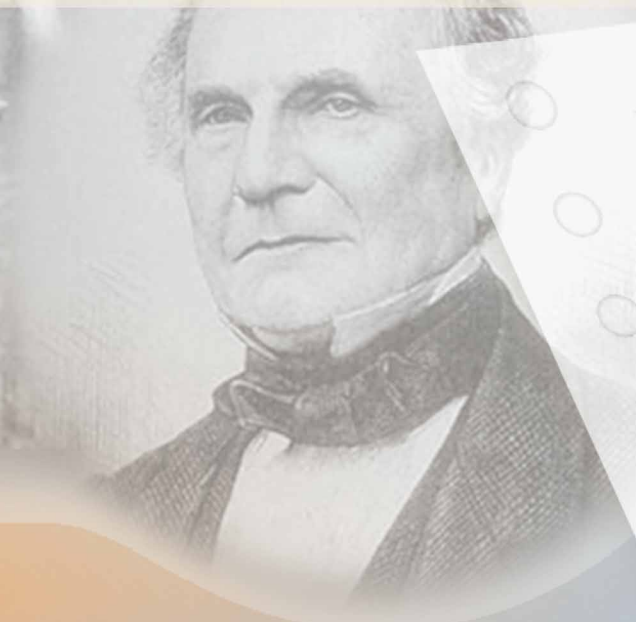
- **Με ποιο τρόπο ο ίδιος υπολογιστής μπορεί να χρησιμοποιηθεί για τη δημιουργία αρχιτεκτονικών σχεδίων αλλά και για τη διαχείριση μιας αποθήκης;**
- **Τι είναι και σε τι χρησιμεύουν οι γλώσσες προγραμματισμού;**
- **Πώς γράφονται τα προγράμματα και πώς εκτελούνται από τον υπολογιστή;**
- **Τι είναι η γλώσσα μηχανής και ποιες θεωρούνται γλώσσες υψηλού επιπέδου;**



ΚΕΦΑΛΑΙΟ 7

Προγραμματισμός υπολογιστή

Στόχος

Να γνωρίσουμε:

- **Τις γλώσσες προγραμματισμού.**
- **Τη διαδικασία κατασκευής προγραμμάτων.**
- **Τις αρχές και τα πρότυπα προγραμματισμού.**
- **Τα προγραμματιστικά περιβάλλοντα.**

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΥΠΟΛΟΓΙΣΤΗ

Η ειδοποιός διαφορά του υπολογιστή από τις άλλες ηλεκτρονικές κατασκευές του ανθρώπου είναι η δυνατότητα **προγραμματισμού** του, με το πρόγραμμα μάλιστα να καθορίζει κάθε στιγμή τη λειτουργία του. Η πληθώρα των θεμάτων που σχετίζονται με τον προγραμματισμό του υπολογιστή είναι τόσο μεγάλη που, όπως είναι φυσικό, αποτέλεσε ένα πολύ μεγάλο κλάδο της επιστήμης των υπολογιστών.
Σε αυτό το κεφάλαιο θα αναφερθούμε σε βασικά θέματα του προγραμματισμού των υπολογιστών και πώς αυτά αντιμετωπίζονται. Θα διαπιστώσουμε ότι, ουσιαστικά, ο προγραμματισμός είναι μια διαδικασία επίλυσης προβλημάτων με τη βοήθεια του υπολογιστή.

7.1 Η προγραμματιζομένη μηχανή

Στο πρώτο κεφάλαιο γνωρίσαμε μερικές από τις εφαρμογές της Πληροφορικής στο σύγχρονο κόσμο. Εάν εξετάσουμε τους υπολογιστές που χρησιμοποιούνται σε αυτές ως προς το υλικό τους μέρος, θα διαπιστώσουμε πως, παρ' όλο που η εφαρμογή είναι διαφορετική, σε πολλές περιπτώσεις το υλικό των χρησιμοποιούμενων υπολογιστών δεν διαφοροποιείται.
Ο ίδιος υπολογιστής μπορεί να χρησιμοποιηθεί από ένα πολυκατάστημα αλλά και από ένα λογιστικό γραφείο ή από ένα νοσοκομείο, με μόνη πιθανή διαφορά στις περιφερειακές συσκευές. Αυτό που επιτρέπει στο υλικό του υπολογιστή να προσαρμόζεται σε τόσο διαφορετικές απαιτήσεις είναι τα προγράμματά του, το λογισμικό.
Ήδη επισημάναμε ότι μια από τις βασικότερες διαφορές ανάμεσα στον υπολογιστή και στις περισσότερες ηλεκτρονικές συσκευές είναι η δυνατότητα προγραμματισμού του. Πρακτικά αυτό σημαίνει ότι η λειτουργία του υπολογιστή δεν καθορίζεται μόνο από τα ηλεκτρονικά του εξαρτήματα και τη μεταξύ τους συνδεσμολογία, αλλά κυρίως από το πρόγραμμα που κατά περίπτωση αυτός εκτελεί. Το γεγονός ότι ο υπολογιστής, ως ηλεκτρονική συσκευή, δεν είναι κατασκευασμένος να κάνει κάτι συγκεκριμένο, δεν είναι αδυναμία του αλλά το πλεονέκτημά του. Ανάλογα με το πρόγραμμα που εκτελεί μετασχηματίζεται στα μάτια του χρήστη σε μια διαφορετική μηχανή. Το λογισμικό εφαρμογών, το οποίο είδαμε σε προηγούμενο κεφάλαιο, είναι εκείνο που καθιστά τον υπολογιστή στα χέρια του αρχιτέκτονα ένα εξελιγμένο σχεδιαστικό εργαλείο, στα χέρια του συγγραφέα μια ευέλικτη γραφομηχανή ή στα χέρια του λογιστή ένα λογιστικό εργαλείο, κ.ο.κ.

7.2 Οι χρήστες

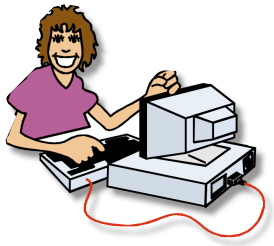
Τους χρήστες των υπολογιστικών συστημάτων μπορούμε να τους κατατάξουμε σε δύο μεγάλες κατηγορίες:

- σε αυτούς που χρησιμοποιούν τον υπολογιστή στη δουλειά τους, όπως είναι οι αρχιτέκτονες, οι συγγραφείς, οι λογιστές, κλπ. και
- σε αυτούς που η δουλειά τους σχετίζεται με τον ίδιο τον υπολογιστή.

Οι χρήστες της πρώτης κατηγορίας ονομάζονται «τελικοί χρήστες», ενώ στη δεύτερη υπάγονται οι επαγγελματίες της Πληροφορικής. Μια μεγάλη μερίδα των τελευταίων κατασκευάζει τα προγράμματα -το λογισμικό- που χρησιμοποιούνται από όλους τους άλλους.

Αυτή η ιδιότητα του κατασκευαστή και του χρήστη δεν είναι απόλυτα μονοσήμαντη. Ο κατασκευαστής λογισμικού, για να κάνει τη δουλειά του, χρησιμοποιεί με τη σειρά του τα προγράμματα άλλων κατασκευαστών, οπότε γίνεται και ο ίδιος με τη σειρά του χρήστης. Ωστόσο ο όρος **τελικός χρήστης** (end user) συνηθίζεται να δηλώνει το χρήστη που δεν είναι ειδικός στην Πληροφορική, αλλά χρησιμοποιεί τις εφαρμογές της Πληροφορικής στην εργασία του.

Ενδιαφέρον παρουσιάζει όχι μόνο η οπτική γωνία από την οποία βλέπει τον υπολογιστή ο προγραμματιστής αλλά και τα εργαλεία λογισμικού που αυτός χρησιμοποιεί στη δουλειά του.



7.3 Πρόγραμμα - Γλώσσες προγραμματισμού

7.3.1 Το Πρόγραμμα

Ήδη προαναφέρθηκε ότι ο υπολογιστής είναι μια προγραμματιζόμενη μηχανή. Δηλαδή, για να εκτελέσει ακόμη και την πιο απλή εργασία, θα πρέπει να του έχουν δοθεί λεπτομερείς οδηγίες για να την επιτελέσει. Αυτές οι οδηγίες αποτελούν το πρόγραμμα και εκτελούνται από την κεντρική μονάδα επεξεργασίας (ΚΜΕ) του υπολογιστή.

Όπως είδαμε στο κεφάλαιο του υλικού των υπολογιστών, οι σύγχρονοι υπολογιστές είναι βασισμένοι στις αρχές που διατύπωσε ο Von Neumann και η ομάδα του κατά τη δεκαετία του 1940. Η μηχανή του Von Neumann χαρακτηρίζεται από ένα μεγάλο αριθμό κελιών μνήμης και μια μονάδα επεξεργασίας που περιέχει ένα μικρό σχετικά αριθμό κελιών, τους καταχωρητές. Η ΚΜΕ μπορεί να «φορτώσει» στοιχεία από τη μνήμη στους καταχωρητές, να εκτελέσει κάποιες αριθμητικές (πρόσθεση, αφαίρεση, κλπ.) και λογικές (AND, OR, NOT, κλπ.) πράξεις με το περιεχόμενο των καταχωρητών και να αποθηκεύσει τις τιμές από τους καταχωρητές πίσω στη μνήμη.

Για μια μηχανή Von Neumann το **πρόγραμμα** αποτελείται από μια σειρά οδηγιών, που ονομάζονται **εντολές**, για την εκτέλεση τέτοιου είδους πράξεων, καθώς επίσης και από ένα

Πρώτος προγραμματιστής θεωρείται η Augusta Ada Byron, κόμμισα του Lovelace (1815 - 1852). Ήταν κόρη του Λόρδου Βύρωνα με αξιόλογη μόρφωση και με ταλέντο στα Μαθηματικά. Όταν ο Charles Babbage έφερε τη διαφορική μηχανή του στο σπίτι της μητέρας της, η Augusta γοητεύτηκε από τις δυνατότητές της, με αποτέλεσμα να συνεργαστεί για χρόνια με τον Babbage στον προγραμματισμό της. Ωστόσο ο Babbage δεν κατάφερε να κάνει τη διαφορική μηχανή του να δουλέψει. Μαζί ανέπτυξαν ένα σύστημα για να κερδίζουν σε ιπποδρομίες. Αλλά ούτε αυτό πέτυχε, με αποτέλεσμα να αναγκαστεί να πουλήσει οικογενειακά κοσμήματα για να καλύψει τα χρέη από τον ιππόδρομο. Η Augusta πέθανε από καρκίνο στα 37 της χρόνια, στην ίδια ηλικία με το διάσημο πατέρα της. Προς τιμήν της έχει ονομαστεί Ada μια γλώσσα προγραμματισμού που υιοθετήθηκε στις αρχές του 1980 από το Υπουργείο Άμυνας των Η.Π.Α.

σύνολο πρόσθετων οδηγιών ελέγχου, που επηρεάζουν τη σειρά με την οποία εκτελούνται οι εντολές.
Είναι γεγονός ότι το ρεπερτόριο αυτών των βασικών λειτουργιών της ΚΜΕ είναι εξαιρετικά περιορισμένο, όμως είναι αρκετό, ώστε με τον κατάλληλο συνδυασμό τους να μπορούμε να δώσουμε στον υπολογιστή τις απαραίτητες οδηγίες για να κάνει όλα όσα βλέπουμε ότι γίνονται από υπολογιστές γύρω μας.

7.3.2 Γλώσσα μηχανής

Επειδή ο κάθε τύπος ΚΜΕ έχει διαφορετικό ρεπερτόριο εντολών, τα προγράμματα που εκτελεί πρέπει να είναι διατυπωμένα στη δική της γλώσσα μηχανής. Αυτά τα προγράμματα αποτελούνται από ακολουθίες 0 και 1.

Οι εντολές που αναγνωρίζει μια τυπική ΚΜΕ ανήκουν σε μια από τις πιο κάτω κατηγορίες:

- εντολές μεταφοράς δεδομένων μεταξύ της κεντρικής μνήμης και των καταχωρητών της ΚΜΕ
- εντολές μεταφοράς δεδομένων μεταξύ των καταχωρητών
- εντολές αριθμητικών πράξεων
- εντολές λογικών πράξεων
- εντολές ελέγχου της ροής εκτέλεσης των εντολών
- διάφορες βοηθητικές εντολές.

Κάθε εντολή στη γλώσσα μηχανής αποτελείται από δύο τμήματα, τον **κωδικό λειτουργίας** (operation code, OP code) και τον **τελεστέο** (operand).

Κωδικός λειτουργίας	Τελεστέος

Ο κωδικός λειτουργίας προσδιορίζει τη λειτουργία της εντολής, για παράδειγμα «πρόσθεσε στο συσσωρευτή» ή «αποθήκευσε το περιεχόμενο του συσσωρευτή στην τάδε θέση μνήμης», και αντιστοιχεί σε μια από τις οδηγίες του συνόλου οδηγιών της ΚΜΕ. Ο τελεστέος αφορά τα δεδομένα στα οποία δρα η εντολή ή τη διεύθυνση στην οποία βρίσκονται αποθηκευμένα. Το μέγεθος (ο αριθμός των bit) κάθε εντολής μπορεί να είναι σταθερός ή μεταβλητός.

Z80: Επεξεργαστής της εταιρείας Zilog στο τέλος της δεκαετίας του '70

Για παράδειγμα, στον επεξεργαστή Z80, η εντολή 1110011010110011, κατευθύνει την ΚΜΕ να προσθέσει στο περιεχόμενο του καταχωρητή με το όνομα accumulator τον αντίστοιχο δυαδικό του $179_{(10)}$ δεκαδικό αριθμό 179. Τα πρώτα 8 ψηφία είναι ο κωδικός λειτουργίας, ενώ τα 8 επόμενα ο τελεστέος της εντολής, ο αριθμός $179_{(10)}$.
Ένα πρόγραμμα γραμμένο σε γλώσσα μηχανής θα έμοιαζε κάπως έτσι:

```
1110011010110011
1011011010111011
1010011110110110100100010
0101011010010011
1110011010110110
1100011010110111110110100
1101011000110101
10101110
1100011010110010
```

(Κωδικός Λειτουργίας/Τελεστέος)

Η κάθε εντολή θα μπορούσε, αντί να παρασταθεί στο δυαδικό σύστημα, να παρασταθεί στο οκταδικό ή στο δεκαεξαδικό, ώστε να είναι μικρότερος ο αριθμός των ψηφίων της.
Τα πρώτα προγράμματα γράφονταν σε γλώσσα μηχανής. Ωστόσο η γραφή προγραμμάτων άμεσα στη γλώσσα μηχανής του υπολογιστή είναι μια πολύ δύσκολη και αντιπαραγωγική εργασία. Δύο από τις βασικότερες αιτίες είναι η δυσκολία απομνημόνευσης των κωδικών των εντολών και η δυσκολία να εντοπιστεί κάποιο πιθανό λάθος ανάμεσα σε όλα αυτά τα 0 και 1.
Γι΄ αυτό, από πολύ νωρίς, αναζητήθηκαν τρόποι να γράφονται τα προγράμματα σε γλώσσα πιο προσιτή στον άνθρωπο. Η επιδίωξη αυτή, οδήγησε στην ανάπτυξη των διαφόρων γλωσσών προγραμματισμού, καθώς επίσης και στην ανεύρεση τρόπων για την εκτέλεση από τον υπολογιστή των προγραμμάτων που γράφονται σε αυτές τις γλώσσες.

7.3.3 Συμβολικές γλώσσες

Μια πρώτη προσπάθεια για τη διευκόλυνση της γραφής προγραμμάτων ήταν η γραφή τους σε **συμβολική γλώσσα** (assembly). Στις συμβολικές γλώσσες, σε κάθε εντολή της γλώσσας μηχανής αντιστοιχίζεται μια μνημονική λέξη η οποία θυμίζει το σκοπό της εντολής. Επί πλέον δίνεται η δυνατότητα να χρησιμοποιούνται, στη θέση αριθμών, ονόματα που αντιπροσωπεύουν σταθερές ή διευθύνσεις μνήμης. Έτσι μπορούμε να γράφουμε HLIKIA αντί για 18.
Η εντολή του προηγουμένου παραδείγματος της γλώσσας μηχανής του Z80 με τη μορφή 11100110 10110011, θα μπορούσε σε συμβολική γλώσσα να έχει τη μορφή:

ADD B3h

B3 είναι η δεκαεξαδική μορφή του 179. Το h δηλώνει ότι πρόκειται για δεκαεξαδικό αριθμό και όχι για όνομα B3

Στη συνέχεια βλέπουμε τμήμα ενός προγράμματος σε συμβολική γλώσσα, για επεξεργαστή της σειράς 80X86 της Intel.

```
mov   ax,DGROUP
mov   ds,ax
mov   ds:[STKHQQ],0
mov   es:[cdataseg],ax
mov   ax,sp
add   ax,bpregz+pgdata+14h
mov   [_atopsp],ax
mov   bx,seg STACK
sub   bx,DGROUP
mov   cl,4
shl   bx,cl
add   ax,bx
mov   [_abrktb].sz,ax
dec   ax
mov   [_asizds],ax
```

7.3.4 Γλώσσες υψηλού επιπέδου

Οι συμβολικές γλώσσες διευκόλυναν τη γραφή προγραμμάτων. Όμως οι εντολές τους εξακολουθούν να είναι σε άμεση (μία προς μία) αντιστοιχία με αυτές της γλώσσας μηχανής, με αποτέλεσμα ο προγραμματισμός να εξακολουθεί να είναι μια εξαιρετικά επίπονη διαδικασία. Επιπλέον τα προγράμματα που γράφονται σε συμβολική γλώσσα ή γλώσσα μηχανής μπορούν να εκτελεστούν μόνο από τον τύπο υπολογιστή για τον οποίο είναι γραμμένα. **Αυτό σημαίνει πως για να «τρέξει» το ίδιο πρόγραμμα σε έναν άλλο τύπο υπολογιστή θα πρέπει να γραφεί από την αρχή.**

Μεταφερσιμότητα προγράμματος

Για να αντιμετωπιστούν τέτοιου είδους προβλήματα, άρχισαν να δημιουργούνται και να εξελίσσονται νέες γλώσσες προγραμματισμού πιο απλές και ανεξάρτητες από το συγκεκριμένο τύπο υπολογιστή, που ονομάστηκαν γλώσσες **υψηλού επιπέδου** (high level languages). Αυτές οι γλώσσες, καθεμία σε διαφορετικό βαθμό, κρύβουν από τον προγραμματιστή τη γλώσσα μηχανής και προσφέρουν ένα πιο φιλικό σύνολο εντολών με τις οποίες συντάσσεται το κάθε πρόγραμμα. Προγράμματα αυτού του είδους με μικρές πιθανόν αλλαγές στον πηγαίο κώδικα μπορούν να εκτελεστούν στη συνέχεια και σε άλλους τύπους υπολογιστών, εκτός από αυτόν για τον οποίο αρχικά κατασκευάστηκαν. Αυτό το χαρακτηριστικό ενός προγράμματος ονομάζεται **μεταφερσιμότητα.**

7.3.5 Ένα συγκριτικό παράδειγμα

Έστω ότι θέλουμε να προσθέσουμε το περιεχόμενο δύο θέσεων μνήμης και το αποτέλεσμα να το καταχωρίσουμε σε μια τρίτη. Για λόγους σύγκρισης, ας δούμε πώς θα μπορούσε αυτό να γραφεί σε γλώσσα υψηλού επιπέδου, σε συμβολική γλώσσα και σε γλώσσα μηχανής:

Γλώσσα υψηλού επιπέδου

Εντολή	Περιγραφή
`A:= B + C`	Πρόσθεσε το περιεχόμενο των μεταβλητών B και C και το αποτέλεσμα καταχώρισέ το στη μεταβλητή A

Συμβολική γλώσσα

Εντολή	Περιγραφή
`LDA B`	Μετάφερε στο συσσωρευτή το περιεχόμενο της θέσης μνήμης με όνομα B
`ADD C`	Πρόσθεσε στο περιεχόμενο του συσσωρευτή το περιεχόμενο της θέσης μνήμης με όνομα C
`STA A`	Μετάφερε και αποθήκευσε το περιεχόμενο του συσσωρευτή στη θέση μνήμης με όνομα A

Γλώσσα μηχανής

Εντολή	Περιγραφή
0000001001011010	Μετάφερε στο συσσωρευτή το περιεχόμενο της θέσης μνήμης με διεύθυνση 01011010
0000101001011110	Πρόσθεσε στο περιεχόμενο του συσσωρευτή το περιεχόμενο της θέσης μνήμης με διεύθυνση 01011110
0000011011011110	Μετάφερε και αποθήκευσε το περιεχόμενο του συσσωρευτή στη θέση μνήμης με διεύθυνση 11011110

Όπως φαίνεται, για να πετύχουμε το ίδιο αποτέλεσμα, απαιτήθηκε μια εντολή σε γλώσσα υψηλού επιπέδου και αντίστοιχα τρεις σε συμβολική γλώσσα και σε γλώσσα μηχανής. Επιπλέον, ενώ στη γλώσσα υψηλού επιπέδου αναφερόμαστε σε μεταβλητές A, B, C, χωρίς να γίνεται αναφορά ούτε σε συγκεκριμένες θέσεις μνήμης ούτε στη διαδικασία της πρόσθεσης, στις άλλες δύο περιπτώσεις συμβαίνει ακριβώς το αντίθετο. Έτσι είναι απαραίτητο να αναφερθούμε και στη διαδικασία της πρόσθεσης και σε συγκεκριμένες θέσεις μνήμης είτε με όνομα είτε με διεύθυνση.

7.3.6 Ιστορία γλωσσών υψηλού επιπέδου

Στη συνέχεια θα αναφερθούμε με συντομία σε μερικές από τις πιο γνωστές γλώσσες υψηλού επιπέδου. Πρέπει να σημειωθεί ότι ο αριθμός των γλωσσών υψηλού επιπέδου είναι εξαιρετικά μεγάλος, αν και μόνο ένας μικρός αριθμός από αυτές καταφέρνει να γίνει αποδεκτός από τους κατασκευαστές προγραμμάτων. Πολλές παραμένουν απλά ερευνητικά εργαλεία.

FORTRAN

Η **FORTRAN** σχεδιάστηκε αρχικά γύρω στο 1954 από μια ομάδα της IBM, με επικεφαλής τον John Backus. Στόχος τους ήταν η κατασκευή μιας γλώσσας υπολογισμού αλγεβρικών παραστάσεων. Η FORTRAN (**FOR**mula **TRAN**slation -Μετάφραση Μαθηματικών Τύπων) ολοκληρώθηκε το 1957 και έγινε πολύ γρήγορα δημοφιλής σε μεγάλο βαθμό χάρη στην υποστήριξη της IBM, η οποία την προσέφερε δωρεάν.

```
      SUBROUTINE RAND(U,N)
      INTEGER N,X,Y,Z
      REAL U(N),V
      COMMON/RANDOM/X,Y,Z
      IF(N.LE.0) RETURN
      DO 1 I=1,N
           X=171*MOD(X,177)-2*(X/177)
           IF(X.LT.0) X=X+30269
           Y=172*MOD(Y,176)-35*(Y/176)
           IF(Y.LT.0) Y=Y+30307
           Z=170*MOD(Z,178)-63*(Z/178)
           IF(Z.LT.0) Z=Z+30323
           V=X/30269.0 + Y/30308.0 + Z/30323.0
1          U(I)=V-INT(V)
      RETURN
      END
```

Τμήμα προγράμματος Fortran

ALGOL

Η **ALGOL** (1963) υπήρξε μια γλώσσα που δεν έγινε ιδιαίτερα γνωστή, κατάφερε όμως να επηρεάσει βαθύτατα το σχεδιασμό των γλωσσών προγραμματισμού και να δημιουργήσει ολόκληρη γενιά γλωσσών παρόμοιων με την ALGOL (ALGOL like). Κι αυτό γιατί υλοποίησε αξιόλογες ιδέες προγραμματισμού, όπως είναι ο δομημένος προγραμματισμός στον οποίο θα αναφερθούμε και σε άλλο σημείο αυτού του κεφαλαίου.

```
BEGIN
REAL X, Y;
   PROCEDURE P;
   BEGIN
   REAL Z;
     . . .
     X := X * Z;
   END P;
   PROCEDURE Q;
   BEGIN
   INTEGER Y, X;
     P;
   END;
   Q;
END
```

Τμήμα προγράμματος Algol

COBOL

Η **COBOL** (**CO**mmon **B**usiness **O**riented **L**anguage - Κοινή Γλώσσα Εμπορικού Προσανατολισμού), σχεδιάστηκε από το Υπουργείο Άμυνας των Η.Π.Α ως απάντηση στην ανάγκη για μια απλή γλώσσα προγραμματισμού, που θα μπορούσε να χρησιμοποιηθεί σε όλους τους στρατιωτικούς τομείς. Η σχεδίαση έγινε από ομάδα στην οποία συνεργάστηκαν επιστήμονες από το στρατό αλλά και τη βιομηχανία, συνεργασία που βοήθησε στην αποδοχή της από τους προγραμματιστές. Μια από τις ιδέες που εισήγαγε ήταν η απεξάρτηση της περιγραφής των δεδομένων από τη μηχανή. Στην COBOL αυτό γίνεται σε μια ενότητα του προγράμματος που ονομάζεται DATA DIVISION. Αυτή η αρχική ιδέα αργότερα οδήγησε στην επινόηση των συστημάτων διαχείρισης βάσεων δεδομένων.

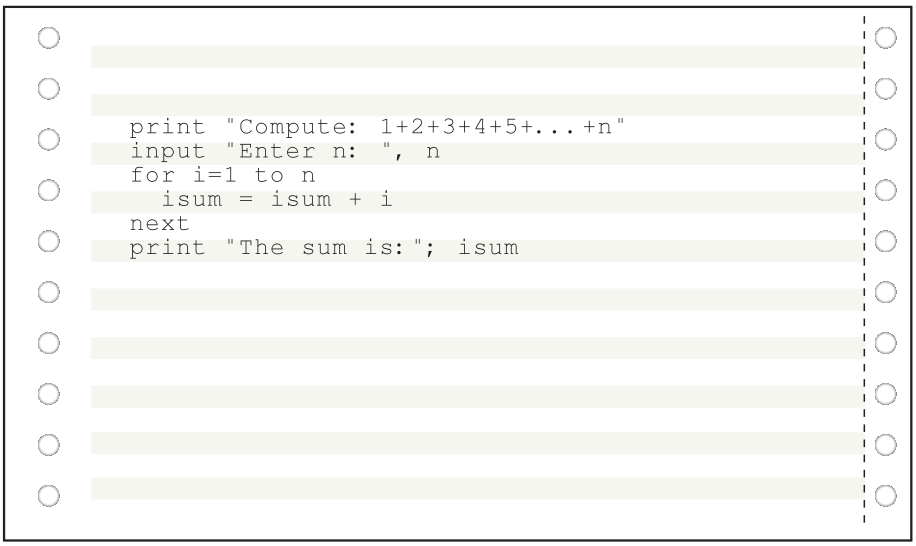
```
IDENTIFICATION DIVISION.
PROGRAM-ID.  AcceptAndDisplay.
AUTHOR.  The Author

DATA DIVISION.
WORKING-STORAGE SECTION.
01 StudentDetails.
   02  StudentId          PIC 9(7).
   02  StudentName.
       03 Surname         PIC X(8).
       03 Initials        PIC XX.
   02  CourseCode         PIC X(4).
   02  Gender             PIC X.
* YYMMDD
01 CurrentDate.
   02  CurYr              PIC 9(4).
   02  CurMth             PIC 99.
   02  CurD               PIC 99.
* YYDDD
01 DayOfYear.
   02  FILLER             PIC 9(4).
   02  YearDay            PIC 9(3).
* HHMMSSss   s = S/100
01 CurrentTime.
   02  CurHr              PIC 99.
   02  CurMin             PIC 99.
   02  FILLER             PIC 9(4).
PROCEDURE DIVISION.
Begin.
    DISPLAY "Enter student details.
    DISPLAY "ID,Surname,Initials,CourseCode,Gender"
    DISPLAY "SSSSSSSNNNNNNNNIICCCCG".
    ACCEPT  StudentDetails.
    ACCEPT  CurrentDate FROM DATE YYYYMMDD.
    ACCEPT  DayOfYear FROM DAY YYYYDDD.
    ACCEPT  CurrentTime FROM TIME.
    DISPLAY "Name is", Initials SPACE Surname.
    DISPLAY "Date is" CurD SPACE CurMth SPACE CurYr.
    DISPLAY "Today is day " YearDay " of the year".
    DISPLAY "The time is " CurHr ":" CurMin.
    STOP RUN.
```

Τμήμα προγράμματος Cobol

BASIC

Η **BASIC** (**B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode) αναπτύχθηκε από τους Kemeney και Kurtz στα μέσα της δεκαετίας του 1960, ως μια γλώσσα για τη διδασκαλία προγραμματισμού σε φοιτητές. Σήμερα, με πολλές επεκτάσεις σε σχέση με τον αρχικό σχεδιασμό, χρησιμοποιείται ως μια κανονική γλώσσα ανάπτυξης εφαρμογών.



```
print "Compute: 1+2+3+4+5+...+n"
input "Enter n: ", n
for i=1 to n
  isum = isum + i
next
print "The sum is:"; isum
```

Τμήμα προγράμματος Basic

SIMULA

Η **SIMULA** εμφανίστηκε το 1966 στη Νορβηγία από τους Dahl και Nygaard. Είναι γλώσσα βασισμένη στην ALGOL με βασικό τομέα εφαρμογής της την προσομοίωση.

```
Class Rectangle (Width, Height); Real Width, Height;
 Begin
    Real Area, Perimeter;
    Procedure Update;
    Begin
      Area := Width * Height;
      Perimeter := 2*(Width + Height)
    End of Update;
    Boolean Procedure IsSquare;
      IsSquare := Width=Height;
    Update;
    OutText("Rectangle created: ");
    OutFix(Width,2,6);
    OutFix(Height,2,6); OutImage
 End of Rectangle;
```

Τμήμα προγράμματος Simula

LISP

Ξεκίνησε στα τέλη της δεκαετίας του '50 από τον John McCarthy και τους συνεργάτες του στο MIT. Αν και ο McCarthy συμμετείχε στη σχεδίαση της ALGOL, η LISP είναι μια τελείως διαφορετική γλώσσα με κύριο πεδίο εφαρμογής την τεχνητή νοημοσύνη.

```
(defun flatten (s-expression)
    (if (atom s-expression)
        s-expression
        (flatten-aux s-expression)
    )
)

(defun flatten-aux (somelist)
    (if (listp (car somelist))
        (append (flatten (car somelist))
           (flatten (cdr somelist))
        )
        (cons (car somelist)
           (flatten (cdr somelist))
        )
    )
)
```

Τμήμα προγράμματος Lisp

LOGO

Σχεδιάστηκε από ομάδα ερευνητών της τεχνητής νοημοσύνης με επικεφαλής τον Seymour Papert. Έχει παιδαγωγικές προδιαγραφές για την επίλυση προβλημάτων και τη διερεύνηση εννοιών ακόμα και από μαθητές μικρής ηλικίας. Οι μηχανισμοί δημιουργίας γραφικών την καθιστούν σημαντικό εργαλείο στη διερεύνηση γεωμετρικών σχημάτων.

```
to disstack :num
setcursor list (-3 + 5 * :num) 4
type ifelse stackemptyp hidden :num ["| |] ["-]
if stackemptyp shown :num [setcursor list (-4 + 5 * :num) 5
                           type "|    | stop]
localmake "stack (thing shown :num)
localmake "col 5*:num-4
for [i [count :stack] 1] ´
    [setcursor list :col :i+4
     carddis pop "stack]
end

to distop :suit
if emptyp top :suit [stop]
if equalp :suit "H [distop1 4 stop]
if equalp :suit "S [distop1 11 stop]
if equalp :suit "D [distop1 18 stop]
distop1 25
end
```

Τμήμα προγράμματος Logo

PASCAL

Η **Pascal** είναι δημιούργημα του Niclaus Wirth. Ο Wirth αρχικά συμμετείχε σε μια επιτροπή η οποία είχε ως σκοπό να βελτιώσει την ALGOL, όμως αποχώρησε μαζί μ' εκείνους που διαφώνησαν για το τελικό αποτέλεσμα και κατασκεύασε μια δική του γλώσσα (1971). Οι στόχοι σχεδιασμού της Pascal επικεντρώθηκαν γύρω από την παροχή ορισμένων δυνατοτήτων που θα οδηγούσαν σε αποδοτικό κώδικα κατά τη μεταγλώττιση, ιδέα που τέθηκε σε εφαρμογή με αξιοθαύμαστο τρόπο.

```
PROGRAM arrayc (output);
TYPE  days = (sun,mon,tues,weds,thurs,fri,sat);
      dname = string(8);

VAR   d: days;

FUNCTION DayName (fd: days): dname;
  TYPE  abbrevs = ARRAY [days] OF
                  PACKED ARRAY [1..5] OF char;
  CONST DayNames = abbrevs
        [ sun:'Sun';mon:'Mon';tues:'Tues';
          weds:'Weds';thurs:'Thurs';fri:'Fri';
          sat:'Satur' ];
  BEGIN
    DayName := trim(DayNames[fd]) + 'day';
  END {DayName};

BEGIN {program}
  FOR d := fri DOWNTO mon DO writeln(DayName(d));
END.
```

Τμήμα προγράμματος Pascal

C

Η **C** αναπτύχθηκε το 1972 από τους Brian Kernighan και Dennis Ritchie στα εργαστήρια της εταιρείας AT&T. Σχεδιάστηκε ως μια γλώσσα κατάλληλη για τη συγγραφή λειτουργικών συστημάτων. Το μεγαλύτερο τμήμα του λειτουργικού συστήματος UNIX γράφτηκε σε C, πράγμα που την έκανε ταυτόσημη του UNIX. Η C συνδυάζει χαρακτηριστικά γλωσσών υψηλού επιπέδου, αλλά ταυτόχρονα επιτρέπει την πρόσβαση στις χαμηλού επιπέδου λειτουργίες του υπολογιστή. Η C έχει καθιερωθεί ως η γλώσσα προγραμματισμού για την ανάπτυξη λειτουργικών συστημάτων και λογισμικού συστήματος. Χρησιμοποιείται επίσης ευρύτατα στην κατασκευή λογισμικού εφαρμογών, αλλά και κάθε είδους λογισμικού το οποίο απαιτεί τη μέγιστη αξιοποίηση των δυνατοτήτων του υπολογιστή.

```
void
InitBlock(float *a, float *b, float *c,
          int blk, int row, int col)
{
    int len, ind;
    int i,j;

    srand(pvm_mytid());
    len = blk*blk;
    for (ind = 0; ind < len; ind++) {
       a[ind] = (float)(rand()%1000)/100.0;
       c[ind] = 0.0;
       }
    for (i = 0; i < blk; i++) {
      for (j = 0; j < blk; j++) {
            if (row == col)
                 b[j*blk+i] = (i==j)? 1.0 : 0.0;
            else
                 b[j*blk+i] = 0.0;
             }
        }
}
```

Τμήμα προγράμματος C

C++

Η C++ είναι η εξέλιξη της C προς τη κατεύθυνση του αντικειμενοστρεφούς προγραμματισμού. Σχεδιάστηκε από τον Bjarne Stroustrup στα Bell Labs της AT&T. Είναι η γλώσσα που κυριαρχεί στην κατασκευή προγραμμάτων σε «παραθυρικά» περιβάλλοντα, όπως είναι αυτά των PCs και Macintosh.

```
        List&  List::
        InsertLast(const Data &d)
        { SHOW("InsertLast(const Data&)");
          Node * const t = new Node(d);

          if (last == NULL) {
            first = t;
          }else {
            last->next = t;
          }
          last = t;
          ++num;
          return *this;
        }
```

Τμήμα προγράμματος C++

SMALLTALK

Πρόκειται για λειτουργικό σύστημα και ταυτόχρονα για γλώσσα αντικειμενοστρεφούς προγραμματισμού, που αναπτύχθηκε στα εργαστήρια της εταιρείας Xerox στο Palo Alto. Θεωρείται η πρώτη, και από πολλούς η μόνη, πραγματική γλώσσα αντικειμενοστρεφούς προγραμματισμού, που θα μπορούσε να χρησιμεύσει ως πρότυπο και μέτρο σύγκρισης για τις υπόλοιπες.

```
insetBy: delta
    | newrect |
    newrect _ delta asRectangle.
    ^Rectangle origin: (origin+(newrect origin))
                corner: (corner-(newrect corner))
!

insetOriginBy: originDeltaPoint corner: cornerDeltaPoint
    ^Rectangle origin: origin + originDeltaPoint
                corner: corner + cornerDeltaPoint
!

merge: aRectangle
    | orig corn |
    orig _ Point x: ((origin x) min:(aRectangle origin x))
                 y: ((origin y) min:(aRectangle origin y)).
    corn _ Point x: ((corner x) max:(aRectangle corner x))
                 y: ((corner y) max:(aRectangle corner y)).
    ^Rectangle origin: orig corner: corn
!
```

Τμήμα προγράμματος SmallTalk

ADA

Το Υπουργείο Άμυνας των Η.Π.Α στα μέσα της δεκαετίας του 70 αποφάσισε να υποστηρίξει τη σχεδίαση μιας νέας γλώσσας κατάλληλης για τον προγραμματισμό Ενσωματωμένων Υπολογιστικών Συστημάτων (**E**mbeded **C**omputer **S**ystems-**ECS**). Οι εφαρμογές που προορίζονται για ΕΥΣ χαρακτηρίζονται από μεγάλο μέγεθος και υψηλού βαθμού πολυπλοκότητα, ενώ χρειάζονται διαρκή εξέλιξη. Συνήθως τέτοιες εφαρμογές υλοποιούνται σε συμβολική γλώσσα, οπότε έπρεπε να βρεθεί κάποιος αποδοτικότερος τρόπος. Το αποτέλεσμα της προσπάθειας ήταν η δημιουργία στις αρχές του 1980 μιας γλώσσας προγραμματισμού που ονομάστηκε Ada, προς τιμήν της Augusta Ada Byron.

```
for Hour in 1..Num_Hours loop
   Integer_Text_IO.Get (Item => Temp);
   Integer_Text_IO.Put (Item => Temp, Width => 4);

   if Temp < Low then
      Low := Temp;
   end if;

   if Temp > High then
      High := Temp;
   end if;
end loop;
```

Τμήμα προγράμματος ADA

JAVA

Η **Java** είναι δημιούργημα της εταιρείας Sun Microsystems. Σχεδιάστηκε στις αρχές της δεκαετίας του 1990, αρχικά ως μια γλώσσα για προγράμματα που θα εκτελούνταν από μικροσυσκευές με την ονομασία OAK. Η αρχική προσπάθεια απέτυχε και το 1995 η SUN τής άλλαξε το όνομα σε Java και την προσάρμοσε στις ανάγκες του παγκόσμιου ιστού και του Διαδικτύου. Από τότε γνωρίζει τεράστια επιτυχία και τείνει να γίνει η γλώσσα προγραμματισμού του Διαδικτύου.

```
public class Fibonacci {
   public static void main(String[] args) {
      int current, prev = 1, prevprev = 0;
      for(int i = 0; i < 20; i++) {
         current = prev + prevprev;
         System.out.print(current + " ");
         prevprev = prev;
         prev = current;      }
      System.out.println();
   }
}
```

Τμήμα προγράμματος Java

PROLOG

Η **PROLOG** (**PRO**gramming in **LOG**ic) αναπτύχθηκε το 1973 στη Γαλλία. Βασίζεται στη μαθηματική λογική και χρησιμοποιείται όπως και η LISP κυρίως σε εφαρμογές τεχνητής νοημοσύνης.

```
neighbor(X, Y) :- hall(X, Y).
neighbor(X, Y) :- hall(Y, X).

go(Start, Finish) :- VisitedRooms = [Start],
        route(Start, Finish, VisitedRooms).

route(exit, exit, VisitedRooms) :-
        can_leave(VisitedRooms),
        printlist(VisitedRooms), nl, fail.

route(Room, Finish, VisitedRooms) :-
        neighbor(Room, NextRoom),
        avoid(DangerousRooms),
        not( member(NextRoom, DangerousRooms) ),
        not( member(NextRoom, VisitedRooms) ),
        route(NextRoom, Finish, [NextRoom|VisitedRooms]).

member(X, [X|_]).
member(X, [_|Y]) :- member(X, Y).
```

Τμήμα προγράμματος Prolog

SQL

Η **SQL** (**S**tructured **Q**uery **L**anguage) αρχικά αναπτύχθηκε το 1974 από την IBM. Έχει καθιερωθεί ως η γλώσσα επικοινωνίας με τις σχεσιακές βάσεις δεδομένων. Ως εμπορικό προϊόν πρωτοεμφανίστηκε από την εταιρεία ORACLE το 1979.

```
CREATE TABLE STATS
(ID INTEGER REFERENCES STATION(ID), MONTH INTEGER CHECK
(MONTH BETWEEN 1 AND 12),
TEMP_F REAL CHECK (TEMP_F BETWEEN -80 AND 150),
RAIN_I REAL CHECK (RAIN_I BETWEEN 0 AND 100),
PRIMARY KEY (ID, MONTH));

SELECT MONTH, ID, RAIN_I, TEMP_F
FROM STATS ORDER BY MONTH,
RAIN_I DESC;
```

Εντολές SQL

Γενιές γλωσσών προγραμματισμού 3GL, 4GL

Μέχρι τώρα παρουσιάσαμε τις διάφορες γλώσσες προγραμματισμού με κάποια χρονολογική σειρά, συχνά όμως γίνεται αναφορά σε **γενιές** γλωσσών προγραμματισμού. Γλώσσα πρώτης γενιάς ονομάζεται η γλώσσα μηχανής. Οι συμβολικές γλώσσες θεωρείται ότι αποτελούν τη δεύτερη γενιά γλωσσών, ενώ οι γλώσσες υψηλού επιπέδου την τρίτη (**3GL**- **3**rd **G**eneration **L**anguage). Τέλος, γλώσσες τέταρτης γενιάς (4GL) ονομάζονται οι γλώσσες που είναι πιο κοντά στη φυσική γλώσσα απ΄ ό,τι οι συνήθεις γλώσσες υψηλού επιπέδου. Χρησιμοποιούνται συνήθως για πρόσβαση σε βάσεις δεδομένων. Τυπικό παράδειγμα είναι η SQL.

Με τη χρήση των γλωσσών υψηλού επιπέδου η ανάπτυξη του λογισμικού υπήρξε ραγδαία. Μεταξύ των λόγων που συνετέλεσαν σε αυτό μπορούν να αναφερθούν οι εξής:

- Η εκμάθηση μιας γλώσσας προγραμματισμού έγινε ευκολότερη.
- Η κατασκευή του λογισμικού έγινε απλούστερη.
- Για τις πιο διαδεδομένες γλώσσες υπήρξε τυποποίηση της μορφής της γλώσσας. Έτσι προγράμματα που έχουν γραφεί σε γλώσσα υψηλού πιπέδου μπορούν να μεταφερθούν και να εκτελεστούν σε υπολογιστές με διαφορετική ΚΜΕ και άρα διαφορετική γλώσσα μηχανής. Η επίσημη τυποποίηση των διαφόρων γλωσσών γίνεται από το **ANSI** (**A**merican **N**ational **S**tandard **I**nstitute).

ANSI

Για τους παραπάνω λόγους τόσο το κόστος ανάπτυξης όσο και το κόστος χρήσης του λογισμικού μειώθηκαν σημαντικά.

7.3.7 Εξάρτηση των γλωσσών από το σκοπό

Οι διάφορες γλώσσες ανάλογα με το σκοπό που εξυπηρετούν, ή ακριβέστερα σύμφωνα με το σκοπό για τον οποίο αρχικά σχεδιάστηκαν, χωρίζονται σε δύο γενικές κατηγορίες, στις ειδικού και στις γενικού σκοπού γλώσσες.

Γλώσσες ειδικού σκοπού

Στην κατηγορία των γλωσσών ειδικού σκοπού ανήκουν εκείνες οι οποίες είναι προσανατολισμένες σε μια συγκεκριμένη κατηγορία εφαρμογών.

Για παράδειγμα:

- Η COBOL είναι προσανατολισμένη στην επίλυση εμπορικών προβλημάτων. Αυτή η κατηγορία προβλημάτων χαρακτηρίζεται από την αποθήκευση και διαχείριση μεγάλου όγκου δεδομένων με σχετικά περιορισμένες απαιτήσεις αριθμητικών πράξεων. Ως αποτέλεσμα η COBOL δίνει μεγάλη βαρύτητα στα θέματα περιγραφής και αποθήκευσης δεδομένων, ενώ μειονεκτεί στο θέμα της αποδοτικής εκτέλεσης μεγάλου αριθμού αριθμητικών πράξεων.
- Η FORTRAN είναι προσανατολισμένη στις επιστημονικές εφαρμογές, όπου κυρίαρχο ρόλο παίζει η ταχύτητα εκτέλεσης μεγάλου πλήθους αριθμητικών πράξεων πάνω σε μικρό σχετικά όγκο δεδομένων.
- Η LISP είναι προσανατολισμένη στη διαχείριση συμβόλων, γεγονός πουτην καθιστά κατάλληλη για εφαρμογές τεχνητής νοημοσύνης.

Γλώσσες γενικού σκοπού

Η δεύτερη κατηγορία, αυτή στην οποία ανήκουν οι γλώσσες γενικού σκοπού, περιλαμβάνει γλώσσες που μπορούν να χρησιμοποιηθούν σε διαφορετικές περιοχές εφαρμογών με εξίσου καλά αποτελέσματα.

Για παράδειγμα, τέτοιες γλώσσες είναι η C και η Java.

Το γεγονός ότι κατατάσσουμε μια γλώσσα στην κατηγορία γλωσσών ειδικού σκοπού δε σημαίνει ότι μπορεί να χρησιμοποιηθεί μόνο σε ένα τομέα προβλημάτων, αλλά ότι συνηθίζεται να χρησιμοποιείται και είναι προσανατολισμένη προς κάποιο τομέα.

7.3.8 Μεταφραστές

Όπως ήδη αναφέραμε, η ΚΜΕ αναγνωρίζει και εκτελεί εντολές σε γλώσσα μηχανής. Το ερώτημα λοιπόν είναι: πώς τα προγράμματα, που είναι γραμμένα σε οποιαδήποτε άλλη μορφή εκτός της γλώσσας μηχανής, εκτελούνται από την ΚΜΕ του υπολογιστή;
Θα πρέπει τα προγράμματα που είναι γραμμένα σε κάποια γλώσσα υψηλού επιπέδου να «μεταφραστούν» σε γλώσσα μηχανής. Το ρόλο αυτό τον αναλαμβάνουν ειδικά προγράμματα. Στην περίπτωση των συμβολικών γλωσσών, χρησιμοποιούνται οι **συμβολομεταφραστές**, ενώ στις γλώσσες υψηλού επιπέδου, οι **μεταγλωττιστές** και οι **διερμηνευτές**.

Συμβολομεταφραστές

Σε προηγούμενο παράδειγμα διαπιστώσαμε ότι οι εντολές των συμβολικών γλωσσών βρίσκονται σε αντιστοιχία μία προς μία με αυτές της γλώσσας μηχανής. Η μετατροπή των προγραμμάτων από συμβολική γλώσσα σε γλώσσα μηχανής γίνεται από ειδικά προγράμματα που ονομάζονται συμβολομεταφραστές (assemblers). Η βασική τους λειτουργία συνίσταται στο να αντικαθιστούν με τη χρήση ενός λεξικού τις συμβολικές εντολές με τις αντίστοιχες της γλώσσας μηχανής.
Οι σύγχρονοι συμβολομεταφραστές, χωρίς να απομακρύνονται από αυτήν τη βασική λειτουργία, προσφέρουν διάφορες ευκολίες, όπως:

- να δίνονται ονόματα σε θέσεις μνήμης ή σε αριθμητικές αξίες και ο συμβολομεταφραστής αναλαμβάνει να κάνει τις μετατροπές
- να επιτρέπουν την κατασκευή μακροεντολών.

Έχει παρατηρηθεί ότι σε ένα πρόγραμμα κάποιες ομάδες εντολών επαναλαμβάνονται σε διαφορετικά σημεία αυτούσιες ή με μικρές παραλλαγές. Ο μηχανισμός των μακροεντολών επιτρέπει να ορίσουμε ένα όνομα που να αντιπροσωπεύει την επαναλαμβανόμενη σειρά εντολών, την οποία καταγράφουμε αναλυτικά για μια φορά. Σε κάθε σημείο που χρειαζόμαστε την ίδια σειρά εντολών, αρκεί να γίνεται αναφορά σε αυτό το όνομα και ο συμβολομεταφραστής αναλαμβάνει την αντικατάσταση του ονόματος της μακροεντολής με τις εντολές που αυτή εμπεριέχει.

Μεταγλωττιστές - Διερμηνευτές

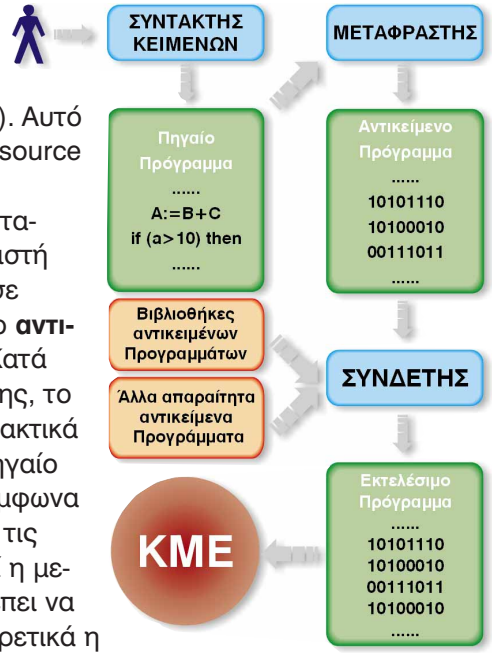
Υπάρχουν δύο τρόποι για την εκτέλεση από τον υπολογιστή προγραμμάτων που είναι γραμμένα σε γλώσσες υψηλού επιπέδου:

- με τη χρήση μεταγλωττιστή (compiler)
- με τη χρήση διερμηνευτή (interpreter).

Μεταγλωττιστής

Στο σχήμα φαίνεται η διαδικασία εκτέλεσης ενός προγράμματος με τη χρήση **μεταγλωττιστή**. Η διαδικασία περιλαμβάνει τα εξής βήματα:

- Γραφή του προγράμματος σε γλώσσα υψηλού επιπέδου -π.χ. PASCAL- με τη βοήθεια ενός συντάκτη κειμένων (editor). Αυτό είναι το **πηγαίο** πρόγραμμα (source code, source program).
- Το πηγαίο αυτό πρόγραμμα μεταγλωττίζεται, από το μεταγλωττιστή της συγκεκριμένης γλώσσας, σε γλώσσα μηχανής. Αυτό είναι το **αντικείμενο** (object) πρόγραμμα. Κατά τη διαδικασία της μεταγλώττισης, το πρόγραμμα ελέγχεται για συντακτικά λάθη, δηλαδή κατά πόσο το πηγαίο πρόγραμμα είναι γραμμένο σύμφωνα με τους συντακτικούς κανόνες τις γλώσσας. Για να ολοκληρωθεί η μεταγλώττιση το πρόγραμμα πρέπει να είναι συντακτικά σωστό, διαφορετικά η διαδικασία της μεταγλώττισης διακόπτεται.



- Το αντικείμενο πρόγραμμα, παρ' όλο που είναι σε γλώσσα μηχανής, δεν μπορεί να εκτελεστεί από την ΚΜΕ, γιατί δεν είναι αυτόνομο. Συνήθως του λείπει κάποιος κώδικας κοινής χρήσης, που βρίσκεται σε βιβλιοθήκες αντικειμένων προγραμμάτων. Όταν λέμε κώδικα κοινής χρήσης, συνήθως εννοούμε κώδικα που διαχειρίζεται τις λειτουργίες εισόδου εξόδου ή κώδικα που αφορά μαθηματικές συναρτήσεις (sin, cos, κλπ.). Τη σύνδεση του αντικειμένου προγράμματος με τα απαραίτητα προγράμματα από τις βιβλιοθήκες αναλαμβάνει ο **συνδέτης** (linker), ο οποίος παράγει και το αυτόνομο εκτελέσιμο πρόγραμμα. Αυτό το τελευταίο είναι που εκτελείται από την ΚΜΕ.

Τα βήματα δηλαδή είναι:

- Συγγραφή
- Μεταγλώττιση
- Σύνδεση
- Εκτέλεση.

Στο περιβάλλον του MS-DOS και των MS-Windows τα εκτελέσιμα προγράμματα διακρίνονται από την επέκταση **com** ή **exe** στο όνομα του αρχείου.

Τα βήματα αυτά μπορεί να γίνονται είτε σαν μια συνεχόμενη διαδικασία είτε το καθένα σε διαφορετικό χρόνο. Στο κάθε βήμα μπορεί να προκύψουν λάθη, οπότε διορθώνουμε το λάθος και η διαδικασία επαναλαμβάνεται.
Στην πιο πάνω διαδικασία ο συντάκτης κειμένων, ο μεταφραστής και ο συνδέτης, είναι προγράμματα τα οποία μας βοηθούν να παραγάγουμε το εκτελέσιμο πρόγραμμα. Από τη στιγμή που το εκτελέσιμο πρόγραμμα παραχθεί, μπορεί να εκτελεστεί αυτόνομα από την ΚΜΕ, χωρίς να είναι απαραίτητος ο συντάκτης κειμένων, ο μεταφραστής ή ο συνδέτης.
Τελείως διαφορετική σε αυτό το σημείο είναι η προσέγγιση της εκτέλεσης προγράμματος γραμμένου σε γλώσσα υψηλού επιπέδου με τη χρήση **διερμηνευτή**.

Διερμηνευτής

Το πρόγραμμα γράφεται και σε αυτήν την περίπτωση με τη βοήθεια κάποιου συντάκτη κειμένων, ο οποίος είναι συνήθως ενσωματωμένος στο πρόγραμμα

του διερμηνευτή. Στη συνέχεια ο διερμηνευτής αναλαμβάνει να εκτελέσει μία μία τις εντολές του προγράμματος σαν να απευθύνονται σε αυτόν και όχι στην ΚΜΕ. Ο διερμηνευτής, δηλαδή, δεν μεταγλωττίζει το πρόγραμμα σε γλώσσα μηχανής όπως κάνει ο μεταγλωττιστής, αλλά εκτελεί ο ίδιος το πρόγραμμα. Συνήθως ο διερμηνευτής μετασχηματίζει το πρόγραμμά μας σε μια ενδιάμεση μορφή και στη συνέχεια το εκτελεί.

Στην περίπτωση αυτή το πρόγραμμά μας δεν εκτελείται άμεσα από την ίδια την ΚΜΕ αλλά από το διερμηνευτή. Η ΚΜΕ δεν εκτελεί άμεσα το πρόγραμμά μας αλλά εκτελεί το πρόγραμμα του διερμηνευτή, ο οποίος με τη σειρά του εκτελεί το πρόγραμμά μας.

Σε αντίθεση με την προσέγγιση της μεταγλώττισης, το πρόγραμμα δεν είναι αυτόνομο, δεν μπορεί να εκτελεστεί ανεξάρτητα, εφόσον απαιτείται η παρουσία του αντίστοιχου διερμηνευτή.

Η εκτέλεση ενός προγράμματος με αυτόν τον τρόπο είναι γενικά πιο αργή σε σχέση με τη μεταγλώττιση, επειδή κατά τη στιγμή της εκτέλεσης παρεμβάλλεται μεταξύ προγράμματος και ΚΜΕ το επίπεδο του διερμηνευτή. Συνήθως οι διερμηνευτές είναι διαλογικά προγράμματα.

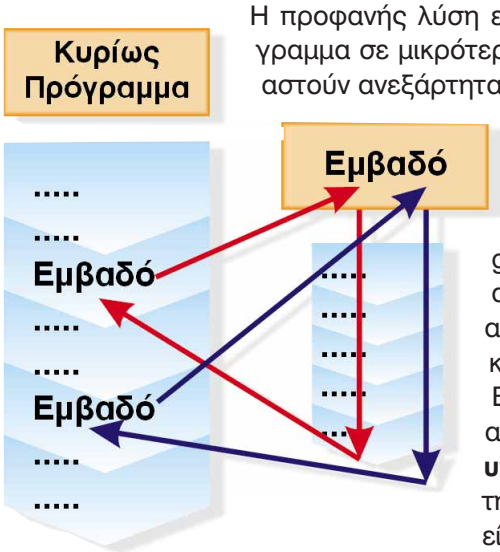
7.4 Αρχές κατασκευής λογισμικού

Ένα πρόγραμμα δεν είναι τίποτα περισσότερο από μια σειρά οδηγιών προς την ΚΜΕ του υπολογιστή, που την καθοδηγεί να κάνει συγκεκριμένες ενέργειες. Τα μικρά προγράμματα μπορούν να κατασκευαστούν συνήθως από ένα άτομο σαν μια μικρή διαδικασία, μια σειρά δηλαδή από συγκεκριμένες οδηγίες. Το ίδιο άτομο μπορεί να συντηρεί και να προσαρμόζει το πρόγραμμα σε νέες ανάγκες, να παίρνει σχεδιαστικές αποφάσεις, κλπ. Το ίδιο ισχύει για προγράμματα που κατασκευάζονται από μικρές ομάδες, με καλή επικοινωνία μεταξύ των μελών τους.

Τι γίνεται όμως όταν το πρόγραμμα είναι μεγάλο και εμπλέκονται στην κατασκευή του δεκάδες ή και εκατοντάδες άτομα σε μία ή περισσότερες ομάδες;

Τμηματικός προγραμματισμός

Η προφανής λύση είναι να τεμαχιστεί το μεγάλο πρόγραμμα σε μικρότερα τμήματα, τα οποία θα κατασκευαστούν ανεξάρτητα και στη συνέχεια θα συνδυαστούν, για να δημιουργηθεί το συνολικό σύστημα. Αυτή η **στρατηγική τμηματικός προγραμματισμός** (modular programming) και αποτελεί τη βασική αρχή πάνω στην οποία στηρίζεται η ανάπτυξη του λογισμικού από τη δεκαετία του ‘50 και μετά.



Υπορουτίνα

Κλήση της υπορουτίνας «εμβαδό» από δύο σημεία ενός προγράμματος

Βασικό εργαλείο για την υλοποίηση αυτής της αρχής ήταν η επινόηση της **υπορουτίνας** (subroutine) στις αρχές της δεκαετίας του ‘50. Η υπορουτίνα είναι ένα τμήμα κώδικα στο οποίο έχουμε δώσει ένα όνομα και μπορεί να εκτελεστεί σε οποιοδήποτε σημείο του προγράμματος απλώς με αναφο-

ρά στο όνομά της. Οι υπορουτίνες δίνουν το βασικό μηχανισμό για την τμηματική ανάπτυξη του λογισμικού. Οι διάφορες υπορουτίνες γράφονται ανεξάρτητα και στη συνέχεια συντίθενται και δημιουργούν το τελικό πρόγραμμα.
Παρ' όλο που οι υπορουτίνες προσφέρουν το βασικό μηχανισμό για τμηματικό προγραμματισμό, αυτές και μόνο δεν επαρκούν για τη δημιουργία ενός σωστά δομημένου λογισμικού. Είναι απαραίτητη μεγάλη πειθαρχία στον τρόπο χρήσης τους. Χωρίς αυτή την πειθαρχία είναι πολύ εύκολο να κατασκευαστούν προγράμματα εξαιρετικά πολύπλοκα και δυσνόητα, που δύσκολα τροποποιούνται εκ των υστέρων.

Δομημένος προγραμματισμός

Στα τέλη της δεκαετίας του 1960 υπήρξε μια μεγάλη προσπάθεια προς την κατεύθυνση της καθιέρωσης ενός πειθαρχημένου τρόπου γραφής προγραμμάτων. Το αποτέλεσμα ήταν η εξέλιξη του τμηματικού προγραμματισμού σε αυτό που ονομάστηκε **δομημένος προγραμματισμός** (structured programming). Βασική τεχνική του δομημένου προγραμματισμού είναι η διάσπαση των λειτουργιών του προγράμματος σε άλλες, απλούστερες και ανεξάρτητες κατά το δυνατό, επιμέρους λειτουργίες και στη συνέχεια η υλοποίησή τους με καθορισμένους τύπους δομών ελέγχου.
Οι δομές αυτές είναι:

- Η **διαδοχή** (ακολουθία). Οι εντολές βρίσκονται σε ακολουθία και εκτελούνται με τη σειρά που είναι γραμμένες.
- Η **επιλογή.** Η εκτέλεση των εντολών εξαρτάται από την τιμή αλήθειας της συνθήκης.
- Η **επανάληψη.** Οι εντολές επαναλαμβάνονται, εφόσον αληθεύει ή δεν αληθεύει η συνθήκη.

Η διάσπαση των λειτουργιών σε απλούστερες, που ονομάζεται και **λειτουργική αποσύνθεση** (functional decomposition), γίνεται από πάνω προς τα κάτω, δηλαδή ξεκινώντας από την πιο σύνθετη λειτουργία στην κορυφή και φθάνοντας στην πιο απλή στη βάση. Το πρόγραμμα διασπάται συνεχώς σε απλούστερα, έως ότου καταλήξουμε σε επίπεδο απλών υπορουτινών. Αυτές οι υπορουτίνες αναπτύσσονται ξεχωριστά και στη συνέχεια συντίθενται και δημιουργούν το τελικό πρόγραμμα.
Οι αρχές του δομημένου προγραμματισμού εφαρμόστηκαν αρχικά στη γλώσσα ALGOL, αλλά έγιναν ευρύτερα γνωστές μέσω της Pascal. Σήμερα όλες σχεδόν οι σύγχρονες γλώσσες είναι σχεδιασμένες έτσι ώστε να υποστηρίζουν αυτές τις αρχές.

Ιδρυτής του δομημένου προγραμματισμού θεωρείται ο Dijkstra, ο οποίος διατύπωσε τις πρώτες υποδείξεις που αφορούσαν τον περιορισμό της εντολής GOTO.

7.5 Πρότυπα προγραμματισμού

7.5.1 Διαδικαστικός προγραμματισμός

Το πρότυπο του διαδικαστικού προγραμματισμού είναι το παλαιότερο και ίσως το ευρύτερα χρησιμοποιούμενο. Συχνά αναφέρεται και ως **προστακτικός προγραμματισμός** (imperative programming). Σύμφωνα με αυτό το πρότυπο, το πρόγραμμα αποτελείται από δύο ξεχωριστά δομικά στοιχεία:
α) από εντολές που περιγράφουν βήμα βήμα τη διαδικασία επίλυσης του προβλήματος
β) από δομές δεδομένων, στις οποίες αποθηκεύονται τα δεδομένα του προβλήματος τα οποία χειρίζονται οι εντολές.

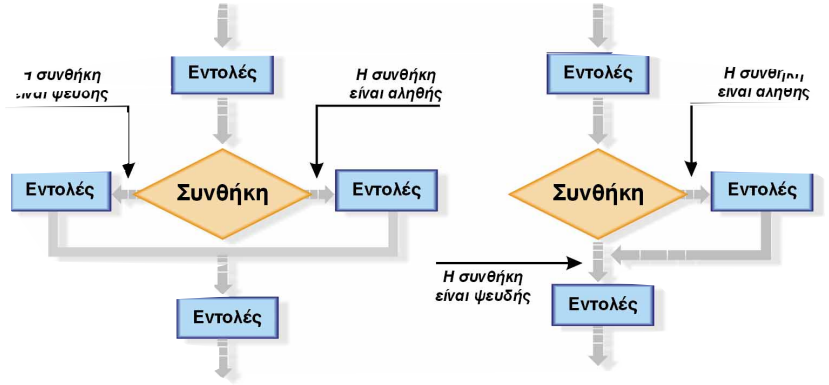
Οι εντολές που, κατ' ελάχιστο, περιέχονται σε κάθε διαδικαστική γλώσσα προγραμματισμού είναι:

Εντολές ανάθεσης. Με αυτές καταχωρίζονται τιμές σε θέσεις μνήμης. Οι θέσεις μνήμης αναφέρονται μέσω συμβολικών ονομάτων που ονομάζονται μεταβλητές και όχι μέσω της διεύθυνσής τους.

Παράδειγμα:

```
A:=B+C;
(πρόσθεσε το περιεχόμενο των θέσεων μνήμης με ονόματα
B και C και το αποτέλεσμα καταχώρισέ το στη θέση A)
```

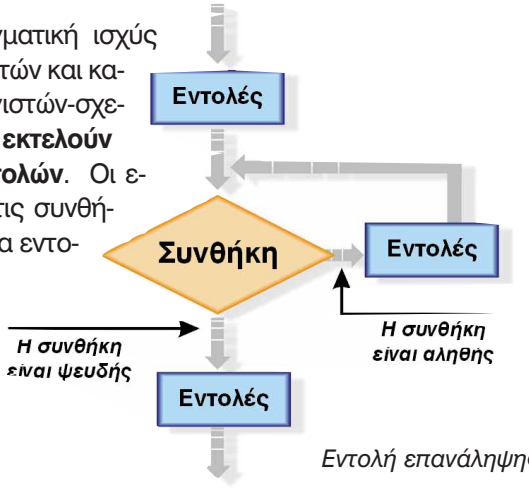
Εντολές συνθήκης. Με αυτές μια ομάδα εντολών εκτελείται ή όχι ανάλογα με την τιμή αλήθειας μιας συνθήκης και χρησιμοποιούνται για να υλοποιήσουν σημεία του αλγόριθμου στα οποία λαμβάνεται κάποια απόφαση.



Παράδειγμα:

```
If vath>=18.5 then
   print «Άριστα»
else
   print «Όχι άριστα»;
(Αν το περιεχόμενο της μεταβλητής vath είναι μεγαλύτερο ή ίσο του 18.5 εκτύπωσε «arista» αλλιώς εκτύπωσε «oxi arista»)
```

Εντολές επανάληψης. Η πραγματική ισχύς των προγραμμάτων των υπολογιστών και κατά συνέπεια των ίδιων των υπολογιστών-σχετίζεται με τη δυνατότητά τους να **εκτελούν κατ' επανάληψη ένα σύνολο εντολών**. Οι εντολές επανάληψης καθορίζουν τις συνθήκες κάτω από τις οποίες μια ομάδα εντολών εκτελείται επαναληπτικά.



Εντολή επανάληψης

Παράδειγμα:

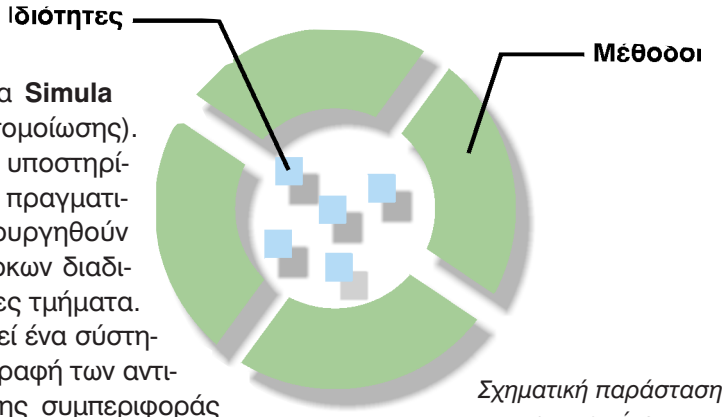
```
while a>0 do
begin
   a := a -1;
   x :=x+a
end;
(Για όσο διάστημα η μεταβλητή a έχει θετικό περιεχόμενο, εκτέλεσε (συνεχώς) τις εντολές a := a -1 και x:=x+a)
```

Εντολές για είσοδο και έξοδο στοιχείων. Είναι οι εντολές με τις οποίες το πρόγραμμα χειρίζεται τη ροή δεδομένων από και προς τις περιφερειακές μονάδες του υπολογιστή. Π.χ.:

```
readln(a);
(Διάβασε από την είσοδο μια τιμή και καταχώρισέ τη στη
μεταβλητή a)
writeln(a);
(Τύπωσε το περιεχόμενο της μεταβλητής a στην έξοδο)
```

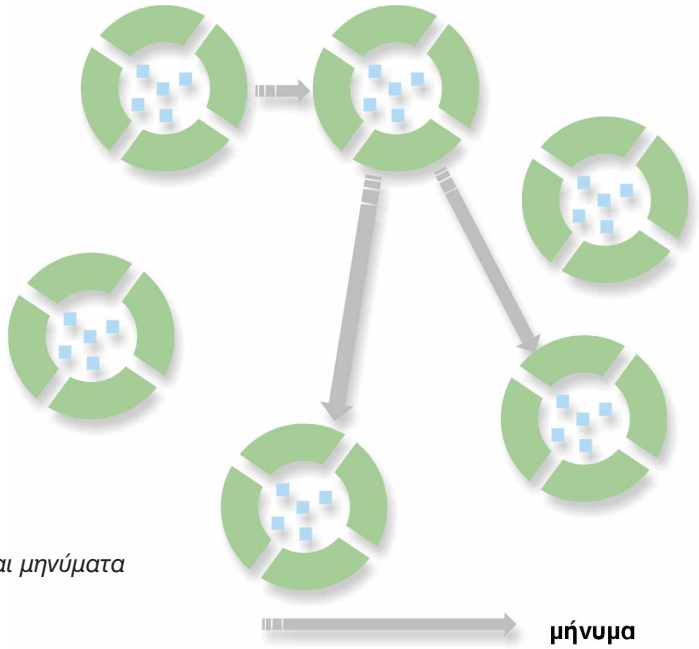
7.5.2 Αντικειμενοστρεφής προγραμματισμός

Ο **αντικειμενοστρεφής προγραμματισμός** (object oriented programming), αν και έχει έλθει στο προσκήνιο τα τελευταία χρόνια, στηρίζεται σε αρχές που έχουν τις ρίζες τους στη δεκαετία του '60 και συγκεκριμένα στη γλώσσα **Simula** (**Simu**lation **La**nguage - Γλώσσα προσομοίωσης). Η Simula δημιουργήθηκε με σκοπό να υποστηρίξει την προσομοίωση διαδικασιών του πραγματικού κόσμου. Στόχος ήταν να δημιουργηθούν ακριβή υπολογιστικά μοντέλα πολύπλοκων διαδικασιών, αποτελούμενα από εκατοντάδες τμήματα. Ένας φυσικός τρόπος για να περιγραφεί ένα σύστημα με προσομοίωση είναι να γίνει περιγραφή των αντικειμένων από τα οποία αποτελείται, της συμπεριφοράς τους καθώς και του τρόπου με τον οποίο αλληλεπιδρούν μεταξύ τους. Αυτή είναι και η φιλοσοφία του αντικειμενοστρεφούς προγραμματισμού.



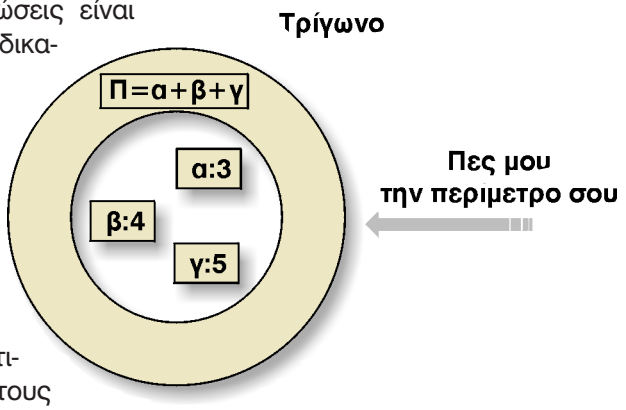
Σχηματική παράσταση αντικειμένου

Στην αντικειμενοστρεφή προσέγγιση προγραμματισμού, το **αντικείμενο** (object)



Αντικείμενα και μηνύματα

είναι ένα προγραμματιστικό «πακέτο», το οποίο αποτελείται από διαδικασίες και δεδομένα σχετιζόμενα μεταξύ τους. Αυτές οι διαδικασίες ονομάζονται **μέθοδοι** (methods), ενώ τα δεδομένα ονομάζονται **ιδιότητες** (properties). Οι μέθοδοι σε πολλές περιπτώσεις είναι γραμμένες σε κάποια διαδικαστική γλώσσα.



Το πρόγραμμα, σε αυτήν την προσέγγιση, αποτελείται από **αντικείμενα**, τα οποία μπορούν να αλληλεπιδρούν μεταξύ τους με **μηνύματα** (messages). Τα μηνύματα ενεργοποιούν τις μεθόδους των αντικειμένων, που με τη σειρά τους μπορεί να στείλουν άλλα μηνύματα.

Για παράδειγμα, σε ένα πρόγραμμα γεωμετρίας ένα αντικείμενο θα μπορούσε να είναι ένα τρίγωνο. Ως ιδιότητες θα μπορούσε να έχει τα μήκη των πλευρών του, ενώ ως μεθόδους τον υπολογισμό της περιμέτρου και του εμβαδού του. Ένα μήνυμα που θα μπορούσε να δεχθεί από ένα άλλο αντικείμενο είναι «`πες μου την περίμετρό σου`».

Ως ιδιαίτερα γνωστές γλώσσες αντικειμενοστρεφούς προγραμματισμού θα πρέπει να αναφερθούν η Smaltak, η C++ και η Java.

7.5.3 Λογικός προγραμματισμός

Ο **λογικός προγραμματισμός** (logic programming) είναι επηρεασμένος και εμπνευσμένος από τη μαθηματική λογική. Ένα πρόγραμμα γραμμένο σε μια γλώσσα λογικού προγραμματισμού, όπως η Prolog, είναι ένα σύνολο από λογικές προτάσεις και από ένα μηχανισμό εξαγωγής συμπερασμάτων, μέσω του οποίου μπορούμε να υποβάλουμε ερωτήσεις στο πρόγραμμα και αυτό να μας απαντήσει αν αληθεύουν ή όχι.

Για παράδειγμα, παρατίθεται ένα πρόγραμμα σε γλώσσα Prolog, που περιγράφει το γενεαλογικό δένδρο των θεών του Ολύμπου. Περιέχει ορισμένες λογικές προτάσεις που αφορούν ορισμούς συγγένειας. Η σύνταξη δεν είναι αυστηρή για λόγους αναγνωσιμότητας.

```
πατέρας(Δίας, Άρης).
πατέρας(Δίας, Διόνυσος).
πατέρας(Άρης, Αρμονία).
πατέρας(Κάδμος, Σεμέλη).
μητέρα(Ήρα, Άρης).
μητέρα(Αρμονία, Σεμέλη).
μητέρα(Σεμέλη, Διόνυσος).
μητέρα(Αφροδίτη, Αρμονία).
αδέλφια(X,Y) αν μητέρα(Z,X) και μητέρα(Z,Y).
αδέλφια(X,Y) αν πατέρας(Z,X) και πατέρας(Z,Y).
```

Οι κανόνες «πατέρας» και «μητέρα» κωδικοποιούν το ποιος είναι πατέρας ή μητέρα ποιου, ενώ το «αδέλφια» ορίζει πως για να είναι δύο αδέλφια (ο Χ και ο Υ) πρέπει να έχουν τον ίδιο πατέρα ή την ίδια μητέρα (τον Ζ).
Στο μηχανισμό εξαγωγής συμπερασμάτων μπορούμε να υποβάλουμε την ερώτηση: «είναι ο Δίας πατέρας του Διόνυσου» και να μας απαντήσει «ΝΑΙ» ή να ρωτήσουμε «ποια είναι τα αδέλφια του Άρη» και να απαντήσει «ο Διόνυσος».

7.5.4 Συναρτησιακός προγραμματισμός

Ο **συναρτησιακός προγραμματισμός** (functional programming), που συχνά ονομάζεται και **εφαρμοστικός προγραμματισμός** (applicative programming), έχει ως βάση τη συνάρτηση με τη μαθηματική της έννοια, δηλαδή αυτή της απεικόνισης από ένα πεδίο ορισμού σε ένα πεδίο τιμών. Η εφαρμογή συναρτήσεων σε δεδομένα που αποτελούν τα ορίσματα της συνάρτησης είναι η μόνη δομή ελέγχου σε μια γλώσσα συναρτησιακού προγραμματισμού. Το σύστημα αντιμετωπίζει τις ίδιες τις συναρτήσεις ως δεδομένα και επομένως αυτές μπορούν να αποτελούν ορίσματα ή αποτελέσματα άλλων συναρτήσεων.
Βασική έννοια σε αυτό το πρότυπο είναι η έννοια της αναδρομής, δηλαδή η άμεση ή έμμεση εφαρμογή μιας συνάρτησης πάνω στον εαυτό της.
Στο πιο κάτω παράδειγμα έχουμε ένα πρόγραμμα της συναρτησιακής γλώσσας Miranda, το οποίο επιλύει τη δευτεροβάθμια εξίσωση $ax^2 + bx + c = 0$.

```
quadsolve a b c
     = error "Μιγαδικές ρίζες", if delta < 0
     = [-b/2*a)],if delta = 0
     = [-b/(2*a) + radix/(2*a),
        -b/(2*a) - radix/(2*a)]if delta > 0
where
delta = b*b - 4*a*c
radix = sqrt delta

-----------------------------------
sqrt a: είναι συνάρτηση που βρίσκει
        την τετραγωνική ρίζα του a
```

7.6 Προγραμματίζοντας

Επειδή ο προγραμματισμός είναι κατά βάση μια δραστηριότητα επίλυσης προβλημάτων, στη συνέχεια θα παρουσιάσουμε μια συστηματική προσέγγιση στην επίλυση προβλημάτων με τη χρήση υπολογιστή, η οποία καλείται **μέθοδος ανάπτυξης λογισμικού.**
Τα βήματα αυτής της μεθόδου είναι:

α) Προσδιορισμός των απαιτήσεων του προβλήματος.
β) Ανάλυση του προβλήματος.
γ) Σχεδιασμός αλγόριθμου για την επίλυση του προβλήματος.
δ) Υλοποίηση του αλγόριθμου.
ε) Έλεγχος και επαλήθευση του τελικού προγράμματος.
στ) Συντήρηση και ενημέρωση του προγράμματος.
ζ) Τεκμηρίωση.

7.6.1 Καθορισμός προβλήματος

Ο προσδιορισμός των απαιτήσεων του προβλήματος είναι ένα στάδιο στο οποίο θα πρέπει να καθοριστεί με σαφήνεια το πρόβλημα και να υπάρξει ξεκάθαρη αντίληψη του είδους των απαιτήσεων για τη λύση του. Ο στόχος είναι να εξαλείψουμε τα ασήμαντα στοιχεία και να οδηγηθούμε στην ουσία του προβλήματος. Αυτό δεν είναι τόσο απλό, όσο ίσως φαίνεται. Μπορεί να σημαίνει ότι πρέπει να ζητήσουμε πρόσθετη πληροφορία από αυτόν ή αυτούς που έθεσαν το πρόβλημα.

7.6.2 Ανάλυση του προβλήματος

Η ανάλυση του προβλήματος περιλαμβάνει:

- τον καθορισμό των «εισόδων», δηλαδή των δεδομένων με τα οποία θα δουλέψουμε (inputs)
- τον καθορισμό των «εξόδων», δηλαδή των απαιτουμένων αποτελεσμάτων (outputs)
- την αναζήτηση άλλων απαιτήσεων ή περιορισμών.

Σε αυτό το στάδιο θα καθοριστεί και η μορφή στην οποία θέλουμε να δίνονται τα αποτελέσματα. Είναι επίσης χρήσιμο να προσδιορίσουμε τις παραμέτρους του προβλήματος και να καταγράψουμε πιθανές σχέσεις μεταξύ τους, ίσως με τη μορφή μαθηματικών σχέσεων.
Οποιαδήποτε παρανόηση είτε σε αυτό το επίπεδο είτε στο προηγούμενο μπορεί να οδηγήσει στην επίλυση άλλου προβλήματος από αυτό που στην πραγματικότητα απαιτείται.

7.6.3 Σχεδιασμός

Ο **αλγόριθμος** είναι μια σειρά από βήματα, τα οποία πρέπει να ακολουθήσουμε για να επιλύσουμε ένα συγκεκριμένο πρόβλημα. Τα βήματα πρέπει να είναι σαφή και να υπάρχει συγκεκριμένο σημείο περάτωσης της διαδικασίας.

Επόμενο βήμα είναι η σχεδίαση του αλγόριθμου για τη λύση του προβλήματος, δηλαδή των βημάτων που περιγράφουν τη λύση. Συχνά ο αλγόριθμος είναι το πιο δύσκολο τμήμα της μεθόδου.
Η λύση δεν χρειάζεται να περιγραφεί από την αρχή με κάθε λεπτομέρεια. Αντί γι' αυτό μπορεί να χρησιμοποιηθεί η «από πάνω προς τα κάτω προσέγγιση». Δηλαδή αρχικά να καταγραφούν τα βασικά βήματα ή υποπροβλήματα, που απαιτούνται για την επίλυση του προβλήματος, και στη συνέχεια να εστιάσουμε την προσοχή μας στο κάθε υποπρόβλημα χωριστά αναλύοντας περαιτέρω τα βήματα.

7.6.4 Υλοποίηση

Η υλοποίηση του αλγόριθμου αφορά τη συγγραφή του προγράμματος. Αυτό σημαίνει ότι κάθε βήμα του αλγόριθμου πρέπει να μετατραπεί σε μια ή περισσότερες εντολές κάποιας γλώσσας προγραμματισμού.
Ο δομημένος προγραμματισμός είναι μια πειθαρχημένη προσέγγιση στη διαδικασία συγγραφής, που οδηγεί στη δημιουργία προγραμμάτων, τα οποία όχι μόνο είναι εύκολο να διαβαστούν και να γίνουν κατανοητά, αλλά και να πε-

ριέχουν τα λιγότερα λάθη. Αυτό πρακτικά σημαίνει την εφαρμογή ορισμένων κανόνων που οδηγούν σε κώδικα ευανάγνωστο και εύκολο να συντηρηθεί.
Όπως έχουμε αναφέρει, η από πάνω προς τα κάτω λειτουργική αποσύνθεση είναι εκείνη που μας δίνει με ένα συστηματικό τρόπο τα βασικά τμήματα (modules) από τα οποία θα χτιστεί το τελικό πρόγραμμα.

7.6.5 Έλεγχος

Μια πολύ σοβαρή φάση της όλης ανάπτυξης του προγράμματος είναι αυτή του ελέγχου. Αφορά την εκτέλεση του προγράμματος για διάφορα -σωστά επιλεγμένα- σύνολα τιμών εισόδου και τον έλεγχο των αντίστοιχων εξόδων.

7.6.6 Συντήρηση

Ο όρος συντήρηση κατ' αρχάς είναι παράξενος, όταν αναφέρεται σε κάτι άυλο όπως είναι ένα πρόγραμμα. Το πρόγραμμα δεν χαλάει, δεν σκουριάζει, δεν φθείρεται και όμως χρειάζεται συντήρηση. Δύο είναι οι βασικοί λόγοι:

- για να διορθωθούν λάθη τα οποία εντοπίστηκαν, αφού το πρόγραμμα παραδόθηκε για χρήση, κατά τη διάρκεια δηλαδή χρήσης του προγράμματος
- για να γίνουν αλλαγές που προέκυψαν με την πάροδο του χρόνου, λόγω αλλαγών στο περιβάλλον χρήσης του προγράμματος. Για παράδειγμα, επειδή άλλαξε ο τρόπος υπολογισμού του Φ.Π.Α. για ένα πρόγραμμα τιμολόγησης.

Συναφής με το θέμα της συντήρησης του προγράμματος είναι η έννοια του αριθμού έκδοσης ή απλώς έκδοσης (version number, version).
Ας υποτεθεί ότι ένα πρόγραμμα με όνομα «`ΤΟ!πρόγραμμα`» παραδόθηκε για χρήση την 1/1/2000. Τους πρώτους μήνες της χρήσης του εντοπίστηκαν κάποια λάθη και την 1/8/2000 δόθηκε για χρήση το πρόγραμμα διορθωμένο. Θα πρέπει το διορθωμένο πρόγραμμα με κάποιο τρόπο να έχει διαφορετική ονομασία για να ξεχωρίζει από την αρχική έκδοση αλλά και συγχρόνως να φαίνεται ότι πρόκειται για το ίδιο πρόγραμμα. Κατά πάγια πρακτική στο χώρο του λογισμικού, αυτό επιτυγχάνεται με τη χρήση αρίθμησης για τον καθορισμό των εκδόσεων. Έτσι το αρχικό πρόγραμμα, ή για την ακρίβεια η αρχική έκδοση, πρέπει να λέγεται όχι απλώς «`ΤΟ!πρόγραμμα`», αλλά «`ΤΟ!πρόγραμμα έκδοση 1.0`». Η διορθωμένη τότε έκδοση μπορεί να λέγεται «`ΤΟ!πρόγραμμα έκδοση 1.1`». Μια επόμενη έκδοση, με πολλές αλλαγές, μπορεί να λέγεται «`ΤΟ!πρόγραμμα έκδοση 2.7`».
Η αρίθμηση γίνεται με παραπάνω από έναν αριθμούς, χωρισμένους με τελεία, και συνηθίζεται να είναι αύξουσα.
Όταν αγοράζουμε λογισμικό θα πρέπει να προσέχουμε ο αριθμός έκδοσης του προγράμματος να είναι ο πιο πρόσφατος.

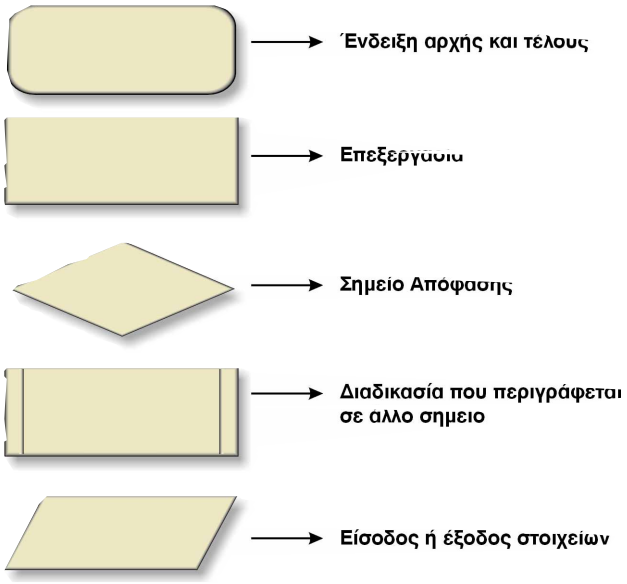
7.6.7 Τεκμηρίωση

Η τεκμηρίωση αφορά το σύνολο των εγγράφων που προκύπτουν καθ' όλη τη διάρκεια της ανάπτυξης του προγράμματος και βοηθούν το έργο της συντή-

ρησής του αλλά και τη χρήση του. Αυτό το είδος της τεκμηρίωσης λέγεται **εξωτερική**. Συνήθως, καθένα από τα στάδια που αναφέραμε παράγει και ένα έγγραφο που αναφέρεται στα θέματα του κάθε σταδίου. Π.χ. στο στάδιο του σχεδιασμού το αντίστοιχο έγγραφο θα πρέπει να περιέχει την περιγραφή του αλγόριθμου που επιλέχθηκε για τη λύση του προβλήματος.
Η **εσωτερική** τεκμηρίωση αφορά οποιαδήποτε μορφή τεκμηρίωσης βρίσκεται καταχωρισμένη στον πηγαίο κώδικα του προγράμματος. Συνήθως αποτελείται από σχόλια σε κατάλληλα σημεία του προγράμματος -π.χ. πριν από κάθε υποπρόγραμμα αδρή περιγραφή της λειτουργίας του- και από σωστή και πειθαρχημένη επιλογή των ονομάτων των διαφόρων οντοτήτων του προγράμματος -μεταβλητές, υπορουτίνες, κλπ. Είναι χρησιμότερο από άποψη τεκμηρίωσης το υποπρόγραμμα υπολογισμού εμβαδού τριγώνου να ονομάζεται «emvado_trigonou» παρά «f» ή «et».
Παρ' όλο που σε ορισμένες περιπτώσεις δεν δίνεται η βαρύτητα που πρέπει στη σωστή τεκμηρίωση, θα πρέπει αυτή να θεωρείται εξίσου σημαντική με το ίδιο το πρόγραμμα.

Τρόποι παράστασης αλγόριθμων

Οι αλγόριθμοι μπορούν να παρασταθούν στο χαρτί με διάφορους τρόπους. Τρεις είναι οι περισσότερο δημοφιλείς.
α) με φυσική γλώσσα
β) με ψευδοκώδικα και
γ) με λογικά διαγράμματα.
Η παράσταση του αλγόριθμου με τη χρήση **φυσικής γλώσσας** είναι η λιγότερο δομημένη προσέγγιση. Σε αυτήν ο αλγόριθμος περιγράφεται σαν ένα απλό κείμενο.
Ο **ψευδοκώδικας** μοιάζει με την περιγραφή του αλγόριθμου σε φυσική γλώσσα, με τη διαφορά ότι χρησιμοποιεί συγκεκριμένες δομές για την περιγραφή του αλγόριθμου, που είναι πολύ κοντά στις εντολές των γλωσσών υψηλού επιπέδου.

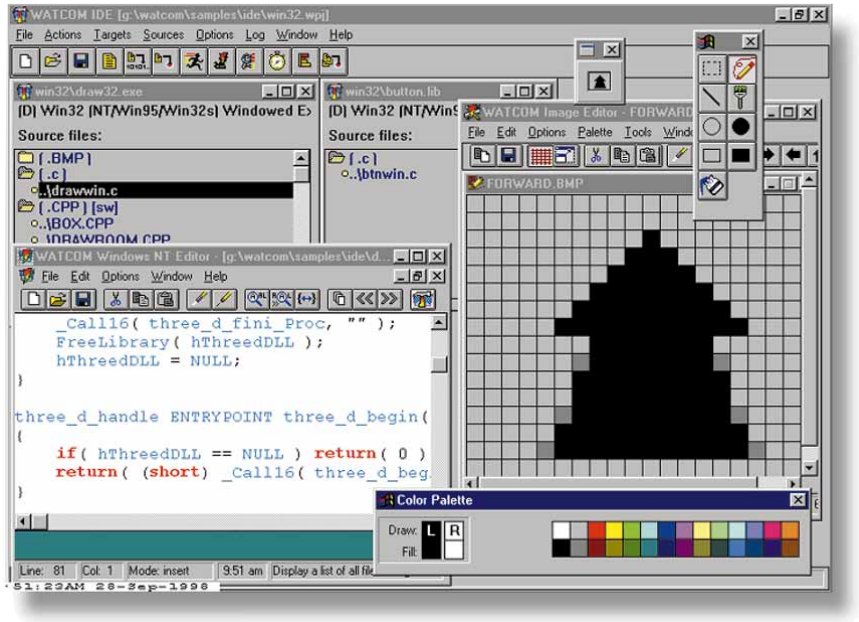


Βασικά στοιχεία λογικών διαγραμμάτων

Τα **λογικά διαγράμματα** είναι ένας γραφικός τρόπος παρουσίασης των βημάτων του αλγόριθμου. Υπάρχει ένα σύνολο από τυποποιημένα σχήματα που παριστάνουν τις διάφορες δομές ενός αλγόριθμου.
Στο παράδειγμα στο τέλος του κεφαλαίου ο αλγόριθμος έχει παρουσιαστεί με ψευδοκώδικα και με λογικό διάγραμμα.

7.7 Προγραμματιστικά περιβάλλοντα

Όπως είδαμε στην αρχή του κεφαλαίου, οι κατασκευαστές λογισμικού κατασκευάζουν τα προγράμματα του υπολογιστή, αλλά και αυτοί με τη σειρά τους χρησιμοποιούν τα προγράμματα άλλων για να κάνουν τη δική τους εργασία. Τα εργαλεία λογισμικού που αξιοποιούνται στην ανάπτυξη των προγραμμάτων αποτελούν το **προγραμματιστικό περιβάλλον**.



Ολοκληρωμένο περιβάλλον ανάπτυξης προγραμμάτων σε γλώσσα C και C++ της εταιρείας WATCOM

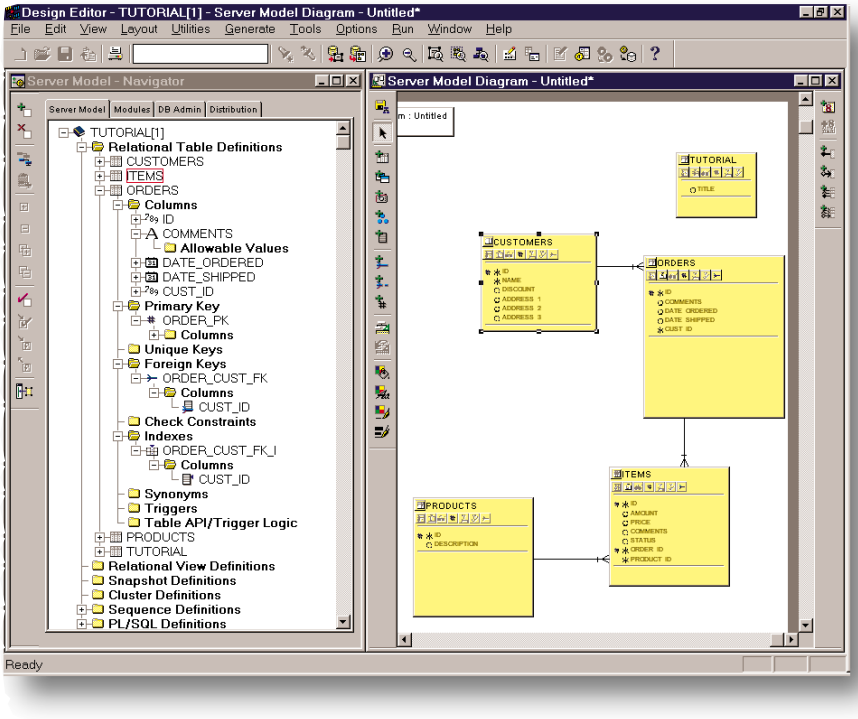
Ένα προγραμματιστικό περιβάλλον συνήθως αποτελείται από εργαλεία όπως:
α) Ένα συντάκτη κειμένων, με τον οποίο γράφεται το πηγαίο πρόγραμμα.
β) Μεταφραστικά προγράμματα (assembler, compiler, interpreter).
γ) Εργαλεία εντοπισμού λαθών (debugger).

Η σύγχρονη τάση σε αυτό το χώρο είναι τα ονομαζόμενα **ολοκληρωμένα** συστήματα. Πρόκειται για προγραμματιστικά περιβάλλοντα τα οποία προσφέρουν όλα τα απαραίτητα προγραμματιστικά εργαλεία μέσα από ένα ενοποιημένο περιβάλλον με κοινή διεπαφή χρήστη (User Interface).
Υπάρχει μια μεγάλη ποικιλία τέτοιων εργαλείων, τα οποία διευκολύνουν τις διάφορες φάσεις ανάπτυξης των προϊόντων λογισμικού. Τα εργαλεία αυτά μπορεί να είναι αυτόνομα προϊόντα ή να αποτελούν τμήμα ενός ολοκληρω-

CASE tools

μένου περιβάλλοντος ανάπτυξης. Στην πιο ολοκληρωμένη τους μορφή ονομάζονται εργαλεία ανάπτυξης λογισμικού με τη βοήθεια υπολογιστή-**CASE tools.**

Με τα εργαλεία αυτά επιτυγχάνεται μεγαλύτερη παραγωγικότητα και υψηλότερη ποιότητα του παραγόμενου λογισμικού και μάλιστα με το μεγαλύτερο βαθμό αυτοματοποίησης σε όλα τα στάδια, από τον καθορισμό των απαιτήσεων έως, σε ορισμένες περιπτώσεις, την αυτόματη παραγωγή κώδικα.



Σχεδίαση της βάσης δεδομένων στο περιβάλλον του Designer 2000 της ORACLE (Case tool)

Τα εργαλεία CASE μπορούν να χρησιμοποιηθούν είτε για την παραγωγή του κώδικα της τελικής εφαρμογής ή πιο συχνά για την κατασκευή ενός προτύπου της εφαρμογής. Σε αυτήν την περίπτωση χρησιμοποιούμε τη μέθοδο της **ταχείας ανάπτυξης πρωτοτύπου** - (rapid prototyping). Με τη μέθοδο αυτή ελέγχεται σ΄ ένα πολύ πρώιμο στάδιο κατά πόσο η σχεδίαση ικανοποιεί τους στόχους του υπό κατασκευή συστήματος.

Μια τάση που κερδίζει συνεχώς έδαφος είναι η σχεδίαση μιας ολόκληρης γενιάς «οπτικών» (visual) εργαλείων και περιβαλλόντων ανάπτυξης, που αξιοποιούν τις δυνατότητες των γραφικών διεπαφών χρήστη, προκειμένου να επιτευχθεί ένα πιο φιλικό και παραγωγικό περιβάλλον εργασίας.

Οπτικός προγραμματισμός

Η αξιοποίηση γραφικών μεθόδων στην ανάπτυξη προγραμμάτων αναφέρεται ως **οπτικός προγραμματισμός** (visual programming). Με αυτόν η παραγωγή του κώδικα επιτυγχάνεται με γραφικές μεθόδους, π.χ. τα βήματα του αλγόριθμου σχεδιάζονται με τη βοήθεια διαγραμμάτων ή τα αντικείμενα της γραφικής διεπαφής χρήστη κατασκευάζονται από παλέτες εργαλείων.

Εκτός από τα προηγούμενα, στην κατηγορία των προγραμματιστικών περιβαλλόντων ανήκουν περιβάλλοντα με τα οποία γίνεται η ανάπτυξη των πιο μο-

ντέρνων εφαρμογών, όπως είναι οι εφαρμογές πολυμέσων και οι εφαρμογές που στηρίζονται στην τεχνολογία των ιστοσελίδων. Ο στόχος αυτών των περιβαλλόντων είναι αφενός μεν να αυξήσουν την παραγωγικότητα, αφετέρου δε να απαλλάξουν τον κατασκευαστή των αντίστοιχων εφαρμογών από λεπτομέρειες υλοποίησης, ώστε να εστιάσει την προσοχή του σε περισσότερο σημαντικά θέματα, όπως είναι η παρουσίαση του περιεχομένου και η αισθητική σχεδίαση της εφαρμογής. Στα κεφάλαια 11 και 12 θα αναφερθούμε σε παραδείγματα τέτοιων περιβαλλόντων.

7.8 Ένα παράδειγμα ανάπτυξης προγράμματος

Στη συνέχεια θα δούμε τον τρόπο με τον οποίο εφαρμόζεται η μεθοδολογία που περιγράψαμε στην παράγραφο «προγραμματίζοντας» κατά την ανάπτυξη ενός προγράμματος. Ως πρόβλημα τίθεται η κατασκευή ενός προγράμματος για την επίλυση εξισώσεων δευτέρου βαθμού.

7.8.1 Προσδιορισμός των απαιτήσεων του προβλήματος

Θέλουμε να κατασκευάσουμε ένα πρόγραμμα που να επιλύει εξισώσεις β' βαθμού, για οποιονδήποτε συνδυασμό των σταθερών όρων της.

7.8.2 Ανάλυση του προβλήματος

Το πρώτο βήμα για τη λύση του προβλήματος είναι να προσδιορίσουμε τι ακριβώς μας ζητείται να κάνουμε. Πρέπει να κατασκευάσουμε πρόγραμμα που να βρίσκει τις ρίζες της εξίσωσης: $\alpha x^2 + \beta x + \gamma$. Θα πρέπει να βρίσκουμε τις ρίζες της εξίσωσης για όλες τις πραγματικές τιμές των α, β, γ. Υπάρχουν ορισμένα ζητήματα ωστόσο, που θα πρέπει να διευκρινιστούν από αυτόν που θέτει το πρόβλημα:
α) Το α μπορεί να είναι μηδέν; (τότε η εξίσωση εκφυλίζεται σε 1ου βαθμού).
β) Θα βρίσκουμε τις μιγαδικές ρίζες;
Ας υποτεθεί ότι πρέπει το α να είναι διάφορο από 0 και στην περίπτωση των μιγαδικών ριζών θα τυπώνεται σχετικό μήνυμα.
Στη συνέχεια προσδιορίζουμε τα δεδομένα εισόδου και τα αντίστοιχα στοιχεία εξόδου του προγράμματος. Επίσης καταγράφουμε τύπους ή άλλες πληροφορίες που πρέπει να γνωρίζουμε για τη λύση του προβλήματος, όπως:

- Είσοδος
  Οι συντελεστές α, β, γ
- Έξοδος
  Οι ρίζες της εξίσωσης
  Μήνυμα για την περίπτωση των μιγαδικών ριζών
  Μήνυμα λάθους αν α=0
- Σχετικοί τύποι

  Οι ρίζες δίνονται από τον τύπο $\frac{-\beta \pm \sqrt{\Delta}}{2\alpha}$, όπου $\Delta = \beta^2 - 4\alpha\gamma$ είναι η

διακρίνουσα
- ◇ αν Δ>0 η εξίσωση έχει δύο πραγματικές ρίζες
- ◇ αν Δ=0 η εξίσωση έχει δύο ίσες ρίζες
- ◇ αν Δ<0 η εξίσωση έχει μιγαδικές ρίζες.

7.8.3 Σχεδιασμός αλγόριθμου για την επίλυση του προβλήματος

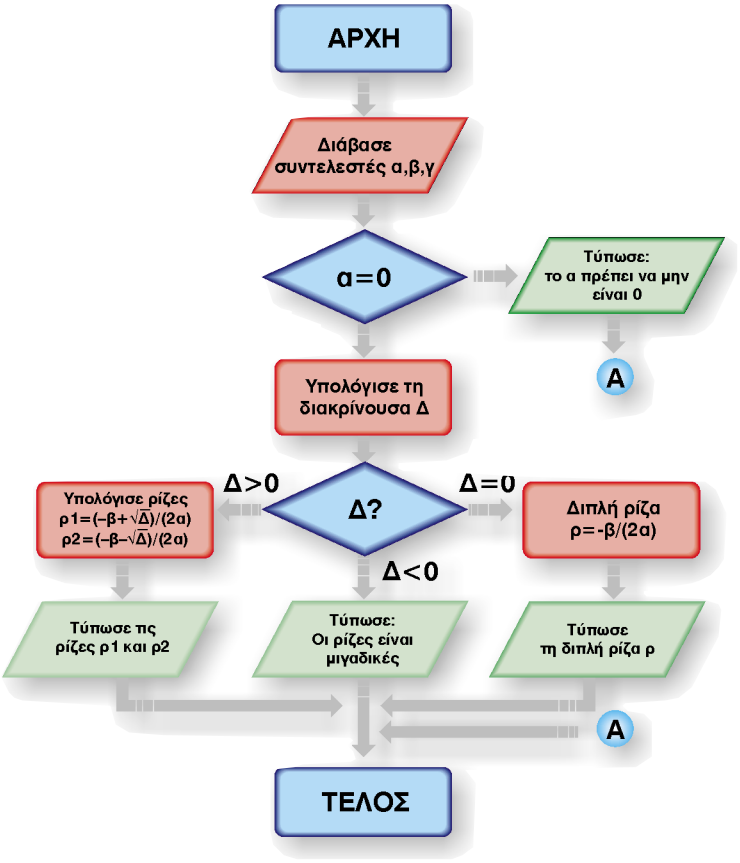
Στη συνέχεια σχεδιάζουμε τον αλγόριθμο που θα επιλύσει το πρόβλημα. Αρχίζουμε καταγράφοντας τα βασικά βήματα ή υποπροβλήματα του αλγόριθμου:

1. Διάβασε τις παραμέτρους της εξίσωσης
2. Έλεγξε την ορθότητα των παραμέτρων
3. Υπολόγισε τις ρίζες της εξίσωσης.

Ακολούθως εξετάζουμε αν κάποια από τα βήματα απαιτούν περαιτέρω ανάλυση σε υπο-υποπροβλήματα. Παρατηρούμε ότι το βήμα 1 είναι απλό και δεν χρειάζεται περαιτέρω ανάλυση. Τα βήματα 2 και 3 όμως χρειάζονται. Έτσι ο αλγόριθμος διαμορφώνεται ως εξής:

1. Διάβασε τις παραμέτρους της εξίσωσης
2. Έλεγξε την ορθότητα των παραμέτρων
   2.1 Αν α=0 τότε
   - Δώσε σχετικό μήνυμα λάθους
   - ΤΕΛΟΣ αλγόριθμου
3. Υπολόγισε τις ρίζες της εξίσωσης
   3.1 Υπολόγισε τη διακρίνουσα Δ
   3.2 Αν Δ>0 τότε
   - υπολόγισε τις δύο διαφορετικές ρίζες
   - τύπωσε τις ρίζες
   3.3 Αν Δ=0 τότε
   - υπολόγισε τη διπλή ρίζα
   - τύπωσε τη ρίζα
   3.4 Αν Δ<0 τότε
   - τύπωσε μήνυμα «οι ρίζες είναι μιγαδικές».
4. ΤΕΛΟΣ αλγόριθμου

Στη συνέχεια εξετάζουμε αν υπάρχουν βήματα που χρειάζονται περαιτέρω ανάλυση και συνεχίζουμε με τον ίδιο τρόπο. Στο πρόβλημά μας δεν υπάρχει ανάγκη για κάτι τέτοιο.
Στη συνέχεια δίνεται ο ίδιος αλγόριθμος με τη μορφή ψευδοκώδικα και λογικού διαγράμματος.




```
Program Defterovathmia
Variables
  a, b, c: real
  d, r1, r2, r: real
Begin
  read(a,b,c)
  if a=0 then
    write('Το a πρέπει να ειναι διάφορο από 0')
  else
    d <- b^2 - 4ac
    if d > 0 then
      r1 <- (-b + √d )/(2a)
      r2 <- (-b - √d )/(2a)
      write(r1, r2)
    else
      if d = 0 then
        r <- -b / (2a)
        write('Διπλή ρίζα', r)
      else
        write('Μιγαδικές ρίζες')
      end if
    end if
  end if
End
```

7.8.4 Υλοποίηση του αλγόριθμου

Αφού σχεδιαστεί ο αλγόριθμος στη συνέχεια τον υλοποιούμε σε κάποια γλώσσα προγραμματισμού. Στο επόμενο σχήμα βλέπουμε μια πιθανή υλοποίηση σε γλώσσα Pascal.

```
program exis_2_vatmoy(input, output);
var
  a, b, c :real;
  diak    :real;
  r1, r2  :real;
begin
readln(a, b, c);
if a=0 then
  writeln('Το a πρέπει να είναι διάφορο από το μηδέν')
  writeln ('εξίσωση α΄ βαθμού')
else
  begin
  diak := b*b-4*a*c;
  if diak>0 then
    begin
    r1 := (-b+sqrt(diak))/(2*a);
    r2 := (-b-sqrt(diak))/(2*a);
    writeln('1η ρίζα:', r1:10:5);
    writeln('2η ρίζα:', r2:10:5)
    end
  else     if diak=0 then
      begin
        r1 := -b/(2*a);
        writeln('Διπλή ρίζα: ', r1:10:5)
      end
  else
    writeln('Οι ρίζες είναι μιγαδικές');
  end
end.
```

7.8.5 Έλεγχος του προγράμματος

Αφού κατασκευάσουμε το πρόγραμμα, θα πρέπει να ελέγξουμε κατά πόσο είναι σωστό. Για το σκοπό αυτό πρέπει να «εκτελέσουμε» ή να «τρέξουμε» το πρόγραμμα, δίνοντάς του να επιλύσει εξισώσεις 2ου βαθμού διαφόρων κατηγοριών. Ο στόχος μας είναι να καλύψουμε όλες τις πιθανές κατηγορίες εξισώσεων, ώστε να διαπιστώσουμε αν σε κάθε περίπτωση δίνεται η αναμενόμενη απάντηση.

Ένα τέτοιο σύνολο τιμών για τους συντελεστές α, β, γ υπάρχει στον πίνακα που ακολουθεί.

α/α	α	β	γ	Κατηγορία εξίσωσης
1	2	4	-5	δύο πραγματικές ρίζες
2	-2	-4	5	δύο πραγματικές ρίζες
3	1	6	9	διπλή ρίζα
4	1	0	0	ειδική περίπτωση (κάποιος συντελεστής 0)
5	1	1	0	ειδική περίπτωση (κάποιος συντελεστής 0)
6	1	0	1	ειδική περίπτωση (κάποιος συντελεστής 0) επίσης μιγαδικές ρίζες
7	0	1	1	εξίσωση α΄ βαθμού
8	2	1	5	μιγαδικές ρίζες

Στο Τετράδιο Εργασίας Μαθητή στο αντίστοιχο κεφάλαιο, υπάρχει εισαγωγή στη γλώσσα προγραμματισμού Pascal εμπλουτισμένη με παραδείγματα.

Η διαδικασία του ελέγχου ενός προγράμματος ή ενός συστήματος λογισμικού αποτελεί ένα πολύ σοβαρό τμήμα της όλης διαδικασίας. Στο παράδειγμά μας ήταν σχετικά απλό να καθορίσουμε ένα μικρό σύνολο τιμών εισόδου, το οποίο είναι αρκετό για να ελεγχθεί το πρόγραμμα. Σε μεγαλύτερης κλίμακας προγράμματα αυτό δεν είναι ούτε τόσο απλό ούτε πάντοτε εφικτό. Ο έλεγχος της ορθότητας ενός προγράμματος είναι εν γένει ένα πολύ σημαντικό αλλά και χρονοβόρο τμήμα, στην όλη διαδικασία παραγωγής του.

Ανακεφαλαίωση

Κύριο διακριτικό γνώρισμα του υπολογιστή είναι η δυνατότητα προγραμματισμού του, με τη λειτουργία του να καθορίζεται από το πρόγραμμα που κάθε στιγμή εκτελεί. Ο προγραμματισμός, αποτελεί έναν από τους βασικούς τομείς της επιστήμης των υπολογιστών.

Τα πρώτα προγράμματα γράφονταν στη γλώσσα μηχανής του υπολογιστή. Στη συνέχεια, για να διευκολυνθεί η διαδικασία του προγραμματισμού, σχεδιάστηκαν σταδιακά γλώσσες προγραμματισμού οι οποίες είναι πιο κοντά στον τρόπο έκφρασης τού ανθρώπου. Ένα πρόγραμμα, λοιπόν, μπορεί να είναι γραμμένο σε:

- Γλώσσα μηχανής.
- Συμβολική γλώσσα.
- Γλώσσα υψηλού επιπέδου.

Από τον πρώτο προγραμματιζόμενο υπολογιστή έως σήμερα, έχει επινοηθεί ένας πολύ μεγάλος αριθμός γλωσσών προγραμματισμού. Μερικές από τις πιο γνωστές και επιτυχημένες εμπορικά είναι η FORTRAN, η COBOL, η C και η SQL. Ανάλογα με το σκοπό για τον οποίο σχεδιάστηκαν, οι **γλώσσες προγραμματισμού** διακρίνονται σε:

- Ειδικού σκοπού, επειδή σχεδιάστηκαν για να καλύπτουν τις απαιτήσεις ενός συγκεκριμένου φάσματος προβλημάτων.
- Γενικού σκοπού, αυτές που σχεδιάστηκαν για να καλύπτουν ένα γενικότερο φάσμα προβλημάτων.

Τα προγράμματα γράφονται είτε σε συμβολικές γλώσσες είτε σε γλώσσες υψηλού επιπέδου. Τη μετατροπή τους σε γλώσσα μηχανής αναλαμβάνουν κατά περίπτωση, οι συμβολομεταφραστές, οι μεταγλωττιστές και οι διερμηνευτές.

Η συγγραφή των προγραμμάτων έχει εξελιχθεί σε μια δομημένη διαδικασία που διέπεται από τις αρχές **δομημένου προγραμματισμού**, σύμφωνα με τις οποίες ένα πρόγραμμα κατασκευάζεται από συνεργαζόμενα δομικά στοιχεία, τα οποία υλοποιούνται με βάση καθορισμένους τύπους προγραμματιστικών δομών. Οι δομές αυτές είναι:

- Η διαδοχή.
- Η επιλογή.
- Η επανάληψη.

Ανάλογα με το πώς προσεγγίζεται ο προγραμματισμός του υπολογιστή, έχουν δημιουργηθεί διάφορα πρότυπα προγραμματισμού. Αυτά είναι:

- Ο διαδικαστικός προγραμματισμός.
- Ο αντικειμενοστρεφής προγραμματισμός.
- Ο λογικός προγραμματισμός.
- Ο συναρτησιακός προγραμματισμός.

Η κατασκευή ενός προγράμματος ακολουθεί ορισμένα βήματα. Τα πιο βασικά είναι:

α) Προσδιορισμός των απαιτήσεων του προβλήματος.
β) Ανάλυση του προβλήματος.
γ) Σχεδιασμός αλγόριθμου για την επίλυση του προβλήματος.
δ) Υλοποίηση του αλγόριθμου.
ε) Έλεγχος και επαλήθευση του τελικού προγράμματος.
στ) Συντήρηση και ενημέρωση του προγράμματος.

ζ) Τεκμηρίωση.

Για να διευκολυνθεί η διαδικασία ανάπτυξης προγραμμάτων, έχουν αναπτυχθεί προγραμματιστικά περιβάλλοντα, τα οποία στην απλή τους μορφή περιλαμβάνουν απλώς ένα συντάκτη κειμένων και τα κατάλληλα μεταφραστικά προγράμματα, ενώ την πιο σύνθετη μορφή τους αποτελούν περιβάλλοντα όπως τα εργαλεία CASE ή τα περιβάλλοντα οπτικού προγραμματισμού.

Ερωτήσεις

1. Τι είναι αυτό που κάνει τον ίδιο υπολογιστή ικανό να εκτελεί εργασίες διαφορετικές μεταξύ τους; (Π.χ. να κάνει στατιστική ανάλυση, να σχεδιάζει σπίτια, να επεξεργάζεται βίντεο).

- ☐ Το υλικό
- ☐ Το λογισμικό
- ☐ Άλλο

2. «Για να χρησιμοποιηθεί ο υπολογιστής ως εργαλείο σε ένα τομέα είναι αρκετό να προμηθευτούμε τα κατάλληλα προγράμματα. Το υλικό δεν έχει καμία σημασία.» Συμφωνείτε;

- ☐ Ναι
- ☐ Όχι

3. Τι ονομάζεται πρόγραμμα υπολογιστή;

4. Τι εννοούμε λέγοντας ότι ο υπολογιστής προγραμματίζεται;

5. Τι είναι η γλώσσα μηχανής;

6. Να αναφέρετε τρεις κατηγορίες εντολών γλώσσας μηχανής.

α) . . . . . .

β) . . . . . .

γ) . . . . . .

7. Οι προγραμματισμός σε συμβολική γλώσσα είναι πιο ..... σε σχέση με τον προγραμματισμό σε γλώσσες υψηλού επιπέδου. (εύκολος / δύσκολος)

8. Αναφέρετε τρεις γλώσσες προγραμματισμού ειδικού σκοπού και τρεις γενικού.

α) . . . . . .

β) . . . . . .

γ) . . . . . .

δ) . . . . . .

ε) . . . . . .

στ) . . . .

9. Η LISP και η SMALLTALK είναι:

- ☐ Γλώσσες προγραμματισμού υψηλού επιπέδου

- ☐ Γλώσσες μηχανής του μικροεπεξεργαστή Z80
- ☐ Άλλο

10. Να περιγράψετε τα βήματα για την εκτέλεση ενός προγράμματος με τη χρήση μεταφραστή.

11. Ποιες από τις παρακάτω διαπιστώσεις είναι σωστές;

- ☐ Ο τμηματικός προγραμματισμός αποτελεί στρατηγική κατασκευής προγραμμάτων
- ☐ Ο δομημένος προγραμματισμός βασίζεται στην αρχή της λειτουργικής αποσύνθεσης
- ☐ Η υπορουτίνα είναι μια δομή επιλογής

12. Ποιος ο σκοπός των εντολών συνθήκης στο διαδικαστικό προγραμματισμό;

13. Τι είναι οι μέθοδοι και τι οι ιδιότητες στον αντικειμενοστρεφή προγραμματισμό;

14. Τα δεδομένα στον αντικειμενοστρεφή προγραμματισμό αποθηκεύονται στις μεθόδους. Συμφωνείτε;

- ☐ Ναι
- ☐ Όχι

15. Τι είναι ένα προγραμματιστικό περιβάλλον και από τι αποτελείται;

16. Ποια βήματα ακολουθούνται κατά την ανάπτυξη λογισμικού;

17. Να συνδέσετε τα στοιχεία της πρώτης στήλης με αυτά της δεύτερης.

Στάδιο	**Αποτέλεσμα**
Συγγραφή προγράμματος •	• Εκτελέσιμο πρόγραμμα
Μεταγλώττιση •	• Πηγαίο πρόγραμμα
Σύνδεση •	• Αντικείμενο πρόγραμμα

Γλωσσάριο

Αλγόριθμος	Μια βήμα προς βήμα διαδικασία η οποία οδηγεί στην επίλυση ενός προβλήματος. Πρέπει να είναι σαφής και να έχει συγκεκριμένο σημείο τερματισμού.
Αντικείμενο πρόγραμμα	Το προϊόν της μετάφρασης του πηγαίου προγράμματος από ένα μεταφραστή.
Αντικειμενοστρεφής προγραμματισμός	Μοντέλο προγραμματισμού στο οποίο τα δεδομένα και ο κώδικας είναι οργανωμένα σε αντικείμενα που επικοινωνούν μεταξύ τους με μηνύματα.
Διερμηνευτής Interpreter	Μεταφραστικό πρόγραμμα, μέσω του οποίου εκτελείται ένα πρόγραμμα που είναι γραμμένο σε γλώσσα υψηλού επιπέδου. (Δες επίσης «μεταφραστής».)
Εργαλεία ανάπτυξης λογισμικού με τη βοήθεια υπολογιστή - Case tools	Ολοκληρωμένα εργαλεία για ανάπτυξη λογισμικού με τη βοήθεια υπολογιστή.
Δομημένος προγραμματισμός	Ένα σύνολο από τεχνικές που έχουν σκοπό τη βελτίωση της ποιότητας του παραγόμενου λογισμικού.
Συντάκτης Editor	Πρόγραμμα με το οποίο γράφονται απλά κείμενα, χωρίς μορφοποίηση. Συνήθως χρησιμοποιείται για τη συγγραφή προγραμμάτων.
Λογικός προγραμματισμός	Μοντέλο προγραμματισμού το οποίο έχει τις αρχές του στη Μαθηματική Λογική.
Μεταγλωττιστής Compiler	Πρόγραμμα το οποίο μεταφράζει-μεταγλωττίζει σε γλώσσα μηχανής ένα πρόγραμμα γραμμένο σε γλώσσα υψηλού επιπέδου.
Πηγαίο πρόγραμμα	Η αρχική μορφή (μορφή κειμένου) ενός προγράμματος γραμμένου σε κάποια γλώσσα προγραμματισμού.
Πρόγραμμα	Ένα σύνολο εντολών γραμμένων σε κάποια γλώσσα προγραμματισμού, με σκοπό την εκτέλεση συγκεκριμένων εργασιών από τον υπολογιστή.
Συμβολική γλώσσα	Γλώσσα προγραμματισμού που αποτελείται από μνημονικές λέξεις, οι οποίες βρίσκονται σε άμεση (μία προς μία) αντιστοιχία με τις εντολές της γλώσσας μηχανής.
Συμβολομεταφραστής Assembler	Πρόγραμμα το οποίο μετατρέπει σε γλώσσα μηχανής ένα πρόγραμμα γραμμένο σε συμβολική γλώσσα.
Συναρτησιακός προγραμματισμός	Μοντέλο προγραμματισμού το οποίο στηρίζεται στη μαθηματική έννοια της συνάρτησης.
Συνδέτης Linker	Ειδικό πρόγραμμα που συνδέει κατάλληλα το αντικείμενο πρόγραμμα με άλλα αντικείμενα προγράμματα, ώστε αυτό να μετατραπεί σε αυτόνομο εκτελέσιμο πρόγραμμα.
Οπτικός προγραμματισμός	Ανάπτυξη προγραμμάτων με γραφικές μεθόδους.
Υπορουτίνα	Ανεξάρτητο τμήμα κώδικα, το οποίο μπορεί να κληθεί για εκτέλεση από διάφορα σημεία ενός προγράμματος. Αποτελεί βασικό εργαλείο στο δομημένο προγραμματισμό.

Ενδιαφέρουσες και χρήσιμες διευθύνσεις του Διαδικτύου

http://www.amzi.com

Εταιρεία που ειδικεύεται στην Prolog. Διαθέτει χωρίς κόστος μία καλή έκδοση της γλώσσας.

http://www.brain.uni-freiburg.de/~klaus/fpc/

Μεταφραστής Pascal για DOS και Linux.

http://www.delorie.com/djgpp/

Μεταφραστής C για DOS.

http://www.microsoft.com

Εκτός από τα γνωστά της λειτουργικά συστήματα, η Microsoft διαθέτει περιβάλλοντα ανάπτυξης για διάφορες γλώσσες όπως C, C++ και Basic.

http://www.inspise.com

Θα βρείτε πληροφορίες για τα πολύ επιτυχημένα περιβάλλοντα ανάπτυξης της Borland.

http://www.iecc.com/compilers/crenshaw/

Θα βρείτε γενικές οδηγίες για τον τρόπο κατασκευής ενός μεταγλωττιστή.

http://archie.inesc.pt/free-dir/free-toc.html

Κατάλογος από μη εμπορικούς μεταφραστές για διάφορες γλώσσες προγραμματισμού.

http://www.vlib.org/Computing.html

Γενικές πληροφορίες για τον προγραμματισμό.

http://www.pascal-central.com/

Καλό σημείο εκκίνησης για αναζήτηση πληροφοριών σχετικών με τη γλώσσα Pascal.

http://www.flamingolingo.com/programming.c/

Διαθέτει πολλές πληροφορίες για τη γλώσσα προγραμματισμού C.

http://www.nvg.ntnu.no/~sk/lang/lang.html

Εκτενής αλφαβητικός κατάλογος με γλώσσες προγραμματισμού.

Ομάδες νέων

Στις ομάδες νέων υπάρχει ένας μεγάλος αριθμός ομάδων που αφορά θέματα προγραμματισμού υ-πολογιστών.
Ενδεικτικά αναφέρουμε τις:
comp.compilers
comp.programming
*comp.lang.** *(ομάδες για γλώσσες προγραμματισμού)*
news.answers (Θα βρείτε πολύ ενδιαφέροντα FAQs)

Βιβλιογραφία

D.E Knuth, **The art of computer Programming, vols 1,2,3,** Εκδόσεις Addison-Wesley, 1969,73

Πρόκειται για ένα κλασσικό σύγγραμμα απαραίτητο για όποιον θα ήθελε να ασχοληθεί σοβαρά με τον προγραμματισμό.

Elliot B. Koffman, **Turbo Pascal, 5th Edition,** Εκδόσεις Addison-Wesley. 1998

Καλογραμμένο βιβλίο το οποίο μέσω της γλώσσας Pascal, παρουσιάζει την τεχνική της επίλυσης προβλημάτων με υπολογιστή.

Ι.Κ.Κάβουρα, **Δομημένος προγραμματισμός με Pascal,** Εκδόσεις Κλειδάριθμος

Παρουσιάζει το δομημένο προγραμματισμό μέσα από τη διδασκαλία της Pascal.

Θεόδωρου Ζ. Καλαμπούκη, **Δομές δεδομένων με Pascal,** Αθήνα 1991

Παρουσίαση των δομών δεδομένων μέσα από την Pascal.

E. Horrowitz, **Βασικές αρχές γλωσσών προγραμματισμού,** Εκδόσεις Κλειδάριθμος

Περιλαμβάνει πληθώρα θεμάτων γύρω από τις θεμελιώδεις έννοιες και αρχές των γλωσσών προγραμματισμού.