καταχωρητές πίσω στην κύρια μνήμη. Η τεχνική αυτή

λογισμικού. Εκδόσεις Κλειδάριθμος.

311 *Behrouz A. Forouzan, Firouz Mosharraf (2010). Εισαγωγή στην επιστήμη των υπολογιστών. Εκδόσεις Κλειδάριθμος.*

ονομάζεται load-store architecture[312][313].

Ονομασίες Βασικών Καταχωρητών:

- Μετρητής Προγράμματος (Program Counter, PC)
- Καταχωρητής Εντολών (Instruction Register, IR)
- Καταχωρητής Διευθύνσεων Μνήμης (Memory Address Register, MAR)
- Καταχωρητής Δεδομένων Μνήμης (Memory Data Register, MDR)
- Συσσωρευτής (Accumulator, AC) ή A, B, C...
- Δείκτης Στοίβας (Stack Pointer, SP)
- Index, Base, Offset Registers
- Καταχωρητής Κατάστασης (Status Register, SR)

Η μνήμη ενός ηλεκτρονικού υπολογιστή μπορεί να διαταχθεί σε μορφή πυραμίδας. Τα κατώτερα στρώματα της πυραμίδας προσφέρουν μεγαλύτερη αλλά πιο αργή μνήμη. Αντιθέτως, τα ανώτερα στρώματα της πυραμίδας προσφέρουν μικρότερη μνήμη αλλά πολύ πιο γρήγορη. Το κατώτατο στρώμα της πυραμίδας είναι οι μαγνητικές ταινίες και ανεβαίνοντας προς τα πάνω συναντά κανείς τους USB Flash δίσκους, τα CD-ROM ή DVD-ROM, τους σκληρούς δίσκους, την κύρια μνήμη RAM, την μνήμη L3 / L2 / L1 Cache του επεξεργαστή και τέλος τους καταχωρητές. Άρα λοιπόν οι καταχωρητές βρίσκονται στην κορυφή της πυραμίδας και προσφέρουν την πιο γρήγορη μνήμη που υπάρχει. Δυστυχώς όμως το μέγεθος της μνήμης αυτής είναι πολύ μικρό και περιορισμένο.

6.2.4 Κρυφή μνήμη ΚΜΕ

Κρυφή μνήμη Κεντρικής Μονάδας Επεξεργασίας (CPU cache) γνωστή και ως ενδιάμεση μνήμη ή λανθάνουσα μνήμη ΚΜΕ ονομάζουμε τη μνήμη που χρησιμοποιείται από την Κεντρική μονάδα επεξεργασίας για να πετύχει ταχύτερη πρόσβαση στην κύρια μνήμη. Αυτή η μνήμη είναι γρηγορότερη, ακριβότερη και μικρότερη σε μέγεθος από την κύρια μνήμη.

Είναι υψηλής ταχύτητας SRAM μνήμη, που χρησιμοποιείται μεταξύ της ΚΜΕ και της κύριας μνήμης. Εντολές και προγράμματα μπορούν να λειτουργήσουν σε υψηλότερες ταχύτητες αν βρεθούν στην cache. Εάν δεν βρεθούν, μια νέα σειρά εντολών φορτώνεται από την κύρια μνήμη.

6.3 DDR SDRAM

Η DDR SDRAM δυναμική μνήμη τυχαίας προσπέλασης διπλού ρυθμού μεταφοράς δεδομένων είναι ένας τύπος μνήμης κατασκευασμένης με ολοκληρωμένο κύκλωμα που χρησιμοποιείται στους υπολογιστές. Έχει μεγαλύτερο ρυθμό μεταφοράς πληροφορίας σε σχέση με την μνήμη SDRAM μεταφέροντας δεδομένα και κατά την ακμή ανόδου και την ακμή καθόδου του σήματος του ρολογιού. Έτσι σχεδόν διπλασιάζεται ο ρυθμός μεταφοράς χωρίς να χρειάζεται αύξηση της συχνότητας του μπροστινού διαύλου.

---

312 David A. Patterson, John L. Hennessy (2010). *Οργάνωση και σχεδίαση υπολογιστών η διασύνδεση υλικού και λογισμικού*. Εκδόσεις Κλειδάριθμος.

313 *Behrouz A. Forouzan, Firouz Mosharraf (2010). Εισαγωγή στην επιστήμη των υπολογιστών. Εκδόσεις Κλειδάριθμος.*

6.4 Δυναμική μνήμη τυχαίας προσπέλασης

Δυναμική μνήμη τυχαίας προσπέλασης (DRAM) είναι ένας τύπος μνήμης τυχαίας προσπέλασης που αποθηκεύει μπιτ δεδομένων σε έναν ξεχωριστό πυκνωτή. Όμως επειδή το φορτίο των πυκνωτών εξασθενεί με το χρόνο , η πληροφορία που έχουμε αποθηκευμένη εξασθενεί γι' αυτό πρέπει περιοδικά να ξαναφορτίζεται ο πυκνωτής, εξού και ο όρος δυναμική[314][315].

6.5 Χώρος διευθύνσεων

Χώρο διευθύνσεων ονομάζουμε ένα σύνολο διευθύνσεων κάποιον οντοτήτων. Αυτές οι οντότητες μπορεί να είναι κελιά της κύριας μνήμης, της εικονικής μνήμης, θύρες εισόδου/εξόδου, οι υπολογιστές ενός δικτύου. Έτσι για παράδειγμα δεδομένου μιας ΚΜΕ και τους εύρους του διαύλου διευθύνσεων της, πχ έστω 8 μπιτς, λέμε ότι ο χώρος διευθύνσεων μνήμης της ΚΜΕ είναι $2^8 = 256$. Δηλαδή η ΚΜΕ μπορεί να 'δει' να απευθυνθεί σε 256 ξεχωριστά κελιά μνήμης.

6.6 Σκληρός Δίσκος

Ο σκληρός δίσκος είναι ένα μαγνητικό αποθηκευτικό μέσο - συσκευή που χρησιμοποιείται στους ηλεκτρονικούς υπολογιστές, στις ψηφιακές βιντεοκάμερες, στα φορητά mp3 players, επιτραπέζια ψηφιακά βίντεο, κονσόλες παιχνιδομηχανών, ψηφιακούς επίγειους και δορυφορικούς τηλεοπτικούς δέκτες κ.τ.λ. Ένας σκληρός δίσκος αποθηκεύει μεγάλες ποσότητες δεδομένων και η συνήθης χωρητικότητα των σκληρών δίσκων που κυκλοφορούν στο εμπόριο είναι 80 GB έως 1 TB. Για μεγαλύτερες χωρητικότητες που αγγίζουν τα 4 TB (terabyte) χρησιμοποιούνται κυκλώματα πολλαπλών σκληρών δίσκων, με τη μορφή συρταρωτής διάταξης. Η ταχύτητα προσπέλασης των δεδομένων είναι ταχύτερη από το DVD αλλά πολύ πιο αργή από την μνήμη του υπολογιστή.
Οι σκληροί δίσκοι χρησιμοποιούνται στους υπολογιστές για την αποθήκευση δεδομένων, κυρίως προγραμμάτων και αρχείων που είναι απαραίτητο να διατηρηθούν, σε αντίθεση με την μνήμη RAM όπου τα δεδομένα διαγράφονται με την διακοπή τροφοδοσίας ηλεκτρικού ρεύματος.

6.6.1 Δομή

Ένας σκληρός δίσκος αποτελείται από:

1. Μαγνητικούς δίσκους κατασκευασμένους από μέταλλο ή πλαστικό και επικαλυμμένους από ένα λεπτό στρώμα οξειδίου του σιδήρου ή άλλο μαγνητικό υλικό.
2. Τον άξονα κίνησης γύρω από τον οποίο περιστρέφονται οι μαγνητικοί δίσκοι με την ίδια ταχύτητα.
3. Κεφαλές ανάγνωσης/εγγραφής επάνω σε βραχίονες πάνω και κάτω από κάθε επιφάνεια δίσκου, που μετακινούνται εμπρός-πίσω. Ο συνδυασμός της κίνησης των βραχιόνων με

314 David A. Patterson, John L. Hennessy (2010). *Οργάνωση και σχεδίαση υπολογιστών η διασύνδεση υλικού και λογισμικού*. Εκδόσεις Κλειδάριθμος.

315 *Behrouz A. Forouzan, Firouz Mosharraf (2010). Εισαγωγή στην επιστήμη των υπολογιστών. Εκδόσεις Κλειδάριθμος.*

την κίνηση των δίσκων, επιτρέπουν στις κεφαλές να έχουν πρόσβαση σε όλα τα σημεία των δίσκων.

4. Ηλεκτρονικά εξαρτήματα που εξυπηρετούν την λειτουργία του σκληρού δίσκου, επικοινωνώντας με τον υπολογιστή και αναλαμβάνοντας την κίνηση των κεφαλών και τη μεταφορά των δεδομένων.

6.6.2 Τρόπος αποθήκευσης

Τα δεδομένα αποθηκεύονται στον σκληρό δίσκο ως ακολουθίες bit (αφού οι υπολογιστές [316][317]λειτουργούν με το δυαδικό σύστημα). Οι κεφαλές γράφουν κάθε bit αλλάζοντας το μαγνητικό πεδίο στην επιφάνεια των μαγνητικών δίσκων και το διαβάζουν απλώς αναγνωρίζοντας το μαγνητικό πεδίο. Κάθε bit δεδομένων καταλαμβάνει τον δικό του χώρο στην επιφάνεια του δίσκου, ωστόσο οι ακολουθίες bit που αποτελούν τα δεδομένα, δεν είναι απαραίτητο να εγγράφονται σειριακά στον δίσκο, αλλά είναι δυνατό να κατακερματιστούν και να εγγραφούν σε διάφορες θέσεις.

6.6.3 Είδη Σκληρών Δίσκων

Με βάση το πρωτόκολλο επικοινωνίας και το interface οι δίσκοι διακρίνονται σε: IDE, SATA, SCSI.

6.7 Κάρτες Επέκτασης

Οι κάρτες επέκτασης είναι ηλεκτρονικά κυκλώματα που συνδέονται στη μητρική πλακέτα του υπολογιστή για να του επιτρέψουν να κάνει διάφορες εργασίες.
Οι κυριότερες κάρτες επέκτασης είναι:

- Η κάρτα γραφικών, η οποία είναι υπεύθυνη για τις πληροφορίες που βλέπουμε στην οθόνη.
- Η κάρτα ήχου, η οποία επιτρέπει στον υπολογιστή μας να παράγει και να επεξεργάζεται ήχο.
- Η κάρτα τηλεόρασης, η οποία μπορεί να μετατρέψει τον υπολογιστή μας σε τηλεοπτικό δέκτη
- Η κάρτα ραδιοφώνου, η οποία επιτρέπει στον υπολογιστή μας να συμπεριφέρεται ως κοινό ραδιόφωνο
- Η κάρτα επεξεργασίας video, η οποία επιτρέπει στον υπολογιστή μας να δέχεται, να επεξεργάζεται και να παράγει video.
- Η κάρτα δικτύου, η οποία δίνει τη δυνατότητα στον υπολογιστή μας να επικοινωνήσει μέσω καλωδίων με άλλους υπολογιστές που βρίσκονται στον ίδιο χώρο ή σε απόσταση.
- Το Modem (Τηλεφωνικός Διαμορφωτής - Αποδιαμορφωτής), μια συσκευή που δίνει τη δυνατότητα στον υπολογιστή μας να επικοινωνεί με άλλους υπολογιστές διαμέσου της

316 David A. Patterson, John L. Hennessy (2010). *Οργάνωση και σχεδίαση υπολογιστών η διασύνδεση υλικού και λογισμικού*. Εκδόσεις Κλειδάριθμος.

317 *Behrouz A. Forouzan, Firouz Mosharraf (2010). Εισαγωγή στην επιστήμη των υπολογιστών. Εκδόσεις Κλειδάριθμος.*

τηλεφωνικής γραμμής. Το modem μετατρέπει το ψηφιακό σήμα του υπολογιστή σε αναλογικό, για να μπορέσει να κυκλοφορήσει μέσα από τις τηλεφωνικές γραμμές και, στη συνέχεια, σε ψηφιακό, για να μπορέσει να το αναγνωρίσει ο υπολογιστής που βρίσκεται στην άλλη άκρη της γραμμής. Είναι πολύ σημαντική συσκευή, γιατί αναλαμβάνει να μας συνδέσει με τον έξω κόσμο δίνοντάς μας τη δυνατότητα να συνδεθούμε με το Internet. Χαρακτηριστικό των modem είναι η ταχύτητα μεταφοράς των πληροφοριών από τον ένα υπολογιστή στον άλλο. Σήμερα, τα modem έχουν ταχύτητα από 56K και πάνω και υποστηρίζουν και λειτουργία fax[318][319].

- Η Mobile connect card (κάρτα σύνδεσης φορητού υπολογιστή με υπηρεσίες κινητού τηλεφώνου). Πρόκειται για μία νέα κάρτα σύνδεσης, η οποία μας επιτρέπει να έχουμε άμεση και συνεχή πρόσβαση στο διαδίκτυο, αλλά και στο εταιρικό μας δίκτυο. Χρησιμοποιεί την υπηρεσία σύνδεσης ενός κινητού τηλεφώνου και, έτσι, μας δίνει τη δυνατότητα να στείλουμε και να λάβουμε από το φορητό υπολογιστή μας SMS, να διαχειριστούμε τον τηλεφωνικό μας κατάλογο και να έχουμε άμεση πληροφόρηση για τον όγκο των δεδομένων που διακινούνται.

6.8 Αντικείμενα Αρχιτεκτονικής

6.8.1 Αρχιτεκτονική Συνόλου Εντολών

Αρχιτεκτονική Συνόλου Εντολών (Instruction Set Architecture, ISA), είναι η λογική αφαίρεση ενός υπολογιστικού συστήματος στο επίπεδο της Γλώσσας Μηχανής (ή της Γλώσσας Assembly χωρίς τις κλήσεις του Λειτουργικού Συστήματος). Είναι το προγραμματιστικό μοντέλο (η εικονική μηχανή) που αντιλαμβάνεται ο προγραμματιστής που προγραμματίζει σε αυτό το (χαμηλότερο δυνατό) επίπεδο. Περιλαμβάνει το σύνολο εντολών, τις μεθόδους διευθυνσιοδότησης (προσπέλασης μνήμης), τη διαχείριση καταχωρητών, τη κωδικοποίηση διευθύνσεων και δεδομένων, το μηχανισμό κλήσης ρουτινών, τη διαχείριση εισόδου/εξόδου, τη διαχείριση των καταστάσεων και σημάτων διακοπής του επεξεργαστή. Πρόκειται ουσιαστικά για το σύνορο μεταξύ περιγραφής ενός υπολογιστικού συστήματος από πλευράς υλικού ή λογισμικού.

6.8.2 Μικροαρχιτεκτονική

Μικροαρχιτεκτονική (Microarchitecture), είναι το αμέσως χαμηλότερο επίπεδο, πιο συγκεκριμένο και λεπτομερές από το επίπεδο Αρχιτεκτονικής Συνόλου Εντολών. Περιλαμβάνει τη λεπτομερή περιγραφή του τρόπου σύνδεσης, λειτουργίας και χρονισμού των συστατικών μερών (του υλικού), έτσι ώστε αυτά να υλοποιούν (εκτελούν στη κυριολεξία) το σύνολο των εντολών. Δηλαδή τη πλήρη περιγραφή του κύκλου Ανάκλησης – Εκτέλεσης όλων των εντολών που υποστηρίζει ο υπολογιστής. Επίσης περιλαμβάνονται και θέματα Παραλληλισμού Επιπέδου Εντολής (Instruction Level Parallelism, ILP), δηλαδή αρχιτεκτονικές βελτιώσεις με στόχο την αύξηση της απόδοσης του

318 David A. Patterson, John L. Hennessy (2010). *Οργάνωση και σχεδίαση υπολογιστών η διασύνδεση υλικού και λογισμικού*. Εκδόσεις Κλειδάριθμος.

319 *Behrouz A. Forouzan, Firouz Mosharraf (2010). Εισαγωγή στην επιστήμη των υπολογιστών. Εκδόσεις Κλειδάριθμος.*

επεξεργαστή.

6.9 Σχεδίαση Συστήματος

Σχεδίαση Συστήματος (System Design) που περιλαμβάνει τη διασύνδεση και λειτουργία των βασικών συστατικών στοιχείων (υλικού) του υπολογιστή, κυρίως εκτός του επεξεργαστή, στο μέτρο που αυτά επηρεάζουν την απόδοση του επεξεργαστή, όπως:

- Ιεραρχίες μνήμης (κρυφή μνήμη, εικονική μνήμη)
- Δίαυλοι, Ρολόγια, Διακόπτες, Ελεγκτές κλπ.
- Συστήματα συν-επεξεργασίας (GPUs, DMAs, NICs)
- Παραλληλισμός σε επίπεδο Επεξεργαστών[320][321].

6.10 Ορισμοί

6.10.1 Αρχιτεκτονική

Λογική Αφαίρεση, Μοντέλο, Οπτική, Σχέδιο με έμφαση στη χρήση, λειτουργία και σχεδίαση Αρχιτεκτονική Λογισμικού, Επιχειρησιακή Αρχιτεκτονική κλπ.

6.10.2 Οργάνωση

Δομή, Σύνθεση Συστατικών Μερών, Υλοποίηση του Σχεδίου με έμφαση στο υλικό μέρος και στις τεχνικές λεπτομέρειες.

6.10.3 Σύστημα

Δομημένη, Ιεραρχική αντιμετώπιση της μορφής Είσοδος – Κατάσταση (Επεξεργασία) – Έξοδος, όπου η Κατάσταση μπορεί να αναλυθεί παραπέρα.

6.11 Εξέλιξη πολυεπίπεδων μηχανών

20ός αιώνας

- Λίγο, πανάκριβο, δύσχρηστο υλικό (1945)
- Επινόηση διερμηνείας και μικροπρογραμματισμού (1950)
- Επινόηση του λειτουργικού συστήματος (1960)
- Μεγέθυνση του μικροκώδικα (1970)
- Μείωση προς εξάλειψη του μικροπρογραμματισμού (1985)

21ος αιώνας

---

320 David A. Patterson, John L. Hennessy (2010). *Οργάνωση και σχεδίαση υπολογιστών η διασύνδεση υλικού και λογισμικού*. Εκδόσεις Κλειδάριθμος.

321 *Behrouz A. Forouzan, Firouz Mosharraf (2010). Εισαγωγή στην επιστήμη των υπολογιστών. Εκδόσεις Κλειδάριθμος.*

- Άφθονο, φθηνό, εύχρηστο υλικό (2000)
- Ισοδυναμία υλικού και λογισμικού
- Αρχική ανάπτυξη λογισμικού λόγω του ακριβού υλικού
- Σταδιακή μετάβαση λογισμικού προς το υλικό (VLSI)
- Πρόσθεση νέων επιπέδων λογισμικού λόγω του φθηνού υλικού

6.12 Γενιές υπολογιστών

Υψηλής κλίμακας I/O chip (VLSI VL82C106)
Μηδενική Γενιά
Μηχανικοί Υπολογιστές (1642 – 1945)[322][323]
Πρώτη Γενιά
Λυχνίες Κενού (1945 – 1955)
Δεύτερη Γενιά
Στερεοί Ημιαγωγοί-Transistors (1955 – 1965)
Τρίτη Γενιά
Ολοκληρωμένα Κυκλώματα (1965 – 1980)
Τέταρτη Γενιά
Ολοκλήρωση υψηλής κλίμακας-VLSI (1980 –?)

322 David A. Patterson, John L. Hennessy (2010). *Οργάνωση και σχεδίαση υπολογιστών η διασύνδεση υλικού και λογισμικού*. Εκδόσεις Κλειδάριθμος.

323 *Behrouz A. Forouzan, Firouz Mosharraf (2010). Εισαγωγή στην επιστήμη των υπολογιστών. Εκδόσεις Κλειδάριθμος.*

Βιβλιογραφία

Ελληνική βιβλιογραφία

- Α. Βακάλη, Η. Γιαννόπουλος, Ν. Ιωαννίδης, Χ. Κοίλιας, Κ. Μαλάμας, Ι. Μανωλόπουλος, Π. Πολίτης (2010 - Έκδοση ΙΑ). *Ανάπτυξη Εφαρμογών σε Προγραμματιστικό Περιβάλλον*. Υπουργείο Εθνικής Παιδείας και Θρησκευμάτων - Παιδαγωγικό Ινστιτούτο, σελ. 25-27. ISBN 960-06-1408-3.
- Σ. Ζάχος, Δ. Φωτάκης. «Μηχανές Turing και Υπολογισιμότητα». Διαφάνειες μαθήματος στην Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών, Εθνικό Μετσόβιο Πολυτεχνείο. Ανακτήθηκε στις 2011-10-06.
- Παπαδοπούλου Βίκη (2007). «Θεωρία Υπολογισμού και Πολυπλοκότητα». European University of Cyprus. Ανακτήθηκε στις 2011-10-06.

Ξενόγλωσση βιβλιογραφία

- Vitanyi, P. "Obituary: Ray Solomonoff, Founding Father of Algorithmic Information Theory"
- 2. Solomonoff, R., "A Preliminary Report on a General Theory of Inductive Inference
- 3. Yongge Wang: Randomness and Complexity. PhD Thesis, 1996
- Assembler language, IBM Knowledge center
- *Saxon, James; Plette, William (1962). "Programming the IBM 1401". Prentice-Hall. LoC 62-20615. [use of the term assembly program]*
- (John Daintith, ed.) A Dictionary of Computing: "meta-assembler"
- David Salomon (1993). *Assemblers and Loaders*
- *Beck, Leland L. (1996). "2". System Software: An Introduction to Systems Programming. Addison Wesley.*
- Hyde, Randall. "Chapter 12 – Classes and Objects". The Art of Assembly Language, 2nd Edition. No Starch Press. © 2010.
- *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference (PDF). Intel Corporation. 1999. Retrieved 18 November 2010.*
- *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference (PDF). Intel Corporation. 1999. pp. 442 and 35. Retrieved 18 November 2010.*
- *Evans, David (2006). "x86 Assembly Guide". University of Virginia. Retrieved 18 November 2010.*
- *"The SPARC Architecture Manual, Version 8" (PDF). SPARC, International. 1992.*
- Z80 Op Codes for ZINT. Z80.de. Retrieved on 2013-07-21.
- *Microsoft Corporation. "MASM: Directives & Pseudo-Opcodes" (PDF). Retrieved March 19, 2011.*
- Griswold, Ralph E. *The Macro Implementation of SNOBOL4*. San Francisco, CA: W. H. Freeman and Company, 1972 (ISBN 0-7167-0447-1), Chapter 1.
- *"Macros (C/C++), MSDN Library for Visual Studio 2008". Microsoft Corp. Retrieved 2010-06-22.*
- *"Concept 14 Macros". MVS Software. Retrieved May 25, 2009.*
- *Answers.com. "assembly language: Definition and Much More from Answers.com".*

Retrieved 2008-06-19.

- *Provinciano, Brian. "NESHLA: The High Level, Open Source, 6502 Assembler for the Nintendo Entertainment System".*
- *Salomon. Assemblers and Loaders (PDF). p. 7. Retrieved 2012-01-17.*
- *"The IBM 650 Magnetic Drum Calculator". Retrieved 2012-01-17.*
- Eidolon's Inn: SegaBase Saturn
- *Jim Lawless (2004-05-21). "Speaking with Don French : The Man Behind the French Silk Assembler Tools". Archived from the original on 21 August 2008. Retrieved 2008-07-25.*
- *Rusling, David A. "The Linux Kernel". Retrieved Mar 11, 2012.*
- *"Writing the Fastest Code, by Hand, for Fun: A Human Computer Keeps Speeding Up Chips". New York Times, John Markoff. 2005-11-28. Retrieved 2010-03-04.*
- *"Bit-field-badness". hardwarebug.org. 2010-01-30. Archived from the original on 5 February 2010. Retrieved 2010-03-04.*
- *"GCC makes a mess". HardwareBug.org. 2009-05-13. Archived from the original on 16 March 2010. Retrieved 2010-03-04.*
- *Randall Hyde. "The Great Debate". Archived from the original on 16 June 2008. Retrieved 2008-07-03.*
- *"Code sourcery fails again". hardwarebug.org. 2010-01-30. Archived from the original on 2 April 2010. Retrieved 2010-03-04.*
- *Click, Cliff. "A Crash Course in Modern Hardware". Retrieved May 1, 2014.*
- *"BLAS Benchmark-August2008". eigen.tuxfamily.org. 2008-08-01. Retrieved 2010-03-04.*
- *"x264.git/common/x86/dct-32.asm". git.videolan.org. 2010-09-29. Retrieved 2010-09-29.*
- *"68K Programming in Fargo II". Archived from the original on 2 July 2008. Retrieved 2008-07-03.*
- *Hyde, Randall (1996-09-30). "Foreword ("Why would anyone learn this stuff?"), op. cit.". Archived from the original on 25 March 2010. Retrieved 2010-03-05.*
- *Randall Hyde. "Which Assembler is the Best?". Archived from the original on 18 October 2007. Retrieved 2007-10-19.*
- *Kernighan, Brian W.; Ritchie, Dennis M. (February 1978). The C Programming Language (1st ed.). Englewood Cliffs, NJ: Prentice Hall. ISBN 0-13-110163-3.* Regarded by many to be the authoritative reference on C.
- Ritchie (1993): "Thompson had made a brief attempt to produce a system coded in an early version of C—before structures—in 1972, but gave up the effort."
- Ritchie (1993): "The scheme of type composition adopted by C owes considerable debt to Algol 68, although it did not, perhaps, emerge in a form that Algol's adherents would approve of."
- *"Verilog HDL (and C)" (PDF). The Research School of Computer Science at the Australian National University. 2010-06-03. Retrieved 2013-08-19.* 1980s: ; Verilog first introduced ; Verilog inspired by the C programming language
- Ritchie (1993)
- *Lawlis, Patricia K. (August 1997). "Guidelines for Choosing a Computer Language:*

Support for the Visionary Organization". Ada Information Clearinghouse. Retrieved 18 July 2006.

- *"Programming Language Popularity". 2009. Retrieved 16 January 2009.*
- *"TIOBE Programming Community Index". 2009. Retrieved 6 May 2009.*
- *Ritchie, Dennis M. (March 1993). "The Development of the C Language". ACM SIGPLAN Notices **28** (3): 201–208. doi:10.1145/155360.155580.*
- *Johnson, S. C.; Ritchie, D. M. (1978). "Portability of C Programs and the UNIX System". Bell System Tech. J. **57** (6): 2021–2048. doi:10.1002/j.1538-7305.1978.tb02141.x. Retrieved 16 December 2012.* (Note: this reference is an OCR scan of the original, and contains an OCR glitch rendering "IBM 370" as "IBM 310".)
- *McIlroy, M. D. (1987). A Research Unix reader: annotated excerpts from the Programmer's Manual, 1971–1986 (PDF) (Technical report). CSTR. Bell Labs. p. 10. 139.*
- Ulf Bilting & Jan Skansholm "Vägen till C" (Swedish) meaning "The Road to C", third edition, Studentlitteratur, year 2000, page 3. ISBN 91-44-01468-6.
- Stallings, William. "Operating Systems: Internals and Design Principles" 5th ed, page 91. Pearson Education, Inc. 2005.
- *Kernighan, Brian W.; Ritchie, Dennis M. (March 1988). The C Programming Language (2nd ed.). Englewood Cliffs, NJ: Prentice Hall. ISBN 0-13-110362-8.*
- *Stroustrup, Bjarne (2002). Sibling rivalry: C and C++ (PDF) (Report). AT&T Labs.*
- *"JTC1/SC22/WG14 – C". Home page. ISO/IEC. Retrieved 2 June 2011.*
- *Andrew Binstock (October 12, 2011). "Interview with Herb Sutter". Dr. Dobbs. Retrieved September 7, 2013.*
- *"TR 18037: Embedded C" (PDF). ISO / IEC. Retrieved 26 July 2011.*
- *Harbison, Samuel P.; Steele, Guy L. (2002). C: A Reference Manual (5th ed.). Englewood Cliffs, NJ: Prentice Hall. ISBN 0-13-089592-X.* Contains a BNF grammar for C.
- *Kernighan, Brian W.; Ritchie, Dennis M. (1996). The C Programming Language (2nd ed.). Prentice Hall. p. 192. ISBN 7 302 02412 X.*
- Page 3 of the original K&R[1]
- ISO/IEC 9899:201x (ISO C11) Committee Draft
- *Kernighan, Brian W.; Ritchie, Dennis M. (1996). The C Programming Language (2nd ed.). Prentice Hall. pp. 192, 259. ISBN 7 302 02412 X.*
- *"10 Common Programming Mistakes in C++". Cs.ucr.edu. Retrieved 26 June 2009.*
- *Schultz, Thomas (2004). C and the 8051 (3rd ed.). Otsego, MI: PageFree Publishing Inc. p. 20. ISBN 1-58961-237-X. Retrieved 10 February 2012.*
- Page 6 of the original K&R[1]
- *Klemens, Ben (2013). 21st Century C. O'Reilly Media. ISBN 1-4493-2714-1.*
- *Feuer, Alan R.; Gehani, Narain H. (March 1982). "Comparison of the Programming Languages C and Pascal". ACM Computing Surveys **14** (1): 73–92. doi:10.1145/356869.356872. (subscription required (help)).*
- Page 122 of K&R2[14]
- For example, gcc provides _FORTIFY_SOURCE. *"Security Features: Compile Time Buffer Checks (FORTIFY_SOURCE)". fedoraproject.org. Retrieved 2012-08-05.*

- *Raymond, Eric S. (11 October 1996). The New Hacker's Dictionary (3rd ed.). MIT Press. p. 432. ISBN 978-0-262-68092-9. Retrieved 5 August 2012.*
- *Summit, Steve. "comp.lang.c Frequently Asked Questions 6.23". Retrieved March 6, 2013.*
- *Summit, Steve. "comp.lang.c Frequently Asked Questions 7.28". Retrieved March 6, 2013.*
- *"Man Page for lint (freebsd Section 1)". unix.com. 2001-05-24. Retrieved 2014-07-15.*
- *McMillan, Robert (2013-08-01). "Is Java Losing Its Mojo?". Wired.*
- *Dr. Dobb's Sourcebook. U.S.A.: Miller Freeman, Inc. November–December 1995.*
- *"Using C for CGI Programming". linuxjournal.com. 1 March 2005. Retrieved 4 January 2010.*
- *Stroustrup, Bjarne (1993). "A History of C++: 1979−1991" (PDF). Retrieved 9 June 2011.*
- *Ritchie, Dennis M. (1993). The Development of the C Language. The second ACM SIGPLAN History of Programming Languages Conference (HOPL-II) (Cambridge, MA, USA — April 20–23, 1993: ACM). pp. 201–208. doi:10.1145/154766.155580. ISBN 0-89791-570-4. Retrieved 2014-11-04.*
- *Gosling, James; Joy, Bill; Steele, Guy; Bracha, Gilad; Buckley, Alex (2014). The Java® Language Specification (PDF) (Java SE 8 ed.).*
- *Gosling, James; Joy, Bill; Steele, Guy L., Jr.; Bracha, Gilad (2005). The Java Language Specification (3rd ed.). Addison-Wesley. ISBN 0-321-24678-0.*
- *Lindholm, Tim; Yellin, Frank (1999). The Java Virtual Machine Specification (2nd ed.). Addison-Wesley. ISBN 0-201-43294-3.*
- *Stroustrup, Bjarne (1997). "1". The C++ Programming Language (Third ed.). ISBN 0-201-88954-4. OCLC 59193992.*
- *Naugler, David (May 2007). "C# 2.0 for C++ and Java programmer: conference workshop". Journal of Computing Sciences in Colleges **22** (5).* Although C# has been strongly influenced by Java it has also been strongly influenced by C++ and is best viewed as a descendant of both C++ and Java.
- *"Chapel spec (Acknowledgements)" (PDF). Cray Inc. 2015-10-01. Retrieved 2016-01-14.*
- *Stroustrup, B. (6 May 2014). "Lecture:The essence of C++. University of Edinburgh.". Retrieved 12 June 2015.*
- *Stroustrup, Bjarne (17 February 2014). "C++ Applications". stroustrup.com. Retrieved 5 May 2014.*
- *"ISO/IEC 14882:2011". International Organization for Standardization.*
- *Stroustrup, Bjarne (7 March 2010). "Bjarne Stroustrup's FAQ: When was C++ invented?". stroustrup.com. Retrieved 16 September 2010.*
- *Stroustrup, Bjarne. "Evolving a language in and for the real world: C++ 1991-2006" (PDF).*
- *Stroustrup, Bjarne. "A History of C ++ : 1979− 1991" (PDF).*
- *Stroustrup, Bjarne. "The C++ Programming Language" (First ed.). Retrieved 16 September 2010.*
- *Stroustrup, Bjarne. "The C++ Programming Language" (Second ed.). Retrieved 16*

September 2010.

- *"Bjarne Stroustrup's FAQ – Where did the name "C++" come from?". Retrieved 16 January 2008.*
- *"C For C++ Programmers". Northeastern University. Retrieved 7 September 2015.*
- *"ISO/IEC 14882:1998". International Organization for Standardization.*
- *"ISO/IEC 14882:2003". International Organization for Standardization.*
- *"ISO/IEC 14882:2014". International Organization for Standardization.*
- *"We have an international standard: C++0x is unanimously approved". Sutter's Mill.*
- *"The Future of C++".*
- *"We have C++14! : Standard C++".*
- Recent milestones: C++17 nearly feature-complete, second round of TSes now under development
- *"ISO/IEC TR 18015:2006". International Organization for Standardization.*
- *"ISO/IEC TR 19768:2007". International Organization for Standardization.*
- *"ISO/IEC TR 29124:2010". International Organization for Standardization.*
- *"ISO/IEC TR 24733:2011". International Organization for Standardization.*
- *"ISO/IEC TS 18822:2015". International Organization for Standardization.*
- *"ISO/IEC TS 19570:2015". International Organization for Standardization.*
- *"ISO/IEC TS 19841:2015". International Organization for Standardization.*
- *"ISO/IEC TS 19568:2015". International Organization for Standardization.*
- *"ISO/IEC TS 19217:2015". International Organization for Standardization.*
- See a list at http://en.cppreference.com/w/cpp/experimental Visited 16 January 2016.
- *B. Stroustrup (interviewed by Sergio De Simone) (30 April 2015). "Stroustrup: Thoughts on C++17 - An Interview". Retrieved 8 July 2015.*
- *Stroustrup, Bjarne (2000). The C++ Programming Language (Special ed.). Addison-Wesley. p. 46. ISBN 0-201-70073-5.*
- *Stroustrup, Bjarne. "Open issues for The C++ Programming Language (3rd Edition)". This code is copied directly from Bjarne Stroustrup's errata page (p. 633). He addresses the use of* '\n' *rather than* std::endl. *Also see Can I write "void main()"? for an explanation of the implicit* return 0; *in the* main *function. This implicit return is not available in other functions.*
- ISO/IEC (2003). *ISO/IEC 14882:2003(E): Programming Languages – C++ §6.6.3 The return statement [stmt.return]* para. 2
- ISO/IEC (2003). *ISO/IEC 14882:2003(E): Programming Languages – C++ §3.6.1 Main function [basic.start.main]* para. 5
- ISO/IEC. *Programming Languages – C++11 Draft (n3797) §3.7 Storage duration [basic.stc]*
- ISO/IEC. *Programming Languages – C++11 Draft (n3797) §3.7.1 Static Storage duration [basic.stc.static]*
- ISO/IEC. *Programming Languages – C++11 Draft (n3797) §3.7.2 Thread Storage duration [basic.stc.thread]*
- ISO/IEC. *Programming Languages – C++11 Draft (n3797) §3.7.3 Automatic Storage*

duration [basic.stc.auto]

- ISO/IEC. *Programming Languages – C++11 Draft (n3797) §3.7.4 Dynamic Storage duration basic.stc.dynamic*
- *"Nobody Understands C++: Part 5: Template Code Bloat". http://blog.emptycrate.com/: EmptyCrate Software. Travel. Stuff. 6 May 2008. Retrieved 8 March 2010.* On occasion you will read or hear someone talking about C++ templates causing code bloat. I was thinking about it the other day and thought to myself, "self, if the code does exactly the same thing then the compiled code cannot really be any bigger, can it?" [...] And what about compiled code size? Each were compiled with the command g++ <filename>.cpp -O3. Non-template version: 8140 bytes, template version: 8028 bytes!
- *Sutter, Herb; Alexandrescu, Andrei (2004). C++ Coding Standards: 101 Rules, Guidelines, and Best Practices. Addison-Wesley.*
- *Henricson, Mats; Nyquist, Erik (1997). Industrial Strength C++. Prentice Hall. ISBN 0-13-120965-5.*
- *Stroustrup, Bjarne (2000). The C++ Programming Language (Special ed.). Addison-Wesley. p. 310. ISBN 0-201-70073-5.* A virtual member function is sometimes called a method.
- *Mycroft, Alan (2013). "C and C++ Exceptions | Templates" (PDF). Cambridge Computer Laboratory - Course Materials 2013-14. Retrieved July 2014.*
- *Stroustrup, Bjarne (2013). The C++ Programming Language. Addison Wesley. p. 345. ISBN 9780321563842.*
- *Stroustrup, Bjarne (2013). The C++ Programming Language. Addison Wesley. pp. 363–365. ISBN 9780321563842.*
- *Stroustrup, Bjarne (2013). The C++ Programming Language. Addison Wesley. pp. 345, 363. ISBN 9780321563842.*
- *Stroustrup, Bjarne (2013). The C++ Programming Language. Addison Wesley. pp. 344, 370. ISBN 9780321563842.*
- *Stroustrup, Bjarne (2013). The C++ Programming Language. Addison Wesley. p. 349. ISBN 9780321563842.*
- *Graziano Lo Russo (2008). "An Interview with A. Stepanov". stlport.org. Retrieved 2015-10-08.*
- *"C++ ABI Summary". 20 March 2001. Retrieved 30 May 2006.*
- *"Bjarne Stroustrup's FAQ – Is C a subset of C++?". Retrieved 5 May 2014.*
- *"C9X – The New C Standard". Retrieved 27 December 2008.*
- *"C++0x Support in GCC". Retrieved 12 October 2010.*
- *"C++0x Core Language Features In VC10: The Table". Retrieved 12 October 2010.*
- *"Clang - C++98, C++11, and C++14 Status". Clang.llvm.org. 12 May 2013. Retrieved 10 June 2013.*
- *"Re: [RFC] Convert builin-mailinfo.c to use The Better String Library" (Mailing list). 6 September 2007. Retrieved 31 March 2015.*
- *"Re: Efforts to attract more users?" (Mailing list). 12 July 2010. Retrieved 31 March 2015.*
- *Andrew Binstock (18 May 2011). "Dr. Dobb's: Interview with Ken Thompson". Retrieved 7 February 2014.*
- *Kreinin, Yossi (October 13, 2009). "Defective C++". Retrieved February 3, 2016.*

- New Languages, and Why We Need Them, MIT Technology Review
- The New Native Languages | Dr Dobb's
- *Torgersen, Mads (October 27, 2008). "New features in C# 4.0". Microsoft. Retrieved October 28, 2008.*
- *Naugler, David (May 2007). "C# 2.0 for C++ and Java programmer: conference workshop". Journal of Computing Sciences in Colleges **22** (5).* Although C# has been strongly influenced by Java it has also been strongly influenced by C++ and is best viewed as a descendant of both C++ and Java.
- *Hamilton, Naomi (October 1, 2008). "The A-Z of Programming Languages: C#". Computerworld. Retrieved February 12, 2010.* We all stand on the shoulders of giants here and every language builds on what went before it so we owe a lot to C, C++, Java, Delphi, all of these other things that came before us. (Anders Hejlsberg)
- *"Chapel spec (Acknowlegements)" (PDF). Cray Inc. 2015-10-01. Retrieved 2016-01-14.*
- *"Web Languages and VMs: Fast Code is Always in Fashion. (V8, Dart) - Google I/O 2013". Google. Retrieved 22 December 2013.*
- Java 5.0 added several new language features (the enhanced for loop, autoboxing, varargs and annotations), after they were introduced in the similar (and competing) C# language [1] [2]
- *Cornelius, Barry (December 1, 2005). "Java 5 catches up with C#". University of Oxford Computing Services. Retrieved June 18, 2014.* In my opinion, it is C# that has caused these radical changes to the Java language. (Barry Cornelius)
- *Lattner, Chris (2014-06-03). "Chris Lattner's Homepage". Chris Lattner. Retrieved 2014-06-03.* The Swift language is the product of tireless effort from a team of language experts, documentation gurus, compiler optimization ninjas, and an incredibly important internal dogfooding group who provided feedback to help refine and battle-test ideas. Of course, it also greatly benefited from the experiences hard-won by many other languages in the field, drawing ideas from Objective-C, Rust, Haskell, Ruby, Python, C#, CLU, and far too many others to list.
- *C# Language Specification (PDF) (4th ed.). Ecma International. June 2006. Retrieved January 26, 2012.*
- *Lander, Rich (20 July 2015). "Announcing .NET Framework 4.6". .NET Blog. Microsoft.*
- *Zander, Jason (November 24, 2008). "Couple of Historical Facts". Retrieved February 23, 2009.*
- *Guthrie, Scott (November 28, 2006). "What language was ASP.Net originally written in?". Retrieved February 21, 2008.*
- *Hamilton, Naomi (October 1, 2008). "The A-Z of Programming Languages: C#". Computerworld. Retrieved October 1, 2008.*
- *Wylie Wong (2002). "Why Microsoft's C# isn't". CNET: CBS Interactive. Retrieved May 28, 2014.*
- *Bill Joy (February 7, 2002). "Microsoft's blind spot". cnet.com. Retrieved January 12, 2010.*
- *Klaus Kreft and Angelika Langer (2003). "After Java and C# - what is next?". Retrieved June 18, 2013.*

- *Klaus Kreft and Angelika Langer (July 3, 2003). "After Java and C# - what is next?". artima.com. Retrieved January 12, 2010.*
- *Osborn, John (August 1, 2000). "Deep Inside C#: An Interview with Microsoft Chief Architect Anders Hejlsberg". O'Reilly Media. Retrieved November 14, 2009*
- *"Generics (C# Programming Guide)". Microsoft. Retrieved March 21, 2011.*
- *Don Box and Anders Hejlsberg (February 2007). "LINQ: .NET Language-Integrated Query". Microsoft. Retrieved March 21, 2011.*
- *Mercer, Ian (April 15, 2010). "Why functional programming and LINQ is often better than procedural code". abodit.com. Retrieved March 21, 2011.*
- *"Andy Retires". Dan Fernandez's Blog. Blogs.msdn.com. January 29, 2004. Retrieved October 4, 2012.*
- *"Technical committees - JTC 1/SC 22 - Programming languages, their environments and system software interfaces". ISO. Retrieved October 4, 2012.*
- *"ISO/IEC 23270:2003 - Information technology - C# Language Specification". Iso.org. August 23, 2006. Retrieved October 4, 2012.*
- *"ISO/IEC 23270:2006 - Information technology - Programming languages - C#". Iso.org. January 26, 2012. Retrieved October 4, 2012.*
- *Kovacs, James (September 7, 2007). "C#/.NET History Lesson". Retrieved June 18, 2009.*
- *Hejlsberg, Anders (October 1, 2008). "The A-Z of Programming Languages: C#". Computerworld.*
- *"Microsoft C# FAQ". Microsoft. Retrieved March 25, 2008.* Archived February 14, 2006, at the Wayback Machine.
- *"Visual C#.net Standard" (JPEG). Microsoft. September 4, 2003. Retrieved June 18, 2009.*
- *"F# FAQ". Microsoft Research. Retrieved June 18, 2009.* Archived February 18, 2009, at the Wayback Machine.
- *Simon, Raphael; Stapf, Emmanuel; Meyer, Bertrand (June 2002). "Full Eiffel on the .NET Framework". Microsoft. Retrieved June 18, 2009.*
- *"Using C# 3.0 from .NET 2.0". Danielmoth.com. May 13, 2007. Retrieved October 4, 2012.*
- *"Mono and Roslyn". Tirania Blog. Miguel de Icaza. Retrieved 9 April 2014.*Work in progress for C# 5.0.
- *"What's new in the C# 2.0 Language and Compiler". Microsoft Developer Network. Microsoft. Retrieved 11 June 2014.*
- *Hejlsberg, Anders; Torgersen, Mads. "Overview of C# 3.0". Microsoft Developer Network. Microsoft. Retrieved 11 June 2014.*
- *Ghosh, Wriju. "C# 3.0 : Partial Methods". MSDN Blogs. Microsoft. Retrieved 11 June 2014.*
- *Burrows, Chris. "C# 4.0 - New C# Features in the .NET Framework 4". Microsoft Developer Network. Microsoft. Retrieved 11 June 2014.*
- *Hejlsberg, Anders. "Future directions for C# and Visual Basic". C# lead architect. Microsoft. Retrieved September 21, 2011.*
- *"An Introduction to New Features in C# 5.0". MSDN Blogs. Microsoft. Retrieved 11 June 2014.*
- *"Language feature implementation status". github. Microsoft. Retrieved 13 February 2015.*

- *"C# 7 Work List of Features · Issue #2136 · dotnet/roslyn". GitHub. Retrieved 2016-01-11.*
- *Visual Studio 2010 and .NET 4 Six-in-One. Wrox Press. 2010. ISBN 0470499486.*
- *Venners, Bill; Eckel, Bruce (August 18, 2003). "The Trouble with Checked Exceptions". Retrieved March 30, 2010.*
- *Archer, Tom (2001). "Part 2, Chapter 4: The Type System". Inside C#. Redmond, Washington: Microsoft Press. ISBN 0-7356-1288-9.*
- *Lippert, Eric (March 19, 2009). "Representation and Identity". Fabulous Adventures In Coding. Blogs.msdn.com. Retrieved October 4, 2012.*
- *"Patent Pledge for Open Source Developers".*
- *"Patent Cooperation Agreement - Microsoft & Novell Interoperability Collaboration". Microsoft. November 2, 2006. Retrieved July 5, 2009.* Microsoft, on behalf of itself and its Subsidiaries (collectively "Microsoft"), hereby covenants not to sue Novell's Customers and Novell's Subsidiaries' Customers for infringement under Covered Patents of Microsoft on account of such a Customer's use of specific copies of a Covered Product as distributed by Novell or its Subsidiaries (collectively "Novell") for which Novell has received Revenue (directly or indirectly) for such specific copies; provided the foregoing covenant is limited to use by such Customer (i) of such specific copies that are authorized by Novell in consideration for such Revenue, and (ii) within the scope authorized by Novell in consideration for such Revenue.
- *"Definitions". Microsoft. November 2, 2006. Retrieved July 5, 2009.*
- *Steinman, Justin (November 7, 2006). "Novell Answers Questions from the Community". Retrieved July 5, 2009.* We maintain that Mono does not infringe any Microsoft patents.
- *"Covenant to Downstream Recipients of Moonlight - Microsoft & Novell Interoperability Collaboration". Microsoft. September 28, 2007. Retrieved March 8, 2008.* "Downstream Recipient" means an entity or individual that uses for its intended purpose a Moonlight Implementation obtained directly from Novell or through an Intermediate Recipient... Microsoft reserves the right to update (including discontinue) the foregoing covenant... "Moonlight Implementation" means only those specific portions of Moonlight 1.0 or Moonlight 1.1 that run only as a plug-in to a browser on a Personal Computer and are not licensed under GPLv3 or a Similar License.
- https://github.com/dotnet/roslyn
- "Compatibility - Mono". Mono-project.com. December 19, 2011. Retrieved October 4, 2012.
- NI Instrument Driver Network
- *"Building a Stand-Alone Application". National Instruments.*
- *"Using the LabVIEW Run-Time Engine". National Instruments.*
- Why Dataflow Programming Languages are Ideal for Programming Parallel Hardware
- *Spolsky, Joel. "Why are the Microsoft Office file formats so complicated? (And some workarounds)". Retrieved March 8, 2009.*
- *"Can I Save VIs in My Current LabVIEW Version for Use in a Previous Version?". Retrieved February 1, 2016.*
- *"Can I Zoom In or Out on a LabVIEW Diagram (for Wiring or Viewing Purposes)?".*

Retrieved February 1, 2016.

- *"Add a zoom function (yes, I said zoom. So sue me)". forums.ni.com. Retrieved 2016-03-31.*
- http://www.ni.com/white-paper/5920/en/
- https://decibel.ni.com/content/docs/DOC-13859
- Embedding a C/C++ Interpreter Ch into LabVIEW for Scripting
- "We looked very carefully at Delphi Object Pascal and built a working prototype of bound method references in order to understand their interaction with the Java programming language and its APIs ... Our conclusion was that bound method references are unnecessary and detrimental to the language. This decision was made in consultation with Borland International, who had previous experience with bound method references in Delphi Object Pascal." (from About Microsoft's "Delegates" at java.sun.com.
- *TechMetrix Research (1999). "History of Java" (PDF). Java Application Servers Report.* The project went ahead under the name "green" and the language was based on an old model of UCSD Pascal, which makes it possible to generate interpretive code
- *"A Conversation with James Gosling - ACM Queue". Retrieved 11 August 2015.*
- Essential Pascal by Marco Cantù
- tiobe.com, Programming Community Index for January 2011.
- Hertzfeld, Andy. "Hungarian folklore.org: Macintosh Stories. Retrieved 2012-03-06.
- "An Interview with JOHN BRACKETT AND DOUG ROSS", p15, Charles Babbage Institute, 2004
- "AUSTRALIAN ATOMIC ENERGY COMMISSION RESEARCH ESTABLISHMENT, LUCAS HEIGHTS, NUCLEAR SCIENCE AND TECHNOLOGY BRANCH REPORT 1977, DIVISIONAL RESEARCH", p.22, International Atomic Energy Agency (IEAE)
- Jon Udell, Crash of the Object-Oriented Pascals, BYTE, July, 1989.
- M.I.Trofimov, The End of Pascal?, BYTE, March, 1990, p.36.
- doi:10.1016/0898-1221(87)90181-7
- PASCAL-XSC: PASCAL for Extended Scientific Computing
- *"XSC Software". Retrieved 11 August 2015.*
- *"Universitaet Wuppertal: Wissenschaftliches Rechnen / Softwaretechnologie". Retrieved 11 August 2015.*
- Michel Gien, "The SOL Operating System", in Usenix Summer '83 Conference, Toronto, ON, (July 1983), pp. 75-78
- cs.berkeley.edu
- Pascal ISO 7185:1990 6.10
- J. Welsh, W. J. Sneeringer, and C. A. R. Hoare, "Ambiguities and Insecurities in Pascal", *Software Practice and Experience 7*, pp. 685–696 (1977)
- *Pascal*, Nell Dale and Chip Weems, "Dangling Else", p. 160–161
- *"Flock-JSCodeGenDemo.7z - dwscript - "Flock" DWScript / JavaScript CodeGen demo - Delphi Web Script general purpose scripting engine - Google Project Hosting". Retrieved 11 August 2015.*
- "Pascal-S: A Subset and Its Implementation", N. Wirth in Pascal – The Language and Its Implementation, by D.W. Barron, Wiley 1979.

- *ISO/IEC 7185:1990 Pascal (PDF). Retrieved 16 September 2014.*
- *Wirth, Niklaus (July 1973). The Programming Language Pascal (Revised Report) (PDF). ETH Zürich. Retrieved 16 September 2014.*
- *Extended Pascal: ISO/IEC 10206:1990. Retrieved 16 September 2014.*
- *"Language standards: Pascal, Extended Pascal, Fortan". http://www.prosperosoftware.com/std.html.* External link in |website= (help);
- *770X3.160-1989 - IEEE/ANSI Standard for the Programming Language Extended Pascal. Retrieved 16 September 2014.*
- *"Virtual Pascal for OS/2". Retrieved 3 April 2016.*
- *"netlabs.org - Project: Open Sibyl". Retrieved 3 April 2016.*
- Brian W. Kernighan (1981). Why Pascal is Not My Favorite Programming Language
- O. Lecarme, P. Desjardins, "More Comments on the Programming Language Pascal," *Acta Informatica 4*, pp. 231–243 (1975)
- The Pascal Programming Language
- Pascal Myths
- Extended Pascal
- Extended Pascal - Chapter 3
- Free Pascal BREAK
- Free Pascal CONTINUE
- Free Pascal EXIT
- *Pascal Architecture*

- 2010. *Cambridge Advanced Learner's Dictionary*. Cambridge: Cambridge University Press. Dictionary online. Available from http://dictionary.cambridge.org/dictionary/british/metalanguage Internet. Retrieved 20 November 2010
- Hofstadter, Douglas. 1980. *Gödel, Escher, Bach: An Eternal Golden Braid*. New York: Vintage Books ISBN 0-14-017997-6
- *Harris, Zellig S. (1991). A theory of language and information: A mathematical approach. Oxford: Clarendon Press. pp. 272–318. ISBN 0-19-824224-7.*
- *Ibid*. p. 277.
- *Borel, Félix Édouard Justin Émile (1928). Leçons sur la theorie des fonctions (in French) (3 ed.). Paris: Gauthier-Villars & Cie. p. 160.*
- Hunter, Geoffrey. 1971. *Metalogic: An Introduction to the Metatheory of Standard First-Order Logic*. Berkeley:University of California Press ISBN 978-0-520-01822-8
- Ritzer, George. 1991. *Metatheorizing in Sociology*. New York: Simon Schuster ISBN 0-669-25008-2
- *Reddy, Michael J. 1979. The conduit metaphor: A case of frame conflict in our language about language. In Andrew Ortony (ed.), Metaphor and Thought. Cambridge: Cambridge University Press*

- *Kindler, E.; Krivy, I. (2011). "Object-Oriented Simulation of systems with sophisticated control". International Journal of General Systems: 313–343.*

- *Lewis, John; Loftus, William (2008). Java Software Solutions Foundations of Programming Design 6th ed. Pearson Education Inc. ISBN 0-321-53205-8.*, section 1.6 "Object-Oriented Programming"
- Deborah J. Armstrong. *The Quarks of Object-Oriented Development*. A survey of nearly 40 years of computing literature which identified a number of fundamental concepts found in the large majority of definitions of OOP, in descending order of popularity: Inheritance, Object, Class, Encapsulation, Method, Message Passing, Polymorphism, and Abstraction.
- John C. Mitchell, *Concepts in programming languages*, Cambridge University Press, 2003, ISBN 0-521-78098-5, p.278. Lists: Dynamic dispatch, abstraction, subtype polymorphism, and inheritance.
- Michael Lee Scott, *Programming language pragmatics*, Edition 2, Morgan Kaufmann, 2006, ISBN 0-12-633951-1, p. 470. Lists encapsulation, inheritance, and dynamic dispatch.
- *Pierce, Benjamin (2002). Types and Programming Languages. MIT Press. ISBN 0-262-16209-1.*, section 18.1 "What is Object-Oriented Programming?" Lists: Dynamic dispatch, encapsulation or multi-methods (multiple dispatch), subtype polymorphism, inheritance or delegation, open recursion ("this"/"self")
- *Booch, Grady (1986). Software Engineering with Ada. Addison Wesley. p. 220. ISBN 978-0805306088.* Perhaps the greatest strength of an object-oriented approach to development is that it offers a mechanism that captures a model of the real world.
- *Jacobsen, Ivar; Magnus Christerson; Patrik Jonsson; Gunnar Overgaard (1992). Object Oriented Software Engineering. Addison-Wesley ACM Press. pp. 43–69. ISBN 0-201-54435-0.*
- *McCarthy, J.; Brayton, R.; Edwards, D.; Fox, P.; Hodes, L.; Luckham, D.; Maling, K.; Park, D.; Russell, S. (March 1960). "LISP I Programmers Manual" (PDF). Boston, Massachusetts: Artificial Intelligence Group, M.I.T. Computation Center and Research Laboratory: 88f.* In the local M.I.T. patois, association lists [of atomic symbols] are also referred to as "property lists", and atomic symbols are sometimes called "objects".
- *McCarthy, John; Abrahams, Paul W.; Edwards, Daniel J.; Hart, swapnil d.; Levin, Michael I. (1962). LISP 1.5 Programmer's Manual (PDF). MIT Press. p. 105. ISBN 0-262-13011-4.* Object — a synonym for atomic symbol
- *"Dr. Alan Kay on the Meaning of "Object-Oriented Programming"". 2003. Retrieved 11 February 2010.*
- *Sutherland, I. E. (30 January 1963). "Sketchpad: A Man-Machine Graphical Communication System" (PDF). Technical Report No. 296, Lincoln Laboratory, Massachusetts Institute of Technology via Defense Technical Information Center (stinet.dtic.mil). Retrieved 3 November 2007.*
- The Development of the Simula Languages, Kristen Nygaard, Ole-Johan Dahl, p.254 Uni-kl.ac.at
- *Ross, Doug. "The first software engineering language". LCS/AI Lab Timeline:. MIT Computer Science and Artificial Intelligence Laboratory. Retrieved 13 May 2010.*
- *Holmevik, Jan Rune (1994). "Compiling Simula: A historical study of technological genesis" (PDF). IEEE Annals of the History of Computing 16 (4): 25–37. doi:10.1109/85.329756. Retrieved 12 May 2010.*

- *Hoare, C. A. (Nov 1965). "Record Handling". ALGOL Bulletin (21): 39–69. doi:10.1145/1061032.1061041.*
- *Kay, Alan. "The Early History of Smalltalk". Retrieved 13 September 2007.*
- 1995 (June) Visual FoxPro 3.0, FoxPro evolves from a procedural language to an object-oriented language. Visual FoxPro 3.0 introduces a database container, seamless client/server capabilities, support for ActiveX technologies, and OLE Automation and null support. Summary of Fox releases
- FoxPro History web site: Foxprohistory.org
- 1995 Reviewers Guide to Visual FoxPro 3.0: DFpug.de
- https://books.google.co.uk/books?id=MHmqfSBTXsAC&pg=PA16&lpg=PA16
- *"The Emerald Programming Language". 2011-02-26.*
- *Neward, Ted (26 June 2006). "The Vietnam of Computer Science". Interoperability Happens. Retrieved 2 June 2010.*
- Meyer, Second Edition, p. 230
- M.Trofimov, *OOOP - The Third "O" Solution: Open OOP.* First Class, OMG, 1993, Vol. 3, issue 3, p.14.
- *Yegge, Steve (30 March 2006). "Execution in the Kingdom of Nouns". steve-yegge.blogspot.com. Retrieved 3 July 2010.*
- *Boronczyk, Timothy (11 June 2009). "What's Wrong with OOP". zaemis.blogspot.com. Retrieved 3 July 2010.*
- *Ambler, Scott (1 January 1998). "A Realistic Look at Object-Oriented Reuse". www.drdobbs.com. Retrieved 4 July 2010.*
- *Shelly, Asaf (22 August 2008). "Flaws of Object Oriented Modeling". Intel Software Network. Retrieved 4 July 2010.*
- *James, Justin (1 October 2007). "Multithreading is a verb not a noun". techrepublic.com. Retrieved 4 July 2010.*
- *Shelly, Asaf (22 August 2008). "HOW TO: Multicore Programming (Multiprocessing) Visual C++ Class Design Guidelines, Member Functions". support.microsoft.com. Retrieved 4 July 2010.*
- *Robert Harper (17 April 2011). "Some thoughts on teaching FP". Existential Type Blog. Retrieved 5 December 2011.*
- *Cardelli, Luca (1996). "Bad Engineering Properties of Object-Oriented Languages". ACM Comput. Surv. (ACM) 28 (4es): 150. doi:10.1145/242224.242415. ISSN 0360-0300. Retrieved 21 April 2010.*
- Armstrong, Joe. In *Coders at Work: Reflections on the Craft of Programming.* Peter Seibel, ed. Codersatwork.com, Accessed 13 November 2009.
- *Stepanov, Alexander. "STLport: An Interview with A. Stepanov". Retrieved 21 April 2010.*
- Rich Hickey, JVM Languages Summit 2009 keynote, Are We There Yet? November 2009.
- *Potok, Thomas; Mladen Vouk; Andy Rindos (1999). "Productivity Analysis of Object-Oriented Software Developed in a Commercial Environment" (PDF). Software – Practice and Experience 29 (10): 833–847. doi:10.1002/(SICI)1097-024X(199908)29:10<833::AID-SPE258>3.0.CO;2-P. Retrieved 21 April 2010.*

- C. J. Date, Introduction to Database Systems, 6th-ed., Page 650
- C. J. Date, Hugh Darwen. *Foundation for Future Database Systems: The Third Manifesto* (2nd Edition)
- *Krubner, Lawrence. "Object Oriented Programming is an expensive disaster which must end". smashcompany.com. Retrieved 14 October 2014.*
- *Graham, Paul. "Why ARC isn't especially Object-Oriented.". PaulGraham.com. Retrieved 13 November 2009.*
- Stevey's Blog Rants
- *Eric S. Raymond (2003). "The Art of Unix Programming: Unix and Object-Oriented Languages". Retrieved 2014-08-06.*
- *Poll, Erik. "Subtyping and Inheritance for Categorical Datatypes" (PDF). Retrieved 5 June 2011.*
- *Abadi, Martin; Cardelli, Luca (1996). A Theory of Objects. Springer-Verlag New York, Inc. ISBN 0-387-94775-2. Retrieved 21 April 2010.*
- David A. Patterson, John L. Hennessy (2010). *Οργάνωση και σχεδίαση υπολογιστών η διασύνδεση υλικού και λογισμικού*. Εκδόσεις Κλειδάριθμος.
- *Behrouz A. Forouzan, Firouz Mosharraf (2010). Εισαγωγή στην επιστήμη των υπολογιστών. Εκδόσεις Κλειδάριθμος.*
- Αθ. Τσουρόπλης & Κ. Κλημόπουλος, Εισαγωγή στην Πληροφορική, Εκδόσεις Νέων Τεχνολογιών, 2005, ISBN 960-8105-84-6
- William Stallings, Οργάνωση & Αρχιτεκτονική Υπολογιστών, Εκδόσεις Τζίολα, 2003, ISBN 960-418-008-8
- *Andrew S. Tanenbaum, Σύγχρονα Λειτουργικά Συστήματα, Εκδόσεις Κλειδάριθμος, 2007, ISBN 960-209-586-5*
- Monitor specifications, TFT Central. Ανακτήθηκε 16/11/2009. (Αγγλικά)
- GLOSSARY LCD-MONITORS, www.prad.de. Ανακτήθηκε 16/11/2009. (Αγγλικά)
- *A Complete Guide to the Digital Video Interface*, DataPro International Inc. Ανακτήθηκε 16/11/2009. (Αγγλικά)
- *Digital Visual Interface, Digital Display Working Group, 1999. Ανακτήθηκε 16/11/2009. (Αγγλικά)*
- *Miller, Stephen W. (1977), Memory and Storage Technology, Montvale.: AFIPS Press*
- *Memory and Storage Technology, Alexandria, Virginia.: Time Life Books, 1988*

Παράρτημα Γλώσσες Προγραμματισμού

Απαρίθμηση γνωστών γλωσσών
Εισαγωγή

Η πληροφορική είναι η επιστήμη που ερευνά την κωδικοποίηση, διαχείριση και μετάδοση συμβολικών αναπαραστάσεων πληροφοριών. Επίσης εξετάζει τη σχεδίαση, υλοποίηση και βελτιστοποίηση αυτοματοποιημένων διατάξεων, συσκευών, υπηρεσιών και συστημάτων συλλογής, αποθήκευσης, επεξεργασίας, εξόρυξης και ανταλλαγής των εν λόγω αναπαραστάσεων. Είναι η μεγάλη ικανότητα των υπολογιστών στην επεξεργασία των πληροφοριών η οποία οδήγησε στην ανάγκη της δημιουργίας της επιστήμης των υπολογιστών η οποία έχει σαν σκοπό να βάλει τους υπολογιστές να συνεπικουρήσουν τηξ πληροφορική. Δεν θα πρέπει ποτέ να ξεχνάμε ποτέ το ρητό του Έντσγκερ Ντάικστρα (Edsger Dijkstra), διάσημου Ολλανδού επιστήμονα της πληροφορικής, ο οποίος είχε δηλώσει:
"Η πληροφορική έχει τόση σχέση με τους υπολογιστές, όση έχει η αστρονομία με το τηλεσκόπιο."
Καθώς λοιπόν διαφάνηκε η ανάγκη εισόδου των μηχανών στην διαχείριση και επεξεργασία της πληροφορίας και καθώς τα υπολογιστικά συστήματα άρχισαν τον εξελικτικό τους αγώνα δρόμου έκαναν την εμφάνιση τους και διάφορες διάλεκτοι με τις οποίες ήταν δυνατή η επικοινωνία ανθρώπου – μηχανής, μιλάμε βεβαίως για τις γλώσσες προγραμματισμού.

Γλώσσα προγραμματισμού λέγεται μια τεχνητή γλώσσα που μπορεί να χρησιμοποιηθεί για τον έλεγχο μιας μηχανής, συνήθως ενός υπολογιστικού συστηατος. Οι γλώσσες προγραμματισμού (όπως άλλωστε και οι ανθρώπινες γλώσσες) ορίζονται από ένα σύνολο συντακτικών και εννοιολογικών κανόνων, που ορίζουν τη δομή και το νόημα, αντίστοιχα, των προτάσεων της γλώσσας.

Οι γλώσσες προγραμματισμού χρησιμοποιούνται για να διευκολύνουν την οργάνωση και διαχείριση πληροφοριών, αλλά και για την ακριβή διατύπωση αλγορίθμων. Ορισμένοι ειδικοί χρησιμοποιούν τον όρο *γλώσσα προγραμματισμού* μόνο για τυπικές που μπορούν να εκφράσουν όλους τους πιθανούς αλγορίθμους. Μη-υπολογιστικές γλώσσες όπως η HTML ή τυπικές γραμματικές όπως η BNF δεν λέγονται συνήθως γλώσσες προγραμματισμού.

Υπάρχουν χιλιάδες διαφορετικές γλώσσες προγραμματισμού, και κάθε χρόνο δημιουργούνται περισσότερες.

Ακολουθεί λίστα με όλες τις γνωστές γλώσσες προγραμματισμού. Η καταγραφή των γλωσσών θα γίνει αλφαβητικά.

Αλφαβητική καταγραφή

A

- A# .NET
- A# (Axiom)
- Ada
- Adenine

14. Apex (Salesforce.com)

15. APL

- A-0 System
- A+
- A++
- ABAP
- ABC
- ABC ALGOL
- ABLE
- ABSET
- ABSYS
- ACC
- Accent
- Ace DASL
- ACL2
- ACT-III
- Action!
- ActionScript
- Agda
- Agilent VEE
- Agora
- AIMMS
- Alef
- ALF
- ALGOL 58
- ALGOL 60
- ALGOL 68
- ALGOL W
- Alice
- Alma-0
- AmbientTalk
- Amiga E
- AMOS
- AMPL

16. App Inventor for Android's visual block language
17. AppleScript
18. Arc
19. ARexx
20. Argus
21. AspectJ
22. Assembly language
23. ATS
24. Ateji PX
25. AutoHotkey
26. Autocoder
27. AutoIt
28. AutoLISP / Visual LISP
29. Averest
30. AWK
31. Axum

B

- B
- Babbage
- Bash
- BASIC
- bc
- BCPL
- BeanShell
- Batch (Windows/Dos)
- Bertrand
- BETA
- Bigwig
- Bistro
- BitC
- BLISS
- BlooP
- Blue
- Boo
- Boomerang
- Bourne shell (including bash and ksh)
- BREW
- BPEL

C

- C
- C--
- C++ – ISO/IEC 14882
- C# – ISO/IEC 23270
- C/AL
- Caché ObjectScript
- C Shell
- Caml
- Cayenne
- CDuce
- Cecil
- Cel
- Cesil
- Ceylon
- CFEngine
- CFML
- CHILL
- CHIP-8
- chomski
- ChucK
- CICS
- Cilk
- CL (IBM)
- Claire
- Clarion
- Clean
- Clipper
- CLIST
- Clojure
- CLU
- CMS-2
- COBOL – ISO/IEC
- Combined Programming Language (CPL)
- COMIT
- Common Intermediate Language (CIL)
- Common Lisp (also known as CL)
- COMPASS
- Component Pascal
- Constraint Handling Rules (CHR)
- Converge
- Cool
- Coq
- Coral 66
- Corn
- CorVision
- COWSEL

- Cg
- Ch
- Chapel
- CHAIN
- Charity
- Charm
- Chef

1989

- Cobra
- CODE
- CoffeeScript
- Cola
- ColdC
- ColdFusion
- COMAL
- CPL
- csh
- CSP
- Cryptol
- Csound
- CUDA
- Curl
- Curry
- Cyclone
- Cython

D

- D
- DASL (Datapoint's Advanced Systems Language)
- DASL (Distributed Application Specification Language)
- Dart
- DataFlex
- Datalog
- DATATRIEVE
- dBase
- dc
- DCL
- Deesel (formerly G)
- Delphi
- DinkC
- DIBOL
- Dog
- Draco
- DRAKON
- Dylan
- DYNAMO

E

- E
- E#
- Ease
- Easy PL/I
- Easy Programming Language
- EASYTRIEVE PLUS
- ECMAScript
- Edinburgh IMP
- EGL
- Eiffel
- ELAN
- Elixir
- Elm
- Emacs Lisp
- Emerald
- Epigram
- EPL
- Erlang
- es
- Escher
- ESPOL
- Esterel
- Etoys
- Euclid
- Euler
- Euphoria
- EusLisp Robot Programming Language
- CMS EXEC (EXEC)
- EXEC 2
- Executable UML

F

- F
- F#
- Factor
- Falcon
- Fantom
- FAUST
- FFP
- Fjölnir
- Flavors
- Flex
- FlooP
- FLOW-MATIC
- FOCAL
- FOCUS
- FOIL
- FORMAC
- Fortran – ISO/IEC 1539
- Fortress
- FoxBase
- FoxPro
- FP
- FPr
- Franz Lisp
- Frege

- FL
- @Formula
- F-Script
- Forth

G

- G
- Game Maker Language
- GameMonkey Script
- GAMS
- GAP
- G-code
- Genie
- GDL
- GJ
- GEORGE
- GLSL
- GNU E
- GM
- Go
- Go!
- GOAL
- Gödel
- Godiva
- Golo
- GOM (Good Old Mad)
- Goo
- Google Apps Script
- Gosu
- GOTRAN
- GPSS
- GraphTalk
- GRASS
- Groovy

H

- Hack
- HAL/S
- Hamilton C shell
- Harbour
- Hartmann pipelines
- Haskell
- Haxe
- High Level Assembly
- HLSL
- Hop
- Hope
- Hugo
- Hume
- HyperTalk

I

- IBM Basic assembly language
- IBM HAScript
- IBM Informix-4GL
- IBM RPG
- ICI
- Icon
- Id
- IDL
- Idris
- IMP
- Inform
- Io
- Ioke
- IPL
- IPTSCRAE
- ISLISP
- ISPF
- ISWIM

J

- J
- J#
- J++
- JADE
- Jako
- JAL
- Janus
- JASS
- Java
- JavaScript
- JCL
- JEAN
- Join Java
- JOSS
- Joule
- JOVIAL
- Joy
- JScript
- JScript .NET
- JavaFX Script
- Julia
- Jython

K

- K
- Kaleidoscope
- Karel
- Karel++
- KEE
- Kixtart
- Klerer-May System
- KIF
- Kojo
- Kotlin
- KRC
- KRL
- KRL (KUKA Robot Language)
- KRYPTON
- ksh

L

- L
- L# .NET
- LabVIEW
- Ladder
- Lagoona
- LANSA
- Lasso
- LaTeX
- Lava
- LC-3
- Leda
- Legoscript
- LIL
- LilyPond
- Limbo
- Limnor
- LINC
- Lingo
- Linoleum
- LIS
- LISA
- Lisaac
- Lisp – ISO/IEC 13816
- Lite-C
- Lithe
- Little b
- Logo
- Logtalk
- LotusScript
- LPC
- LSE
- LSL
- LiveCode
- LiveScript
- Lua
- Lucid
- Lustre
- LYaPAS
- Lynx

M

- M2001
- M4
- M#
- Machine code
- MAD (Michigan Algorithm Decoder)
- MAD/I
- Magik
- Magma
- make
- Maple
- MAPPER now part of BIS
- MARK-IV now VISION:BUILDER
- Mary
- MASM Microsoft Assembly x86
- Mathematica
- MATLAB
- Maxima (see also Macsyma)
- Max (Max Msp – Graphical Programming Environment)
- MaxScript internal language 3D Studio Max
- Maya (MEL)
- MDL
- Mercury
- Mesa
- Metacard
- Metafont
- Microcode
- MicroScript
- MIIS
- MillScript
- MIMIC
- Mirah
- MirandN
- NASM
- NATURAL
- Napier88
- Neko
- ML
- Moby
- Model 204
- Modelica
- Modula
- Modula-2
- Modula-3
- Mohol
- MOO
- Mortran
- Mouse
- MPD
- MSIL – deprecated name for CIL
- MSL
- MUMPS
- Mystic Programming Language (MPL)

- Nemerle
- nesC
- NESL
- Net.Data
- NetLogo
- NetRexx
- NewLISP
- NEWP
- Newspeak
- NewtonScript
- NGL
- Nial
- Nice
- Nickle
- Nim
- NPL
- Not eXactly C (NXC)
- Not Quite C (NQC)
- NSIS
- Nu
- NWScript
- NXT-G
- a
- MIVA Script

N

- NASM
- NATURAL
- Napier88
- Neko
- Nemerle
- nesC
- NESL
- Net.Data
- NetLogo
- NetRexx
- NewLISP
- NEWP
- Newspeak
- NewtonScript
- NGL
- Nial
- Nice
- Nickle
- Nim
- NPL
- Not eXactly C (NXC)
- Not Quite C (NQC)
- NSIS
- Nu
- NWScript
- NXT-G

O

- o:XML
- Oak
- Oberon
- OBJ2
- Obliq
- OCaml
- occam
- occam-π
- OpenEdge ABL
- OPL
- OPS5
- OptimJ

- Object Lisp
- ObjectLOGO
- Object REXX
- Object Pascal
- Objective-C
- Objective-J
- Octave
- OmniMark
- Onyx
- Opa
- Opal
- OpenCL
- Orc
- ORCA/Modula-2
- Oriel
- Orwell
- Oxygene
- Oz

P

- P″
- P#
- ParaSail (programming language)
- PARI/GP
- Pascal – ISO 7185
- PCASTL
- PCF
- PEARL
- PeopleCode
- Perl
- PDL
- Perl6
- Pharo
- PHP
- Phrogram
- Pico
- Picolisp
- Pict
- Pike
- PIKT
- PILOT
- Pipelines
- Pizza
- PL-11
- PL/0
- PL/B
- PL/C
- PL/I – ISO 6160
- PL/M
- PL/P
- PL/SQL
- PL360
- PLANC
- Plankalkül
- Planner
- PLEX
- PLEXIL
- Plus
- POP-11
- PostScript
- PortablE
- Powerhouse
- PowerBuilder – 4GL GUI applcation generator from Sybase
- PowerShell
- PPL
- Processing
- Processing.js
- Prograph
- PROIV
- Prolog
- PROMAL
- Promela
- PROSE modeling language
- PROTEL
- ProvideX
- Pro*C
- Pure
- Python

Q

- Q (equational programming language)
- Q (programming language from Kx Systems)
- Qalb
- QtScript
- QuakeC
- QPL

R

- R
- R++
- Racket
- RAPID
- Rapira
- Ratfiv
- Ratfor
- rc
- Red
- Redcode
- REFAL
- Reia
- Revolution
- rex
- REXX
- Rlab
- ROOP
- RPG
- RPL
- RSL
- RTL/2
- Ruby
- RuneScript
- Rust

- REBOL

S

- S
- S2
- S3
- S-Lang
- S-PLUS
- SA-C
- SabreTalk
- SAIL
- SALSA
- SAM76
- SAS
- SASL
- Sather
- Sawzall
- SBL
- Scala
- Scheme
- Scilab
- Scratch
- Script.NET
- Sed
- Seed7
- Self
- SenseTalk
- SequenceL
- SETL
- SIMPOL
- SIGNAL
- SiMPLE
- SIMSCRIPT
- Simula
- Simulink
- SISAL
- SLIP
- SMALL
- Smalltalk
- Small Basic
- SML
- Snap!
- SNOBOL(SPITBOL)
- Snowball
- SOL
- Span
- SPARK
- Speedcode
- SPIN
- SP/k
- SPS
- SQR
- Squeak
- Squirrel
- SR
- S/SL
- Stackless Python
- Starlogo
- Strand
- Stata
- Stateflow
- Subtext
- SuperCollider
- SuperTalk
- Swift (Apple programming language)
- Swift (parallel scripting language)
- SYMPL
- SyncCharts
- SystemVerilog

T

- T
- TACL
- TACPOL
- TADS
- TAL
- Tcl
- Tea
- TECO
- TELCOMP
- TeX
- TEX
- TIE
- Timber
- TMG, compiler-compiler
- Tom
- TOM
- TouchDevelop
- Topspeed
- TPU
- Trac
- TTM
- T-SQL
- TTCN
- Turing
- TUTOR
- TXL
- TypeScript
- Turbo C++

U

- Ubercode
- UCSD Pascal
- Umple
- Unicon
- Uniface
- UNITY
- Unix shell
- UnrealScript

V

- Vala
- VBA
- VBScript
- Visual Basic
- Visual Basic .NET
- Visual DataFlex
- Visual DialogScript
- Visual Fortran
- Visual FoxPro
- Visual J++
- Visual J#
- Visual Objects
- Visual Prolog
- VSXu
- vvvv

W

- WATFIV, WATFOR
- WebDNA
- WebQL
- Windows PowerShell
- Winbatch
- Wolfram Language
- Wyvern

X

- X++
- X#
- X10
- XBL
- XC (exploits XMOS architecture)
- xHarbour
- XL
- Xojo
- XOTcl
- XPL
- XPL0
- XQuery
- XSB
- XSLT – see XPath
- Xtend

Y

- Yorick
- YQL

Z

- Z notation
- Zeno
- ZOPL
- ZPL

Σύντομη περιγραφή γλωσσών

Βεβαίως το πλήθος των γλωσσών προκαλεί ίλιγγο αν όμως σκεφτούμε ότι δεν είναι τίποτα άλλο από εργαλεία τα οποία χρησιμοποιούνται για να λύσουν προβλήματα απολύτως διαφορετικά μεταξύ τους, από απλούς μαθηματικούς υπολογισμούς μέχρι δύσκολες αστρονομικές προβλέψεις τροχιών και από αναπαραστάσεις τρισδιάστατων γραφικών μέχρι ιδιαίτερα πολύπλοκους συσχετισμούς κατηγορημάτων στην τεχνητή νοημοσύνη, τότε είναι ξεκάθαρος ακόμα και στον αδαή παρατηρητή ο λόγος ανάπτυξης και ύπαρξης τόσο πολλών και διαφορετικών μεταξύ τους εργαλείων – γλωσσών.
Παρακάτω ακολουθούν κάποιες βασικές πληροφορίες για κάποιες από τις γλώσσες που είναι μέχρι σήμερα γνωστές.

A Sharp (.NET)
Η συγκεκριμένη γλώσσα προγραμματισμού είναι μια πύλη της γλώσσας προγραμματισμού ADA στην πλατφόρμα Microsoft .NET. Αναπτύχθηκε αρχικά από το Τμήμα Επιστήμης Υπολογιστών της

Ακαδημίας Πολεμικής Αεροπορίας των Η.Π.Α.

A Sharp (Axiom)
Πρόκειται για μια αντικειμενοστραφή γλώσσα προγραμματισμού η οποία δίδεται σαν ξεχωριστό συστατικό της v2 του υπολογιστικού αλγεβρικού συστήματος Axiom.

A-0
Το A-0 σύστημα, (Arithmetic Language version 0) γράφτηκε από τον Grace Hopper το 1951 και 1952 για το υπολογιστικό σύστημα UNIVAC 1 (πρόκειται για τον πρώτο εμπορικό υπολογιστή που παράχθηκε στις Η.Π.Α.) και αποτέλεσε τον πρώτο compiler που αναπτύχθηκε για έναν ηλεκτρονικό υπολογιστή. Το A-0 ακολουθήσαν οι A-1, A-2, A-3 (κυκλοφόρισε ως ARITH-MATIC), AT-3 (κυκλοφόρισε ως MATH-MATIC) και B-0 (κυκλοφόρισε ως FLOW-MATIC). Ο πηγαίος κώδικας της A-2 δόθηκε στους πελάτες καλώντας τους να στείλουν πίσω στους κατασκευαστές τις βελτιώσεις τους επί του συστήματος καθιστώντας έτσι την A-2 ένα πρώιμο και ίσως το πρώτο παράδειγμα λογισμικού ανοιχτού κώδικα.

A+
Η A+ είναι μια γλώσσα προγραμμτισμού η οποία προέκυψε ως απόγονος της γλώσσας A η οποία με την σειρά της φτιάχθηκε για να αντικαταστήσει την APL γλώσσα προγραμματισμού το 1988. Ο Arthur Whitney ουστιαστικά είναι ο υπέυθυνος για το A κομμάτι της A+ και διάφοροι άλλοι συναδελφοι του στο Morgan Stanley την επέκτειναν προσθέτοντας γραφικό περιβάλλον διεπαφής καθώς και επιπλέον χαρακτηριστικά του συνόλου που τελικά ονομάστηκε A+. Η A+ σχεδιάστηκε έχοντας κατά νου ιδιαίτερα απαιτητικούς αριθμητικούς υπολογισμούς και κυρίως αυτούς που συναντούμε στις οικονομικές εφαρμογές. Η A+ τρέχει σε διάφορες εκδόσεις του UNIX μεταξύ των οποίων και στο LINUX.

A++
Η A++ αναπτύχθηκε το 2002 από τους Δρ. Barry και Δρ. Hamm. Σχεδιάστηκε περισσότερο σαν ένα εργαλείο εκμάθησης παρά σαν μια γλώσσα με σκοπό την επίλυση υπολογιστικών – πληροφοριακών προβλημάτων. Σκοπός τους ήταν να αποτελέσει η γλώσσα ένα αποτελεσματικό εργαλείο για την εξοικείωση του μαθητή με τον πυρήνα του προγραμματισμού. Μοιάζει αρκετά με την C++.

ABAP
ABAP είναι το ακρωνύμιο για την έκφραση **A**dvanced **B**usiness **A**pplication **P**rogramming. Πρόκειται για μια από τις πολλές γλώσσες προγραμματισμού εξειδικευμένης εφαρμογής τέταρτης γενιάς (4GLs) η οποία αναπτύχθηκε στην δεκαετία του 1980 από την Γερμανική εταιρία λογισμού SAP. Αρχικά αποτέλεσε την γλώσσα αναφοράς για την πλατφόρμα R/2 της SAP, μια πλατφόρμας η οποία επέτρεπε σε μεγάλους οργανισμούς να φτιάξουν επιχειρηματικές εφαρμογές σχεδιασμένες να τρέχουν σε mainframes. Αποτέλεσε την βάση για την R/3 πλατφόρμα της SAP και παραμένει η γλώσσα δημιουργίας των client – server εφαρμογών για την πλατφόρμα R/3.

ABC
Η ABC είναι μια γενικού σκοπού γλώσσα προγραμματισμού η οποία αναπτύχθηκε από τους Leo Geurts, Lambert Meertens και Steven Pemberton στο CWI της Ολλανδίας το 1987. Είναι διαδραστική, δομημένη, υψηλού επιπέδου και φτιάχτηκε με σκοπό να αντικαταστήσει τις Basic, Pascal ή AWK. Είναι περισσότερο προσανατολισμένη στην διδασκαλία παρά στο να αποτελέσει γλώσσα προγραμμτισμού συστημάτων.

ABC ALGOL
Η ABC ALGOL είναι μια επέκταση της γλώσσας προγραμματισμού ALGOL 60 με κύρια στόχευση τα μαθηματικά. Παρά τα προτερήματα της έναντι της ALGOL 60 ποτέ δεν χρησιμοποιήθηκε τόσο

εκτενώς όσο η απλή ALGOL.

ABEL
Η ABEL (Advanced Boolean Expression Language) είναι μια γλώσσα περιγραφής υλικού και ένα σχετικό σετ σχεδιαστικών εργαλείων για τον προγραμματισμό των PLDs. Δημιουργήθηκε το 1983 από την Data I/O Corporation στο Redmont της Washington.

ABSET
Πρώιμη γλώσσα προγραμματισμού από το πανεπιστήμιο του Aberdeen.

ABSYS
Πρώιμη γλώσσα προγραμματισμού από πανεπιστήμιο του Aberdeen. Εμφανίστηκε το 1967. Είναι από τις πρώτες υλοποιήσεις γλώσσας λογικού προγραμματισμού και είχε μέσα της αρκετά από τα χαρακτηριστικά της Prolog. Το όνομα ABSYS είναι ακρονύμιο από το Aberdeen System.

ACC
Η ACC είναι μια γλώσσα προγραμματισμού πολύ κοντινή με την C φτιαγμένη για το MS-DOS των προσωπικών υπολογιστών της IBM. Τόσο ο compiler όσο και τα μεταγλωττισμένα προγράμματα μπορούν να τρέξουν σε οποιονδήποτε υπολογιστή με επεξεργαστή intel 80386 ή νεότερο ο οποίος τρέχει MS-DOS.

ACCENT
Ένας πολύ υψηλού επιπέδου μεταφραστής ο οποίος εκδόθηκε το 1990 από την εταιρία CaseWare. Ο προσανατολισμός της ήταν τα αλφαρηθμιτικά και οι μήτρες.

Ace DASL
Η γλώσσα προγραμματισμού DASL (Distributed Application Specification Language) είναι μια γλώσσα προγραμματισμού υψηλού επιπέδου η οποία αρχικά αναπτύχθηκε στην Sun Microsystems Laboratories μεταξύ 1990 και 2003 ως μέρος του Ace project. Ο κύριος στόχος της DASL ήταν η ταχεία ανάπτυξη εφαρμογών ιστού βασισμένες στην J2EE αρχιτεκτονική της SUN εξουδετερώνοντας την ανάγκη εκμάθησης δύσκολων λεπτομερειων που αφορούσαν τον τρόπο λειτουργίας της εκάστοστε πλατφόρμας. Η DASL παράγει το gui της εφαρμογής κατευθείαν από την λογική περιγραφή. Ο προγραμματιστής μπορεί εκ των υστέρων να τροποποιήσει την εικόνα και την λειτουργικότητα του gui.

ACL2
Η ACL2 (A Computational Logic for Applicative Common Lisp) είναι ένα σύστημα λογισμικολυ το οποίο αποτελείται από μια γλώσσα προγραμματισμού, μια επεκτάσιμη θεωρία βασισμένη σε μια λογική σειράς και ένα μηχανισμό απόδειξης μαθηματικών θεωρημάτων με υπολογιστή. Σκοπός της ACL2 είναι να υποστηρίξει μια αυτοματοποιημένη συλλογιστική στις επαγωγικές λογικές θεωρίες κυρίως για τους σκοπούς της επαλήθευσης της ορθής λειτουργίας λογισμικού και υλικού. Σχεδιάστηκε το 1990 από τους Robert S. Boyer, J. Strother Moore και Matt Kaufmann. Έτος κυκλοφορίας 1996.

ACTION
Η ACTION είναι μια γλώσσα προγραμματισμού και ένα περιβάλλον ανάπτυξης λογισμικού ταυτόχρονα με ενσωματωμένο συντάκτη, μεταγλωττιστή και αποσφαλματωτή για την οικογένεια υπολογιστών Atari 8-bit. Η ανάπτυξη έγινε από τον Clinton Parker και κυκλοφόρησε το 1983 από την εταιρία Optimized Systems Software. Δύο γνωστά εμπορικά συστήματα τα οποία έγιναν με χρήση της ACTION είναι η Homepack σουίτα παραγωγικότητας και το Games Computer Play. Η γλώσσα δεν εισήχθει ποτέ σε άλλα συστήματα πέραν του αρχικού για το οποίο προοριζόταν.

ActionScript
Η ActionScript είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού η οποία αναπτύχθηκε από την εταιρία Macromedia το 1998 και σχεδιάστηκε από τον Gary Grossman. Πρόκειται για ένα υπερσύνολο του συντακτικού και της σημειολογίας της Javascript. Χρησιμοποιείται κυρίως για την ανάπτυξη ιστοσελίδων και λογισμικού προσανατολισμένου στον Adobe Flash Player.

ADA
Η ADA είναι μια γλώσσα προγραμματισμού υψηλού επιπέδου η οποία πρωτοεμφανίστηκε το 1980. Η ADA βελτιώνει την ασφάλεια του κώδικα και την συντήρηση του χρησιμοποιώντας τον μεταγλωττιστή για τον εντοπισμό των λαθών μη μπορώντας όμως έτσι αν εντοπίσει τα λογικά λάθη στον κώδικα. Η ADA αρχικά σχεδιάστηκε από μια ομάδα με αρχηγό τον Jean Ichbiah του CLL Honeywell Bull κατόπιν παραγγελλίας του Υπουργείου Άμυνας των Η.Π.Α. από το 1977 έως το 1983 με στόχο να σταματήσει την χρήση των δεκάδων διαφορετικών γλωσσών προγραμματισμού που χρησιμοποιούσε ως τότε το Υπουργείο.

AGDA
Η AGDA είναι μια γλώσσα προγραμματισμού η οποία σχεδιάστηκε και αναπτύχθηκε από τον Ulf Norell και πρωτοεμφανίστηκε το 2007.

AIMMS
Η AIMMS (Advanced Interactive Multidimensional Modeling System είναι ένα σύστημα λογισμικού σχεδιασμένο για την δημιουργία μοντέλων και για την επίλυση προβλημάτων βελτιστοποίησης καθώς και προβλημάτων χρονοδρομολόγησης. Ουσιαστικά αποτελείται από μια αλγεβρικού μοντέλου γλώσσα και ένα ενσωματωμένο περιβάλλον ανάπτυξης λογισμικού τόσο για την αλλαγή των μοντέλων όσο και για την δημιουργία guis γύρω από τα μοντέλα αυτά. Σχεδιαστηκε από τον Johannes J. Bisschop και από τον Marcel Roelofs και πρωτοεμφανίστηκε το 1993. Θεωρείται μια από τις πέντε σημαντικότερες γλώσσες αλγεβρικού μοντέλου.

ALF
Η ALF (Algebraic Logic Funtctional programming language) είναι μια γλώσσα προγραμματισμού η οποία συνδυάζει διαδικασιακό και λογικό προγραμματισμό. Ο θεμέλιος λίθος της είναι λογική ρήτρα Horv με ισότητα η οποία περιέχει κατηγορήματα και ρήτρες Horn για τον λογικό προγραμματισμό και συναρτήσεις και εξισώσεις για τον διαδικασιακό προγραμματισμό. Η ALF θεωρείται μια γλώσσα προγραμματισμού τεχνητής νοημοσύνης.

ALGOL
Η οικογένεια γλωσσων προγραμματισμού ALGOL η οποία περιλαμβάνει τα παρακάτω μέλη. ALGOL-58, ALGOL-60, ALGOL-68. Οι επίσημες εκδόσεις της ALGOL ονομαζίτονται βάση του έτους πρώτης κυκλοφορίας τους. Η ALGOL 68 είναι ουσιαστικά διαφορετική από την 60 και για αυτό δέχθηκε μερικώς κριτική. Σχεδιαστές της ALGOL, Backus, Bauer, Green, Katz, McCarthy, Naur, Perlis, Rutishauser, Samelson, van Wijingaarden, Vauquiois, Wegstein, Woodger. Η οικογένεια γλωσσών ALGOL αποτέλεσε την βάση για πολλές επόμενες γλώσσες προγραμματισμού όπως B, Pascal, C...

ALGOL W
Η ALGOL W είναι μια γλώσσα προγραμματισμού η οποία παρουσιάστηκε από τους Niklaus Wirth και Tony Hoare σαν διάδοχος της ALGOL 60. Η αρμόδια επιτροπή απεφάνθη ότι στην πραγματικότητα δεν αποτελούσε ουσιαστική αναβάθμιση από την ALGOL 60. Η υλοποίηση της έγινε σε PL/360 μια γλώσσα assembly η οποία έμοιαζε με την ALGOL και την οποία είχε επίσης σχεδιάσει ο Wirth.

ALICE ML

Η ALICE ML είναι μια γλώσσα προγραμματισμού η οποια σχεδιάστηκε από το Programming Systems Lab του Πανεπιστημίου του Saarland. Πρόκειται ουσιαστικά για μια διάλεκτο της κανονικής ML με κάποια επιπλέον χαρακτηριστικά. Πρωτοεμφανίζεται το 2000.

Alma-0

Πρόκειται ουσιαστικά για μια επαυξημένη έκδοση της Modula-2 μαζί με λογικό προγραμματισμό και δυνατότητα βάδισης προς τα πίσω. Σχεδιάστηκε από τους Krzysztof Apt, Marc Bezem, Jacob Brunekree, Vincent Partington, Andrea Schaerf και πρωτοεμφανίστηκε το 1997. Οι σχεδιαστές της ισχυρίζονται ότι οι λύσεις με προσανατολισμό αναζήτησης οι οποίες είναι ενσωματωμένες στην Alma-0 είναι κατά πολυ΄απλούστερες από τις λύσεις που είχε υλοποιήσει ο ανταγωνισμός.

AMIGA E

Η εν λόγω γλώσσα προγραμματισμού φτιάχτηκε από τον Wouter van Oortmerssen στην Amiga. Η Amiga E είναι ένα συνοθύλευμα χαρακτηριστικών από διάφορες γλώσσες προγραμματισμού, ωστόσο ακολουθεί πιστά τα χνάρια της C.

AMPL

Η AMPL (A Mathematical Programming language) είναι μια γλώσσα αλγεβρικού μοντέλου για την περιγραφή και την επίλυση ιδιαίτερα πολύπλοκων προβλημάτων που απαιτούν μαθημαατικούς υπολογισμούς μεγάλης κλίμακας. Φτιάτχτηκε από τους Robert Fourer, David Gay και Brian Kernighan στα Bell Laboratories το 1985.

APEX

Πρόκειται για μια ιδιόκτητη γλώσσα προγραμματισμού η οποία παρέχεται από την Force.com πλατφόρμα στους προγραμματιστές και μοιάζει πάρα πολυ με την Java.

APL

Η γλώσσα προγραμματισμού APL (από τον τίτλο του βιβλίου A Programming Language) αναπτύχθηκε την δεκαετία του 1960 από τον Kenneth E. Iverson. Ο κεντρικός τύπος δεδομένων που έχει είναι η πολυδιάστατη μήτρα. Χρησιμοποιεί μια πλειάδα γραφικών συμβόλων ώστε να αναπαραστήσει τους περισσότερους operators με αποτέλεσμα ο παραγόμενος κώδικας να είναι εξαιρετικά συνοπτικός. Επηρέασε σημαντικά το κομμάτι των μοντέλων, τα υπολογιστικά φύλλα και τον διαδικασιακό προγραμματισμό. Χρησιμοποιείται ακόμα και σήμερα για κάποιες εφαρμογές.

App Invertor

Πρόκειται για μια ανοιχτού κώδικα εφαρμογή ιστού η οποία αρχικά παρεχόταν από την Google και πλέον συντηρείται από το MIT. Επιτρέπει στους νεοεισερχόμενους στον προγραμματισμό να φτιάξουν εφαρμογές λογισμικού για το λειτουργικό σύστημα Android. Χρησιμοποιεί γραφικό περιβάλλον διεπαφής το οποίο με διαδικασίες σύρε – άφησε επιτρέπει την δημιουργία εφαρμογών οι οποίες τρέχουν στο Android. Η πρώτη της έκδοση δόθηκε στο κοινό το 2010.

AppleScript

Πρόκειται για μια γλώσσα δέσμης ενεργειών (scripting) η οποία αναπτύχθηκε από την εταιρία Apple το 1993 και ενσωματώθηκε στο λειτουργικό της από την έκδοση 7 και έπειτα. Έχει περιορισμένες επεξεργαστικές δυνατότητες, κάνει βασικούς υπολογισμούς, μπορεί να πολύπλοκη επεξεργασία κειμένου και είναι επεκτάσιμη.

ARC

Πρόκειται για μια διάλεκτο της Lisp η οποία αναπτύχθηκε από τους Paul Graham και Robert Morris. Δόθηκε στο κοινό το 2008.

Arexx
Πρόκειται για μια υλοποίηση της γλώσσας REXX γαι την Amiga η οποία γράφτηκε το 1987 από τον William S. Hawes υλοποιώντας χαρακτηριστικά της Amiga πέραν των καθιερωμένων χαρακτηριστικών της REXX. Είναι μια γλώσσα μεταφραστή (interpeter) και όχι μεταγλωττιστή (compiler).

Argus
Είναι μια γλώσσα προγραμματισμού η οποία δημιουργήθηκε στο MIT από την Barbara Liskov σε συνεργασία με τους Maurice Herlihym, Paul Johnson, Robert Scheifer και William Weihl. Πρόκειται ουσιαστικά για μια επέκταση της γλώσσας CLU και ενσωματώνει τα περισσότερα από το συντακτικό της και τη σημειολογία της.

Assembly
Μια γλώσσα assembly είναι μια γλώσσα προγραμματισμού χαμηλού επιπέδου για έναν υπολογιστή ή κάποια άλλη προγραμματιζόμενη συσκευή στην οποία υπάρχει ένας πολυ ισχυρός δεσμός ανάμεσα στην γλώσσα και στην αρχιτεκτονική της γλώσσας μηχανής της συσκευής. Κάθε γλώσσα assembly είναι συγκεκριμένη για κάθε υπολογιστική αρχιτεκτονική. Ιστορικά η πρώτη γλώσσα assembly έκανη την εμφάνιση της το 1949.

Averest
Πρόκειται για μια σύγχρονη γλώσσα προγραμματισμού και μια σουίτα εργαλείων για τον προσδιορισμό, πιστοποίηση και υλοποίηση αναδραστικών συστημάτων. Περιέχει έναν μεταγλωττιστή για τα σύγχρονα προγράμματα και ένα εργαλείο για την σύνθεση υλικού/λογισμικού. Μπορεί να χρησιμοποιηθεί για να μοντελοποιήσει πεπερασμένα και άπειρα συστήματα, σε διάφορα επίπεδα αφαίρεσης. Είναι επίσης χρήσιμη για τον σχεδιασμό του υλικού, την μοντελοποίηση των επικοινωνιών κλπ.

AWK
Είναι μια γλώσσα μετάφρασης σχεδιασμένη για επεξεργασία κειμένου και τυπικά χρησιμοποιείται για εξαγωγή δεδομένων και σαν εργαλείο αναφορών. Είναι στάνταρ χαρακτηριστικό σε κάθε τύπου Unix λειτουργικό σύστημα. Πρωτοεμφανίστηκε το 1977 και σχεδιάστηκε από τους Alfred Aho, Peter Weinberger και Brian Kernighan.

Axum
Είναι μια γλώσσα προγραμματισμού η οποία προορίζεται για την δημιουργία συγχρονισμένου λογισμικού. Βασίστηκε στο μοντέλο Actor και δημιουργήθηκε από την Microsoft κατά τα έτη 2009-2011.

B
Η B είναι μια γλώσσα προγραμματισμού η οποία εξελίχτηκε στα Bell Labs το 1969 και είναι το αποτέλεσμα της δουλειάς των Ken Thompson και Dennis Ritchie. Η B σχεδιάστηκε ως μια αναδρομική, μη αριθμητική και με ανεξαρτησία πλατφόρμας γλώσσα κατάλληλη για την συγγραφή λογισμικού για γλώσσες και συστήματα.

Babbage
Υψηλού επιπέδου γλώσσα Assembly η οποία εμφανίστηκε το 1971 για την GEC 4000 σειρά μικροϋπολογιστών. Πήρε το όνομα της από τον Charles Babbage Βρετανό ιχνηλάτη της επιστήμης των υπολογιστών.

Bash
Γλώσσα εντολών του φλοιού του Unix. Γράφτηκε από τον Brian Fox για το προτζεκτ GNU σαν αντικαταστάτης του φλοιού Bourne. Δόθηκε στο κοινό το 1989 σας ο φλοιός του λειτουργικού συστήματος GNU και σαν ο προκαθορισμένος φλοιός για το Linux και το OS X.

BASIC
Ακρωνύμιο για το Beginner's All-purpose Symbolic Instruction Code. Πρόκειται ουσιαστικά για μια ολόκληρη οικογένεια γλωσσών προγραμματισμού υψηλού επιπέδου των οποίων η φιλοσοφία έγκειται στην ευκολία χρήσης. Το 1964 οι John G. Kemeny και Thomaw E. Kurtz σχεδίασαν την αρχική BASIC γλώσσα ώστε να μπορέσουν οι φοιτητές και από άλλα επιστημονικά πεδία πέραν της πληροφορικής και των μαθηματικών να χρησιμοποιήσουν υπολογιστές. Την εποχή εκείνη όλοι οι υπολογιστές απαιτούσαν την συγγραφή του λογισμικού τους κάτι το οποίο έτειναν να μαθαίνουν και να κάνουν μόνο οι επιστήμονες των μαθηματικών και της πληροφορικής.

BC
Πρόκειται για ακρωνύμιο του Basic Calculator. Πρόκειται για μια γλώσσα υπολογμισμών η οποία μοιάζει πάρα πολύ με την C. Συνήθως χρησιμοποιείται σαν μια μαθηματική script γλώσσα ή σαν ένας διαδραστικός φλοιός υπολογισμών. Πρωτοεμφανίστηκε στην 6 έκδοση του Unix και την έχουν γραψει οι Robert Morris και Lorinda Cherry των Bell Labs.

BCPL
Είναι ακρωνύμιο του Basic Combined Programming Language. Σχεδιάστηκε από τον Martin Richards στο πανεπιστήμιο του Cambridge το 1966. Ο αρχικός λόγος ύπαρξης της BCPL ήταν η συγγραφή άλλων μεταγλωττιστών και πλέον δεν είναι σε χρήση. Ωστόσο η επιρροή της είναι ακόμη και σήμερα αρκετά αισθητή μιας και από την BCPL προήλθε η γλώσσα προγραμματισμού B πάνω στην οποια βασίστηκε η C.

BeanShell
Πρόκειται για μια script γλώσσα η οποία μοιάζει με την Java. Σχεδιάστηκε από τον Patrick Niemeyer το 1966 και τρέχει το περιβάλλον JRE χρησιμοποιώντας συντακτικό αντίστοιχο της Java.

Batch file
Batch file είναι ένα είδος script αρχείου σε DOS, OS/2 και Windows. Περιέχει μια σειρά εντολών οι οποίες πρόκειται να εκτελεστούν από τον μεταφραστή της γραμμής εντολών.

Bertrand
Η συγκεκριμένη γλώσσα προγραμματισμού πήρε το όνομα της από τον Bertrand Russel και σχεδιάστηκε από τον Wm Leler στα μέσα της δεκαετίας του 1980. Πρόκειται για μια γλώσσα προγραμματισμού που σκοπό έχει την ανάπτυξη συστημάτων περιορισμένου προγραμματισμού.

BETA
Η BETA είναι μια γνήσια γλώσσα αντικειμενοστραφούς προγραμματισμού η οποία δημιουργήθηκε εντός της “Σκανδιναβικής Σχολής” αντικειμενοστραφούς προγραμματισμού όπου άλλωστε δημιουργήθηκε και η πρώτη αντικειμενοστραφής γλώσσα η Simula. Ο σχεδιασμός της έγινε από τους Birger Moll3r-Pedersen και Kristen Nygaard. Είναι η γλώσσα η οποία εισήγαγε τις ένθετες κλάσεις και τις ενοποιημένες κλάσεις με διαδικασίες.

Bigwig
Πρόκειται για μιάγλωσσα προγραμματισμού υψηλού επιπέδου σχεδιασμένη για να χτίζει εφαρμογές ιστού. Εσωματώνεται σαν τμήμα του Apeche HTTP σέρβερ. Από το 2011 και μετά υπάρχουν εκδόσεις για Linux, Solaris αλλά όχι για Windows. Αναπτύχθηκε στο ερευνητικό κέντρο Basic Research in Computer Science στο τμήμα Επιστήμης Υπολογιστών του πανεπιστημίου Aarhus στη Δανία και δανείζεται πολλά χαρακτηριστικά από την mawl. Πρόκειται για λογισμικό ανοιχτού κώδικα.

Bistro

Είναι μια γλώσσα προγραμματισμού η οποία σχεδιάστηκε και αναπτύχθηκε από τον nicolas Boyd το 1999. Σκοπός της είναι η ενσωμάτωση χαρακτηριστικών της Smaltalk και της Java ώστε να τρέχει σαν μια διαφοροποιημένη Smaltalk πάνω από οποιαδήποτε java virtualmachine.

BitC

Είναι μια γλώσσα προγραμματισμού η οποία αναπτύχθηκε από του ερευνητές του πανεπιστημίου John Hopkins και το The Eros γκρουπ σαν μέρος του προγράμματος Coyotos το 2006. Σκοπός της είναι η πιστοποίηση προγραμμάτων.

BLISS

Πρόκειται για μια γλώσσα προγραμματισμού συστημάτων η οποία αναπτύχθηκε στο πανεπιστήμιο Carnegie Mellon από τους W. A. Wulf, D. B. Russell και A. N. Haberman γύρω στο 1970. Πιθανώς πρόκειται για την πλέον γνωστή γλώσσα προγραμματισμού συστημάτων μέχρι και την χρονική στιγμή εμφάνισης της C μερικά χρόνια αργότερα. Από το χρονικό σημείο εκείνο και μετά η C πήρε τα πρωτεία και η BLISS χάθηκε στη λήθη. Κατά την αεναρκτήρια πορεία της C υπήρχαν αρκετά έργα στα Bell Labs σε αντιπαράθεση σχετικά με τα οφέλει της BLISS επί της C.

BlooP και FlooP

Οι BlooP και FlooP είναι απλές γλώσσες προγραμματισμού οι οποίες σχεδιάστηκαν από τον Douglas Hofstadter προκειμένου να στηρίξουν τα επιχειρήματά του στο βιβλίο Godel, Escher, Bach. Η BlooP είναι μη ολοκληρωμένη κατά Turing της οποίας ο κύριος έλεγχος είναι ένας δεσμευμένος βρόχος (και άρα η αναδρομή δεν είναι εφικτή). Η FlooP από την άλλη είναι ολόιδια με την BlooP με την διαφορά ότι μπορεί να αναπαραστήσει μη δεσμευμένους βρόχους. Είναι ολοκληρωμένη κατά Turing και μπορεί να εκφράσει όλες τις υπολογιστικές συναρτήσεις. Οι BlooP και FlooP μπορούν να θεωρηθούν ως υπολογιστικά μοντέλα και κάποιες φορές χρησιμοποιούνται στην διδασκαλία υπολογισμών.

Blue

Η Blue είναι ένα σύστημα για την διασκαλία του αντικειμενοστραφούς προγραμματισμού. Αναπτύχθηκε στο πανεπιστήμιο του Σίδνεϋ και είναι διαθέσιμη από το 1997. Είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης λογισμικού. Η ανάπτυξη της σταμάτησε όταν ένας εκ των ηγήτορων της ανάπτυξης της ο Michael Kolling άρχισε να εφαρμόζει το σχέδιο του περιβάλλοντος ανάπτυξης λογισμικού στην Java με αποτέλεσμα το BlueJ.

Boo

Η Boo είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού γενικού σκοπού με στόχο την ανάπτυξη εφαρμογών ιστού. Αναπτύχθηκε από τον Rodrigo B. De oliveira το 2003.

Boomerang

Είναι μια γλώσσα προγραμματισμού με σκοπό την συγγραφή φακών (αμφίδρομοι μετασχηματισμοί ολοκληρωμένης συμπεριφοράς) οι οποίοι λειτουργούν για την συγκεκριμένη περίπτωση σε δεδομένα κειμένου. Σχεδιάστηκε από τους Nate Foster, Benjamin C. Pierce και Michael Greenberg και εμφανίστηκε το 2008.

Bourne Shell

Ο φλοιός Bourne ήταν ο κάθε αυτό φλοιός του Unix από την έκδοση 7 και μετά. Πρόκειται για ένα μεταφραστή γραμμής εντολών ο οποίος αναπτύχθηκε από τον Stephen Bourne στα Bell Labs και αποτέλεσε τον αντικαταστάτη του φλοιού Thompson. Δόθηκε στο κοινό το 1977. Παρόλο που χρησιμοποιείται σαν μεταφραστής γραμμής εντολών είχε επίσης σκοπό να είναι και μια γλώσσα δέσμης ενεργειών και εμπεριέχει τα περισσότερα από τα χαρακτηριστικά τα οποία χρησιμοποιούνται στον δομημένο προγραμματισμό.

BREW

Αρκτικόλεξο για το Binary Runtime Enviroment for Wireless. Πρόκειται για μια πλατφόρμα ανάπτυξης εφαρμογών η οποία δημιουργήθηκε από την Qualcomm με σκοπό την υποστήριξη εφαρμογών τρίτων όπως είναι τα παιχνίδια και οι μικροεφαρμογές στα κινητά τηλέφωνα.

BPEL
Το πραγματικό όνομα της συγκεκριμένης γλώσσας προγραμματισμού είναι Web Services Business Process Execution Language. Πρόκειται για μια σύμφωνα με το OASIS εκτέλσιμη γλώσσα γαι τον προσδιορισμό δράσεων μέσα σε διαδικασίες business με υπηρεσίες ιστού.

C
Είναι μια γλώσσα προγραμματισμού γενικού σκοπού. Αρχικά αναπτύχθηκε από τον Dennis Ritchie μεταξύ 1969 και 1973 στα At&T Bell Labs και χρησιμοποιήθηκε για την επανυλοποίηση του λειτουργικού συστήματος Unix. Έκτοτε έγινε η ευρύτερα χρησιμοποιούμενη γλώσσα προγραμματισμού όλων των εποχών μη μεταγλωττιστές C από διάφορους παρόχους να καλύπτουν την πλειοψηφία των υπαρχόντων υπολογιστικών αρχιτεκτονικών και λειτουργικών συστημάτων.

C--
Είναι μια γλώσσα προγραμματισμού που μοιάζει αρκετά με την C. Δημιουργήθηκε από τους ερευνητές του λειτουργικού προγραμματιμού Simon Peyton Jonew και Norman Ramsey το 1997. Σχεδιάστηκε να μπορεί να παραχθεί κυρίως από μεταγλωττιστές γλωσσών υπερ-υψηλού επιπέδου παρά από ανθρώπους προγραμματιστές.

C++
Πρόκειται για γλώσσα προγραμματισμού γενικού σκοπού η οποία είναι αντικειμενοστραφής με γενικά προγραμματιστικά χαρακτηριστικά καθώς επίσης και δυνατότητες χαμηλού επιπέδου διαχείρισης μνήμης. Σχεδιάστηκε από τον Bjarne Stroustrup και πρωτοεμφανίστηκε το 1983. Πριν την αρχική της τυποποίηση το 1998 η C++ αναπτυσσόταν στα Bell Labs από τον Stroustrup από το 1979 σαν μια επέκταση της C καθώς αυτός ήθελε μια βασδη αποτελεσματική, ευέλικτη και κοντά στην C η οποία επίσης παρείχε πολύ υψηλές δυνατότητες στην ανάπτυξη λογισμικού.

C#
Η γλώσσα προγραμματισμού C Sharp όπως προφέρεται αναπτύχθηκε από την Microsoft μέσα στα όρια της .NET πρωτοβουλίας και αργότερα έγινε αποδεκτή ως πρότυπο από την Ecma και την ISO. Σκοπός της είναι να είναι μια απλή, μοντέρνα και γενικού σκοπού αντικειμενοστραφής γλώσσα προγραμματισμού η οποία εμφανίστηκε το 2000 με αρχηγό εξέλιξης τον Anders Hejlsberg.

C/AL
Ακρωνύμιο του Client/Server Application Language, πρόκειται για μια γλώσσα η οποία χρησιμοποιείται εντός των ορίων του C/SIDE που είναι το Client/Server ολοκληρωμένο περιβάλλον στο Microsoft Dynamics Nav. Η C/AL είναι μια συγκεκριμένης στόχευσης γλώσσα προγραμματισμού η οποία κυρίως χρησιμοποιείται για την ανάκτηση, εισαγωγή και τροποποίηση εγγραφών σε βάσεις δεδομένων Navision. Ο αρχικός μεταγλωτιστής C/AL γράφτηκε από τον Michael Nielsen.

Cache ObjectScript (COS)
Πρόκειται για τμήμα του Cache συστήματος βάσεων δεδομένων η οποία πωλείται από την InterSystems. Η γλώσσα είναι ουσιαστικά ένα υπερσύνολο της MUMPS. Πρωτοεμφανίστηκε το 1997.

C shell
Είναι ένας φλοιός Unix ο οποίος κατασκευάστηκε από τον Bill Joy όταν ήταν ακόμα προπτυχιακός

φοιτητής στο Berkley της California στο τελείωμα της δεκαετίας του 1970.

Cduce
Είναι μια γλώσσα προγραμματισμού με XML προσανατολισμό η οποία ουσιαστικά επεκτείνει την Xduce.

Cecil
Είναι μια καθαρά αντικειμενοστραφής γλώσσα προγραμματισμού η οποία αναπτύχθηκε από τον Craig Chambers στο πανεπιστήμιο της Washington το 1992 σαν μέρος του σχεδίου Vortex. Η Cecil έχει πολλές ομοιότητες με άλλες αντικειμενοστραφείς γλώσσες προγραμματισμού και οι κύριοι σχεδιαστικοί στόχοι ήταν η επεκτασιμότητα, το τετράγωνη λογική, η αποτελεσματικότητα και η ευκολία χρήσης.

Cel
Είναι μια πρωτότυπη αντικειμενοστραφής γλώσσα προγραμματισμού βασισμένη στην Self με σκοπό να φτιαχτεί μια έκδοση της Self η οποία θα μπορούσε να τρέξει σε πολλά λειτουργικά συστήματα χωρίς ισχυρούς δεσμούς στο Self GUI. Πρωτοεμφανίστηκε το 1998.

Cesil
Ακρωνύμιο για το Computer Education in School Instruction Language. Πρόκειται για μια γλώσσα προγραμματισμού με σκοπό να εισαγάγει τους μαθητές των Βρετανικών σχολείων στην γλώσσα Assembly. Δεν μπορεί να χαρακτηριστεί ο υψηλού ή χαμηλού επιπέδου και αποτελείται από 14 εντολές.

CFEngine
Είναι ανοιχτού κώδικα σύστημα διαχείρισης και διαμόρφωσης το οποίο φτιάχτηκε από τον Mark Burges το 2015. Πρωταρχικη του λειτουργία είναι να παρέχει αυτόματη διαμόρφωση και συντήρηση σε μεγάλης κλίμακας υπολογιστικά συστήματα, μεταξύ των οποίων και η ενοποιημένη διαχείριση σε σέρβερ, προσωπικούς υπολογιστές, δικτυακές συσκευές, κινητά τηλέφωνα και τάμπλετ.

CFML
Η CFML (ColdFusion Markup Language) είναι μια γλώσσα δέσμης ενεργειών για ανάπτυξη εφαρμογών ιστού η οποία τρέχει σε JVM, .NET framework και στην Google App Engine. Φτιάχτηκε από τον Jeremy Allaire το 1995.

Cg
Η Cg (C for Graphics) είναι μια γλώσσα προγραμματισμού υψηλού επιπέδου η οποία αναπτύχθηκε από την Nvidia σε στενή συνεργασία με την Microsoft για τον προγραμματισμό του vertex και του pixel shader. Πρόκειται για μια γλώσσα η οποία είναι βασισμένη στην C και παρόλο που μοιράζονται το ίδιο συντακτικό, μερικά χαρακτηριστικά της C έχουν τροποποιηθεί και κάποιοι τύποι δεδομένων έχουν προστεθεί ώστε να είναι η Cg περισσότερο κατάλληλη για τον προγραμματισμό των μονάδων γραφικών.

Ch
Είναι μια ιδιόκτητη σουίτα πολλαπλής πλατφόρμας η οποία περιέχει μεταφραστές C και C++. Αρχικά σχεδιάστηκε από τον Harry H. Cheng σας μια γλώσσα δέσμης ενεργειών ώστε αρχάριοι να εισάγονται στον μαθηματικό προγραμματισμό και στον προγραμματισμό των C/C++. Εμφανίστηκε το 2001 και πλέον αναπτύσσεται και βρίσκεται στην αγορά υπό την SoftIntegration. Μπορεί να ενσωματωθεί σε όλες τις C/C++ εφαρμογές και είναι κατάλληλη τόσο για μαθηματικούς υπολογισμούς όσο και για γραφικό σχεδιασμό.

Chapel

Η Chapel (Cascade High Productivity Language) είναι μια παράλληλη γλώσσα προγραμματισμού η οποία αναπτύχθηκε από την Cray. Αναπτύχθηκε σαν μέρος του Cray Cascade Project, που είναι ουσιαστικά η συμμετοχή της Cray στο High Productivity Computing Systems πρόγραμμα της DARPA που κάνει την έρευνα και ανάπτυξη του στρατού των Η.Π.Α.. Σκοπός ήταν να αυξηθεί η πραγωγικότητα των υπερυπολογιστων μέχρι το έτος 2010. Οι σχεδιαστές της Chapel είναι οι David Callahan, Hans Zima, Brad Chamberlain και john Pievyak. Η πρώτη της εμφάνιση έγινε το 2009.

Charity

Πρόκειται για μια πειραματική "αγνά λειτουργική" (purely funtcional) γλώσσα προγραμματισμού η οποία αναπτύχθηκε στο Πανεπιστήμιο του Calgary κάτω από την εποπτεία του Robin Cocket το 1992, ο οποίος βασίστηλκε στις ιδεές του Hagino Tatsuya.

Charm

Η γλώσσα προγραμματισμού Charm είναι μια αντικειμενοστραφής γλώσσα η οποία εμφανίστηκε το 1996. Έχει αρκετές ομοιότητες με τις RTL/2, Pascal και C αλλά και πολλά επιπλέον χαρακτηριστικά σε σχέση με αυτές. Σχεδιάστηκε από τον P. Nowosad για το λειτουργικό σύστημα RISC OS.

JAVA

Η java είναι μια γενικού σκοπού αντικειμενοστραφής γλώσσα προγραμματισμού και σχεδιάστηκε έτσι ώστε να έχει όσο το δυνατόν λιγοτερες εξαρτήσεις υλοποίησης και εφαρμογής. Το μότο κατά την εμφάνιση της Java το 1995 ήταν το "WORA" (Write Once Run Anywhere) με την έννοια ότι ο μεταγλωτισμένος κώδικας Java θα έπρεπε να μπορεί να τρέχει σε οποιαδήποτε πλατφόρμα χωρίς να χρειάζεται επαναμεταγλώτιση. Σε πολύ μεγάλο αυτό επετεύχθη. Πράγματι ο κώδικας java μεταγλωτίζεται σε bytecode ο οποίος εν συνεχεία μπορεί να τρέξει θεωρητικά απροβλημάτιστα σε κάθε JVM ανεξάρτητα από την υπολογιστική αρχιτεκτονική που φιλοξενεί την JVM. Η JAVA κατέληξε να είναι από τις δημοφιλέστερες γλώσσες προγραμματισμού ιδιαιτέρως σε εφαρμογές Client Server. Αρχικά αναπτύχθηκε από τον James Gosling στην Sun Microsystems. Το μεγαλύτερο μέρος του συντακτικού της Java προέρχεται από τις C και C++ αλλά έχει κατά πολύ λιγότερες χαμηλού επιπέδου δεξιότητες και από τις δύο προηγούμενες.

Pascal

Η Pascal είναι μια γλώσσα προγραμματισμού η οποία σχεδιάστηκε το 1968-69 και κυκλοφόρησε το 1970 από τον Niklaus Wirth σαν μια μικρή και αποτελεσματική γλώσσα η οποία ενθάρρυνε τις καλές προγραμματιστικές τεχνικές όπως τον δομημένο προγραμματισμό και την δόμηση των δεδομένων. Το 1985 αναπτύχθηκε και η αντικειμενοστραφής έκδοση της η Object Pascal.