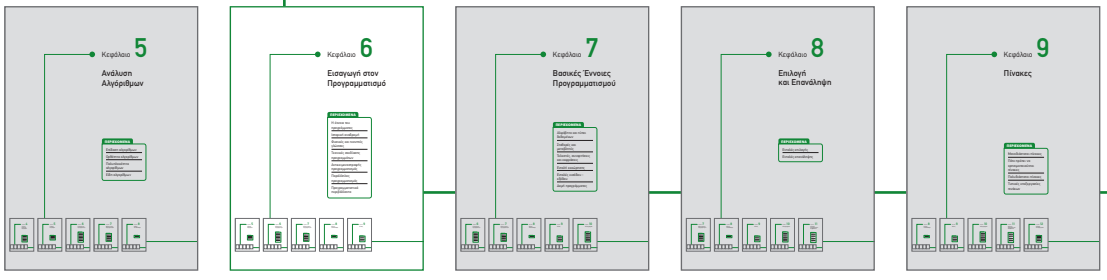
Κεφάλαιο 6

Εισαγωγή στον Προγραμματισμό

ΠΕΡΙΕΧΟΜΕΝΑ



6.1 | Η έννοια του προγράμματος

Η επίλυση ενός προβλήματος με τον υπολογιστή περιλαμβάνει, όπως έχει ήδη αναφερθεί, τρία εξίσου σημαντικά στάδια.

- Τον ακριβή προσδιορισμό του προβλήματος.
- Την ανάπτυξη του αντίστοιχου αλγορίθμου.
- Τη διατύπωση του αλγορίθμου σε κατανοητή μορφή από τον υπολογιστή.

Ο προγραμματισμός ασχολείται με το τρίτο αυτό στάδιο, τη δημιουργία του προγράμματος δηλαδή του συνόλου των εντολών που πρέπει να δοθούν στον υπολογιστή, ώστε να υλοποιηθεί ο αλγόριθμος για την επίλυση του προβλήματος. Το πρόγραμμα, το οποίο γράφεται σε κάποια γλώσσα προγραμματισμού, δεν είναι απλά η υλοποίηση του αλγορίθμου, αλλά βασικό στοιχείο του είναι τα δεδομένα και οι δομές δεδομένων επί των οποίων ενεργεί. Αναφέρθηκε ήδη ότι οι αλγόριθμοι και οι δομές δεδομένων είναι μια αδιάσπαστη ενότητα.

Οι γλώσσες προγραμματισμού αναπτύχθηκαν με σκοπό την επικοινωνία του ανθρώπου (προγραμματιστή) με τη μηχανή (υπολογιστή).

Ο προγραμματισμός είναι αυτός που δίνει την εντύπωση ότι οι υπολογιστές είναι έξυπνες μηχανές που επιλύουν τα πολύπλοκα προβλήματα.

Η εντύπωση αυτή όμως είναι απλώς μία ψευδαίσθηση. Ο υπολογιστής, ως γνωστόν, είναι μία μηχανή που καταλαβαίνει μόνο δύο καταστάσεις, οι οποίες αντιπροσωπεύονται με δύο αριθμούς, το μηδέν και το ένα, τα ψηφία του δυαδικού συστήματος. Το μόνο πράγμα που κάνει ο υπολογιστής είναι στοιχειώδεις ενέργειες σε ακολουθίες αυτών των δύο ψηφίων, αλλά αυτές τις ενέργειες τις εκτελεί με ασύλληπτη ταχύτητα. Ο υπολογιστής μπορεί απλά να αποθηκεύει στη μνήμη τις ακολουθίες των δυαδικών ψηφίων, να τις ανακτά, να κάνει στοιχειώδεις αριθμητικές πράξεις με αυτές και να τις συγκρίνει.

6.2 | Ιστορική αναδρομή

Από τη δημιουργία του πρώτου υπολογιστή μέχρι σήμερα έχουν αλλάξει πάρα πολλά πράγματα. Οι πρώτοι υπολογιστές, τεράστιοι σε μέγεθος αλλά με πάρα πολύ περιορισμένες δυνατότητες και μικρές ταχύτητες επεξεργασίας εξελίχθηκαν σε πολύ μικρούς σε μέγεθος υπολογιστές με τεράστιες όμως δυνατότητες και ταχύτητες επεξεργασίας.

Ενώ λοιπόν το υλικό (hardware) των υπολογιστών βελτιώνεται, τελειοποιείται και ταυτόχρονα παρέχει νέες δυνατότητες επεξεργασίας, οι βασικές αρχές λειτουργίας των υπολογιστών που διατυπώθηκαν το μακρινό 1945 από τον Φον Νόυμαν, δεν άλλαξαν πρακτικά καθόλου. Την ίδια αργή εξέλιξη ουσιαστικά έχουν και οι γλώσσες προγραμματισμού, οι οποίες, αν και εξελίσσονται και συνεχώς εμπλουτίζονται με νέες δυνατότητες, τα χαρακτηριστικά τους και οι βασικές τους ιδιότητες ουσιαστικά παραμένουν τα ίδια.

6.3 | Φυσικές και τεχνητές γλώσσες

Οι γλώσσες προγραμματισμού αναπτύχθηκαν για να μπορεί ο προγραμματιστής να δίνει τις εντολές που πρέπει να εκτελέσει ο υπολογιστής. Χρησιμοποιούνται δηλαδή για την επικοινωνία του ανθρώπου και της μηχανής, όπως αντίστοιχα οι φυσικές γλώσσες χρησιμοποιούνται για την επικοινωνία μεταξύ των ανθρώπων. Οι γλώσσες προγραμματισμού, που είναι τεχνητές γλώσσες, ακολουθούν τις βασικές έννοιες και αρχές της γλωσσολογίας, επιστήμη που μελετά τις φυσικές γλώσσες.

Μία γλώσσα προσδιορίζεται από το αλφάβητό της, το λεξιλόγιό της, τη γραμματική της και τέλος τη σημασιολογία της.

Το αλφάβητο

Αλφάβητο μίας γλώσσας καλείται το σύνολο των στοιχείων που χρησιμοποιείται από τη γλώσσα.

Για παράδειγμα, η ελληνική γλώσσα περιέχει τα εξής στοιχεία: Τα γράμματα του αλφαβήτου πεζά και κεφαλαία 48 δηλαδή χαρακτήρες (Α-Ω και α-ω), τα 10 ψηφία (0-9) και όλα τα σημεία στίξης. Αντίστοιχα η αγγλική γλώσσα περιλαμβάνει τα γράμματα του αγγλικού αλφαβήτου (A-Z και a-z) καθώς και τα ψηφία και όλα τα σημεία στίξης που χρησιμοποιούνται.

Το λεξιλόγιο

Το λεξιλόγιο αποτελείται από ένα υποσύνολο όλων των ακολουθιών που δημιουργούνται από τα στοιχεία του αλφαβήτου, τις λέξεις που είναι δεκτές από τη γλώσσα. Για παράδειγμα, στην ελληνική γλώσσα η ακολουθία των γραμμάτων ΑΒΓΑ είναι δεκτή αφού αποτελεί λέξη, αλλά η ακολουθία ΑΒΓΔΑ δεν αποτελεί λέξη της ελληνικής γλώσσας, άρα δεν είναι δεκτή.

Η Γραμματική

Η Γραμματική αποτελείται από το **τυπικό** ή **τυπολογικό** (accidence) και το **συντακτικό** (syntax).

Τυπικό είναι το σύνολο των κανόνων που ορίζει τις μορφές με τις οποίες μία λέξη είναι αποδεκτή. Για παράδειγμα, στην ελληνική γλώσσα οι λέξεις γλώσσα, γλώσσας, γλώσσες είναι δεκτές, ενώ η λέξη γλώσσατ δεν είναι αποδεκτή.

Συντακτικό είναι το σύνολο των κανόνων που καθορίζει τη νομιμότητα της διάταξης και της σύνδεσης των λέξεων της γλώσσας για τη δημιουργία προτάσεων.

Η γνώση του συντακτικού επιτρέπει τη δημιουργία σωστών προτάσεων στις φυσικές γλώσσες, ενώ στις γλώσσες προγραμματισμού τη δημιουργία σωστών εντολών.

Η σημασιολογία

Η σημασιολογία (Semantics) είναι το σύνολο των κανόνων που καθορίζει το νόημα των λέξεων και κατά επέκταση των εκφράσεων και προτάσεων που χρησιμοποιούνται σε μία γλώσσα.

Στις γλώσσες προγραμματισμού οι οποίες είναι τεχνητές γλώσσες, ο δημιουργός της γλώσσας αποφασίζει τη σημασιολογία των λέξεων της γλώσσας.

Κάθε γλώσσα προσδιορίζεται από το αλφάβητό της, το λεξιλόγιό της, τη γραμματική της και τη σημασιολογία της.

Διαφορές φυσικών και τεχνητών γλωσσών

Μία βασική διαφορά μεταξύ φυσικών και τεχνητών γλωσσών είναι η δυνατότητα εξέλιξής τους. Οι φυσικές γλώσσες εξελίσσονται συνεχώς, νέες λέξεις δημιουργούνται, κανόνες γραμματικής και σύνταξης αλλάζουν με την πάροδο του χρόνου και αυτό γιατί η γλώσσα χρησιμοποιείται για την επικοινωνία μεταξύ ανθρώπων, που εξελίσσονται και αλλάζουν ανάλογα με τις εποχές και τον κοινωνικό περίγυρο.

Αντίθετα οι τεχνητές γλώσσες χαρακτηρίζονται από στασιμότητα, αφού κατασκευάζονται συνειδητά για ένα συγκεκριμένο σκοπό.

Ωστόσο συχνά οι γλώσσες προγραμματισμού βελτιώνονται και μεταβάλλονται από τους δημιουργούς τους, με σκοπό να διορθωθούν αδυναμίες ή να καλύψουν μεγαλύτερο εύρος εφαρμογών ή τέλος να ακολουθήσουν τις νέες εξελίξεις. Οι γλώσσες προγραμματισμού αλλάζουν σε επίπεδο διαλέκτου (για παράδειγμα GW-Basic και QuickBasic) ή σε επίπεδο επέκτασης (για παράδειγμα Basic και Visual Basic).

6.4 | Τεχνικές σχεδίασης προγραμμάτων

Από την αρχή της εμφάνισης των υπολογιστών γίνονται συνεχείς προσπάθειες ανάπτυξης μεθοδολογιών και τεχνικών προγραμματισμού, που θα εξασφαλίζουν τη δημιουργία απλών και κομψών προγραμμάτων, την εύκολη γραφή τους όσο και την κατανόησή τους.

6.4.1 Ιεραρχική σχεδίαση προγράμματος

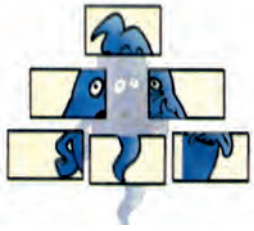
Η ιεραρχική σχεδίαση ή ιεραρχικός προγραμματισμός χρησιμοποιεί τη στρατηγική της συνεχούς διαίρεσης του προβλήματος σε υποπροβλήματα.

Η τεχνική της ιεραρχικής σχεδίασης και επίλυσης ή η διαδικασία σχεδίασης "από επάνω προς τα κάτω" όπως συχνά ονομάζεται (top-down program design) περιλαμβάνει τον καθορισμό των βασικών λειτουργιών ενός προγράμματος, σε ανώτερο επίπεδο, και στη συνέχεια τη διάσπαση των λειτουργιών αυτών σε όλο και μικρότερες λειτουργίες, μέχρι το τελευταίο επίπεδο που οι λειτουργίες είναι πολύ άπλες, ώστε να επιλυθούν εύκολα.

Σκοπός της ιεραρχικής σχεδίασης είναι η διάσπαση λοιπόν του προβλήματος σε μια σειρά από απλούστερα υποπροβλήματα, τα οποία να είναι εύκολο να επιλυθούν οδηγώντας στην επίλυση του αρχικού προβλήματος.

Για την υποβοήθηση της ιεραρχικής σχεδίασης χρησιμοποιούνται διάφορες διαγραμματικές τεχνικές, όπως για παράδειγμα το διάγραμμα του σχήματος 6.9.

6.4.2 Τμηματικός προγραμματισμός

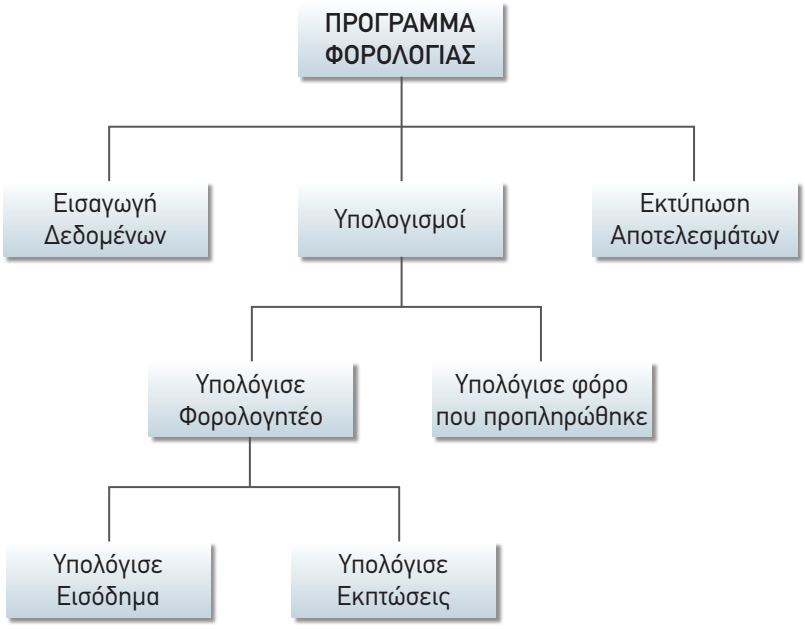


Η ιεραρχική σχεδίαση προγράμματος υλοποιείται με τον τμηματικό προγραμματισμό. Μετά την ανάλυση του προβλήματος σε αντίστοιχα υποπροβλήματα, κάθε υποπρόβλημα αποτελεί ανεξάρτητη **ενότητα** (module), που γράφεται ξεχωριστά από τα υπόλοιπα τμήματα προγράμματος.

Η σωστή διαίρεση του αρχικού προβλήματος σε υποπροβλήματα και κατά συνέπεια του αρχικού προγράμματος σε τμήματα προγράμματος

είναι μία διαδικασία αρκετά πολύπλοκη και θα εξεταστεί σε επόμενο κεφάλαιο.

Εδώ πρέπει να σημειωθεί ότι ο τμηματικός προγραμματισμός διευκολύνει τη δημιουργία του προγράμματος, μειώνει τα λάθη και επιτρέπει την ευκολότερη παρακολούθηση, κατανόηση και συντήρηση του προγράμματος από τρίτους.



Σχ. 6.9. *Ιεραρχική σχεδίαση υπολογισμού του φόρου εισοδήματος*

6.4.3 Δομημένος προγραμματισμός

Η μεθοδολογία που σήμερα έχει επικρατήσει απόλυτα και σχεδόν όλες οι σύγχρονες γλώσσες προγραμματισμού υποστηρίζουν είναι ο δομημένος προγραμματισμός (structured programming). Ο δομημένος προγραμματισμός παρουσιάστηκε στα μέσα του 1960.

Συγκεκριμένα το 1964 σε ένα συνέδριο στο Ισραήλ παρουσιάστηκε ένα κείμενο των Bohm και Jacopini με τις θεωρητικές αρχές του δομημένου προγραμματισμού. Οι απόψεις τους δεν έγιναν αρχικά ευρύτερα γνωστές και αποδεκτές, αλλά το 1968 ο καθηγητής Edsger Dijkstra δημοσίευσε ένα κείμενο που έκανε ιδιαίτερη αίσθηση και έμελλε να αλλάξει σταδιακά τον τρόπο προγραμματισμού καθώς και τις ίδιες τις γλώσσες προγραμματισμού. Ο τίτλος της μελέτης αυτής ήταν "GO TO Statement Considered Harmful - η εντολή GOTO θεωρείται επιβλαβής" και θεμελίωνε το δομημένο προγραμματισμό. Χρειάστηκε όμως να περάσουν αρκετά χρόνια, ώστε να αρχίσει να διαδίδεται η χρήση του δομημένου προγραμματισμού.

Την εποχή εκείνη δεν υπήρχε μία μεθοδολογία για την ανάπτυξη των προγραμμάτων, τα προγράμματα ήταν μεγάλα και ιδιαίτερα μπερδεμένα με αποτέλεσμα να ξοδεύεται πάρα πολύς χρόνος τόσο στη συγγραφή όσο κύρια στη διόρθωση και τη μετέπειτα συντήρησή τους. Βασικός λόγος για

τα προβλήματα αυτά ήταν η αλόγιστη χρήση μίας εντολής, της εντολής GOTO που χρησιμοποιούμενη άλλαζε διαρκώς τη ροή του προγράμματος. Ο δομημένος προγραμματισμός αναπτύχθηκε από την ανάγκη να υπάρχει μία κοινή μεθοδολογία στην ανάπτυξη των προγραμμάτων και τη μείωση των εντολών GOTO που χρησιμοποιούνται στο πρόγραμμα.

GOTO: ΤΟ ΜΑΥΡΟ ΠΡΟΒΑΤΟ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Στην ιστορία του προγραμματισμού καμία άλλη εντολή δεν συζητήθηκε τόσο πολύ όσο η εντολή **GOTO** (πήγαινε). Η εντολή GOTO έχει ως αποτέλεσμα την αλλαγή της ροής του προγράμματος, της διακλάδωσης σε μία άλλη εντολή του προγράμματος εκτός από την επόμενη. Η εντολή αυτή χώρισε τους προγραμματιστές σε δύο αντιμαχόμενες ομάδες. Η μία αποτελείτο από φανατικούς υποστηρικτές της χρήσης του GOTO, οι οποίοι με τη χρήση της έλυναν εύκολα και αβασάνιστα προβλήματα της ανάπτυξης των προγραμμάτων τους και η δεύτερη με πολέμιους που έβλεπαν ότι η εντολή αυτή ήταν υπεύθυνη για τη δυσκολία στην αρχική σχεδίαση της λύσης, στην παρακολούθηση και κατανόηση του προγράμματος και τέλος στη συντήρηση. Ο δομημένος προγραμματισμός προήλθε από την ανάγκη του περιορισμού της ανεξέλεγκτης χρήσης του GOTO.

Η χρήση της εντολής αυτής θα παρουσιαστεί με ένα απλό παράδειγμα, ενώ για λόγους σύγκρισης δίνεται ο ψευδοκώδικας με χρήση της δομής επιλογής, όπως παρουσιάστηκε στα προηγούμενα κεφάλαια.


```
,,,,,
ΑΝ Αριθμός>0 ΤΟΤΕ GOTO 1
ΑΝ Αριθμός=0 ΤΟΤΕ GOTO 2
ΓΡΑΨΕ 'Αρνητικός'
GOTO 4
1:ΓΡΑΨΕ 'Θετικός'
GOTO 4
2: ΓΡΑΨΕ 'Μηδέν'
GOTO 4
4:! Συνέχεια,
```

```
,,,,
ΑΝ Αριθμός>0 ΤΟΤΕ ΓΡΑΨΕ 'Θετικός'
ΑΛΛΙΩΣ_ΑΝ Αριθμός=0 ΤΟΤΕ ΓΡΑΨΕ 'Μηδέν'
ΑΛΛΙΩΣ ΓΡΑΨΕ 'Αρνητικός'
ΤΕΛΟΣ_ΑΝ
,,,
```

Η χρήση του GOTO κάνει ακόμα και αυτό το μικρό τμήμα προγράμματος δύσκολο στην κατανόησή του και στην παρακολούθησή του.

Όλες οι σύγχρονες γλώσσες προγραμματισμού υποστηρίζουν το δομημένο προγραμματισμό και διαθέτουν εντολές που καθιστούν τη χρήση του GOTO περιττή. Για λόγους όμως συμβατότητας με τις παλιότερες εκδόσεις τους καθώς και για λόγους συντήρησης παλιών προγραμμάτων, μερικές τη διατηρούν στο ρεπερτόριο των εντολών τους.

Στη συνέχεια αυτού του βιβλίου η εντολή GOTO δεν θα μας απασχολήσει και καλό είναι να μη χρησιμοποιείται στην ανάπτυξη προγραμμάτων.

Ο δομημένος προγραμματισμός δεν είναι απλώς ένα είδος προγραμματισμού, είναι μία μεθοδολογία σύνταξης προγραμμάτων που έχει σκοπό να βοηθήσει τον προγραμματιστή στην ανάπτυξη σύνθετων προγραμμάτων, να μειώσει τα λάθη, να εξασφαλίσει την εύκολη κατανόηση των προγραμμάτων και να διευκολύνει τις διορθώσεις και τις αλλαγές σε αυτά.

Ο δομημένος προγραμματισμός στηρίζεται στη χρήση τριών και μόνο στοιχειωδών λογικών δομών, τη δομή της ακολουθίας, τη δομή της επιλογής και τη δομή της επανάληψης. Όλα τα προγράμματα μπορούν να γραφούν χρησιμοποιώντας μόνο αυτές τις τρεις δομές καθώς και συνδυασμό τους. Κάθε πρόγραμμα όπως και κάθε ενότητα προγράμματος έχει μόνο μία είσοδο και μόνο μία έξοδο.

Οι τρεις αυτές δομές παρουσιάστηκαν στο κεφάλαιο 2 και θα επαναληφθούν στα επόμενα κεφάλαια.

Αν και ο δομημένος προγραμματισμός αρχικά εμφανίστηκε σαν μία προσπάθεια περιορισμού των εντολών GOTO, σήμερα αποτελεί τη βασική μεθοδολογία προγραμματισμού.

Ο δομημένος προγραμματισμός ενθαρρύνει και βοηθάει την ανάλυση του προγράμματος σε επιμέρους τμήματα, έτσι ώστε σήμερα ο όρος δομημένος προγραμματισμός περιέχει τόσο την ιεραρχική σχεδίαση όσο και τον τμηματικό προγραμματισμό.

Πλεονεκτήματα του δομημένου προγραμματισμού

Επιγραμματικά μπορούμε να αναφέρουμε τα εξής πλεονεκτήματα του δομημένου προγραμματισμού.

- Δημιουργία απλούστερων προγραμμάτων.
- Άμεση μεταφορά των αλγορίθμων σε προγράμματα.
- Διευκόλυνση ανάλυσης του προγράμματος σε τμήματα.
- Περιορισμός των λαθών κατά την ανάπτυξη του προγράμματος.
- Διευκόλυνση στην ανάγνωση και κατανόηση του προγράμματος από τρίτους.
- Ευκολότερη διόρθωση και συντήρηση.

6.5 | Αντικειμενοστραφής προγραμματισμός

Μία νέα ιδέα στον προγραμματισμό γεννήθηκε στις παγωμένες νορβηγικές ακτές στα τέλη της δεκαετίας του '70 και πέρασε πολύ γρήγορα στην άλλη μεριά του Ατλαντικού. Πρόκειται για μια νέα τάση αντιμετώπισης προγραμματιστικών αντιλήψεων και δομών που ονομάζεται **αντικειμενοστραφής** (object-oriented) προγραμματισμός. Την τελευταία δεκαετία έχει γίνει η επικρατούσα κατάσταση και έχει αλλάξει ριζικά τα μέχρι πριν από λίγα χρόνια γνωστά και σταθερά σημεία αναφοράς των προγραμματιστών.

Η ιδέα του αντικειμενοστραφούς προγραμματισμού ή της αντικειμενοστραφούς σχεδίασης έχει τις ρίζες της σε πολύ απλοϊκή ιδέα. Ένα πρόγραμμα περιγράφει "ενέργειες" (επεξεργασία) που εφαρμόζονται

πάνω σε δεδομένα. Ένα βασικό ερώτημα που τίθεται είναι αν η φιλοσοφία, η δομή του προγράμματος είναι προτιμότερο να στηρίζεται στις "ενέργειες" ή στα δεδομένα. Η απάντηση σε αυτό το ερώτημα προσδιορίζει και τη βασική διαφορά ανάμεσα στις παραδοσιακές προγραμματιστικές τεχνικές και στην αντικειμενοστραφή προσέγγιση.

Η αντικειμενοστραφής σχεδίαση εκλαμβάνει ως πρωτεύοντα δομικά στοιχεία ενός προγράμματος τα δεδομένα, από τα οποία δημιουργούνται με κατάλληλη μορφοποίηση τα **αντικείμενα** (objects). Αυτή η σχεδίαση αποδείχθηκε ότι επιφέρει καλύτερα αποτελέσματα, αφού τα προγράμματα που δημιουργούνται είναι περισσότερο ευέλικτα και επαναχρησιμοποιήσιμα. Βέβαια, δημιουργούνται μία σειρά από εύλογα ερωτήματα, όπως "Τι ακριβώς είναι ένα αντικείμενο;", "Πώς προσδιορίζουμε και περιγράφουμε ένα αντικείμενο;", "Πώς το πρόγραμμα χειρίζεται τα αντικείμενα;" "Πώς τα αντικείμενα συσχετίζονται μεταξύ τους;". Απαντήσεις σε αυτά τα ερωτήματα καθώς και αναλυτική παρουσίαση του αντικειμενοστραφούς προγραμματισμού υπάρχουν στο κεφάλαιο 11.

Φυσικά ο αντικειμενοστραφής προγραμματισμός χρησιμοποιεί την ιεραρχική σχεδίαση, τον τμηματικό προγραμματισμό και ακολουθεί τις αρχές του δομημένου προγραμματισμού.

6.6 | Παράλληλος προγραμματισμός

Μία άλλη μορφή προγραμματισμού που αναπτύσσεται τελευταία και πιθανόν στο μέλλον να γνωρίσει μεγάλη άνθηση είναι ο **παράλληλος προγραμματισμός**. Σχετικά πρόσφατα εμφανίστηκαν υπολογιστές που ξεφεύγουν από την κλασική αρχιτεκτονική και διαθέτουν περισσότερους από έναν επεξεργαστές. Οι επεξεργαστές αυτοί μοιράζονται την ίδια μνήμη και λειτουργούν παράλληλα εκτελώντας διαφορετικές εντολές του ιδίου προγράμματος. Οι υπολογιστές αυτοί εμφανίζονται θεωρητικά να πετυχαίνουν ταχύτητες, που είναι ασύλληπτες για τους τυπικούς υπολογιστές με έναν επεξεργαστή. Για να εκμεταλλευτούμε όμως την ταχύτητα που προσφέρει η αρχιτεκτονική τους, πρέπει το πρόβλημα να διαιρεθεί σε τμήματα που εκτελούνται παράλληλα και στη συνέχεια να προγραμματιστεί σε ένα προγραμματιστικό περιβάλλον που να επιτρέπει τον παράλληλο προγραμματισμό.

Μια γλώσσα προγραμματισμού που υποστηρίζει παράλληλο προγραμματισμό είναι η OCCAM.

Όπως αναφέρθηκε και στο κεφάλαιο 5, ο παράλληλος προγραμματισμός αποτελεί μία σημαντική επιστημονική περιοχή, η οποία ξεφεύγει από τα όρια αυτού του βιβλίου.

6.7 | Προγραμματιστικά περιβάλλοντα

Κάθε πρόγραμμα που γράφτηκε σε οποιαδήποτε γλώσσα προγραμματισμού πρέπει να μετατραπεί σε μορφή αναγνωρίσιμη και εκτελέσιμη από τον υπολογιστή, δηλαδή σε εντολές γλώσσας μηχανής.

Η μετατροπή αυτή επιτυγχάνεται με τη χρήση ειδικών μεταφραστικών προγραμμάτων. Υπάρχουν δύο μεγάλες κατηγορίες τέτοιων προγραμμάτων, οι **μεταγλωττιστές** (compilers) και οι **διερμηνευτές** (interpreters).

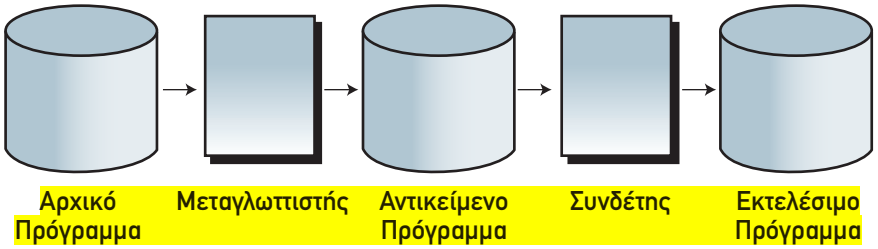
Ο μεταγλωττιστής δέχεται στην είσοδο ένα πρόγραμμα γραμμένο σε μια γλώσσα υψηλού επιπέδου και παράγει ένα ισοδύναμο πρόγραμμα σε γλώσσα μηχανής. Το τελευταίο μπορεί να εκτελείται οποτεδήποτε από τον υπολογιστή και είναι τελείως ανεξάρτητο από το αρχικό πρόγραμμα. Αντίθετα ο διερμηνευτής διαβάζει μία προς μία τις εντολές του αρχικού προγράμματος και για καθεμία εκτελεί αμέσως μια ισοδύναμη ακολουθία εντολών μηχανής.

Το αρχικό πρόγραμμα λέγεται **πηγαίο** πρόγραμμα (source), ενώ το πρόγραμμα που παράγεται από το μεταγλωττιστή λέγεται **αντικείμενο** πρόγραμμα (object).

Το αντικείμενο πρόγραμμα είναι μεν σε μορφή κατανοητή από τον υπολογιστή, αλλά συνήθως δεν είναι σε θέση να εκτελεστεί. Χρειάζεται να συμπληρωθεί και να συνδεθεί με άλλα τμήματα προγράμματος απαραίτητα για την εκτέλεσή του, τμήματα που είτε τα γράφει ο προγραμματιστής είτε βρίσκονται στις **βιβλιοθήκες** (libraries) της γλώσσας. Το πρόγραμμα που επιτρέπει αυτή τη σύνδεση ονομάζεται **συνδέτης – φορτωτής** (linker- loader). Το αποτέλεσμα του συνδέτη είναι η παραγωγή του **εκτελέσιμου προγράμματος** (executable), το οποίο είναι το τελικό πρόγραμμα που εκτελείται από τον υπολογιστή. Για το λόγο αυτό η συνολική διαδικασία αποκαλείται μεταγλώττιση και σύνδεση.

Η δημιουργία του εκτελέσιμου προγράμματος γίνεται μόνο στην περίπτωση που το αρχικό πρόγραμμα δεν περιέχει συντακτικά λάθη. Τις περισσότερες φορές κάθε πρόγραμμα αρχικά θα έχει λάθη. Τα λάθη του προγράμματος είναι γενικά δύο ειδών, λογικά και συντακτικά. Τα λογικά λάθη εμφανίζονται μόνο στην εκτέλεση, ενώ τα συντακτικά λάθη στο στάδιο της μεταγλώττισης.

Εκτενής παρουσίαση των λαθών και των τρόπων που αντιμετωπίζονται γίνεται σε επόμενο κεφάλαιο. Εδώ αναφέρουμε ότι τα λογικά λάθη που είναι τα πλέον σοβαρά και δύσκολα στη διόρθωσή τους οφείλονται σε σφάλματα κατά την υλοποίηση του αλγορίθμου, ενώ τα συντακτικά οφείλονται σε αναγραμματισμούς ονομάτων εντολών, παράλειψη δήλωσης δεδομένων και πρέπει πάντα να διορθωθούν, ώστε να παραχθεί το τελικό εκτελέσιμο πρόγραμμα.

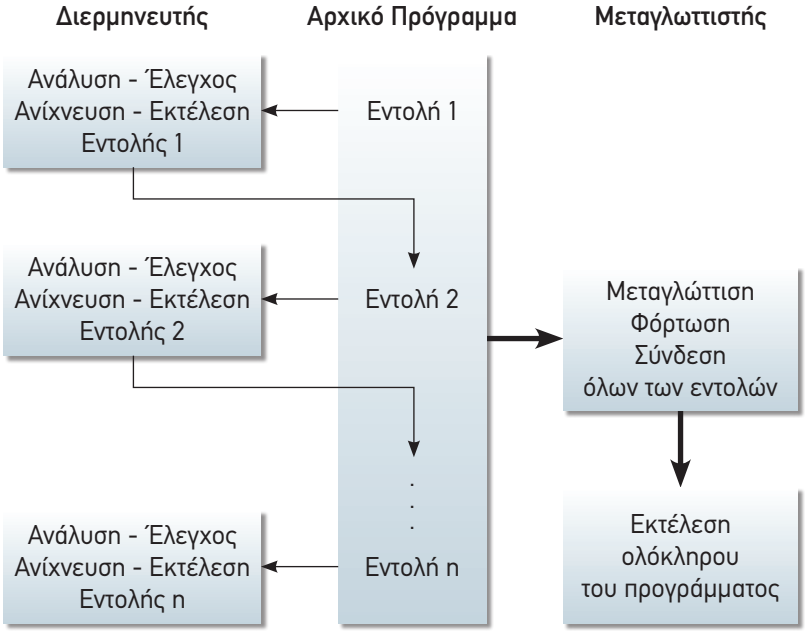


Σχ. 6.10. *Μεταγλώττιση και σύνδεση προγράμματος*

Ο μεταγλωττιστής ή ο διερμηνευτής ανιχνεύει λοιπόν τα συντακτικά λάθη και εμφανίζει κατάλληλα διαγνωστικά μηνύματα. Το στάδιο που ακολουθεί είναι η διόρθωση των λαθών. Το διορθωμένο πρόγραμμα επαναϋ-

ποβάλλεται για μεταγλώττιση και η διαδικασία αυτή επαναλαμβάνεται, μέχρις ότου εξαλειφθούν πλήρως όλα τα λάθη.

Η χρήση μεταγλωττιστή έχει το μειονέκτημα ότι, προτού χρησιμοποιηθεί ένα πρόγραμμα, πρέπει να περάσει από τη διαδικασία της μεταγλώττισης και σύνδεσης. Από την άλλη μεριά η χρήση διερμηνευτή έχει το πλεονέκτημα της άμεσης εκτέλεσης και συνεπώς και της άμεσης διόρθωσης. Όμως η εκτέλεση του προγράμματος καθίσταται πιο αργή, σημαντικά μερικές φορές, από εκείνη του ισοδύναμου εκτελέσιμου προγράμματος που παράγει ο μεταγλωττιστής. Πάντως τα σύγχρονα προγραμματιστικά περιβάλλοντα παρουσιάζονται συνήθως με μεικτές υλοποιήσεις, όπου χρησιμοποιείται διερμηνευτής κατά τη φάση δημιουργίας του προγράμματος και μεταγλωττιστής για την τελική έκδοση και εκμετάλλευση του προγράμματος.



Σχ. 6.11. *Διαδικασία μετάφρασης και εκτέλεσης ενός προγράμματος*

Τα σύγχρονα ολοκληρωμένα προγραμματιστικά περιβάλλοντα δεν παρέχουν απλώς ένα μεταφραστή μιας γλώσσας προγραμματισμού. Περιέχουν όλα τα προγράμματα και τα εργαλεία που απαιτούνται και βοηθούν τη συγγραφή, την εκτέλεση και κύρια τη διόρθωση των προγραμμάτων.

Για την αρχική σύνταξη των προγραμμάτων και τη διόρθωσή τους στη συνέχεια χρησιμοποιείται ένα ειδικό πρόγραμμα που ονομάζεται **συντάκτης** (editor). Ο συντάκτης είναι ουσιαστικά ένας μικρός επεξεργαστής κειμένου, με δυνατότητες όμως που διευκολύνουν τη γρήγορη γραφή των εντολών των προγραμμάτων.

Για τη δημιουργία, τη μετάφραση και την εκτέλεση ενός προγράμματος απαιτούνται τουλάχιστον τρία προγράμματα: ο συντάκτης, ο μεταγλωττιστής και ο συνδέτης. Τα σύγχρονα προγραμματιστικά περιβάλλοντα παρέχουν αυτά τα προγράμματα με ενιαίο τρόπο.

Το κάθε προγραμματιστικό περιβάλλον έχει φυσικά διαφορετικά εργαλεία και ιδιότητες. Για παράδειγμα, ένα περιβάλλον οπτικού (visual) προγραμματισμού πρέπει να περιέχει οπωσδήποτε και ειδικό συντάκτη που να διευκολύνει τη δημιουργία γραφικών αντικειμένων (για παράδειγμα, φόρμες, λίστες, παράθυρα διαλόγου) παρέχοντας στον προγραμματιστή τα αντίστοιχα γραφικά εργαλεία.

1. Ποια είναι τα τρία στάδια επίλυσης ενός προβλήματος με υπολογιστή;

1. Ακριβής προσδιορισμός του προβλήματος
2. Ανάπτυξη του αντίστοιχου αλγορίθμου
3. Διατύπωση του αλγορίθμου σε μορφή κατανοητή από τον υπολογιστή

Σύγκριση με ερωτ. 11 του 1ου κεφ.: στο 1ο κεφάλαιο μιλάει για αντιμετώπιση προβλήματος γενικά, όχι ειδικά για επίλυση προβλήματος με υπολογιστή. Εκεί τα στάδια είναι η κατανόηση, η ανάλυση και η επίλυση.

2. Με τι ασχολείται ο προγραμματισμός;

Ο προγραμματισμός ασχολείται με την δημιουργία του προγράμματος, δηλ. με την μετατροπή του αλγορίθμου σε μορφή κατανοητή από τον υπολογιστή.

3. Τι είναι πρόγραμμα;

Πρόγραμμα είναι το σύνολο των εντολών που πρέπει να δοθούν στον υπολογιστή για να υλοποιήσει έναν αλγόριθμο. Βασικό συστατικό ενός προγράμματος, εκτός από τις εντολές, είναι τα δεδομένα και οι δομές δεδομένων που επεξεργάζεται.

4. Ποιος είναι ο σκοπός των γλωσσών προγραμματισμού;

Ο σκοπός των γλωσσών προγραμματισμού είναι η επικοινωνία ανθρώπου (προγραμματιστή) και μηχανής (υπολογιστή).

5. Ποιες είναι οι δυνατότητες ενός Η/Υ; Είναι ο Η/Υ μια έξυπνη μηχανή που λύνει πολύπλοκα προβλήματα;

Ο Η/Υ το μόνο που μπορεί να κάνει είναι στοιχειώδεις ενέργειες με ακολουθίες δυαδικών ψηφίων: να τις αποθηκεύει στη μνήμη, να κάνει αριθμητικές πράξεις με αυτές, να τις συγκρίνει μεταξύ τους.

Ο Η/Υ δεν είναι μια έξυπνη μηχανή, αλλά μπορεί να λύνει πολύπλοκα προβλήματα επειδή είναι προγραμματίσιμος. Τα προγράμματα όμως γράφονται από ανθρώπους και όχι απο υπολογιστές. Οι υπολογιστές εκτελούν τις εντολές του προγράμματος, βέβαια με πολύ μεγάλη ταχύτητα.

6. Από τι προσδιορίζεται μια γλώσσα, φυσική ή τεχνητή;

Κάθε γλώσσα προσδιορίζεται από το αλφάβητο, το λεξιλόγιο, τη γραμματική (τυπικό και συντακτικό) και τη σημασιολογία.Συ

7. Τι είναι αλφάβητο, λεξιλόγιο, γραμματική και σημασιολογία μιας γλώσσας;

Αλφάβητο είναι το σύνολο των στοιχείων που χρησιμοποιούνται από τη γλώσσα.

Λεξιλόγιο είναι το σύνολο εκείνων των συνδυασμών των στοιχείων του αλφαβήτου που είναι αποδεκτοί από τη γλώσσα (σύνολο λέξεων).

Γραμματική περιλαβάνει το τυπικό και το συντακτικό.

> **Τυπικό** είναι το σύνολο των κανόνων που ορίζουν με ποιες μορφές μια λέξη είναι αποδεκτή.
>
> **Συντακτικό** είναι το σύνολο των κανόνων που ορίζουν με ποιους τρόπους μπορούν να συνδυαστούν οι λέξεις για να παραχθούν σωστές προτάσεις. Στις τεχνητές γλώσσες, οι προτάσεις αντιστοιχούν σε σωστές εντολές.

Σημασιολογία είναι το σύνολο των κανόνων που ορίζουν τη σημασία και το νόημα κάθε λέξης, άρα και κάθε πρότασης. Στις τεχνητές γλώσσες, ο δημιουργός της γλώσσας αποφασίζει για τη σημασιολογία των λέξεων.

8. Ποιες είναι οι ομοιότητες και διαφορές μεταξύ φυσικών και τεχνητών γλωσσών;

Ομοιότητα: Τόσο οι φυσικές όσο και οι τεχνητές γλώσσες ακολουθούν τις αρχές της γλωσσολογίας, έχουν δηλ. αλφάβητο, λεξιλόγιο, γραμματική και σημασιολογία.

Διαφορά: Οι φυσικές γλώσσες εξελίσσονται φυσικά και αβίαστα. Οι τεχνητές γλώσσες χαρακτηρίζονται από στασιμότητα και εξελίσσονται μόνο όταν ο δημιουργός τους αποφασίσει να προσθέσει σε αυτές νέες δυνατότητες ή να τις βελτιώσει.

9.Λέξεις και προτάσεις υπάρχουν στις φυσικές γλώσσες. Σε τι αντιστοιχούν στις τεχνητές γλώσσες;

Λέξεις στις τεχνητές γλώσσες είναι: οι δεσμευμένες λέξεις, οι λέξεις που χρησιμοποιούνται για τις εντολές και τα ονόματα που μπορούμε να χρησιμοποιήσουμε σύμφωνα με τους κανόνες ονοματολογίας που ισχύουν σε κάθε γλώσσα.

Προτάσεις στις τεχνητές γλώσσες είναι: σωστά γραμμένες εκφράσεις, σωστά γραμμένες εντολές.

10. Τι είναι η ιεραρχική σχεδίαση ενός προγράμματος; Ποιοι άλλοι όροι είναι συνώνυμοι με αυτόν;

Η ιεραρχική σχεδίαση ενός προγράμματος είναι μια τεχνική σχεδίασης “από πάνω προς τα κάτω”, σύμφωνα με την οποία το πρόβλημα διαιρείται συνεχώς σε απλούστερα υποπροβλήματα. Τα τελικά υποπροβλήματα είναι εύκολο να επιλυθούν και η λύση τους οδηγεί στην λύση και του αρχικού προβλήματος.

Συνώνυμος όρος: ιεραρχικός προγραμματισμός (προσοχή: ΔΕΝ είναι προγραμματισμός, δηλ δεν γράφουμε πρόγραμμα. Απλώς αναλύουμε το αρχικό πρόβλημα σε μικρότερα)

11. Ποιο στάδιο ακολουθεί την ιεραρχική σχεδίαση;

Το επόμενο στάδιο μετά την ιεραρχική σχεδίαση είναι ο τμηματικός προγραμματισμός. Σε αυτό το στάδιο γράφουμε ανεξάρτητα μικρά τμήματα κώδικα, ένα για κάθε υποπρόβλημα. (Δεν γράφουμε δηλ. ένα μεγάλο πρόγραμμα το οποίο θα λύνει το αρχικό πρόβλημα, αλλά πολλά μικρά υποπρογράμματα, ένα για κάθε υποπρόβλημα.)

12. Τι είναι ο δομημένος προγραμματισμός;

Ο δομημένος προγραμματισμός είναι μια μεθοδολογία ανάπτυξης προγραμμάτων, σύμφωνα με την οποία όλα τα προγράμματα μπορούν να γραφούν χρησιμοποιώντας μόνο δομή ακολουθίας, δομή επιλογής, δομή επανάληψης ή συνδυασμό τους. Κάθε πρόγραμμα και κάθε ενότητα προγράμματος θα έχει μόνο μία είσοδο (μία αρχή) και μία έξοδο (ένα σημείο τέλους).

Σήμερα, ο όρος δομημένος προγραμματισμός περιέχει τόσο την ιεραρχική σχεδίαση όσο και τον τμηματικό προγραμματισμό.

13. Ο δομημένος προγραμματισμός είναι ένα είδος προγραμματισμού;

Ο δομημένος προγραμματισμός δεν είναι απλώς ένα είδος προγραμματισμού. Είναι μία μεθοδολογία για την ανάπτυξη προγραμμάτων, που περιλαμβάνει τόσο την ιεραρχική σχεδίαση όσο και τον τμηματικό προγραμματισμό.

14. Σχολιάστε την χρήση της εντολής GOTO.

Η εντολή GOTO χρησιμοποιήθηκε στο παρελθόν και οδηγούσε σε μπερδεμένα προγράμματα, διότι η υπερβολική χρήση της άλλαζε συνεχώς τη ροή του προγράμματος. Καταργώντας την χρήση της GOTO, προέκυψε ο δομημένος προγραμματισμός.

15. Ποια τα πλεονεκτήματα του δομημένου προγραμματισμού;

α. Δημιουργία απλούστερων προγραμμάτων

β. Άμεση μεταφορά των αλγορίθμων σε προγράμματα

γ. Διευκόλυνση ανάλυσης του προγράμματος σε τμήματα

δ. Περιορισμός των λαθών κατά την ανάπτυξη του προγράμματος

ε. Διευκόλυνση στην κατανόηση του προγράμματος από τρίτους

στ. Ευκολότερη διόρθωση και συντήρηση του προγράμματος

16. Σε ποια γλώσσα πρέπει να γραφεί (εκφραστεί) ένα πρόγραμμα (ή ο κώδικας ενός προγράμματος) ώστε να είναι εκτελέσιμο (να μπορεί να εκτελεστεί) από έναν Η/Υ;

Τα προγράμματα γράφονται συνήθως σε γλώσσες υψηλού επιπέδου, δηλ. σε γλώσσες κατανοητές από τον άνθρωπο. Για να γίνει το πρόγραμμα κατανοητό από τον υπολογιστή (και να μπορεί να εκτελεστεί) πρέπει να μεταφραστεί σε γλώσσα μηχανής.

17. Τι είναι ο μεταγλωττιστής;

Ο μεταγλωττιστής είναι ένα εργαλείο (λογισμικό) το οποίο δέχεται στην είσοδο ένα πρόγραμμα γραμμένο σε γλώσσα υψηλού επιπέδου (αρχικό ή πηγαίο πρόγραμμα) και παράγει ένα ισοδύναμο πρόγραμμα σε γλώσσα μηχανής (αντικείμενο πρόγραμμα). Αυτή η μετάφραση του προγράμματος είναι δυνατή εφόσον το αρχικό πρόγραμμα δεν έχει συντακτικά λάθη.

Το αντικείμενο πρόγραμμα δεν είναι εκτελέσιμο, διότι πρέπει πρώτα να περάσει από ένα άλλο εργαλείο που λέγεται συνδέτης-φορτωτής.

18. Τι είναι ο διερμηνευτής;

Ο διερμηνευτής είναι ένα εργαλείο (λογισμικό) το οποίο δέχεται στην είσοδο ένα πρόγραμμα γραμμένο σε γλώσσα υψηλού επιπέδου, διαβάζει μία προς μία τις εντολές του

προγράμματος και αμέσως εκτελεί μια ισοδύναμη ακολουθία εντολών γλώσσας μηχανής. Στην πρώτη εντολή όπου θα υπάρξει συντακτικό λάθος, ο διερμηνευτής σταματάει.
Ουσιαστικά ο διερμηνευτής για κάθε εντολή κάνει 3 πράγματα: (α) την μεταφράζει σε γλώσσα μηχανής, (β) κάνει τη σύνδεση με βιβλιοθήκες αν χρειάζεται, (γ) την εκτελεί.

19. Σε ποιες μορφές μπορεί να υπάρξει ένα πρόγραμμα κατά τον κύκλο της ζωής του;

Αρχικό ή **Πηγαίο πρόγραμμα**: η πρώτη μορφή του προγράμματος σε γλώσσα υψηλού επιπέδου.
Αντικείμενο πρόγραμμα: παράγεται από τον μεταγλωττιστή, είναι σε γλώσσα μηχανής αλλά δεν μπορεί ακόμη να εκτελεστεί.
Εκτελέσιμο πρόγραμμα: παράγεται από τον συνδέτη-φορτωτή, είναι σε γλώσσα μηχανής και μπορεί να εκτελεστεί.

20. Ποια εργαλεία αποτελούν ένα σύγχρονο προγραμματιστικό περιβάλλον;

Ένα προγραμματιστικό περιβάλλον περιέχει όλα τα εργαλεία που απαιτούνται για τη συγγραφή, τη διόρθωση και την εκτέλεση προγραμμάτων. Περιέχει δηλαδή συντάκτη, μεταγλωττιστή, συνδέτη. Τα τελευταία χρόνια περιέχει και διερμηνευτή.

21. Τι είναι οι βιβλιοθήκες μιας γλώσσας προγραμματισμού;

Οι βιβλιοθήκες είναι σύνολα υποπρογραμμάτων, τα οποία μπορούν να χρησιμοποιηθούν από το προγραμματιστή, σαν επιπλέον δυνατότητες της γλώσσας προγραμματισμού. Τα υποπρογράμματα αυτά μπορεί να τα έχει γράψει ο δημιουργός της γλώσσας ή άλλοι προγραμματιστές.

22. Ποιες κατηγορίες λαθών μπορεί να υπάρξουν σε ένα πρόγραμμα; Πως γίνεται η διόρθωση;

Συντακτικά λάθη: Μπορεί να υπάρχουν στο πηγαίο (αρχικό) πρόγραμμα και οφείλονται σε λάθος γραμμένες εντολές. Τα συντακτικά λάθη τα ανακαλύπτει ο μεταγλωττιστής ή ο διερμηνευτής. Αν υπάρχουν συντακτικά λάθη, η μετάφραση του προγράμματος δεν ολοκληρώνεται και η επόμενη φάση είναι η φάση της διόρθωσης των λαθών από τον προγραμματιστή.

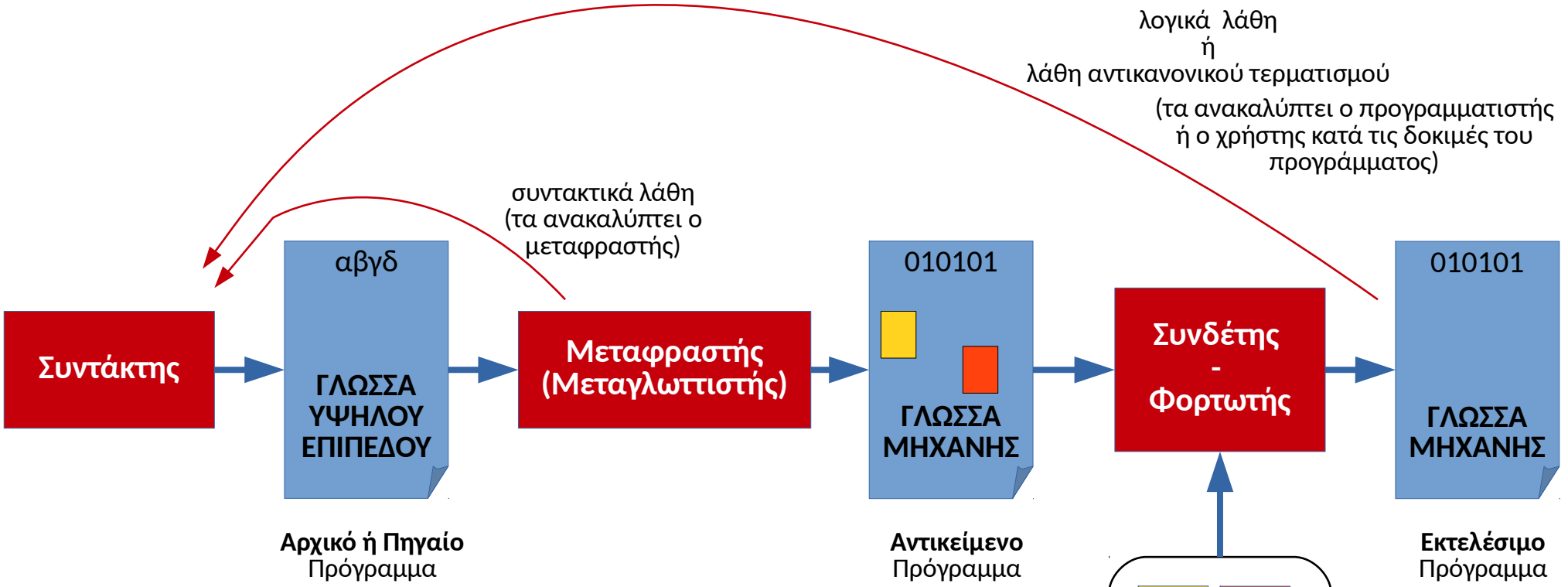
(Ο προγραμματιστής δηλ. χρησιμοποιεί πάλι τον συντάκτη, φορτώνει τον πηγαίο κώδικα και διορθώνει τα λάθη.)

Λογικά λάθη: Είναι τα πιο σοβαρά λάθη και δεν υπάρχει αυτόματος τρόπος για να τα ανακαλύψουμε. Τα ανακαλύπτει είτε ο προγραμματιστής είτε ο χρήστης κάνοντας πολλές δοκιμές του προγράμματος. Αν ανακαλυφθεί ένα λογικό λάθος, τότε ο προγραμματιστής επανέρχεται στο αρχικό πρόγραμμα και διορθώνει την λογική του προγράμματος.
(Ο προγραμματιστής δηλ. χρησιμοποιεί πάλι τον συντάκτη, φορτώνει τον πηγαίο κώδικα και διορθώνει τα λάθη.)

Λάθη χρόνου εκτέλεσης: Οφείλονται σε τιμές που προκαλούν αδυναμία εκτέλεσης κάποιων εντολών. Συχνά τα λάθη αυτά μπορούν να διορθωθούν με επιπλέον ελέγχους μέσα στο πρόγραμμα. Δεν υπάρχει αυτόματος τρόπος να τα ανακαλύψουμε, αλλά μόνο μέσα από πολλές δοκιμές, όπως τα λογικά λάθη.
(Ο προγραμματιστής δηλ. χρησιμοποιεί πάλι τον συντάκτη, φορτώνει τον πηγαίο κώδικα και διορθώνει τα λάθη.)

23. Συγκρίνετε την λειτουργία του μεταγλωττιστή και του διερμηνευτή. Ποιος είναι προτιμότερος;

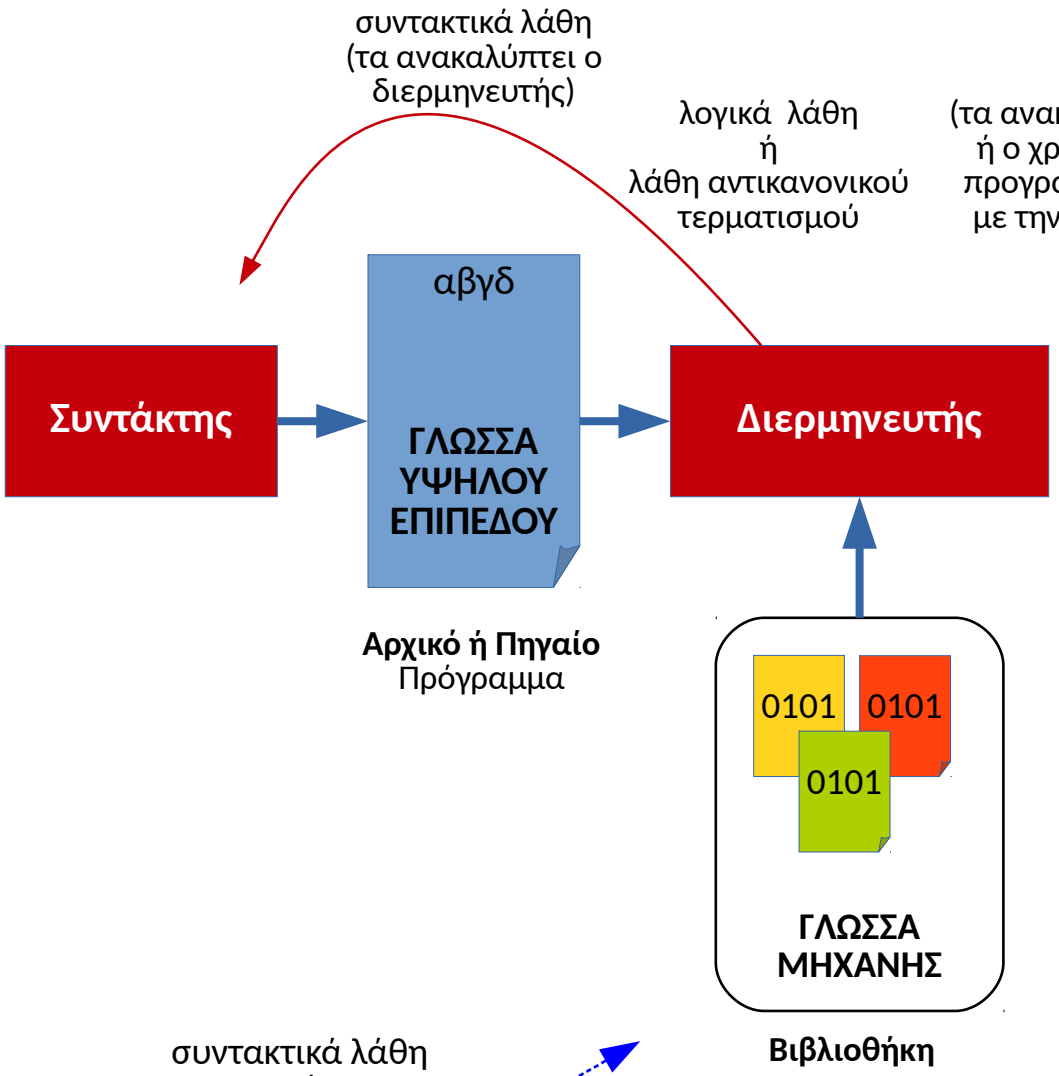
Ο διερμηνευτής είναι προτιμότερο να χρησιμοποιείται κατά την ανάπτυξη του προγράμματος, όπου ο προγραμματιστής θέλει να βλέπει άμεσα τα λάθη και να τα διορθώνει.
Όταν πλέον ο προγραμματιστής θα έχει βεβαιωθεί ότι το πρόγραμμα δεν έχει λάθη, είναι προτιμότερο να χρησιμοποιήσει μεταφραστή κ συνδέτη, ώστε να παραχθεί εκτελέσιμο πρόγραμμα. Έτσι κάθε φορά που θα πρέπει να εκτελεστεί το πρόγραμμα, δεν θα γίνεται ξανά η διαδικασία της μεταγλώττισης.



Συντάκτης: Επεξεργαστής Κειμένου που με διάφορους τρόπους διευκολύνει τον προγραμματιστή στη συγγραφή του πηγαίου προγράμματος.

Μεταφραστής: Ελέγχει το πηγαίο πρόγραμμα για συντακτικά λάθη. Αν δεν βρεθούν λάθη, παράγει ισοδύναμο πρόγραμμα σε γλώσσα μηχανής, το αντικείμενο πρόγραμμα. Αν βρεθούν λάθη, η μετάφραση σταματάει.

Συνδέτης-Φορτωτής: Συμπληρώνει το αντικείμενο πρόγραμμα με τα υποπρογράμματα που προέρχονται από βιβλιοθήκες. Συνδέει το αντικείμενο πρόγραμμα με τις απαραίτητες βιβλιοθήκες και παράγει το εκτελέσιμο πρόγραμμα.



Συντάκτης: Επεξεργαστής Κειμένου που με διάφορους τρόπους διευκολύνει τον προγραμματιστή στη συγγραφή του πηγαίου προγράμματος.

Διερμηνευτής: Επεξεργάζεται μία μία τις εντολές του πηγαίου προγράμματος.
Η εντολή ελέγχεται πρώτα για συντακτικά λάθη.
Αν είναι σωστή, την μεταφράζει σε γλώσσα μηχανής, αν χρειάζεται κάνει σύνδεση με βιβλιοθήκες, και στη συνέχεια εκτελεί την εντολή και προχωράει στην επόμενη.
Αν η εντολή έχει συντακτικά λάθη, τότε η όλη διαδικασία σταματάει.

Δεν παράγει εκτελέσιμο πρόγραμμα.
Κάθε φορά που θέλουμε να εκτελέσουμε το πρόγραμμα, γίνεται ξανά μετάφραση, σύνδεση και εκτέλεση μίας μίας εντολής.

Βιβλιοθήκη: Συλλογή υποπρογραμμάτων σε γλώσσα μηχανής, άρα έχουν παραχθεί από Μεταφραστή και όχι Διερμηνευτή