Κεφάλαιο 1

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΥΠΟΛΟΓΙΣΤΩΝ

1.1. Εισαγωγή

Ο προγραμματισμός των υπολογιστών αποτελεί το κυριότερο εργαλείο της επιστήμης των υπολογιστών για να επιλύει σύνθετα υπολογιστικά προβλήματα και για να επιτύχει αυτή την πολύπλοκη αποστολή του, εφαρμόζει ανάλογα με τη φύση του προβλήματος και διαφορετικές γλώσσες προγραμματισμού.

Λέμε *γλώσσα προγραμματισμού* (computer language) μια νέα τεχνητή γλώσσα η οποία μπορεί να χρησιμοποιηθεί για τον έλεγχο και τη λειτουργία του υπολογιστή. Μια γλώσσα προγραμματισμού ορίζεται από ένα συγκεκριμένο και αυστηρό σύνολο συντακτικών και εννοιολογικών κανόνων οι οποίοι ορίζουν τη δομή της γλώσσας.

Οι γλώσσες προγραμματισμού γενικά, διευκολύνουν την οργάνωση και τη διαχείριση των πληροφοριών.

Υπάρχουν πάρα πολλές και διαφορετικές γλώσσες προγραμματισμού οι οποίες δημιουργήθηκαν για να ικανοποιήσουν τις απαιτήσεις και τις ανάγκες των πολυάριθμων προβλημάτων των σύγχρονων επιστημών και η κατηγοριοποίηση τους δεν είναι εύκολη υπόθεση.

Όταν το ζητούμενο αποτέλεσμα θα υπολογίζεται περιγράφοντας μόνο τις επιθυμητές ιδιότητές του τότε, έχουμε την ανάγκη μιας γλώσσας της οικογένειας του *δηλωτικού προγραμματισμού* (declarative programming).

Παραδείγματα γλωσσών δηλωτικού προγραμματισμού είναι οι γλώσσες Haskell, Prolog, Lisp και HTML.

Όταν το ζητούμενο αποτέλεσμα θα υπολογίζεται αλλάζοντας την κατάσταση της μνήμης του υπολογιστή με τη βοήθεια εντολών τότε, έχουμε την ανάγκη μιας γλώσσας της οικογένειας του *προστακτικού ή επιτακτικού προγραμματισμού* (imperative programming).

Στην επιστήμη των υπολογιστών, ο προστακτικός ή επιτακτικός προγραμματισμός είναι ένα παράδειγμα προγραμματισμού που χρησιμοποιεί δηλώσεις που αλλάζουν την κατάσταση ενός προγράμματος. Με τον ίδιο τρόπο που η προστακτική διάθεση στις φυσικές γλώσσες εκφράζει εντολές, ένα πρόγραμμα του προστακτικού προγραμματισμού αποτελείται από εντολές για την εκτέλεσή τους από τον υπολογιστή.

Ο προστακτικός προγραμματισμός επικεντρώνεται στην περιγραφή του τρόπου λειτουργίας ενός προγράμματος. Ο όρος προστακτικός προγραμματισμός χρησιμοποιείται συχνά σε αντίθεση με τον δηλωτικό προγραμματισμό, ο οποίος επικεντρώνεται στο τι πρέπει να ολοκληρώσει το πρόγραμμα χωρίς να προσδιορίζει τον τρόπο με τον οποίο το πρόγραμμα πρέπει να επιτύχει το αποτέλεσμα.

Παραδείγματα γλωσσών προστακτικού προγραμματισμού είναι κυρίως οι γλώσσες προγραμματισμού: C, Fortran, Pascal κ.τ.λ.

Ο *διαδικαστικός προγραμματισμός* είναι ένας τύπος προστακτικού προγραμματισμού στον οποίο το πρόγραμμα είναι χτισμένο από μία ή περισσότερες διαδικασίες (που ονομάζονται επίσης υπορουτίνες ή συναρτήσεις). Οι όροι χρησιμοποιούνται συχνά ως συνώνυμα, αλλά η χρήση των διαδικασιών έχει δραματική επίδραση στο πώς εμφανίζονται τα προγράμματα και πώς κατασκευάζονται.

Από τη δεκαετία του 1960, ο δομημένος προγραμματισμός έχει προταθεί ως τεχνική για τη βελτίωση της διατηρησιμότητας και της συνολικής ποιότητας των προγραμμάτων. Ο δομημένος προγραμματισμός έγινε πολύ γρήγορα δημοφιλής στους προγραμματιστές αφού δημιουργήθηκε για να βελτιώσει και να συστηματοποιήσει τις τεχνικές του παλαιότερου διαδικαστικού προγραμματισμού. Έτσι, στην έννοια της διαδικασίας προστέθηκε η αναγκαιότητα της ανάλυσης ενός προγράμματος σε θεμελιώδεις διαδικασίες.

Οι έννοιες του αντικειμενοστρεφούς προγραμματισμού επιχειρούν να επεκτείνουν αυτή την προσέγγιση.Ο διαδικαστικός προγραμματισμός θα μπορούσε να θεωρηθεί ένα βήμα προς το δηλωτικό προγραμματισμό.

Η γλώσσα FORTRAN είναι η πρώτη γλώσσα προγραμματισμού υπολογιστών η οποία επέτρεψε να κωδικοποιηθεί ένας αριθμητικός υπολογισμός μ' ένα τρόπο ο οποίος να μοιάζει μ' ότι γράφουμε στα μαθηματικά.

Αυτό φυσικά, ήταν και τ' όνειρο του σχεδίου για τη γλώσσα FORTRAN, να κωδικοποιηθούν οι μαθηματικοί τύποι.

Η γλώσσα **C,** αναπτύχτηκε στις αρχές της δεκαετίας του 1970 από τον Dennis Richie στα εργαστήρια των Bell Labs προκειμένου να χρησιμοποιηθεί για την ανάπτυξη του γνωστού λειτουργικού συστήματος UNIX. Για το λόγο αυτό χρησιμοποιείται κυρίως για την ανάπτυξη και βελτίωση των λειτουργικών συστημάτων και των διαδικασιών ελέγχου των υπολογιστών. Φυσικά, μπορεί να χρησιμοποιηθεί και για πολλές άλλες εφαρμογές του *διαδικαστικού προγραμματισμού* (procedural programming).

Ο διαδικαστικός προγραμματισμός βασίζεται στην έννοια της κλήσης διαδικασιών. *Διαδικασία* (procedure) είναι ένα αυτοτελές σύνολο εντολών προς εκτέλεση. Μια διαδικασία είναι γνωστή και με τις λέξεις *ρουτίνα*, *υπορουτίνα, μέθοδος*, *συνάρτηση (function).*

Το αυτοτελές σύνολο των εντολών μιας διαδικασίας μπορούμε να το διαχειριστούμε ευκολότερα και με μεγαλύτερη ασφάλεια, από ότι το σύνολο των εντολών ενός προγράμματος. Μπορούμε επίσης, να το συντηρήσουμε πιο εύκολα και ταχύτερα και να διορθώσουμε τα ορθογραφικά και συντακτικά λάθη. Οι ενέργειες για τη διόρθωση των λαθών ενός προγράμματος είναι γνωστές με τον όρο *αποσφαλμάτωση* (debugging).

Η έννοια της διαδικασίας αποτελεί για τη γλώσσα προγραμματισμού C το βασικό συστατικό της, το οποίο επιτρέπει στους προγραμματιστές να αναπτύξουν τις εφαρμογές τους με ταχύτητα και απλότητα.

Με την ομαδοποίηση των διαδικασιών και τη δημιουργία αρχείων διαδικασιών δημιουργούνται οι *βιβλιοθήκες* (library) δηλαδή, μια συλλογή από έτοιμα υποπρογράμματα τα οποία χρησιμοποιούνται πολύ εύκολα για την ταχύτερη και ασφαλέστερη ανάπτυξη νέων εφαρμογών.

Οι βιβλιοθήκες περιέχουν επαναχρησιμοποιούμενο κώδικα και δηλώσεις οι οποίες μπορούν να χρησιμοποιηθούν άμεσα από τους προγραμματιστές μέσα στα νέα προγράμματα τα οποία αναπτύσσουν. Αυτή η τεχνική επιτρέπει το διαμοιρασμό και τη χρήση κώδικα και δεδομένων με πιο αποτελεσματικό τρόπο.

Η γλώσσα C, υιοθέτησε τις αρχές του δομημένου προγραμματισμού και προσαρμόστηκε γρήγορα στις απαιτήσεις της εξελισσόμενης επιστήμης των υπολογιστών με αποτέλεσμα να γίνει πολύ γρήγορα δημοφιλής στους προγραμματιστές των υπολογιστών αφού παρείχε όλα τα στοιχεία για ασφαλή και αποδοτικό προγραμματισμό.

Είναι μια γλώσσα προγραμματισμού γενικού σκοπού, προσφέρει οικονομία στην έκφραση, μοντέρνο έλεγχο της ροής του προγράμματος, πλήρεις δομές δεδομένων καθώς και ένα πλούσιο σύνολο τελεστών. Η απουσία περιορισμών και η γενικότητά της την κάνουν ιδιαίτερα εύχρηστη και αποτελεσματική για την υλοποίηση μεγάλου εύρους εφαρμογών.

Επιπλέον, παρουσιάζει πολλά και χαρακτηριστικά πλεονεκτήματα που αφορούν την:

- **Συντήρηση**: Τα προγράμματα είναι εύκολο να συντηρηθούν.
- **Αναγνωσιμότητα**: Τα προγράμματα διαβάζονται εύκολα.
- **Μεταφερσιμότητα**: Τα προγράμματα εύκολα μεταφέρονται σε διαφορετικά λειτουργικά συστήματα και υπολογιστές.

Τέλος, η γλώσσα C, διαθέτει όλες τις χαρακτηριστικές ιδιότητες των γλωσσών μέσου επιπέδου και με τη δύναμη της δομής της αποτελεί υπόδειγμα για τη δημιουργία πολλών άλλων νέων γλωσσών προγραμματισμού.

Η εκμάθηση της γλώσσας C κρίνεται απαραίτητη γιατί αποτελεί τη βάση πολλών σύγχρονων γλωσσών προγραμματισμού.

1.1.1. Ιστορικά στοιχεία της γλώσσας Fortran

Το όνομα FORTRAN προέρχεται από τις λέξεις **FOR**mula **TRAN**slation, και σημαίνει μετάφραση τύπων. Δημιουργήθηκε την τριετία 1954-1957 από την εταιρεία IBM υπό την επίβλεψη του John Backus.

Η πρώτη έκδοση της γλώσσας ήταν η FORTRAN I και χρησιμοποιήθηκε για πρώτη φορά τον Απρίλιο του 1957 σ' ένα υπολογιστή της IBM (Μοντέλο 704).

Την άνοιξη του 1958 μετά από μερικές αλλαγές και προσθέσεις εμφανίστηκε η FORTRAN II. Όμως η πιο διαδεδομένη και γνωστή έκδοση της FORTRAN ήταν η FORTRAN IV η οποία πρωτοπαρουσιάστηκε το 1962.

Το 1966 έγινε μια προσπάθεια από το ANSI (American National Standard Institute) και δημιουργούνται διεθνείς κανονισμοί (Standards) οι οποίοι κανονικοποιούν και ομοιογενοποιούν τις δυνατότητες της γλώσσας. Έτσι έχουμε τη FORTRAN 66 (απόφαση ANSI X3.0-1966) την οποία αποδέχονται όλοι δηλαδή, ένα πρόγραμμα γραμμένο σε γλώσσα FORTRAN 66, να μπορεί να γίνει δεκτό χωρίς καμία αλλαγή από κάθε υπολογιστή ο οποίος διαθέτει μεταγλωττιστή (Compiler) της FORTRAN.

Επόμενη έκδοση είναι η Standard FORTRAN 77 η οποία είναι σύμφωνη με την απόφαση X3.9-1978 του ANSI, καθώς και με την απόφαση 1539-1980 του ISO (International Standard Organisation).

Στις αρχές της δεκαετίας του 90, εμφανίστηκαν οι νέες προδιαγραφές της γλώσσας FORTRAN, για να την προσαρμόσουν στις νέες απαιτήσεις της επιστήμης της Πληροφορικής, γι' αυτό και η νέα έκδοση είναι γνωστή με τ' όνομα Fortran 90 (ANSI X3.198-1992 και ISO/IEC 1539-1991). Από την έκδοση αυτή, καταργούνται τα κεφαλαία γράμματα γραφής του ονόματος FORTRAN και παραμένει κεφαλαίο μόνο το πρώτο γράμμα.

Η επόμενη έκδοση της γλώσσας, με πολύ μικρές αλλαγές από την προηγούμενη, είναι η Fortran 95 (ANSI X3J3/96-007 και ISO/IEC 1539-1:1997) η οποία μετά τους απαραίτητους ελέγχους διατίθεται από το 1997.

Η Fortran 90/95 δεν διέγραψε κανένα από τα χαρακτηριστικά της FORTRAN 77, μερικά μόνο χαρακτηριστικά της FORTRAN 77 προσδιορίζονται σαν πεπαλαιωμένα (obsolescent) ή επιβλαβή (harmful).

Στη Fortran 95 κάποια χαρακτηριστικά της FORTRAN 77 δεν συμπεριλαμβάνονται πλέον επειδή θεωρήθηκαν ότι έχουν αναπληρωθεί από παρόμοιες εντολές. Τα χαρακτηριστικά αυτά είναι:

- Η εντολή **ASSIGN**
- Η εντολή **PAUSE**
- Η προδιαγραφή **H**
- Οι μεταβλητές ελέγχου της εντολής **DO** δέχονται μόνο ακέραιες τιμές.

Η έκδοση της γλώσσας Fortran 2003 (ISO/IEC 1539-1:2004) είναι προσαρμοσμένη στις απαιτήσεις του αντικειμενοστρεφούς προγραμματισμού (Object Oriented Programming) με την υποστήριξη του πολυμορφισμού (polymorphism) και της κληρονομικότητας (inheritance).

Η Fortran 2003 υποστηρίζει το ISO 10646 δηλαδή, τη χρήση των ειδικών χαρακτήρων όλων των γραπτών γλωσσών.

Επίσης, η νέα έκδοση προβλέπει:

- τυποποιημένη διαλειτουργικότητα με τη γλώσσα προγραμματισμού C
- επιλογή της υποδιαστολής αντί για την τελεία ως διαχωριστικού μεταξύ του ακέραιου και δεκαδικού μέρους ενός πραγματικού αριθμού και
- συνεργασία με το λειτουργικό σύστημα του υπολογιστή προκειμένου να χρησιμοποιεί τις μεταβλητές περιβάλλοντος (environment variables) και να έχει πρόσβαση στη γραμμή εντολών (command line arguments).

Η προτελευταία έκδοση είναι η Fortran 2008, όπου μεταξύ των νέων δυνατοτήτων της περιλαμβάνονται τα ακόλουθα:

Α. Το μέγιστο πλήθος των διαστάσεων ενός πίνακα από 7 αυξάνεται σε 15

Β. Υποστήριξη παράλληλης επεξεργασίας

Γ. Νέες μαθηματικές εσωτερικές συναρτήσεις για υπολογισμούς των συναρτήσεων Bessel και Gamma

Η τελευταία έκδοση είναι η Fortran 2018, η οποία περιλαμβάνει μικρές αναθεωρήσεις, ορισμένες διευκρινίσεις και διορθώσεις σε ασυνέπειες και μεταξύ των νέων δυνατοτήτων της έχουμε:

Α. Βελτίωση της ενσωμάτωσης των γλωσσών C και Fortran

Β. Ενίσχυση των coarrays

Επίσης, όπως συνέβη και με τις προηγούμενες εκδόσεις της Fortran, μερικά χαρακτηριστικά προτείνονται για μελλοντική απόσυρση από τις επόμενες εκδόσεις της Fortran 2018. Αυτά είναι:

- οι επαναλήψεις DO με αριθμό ετικέτας (DO 10 I = ...)
- οι εντολές *EQUIVALENCE, COMMON* και *BLOCK DATA*

Τέλος, οριστικά καταργούνται οι εντολές της Fortran:

- το αριθμητικό *IF*
- η εντολή *DO* όταν δεν τελειώνει (δεν κλείει) με μια εντολή *CONTINUE* ή *END DO*

Η Fortran 2003 καθώς και οι εκδόσεις 2008 και 2018 διατηρούν όλες τις τροποποιήσεις των εκδόσεων 90/95 και αποδέχονται πλήρως τα χαρακτηριστικά της FORTRAN 77. Δηλαδή, όλα τα παλαιά και δοκιμασμένα προγράμματα με οποιαδήποτε προηγούμενη έκδοση της FORTRAN εξακολουθούν να υποστηρίζονται από τις νεότερες εκδόσεις της Fortran.

Θ' αναρωτηθεί κανείς, γιατί η γλώσσα FORTRAN επιζεί τόσα χρόνια και μάλιστα χρησιμοποιείται πιο πολύ, από πολλές άλλες γλώσσες πιο πλούσιες, πιο δυναμικές και πιο καινούργιες;

Η κυριότερη αιτία είναι χωρίς αμφιβολία η απλότητά της. Πολλές φορές όμως πίσω από την φαινομενική απλότητα της γλώσσας κρύβονται πολλές δυσκολίες και ειδικές περιπτώσεις οι οποίες αναγκάζουν τους προγραμματιστές να καταφεύγουν συχνά σε ειδικές λύσεις έτσι ώστε η χρήση της να γίνεται εύκολη και αποδοτική.

Ένας ακόμη σοβαρός λόγος για τη διατήρηση και ανάπτυξη της γλώσσας είναι ότι τα περισσότερα ερευνητικά προγράμματα τα οποία αναπτύχθηκαν τα τελευταία 60 χρόνια, όπως επιστημονικές ή τεχνικές εφαρμογές, προγράμματα στατιστικής ή επιχειρησιακής έρευνας κ.τ.λ. είναι γραμμένα σε γλώσσα FORTRAN.

Επίσης, υπάρχουν έτοιμα και δημοσιευμένα σε διεθνή περιοδικά μικρά προγράμματα (υποπρογράμματα) σε γλώσσα FORTRAN ικανά να λύσουν κάθε πρόβλημα μαθηματικών, φυσικής, αστρονομίας, και πολλών άλλων επιστημονικών τομέων. Έτσι, ο κάθε ερευνητής, επιστήμονας, φοιτητής ή ερασιτέχνης του προγραμματισμού μπορεί να τα χρησιμοποιήσει πολύ εύκολα προσαρμόζοντάς τα στις ανάγκες του προβλήματος το οποίο τον απασχολεί.

Ένα πρόγραμμα γραμμένο σε οποιαδήποτε έκδοση της γλώσσας FORTRAN μπορεί να "τρέξει" σ' όλους τους υπολογιστές ανεξάρτητα από την εταιρία κατασκευής και του μεγέθους, κι αυτό γιατί η γλώσσα FORTRAN πέτυχε τον μεγαλύτερο βαθμό κανονικοποίησης (standardisation) από κάθε άλλη γλώσσα προγραμματισμού.

Τέλος, με την προσαρμογή της γλώσσας στις νέες τεχνικές, όπως το cluster και grid computing, η Fortran παραμένει στην πρώτη γραμμή των επιλογών ως κύριας γλώσσας προγραμματισμού από πολλά σύγχρονα ερευνητικά εργαστήρια και κυβερνητικούς οργανισμούς γεγονός που επιβάλει την ένταξη και διδασκαλία της σε σχολές μηχανικών πολλών ειδικοτητων.

Τι γίνεται όμως με κάθε νέα έκδοση της γλώσσας;

Κάθε νέα έκδοση είναι κι ένα υπερσύνολο των εντολών της προηγούμενης. Δηλαδή οι χρήστες και οι προγραμματιστές διαπιστώνουν τις ελλείψεις και τις αδυναμίες της γλώσσας και σχεδόν κάθε δεκαετία δημιουργούνται νέα πρότυπα το οποία συμπληρώνουν μεθοδικά τα προηγούμενα.

Έτσι, όλα τα προγράμματα γραμμένα σε παλαιότερες εκδόσεις της FORTRAN "τρέχουν" και με τους μεταγλωττιστές (compilers) κάθε καινούργιας έκδοσης. Πολύ απλά, δίνεται η δυνατότητα για βελτιώσεις των προγραμμάτων τόσο στη σύνταξη όσο και στην εκτέλεση μέσω της σταδιακής ανανέωσης και αντικατάστασης των παλαιών προδιαγραφών της γλώσσας οι οποίες και δεν συνιστώνται για ένα σύγχρονο και αποδοτικό προγραμματισμό.

Και μια τελευταία ερώτηση-παρατήρηση.

Γιατί πρέπει να μάθουμε τη γλώσσα προγραμματισμού FORTRAN;

Η απάντηση ίσως δεν είναι απλή, αλλά θα γίνει μια προσπάθεια να απαντηθεί η ερώτηση, η οποία τίθεται πολύ συχνά και πρέπει να γνωρίζει κάθε νέος προγραμματιστής τους λόγους για τους οποίους αξίζει να μάθει αυτή τη γλώσσα.

Είναι γνωστό ότι υπάρχουν πάρα πολλές φυσικές γλώσσες οι οποίες έχουν εξελιχθεί ανά τους αιώνες και μεταξύ αυτών υπάρχουν κάποιες οι οποίες έχουν επικρατήσει σε διάφορες χρονικές περιόδους για διάφορους κάθε φορά λόγους (εθνικούς, πολεμικούς, οικονομικούς, πολιτισμικούς κτλ.).

Τα τελευταία 60 χρόνια έχουν επίσης προταθεί και έχουν χρησιμοποιηθεί, άλλες περισσότερο και άλλες λιγότερο, πάρα πολλές γλώσσες προγραμματισμού των υπολογιστών. Μερικές γλώσσες προγραμματισμού έχουν εμφανιστεί ως ειδικές γλώσσες με περιορισμένο πεδίο εφαρμογών και άλλες ως γενικές μ' ένα αρκετά ευρύ πεδίο εφαρμογών.

Η FORTRAN είναι μια γλώσσα προσαρμοσμένη στην επίλυση επιστημονικών και τεχνολογικών προβλημάτων. Άρα είναι μια ειδική γλώσσα, αλλά μπορεί να χρησιμοποιηθεί και σε κάθε άλλο πεδίο με αυξημένες υπολογιστές ανάγκες, όπως η αριθμητική ανάλυση, η στατιστική κτλ.

Επομένως, η εκμάθηση της FORTRAN αφ' ενός επιτρέπει την εύκολη εισαγωγή στις τεχνικές του προγραμματισμού των υπολογιστών και αφ' ετέρου κάνει συμμέτοχο το γνώστη της FORTRAN στη μεγάλη οικογένεια των ειδικών οι οποίοι διαπραγματεύονται σύγχρονα επιστημονικά προβλήματα.

Επίσης, εκτιμάται ότι οι αναμενόμενες εκδόσεις της FORTRAN θα μπορούν να προσαρμοστούν ευκολότερα στους νέους τύπους υπολογιστών, όπως είναι οι υπολογιστές παράλληλης επεξεργασίας (massively parallel computers) οι οποίοι θα δώσουν την ευκαιρία για νέες έρευνες και εφαρμογές.

Συνοψίζοντας, μπορούμε να πούμε ότι η γλώσσα FORTRAN είναι αρκετά εύκολη στην εκμάθησή της, ότι είναι προσανατολισμένη στην έρευνα, ότι είναι περισσότερο αποτελεσματική στις μαθηματικές πράξεις, ότι υπάρχουν διαθέσιμες και πολλές φορές δωρεάν, βιβλιοθήκες έτοιμων προγραμμάτων για κάθε χρήση, ότι είναι εύκολη στη χρήση της και από τις πιο σταθερές γλώσσες, ότι παρέχει βελτιστοποιημένο κώδικα και πολύ χρήσιμα βοηθητικά διαγνωστικά μηνύματα κατά τις φάσεις της μεταγλώττισης (compilation) και της φόρτωσης (link), και τέλος, ότι παραμένει το πιο ισχυρό εργαλείο για τις αριθμητικές και επιστημονικές εφαρμογές.

1.1.2. Ιστορικά στοιχεία της γλώσσας C

Η γλώσσα C αναπτύχθηκε από τον Dennis Ritchie στα AT&T Bell Labs (Murray Hill, New Jersey, USA) από το 1969 μέχρι το 1973. Ονομάστηκε "γλώσσα C" επειδή πολλά χαρακτηριστικά και ιδέες προήλθαν από μια παλαιότερη γλώσσα προγραμματισμού, η οποία ονομαζόταν "B".

Το 1978, ο Dennis Ritchie και ο Brian Kernighan εκδίδουν το βιβλίο με τίτλο *"The C Programming Language"* το οποίο για πολλά χρόνια θεωρείτο ως ο ανεπίσημος ορισμός της γλώσσας. Η έκδοση της γλώσσας C του βιβλίου αυτού αναφέρεται συνήθως ως "K&R C" .

Το 1983, το American National Standards Institute (ANSI) στην προσπάθεια εναρμόνισης των διαφορετικών εκδόσεων της γλώσσας και δημιουργίας μιας έκδοσης η οποία θα επέτρεπε τη φορητότητα των προγραμμάτων της γλώσσας C, ορίζει μια επιτροπή (με κωδικό X3J11), και με στόχο την κανονικοποίηση της γλώσσας.

Το πρώτο *πρότυπο* (standard) ολοκληρώθηκε το 1989 και είναι γνωστό ως ANSI X3.159-1989 "Programming Language C".

Το **Παράρτημα ΣΤ,** στο τέλος του βιβλίου, περιέχει όλες τις οδηγίες για την τυποποιημένη έκδοση της γλώσσας C. Αυτή η έκδοση της γλώσσας C ονομάστηκε ANSI C, και αργότερα **C89** (για να διαχωρίζεται από την επόμενη έκδοση, τη **C99**).

Το 1990, το πρότυπο ANSI υιοθετήθηκε και από τον *Οργανισμό Διεθνών Προτύπων* (International Organization for Standardization (ISO)) με τον κωδικό ISO/IEC 9899:1990 και έτσι προέκυψε η τυποποιημένη γλώσσα C90. Ουσιαστικά, οι εκφράσεις "C89" και "C90" αναφέρονται στην ίδια έκδοση της γλώσσας C και με τα ίδια χαρακτηριστικά.

Το πρότυπο της γλώσσας C89 κατά τη διάρκεια της δεκαετίας του '90, βελτιώθηκε σημαντικά μετά από τις εύστοχες παρατηρήσεις των διαρκώς αυξανόμενων χρηστών της γλώσσας και τις εξελίξεις της επιστήμης των υπολογιστών. Η βελτιώσεις αυτές προετοίμασαν την έκδοση του νέου προτύπου το έτος 1999 με τον κωδικό ISO 9899:1999. Το πρότυπο αυτό αναφέρεται ως "**C99**" και υιοθετήθηκε ως πρότυπο του ANSI το Μάρτιο του 2000.

Επειδή τα πρότυπα της γλώσσας συνεχώς ενημερώνονται και βελτιώνονται, η επόμενη έκδοση ήταν η γλώσσα "**C11**" (**ISO/IEC 9899:2011).**

Το τρέχον διεθνές πρότυπο που καθορίζει τη γλώσσα προγραμματισμού C είναι το **ISO / IEC 9899: 2018** - ονομάζεται επίσης **C17** και **C18**. Αυτή η έκδοση αντιμετωπίζει πολλά ελαττώματα που αναφέρθηκαν για την έκδοση **C11** και δεν εισάγει νέα γλωσσικά χαρακτηριστικά.

Οι σύγχρονοι μεταγλωττιστές της γλώσσας C, υποστηρίζουν όλα τα χαρακτηριστικά των προτύπων **C89** και **C99** και τα περισσότερα χαρακτηριστικά των νέων προτύπων **C11** και **C18**.

Έτσι, για να εξασφαλίσουμε, προς το παρόν, τη φορητότητα (portability) στα προγράμματά μας δηλαδή, να μπορούν να μεταγλωττίζονται χωρίς αλλαγές σε όλους τους μεταγλωττιστές, θα πρέπει να επιμένουμε και να εμπιστευόμαστε περισσότερο τα πρότυπα **C89** και **C99**.

Η γλώσσα C ενέπνευσε και αποτέλεσε οδηγό για πολλές νεότερες γλώσσες προγραμματισμού και πιο ειδικά για τις βασικές γλώσσες προγραμματισμού τις προσανατολισμένες στο διαδίκτυο, όπως είναι οι γλώσσες: **C++, C# , Java, JavaScript, PHP** Κτλ.

Η καλή γνώση και εμπειρία του προγραμματισμού με τη γλώσσα C, μπορεί να βοηθήσει σημαντικά στην ομαλή μετάβαση σε όλες αυτές τις νεότερες γλώσσες προγραμματισμού.

1.2. Βασικοί ορισμοί

Πηγαίο πρόγραμμα (Source code). Λέμε *πηγαίο πρόγραμμα* ή *πηγαίο κώδικα*, το αρχικό κείμενο ενός προγράμματος δηλαδή, τις εντολές τις οποίες γράφει ο προγραμματιστής και τις οποίες μπορεί να διακρίνει και να επεξεργαστεί εύκολα κάθε χρήστης του προγράμματος στην οθόνη του υπολογιστή με τη βοήθεια ενός *διορθωτή κειμένου* (editor). Το πηγαίο πρόγραμμα είναι ένα αρχείο κειμένου (text file) το οποίο αποθηκεύεται με επέκταση το γράμμα **c** για τη γλώσσα C και επέκταση **for** για τη γλώσσα Fortran, σε κάποιο βοηθητικό αποθηκευτικό μέσο χρήστη (σκληρό δίσκο, δισκέτα, USB Flash κ.τ.λ.).

Αντικειμενικό πρόγραμμα (Object Code). Λέμε *αντικειμενικό πρόγραμμα* ή *αντικειμενικό κώδικα*, το αποτέλεσμα της μετάφρασης του πηγαίου προγράμματος σε κώδικα μηχανής. Ένα αντικειμενικό πρόγραμμα έχει επέκταση το γράμμα : **o**, ή τρία γράμματα: **obj** και διατηρεί το όνομα του πηγαίου προγράμματος. Το αντικειμενικό πρόγραμμα χρησιμοποιείται για τη δημιουργία του τελικού εκτελέσιμου προγράμματος. Π.χ. το αντικειμενικό πρόγραμμα του προγράμματος **first.c** μετά τη μεταγλώττισή του θα έχει όνομα **first.obj**, όπως και το αντικειμενικό πρόγραμμα του προγράμματος **second.for** μετά τη μεταγλώττισή του θα έχει όνομα **second.obj**.

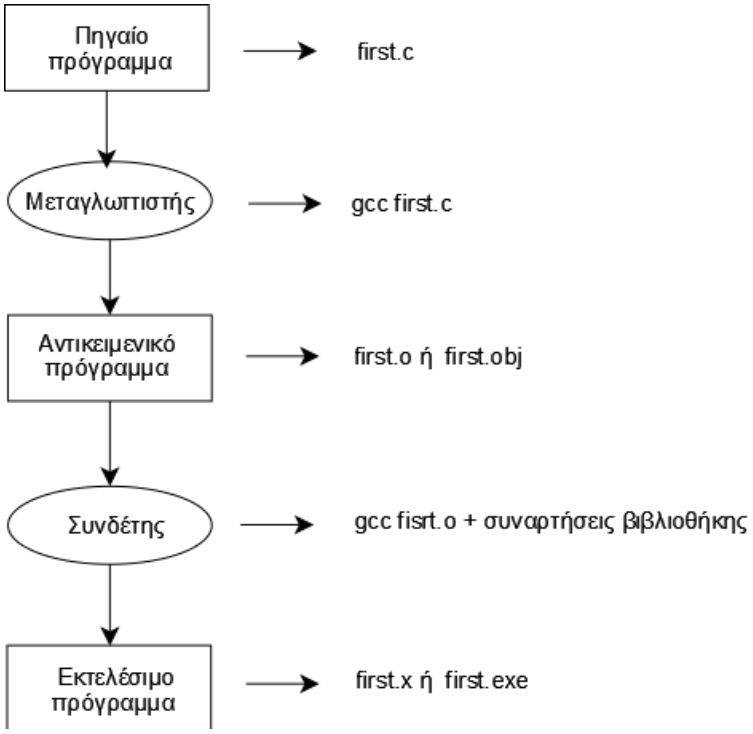
Μεταγλωττιστής (Compiler). Λέμε *μεταγλωττιστή*, το ειδικό πρόγραμμα το οποίο ελέγχει τα ορθογραφικά και συντακτικά λάθη του πηγαίου προγράμματος και ετοιμάζει το αντικειμενικό

πρόγραμμα. Για τη γλώσσα C, διατίθενται πολλές εκδόσεις προγραμμάτων μεταγλώττισης όπως: GCC, MSVC, Borland C, Watcom C. Στο λειτουργικό σύστημα των Windows, ο μεταγλωττιστής της γλώσσας C, είναι ενσωματωμένος στο περιβάλλον του Visual Studio.

Στο **Παράρτημα Δ** του βιβλίου, αναφέρονται οι πιο γνωστοί μεταγλωττιστές της γλώσσας Fortran και στο **Παράρτημα Ε** οι πιο γνωστοί μεταγλωττιστές της γλώσσας C καθώς και οι διευθύνσεις διάθεσής τους από το διαδίκτυο, έτσι ώστε να διαλέξουν οι προγραμματιστές ανάλογα με το περιβάλλον εργασίας τους τον πιο κατάλληλο για την εκμάθηση της γλώσσας και για την ανάπτυξη των προγραμμάτων τους. Μερικοί από αυτούς διατίθενται εντελών δωρεάν, υπάρχουν όμως και μεταγλωττιστές οι οποίοι διατίθενται με κάποιο χρηματικό κόστος.

Συνδέτης (linker, ή loader, ή bilder). Λέμε *συνδέτη*, ένα ειδικό βοηθητικό πρόγραμμα το οποίο αναλαμβάνει να συνδέσει ξεχωριστά μεταγλωττισμένες συναρτήσεις τις οποίες γράφει ο προγραμματιστής καθώς και τις συναρτήσεις βιβλιοθήκης σ' ένα τελικό εκτελέσιμο πρόγραμμα. Το αποτέλεσμα του συνδέτη είναι ένα αρχείο το οποίο για το λειτουργικό σύστημα των Windows διατηρεί το αρχικό όνομα του αρχείου και έχει επέκταση .**exe**, ενώ για τα συστήματα Unix και Linux έχει όνομα **a.out,** εκτός και αν οριστεί από τον προγραμματιστή ένα άλλο, διαφορετικό, όνομα.

Βιβλιοθήκη (library). Λέμε *βιβλιοθήκη*, ένα αρχείο το οποίο περιέχει μια ή και περισσότερες συναρτήσεις, οι οποίες μπορούν να χρησιμοποιηθούν σ' ένα πρόγραμμα. Συνήθως, στη γλώσσα C, έχει επέκταση το γράμμα **h**.



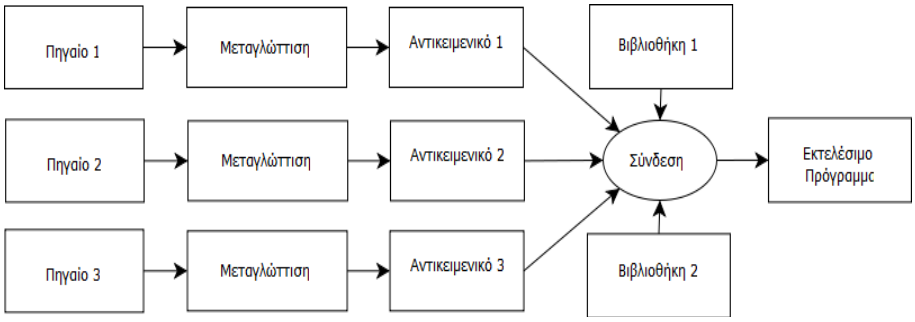
Σχήμα 1.1. Παράσταση των διαδοχικών φάσεων ενός πηγαίου προγράμματος

Χρόνος εκτέλεσης (Run time). Λέμε *χρόνο εκτέλεσης*, τη χρονική περίοδο κατά την οποία ένα πρόγραμμα εκτελείται στον υπολογιστή. Συνήθως, κατά το χρόνο εκτέλεσης λέμε ότι το πρόγραμμα "τρέχει" (run).

Αν συμβεί κατά τη διάρκεια του χρόνου εκτέλεσης του προγράμματος ένα λάθος (run time error), αυτό αποτελεί ένα σφάλμα της εκτέλεσης του προγράμματος το οποίο αναγνωρίζεται από τον υπολογιστή αλλά οφείλεται συνήθως στον προγραμματιστή π.χ. λανθασμένη απόδοση του αλγορίθμου ή λανθασμένη χρήση τελεστών. Στην περίπτωση αυτή σταματά η εκτέλεση των υπολοίπων εντολών του προγράμματος και στην οθόνη εμφανίζεται το σχετικό μήνυμα λάθους το οποίο ενημερώνει το χρήστη για την αιτία της διακοπής του προγράμματος.

Στο Σχήμα 1.1. διακρίνουμε τις τρεις διαδοχικές φάσεις τις οποίες πρέπει να "περάσει" υποχρεωτικά το πηγαίο πρόγραμμα κάθε γλώσσας προγραμματισμού (π.χ.το πρόγραμμα **first.c**) για να είναι έτοιμο να "τρέξει" δηλαδή, να εκτελέσει τον αλγόριθμο του προβλήματος για το οποίο έχει αναπτυχθεί.

Ένας τυπικός προγραμματιστής δεν ενδιαφέρεται παρά μόνο για την πρώτη φάση. Τη φάση δηλαδή, της σύνταξης του πηγαίου κώδικα του προγράμματος. Τις υπόλοιπες δύο φάσεις (μεταγλώττιση και σύνδεση) αναλαμβάνει είτε το ολοκληρωμένο περιβάλλον στο οποίο εργάζεται ο προγραμματιστής είτε το ίδιο το λειτουργικό σύστημα, για τον έλεγχο και τη δημιουργία του εκτελέσιμου προγράμματος (αρχείο **first.exe**).



Σχήμα 1.2. Σύνδεση χωριστά μεταγλωττισμένων πηγαίων προγραμμάτων

Επίσης, οι σύγχρονες γλώσσες προγραμματισμού επιτρέπουν την εύκολη διαμέλιση ενός μεγάλου σε όγκο εντολών προγράμματος σε μικρές ενότητες ανεξάρτητων προγραμμάτων τα οποία αποθηκεύονται σε διαφορετικά αρχεία και σε διαφορετικά αποθηκευτικά μέσα αλλά με τη βοήθεια του συνδέτη μπορούν να φορτωθούν όλα μαζί για το σχηματισμό του τελικού εκτελέσιμου αρχείου του προγράμματος.

Η ωφέλεια από αυτή τη διαδικασία εμφανίζεται όταν θέλουμε να τροποποιήσουμε μόνο ένα τμήμα του πηγαίου κώδικα το οποίο βρίσκεται σε ένα συγκεκριμένο αρχείο. Στην περίπτωση αυτή, δεν απαιτείται εκ νέου μεταγλώττιση όλου του κώδικα του προγράμματος παρά μόνο του αρχείου στο οποίο βρίσκεται το τροποποιημένο τμήμα του πηγαίου κώδικα.

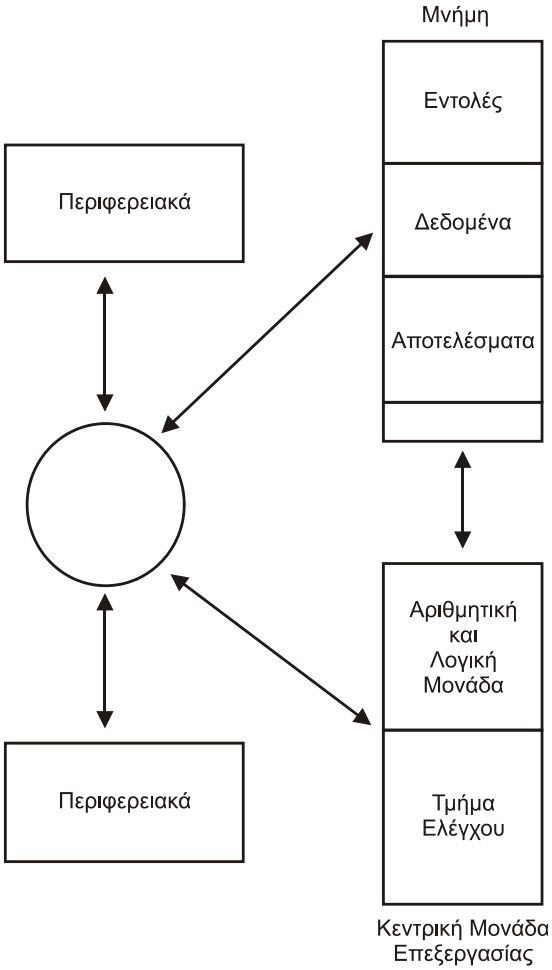
Στη συνέχεια, ο συνδέτης θα αναλάβει τη δημιουργία ενός εκτελέσιμου προγράμματος όπως χαρακτηριστικά εμφανίζεται στο σχήμα 1.2. όπου τρία διαφορετικά πηγαία αρχεία με τη βοήθεια της διαδικασίας της σύνδεσης (Linking) και δύο πρόσθετων βιβλιοθηκών δημιουργούν το τελικό εκτελέσιμο αρχείο του προγράμματος.

1.3. Δομή και λειτουργία του υπολογιστή

Κάθε υπολογιστής (Σχήμα 1.3.) αποτελείται, βασικά, από τρία μέρη:

- Τη **μνήμη** (memory)
- Την **Κεντρική Μονάδα Επεξεργασίας** – ΚΜΕ, CPU (Central Processing Unit)
- Τις **περιφερειακές μονάδες** (peripheral units)

Η μνήμη μπορεί να θεωρηθεί σαν ένα σύστημα με πολλές καταστάσεις. Η Κεντρική Μονάδα Επεξεργασίας (ΚΜΕ) είναι ένα σύνολο από ηλεκτρονικά κυκλώματα και οι περιφερειακές μονάδες χρησιμεύουν για την είσοδο, έξοδο και αποθήκευση των δεδομένων και αποτελεσμάτων της επεξεργασίας από τον υπολογιστή.



Σχήμα 1.3. Παράσταση υπολογιστή (μοντέλο Von Neumann -1947)

1.3.1. Η μνήμη

Η μνήμη ενός υπολογιστή (Μοντέλο Von Neumann) μπορεί εύκολα να παρασταθεί σαν ένα σύνολο από p φυσικά συστήματα. Το κάθε ένα από αυτά τα συστήματα έχει q στοιχεία. Κάθε φυσικό σύστημα λέγεται λέξη (word) και επομένως η μνήμη δεν είναι τίποτε άλλο παρά ένα σύνολο από p συνεχόμενες λέξεις και η κάθε λέξη έχει q στοιχεία. Οι πιο συνηθισμένες τιμές για το q είναι: 8, 16, 24, 32, 64.

Κάθε ένα στοιχείο q λέγεται "bit" (binary digit) και μπορεί να πάρει μια μεταξύ δύο καταστάσεων. Οι δύο καταστάσεις μπορούν να λέγονται αυθαίρετα είτε 0 και 1, είτε αληθές και ψευδές κτλ. Έτσι, κάθε λέξη μνήμης θα είναι ένα σύστημα από q δυαδικά στοιχεία ή ακόμη ένα σύστημα από 2^q πιθανές καταστάσεις.

Επειδή το μέγεθος των λέξεων μνήμης δεν είναι σταθερό για όλους τους υπολογιστές ούτε για τους διάφορους τύπους των μεταβλητών (ακέραιοι, δεκαδικοί κτλ), για να μπορέσουμε να

μετρήσουμε τη χωρητικότητα της μνήμης ενός υπολογιστή καταφεύγουμε σε μικρότερες μονάδες μέτρησης.

Ένα σύνολο από 8 bits αποτελεί μια θέση μνήμης όπου μπορεί να παρασταθεί ένας χαρακτήρας (αριθμός, γράμμα, σύμβολο αριθμητικής πράξης κτλ). Η θέση αυτή λέγεται "**byte**".

Μετρούμε λοιπόν, τη χωρητικότητα ενός υπολογιστή σε bytes ή K-Bytes, όπου K σημαίνει "Kilo=χίλια", αλλά εδώ το χίλια είναι λίγο διαφορετικό, ισοδυναμεί με 2^{10} =1024. Π.χ. 64K, 128K, 512K.

Κάθε λέξη μνήμης μπορεί να περιέχει είτε πληροφορίες οι οποίες παριστάνουν δεδομένα, είτε πληροφορίες οι οποίες παριστάνουν αποτελέσματα, είτε ακόμη και εντολές οι οποίες περιλαμβάνονται μεταξύ των πράξεων τις οποίες μπορεί να κάνει ένας υπολογιστής.

Κάθε φυσικό σύστημα της μνήμης (byte ή word) έχει και μια *διεύθυνση* (address). Έτσι, η μνήμη αποτελεί ένα τεράστιο χώρο, όπου κάθε φυσικό σύστημα έχει μια ακριβή θέση, η οποία αναγνωρίζεται εύκολα και γρήγορα από τη διεύθυνσή του. Το σχήμα 1.4., αποτελεί μια τυπική αναπαράσταση της μνήμης ενός υπολογιστή.

Για την αποθήκευση των δεδομένων στη μνήμη χρησιμοποιούνται flip-flops ή πυκνωτές, αλλά για την προσπέλασή της απαιτούνται και δύο καταχωρητές (register), ο MDR (Memory Data Register) δηλαδή, ο καταχωρητής δεδομένων της μνήμης και ο MAR (Memory Address Register) δηλαδή, ο καταχωρητής διευθύνσεων της μνήμης. Όταν μια λέξη πρόκειται να αποθηκευτεί στη μνήμη, πρέπει πρώτα να αποθηκευτεί στον καταχωρητή δεδομένων της μνήμης MDR και η διεύθυνση, στην οποία πρόκειται να γίνει η αποθήκευση, πρέπει να γραφεί στον καταχωρητή διευθύνσεων της μνήμης MAR.

Στη συνέχεια, τα δεδομένα μεταφέρονται από τον MDR στην αντίστοιχη λέξη της μνήμης, η διεύθυνση της οποίας είναι γραμμένη στον MAR. Η διαδικασία αυτή ονομάζεται εγγραφή στη μνήμη. Η μνήμη διατηρεί το περιεχόμενό της αναλλοίωτο, όσο χρόνο ο υπολογιστής βρίσκεται σε λειτουργία και το ανανεώνει όταν συμβεί επανεγγραφή.

Όταν πραγματοποιηθεί μια επανεγγραφή σε μια θέση της μνήμης, το προηγούμενο περιεχόμενό της εξαφανίζεται για πάντα.

Διεύθυνση μνήμης	**Περιεχόμενο**	**Επεξήγηση**
000000000000	000010010001	Λειτουργικό
000000000001	000001010010	Σύστημα
000000000010	000000110011	(Operating System)
. . . . .		
000000001001	000000010001	Εντολές του
000000001010	000000010010	προγράμματος
000000001011	000000010011	(Code segment)
000000001100	000000000000	
. . . . .		
000000010001	000010000000	
000000010010	000010000001	Δεδομένα
000000010011	000000000000	(Data segment)
000000000000	000000000000	
. . . . .		
000000000000	000000000000	Αποτελέσματα

000000000000 000000000000 (Results segment)

. . . . .

Σχήμα 1.4. Παράσταση της μνήμης ενός υπολογιστή

Η αντίστροφη διαδικασία της εγγραφής ονομάζεται ανάγνωση μιας λέξης από τη μνήμη. Κατά την ανάγνωση, το περιεχόμενο μιας λέξης, της οποίας η διεύθυνση είναι αποθηκευμένη στον MAR, μεταφέρεται στον MDR, για να μεταφερθεί στη συνέχεια στις υπόλοιπες μονάδες του υπολογιστή. Το μήκος του καταχωρητή MDR είναι όσο το μήκος της λέξης του υπολογιστή, ενώ το μήκος του καταχωρητή MAR είναι k bits, όπου 2^k είναι το μέγιστο πλήθος θέσεων της μνήμης.

1.3.2. Η Κεντρική Μονάδα Επεξεργασίας (ΚΜΕ)

Η κεντρική μονάδα επεξεργασίας ενός σύγχρονου υπολογιστή (**ΚΜΕ**) αποτελείται από δύο μέρη:

- Από το τμήμα **ελέγχου** (Control Unit)
- Από την **αριθμητική και λογική μονάδα** (Arithmetic and Logical Unit ή ALU)

Το τμήμα ελέγχου καθορίζει τη διαδοχή των εντολών μέσα στο χρόνο και συντονίζει όλες τις εργασίες του συστήματος του υπολογιστή, σύμφωνα με τις οδηγίες οι οποίες δίνονται από το χειριστή ή τον προγραμματιστή μέσω των εντολών ελέγχου.

Το τμήμα ελέγχου συνήθως, αποτελείται από δύο καταχωρητές, τους IR και PC και από τα απαραίτητα κυκλώματα για τον έλεγχο και συντονισμό της λειτουργίας του υπολογιστή. Ο καταχωρητής IR ονομάζεται καταχωρητής εντολών (Instruction Register) και είναι εκείνος ο καταχωρητής ο οποίος δέχεται τις εντολές του προγράμματος από τη μνήμη.

Οι εντολές εισέρχονται στη μνήμη σειριακά για να αναγνωρισθούν, να αναλυθούν και τέλος, να εκτελεστούν. Ο καταχωρητής PC ονομάζεται μετρητής προγράμματος (Program Counter) και είναι εκείνος ο καταχωρητής ο οποίος δίνει το περιεχόμενό του στη διεύθυνση της θέσης της μνήμης στην οποία είναι αποθηκευμένη η επόμενη εντολή. Η εντολή αυτή θα μεταφερθεί από τη μνήμη στον καταχωρητή IR.

Όταν γίνει η μεταφορά, ο μετρητής προγράμματος θα αυξηθεί αυτόματα, έτσι ώστε να δείχνει την επόμενη εντολή η οποία πρέπει να μεταφερθεί στη μνήμη. Το μήκος του καταχωρητή IR είναι ίσο με το μήκος λέξης του υπολογιστή, ενώ το μήκος του μετρητή προγράμματος ισούται με k, όπου 2^k είναι το πλήθος των διευθύνσεων της μνήμης του υπολογιστή.

Η αριθμητική και λογική μονάδα αποτελείται από ηλεκτρονικά κυκλώματα τα οποία είναι ικανά να πραγματοποιούν αριθμητικές πράξεις, λογικές πράξεις (συγκρίσεις μεταξύ δύο ή περισσοτέρων μεγεθών), αποθήκευση δεδομένων, καθώς και διάφορες άλλες βοηθητικές λειτουργίες.

Για την εκτέλεση όλων αυτών των πράξεων πρέπει να μεταφέρονται δεδομένα προς και από τη μνήμη. Για το λόγο αυτό, υπάρχουν οι κατάλληλες συνδέσεις μεταξύ της μνήμης και της αριθμητικής μονάδας.

1.3.3. Τα περιφερειακά

Για να επικοινωνήσουμε μ' ένα υπολογιστή κατά τέτοιο τρόπο ώστε η εξωτερική μορφή των δεδομένων να μετατρέπεται σε μια εσωτερική μορφή επεξεργάσιμη στη μνήμη του υπολογιστή

και η εσωτερική μορφή των αποτελεσμάτων της επεξεργασίας να μετατρέπεται σε μια εξωτερική μορφή χρειαζόμαστε τα περιφερειακά όργανα.

Στη συνέχεια, θα εννοούμε «εσωτερική μορφή» τη μορφή εκείνη η οποία μπορεί να επεξεργαστεί από τον υπολογιστή και «εξωτερική μορφή» τη μορφή η οποία είναι κατανοητή από τον άνθρωπο.

Αυτή η μετατροπή γίνεται με τη βοήθεια εξωτερικών συσκευών οι οποίες λέγονται περιφερειακά (peripherals) και οι οποίες εξασφαλίζουν την επικοινωνία μεταξύ της μηχανής και του εξωτερικού κόσμου. Περιφερειακές συσκευές ενός υπολογιστή είναι η οθόνη, το πληκτρολόγιο, οι εκτυπωτές κτλ.

Οι περιφερειακές συσκευές δέχονται τις πληροφορίες οι οποίες παρέχονται από το χρήστη του υπολογιστή και δημοσιοποιούν τα αποτελέσματα της επεξεργασίας.

Όλες οι περιφερειακές συσκευές περιέχουν έναν καταχωρητή δεδομένων Χ, ο οποίος δέχεται τις λέξεις οι οποίες προορίζονται από και προς την κεντρική μονάδα επεξεργασίας. Επίσης, περιέχουν ένα δεύτερο καταχωρητή Υ, ο οποίος περιέχει τη διεύθυνση του περιφερειακού από ή προς το οποίο θα γίνει η μεταφορά της λέξης. Το μήκος του καταχωρητή Χ είναι μικρότερο ή ίσο με το μήκος των λέξεων του υπολογιστή και το μήκος του καταχωρητή Υ εξαρτάται από τον αριθμό των περιφερειακών μονάδων τις οποίες μπορεί να υποστηρίξει ο υπολογιστής.

Οι περιφερειακές μονάδες γενικά, έχουν μικρή ταχύτητα στην επικοινωνία τους με τους υπολογιστές, αρκετά μικρότερη από την ταχύτητα με την οποία μπορεί να γίνει η επεξεργασία των αντίστοιχων δεδομένων από την ΚΜΕ. Για να επιτευχθεί ο συγχρονισμός της διαφοράς ταχύτητας μεταξύ της ΚΜΕ και των περιφερειακών μονάδων, προστίθενται μέσα στη μονάδα εισόδου-εξόδου τα κατάλληλα ηλεκτρονικά κυκλώματα (flip-flops) για κάθε περιφερειακό, τα οποία καθορίζουν αν έχει τελειώσει ή όχι η μεταφορά μιας λέξης από ή προς τον καταχωρητή Χ.

1.3.4. Η λειτουργία του Υπολογιστή

Κάνοντας μια μικρή ανακεφαλαίωση μπορούμε να πούμε ότι ένας υπολογιστής είναι μια μηχανή, σαν όλες τις άλλες μηχανές τις οποίες επινόησε και χρησιμοποιεί ο σύγχρονος άνθρωπος, η οποία διαθέτει μια «**κεντρική μνήμη**» η οποία αποτελείται από πάρα πολλές «**λέξεις**» μέσα στις οποίες μπορούμε να «**αποθηκεύσουμε**» δεδομένα και πληροφορίες σε δυαδική μορφή. Τα «αποθηκευμένα» στοιχεία, μπορεί να είναι αποτέλεσμα διαφορετικών κωδικοποιήσεων όπως:

- Εντολών, οι οποίες θα εκτελεστούν κατά τη διάρκεια του προγράμματος
- Χαρακτήρων, ενός κειμένου
- Αριθμών, κάθε φύσεως (πραγματικών, μιγαδικών, ακεραίων κ.λπ.)

Οι εντολές αποθηκεύονται σε διαδοχικές λέξεις της μνήμης, σύμφωνα με τη σειρά εκτέλεσής τους η οποία προβλέπεται από το πρόγραμμα. Κάθε εντολή, πρέπει να συμπληρώσει δύο φάσεις, τις οποίες ονομάζουμε φάση ανάκλησης (fetch phase) και φάση εκτέλεσης (execution phase). Οι φάσεις αυτές συχνά αναφέρονται και σαν κύκλοι, δηλαδή κύκλος ανάκλησης (fetch cycle) και κύκλος εκτέλεσης (execution cycle). Κατά τη φάση ανάκλησης, η εντολή, της οποίας η διεύθυνση βρίσκεται στο μετρητή προγράμματος PC, μεταφέρεται από τη μνήμη στον MDR και ύστερα στον καταχωρητή εντολών IR της μονάδας ελέγχου. Όταν ολοκληρωθεί η μεταφορά της εντολής στον IR, ο μετρητής προγράμματος PC αυξάνεται αυτόματα, ώστε να προετοιμαστεί η μεταφορά της επόμενης εντολής. Όταν κάθε μια εντολή καταλαμβάνει μία θέση μνήμης, ο μετρητής προγράμματος αυξάνεται κάθε φορά κατά 1.

Όταν μια εντολή καταλαμβάνει περισσότερες θέσεις μνήμης, ο μετρητής προγράμματος αυξάνεται κάθε φορά κατά το πλήθος των θέσεων μνήμης τις οποίες απασχολεί μια λέξη μνήμης. Μετά το πέρας της φάσης ανάκλησης ξεκινά η φάση εκτέλεσης. Κατά τη φάση αυτή, η

εντολή οι οποία βρίσκεται στον καταχωρητή εντολών, αποκωδικοποιείται και αναλύεται στις επιμέρους εργασίες οι οποίες πρέπει να εκτελεστούν.

Με την ολοκλήρωση της φάσης εκτέλεσης, έχουμε ένα νέο κύκλο ανάκλησης - εκτέλεσης για μια νέα εντολή. Κατά τον κύκλο αυτό, πρώτα θα γίνει η μεταφορά από τη μνήμη της επόμενης εντολής, αφού κατά τον προηγούμενο κύκλο είχε αυξηθεί κατάλληλα το περιεχόμενο του μετρητή προγράμματος.

Στη συνέχεια, θα ακολουθήσει η εκτέλεση της νέας εντολής, οπότε συμπληρώνεται ο νέος κύκλος. Οι κύκλοι των δύο φάσεων (ανάκλησης – εκτέλεσης) θα επαναλαμβάνονται συνέχεια, μέχρι να εμφανισθεί και να εκτελεστεί μία συγκεκριμένη εντολή, με την οποία διακόπτεται η λειτουργία των δύο φάσεων στον υπολογιστή.

1.3.5. Hardware και Sofrware

Κάθε μια στοιχειώδης πράξη σ' έναν υπολογιστή υλοποιείται με τη βοήθεια ηλεκτρονικών κυκλωμάτων. Ακόμη με τη βοήθεια των ηλεκτρονικών κυκλωμάτων γίνονται όλες οι ανταλλαγές των πληροφοριών με τον εξωτερικό κόσμο.

Το σύνολο όλων αυτών των ηλεκτρονικών κυκλωμάτων αποτελεί το "**HARDWARE**". Εμείς μπορούμε να το λέμε «το μηχανικό μέρος ενός υπολογιστή» ή το **υλικό**. Για να προγραμματίσει κανείς σ' αυτό το επίπεδο χρειάζεται πολύ χρόνο και δύσκολα αποφεύγονται τα λάθη γιατί οι πληροφορίες βρίσκονται σε δυαδική μορφή.

Έτσι, ένας υπολογιστής, για να μπορεί να αποτελεί ένα ολοκληρωμένο και λειτουργικό σύστημα, θεωρείται ότι συνοδεύεται από το "**SOFTWARE**". Εμείς πάλι περιφραστικά, μπορούμε να το λέμε «το προγραμματισμένο μέρος ενός υπολογιστή» ή το **λογισμικό**.

Για τους χρήστες των υπολογιστών η διαφορά μεταξύ του υλικού και του λογισμικού δεν είναι φανερή. Συνήθως, δεν γνωρίζουμε αν αυτή ή κάποια άλλη πράξη γίνεται από το υλικό ή από το λογισμικό.

Εκείνο το οποίο πρέπει να γνωρίζουμε είναι ότι το λογισμικό συμπληρώνει τα κενά του υλικού.

Το λογισμικό ενός υπολογιστή περιλαμβάνει:

- Το **λειτουργικό σύστημα** (Operating System).

Είναι το σύστημα το οποίο ελέγχει και εποπτεύει την ομαλή λειτουργία του υπολογιστή. Ρυθμίζει τη σειρά εκτέλεσης των διαφόρων προγραμμάτων και συντονίζει τη λειτουργία και επικοινωνία των περιφερειακών με τον υπολογιστή (είσοδος-έξοδος).

- Τα προγράμματα μεταγλώττισης

Διακρίνουμε δύο κύριες κατηγορίες προγραμμάτων μεταγλώττισης:

1. Assemblers
2. Compilers ή Interpreters

Πρόγραμμα "**assembler**" είναι ο μεταφραστής ενός προγράμματος από τη «μητρική γλώσσα» του υπολογιστή (assembly language) στη γλώσσα μηχανής (machine language). Η "assembly language" είναι η βασική γλώσσα την οποία υποστηρίζει ο επεξεργαστής του υπολογιστή γι αυτό και λέγεται μητρική γλώσσα. Στα ελληνικά, ο όρος assembler συνήθως αποδίδεται με τον όρο **συμβολομεταφραστής**.

Ένα πρόγραμμα "**Compile**r" ή "**Interpreter**" είναι ο μεταφραστής από μια γλώσσα ανεξάρτητη από τον υπολογιστή στη γλώσσα μηχανής. Μια γλώσσα θεωρείται ανεξάρτητη όταν

μπορεί να χρησιμοποιηθεί σε διαφορετικούς υπολογιστές. Τέτοιες γλώσσες είναι η FORTRAN, η ALGOL, η COBOL, η C, η BASIC κτλ.

Στα ελληνικά, ο όρος **Compiler** συνήθως αποδίδεται με τον όρο **μεταγλωττιστής** ενώ ό όρος **Interpreter** με τον όρο **διερμηνευτής**.

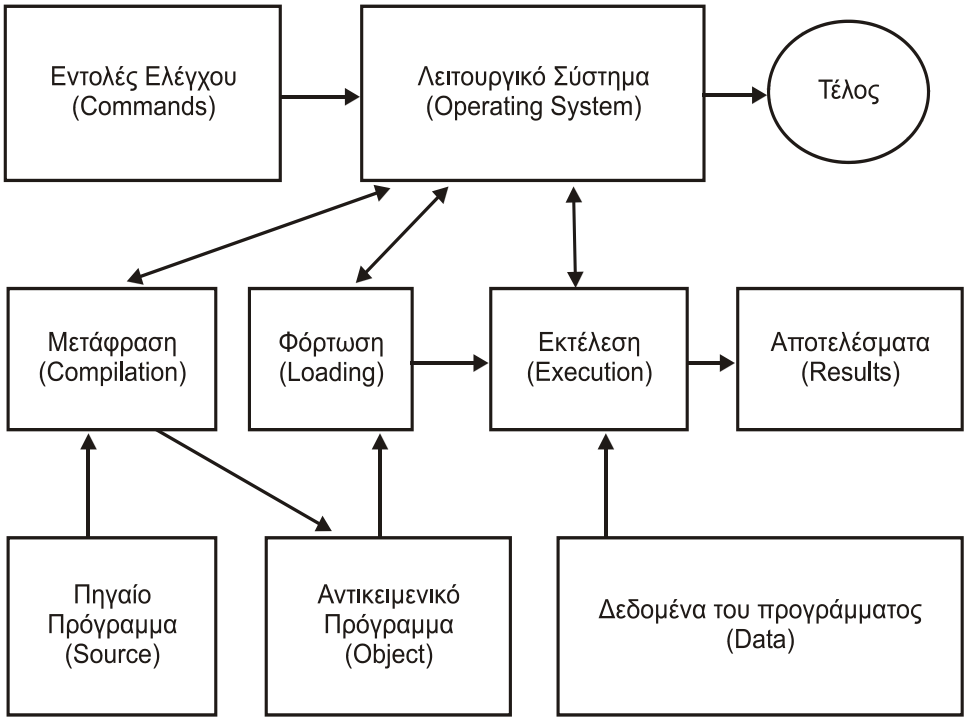
Οι "assembly languages" χαρακτηρίζονται σαν γλώσσες χαμηλού επιπέδου (low level languages), ενώ οι ανεξάρτητες της μηχανής χαρακτηρίζονται σαν γλώσσες υψηλού επιπέδου ή ανεπτυγμένες (high level languages).

1.3.6. Η εκτέλεση ενός προγράμματος

Ένα πρόγραμμα το οποίο έχει γραφτεί σε μια συμβολική γλώσσα προγραμματισμού, το **πηγαίο πρόγραμμα** (Source program), περιέχει τις αρχικές και πλέον βασικές πληροφορίες οι οποίες είναι απαραίτητες για να ξεκινήσει η επεξεργασία του από τον υπολογιστή.

Το αποτέλεσμα του ελέγχου ενός τέτοιου πηγαίου προγράμματος από τον υπολογιστή με τη βοήθεια του μεταφραστή ή όπως αλλιώς λέγεται μεταγλωττιστή (compiler) είναι ένα πρόγραμμα σε γλώσσα μηχανής και λέγεται **αντικειμενικό πρόγραμμα** (Object program ή object code).

Το αντικειμενικό πρόγραμμα δεν μπορεί από μόνο του να ξεκινήσει την εκτέλεση των πράξεων του πηγαίου προγράμματος, αλλά αποτελεί το ενδιάμεσο στάδιο στη διαδικασία του ελέγχου για την έναρξη της εκτέλεσης των πράξεων ενός προγράμματος από τον υπολογιστή. Όταν δεν υπάρχει κανένα ορθογραφικό ή συντακτικό λάθος στο πηγαίο πρόγραμμα τότε και μόνο τότε, το αντικειμενικό πρόγραμμα "φορτώνεται" στη μνήμη του υπολογιστή από ένα άλλο πρόγραμμα το οποίο λέγεται "**φορτωτής** " LOADER " (Load program) ή "LINKER".



Σχήμα 1.5. Διάγραμμα προετοιμασίας και εκτέλεσης ενός προγράμματος

Όταν δεν βρεθεί κανένα λάθος το οποίο μπορεί να προκαλέσει διακοπή της φόρτωσης του προγράμματος στη μνήμη τότε, δημιουργείται από τον λειτουργικό σύστημα του υπολογιστή το **εκτελέσιμο πρόγραμμα** (executable) με το οποίο μπορεί να ξεκινήσει η εκτέλεση των πράξεων

του πηγαίου προγράμματος και το οποίο θα προκαλέσει την εμφάνιση ή εκτύπωση των αποτελεσμάτων.

Το σχήμα 1.5. εμφανίζει συνοπτικά τα διάφορα βοηθητικά προγράμματα τα οποία θα λάβουν μέρος κατά τη διάρκεια της προετοιμασίας για την εκτέλεση ενός πηγαίου προγράμματος, καθώς και τις σχέσεις οι οποίες υφίστανται μεταξύ των δεδομένων, των προγραμμάτων και των αποτελεσμάτων.

Εντολές ελέγχου, λέγονται οι εντολές τις οποίες δίνει ο προγραμματιστής για να υποδείξει στο λειτουργικό σύστημα του υπολογιστή (operating system) πότε θ' αρχίσει και πότε θα τελειώσει κάθε μια από τις τρεις φάσεις δηλαδή, η μετάφραση, η φόρτωση και η εκτέλεση.

Όταν το πηγαίο πρόγραμμα δεν είναι σύμφωνο με τους αυστηρούς κανόνες κάθε γλώσσας προγραμματισμού (δεν επιτρέπεται ούτε ένα συντακτικό ή ορθογραφικό λάθος) τότε, το πρόγραμμα μετάφρασης (μεταγλώττισης) στέλνει ένα διαγνωστικό μήνυμα λάθους (error message) το οποίο εμφανίζεται στην οθόνη. Εκεί σταματά ο υπολογιστής την περαιτέρω εξέλιξη και ελέγχου του προγράμματος και πρέπει να διορθωθούν προσεκτικά όλα τα λάθη τα οποία έχουν σημειωθεί και να δοκιμάσουμε από την αρχή τον έλεγχο του πηγαίου προγράμματος. Το ίδιο συμβαίνει και κατά την εκτέλεση του φορτωτή.

Δηλαδή, εάν υπάρχει έστω κι ένα λάθος π.χ. δεν ορίζεται σωστά ή απουσιάζει η χρήση μιας βιβλιοθήκης υποπρογραμμάτων τότε, εμφανίζεται ένα σχετικό μήνυμα λάθους και πρέπει πρώτα να διορθωθεί αυτό το λάθος και να δοκιμάσουμε από την αρχή τον έλεγχο.

Διαπιστώνουμε πολύ εύκολα, ότι το λειτουργικό σύστημα προωθεί το αρχικό πηγαίο πρόγραμμα από τη μία φάση στην άλλη όχι αυτόματα, αλλά μόνον όταν δεν υπάρχουν διαγνωστικά λαθών. Δηλαδή, η φάση μεταγλώττισης είναι εντελώς ανεξάρτητη από τη φάση της φόρτωσης και αυτή με τη σειρά της από τη φάση της εκτέλεσης.

Πίνακας 1.1. Εκτυπώσιμοι χαρακτήρες του κώδικα ASCII

Χαρακτήρας	***Τιμή***	***Χαρακτήρας***	***Τιμή***	***Χαρακτήρας***	***Τιμή***
	32	@	64	`	96
!	33	A	65	a	97
"	34	B	66	b	98
#	35	C	67	c	99
$	36	D	68	d	100
%	37	E	69	e	101
&	38	F	70	f	102
'	39	G	71	g	103
(	40	H	72	h	104
)	41	I	73	i	105

*	42	J	74	j	106
+	43	K	75	k	107
,	44	L	76	l	108
-	45	M	77	m	109
.	46	N	78	n	110
/	47	O	79	o	111
0	48	P	80	p	112
1	49	Q	81	q	113
2	50	R	82	r	114
3	51	S	83	s	115
4	52	T	84	t	116
5	53	U	85	u	117
6	54	V	86	v	118
7	55	W	87	w	119
8	56	X	88	x	120
9	57	Y	89	y	121
:	58	Z	90	z	122
;	59	[	91	{	123
<	60	\	92	|	124
=	61	]	93	}	125
>	62	^	94	~	126
?	63	_	95	DEL	127

1.4. Κωδικοποίηση χαρακτήρων

Οι χαρακτήρες (γράμματα, αριθμοί, σύμβολα πράξεων και στίξης) οι οποίοι χρησιμοποιούνται στους υπολογιστές ως περιεχόμενο μιας μεταβλητής χαρακτήρα ή μιας σειράς χαρακτήρων (string) είναι χαρακτήρες του τυπικού κώδικα **ASCII** (American Standard Code for Information Interchange).

Ο τυπικός κώδικας ASCII περιέχει 128 χαρακτήρες και κωδικοποιείται σε 7 bits. Οι 32 πρώτοι χαρακτήρες δεν είναι εκτυπώσιμοι αλλά είναι χαρακτήρες ελέγχου του υπολογιστή. Π.χ. το πλήκτρο **ESC** αντιστοιχεί στη θέση 27 του πίνακα **ASCII.**

Οι εκτυπώσιμοι χαρακτήρες του τυπικού κώδικα ASCII εμφανίζονται στον Πίνακα 1.1. και είναι αποδεκτοί και αναγνωρίσιμοι από όλα τα σύγχρονα λειτουργικά συστήματα των υπολογιστών.

Με τη διάδοση της πληροφορικής και των εφαρμογών της σε όλο το πλανήτη, δημιουργήθηκαν πολλές και διαφορετικές επεκτάσεις του κώδικα ASCII (Extended ASCII) ο οποίος κωδικοποιείται σε 8 bits και στον οποίο περιέχονται και χαρακτήρες άλλων γραπτών γλωσσών όπως είναι οι ελληνικοί χαρακτήρες. Δυστυχώς, ο επεκτάσιμος κώδικας ASCII (Extended ASCII) δεν είναι τυποποιημένος ομοιόμορφα σε όλες τις χώρες όπου και εφαρμόζεται.

Οι υπολογιστές αποθηκεύουν τους χαρακτήρες αντιστοιχώντας στο καθένα τους από ένα διαφορετικό και μοναδικό αριθμό. Αυτή η αντιστοιχία ονομάζεται *κωδικοσελίδα*. Λόγω του περιορισμού του μεγέθους των κωδικοσελίδων σε καμία κωδικοσελίδα δεν είναι δυνατόν να υπάρχει αντιστοιχία όλων των γνωστών χαρακτήρων των υπαρχουσών γραπτών γλωσσών. Ακόμα και για μία και μόνη γλώσσα, όπως π.χ. τα Αγγλικά, μία κωδικοσελίδα δεν επαρκούσε για να καλύψει όλα τα γράμματα (Κεφαλαία, πεζά), τα σημεία στίξης και τα τεχνικά σύμβολα ευρείας χρήσης.

Εκτός του μεγέθους, οι κωδικοσελίδες διαφωνούν ριζικά μεταξύ τους. Έτσι, δύο κωδικοσελίδες μπορεί να χρησιμοποιούν τον ίδιο αριθμό για δύο *διαφορετικούς* χαρακτήρες, ή να χρησιμοποιούν διαφορετικούς αριθμούς για τον *ίδιο* χαρακτήρα.

Οι πιο βασικές κωδικοσελίδες οι οποίες χρησιμοποιούνται από όλους τους υπολογιστές στην Ελλάδα είναι:

- **DOS 437/737** για το λειτουργικό σύστημα MS-DOS
- **Windows 1253** για όλες τις εκδόσεις των Windows
- **ISO 8859-7** γνωστή και ως ΕΛΟΤ 928

Και οι τρεις κωδικοσελίδες για τις 128 πρώτες θέσεις (7 bits) χρησιμοποιούν τον κώδικα ASCIII γι αυτό και συμφωνούν απολύτως. Τις επόμενες 128 θέσεις δεν τις χρησιμοποιούν με τον ίδιο τρόπο. Για το λόγο αυτό το ίδιο πρόγραμμα όταν περιέχει ελληνικά γράμματα στις εντολές εξόδου, εμφανίζει με διαφορετικό τρόπο τα αποτελέσματα, ανάλογα με το περιβάλλον εκτέλεσης του προγράμματος.

Μια εναλλακτική πρόταση για την επίλυση αυτού του προβλήματος αποτελεί η χρησιμοποίηση των greeklish δηλαδή, η χρήση λατινικών γραμμάτων αντί για τα ελληνικά γράμματα. Με τον τρόπο αυτό μπορούμε να χρησιμοποιήσουμε στις εντολές εξόδου τα greeklish έτσι ώστε ανεξάρτητα του περιβάλλοντος εργασίας τα μηνύματα προς το χρήστη

καθώς και όλες οι εμφανιζόμενες πληροφορίες οι οποίες θα περιέχουν ελληνικά γράμματα να είναι εν μέρει κατανοητές.

Π.χ. για την πρόταση:

Αποτελέσματα του προγράμματος

μπορούμε να γράψουμε :

Apotelesmata tou programmatos

Τα τελευταία χρόνια το **Unicode**, αλλάζει αυτή την κατάσταση γιατί κάνει χρήση 16 bits για την αναπαράσταση κάθε χαρακτήρα (2^{16} = 65536 χαρακτήρες).

Το **Unicode** παρέχει έναν μοναδικό αριθμό για κάθε γνωστό χαρακτήρα, ανεξάρτητα από το λειτουργικό σύστημα, ανεξάρτητα από το λογισμικό, ανεξάρτητα από τη γλώσσα και είναι η επίσημη μέθοδος εφαρμογής της τυποποίησης των χαρακτήρων **ISO/IEC 10646**.

Το **Unicode** υποστηρίζεται πλέον από τα λειτουργικά συστήματα και όλους τους σύγχρονους φυλλομετρητές του Διαδικτύου (browsers) αλλά δυστυχώς, δεν υποστηρίζεται ακόμη αυτόματα από όλα τα περιβάλλοντα ανάπτυξης και εκτέλεσης προγραμμάτων.

Σε όλες τις εκδόσεις του λειτουργικού συστήματος των **Windows** της **Microsoft** εφαρμόζεται η κωδικοσελίδα **Windows 1253,** ενώ στο προηγούμενο λειτουργικό σύστημα της Microsoft το **MS-DOS,** χρησιμοποιείται η κωδικοσελίδα **437/737**.

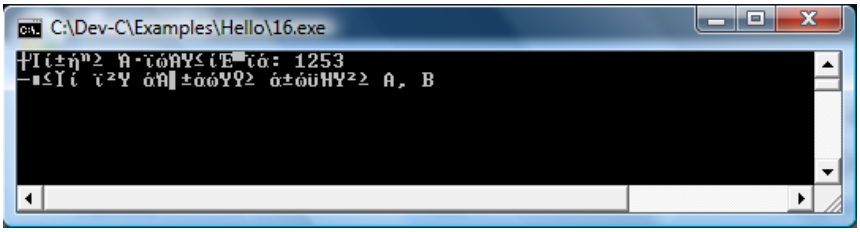
Έτσι, ότι παράγεται ή εκτελείται στο γραφικό περιβάλλον των **Windows** ακολουθεί τη κωδικοσελίδα **Windows 1253,** ενώ ότι εκτελείται στο μη γραφικό περιβάλλον του **MS-DOS (**μαύρη οθόνη)**,** ακολουθεί τη κωδικοσελίδα **437/737.**

Όταν δοκιμάσουμε να τρέξουμε τον εκτελέσιμο κώδικα ενός προγράμματος το οποίο περιέχει μηνύματα εξόδου με ελληνικούς χαρακτήρες, είτε από το ολοκληρωμένο περιβάλλον δημιουργίας του, είτε το τελικό αρχείο με επέκταση **.exe** στο οποίο θα κάνουμε διπλό κλικ είτε ακόμη, από τη γραμμή εντολών (command mode) γράφοντας το πλήρες όνομα του εκτελέσιμου αρχείου τότε, στην οθόνη του υπολογιστή θα διαπιστώσουμε ότι η εμφάνιση των μηνυμάτων δεν είναι αναγνωρίσιμη και αναγνώσιμη.

Η αιτία αυτής της αποτυχίας εμφάνισης των αποτελεσμάτων σε αναγνώσιμη μορφή οφείλεται στη διαφορετική κωδικοσελίδα η οποία χρησιμοποιείται στα δύο διαφορετικά περιβάλλοντα: αφ' ενός της ανάπτυξης του κώδικα στο γραφικό περιβάλλον των **Windows** όπου ισχύει η κωδικοσελίδα **Windows 1253** και αφ' ετέρου του περιβάλλοντος εκτέλεσης του προγράμματος, το οποίο εκτελείται στο μη γραφικό περιβάλλον του **MS-DOS (**μαύρη οθόνη) και ακολουθεί τη κωδικοσελίδα **437/737.**

Για τη θεραπεία του προβλήματος της διαφορετικής εμφάνισης των ελληνικών γραμμάτων στη μαύρη οθόνη, κατά την εκτέλεση ενός προγράμματος, προτείνεται η αλλαγή της κωδικοσελίδας.

Έτσι, αντί για τη κωδικοσελίδα **437/737**, με την οποία λειτουργεί το **MS-DOS,** να ζητήσουμε την αλλαγή της σε **Windows 1253,** με ταυτόχρονη αλλαγή της γραμματοσειράς εμφάνισης των αποτελεσμάτων στην οθόνη του υπολογιστή (font). Ο συνδυασμός αυτών των δύο αλλαγών επιτρέπει τη σωστή εμφάνιση ταυτόχρονα Λατινικών και Ελληνικών χαρακτήρων στα μηνύματα εξόδου του προγράμματος.



Εικόνα 1.1. Έναρξη εκτέλεσης ενός Περιγράμματος

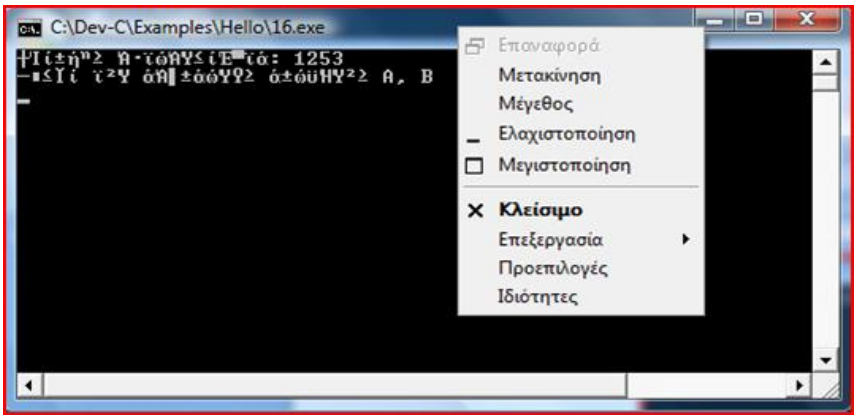
Η αλλαγή της κωδικοσελίδας επιτυγχάνεται με την κλήση της συνάρτησης του λειτουργικού συστήματος του υπολογιστή **system()** και παράμετρο το **"chcp 1253".**

Επομένως, για την αλλαγή (**CH**ange **C**ode **P**age) της κωδικοσελίδας εμφάνισης των αποτελεσμάτων σε **Windows 1253** πρέπει να προσθέσουμε στην αρχή του πηγαίου κώδικα την εντολή:

system("chcp 1253");

Η αλλαγή της γραμματοσειράς εμφάνισης των αποτελεσμάτων στην οθόνη επιτυγχάνεται μετά την έναρξη εκτέλεσης του προγράμματος επεμβαίνοντας στην οθόνη, όπου εμφανίζονται τα αποτελέσματα.

Αν ζητήσουμε την εκτέλεσή ενός προγράμματος, στην οθόνη θα εμφανιστεί έστω η εικόνα 1.1. όπου δεν είναι δυνατόν να αναγνωρίσουμε το μηνύματα που εμφανίζονται.

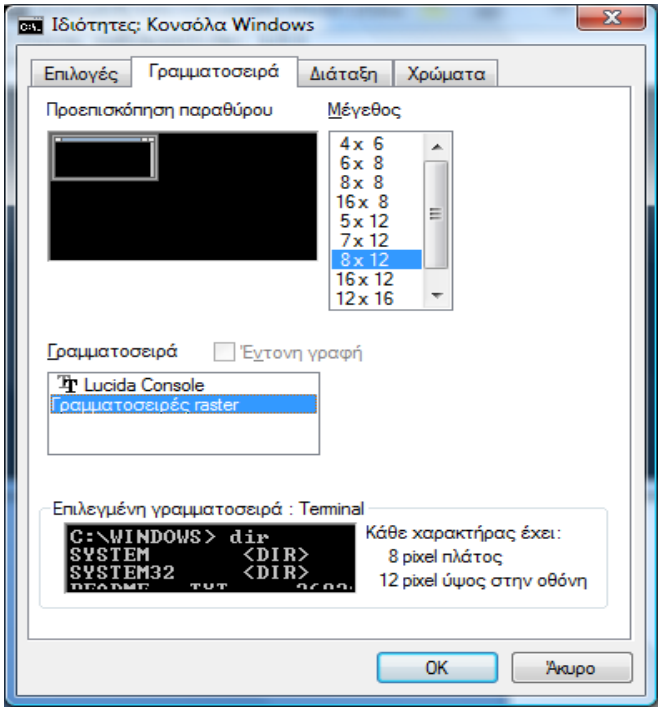


Εικόνα 1.2. Εμφάνιση του αναδυόμενου παραθύρου

Θα πρέπει να οδηγήσουμε το δείκτη του ποντικού στη γραμμή κατάστασης του προγράμματος (πάνω από τη μαύρη οθόνη) και να πατήσουμε δεξί κλικ στο ποντίκι. Το αναδυόμενο παράθυρο που θα εμφανιστεί είναι αυτό της εικόνας 1.2.

Επιλέγουμε την τελευταία γραμμή (κάνοντας απλό αριστερό κλικ πάνω στην ένδειξη **Ιδιότητες)** και αμέσως εμφανίζεται η εικόνα 1.3., στην οποία θα πρέπει να επιλέξουμε την κατάλληλη γραμματοσειρά η οποία στην προκειμένη περίπτωση είναι η **Lucida Console.**

Με την επιλογή της γραμματοσειράς (με το απλό αριστερό κλικ πάνω στο κείμενο **Lucida Console)** εμφανίζεται η εικόνα 1.4. όπου παρατηρούμε ότι έχουμε και αλλαγή στο μέγεθος της εμφάνισης των χαρακτήρων στην οθόνη.

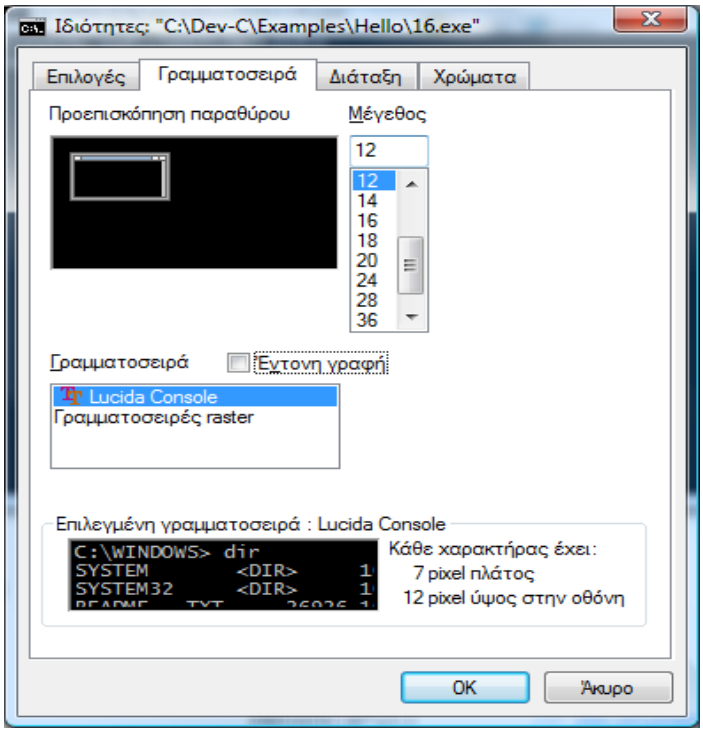


Εικόνα 1.3. Επιλογή γραμματοσειράς

Είναι γνωστό ότι όλοι οι χαρακτήρες εμφανίζονται στη μαύρη οθόνη σε μη γραφική μορφή (text mode) σαν σύνολο κουκίδων μέσα σε ένα προκαθορισμένο ορθογώνιο ανάλογα με την επιλεγείσα ανάλυση.

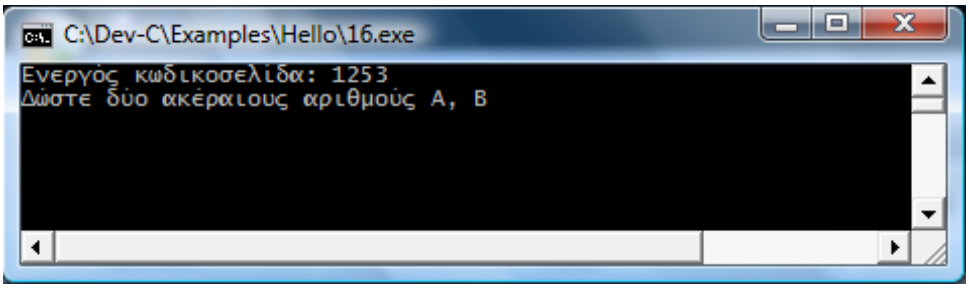
Εξ ορισμού, όταν ξεκινά η εκτέλεση ενός προγράμματος, η προεπιλεγμένη από το σύστημα ανάλυση των χαρακτήρων συνήθως είναι ένα ορθογώνιο 8x12 pixels και γραμματοσειρά εμφάνισης των χαρακτήρων η **Terminal,** η οποία παρ' όλο που είναι σύμφωνη με την κωδικοσελίδα **Windows 1253** δεν περιέχει τους ελληνικούς χαρακτήρες.

Με την επιλογή της γραμματοσειράς **Lucida Console,** η οποία είναι σύμφωνη με την κωδικοσελίδα **Windows 1253** και περιέχει τους ελληνικούς χαρακτήρες, μπορούμε να επιτύχουμε την εμφάνιση ταυτόχρονα Λατινικών και Ελληνικών χαρακτήρων στα μηνύματα εξόδου ενός εκτελέσιμου προγράμματος.



Εικόνα 1.4. Επιλογή της γραμματοσειράς Lucida Console

Παρατηρούμε ταυτόχρονα ότι, η ανάλυση της γραμματοσειράς **Lucida Console** είναι ένα ορθογώνιο 7x12 pixels της οθόνης δηλαδή, οι χαρακτήρες θα είναι πιο περιορισμένοι σε πλάτος αλλά ίδιου ύψους.



Εικόνα 1.5. Εμφάνιση ελληνικών χαρακτήρων

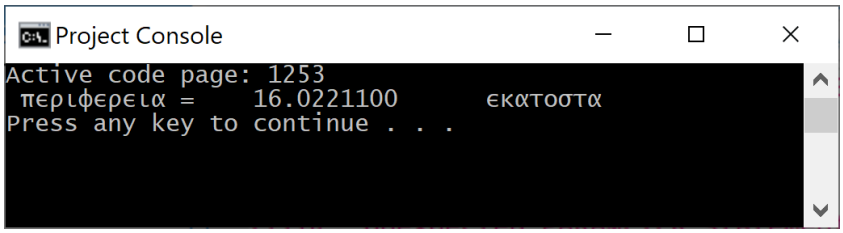
Όταν τελικά επιβεβαιώσουμε τις αλλαγές στις ιδιότητες του παραθύρου, επιλέγοντας το **OK (** ή πατώντας απλά το πλήκτρο **ENTER)**, το αποτέλεσμα φαίνεται στην εικόνα 1.5.

Με τον τρόπο αυτό έχουμε επιτύχει να εμφανίζονται σωστά τα ελληνικά μηνύματα ενός προγράμματος της γλώσσας C, στην οθόνη κάθε υπολογιστή που έχει εγκατεσημένη μια οποιαδήποτε έκδοση του λειτουργικού συστήματος των Windows!

Οι ίδιες διαδικασίες πρέπει να γίνουν και σε κάθε περιβάλλον εργασίας με τη γλώσσα Fortran, αρκεί να προστεθεί στον πηγαίο κώδικα η εντολή:

i=system("chcp 1253")

Η επόμενη εικόνα εμφανίζει το αποτέλεσμα εκτέλεσης ενός προγράμματος σε γλώσσα Fortran.



Συνιστάται να γίνεται συστηματική εφαρμογή αυτών των διαδικασιών έτσι ώστε να εμφανίζονται με ελληνικά γράμματα τα μηνύματα εξόδου στην οθόνη των προγραμμάτων τα οποία διευκολύνουν τόσο τη διαδικασία ανάπτυξης του πηγαίου κώδικα όσο και το διαρκή έλεγχο των αποτελεσμάτων των προγραμμάτων.

Για την ανάπτυξη του πηγαίου κώδικα και τη λεπτομερή δοκιμή του μπορεί να χρησιμοποιηθεί κάποιο ολοκληρωμένο περιβάλλον ανάπτυξης και εκτέλεσης προγραμμάτων.

Στα **Παραρτήματα Δ και Ε** του βιβλίου, έχουν συγκεντρωθεί τα ονόματα και οι διευθύνσεις στο Διαδίκτυο, των πιο γνωστών προϊόντων τα οποία περιλαμβάνουν ένα απλό περιβάλλον ανάπτυξης κώδικα και ένα ενσωματωμένο μεταγλωττιστή της γλώσσας.

Τα περισσότερα προϊόντα διατίθενται δωρεάν και επομένως μπορούν να αναζητηθούν και να εγκατασταθούν πολύ εύκολα σε όλους τους σύγχρονους υπολογιστές. Χρειάζεται μόνο λίγη προσοχή, να διαλέξουμε το σωστό λειτουργικό σύστημα και την έκδοσή του, για το οποίο διατίθενται.

Σημείωση. *Όλα τα παραδείγματα του βιβλίου έχουν ελεγχθεί από το ολοκληρωμένο περιβάλλον* ***Code::Blocks,*** *(http://www.codeblocks.org/) το οποίο είναι ανοικτού κώδικα (open source) και προσφέρεται εντελώς δωρεάν από το διαδίκτυο.*

1.5. Αριθμητικά Συστήματα

Λέμε αριθμητικό σύστημα ένα σύνολο τιμών που χρησιμοποιούνται για την αναπαράσταση μίας ποσότητας. Η μελέτη των αριθμητικών συστημάτων απασχόλησε τον άνθρωπο από πολύ παλιά και δεν περιορίζεται μόνο στους σύγχρονους υπολογιστές.

Οι υπολογιστές εκτελούν τις πράξεις χρησιμοποιώντας όχι το δεκαδικό σύστημα αρίθμησης, αλλά το δυαδικό που περιλαμβάνει μόνο τους αριθμούς «1» και «0».

Οι Βαβυλώνιοι χρησιμοποιούσαν το εξηνταδικό σύστημα (sexagesimal) και οι Ρωμαίοι επινόησαν ένα αριθμητικό σύστημα για τους αριθμούς από το 1 έως το 1 εκατομμύριο με τη χρήση μόνο 7 συμβόλων (τα γράμματα I, V, X, L, C, D, M) ενώ το γνωστό αραβικό σύστημα αρίθμησης ξεκίνησε πριν περίπου 2000 χρόνια.

Ένας αριθμός N σε οποιοδήποτε σύστημα αρίθμησης μπορεί να παρασταθεί από τη σχέση:

$$N=am\text{-}1\ b^{m-1} + am\text{-}2\ b^{m-2} + \ldots\ a1\ b^{1} + a0\ b^{0} + a\text{-}1\ b^{-1} + a\text{-}2\ b^{-2} + \ldots\ a\text{-}n\ b^{-n}$$

Όπου, τα ψηφία $a_{m-1}\ b^{m-1} + a_{m-2}\ b^{m-2} + \ldots\ a_1\ b^{1} + a_0\ b^{0}$

αποτελούν το ακέραιο μέρος του αριθμού,
ενώ τα ψηφία $a_{-1}\ b^{-1} + a_{-2}\ b^{-2} + \ldots\ a_{-n}\ b^{-n}$ είναι το κλασματικό του μέρος.

Με το γράμμα **b** παριστάνουμε τη βάση του αριθμητικού συστήματος στο οποίο εκφράζεται ο αριθμός (b>=2) και με $\mathbf{a_i}$ συμβολίζουμε τα ψηφία του αριθμού ($0 \le a_i \le b\text{-}1$). Το ψηφίο a_i πολλαπλασιάζεται με τον αριθμό b^i, γι' αυτό λέμε ότι η **τάξη** (order) του ψηφίου a_i είναι i.

Αν ο αριθμός N έχει m ακέραια ψηφία, οι εκθέτες i παίρνουν θετικές τιμές από 0 έως m-1 για το ακέραιο μέρος του και αν τα κλασματικά του ψηφία είναι n, οι εκθέτες i παίρνουν αρνητικές τιμές από -1 έως –n για το κλασματικό του τμήμα.

Π.χ. ο δεκαδικός αριθμός 37,428 με τον τρόπο αυτό γράφεται:

$$\mathbf{3}\cdot 10^{1} + \mathbf{7}\cdot 10^{0} + \mathbf{4}\cdot 10^{-1} + \mathbf{2}\cdot 10^{-2} + \mathbf{8}\cdot 10^{-3}$$

δηλαδή $a_1=3$, $a_0=7$, $a_{-1}=4$, $a_{-2}=2$, $a_{-3}=8$.

Ένα αριθμητικό σύστημα βάσης **b** χρειάζεται b διαφορετικά «ψηφία» για την παράσταση των αριθμών, που παίρνουν τιμές από **0** έως **b-1**.

Ένας ακέραιος αριθμός που έχει **m** ψηφία, στο σύστημα αυτό μπορεί να πάρει τιμές από **0** έως $\mathbf{b^{m-1}}$, δηλαδή $\mathbf{b^{m}}$ διαφορετικές τιμές.

Τα συνηθέστερα αριθμητικά συστήματα τα οποία χρησιμοποιούνται στην επιστήμη των υπολογιστών, είναι αυτά τα οποία έχουν βάση τους αριθμούς:

- 2 (δυαδικό σύστημα, binary system),
- 8 (οκταδικό σύστημα, octal system),
- 10 (δεκαδικό σύστημα, decimal system) και
- 16 (δεκαεξαδικό σύστημα, hexadecimal system).

Το δεκαεξαδικό σύστημα χρειάζεται 16 ψηφία για την παράσταση των αριθμών, και για τα επιπλέον 6 ψηφία χρησιμοποιούνται οι χαρακτήρες A-F, δηλαδή A=10, B=11, C=12, D=13, E=14 και F=15.

Τα ψηφία που χρησιμοποιεί το κάθε σύστημα αρίθμησης είναι:

ΔΥΑΔΙΚΟ : 0, 1

ΟΚΤΑΔΙΚΟ : 0, 1, 2, 3, 4, 5, 6, 7

ΔΕΚΑΔΙΚΟ : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

ΔΕΚΑΕΞΑΔΙΚΟ : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Συνήθως, χρησιμοποιούμε το δεκαδικό σύστημα αρίθμησης και αφού είμαστε εξοικειωμένοι με αυτό παριστάνουμε τους αριθμούς μόνο με τα ψηφία τους, π.χ. λέμε 15 και όχι $1\cdot 10^{1}+5\cdot 10^{0}$.

Το ίδιο μπορούμε να κάνουμε και με οποιοδήποτε άλλο αριθμητικό σύστημα, αρκεί να δηλώνουμε το σύστημα αυτό. Ο προσδιορισμός του συστήματος γίνεται συνήθως με ένα δείκτη που συνοδεύει τον αριθμό και δηλώνει τη βάση του αριθμητικού συστήματος.

‘Ετσι, ένας αριθμός **X** μπορεί να εκφραστεί σε οποιοδήποτε αριθμητικό σύστημα με βάση **b**, και το συμβολίζουμε $\mathbf{(X)_b}$. Ο ίδιος αριθμός (π.χ. 28 στο δεκαδικό σύστημα αρίθμησης) εκφράζεται στα διάφορα συστήματα με τον ακόλουθο τρόπο.

$$(28)_{10} = 2x10^{1} + 8x10^{0}$$
$$(28)_{10} = 3x8^{1} + 4x8^{0} = (34)_{8}$$
$$(28)_{10} = 1x16^{1} + 12x16^{0} = (1C)_{16}$$
$$(28)_{10} = 1x2^{4} + 1x2^{3} + 1x2^{2} + 0x2^{1} + 0x2^{0} = (11100)_{2}$$

1.5.1. Μετατροπές αριθμών σε συστήματα με άλλη βάση

Η μετατροπή ενός αριθμού από ένα σύστημα με βάση **b**, στο δεκαδικό σύστημα αρίθμησης, γίνεται αν αναπτύξουμε τον αριθμό σε μία ακολουθία δυνάμεων του **b** και στη συνέχεια προσθέσουμε όλους τους όρους.

Ξεκινάμε πάντα από το λιγότερο σημαντικό ψηφίο δηλαδή, με μηδενική δύναμη, μέχρι να πάρουμε όλους τους όρους, αυξάνοντας τη δύναμη της βάσης κάθε φορά κατά ένα, ή μετράμε το πλήθος των ψηφίων και η δύναμη του μεγιστοβάθμιου όρου προκύπτει αν αφαιρέσουμε ένα (1) από το πλήθος των ψηφίων, δηλαδή αν το πλήθος των ψηφίων είναι 6 τότε η δύναμη του μεγιστοβάθμιου όρου είναι το 5.

Παραδείγματα:

- Να μετατραπεί ο αριθμός $(101011)_2$ στο δεκαδικό σύστημα αρίθμησης

  $1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 =$

  $\mathbf{32 + 0 + 8 + 0 + 2 + 1 = (43)_{10}}$
- Να μετατραπεί ο αριθμός $(3765)_8$ στο δεκαδικό σύστημα αρίθμησης

  $3 \cdot 8^3 + 7 \cdot 8^2 + 6 \cdot 8^1 + 5 \cdot 8^0 = \mathbf{1536 + 448 + 48 + 5} = \mathbf{(2037)_{10}}$
- Να μετατραπεί ο αριθμός $(2E)_{16}$ στο δεκαδικό σύστημα αρίθμησης

  $2 \cdot 16^1 + E \cdot 16^0 = 2 \cdot 16^1 + 14 \cdot 16^0 = \mathbf{32 + 14 = (46)_{10}}$

Η μετατροπή ενός αριθμού από το δεκαδικό σύστημα αρίθμησης σε οποιοδήποτε άλλο σύστημα αρίθμησης με βάση **b** γίνεται με επαναλαμβανόμενη διαίρεση του δεκαδικού αριθμού με τη βάση **b** μέχρι το πηλίκο της διαίρεσης να γίνει 0.

Οι συντελεστές του επιθυμητού αριθμού στο σύστημα με βάση **b** λαμβάνονται από τα υπόλοιπα των διαιρέσεων, με το υπόλοιπο της πρώτης διαίρεσης να αντιστοιχεί στο ελάχιστο σημαντικό ψηφίο (Least Significant Digit – LSD) και το τελευταίο υπόλοιπο στο μέγιστο σημαντικό ψηφίο (Most Significant Digit – MSD).

<u>Παραδείγματα:</u>

Να μετατραπεί ο αριθμός $\mathbf{(143)_{10}}$ στο δυαδικό, οκταδικό και δεκαεξαδικό σύστημα αρίθμησης

- Μετατροπή αριθμού στο δυαδικό σύστημα αρίθμησης

	143	2							
LSD	**1**	71	2						
		1	35	2					
			1	17	2				
				1	8	2			
					0	4	2		
						0	2	2	
							0	1	2
							MSD	**1**	0

Άρα θα έχουμε $(143)_{10} = (10001111)_2$

- Μετατροπή αριθμού στο οκταδικό σύστημα αρίθμησης

	143	8							
LSD	**7**	17	8						
		1	2	8					
		MSD	**2**	0					

Άρα θα έχουμε $(143)_{10} = (217)_8$

- Μετατροπή αριθμού στο δεκαεξαδικό σύστημα αρίθμησης

	143	16							
LSD	**15**	8	16						
	MSD	**8**	0						

Άρα θα έχουμε $(143)_{10} = (8F)_{16}$

1.5.2. Μετατροπή αριθμών μικρότερων της μονάδας

Η μετατροπή ενός αριθμού μικρότερου της μονάδας από το δεκαδικό σύστημα αρίθμησης σε οποιοδήποτε άλλο σύστημα αρίθμησης με βάση **b** γίνεται αν εκτελούμε συνεχείς πολλαπλασιασμούς με τη βάση του νέου συστήματος αρίθμησης. Η διαδικασία ολοκληρώνεται όταν το υπόλοιπο είναι μηδέν ή μέχρι να πετύχουμε μια ικανοποιητική προσέγγιση.

Δηλαδή, κρατάμε το ακέραιο ψηφίο που προκύπτει μετά τον πολλαπλασιασμό του αριθμού με τη βάση στην οποία θέλουμε να μετατρέψουμε τον αριθμό. Κατόπιν το ακέραιο μέρος που προκύπτει αφαιρείται από τον αριθμό. Ο νέος αριθμός που προκύπτει πολλαπλασιάζεται εκ νέου με τη βάση κ.ο.κ. Έτσι, λαμβάνουμε τα προκύπτοντα ακέραια μέρη και πολλαπλασιάζουμε με τη βάση του συστήματος μόνο το νέο κλασματικό μέρος.

Παραδείγματα:

Να μετατραπεί ο αριθμός $\mathbf{(0,346)_{10}}$ στο δυαδικό, οκταδικό και δεκαεξαδικό σύστημα αρίθμησης

- Μετατροπή αριθμού στο δυαδικό σύστημα αρίθμησης:

0,346	0,692	0,384	0,768	0,536
X 2	X 2	X 2	X 2	X 2
0,692	1,384	0,768	1,536	1,072
0	**1**	**0**	**1**	**1**

Άρα θα έχουμε $(0,346)_{10} = (0,01011)_2$

- Μετατροπή αριθμού στο οκταδικό σύστημα αρίθμησης:

0,346	0,768	0,384	0,072	0,576
X 8	X 8	X 8	X 8	X 8
2,768	6,144	3,072	0,576	4,608
2	**6**	**3**	**0**	**4**

Άρα θα έχουμε $(0,346)_{10} = (0,26304)_8$

- Μετατροπή αριθμού στο δεκαεξαδικό σύστημα αρίθμησης:

0,346	0,536	0,576	0,216	0,456
X 16	X 16	X 16	X 16	X 16
5,536	8,576	9,216	3,456	7,296
5	**8**	**9**	**3**	**7**

Άρα θα έχουμε $(0,346)_{10} = (0,58937)_{16}$

Σημείωση. *Όταν ο αριθμός περιέχει και ακέραιο και κλασματικό μέρος, τότε η μετατροπή γίνεται χωριστά για το ακέραιο μέρος και χωριστά για το κλασματικό μέρος. Ο τελικός αριθμός προκύπτει από τη σύνθεση του ακεραίου και του κλασματικού μέρους.*

Ένας συνηθισμένος τρόπος για να μετατρέψουμε έναν αριθμό από οποιοδήποτε σύστημα αρίθμησης σ' έναν ισοδύναμο αριθμό διαφορετικού συστήματος αρίθμησης, είναι να το μετατρέψουμε αρχικά στον ισοδύναμο δεκαδικό και μετά στο επιθυμητό σύστημα.

Πίνακας 1.2. Παράσταση αριθμών στα κύρια αριθμητικά συστήματα

ΔΕΚΑΔΙΚΟ	ΔΥΑΔΙΚΟ	ΟΚΤΑΔΙΚΟ	ΔΕΚΑΕΞΑΔΙΚΟ
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Ένας άλλος σύντομος τρόπος μετατροπής ενός <u>ακέραιου αριθμού</u> του δεκαεξαδικού συστήματος αρίθμησης, είναι να μετατρέψουμε αρχικά το δεκαεξαδικό αριθμό στον αντίστοιχο δυαδικό αριθμό αντικαθιστώντας κάθε δεκαεξαδικό ψηφίο με τα αντίστοιχα 4 δυαδικά ψηφία.

Κατόπιν, ξεκινώντας από τα δεξιά, χωρίζουμε εκ νέου τον αριθμό, αυτή τη φορά σε τριάδες ψηφίων. Κάθε τριάδα θα αντιστοιχεί σε έναν αριθμό του οκταδικού συστήματος αρίθμησης όπως προκύπτει από τον πίνακα 1.2..

Π.χ. ο αριθμός **$(ABCD)_{16}$** θα γίνει :

A	B	C	D
1010	1011	1100	1101

Ο αντίστοιχος δυαδικός αριθμός θα είναι: **1010 1011 1100 1101**. Αν τον χωρίσουμε σε τριάδες, ξεκινώντας πάντα από τα δεξιά προς τα αριστερά, θα έχουμε:

001	010	101	111	001	101

1	**2**	**5**	**7**	**1**	**5**

Επομένως, ο αντίστοιχος αριθμός στο οκταδικό σύστημα αρίθμησης θα είναι **$(125715)_8$**

1.5.3. Πράξεις στο δυαδικό σύστημα αρίθμησης

Κάθε υπολογιστής, εκτελεί αριθμητικές και λογικές πράξεις με τη βοήθεια της αριθμητικής και λογικής μονάδας (ALU). Οι τέσσερεις αριθμητικές πράξεις (πρόσθεση, αφαίρεση, πολλαπλασιασμός και διαίρεση) πραγματοποιούνται στο δυαδικό σύστημα αρίθμησης και εκτελούνται ακριβώς όπως και στο δεκαδικό σύστημα αρίθμησης. Πρέπει όμως κάθε προγραμματιστής να γνωρίζει τον τρόπο εκτέλεσης των τεσσάρων πράξεων στο δυαδικό σύστημα αρίθμησης προκειμένου να κατανοεί καλλίτερα τον τρόπο με τον οποίο πραγματοποιούνται οι πράξεις στα υπολογιστικά συστήματα.

Α. Πρόσθεση δυαδικών αριθμών

Βασική παρατήρηση. Εάν από την πρόσθεση δύο ψηφίων ίδιας τάξης φθάσουμε ή υπερβούμε τη βάση του συστήματος (τον αριθμό **2)**, μεταφέρουμε την πλεονάζουσα δυάδα στο αμέσως μεγαλύτερης τάξης ψηφίο.

Προσθέτουμε δύο δυαδικούς αριθμούς, κατά στήλες, αρχίζοντας από τα δεξιά και συνεχίζουμε προς τα αριστερά, όπως προσθέτουμε δύο δεκαδικούς αριθμούς.

Στην περίπτωση που προσθέτουμε περισσότερους από τρεις δυαδικούς αριθμούς, χρειάζεται ιδιαίτερη προσοχή, γιατί η προς μεταφορά δυάδα μπορεί να μετατοπισθεί περισσότερες από μία θέσεις προς τα αριστερά.

Π.χ. για την πρόσθεση των αριθμών **$(10110)_2$** και **$(11101)_2$** θα έχουμε:

Κρατούμενο			1	1	0	0
Αριθμός Α		1	0	1	1	0
Αριθμός Β		1	1	1	0	1
Άθροισμα	1	1	0	0	1	1

Άρα θα έχουμε σαν αποτέλεσμα τον αριθμό **$(110011)_2$**

Β. Αφαίρεση δυαδικών αριθμών

Η αφαίρεση των δυαδικών αριθμών πραγματοποιείται όπως ακριβώς και στους δεκαδικούς αριθμούς. Βασική προϋπόθεση της αφαίρεσης είναι η λήψη μιας δανεικής (borrow) δυάδας από το μεγαλύτερης τάξης ψηφίο, στην περίπτωση που το ψηφίο του μειωτέου είναι μικρότερο του ψηφίου του αφαιρετέου.

Π.χ. για την αφαίρεση του αριθμού **$(10001111)_2$** (αφαιρετέος) από τον αριθμό **$(1010101)_2$** (μειωτέος) θα έχουμε:

Μειωτέος		1	0	1	0	1	0	1
Αφαιρετέος		1	0	0	0	1	1	1
Διαφορά		0	0	0	1	1	1	0

Άρα θα έχουμε σαν αποτέλεσμα τον αριθμό **$(0001110)_2$**

Γ. Πολλαπλασιασμός και διαίρεση δυαδικών αριθμών

Η αριθμητική και λογική μονάδα (Arithmetic and Logical Unit ή ALU) σε κάθε υπολογιστή εκτελεί μόνο πρόσθεση και αφαίρεση. Επομένως, ο πολλαπλασιασμός των δυαδικών αριθμών ανάγεται σε επαναλαμβανόμενες προσθέσεις τις οποίες αναλαμβάνουν οι αθροιστές. Το ίδιο ισχύει και για τη διαίρεση. Η διαίρεση των δυαδικών αριθμών ανάγεται σε επαναλαμβανόμενες αφαιρέσεις.

Στο δυαδικό σύστημα αρίθμησης, ένας εύκολος τρόπος να πολλαπλασιάσουμε με το **2** είναι να ολισθήσουμε όλα τα ψηφία προς τα αριστερά κατά μια θέση και να προσθέσουμε ένα **0** (μηδέν) στα δεξιό άκρο.

Π.χ. 11 x 10 = 110

Η ολίσθηση προς τα αριστερά δύο φορές, ισοδυναμεί με τον πολλαπλασιασμό επί 4:

Π.χ. 11 x 100 = 1100

Έτσι, ένας δυαδικός αριθμός όταν πολλαπλασιάζεται κατά $\mathbf{2^n}$ ολισθαίνει **n** θέσεις προς τα αριστερά.

Συμπληρωματικά, η ολίσθηση προς τα δεξιά ισοδυναμεί με διαίρεση δια του **2**.

Π.χ. 110 ÷ 10 = 11

1.6. Παράσταση αριθμών και χαρακτήρων

Ένας καλός προγραμματιστής για να γίνει και αποδοτικός πρέπει να γνωρίζει πολύ καλά τον τρόπο με τον οποίο αποθηκεύονται και επεξεργάζονται τα δεδομένα και οι πληροφορίες στη μνήμη του υπολογιστή.

Στη συνέχεια, θα παρουσιαστούν οι τρόποι παράστασης των ακεραίων και των πραγματικών στη μνήμη, καθώς και τα ισχύοντα πρότυπα.

Η μνήμη των υπολογιστών είναι οργανωμένη σε λέξεις (words) δηλαδή, σε ομάδες από bits ή bytes τα οποία ορίζουν το μήκος των λέξεων όπου αποθηκεύονται:

- οι αριθμοί για επεξεργασία και
- οι εκτελέσιμες εντολές των προγραμμάτων

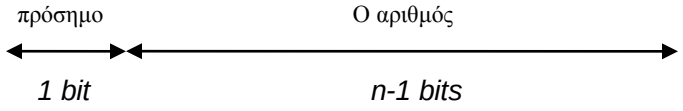
Το μήκος των λέξεων δεν είναι σταθερό σε όλους τους υπολογιστές, αλλά ορίζεται από την ακολουθούμενη τεχνολογία και αρχιτεκτονική των υπολογιστών.

1.6.1. Παράσταση ακεραίων αριθμών

Κάθε ακέραιος αριθμός, εξ ορισμού, καταλαμβάνει τόσο χώρο στη μνήμη όσο μία λέξη της μνήμης του υπολογιστή.

Έτσι, σε μία λέξη της μνήμης του υπολογιστή με **n** δυαδικά ψηφία, μπορούμε να αποθηκεύσουμε $\mathbf{2^n}$ το πλήθος διαφορετικούς αριθμούς του δυαδικού συστήματος αρίθμησης, από το 0 (μηδέν) μέχρι $\mathbf{2^n - 1}$.

Όταν θέλουμε με **n** δυαδικά ψηφία να παραστήσουμε τους ακέραιους αριθμούς μαζί με το πρόσιμό τους (προσημασμένους) τότε, εκμεταλλευόμαστε το αριστερότερο bit του αριθμού, το περισσότερο σημαντικό ψηφίο, στο οποίο κωδικοποιούμε το πρόσημό του, όπως εμφανίζεται στην εικόνα 1.6.



Εικόνα 1.6. *Παράσταση μιας λέξης μνήμης ενός ακεραίου αριθμού*

Αν το πρόσημο έχει την τιμή 0, τότε ο αριθμός είναι θετικός, ενώ αν έχει την τιμή 1 είναι αρνητικός. Με τα υπόλοιπα n-1 δυαδικά ψηφία κωδικοποιούμε την απόλυτη τιμή του αριθμού δηλαδή, το μέτρο του.

Π.χ. οι αριθμοί$(01110010)_2$ και $(00100001)_2$ είναι θετικοί,

ενώ οι αριθμοί $(11100101)_2$ και $(10011001_{)2}$ είναι αρνητικοί.

Σημείωση. *Τα ψηφία ενός αριθμού γραμμένου στο δυαδικό σύστημα ονομάζονται bits (**b**inary dig**its**, δυαδικά ψηφία). Το πιο αριστερό ψηφίο του αριθμού ονομάζεται περισσότερο σημαντικό ψηφίο (Most Significant Bit, MSB), γιατί πολλαπλασιάζεται με το μεγαλύτερο συντελεστή και το πιο δεξιό ψηφίο του αριθμού ονομάζεται λιγότερο σημαντικό ψηφίο (Least Significant Bit, LSB), γιατί πολλαπλασιάζεται με το μικρότερο συντελεστή.*

Επειδή ο μεγαλύτερος ακέραιος αριθμός ο οποίος μπορεί να παρασταθεί στη μνήμη με **n-1** bits είναι ο $\mathbf{2^{n-1}-1}$, γι' αυτό το λόγο η τιμή του αριθμού δεν μπορεί να ξεπεράσει αυτό το όριο.

Έτσι, σ' έναν υπολογιστή όπου το μήκος της λέξης του είναι 8 bits, ο αριθμός 21 θα παρασταθεί ως 00010101. Το αριστερότερο bit δηλώνει ότι ο αριθμός είναι θετικός και τα υπόλοιπα bits παριστούν τον αριθμό 21.

Ο μεγαλύτερος θετικός αριθμός ο οποίος μπορεί ν' αποθηκευτεί σε μια λέξη μνήμης των 8 bits είναι ο 01111111 δηλαδή ο 127 (2^{8-1}-1 = 2^7-1 = 128-1) ενώ σε μια λέξη μνήμης των 32 bits είναι ο αριθμός 01111111111111111111111111111111 δηλαδή, ο αριθμός 2147483647 (2^{32-1}-1 = 2^{31}-1).

Ο αριθμός 0 (μηδέν) μπορεί να παρασταθεί με δύο τρόπους:

είτε σαν 00…00 είτε σαν 10…00.

1.6.2. Παράσταση των πραγματικών αριθμών στη μνήμη

Η μορφή με την οποία αποθηκεύεται ένας πραγματικός αριθμός στη μνήμη του υπολογιστή εμφανίζεται στην εικόνα 1.7. και ονομάζεται παράσταση *κινητής υποδιαστολής* ή *επιπλέουσας υποδιαστολής* (floating point).

Για την παράσταση αυτή ο πραγματικός αριθμός καταλαμβάνει τέσσερα (4) συνεχόμενα bytes στη μνήμη (32 bits).

	Πρόσημο	*εκθέτης*	*συντελεστής*
Θέση του Bit:	31	*30 23*	*22 0*

Εικόνα 1.7. Παράσταση στη μνήμη ενός πραγματικού αριθμού (32 bits)

Στην παράσταση κινητής υποδιαστολής ένας πραγματικός αριθμός X εκφράζεται σε *εκθετική μορφή* (exponential representation) με τον ακόλουθο τρόπο:

$$\mathbf{X = s * m * b^e}$$

όπου:

s, το πρόσημο του αριθμού (s = + ή s = -)

m, ο συντελεστής (mantissa)

b, η βάση του συστήματος αρίθμησης (συνήθως 2 ή 8 ή 10 ή 16) και

e, ο εκθέτης του X

Έτσι, για ένα πραγματικό αριθμό *κινητής υποδιαστολής* ο οποίος αποθηκεύεται σε τέσσερα (4) συνεχόμενα bytes στη μνήμη, το πρόσημο είναι μόνο ένα (1) bit, ο εκθέτης επτά (7) bits και ο συντελεστής είκοσι τέσσερα (24) bits.

Γενικά, ο *συντελεστής* (mantissa) **m**, αποτελείται το μέγιστο από **p** ψηφία τα οποία ανήκουν στη βάση **e** δηλαδή, εκφράζει την ακρίβεια του αριθμού **X**.

Με τον τρόπο αυτό ένας πραγματικός αριθμός στους υπολογιστές εκφράζεται σαν το γινόμενο ενός κλασματικού αριθμού και μιας δύναμης του 2. Αυτή η παράσταση του αριθμού δεν είναι μοναδική.

Π.χ. Ο αριθμός 101,011 σε εκθετική μορφή, μπορεί να γραφτεί:

$0{,}101011 * 2^3$ όπου $m = 0{,}101011$ και $e = (3)_{10} = (11)_2$

$1{,}01011 * 2^2$ όπου $m = 1{,}01011$ και $e = (2)_{10} = (10)_2$

$10{,}1011 * 2^1$ όπου $m = 10{,}1011$ και $e = (1)_{10} = (1)_2$

Η έννοια του εκθέτη δεν είναι μοναδική, ούτε είναι απόλυτη, με την έννοια των μαθηματικών, αλλά είναι σχετική με το επιλεγόμενο μέγεθος του συντελεστή.

Φυσικά, το ίδιο συμβαίνει και στο δεκαδικό σύστημα αρίθμησης όπου κάθε αριθμός μπορεί να παρασταθεί με πολλές εκθετικές μορφές.

Π.χ. ο αριθμός 1234, μπορεί να γραφτεί:

$1{,}234 * 10^3$ ή

$12{,}34 * 10^2$ ή *ακόμη*

$0{,}001234 * 10^6$

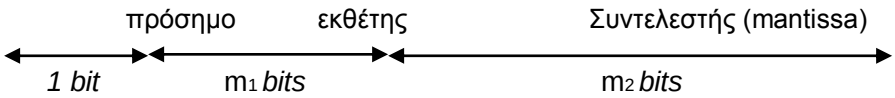
Επειδή υπάρχουν πολλές εναλλακτικές παραστάσεις ενός αριθμού σε εκθετική μορφή, στους υπολογιστές έχει επιλεγεί μία από τις παραστάσεις αυτές, η οποία έχει την ιδιότητα $½ <= m < 1$ και ονομάζεται *κανονική μορφή* (normal form).

Η κανονική μορφή έχει δύο βασικά χαρακτηριστικά:

- Το ακέραιο μέρος του αριθμού είναι πάντα 0 (μηδέν). Έτσι, δε χρειάζεται να αποθηκεύουμε το ακέραιο μέρος γιατί, η τιμή του είναι γνωστή και πάντα δεδομένη
- Το πρώτο κλασματικό ψηφίο του αριθμού είναι πάντα 1. Αυτό συμβαίνει επειδή, οι κλασματικοί αριθμοί οι οποίοι είναι μεγαλύτεροι από το ½ ($=2^{-1}$), στο δυαδικό σύστημα περιέχουν πάντα τον προσθετέο 2^{-1}

Άρα, από τις εκθετικές μορφές του αριθμού $(101{,}011)_2$, η κανονική μορφή του είναι η μορφή $0{,}101011 * 2^3$.

Όταν ένας αριθμός εκφράζεται στην κανονική εκθετική μορφή (κανονικοποιημένος) μπορούμε να τον κωδικοποιήσουμε αφιερώνοντας $\mathbf{m_1}$ δυαδικά ψηφία στον εκθέτη και $\mathbf{m_2}$ δυαδικά ψηφία στο συντελεστή, όπως υποδεικνύεται στην εικόνα 1.8., όπου διατηρείται και ένα ψηφίο το οποίο θα κωδικοποιεί το πρόσημο του αριθμού.



Εικόνα 1.8. Παράσταση στη μνήμη ενός πραγματικού αριθμού (32 bits)

Το κλασματικό μέρος του συντελεστή παριστάνεται σαν ένας δυαδικός αριθμός με $\mathbf{m_2}$ ψηφία. Εάν ο συντελεστής έχει λιγότερα από $\mathbf{m_2}$ ψηφία, προστίθενται μηδενικά στο τέλος, ενώ αν έχει περισσότερα από $\mathbf{m_2}$ ψηφία, τότε στρογγυλοποιείται.

Στη *στρογγυλοποίηση* (rounding), αγνοούνται τα επί πλέον ψηφία, αλλά εάν το πρώτο επί πλέον ψηφίο είναι 1, τότε προστίθεται η μονάδα (1) στο λιγότερο σημαντικό ψηφίο του συντελεστή.

Ο εκθέτης παριστάνεται και αυτός σαν ένας δυαδικός αριθμός με m_1 ψηφία. Το πιο σημαντικό από τα ψηφία της λέξης έχει την τιμή 0, όταν ο εκθέτης είναι θετικός και την τιμή 1 όταν ο εκθέτης είναι αρνητικός.

Οι πράξεις με πραγματικούς αριθμούς κινητής υποδιαστολής στον υπολογιστή εκτελούνται με τον ακόλουθο τρόπο:

Για την πρόσθεση δύο πραγματικών αριθμών κινητής υποδιαστολής, πρέπει πρώτα να κανονικοποιηθούν για να έχουν τον ίδιο εκθέτη. Αν ο ένας αριθμός έχει εκθέτη ε_1 και ο άλλος ε_2, και ισχύει $\varepsilon_1 < \varepsilon_2$, τότε αυξάνεται ο εκθέτης ε_1 κατά ε_2-ε_1 και *ολισθαίνει* (shifting) ο συντελεστής του αριθμού αυτού προς τα δεξιά κατά ε_2-ε_1 ψηφία.

Κατά την ολίσθηση, τα δεξιότερα ψηφία του αριθμού χάνονται έτσι, ο αριθμός μπορεί να μεταβληθεί και το τελικό αποτέλεσμα, μετά την ολίσθηση, μπορεί να μην είναι ακριβές. Στη συνέχεια, προστίθενται οι συντελεστές των αριθμών και μετατρέπεται ξανά το αποτέλεσμα στην κανονική μορφή στρογγυλοποιώντας το συντελεστή.

Π.χ. όταν έχουμε 8 ψηφία για το συντελεστή και 4 ψηφία για τον εκθέτη τότε, ο αριθμός $x = (16,125)_{10}$

θα παριστάνεται ως: $0,10000001 * 2^5$ *και*

ο αριθμός $y = (4,3125)_{10}$

θα παριστάνεται ως: $0,10001010 * 2^3$

Για την πρόσθεσή τους, μετατρέπεται πρώτα ο αριθμός με το μικρότερο εκθέτη δηλαδή, ο y αυξάνοντας τον εκθέτη του κατά 2 και ολισθαίνει ο συντελεστής του προς τα δεξιά κατά 2 ψηφία.

Στη συνέχεια, προστίθενται οι δύο συντελεστές.

Το άθροισμά τους δεν απαιτεί κανονικοποίηση, αφού είναι και το τελικό αποτέλεσμα.

Η τιμή του αποτελέσματος θα είναι:

$0,10100.011 * 2^5$ δηλαδή ο αριθμός **20,375**

Το ορθό αποτέλεσμα της πρόσθεσης είναι 20,4375. Το σφάλμα στην πρόσθεση ανέρχεται σε 0,625 και οφείλεται στη μετατροπή του **y** για να αποκτήσει τον ίδιο εκθέτη με το **x**.

Οι τυπικές πράξεις του πολλαπλασιασμού και της διαίρεσης με πραγματικούς αριθμούς κινητής υποδιαστολής στον υπολογιστή εκτελούνται με τον ακόλουθο τρόπο:

Για τον πολλαπλασιασμό δύο αριθμών, προστίθενται οι εκθέτες τους και πολλαπλασιάζονται οι συντελεστές τους. Στη συνέχεια, μετατρέπεται το αποτέλεσμα στην κανονική μορφή.

Για τη διαίρεση δύο αριθμών, αφαιρούνται οι εκθέτες και διαιρούνται οι συντελεστές.

Η μετατροπή των αριθμών ή η στρογγυλοποίησή τους κατά την εκτέλεση των πράξεων σε ένα αυτόματο υπολογιστικό σύστημα επιφέρει συνήθως ένα μικρό σφάλμα στο αποτέλεσμα.

Επειδή, μετά από πολλές πράξεις αυτά τα μικρά σφάλματα μπορούν να συσσωρευτούν, τα τελικά αποτελέσματα ενός προγράμματος μπορεί να μην είναι απολύτως ακριβή.

Για να έχουμε το μικρότερο δυνατό σφάλμα όσον αφορά την ακριβή μαθηματική λύση, πρέπει το υπολογιστικό σύστημα να λειτουργεί με το μέγιστο δυνατό πλήθος ψηφίων στο συντελεστή (mantissa).

Σημείωση. *Σε όλες τις μετατροπές το πλήθος των ψηφίων του συντελεστή (mantissa) παραμένει σταθερό.*

1.6.3. Το πρότυπο IEEE 754

Το 1985 το IEEE (Institute of Electrical and Electronics Engineers) στην προσπάθεια δημιουργίας μιας τυποποίησης της επεξεργασίας των πραγματικών αριθμών καθιέρωσε το πρότυπο IEEE 754 βελτιώνοντας το παλαιότερο πρότυπο CEI 60559 (Commision Electrotechnique International), για τον τρόπο παρουσίασης και επεξεργασίας ενός πραγματικού αριθμού κινητής υποδιαστολής στους σύγχρονους υπολογιστές.

Πίνακας 1.3. Κατανομή των bits στη μνήμη

	Κωδικοποίηση	**Πρόσημο**	**Εκθέτης**	**Συντελεστής (Mantissa)**
Απλή ακρίβεια (τύπος float)	32 bits	1 bit	8 bits	23 bits
Διπλή ακρίβεια (τύπος double)	64 bits	1 bit	11 bits	52 bits

Το πρότυπο IEEE 754, ορίζει δύο διαφορετικές μορφές για τους πραγματικούς αριθμούς κινητής υποδιαστολής και τις αντίστοιχες πράξεις οι οποίες μπορούν να πραγματοποιηθούν.

Η πρώτη μορφή αφορά μια λέξη μνήμης των 32 bits (τύπος **float**) και λέγεται *απλής ακρίβειας* (simple precision) ενώ η δεύτερη μορφή μια λέξη μνήμης των 64 bits (τύπος **double**) και λέγεται *διπλής ακρίβειας* (double precision).

Η κατανομή των bits στις δύο μορφές περιγράφεται στον Πίνακα 1.3.

Όλες οι σύγχρονες αρχιτεκτονικές των υπολογιστών περιλαμβάνουν μια υλοποίηση του προτύπου IEEE 754 μέσα στους επεξεργαστές τους, επιτυγχάνοντας έτσι ταχύτερη εκτέλεση των πράξεων.

1.6.4. Το πρότυπο ISO 10646

Πρώτη απόπειρα κωδικοποίησης των χαρακτήρων οι οποίοι χρησιμοποιούνταν στους υπολογιστές έγινε το 1963 με την κωδικοποίησή τους σε ένα byte (7-bits) και με το όνομα ASCII (American Standard Code for Information Interchange) γνωστή και ως ANSI X3.4.

Το 1967 προστέθηκαν και τα πεζά (μικρά) γράμματα των χαρακτήρων στον κώδικα ASCII ο οποίος περιλαμβάνει 33 χαρακτήρες ελέγχου, 33 σύμβολα (στίξης, πράξεων κτλ) και όλα τα γράμματα του λατινικού αλφάβητου, κεφαλαία και πεζά γράμματα (a-z), συνολικά 128 θέσεις (27).

Για να προσαρμοστεί στις απαιτήσεις των χρηστών των υπολογιστών παγκοσμίως η IBM δημιούργησε μια επέκταση του κώδικα ASCII γνωστή ως κωδικοσελίδα (code pages) χρησιμοποιώντας και τις οκτώ θέσεις ενός byte δηλαδή 256 θέσεις. Κάθε κωδικοσελίδα περιέχει τους χαρακτήρες είτε μιας μόνο γλώσσας, όπως τα ελληνικά είτε μιας οικογένειας γλωσσών, όπως τα σλαβικά (Western European).

Ένα πρόγραμμα για να κάνει χρήση μιας κωδικοσελίδας πρέπει να την προσδιορίσει, αλλά δυστυχώς μια κωδικοσελίδα εξαρτάται από το λειτουργικό σύστημα και προκαλεί πολλές δυσκολίες.

Το 1980, ο διεθνής οργανισμός τυποποίησης, International Standards Organization (ISO) δημιούργησε δέκα έξι διαφορετικά τυποποιημένα σύνολα χαρακτήρων των οκτώ bits (256 θέσεων), παρόμοια με αυτά των κωδικοσελίδων, γνωστών ως ISO 8859-1 μέχρι 8859-16, τα οποία περιλαμβάνουν τους βασικούς χαρακτήρες του κώδικα ASCII.

Π.χ. το ISO 8859-1, είναι γνωστό και ως "Latin-1" ή "Extended Latin" και περιλαμβάνει 22 διαφορετικές γλώσσες των ευρωπαϊκών χωρών.

Το ISO 8859-7 περιλαμβάνει τους ελληνικούς χαρακτήρες.

Από το 2004 έχει σταματήσει η υποστήριξη του ISO 8859 και η προσπάθεια τυποποίησης των χαρακτήρων μεταφέρθηκε στο ISO/CEI 10646.

Από το 1991, ξεκίνησε μια προσπάθεια ομοιογενοποίησης και αρμονικής συνεργασίας όλων των λειτουργικών συστημάτων και προγραμμάτων με τη συνεργασία του ISO και μιας ομάδας εργασίας γνωστής ως Unicode.

Η συνεργασία αυτή αποβλέπει στη δημιουργία ενός πίνακα αντιστοιχίας όλων των χαρακτήρων όλων των γλωσσών της υφηλίου και να αποδώσει ένα όνομα σε κάθε χαρακτήρα. Η κωδικοποίηση του Unicode περιλαμβάνει αυτούσια την τυποποίηση ISO/CEI 10646 γνωστής και ως UCS (Universal Character Set). Η τυποποίηση ISO/CEI 10646 περιλαμβάνει σήμερα, περίπου 100.000 διαφορετικούς χαρακτήρες και έχει τη δυνατότητα να κωδικοποιήσει μέχρι 1 εκατομμύριο χαρακτήρες (από 0x0 μέχρι 0x10FFFF). Κάθε χαρακτήρας σημειώνεται ως U+nnnn, όπου nnnn είναι ένας αριθμός του δεκαεξαδικού συστήματος αρίθμησης και αποτελεί μια σειρά από 4 έως 6 ψηφία. Π.χ. ο χαρακτήρας A (του λατινικού αλφάβητου) αντιστοιχεί στον κωδικό U+0041.

Η κωδικοποίηση των χαρακτήρων τόσο από το Unicode όσο και από το UCS ή ISO/CEI 10646, πραγματοποιείται σε τρεις διαφορετικές μορφές ανάλογα με τον αριθμό των χρησιμοποιούμενων bits και λέγεται UTF (UCS transformation format).

Έτσι έχουμε:

- Το **UTF-8**, το οποίο χρησιμοποιεί συνέχειες ενός μέχρι τεσσάρων κωδικών των 8 bits
- Το **UTF-16**, το οποίο χρησιμοποιεί συνέχειες ενός ή δύο κωδικών των 16 bits
- Το **UTF-32**, το οποίο χρησιμοποιεί ένα κωδικό των 32 bits

Η κωδικοποίηση των χαρακτήρων έχει επικρατήσει να υλοποιείται με τη μορφή **UTF8** και αυτή θα πρέπει να χρησιμοποιείται στην ανάπτυξη νέου κώδικα.

Σημείωση. *Οι 256 πρώτοι χαρακτήρες του Unicode ή UCS είναι ταυτόσημοι με αυτούς του ISO 8859-1 και οι 127 πρώτοι χαρακτήρες του Unicode ή UCS είναι ταυτόσημοι με αυτούς του ASCII.*

1.7. Αλγόριθμοι και Λογικά διαγράμματα

Η έρευνα και η περιγραφή των αλγόριθμων απασχόλησαν τον άνθρωπο από πολύ παλιά. Είναι αρκετά γνωστός ο Αλγόριθμος του Ευκλείδη για την εύρεση του μέγιστου κοινού διαιρέτη (ΜΚΔ) δύο αριθμών ο οποίος έχει αναφερθεί για πρώτη φορά στα Στοιχεία του Ευκλείδη (300 π.Χ.) και το κόσκινο του Ερατοσθένη.

Η λέξη αλγόριθμος (algorithm) προέρχεται από την εργασία του Πέρση μαθηματικού Abu Ja'far Mohammed ibn Musa al Khowarizmi, που έζησε περί το 825 μ.Χ. Όταν αργότερα μεταφράστηκε στα λατινικά, άρχιζε με τη φράση "Algoritmi dixit..." που σήμαινε "Ο αλγόριθμος λέει .... ". Ο όρος αλγόριθμος σήμαινε κάτι σαν "συστηματική διαδικασία αριθμητικών χειρισμών". Από τα μέσα του 20ού αιώνα βρίσκεται η λέξη αλγόριθμος στην επικαιρότητα λόγω της τεράστιας ανάπτυξης της επιστήμης των υπολογιστών.

*<u>Ορισμός</u>: Λέμε **αλγόριθμο** (algorithm), μια πεπερασμένη σειρά από πράξεις ή ενέργειες αυστηρά καθορισμένων οι οποίες πρέπει να γίνουν σε πεπερασμένο χρόνο για να επιτύχουμε την επίλυση ενός προβλήματος.*

Ένας αλγόριθμος είναι ανεξάρτητος από τη μηχανή (υπολογιστή) όπου θα εκτελεσθεί και από τα ιδιαίτερα χαρακτηριστικά των δεδομένων.

Μπορούμε ακόμη να πούμε ότι η τάξη των πράξεων είναι συνήθως πάρα πολύ μεγάλη και η εκτέλεσή τους γίνεται σταδιακά μέσα στον χρόνο.

Η χρησιμοποίηση των υπολογιστών για την επίλυση σύνθετων προβλημάτων έδωσε μια τεράστια ώθηση στην ανάπτυξη και εφαρμογή των αλγόριθμων. Οι αλγόριθμοι αποτελούν σήμερα μια πολύ βασική περιοχή έρευνας της επιστήμης των υπολογιστών.

Θεωρείται απαραίτητο να οργανώνονται και να διαχειρίζονται σωστά τα δεδομένα έτσι ώστε να επιτυγχάνονται διαδικασίες ταχύτερης επεξεργασίας τους. Η ανάπτυξη ενός αλγορίθμου και η επιλογή των κατάλληλων δομών των δεδομένων πρέπει να συνδυάζονται αφού το ένα συχνά εξαρτάται από τη γνώση του άλλου.

1.7.1. Περιγραφή και χαρακτηριστικά των αλγορίθμων

Η επίλυση ενός προβλήματος σ' ένα σύγχρονο υπολογιστικό περιβάλλον συνήθως περιλαμβάνει:

- την καταγραφή της υπάρχουσας κατάστασης για το πρόβλημα,
- την κατανόηση των ιδιαιτεροτήτων του προβλήματος,
- την αποτύπωση των συνθηκών και προϋποθέσεων για την υλοποίηση του,
- την πρόταση επίλυσης του προβλήματος με τη βοήθεια ενός αλγορίθμου,
- την τελική επίλυση του προβλήματος με τη χρήση των υπολογιστών

Για να λύσουμε ένα πρόβλημα, όσο απλό και αν θεωρείται, μπορούμε να φανταστούμε πολλούς αλγόριθμους. Ποια είναι όμως εκείνα τα κριτήρια τα οποία θα μας επιτρέψουν να αποφανθούμε ότι ένας αλγόριθμος είναι καλλίτερος ή χειρότερος από έναν άλλο;

Συνήθως τα κριτήρια είναι υποκειμενικά και πολλές φορές συνδέονται με το υπολογιστικό σύστημα.

Θα αναφερθούν εδώ τρία κριτήρια, ίσως τα πιο βασικά.

α. Ο χρόνος ο οποίος απαιτείται για να επιτύχουμε το αποτέλεσμα.

β. Ο χώρος τον οποίο καταλαμβάνουμε μέσα στη μηχανή (μνήμη και περιφερειακά).

γ. Ο αριθμός των πράξεων για να βρούμε το αποτέλεσμα.

Παράδειγμα.

Έστω το πολυώνυμο:

$$P(x) = \alpha_n x^n + \alpha_{n-1} x^{n-1} + \ldots\ldots + \alpha_1 x^1 + \alpha_0$$

Αν γνωρίζουμε τις τιμές των συντελεστών α_n , α_{n-1} , . . .α_1 , α_0 *και την τιμή του x, να υπολογιστεί η τιμή P(x) .*

Αλγόριθμος 1. Κάνοντας όλες τις πράξεις θα χρειαστούμε n προσθέσεις και n(n+1)/2 πολλαπλασιασμούς.

Αλγόριθμος 2. Αν υπολογίσουμε το x^i συναρτήσει του x^{i-1} τότε, θα χρειαστούμε μόνο 2n-1 πολλαπλασιασμούς και n προσθέσεις.

Αλγόριθμος 3. Μπορούμε να γράψουμε το πολυώνυμο P(x) σύμφωνα με τον τύπο του Horner ως εξής :

$$P(x) = ((\ldots(\alpha_n x + \alpha_{n-1})x + \alpha_{n-2})x + \ldots + \alpha_2)x + \alpha_1 x + \alpha_0$$

Εδώ ελαττώνουμε τον αριθμό των πολλαπλασιασμών σε n και οι προσθέσεις παραμένουν πάλι n.

Προτού λοιπόν, να καταλήξουμε για να εφαρμόσουμε έναν αλγόριθμο πρέπει να εξετάσουμε τις διάφορες πιθανές μεθόδους και λύσεις για να διαλέξουμε εκείνη οποία θα μας δώσει τον πιο αποτελεσματικό αλγόριθμο.

Πολλές φορές, όταν βρούμε ένα αλγόριθμο για το πρόβλημα το οποίο μας απασχολεί, επειδή μας ικανοποιεί από πρακτική πλευρά, μας εμποδίζει να ψάξουμε για κάτι καλλίτερο και αυτό έχει σαν αποτέλεσμα την άσκοπη χρησιμοποίηση του υπολογιστή.

Ας γίνουμε όμως πιο κατανοητοί μ' ένα άλλο χαρακτηριστικό παράδειγμα.

Εφαρμόζοντας τους κανόνες παιγνιδιού του σκακιού, να υπολογίσουμε τις θέσεις-κινήσεις οι οποίες κερδίζουν.

Υπολογίζεται ότι για 20 πιθανές επιθέσεις ο αριθμός των δυνατών κινήσεων είναι 10^{120}. Αν θεωρήσουμε ότι κάθε μια κίνηση γίνεται ανά 1μsec και 1μsec = 10^{-6} sec θα έχουμε:

$$10^6 * 60 * 60 * 24 * 365 = 10^{14} \mu sec/\text{έτος}$$

Χρειαζόμαστε λοιπόν : $10^{120} / 10^{14} = 10^{106}$ έτη.

Δηλαδή, ένας υπολογιστής πρέπει να δουλεύει 10^{106} έτη για να βρούμε τις θέσεις οι οποίες κερδίζουν !!!

Πρέπει να είμαστε απαισιόδοξοι μπροστά σε τέτοιους αλγόριθμους;

Βέβαια όχι. Σχεδόν πάντα μπορούμε να βρούμε έναν αλγόριθμο ίσως πιο πολύπλοκο, αλλά πιο αποτελεσματικό και πιο γρήγορο, αποφεύγοντας τις περιττές πράξεις και τους άσκοπους υπολογισμούς.

Οι αλγόριθμοι διακρίνονται κυρίως σε :

- *Επαναληπτικούς*, οι οποίοι εφαρμόζονται σε προβλήματα ταξινόμησης, αναζήτησης και απλών δομών, όπως στοίβες, ουρές, λίστες και σωροί
- *Αναδρομικούς*, οι οποίοι εφαρμόζονται σε προβλήματα ταξινόμησης και αριθμητικών πράξεων

Άλλες βασικές κατηγορίες αλγορίθμων είναι οι αλγόριθμοι γράφων, οι αλγόριθμοι δικτύων, οι αλγόριθμοι επεξεργασίας κειμένου (αναγνώρισης κειμένου και προτύπων) κτλ.

Ένας πρακτικός και σύντομος οδηγός εύρεσης και εφαρμογής του κατάλληλου αλγορίθμου για την επίλυση ενός προβλήματος περιλαμβάνει:

1. Απόλυτη κατανόηση της εκφώνησης του προβλήματος
2. Αναζήτηση των υπολογιστικών στοιχείων που σχετίζονται με το πρόβλημα
3. Αναζήτηση μη υπολογιστικών στοιχείων από σχετικά πεδία άλλων επιστημών π.χ. Φυσικής, Μαθηματικών, Γεωγραφίας κ.λπ.
4. Διατύπωση με απλά λόγια του προβλήματος και απεικόνιση της κατάστασης σε μορφή σχεδιαγράμματος όπου θα αναφέρονται τα αντικείμενα και οι χρόνοι.
5. Αν έχετε επιλύσει παρόμοιο πρόβλημα επωφεληθείτε από την εμπειρία σας και προσπαθήστε να την εφαρμόσετε στο νέο πρόβλημα που θέλετε να λύσετε
6. Διερευνήστε προσεκτικά τα δεδομένα ή τους πόρους που παρέχονται

7. Καθορίστε επακριβώς τα δεδομένα και/ή τα αποτελέσματα που αναμένονται

1.7.2. Λογικό διάγραμμα

Όταν σχεδιάζουμε να γράψουμε κάποιο σημαντικό κείμενο αρχίζουμε πρώτα να γράφουμε περιληπτικά το πλάνο εργασίας όπου συγκεντρώνουμε τις κυριότερες ιδέες, καθώς και τη σειρά με την οποία θα αναπτυχθούν δηλαδή, πρώτα ο πρόλογος, μετά το κυρίως θέμα και τέλος ο επίλογος.

Ένα τέτοιο πλάνο εργασίας στον προγραμματισμό θα το λέμε *λογικό διάγραμμα* (Block diagram ή Flowchart).

Σαν παράδειγμα θα παρουσιαστεί το λογικό διάγραμμα του αλγόριθμου του Ευκλείδη για να βρούμε το Μέγιστο Κοινό Διαιρέτη (ΜΚΔ) δύο αριθμών, έστω m και n.

Ο αλγόριθμος του Ευκλείδη λέει ότι ο ΜΚΔ των δύο αριθμών θα είναι ο παρανομαστής της διαίρεσης του μεγαλύτερου με το μικρότερο, με υπόλοιπο μηδέν.

Έχουμε λοιπόν:

1. Έστω $m, n > 0$ ($m > n$)
2. Διαιρούμε το m με το n έτσι ώστε r να είναι το υπόλοιπο της διαίρεσης.
3. Αν $r = 0$ τότε ο ΜΚΔ των m και n θα είναι ο n και εκεί τελειώνει ο αλγόριθμος.
4. Αν $r \neq 0$ τότε δίνουμε στο m την τιμή του n και στο n την τιμή του r και ξαναρχίζουμε από το στάδιο 2.

Βλέπουμε ότι ένας αλγόριθμος παρουσιάζεται σαν μια πεπερασμένη σειρά από στάδια ή βήματα. Σε κάθε βήμα πραγματοποιούμε είτε μια απλή αριθμητική πράξη, είτε μια πράξη αντικατάστασης, είτε ένα τεστ (σύγκριση).

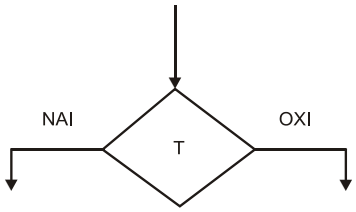
Για ν' αποφύγουμε τις γλωσσικές αμφισβητήσεις και για να έχουμε μια ενιαία αντιμετώπιση από όλους τους προγραμματιστές, χρησιμοποιούνται απλά και καθορισμένα σύμβολα για να την παράσταση των διάφορων σταδίων ενός αλγόριθμου.

Κυρίως χρησιμοποιούμε έξι σύμβολα.

α. Τα βέλη.

Τα βέλη δείχνουν τη φορά της διαδοχής των βημάτων.

β. Τα ρομβοειδή "κουτιά" με μια είσοδο και δύο εξόδους.

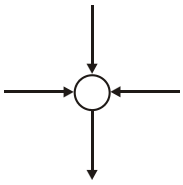


Τα ρομβοειδή "κουτιά" χρησιμεύουν όταν έχουμε μια πράξη απόφασης (τεστ). Αν πραγματοποιείται (επαληθεύεται) η συνθήκη Τ τότε, ακολουθούμε τη φορά του βέλους το οποίο γράφει ΝΑΙ, αν δεν πραγματοποιείται η συνθήκη Τ τότε, ακολουθούμε τη φορά του βέλους το οποίο γράφει ΟΧΙ.

γ. Τα "ορθογώνια κουτιά" με μια είσοδο και μια έξοδο.

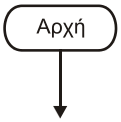
Παριστάνουν μια ή περισσότερες αριθμητικές πράξεις ή πράξεις αντικατάστασης (Μεταβολή των τιμών των μεταβλητών του προβλήματος).

δ. Οι κόμβοι ή ενώσεις με δύο ή περισσότερες εισόδους και μόνο μία έξοδο.



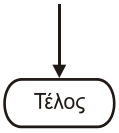
Χρησιμεύουν σαν σημείο σύνδεσης δύο ή περισσότερων λογικών δρόμων. Αυτές οι ενώσεις μας δεσμεύουν έτσι ώστε τα κουτιά με μία ή δύο εξόδους να δέχονται πάντα μία μόνο είσοδο.

ε. Η αρχή του αλγόριθμου



Συνήθως θα εννοούνται οι αρχικές τιμές του προβλήματος.

στ. Το τέλος του αλγόριθμου.



Συχνά θα ισοδυναμεί και με την εκτύπωση ή εμφάνιση στην οθόνη των αποτελεσμάτων της επεξεργασίας του προβλήματος.

Ο Αλγόριθμος του Ευκλείδη χρησιμοποιώντας τα πιο πάνω σύμβολα μας δίνει το λογικό διάγραμμα της εικόνας 1.9.

Εφαρμόζοντας τον αλγόριθμο του Ευκλείδη στους αριθμούς **297** και **63** θα έχουμε:

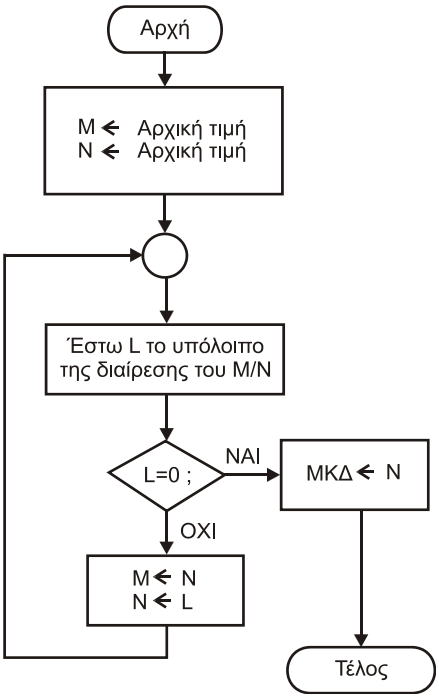
1. M ← 297, N ← 63
2. M/N ↔ 297/63 = 4 + **45**/63 άρα L = **45**
3. L = 0 ; ΟΧΙ τότε, M ← 63, N ← 45 και επιστροφή στο 2.

2. M/N ↔ 63/45 = 1 + **18**/45 άρα L = **18**
3. L = 0 ; ΟΧΙ τότε, M ← 45, N ←18 και επιστροφή στο 2.

2. M/N ↔ 45/18 = 2 + **9**/17 άρα L = **9**
3. L = 0 ; ΟΧΙ τότε, M ←18, N ← 9 και επιστροφή στο 2.

2. M/N ↔ 18/9 = 2 + **0** άρα L = **0**
3. L = 0 ; ΝΑΙ (**ΤΕΛΟΣ**)

Άρα Μ.Κ.Δ. ← 9



Εικόνα 1.9. *Λογικό διάγραμμα του αλγόριθμου του Ευκλείδη για να βρούμε τον ΜΚΔ δύο αριθμών.*

Βλέπουμε ότι μετά το τρίτο βήμα (M ← 63, N ← 45), επαναλαμβάνουμε τις ίδιες πράξεις. Εκείνο το οποίο αλλάζει είναι οι τιμές των μεταβλητών.

Αυτή η επανάληψη λέγεται ανακύκλωση ή μπούκλα ή και βρόγχος (loop).

Πολλές φορές ένας προγραμματιστής παραμελεί τη φάση της κατασκευής του λογικού διαγράμματος. Το λογικό διάγραμμα βοηθά για να αποφύγουμε τις περιπτώσεις οι οποίες δεν είναι προφανείς και επιτρέπει τη διερεύνηση του προβλήματος μέχρι και στην πιο μικρή λεπτομέρεια.

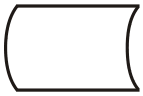
***<u>Σημείωση</u>*:** *Μια μεταβλητή M έχει ένα **όνομα** και μια **τιμή**. Μια πράξη αντικατάστασης αλλάζει την τιμή της μεταβλητής με μία νέα τιμή. Όταν χρησιμοποιούμε τον συμβολισμό **M ← N** , αυτό σημαίνει ότι η τιμή της μεταβλητής M θα αντικατασταθεί από την τιμή της μεταβλητής N. Η τιμή της μεταβλητής M χάνεται οριστικά. Έτσι **M ← M+1** θα σημαίνει: Στην τιμή της μεταβλητής M πρόσθεσε 1 και δώσε αυτή τη **νέα τιμή** στη μεταβλητή M.*

Η κατασκευή του λογικού διαγράμματος δεν παρουσιάζει δυσκολίες για έναν προγραμματιστή με μια σχετική πείρα. Πρέπει όμως να αποτελεί για τους αρχάριους στον προγραμματισμό το πρώτο στάδιο το οποίο είναι απαραίτητο για την ολοκληρωμένη εκμάθηση του προγραμματισμού.

Επειδή τα έξι βασικά σύμβολα δεν είναι τα μοναδικά, θα αναφέρουμε εδώ και μερικά ακόμη βοηθητικά σύμβολα τα οποία χρησιμοποιούνται συχνά για πιο παραστατική απεικόνιση των πράξεων του λογικού διαγράμματος.

α. Εκτύπωση των αποτελεσμάτων σε χαρτί.

β. Εγγραφή των αποτελεσμάτων σε μαγνητικό ή οπτικό δίσκο ή διάβασμα τιμών από δίσκο.



Πολλοί συγγραφείς βιβλίων προγραμματισμού υπολογιστών, εκτός από αυτά τα βασικά σύμβολα, χρησιμοποιούν και πολλά άλλα σύμβολα τα οποία συνήθως διαφέρουν πολύ λίγο μεταξύ τους. Συνήθως, στο τέλος κάθε βιβλίου με τη μορφή παραρτήματος, υπάρχει διαθέσιμος ένας επεξηγηματικός πίνακας όλων αυτών των συμβόλων, γιατί τις περισσότερες φορές ο ορισμός τους είναι αυθαίρετος αφού δεν έχει γίνει μέχρι σήμερα καμιά προσπάθεια τυποποίησής τους.

Φυσικά, υπάρχουν και πολλά βοηθήματα, δηλαδή έτοιμα εμπορικά προγράμματα ή προγράμματα ελεύθερης χρήσης, τα οποία επιτρέπουν τη γρήγορη σχεδίαση των λογικών διαγραμμάτων με την ενσωμάτωση στις βιβλιοθήκες τους προσχεδιασμένων όλων των συμβόλων τα οποία χρειάζονται για την αποτελεσματική απεικόνιση των αλγορίθμων επίλυσης των προβλημάτων. Κλασικό παράδειγμα είναι το πρόγραμμα VISIO της Microsoft.

Άλλο ένα τυπικό πρόγραμμα για διαγράμματα ροής είναι το **Dia**, ανοιχτού κώδικα, που παρέχεται από τη διεύθυνση:

https://wiki.gnome.org/action/show/Apps/Dia?action=show&redirect=Dia

Είναι ιδιαίτερα εύκολο στη χρήση, πολύ ελαφρύ και με αρκετές δυνατότητες. Επίσης, παρέχει από την αρχή βιβλιοθήκη σχημάτων για διεργασίες Μηχανικών. Το Dia είναι απολύτως δωρεάν, αποτελεί μέρος του GNOME Office project, τρέχει τόσο σε Windows όσο και σε Linux και σε MacOS.

Ένας αλγόριθμος όταν περιγραφεί με μια μορφή την οποία δέχεται ένας υπολογιστής, λέγεται ***πρόγραμμα*** (program).

Πρέπει λοιπόν, να χρησιμοποιήσουμε μια νέα "γλώσσα" για να μεταφράσουμε τον αλγόριθμο σ' εκείνη τη μορφή η οποία θα είναι κατανοητή από τον υπολογιστή. Το ιδανικό θα ήταν να μπορούσαμε να χρησιμοποιήσουμε τη μητρική μας γλώσσα. Δυστυχώς, για να μπορέσουμε να μεταφράσουμε έναν αλγόριθμο στη γλώσσα της μηχανής (δυαδική μορφή) θα πρέπει να ικανοποιούνται τρία αυστηρά κριτήρια.

Δηλαδή, η μετάφραση πρέπει να είναι:

- Ακριβής
- Χωρίς διφορούμενες εκφράσεις

- Χωρίς περιττά λόγια

Επειδή καμιά από τις μητρικές μας γλώσσες δεν μπορεί να ικανοποιήσει αυτά τα τρία κριτήρια για το λόγο αυτό δημιουργήθηκαν (από το 1950 και μετά) καινούργιες γλώσσες οι οποίες ικανοποιούν αυτά τα τρία κριτήρια και μπορούν να χρησιμοποιηθούν εύκολα από όλους εκείνους οι οποίοι θέλουν να γράψουν ένα πρόγραμμα.

Αυτές οι νέες γλώσσες λέγονται *γλώσσες προγραμματισμού υπολογιστών.* Μοιάζουν αρκετά με τις μητρικές γλώσσες (κυρίως με τα Αγγλικά) και μπορούν εύκολα να γίνουν κατανοητές.

1.7.3. Παραδείγματα

Για να καταστεί περισσότερο κατανοητή η διαδικασία της σύλληψης και κατασκευής του λογικού διαγράμματος από τη φραστική διατύπωση του αλγόριθμου του προβλήματος, θα γίνει μια απόπειρα να παρουσιαστούν πέντε χαρακτηριστικά παραδείγματα λογικών διαγραμμάτων με αυξανόμενο βαθμό δυσκολίας.

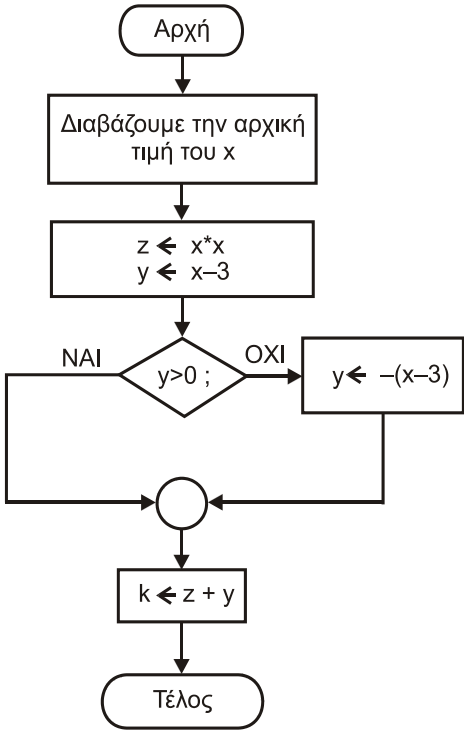
Τα παραδείγματα αυτά αποτελούν και υποδείγματα συγγραφής προγραμμάτων στα επόμενα κεφάλαια.

Η παράλληλη μελέτη

Αλγόριθμου - λογικού διαγράμματος - προγράμματος

μπορεί να βοηθήσει σε κάθε περίπτωση.

Κυρίως είναι απαραίτητη πρώτα η σύγκριση του λογικού διαγράμματος με τον προτεινόμενο αλγόριθμο και στη συνέχεια η σύγκριση του λογικού διαγράμματος με το πρόγραμμα για τον εντοπισμό των πιθανών λογικών λαθών.



Εικόνα 1.10. *Λογικό διάγραμμα του παραδείγματος α.*

1.7.4. Παράδειγμα α.

Εκφώνηση του προβλήματος.

Να υπολογιστεί η τιμή της παράστασης : $x^2 + |x - 3|$ *, για μια καθορισμένη τιμή του* x.

Αλγόριθμος

1 Διαβάζουμε την τιμή του x

2. Υπολογίζουμε την τιμή του x^2 (έστω z)

3. Υπολογίζουμε την τιμή x-3 (έστω y)

4. Ελέγχουμε το αποτέλεσμα (την τιμή του y)

5. Αν είναι θετικός αριθμός, πηγαίνουμε στο 7, διαφορετικά πάμε στο 6

6. Αλλάζουμε το πρόσημο του y

7 Προσθέτουμε το z και το y (έστω k) το οποίοι είναι και η τιμή της παράστασης

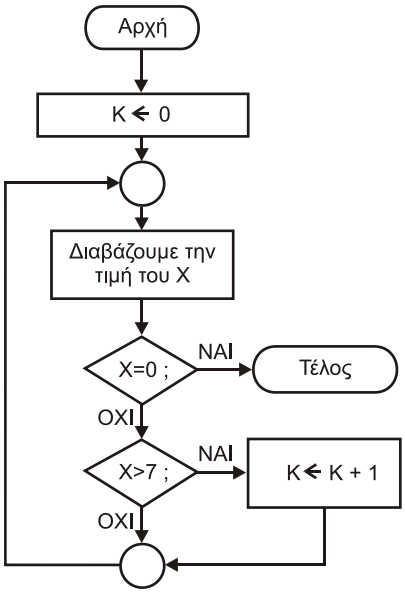
Το λογικό διάγραμμα αυτού του αλγόριθμου εμφανίζεται στην εικόνα 1.10.

1.7.5. Παράδειγμα β.

Εκφώνηση του προβλήματος.

Έστω ότι θέλουμε να βρούμε ανάμεσα από πολλούς αγνώστου πλήθους, μονοψήφιους και διάφορους του μηδενός αριθμούς τους οποίους πληκτρολογούμε, πόσοι είναι μεγαλύτεροι από τον αριθμό 7.

Υπόδειξη*, Επειδή δεν γνωρίζουμε το ακριβές πλήθος των αριθμών πρέπει να πληκτρολογήσουμε, στο τέλος, ένα ακόμη συμβατικό αριθμό για να υποδείξουμε τον τελευταίο αριθμό έτσι ώστε να σταματήσουν οι πράξεις όταν συναντηθεί αυτή η τιμή. Στο πρόβλημα αυτό ο συμβατικός αριθμός μπορεί να είναι το μηδέν επειδή βρίσκεται εκτός του πεδίου τιμών των αριθμών τους οποίους πληκτρολογούμε.*



Εικόνα 1.11. *Λογικό διάγραμμα του παραδείγματος β.*

Αλγόριθμος

1. Κατ' αρχήν μηδενίζουμε ένα δείκτη K, ο οποίος θα είναι ο δείκτης του πλήθους των αριθμών οι οποίοι είναι μεγαλύτεροι από το 7.
2. Διαβάζουμε έναν αριθμό, έστω X
3. Εξετάζουμε αν είναι ίσος με το μηδέν

   α. Αν X = 0 τότε τελειώσαμε

   β. Αν X $\neq$ 0 τότε πηγαίνουμε στο 4
4. Εξετάζουμε αν το X είναι μεγαλύτερο από το 7.

   α. Αν X >7 τότε αυξάνουμε το δείκτη K κατά 1 και γυρίζουμε στο 2.

   β. Αν X = 7 τότε γυρίζουμε στο 2

Το λογικό διάγραμμα αυτού του αλγόριθμου εμφανίζεται στην εικόνα 1.11.

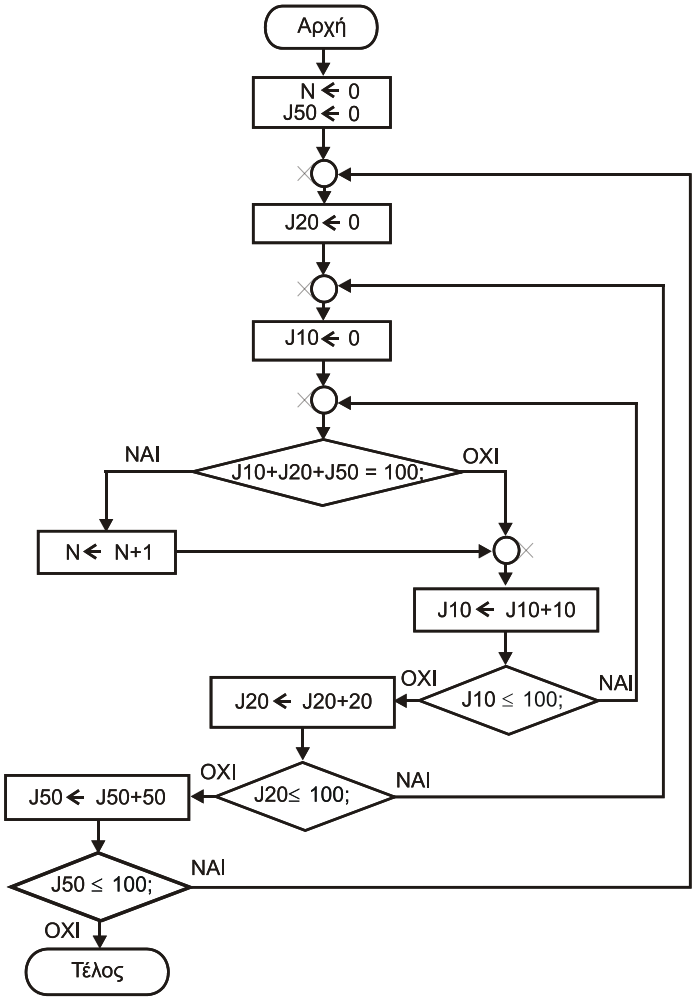
1.7.6. Παράδειγμα γ.

Εκφώνηση του προβλήματος.

Να ευρεθούν πόσοι είναι όλοι οι δυνατοί συνδυασμοί με τους οποίους μπορεί να σχηματιστεί ένα ευρώ όταν διαθέτουμε ένα μεγάλο αριθμό (άπειρο) από κέρματα των δέκα, είκοσι και πενήντα λεπτών.

Αλγόριθμος.

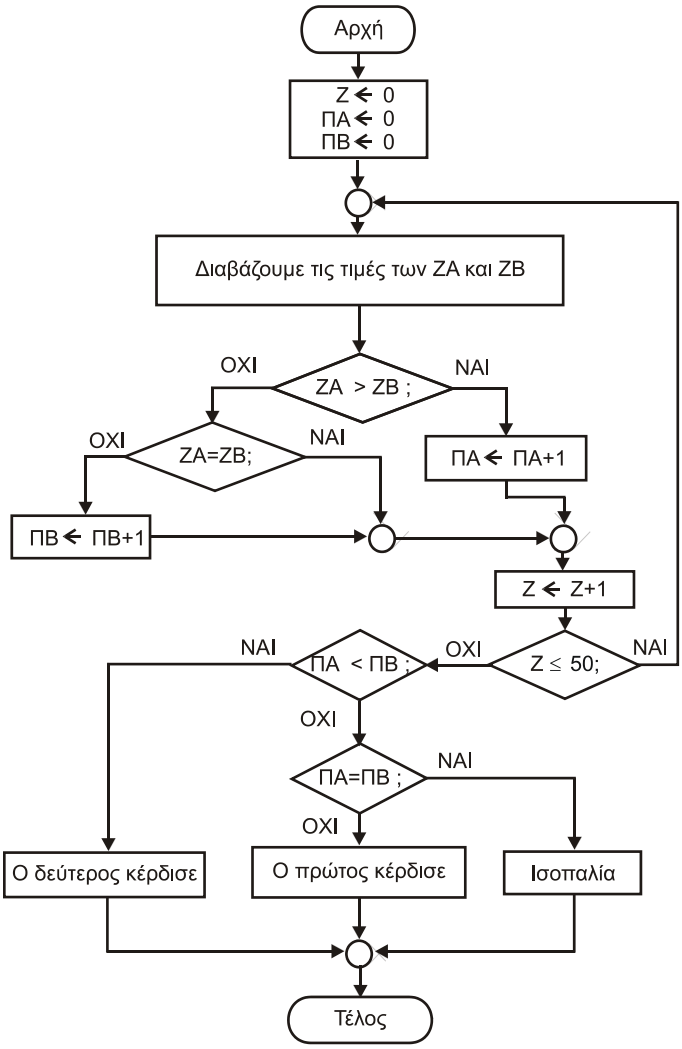
1. Μηδενίζουμε το δείκτη N, ο οποίος θα εκφράζει το πλήθος των συνδυασμών.
2. Μηδενίζουμε άλλους τρεις δείκτες: Το δείκτη J10 για τα δέκα λεπτά, το δείκτη J20 για τα είκοσι λεπτά και το δείκτη J50 για τα πενήντα λεπτά.
3. Ελέγχουμε αν το άθροισμα των τριών δεικτών J10, J20 και J50 είναι ίσο με 100, δηλαδή, αν σχηματίζουν ένα ευρώ.
   α. Αν είναι ίσο με 100 τότε, αυξάνουμε το δείκτη N κατά 1. Δηλαδή, έχουμε βρει ένα αναζητούμενο συνδυασμό και επομένως προσθέτουμε ένα στο πλήθος των συνδυασμών.
   β. Αν δεν είναι ίσο με 100 τότε, πηγαίνουμε στο 4.
4. Αυξάνουμε το δείκτη J10 κατά δέκα, γιατί ο δείκτης J10 εκφράζει τα δέκα λεπτά.
5. Ελέγχουμε αν ο δείκτης J10 είναι μικρότερος ή ίσος με το 100.
   α. Αν J10 $\leq$ 100 τότε πηγαίνουμε στο 3.
   β Αν το J10 γίνει μεγαλύτερο από το 100 τότε πηγαίνουμε στο 6.
6. Αυξάνουμε το δείκτη J20 κατά είκοσι, αφού ο δείκτης J20 εκφράζει τα είκοσι λεπτά.
7. Ελέγχουμε αν ο δείκτης J20 είναι μικρότερος ή ίσος με το 100.
   α. Αν J20 $\leq$ 100 τότε πηγαίνουμε πίσω στο 3, αφού πρώτα μηδενίσουμε το δείκτη J10. Η τιμή του δείκτη J10 πρέπει να μηδενιστεί εδώ, γιατί θα πρέπει να δοκιμάζουμε την τιμή του αθροίσματος για μια νέα τιμή του δείκτη J20 και όλους τους συνδυασμούς του δείκτη J10.
   β. Αν J20 > 100, αυτό σημαίνει ότι εξαντλήσαμε όλες τις τιμές για τα είκοσι λεπτά πηγαίνουμε στο 8.
8. Αυξάνουμε το δείκτη J50 κατά 50, αφού ο δείκτης J50 εκφράζει τα πενήντα λεπτά.



Εικόνα 1.12. *Λογικό διάγραμμα του παραδείγματος γ.*

9. Ελέγχουμε αν ο δείκτης J50 είναι μικρότερος ή ίσος με το 100.
   α. Αν J50 > 100 αυτό σημαίνει ότι τελειώσαμε όλους τους δυνατούς συνδυασμούς, πηγαίνουμε στο τέλος.
   β. Αν J50 ≤ 100 τότε πηγαίνουμε στο 3, αφού μηδενίσουμε τους δείκτες J20 και J10. Ο μηδενισμός των δεικτών J10 και J20 σημαίνει ότι, ξαναρχίζουμε από την αρχή για μια νέα τιμή του J50 και όλους τους δυνατούς συνδυασμούς με τους δείκτες J10 και J20 για να πετύχουμε άθροισμα ίσο με 100.

Το λογικό διάγραμμα αυτού του αλγόριθμου εμφανίζεται στην εικόνα 1.12.



Εικόνα 1.13. *Λογικό διάγραμμα του παραδείγματος δ.*

1.7.7. Παράδειγμα δ.

Εκφώνηση του προβλήματος.

Δύο παίκτες ρίχνουν από ένα ζάρι. Εκείνος ο οποίος θα φέρει μεγαλύτερο νούμερο κερδίζει ένα πόντο. Αν φέρουν το ίδιο νούμερο ξαναρίχνουν. Αυτό επαναλαμβάνεται 50 φορές. Νικητής είναι εκείνος ο οποίος στο τέλος θα έχει σημειώσει τους περισσότερους πόντους. Φυσικά δεν πρέπει ν' αποκλειστεί η περίπτωση της ισοπαλίας.

Αλγόριθμος

1. Μηδενίζουμε τρεις δείκτες:
   α. Το Ζ το οποίο θα εκφράζει πόσες ζαριές έχουν ριχτεί.
   β. Το ΠΑ το οποίο θα εκφράζει τους πόντους του Α’ παίκτη.
   γ. Το ΠΒ το οποίο θα εκφράζει τους πόντους του Β' παίκτη.
2. Διαβάζουμε τις τιμές τις οποίες έχουν τα ζάρια των δύο παικτών, έστω ΖΑ, ΖΒ.
3. Εξετάζουμε ποια τιμή είναι μεγαλύτερη.

α. Αν ΖΑ > ΖΒ τότε αυξάνουμε το δείκτη ΠΑ κατά ένα (1) και πηγαίνουμε στο 5.

β. Αν ΖΑ < ΖΒ τότε πηγαίνουμε στο 4.

4. Εξετάζουμε ξανά τις τιμές των δεικτών ΖΑ και ΖΒ.

   α. Αν ΖΑ = ΖΒ αυτό σημαίνει ότι έχουμε την ίδια ζαριά και πηγαίνουμε στο 5.

   β. Αν ΖΒ ≠ ΖΑ τότε, αυξάνουμε το δείκτη ΠΒ κατά ένα (1) και πηγαίνουμε στο 5.

5. Αυξάνουμε κατά ένα (1) το δείκτη Ζ.

6. Εξετάζουμε αν ο δείκτης Ζ έφτασε το 50.

   α. Αν Ζ ≤ 50 πηγαίνουμε στο 2

   β. Αν Ζ >50 πηγαίνουμε στο 7

7. Εξετάζουμε τους δείκτες ΠΑ και ΠΒ.

   α. Αν ΠΑ < ΠΒ τότε, κέρδισε ο δεύτερος παίκτης.

   β. Αν ΠΑ = ΠΒ τότε, έχουμε ισοπαλία.

   γ. Αν ΠΑ > ΠΒ τότε, κέρδισε ο πρώτος παίκτης.

Το λογικό διάγραμμα αυτού του αλγόριθμου εμφανίζεται στην εικόνα 1.13.

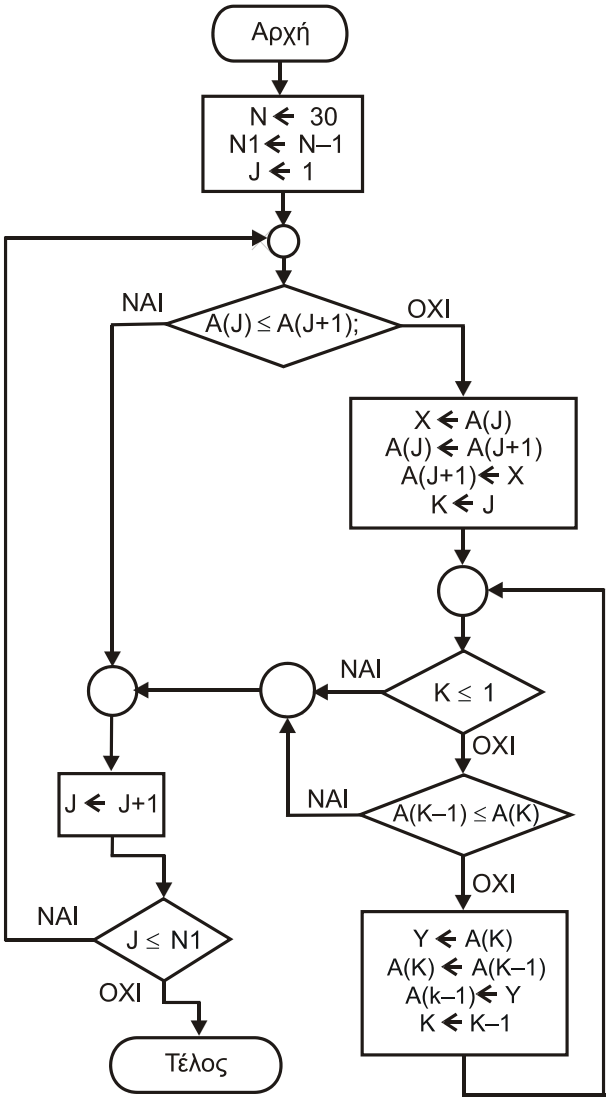
1.7.8. Παράδειγμα ε.

Εκφώνηση του προβλήματος.

Έστω ένας μονοδιάστατος πίνακας Α, με 30 στοιχεία. Να διαταχθούν τα στοιχεία του κατά αύξουσα σειρά μεγέθους με τη μέθοδο της φυσαλίδας. (Περισσσότερα στη διεύθυνση: https://en.wikipedia.org/wiki/Bubble_sort).

Αλγόριθμος

Η μέθοδος της φυσαλίδας (BUBBLE sort) για την ταξινόμηση των στοιχείων ενός πίνακα μας δίνει τον ακόλουθο αλγόριθμο:



Εικόνα 1.14. Λογικό διάγραμμα του παραδείγματος ε.

1. Συγκρίνουμε το στοιχείο Α(1) με το Α(2)
   - Αν Α(1) >Α(2) αλλάζουμε τη θέση τους
   - Αν Α(1) ≤ Α(2) συνεχίζουμε στο 2
2. Συγκρίνουμε το Α(2) με το Α(3)
   - Αν Α(2) ≤ Α(3) συνεχίζουμε στο 3
   - Αν Α(2) > Α(3) αλλάζουμε αμοιβαία το Α(3) με το Α(2) και στη συνέχεια συγκρίνουμε το Α(2) με το Α(1), όπως στην 1
3. Γενικεύοντας, συγκρίνουμε τα Α(Ι) και Α(Ι+1)

   α. Αν Α(Ι) ≤ Α(Ι+1) τότε αυξάνουμε το Ι κατά 1 (Ι=Ι+1) και συνεχίζουμε στο 3, μέχρις ότου γίνει Ι=29 δηλαδή, ότι τελειώσαμε.

   β. Αν Α(Ι) >Α(Ι+1) τότε αλλάζουμε το Α(Ι) με το Α(Ι+1) βάζουμε Κ = Ι

4. Συγκρίνουμε το A(K-1) με το A(K)
   - Αν A(K) < A(K-1) αλλάζουμε τη θέση τους, κάνουμε K = K-1 και συνεχίζουμε στο 4 μέχρις ότου K>1, οπότε συνεχίζουμε στο 3
   - Αν A(K) ≥ A(K-1) τότε συνεχίζουμε πίσω στο 3

   Ο συνολικός αριθμός επαναλήψεων για ***n*** στοιχεία είναι:

$$\sum_{i=1}^{n}(n-i) = \sum_{i=1}^{n} n - \sum_{i=1}^{n} i =$$

$$n^2 - \frac{n(n+1)}{2} = n^2 - \frac{n^2}{2} - \frac{n}{2} = \frac{n^2-n}{2}$$

Επομένως, ο χρόνος εκτέλεσης του αλγόριθμου ταξινόμησης φυσαλίδας είναι της τάξεως $\boldsymbol{n^2}$.

Το λογικό διάγραμμα αυτού του αλγόριθμου εμφανίζεται στην εικόνα 1.14.

1.8. Λογικά λάθη προγραμμάτων

Το αντικείμενο με το οποίο θα ασχοληθούμε σ' αυτό το βιβλίο είναι σαφώς προσδιορισμένο. Θα μάθουμε να γράφουμε σωστά και γρήγορα ένα πρόγραμμα. Οι φάσεις της μεταγλώττισης της φόρτωσης και της εκτέλεσης του προγράμματος συνήθως γίνονται αυτόματα.

Έτσι, ο υπολογιστής θα εκτελέσει όλες τις πράξεις ενός προγράμματος χωρίς να μπορεί να κάνει έλεγχο της λογικής ορθότητας του προγράμματος. Τα λανθασμένα αποτελέσματα ενός προγράμματος το οποίο δεν έχει υποστεί έλεγχο των αποτελεσμάτων του, οφείλονται αποκλειστικά και μόνο σε λάθη του προγραμματιστή και λέγονται λογικά λάθη (logical errors).

Τα λογικά λάθη ευθύνονται για τα λανθασμένα αποτελέσματα ενός προγράμματος. Τα λογικά λάθη μπορούν να βρεθούν μόνο από ένα πεπειραμένο προγραμματιστή με τον υπομονετικό και χρονοβόρο έλεγχο όλου του προγράμματος (debugging) και με παράλληλη σύγκριση του λογικού διαγράμματος δηλαδή, της λύσης του προβλήματος, γι' αυτό και κατά τη φάση της ανάπτυξη ενός προγράμματος πρέπει να καταβληθεί μεγάλη προσπάθεια για τον έλεγχο των αποτελεσμάτων πριν δοθεί το πρόγραμμα σε ευρεία χρήση.

Έλεγχο των αποτελεσμάτων σημαίνει ότι πρέπει να "τρέξει" (**run**) το πρόγραμμα πολλές φορές και για διαφορετικά σύνολα αρχικών ή ενδιάμεσων τιμών και να γίνει προσεκτικός έλεγχος όχι μόνο των τελικών αποτελεσμάτων, αλλά και πολλών ενδιάμεσων αποτελεσμάτων.

Ο κλασσικότερος τρόπος ελέγχου είναι να γίνει έλεγχος για όλες τις ακραίες αρχικές τιμές, εκεί όπου συνήθως παρατηρούνται αποκλίσεις και διαφορετικά αποτελέσματα λόγω παραλήψεων.

Η έκφραση "τρέχω" ένα πρόγραμμα στον υπολογιστή (**run a program**) σημαίνει ότι ο υπολογιστής ασχολείται με την εκτέλεση των εντολών του προγράμματος δηλαδή, των πράξεων .

Ένας άλλος σοβαρός λόγος εξ αιτίας του οποίου προκαλούνται λογικά λάθη είναι η λανθασμένη μεταφορά των συμβόλων των πράξεων (π.χ. αντί για το σύμβολο της πρόσθεσης να τοποθετηθεί το σύμβολο της αφαίρεσης) ή ακόμη η αντικατάσταση του γράμματος Ο (όμικρον) από τον αριθμό 0 (μηδέν).

Λόγω της πολυπλοκότητας την οποία παρουσιάζει συνήθως ένα πρόγραμμα, ακόμη και μικρού μεγέθους, είναι πολύ δύσκολο να εντοπιστούν αυτόματα όλα τα λογικά λάθη γι' αυτό και πολύ συχνά εμφανίζονται νέες εκδόσεις (new versions) των προγραμμάτων όπου σε κάθε νέα

έκδοση διορθώνονται τα λάθη τα οποία έχουν εντοπιστεί από τους διάφορους χρήστες του προγράμματος, αλλά παράλληλα εμπλουτίζονται και εκσυγχρονίζονται με νέες δυνατότητες και επιλογές (upgrades).

Το ίδιο ισχύει και για τους μεταφραστές των γλωσσών προγραμματισμού (compilers) τους οποίους συνεχώς βελτιώνουν και συμπληρώνουν οι κατασκευαστές (Microsoft, Intel, Sun κτλ) ή και οι ανεξάρτητοι οργανισμοί ανοικτού κώδικα (GNU).

Ένα πρόγραμμα γραμμένο και δοκιμασμένο σε μία παλαιότερη έκδοση μιας κανονικοποιημένης γλώσσας προγραμματισμού, όπως είναι η FORTRAN συνήθως, μεταφράζεται και τρέχει και σε κάθε νεώτερη έκδοση ενώ δεν συμβαίνει πάντα το αντίθετο.

Σημαντική παρατήρηση. *Όταν παρατηρούμε ότι το πρόγραμμα δίνει αποτελέσματα, αλλά οι τιμές δεν είναι οι αναμενόμενες αυτό σημαίνει ότι, το πρόγραμμα για τον υπολογιστή δεν παρουσιάζει κανένα πρόβλημα σύνταξης ή εκτέλεσης. Υπεύθυνος για τα λανθασμένα αποτελέσματα του προγράμματος είναι ο προγραμματιστής ο οποίος δεν έχει κάνει σωστά τη μεταφορά του προβλήματος στον υπολογιστή. Χρειάζεται υπομονή και πείρα για τη διόρθωση των λαθών.*

1.9. Ανακεφαλαίωση

Ένα πρόγραμμα το οποίο έχει γραφεί σύμφωνα με τους ορθογραφικούς και συντακτικούς κανόνες της γλώσσας, ακολουθεί υποχρεωτικά τέσσερεις διακεκριμένες φάσεις:

1. Δημιουργία (*συγγραφή*) του προγράμματος (*πηγαίος κώδικας*)
2. Μεταγλώττιση του προγράμματος (*μετατροπή σε αντικειμενικό κώδικα*) με την επίκληση ενός *μεταγλωττιστή* (compiler)
3. Σύνδεση του προγράμματος με τις κατάλληλες συναρτήσεις βιβλιοθήκης (*δημιουργία του εκτελέσιμου κώδικα*)
4. Εκτέλεση του προγράμματος

Στο βιβλίο αυτό θα ασχοληθούμε <u>αποκλειστικά και μόνο</u> με την πρώτη φάση δηλαδή, με τον τρόπο της ορθής συγγραφής του πηγαίου κώδικα.

Θα μάθουμε να γράφουμε σωστά και αποδοτικά ένα πρόγραμμα. Οι επόμενες τρεις φάσεις δηλαδή, της *μεταγλώττισης* (Compilation), της *σύνδεσης* (Link) και της *εκτέλεσης* (Run) ενός προγράμματος διευκολύνονται από το ολοκληρωμένο περιβάλλον της εγκατάστασης της γλώσσας, το οποίο χρησιμοποιούμε, και το λειτουργικό σύστημα.

Παράλληλα με την προσεκτική μελέτη του βιβλίου πρέπει να δοκιμάσετε όλα τα παραδείγματα (όσο απλά και αν φαίνονται !). Όταν το επίπεδο των γνώσεων γίνει ικανοποιητικό συνιστάται η αναζήτηση και επίλυση αγνώστων ασκήσεων και προβλημάτων. Στον προγραμματισμό των υπολογιστών όσα παραδείγματα και να μελετήσετε πάντα θα υπάρχει κάποια άγνωστη λεπτομέρεια.

Δεν υπάρχει καλός ή κακός προγραμματισμός ούτε καλός ή κακός προγραμματιστής. Χρειάζεται μόνο υπομονή και εξάσκηση. Η εμπειρία θα προκύψει κατά τη διαδρομή. Ξεκινήστε με τα απλά και μικρά παραδείγματα του βιβλίου και σταδιακά προσπαθήστε να κατανοήσετε τα δυσκολότερα.

Μην απελπίζεστε όμως! Πειραματιστείτε εντατικά, όχι μόνο με μολύβι και χαρτί αλλά με τη συγγραφή νέου κώδικα με τη βοήθεια ενός σύγχρονου περιβάλλοντος εργασίας και αφήστε τον έλεγχο στον υπολογιστή.

Για την ανάπτυξη και δοκιμή των προγραμμάτων, μπορεί να χρησιμοποιηθεί το ολοκληρωμένο περιβάλλον ***Code::Blocks,*** το οποίο μπορεί να αναζητηθεί στη διεύθυνση:

http://www.codeblocks.org/ καθώς και το πλήρες περιβάλλον ***Dev-C++*** το οποίο μπορεί να αναζητηθεί στη διεύθυνση: http://www.bloodshed.net/dev/devcpp.html.

Και τα δύο αυτά ολοκληρωμένα περιβάλλοντα είναι ανοικτού κώδικα (open source) και προσφέρονται εντελώς δωρεάν από το διαδίκτυο.

Επίσης, μπορεί να χρησιμοποιηθεί το περιβάλλον ανάπτυξης και δοκιμής προγραμμάτων σε περιβάλλον του λειτουργικού συστήματος Windows σε γλώσσα C, με τη βοήθεια ενός διερμηνευτή (interpreter). Η φοιτητική/μαθητική έκδοση του προγράμματος (**Ch Student Edition)** διατίθεται δωρεάν μετά από εγγραφή, στη διεύθυνση:

http://www.softintegration.com/products/chstudent/

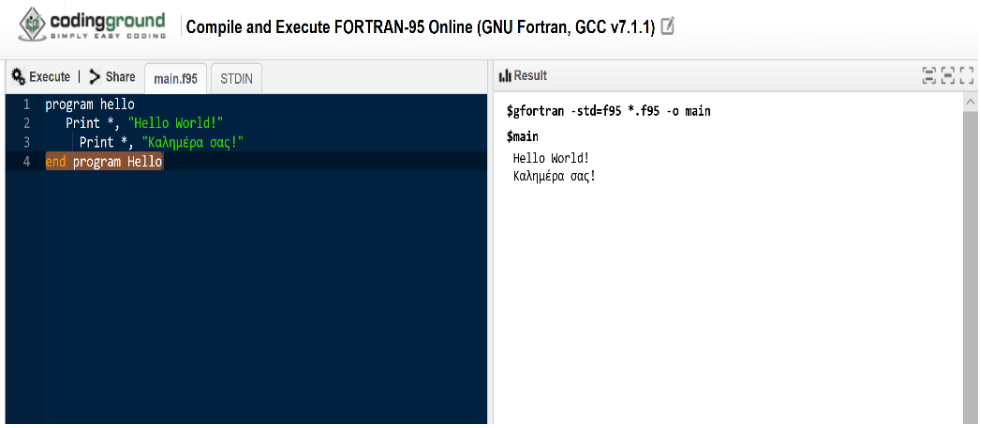
Όσον αφορά την άμμεση (on line) δοκιμή προγραμμάτων με τη γλώσσα Fortran, μπορεί να χρησιμοποιηθεί η διεύθυνση:

https://www.tutorialspoint.com/compile_fortran_online.php

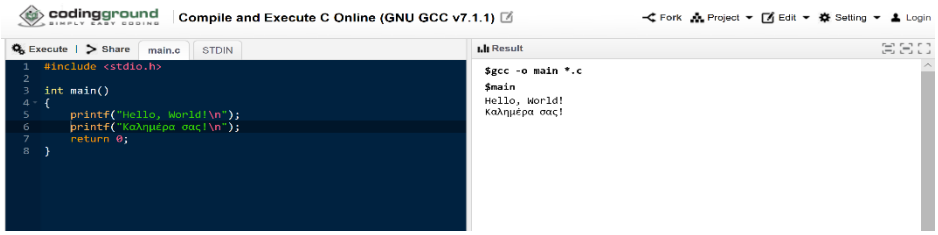
Για την άμμεση (on line) δοκιμή προγραμμάτων με τη γλώσσα C, μπορεί να χρησιμοποιηθεί με τη βοήθεια του διαδικτύου η διεύθυνση:

https://www.tutorialspoint.com/compile_c_online.php

Στις διευθύνσεις αυτές έχουμε άμμεση δοκιμή, δηλαδή έλεγχο και εκτέλεση των προγραμμάτων, όπως εμφανίζονται στις εικόνες 1. 15. και 1.16.



Εικόνα 1.15. *Online εκτέλεση προγράμματος της γλώσσας Fortran*



Εικόνα 1.16. *Online εκτέλεση προγράμματος της γλώσσας C*