Ανάπτυξη εφαρμογών σε προγραμματιστικό περιβάλλον

Γ’ Λυκείου

ΚΕΦΑΛΑΙΟ 6

Εισαγωγή στον Προγραμματισμό

Χρήστος Μουρατίδης - Έκδοση 2020
mouratx@yahoo.com
http://users.sch.gr/mouratx

Περιεχόμενα

Η έννοια του προγράμματος

Όπως έχουμε περιγράψει στα προηγούμενα Κεφάλαια, η επίλυση ενός προβλήματος μέσω υπολογιστή περιλαμβάνει 3 στάδια:

- Ο ακριβής προσδιορισμός των δεδομένων-ζητούμενων του προβλήματος.
- Η ανάπτυξη του αλγορίθμου, π.χ. σε ψευδογλώσσα, ο οποίος περιγράφει το πώς θα λυθεί.
- Τη διατύπωση του αλγορίθμου σε μορφή κατανοητή από τον υπολογιστή.

Ο προγραμματισμός εστιάζει στο 3ο στάδιο, δηλαδή την υλοποίηση του αλγορίθμου σε μία γλώσσα προγραμματισμού καθώς και τη χρήση των κατάλληλων δομών δεδομένων[1].

Ο προγραμματισμός είναι η διατύπωση του αλγορίθμου σε μορφή κατανοητή από τον Η/Υ ώστε να τον εκτελέσει («τρέξει» όπως λέμε στην ορολογία της Πληροφορικής).

Η διατύπωση γίνεται χρησιμοποιώντας μία γλώσσα προγραμματισμού.

Κάτι που επίσης πρέπει να υπενθυμίσουμε είναι ότι ο υπολογιστής καταλαβαίνει μόνο δύο ψηφία: το `0` και το `1` (bits)[2]. Όλα τα δεδομένα και οι εντολές (τα προγράμματα) πρέπει να αποθηκεύονται και να διακινούνται στο εσωτερικό του (μνήμη, επεξεργαστής κλπ.) σε μία δυαδική μορφή (ακολουθίες από bits).

Ευτυχώς, δεν χρειάζεται να προγραμματίζουμε σε δυαδική μορφή αλλά χρησιμοποιούμε τις λεγόμενες γλώσσες προγραμματισμού υψηλού επιπέδου, που είναι σε μία μορφή παρόμοια της αγγλικής γλώσσας και ειδικά σχεδιασμένη για επικοινωνία με υπολογιστές.

[1] Στο Κεφάλαιο 3 είδαμε τη βασική ισότητα: Πρόγραμμα = Αλγόριθμος + Δομές δεδομένων.

[2] Το 0 αντιπροσωπεύει οπτικά ένα *«ανοικτό τρανζίστορ»* και το 1 ένα *«κλειστό τρανζίστορ»*, Αυτές είναι μόνο οι δύο καταστάσεις που προκύπτουν στο υλικό του υπολογιστή και τα πάντα πρέπει να κωδικοποιηθούν με τη χρήση αυτών των τρανζίστορ.

Φυσικές και τεχνητές γλώσσες

Οι φυσικές γλώσσες είναι αυτές που μιλιούνται μεταξύ των ανθρώπων ενώ οι τεχνητές γλώσσες (π.χ. γλώσσες προγραμματισμού) κατασκευάζονται σε εργαστήριο για κάποιον ειδικό σκοπό (π.χ. επικοινωνία με τον υπολογιστή).

Μία γλώσσα (φυσική ή τεχνητή), προσδιορίζεται από:

- Το **αλφάβητό** της
- Το **λεξιλόγιό** της
- Τη **γραμματική** της
- Τη **σημασιολογία** της (Semantics)

Αλφάβητο

Ως αλφάβητο ορίζουμε **το σύνολο των αποδεκτών χαρακτήρων της γλώσσας**. Από τους χαρακτήρες αυτούς σχηματίζονται οι λέξεις της γλώσσας.

Σε μία φυσική γλώσσα, όπως τα Ελληνικά, οι αποδεκτοί χαρακτήρες είναι τα γράμματα Α-Ω (κεφαλαία και πεζά), τα ψηφία 0-9 και τα σημεία στίξης.

Λεξιλόγιο

Το λεξιλόγιο μίας γλώσσας **περιλαμβάνει όλες τις έγκυρες και αποδεκτές λέξεις**. Στην ουσία, είναι ένα υποσύνολο από όλες τις δυνατές ακολουθίες που μπορούμε να σχηματίσουμε από τα στοιχεία του αλφαβήτου.

Σε μία φυσική γλώσσα, όπως τα Ελληνικά, η λέξη ΔΙΑΒΑΖΩ είναι αποδεκτή ενώ η λέξη ΖΩΒΑΓΩ όχι.

Γραμματική

Η γραμματική περιλαμβάνει το **τυπολογικό** και το **συντακτικό**.

- Το **τυπολογικό** ορίζει τους κανόνες σύμφωνα με του οποίους μία λέξη θα είναι αποδεκτή.

  Για παράδειγμα, στην ελληνική γλώσσα για τη λέξη ‘’δίνω’’, αποδεκτές μορφές είναι και το ‘’δίνεις’’, ‘’δίνουν’’ αλλά όχι το ‘’δίνουτ’’.

- Το **συντακτικό** είναι ένα σύνολο κανόνων που ορίζει το πώς πρέπει να σχηματίζονται οι προτάσεις από τις λέξεις της γλώσσας ώστε οι προτάσεις αυτές να είναι έγκυρες και αποδεκτές. **Σε μία γλώσσα προγραμματισμού** αυτό που ενδιαφέρει είναι η **σωστή σύνταξη των εντολών**.

Σημασιολογία

Είναι το **σύνολο των κανόνων που καθορίζει το νόημα των λέξεων και προτάσεων της γλώσσας**. Σε μία γλώσσα προγραμματισμού αυτό καθορίζεται από το δημιουργό της ενώ σε μία φυσική γλώσσα από αυτόν που εκφέρει την πρόταση.

Διαφορές μεταξύ φυσικών και τεχνητών γλωσσών

Παρακάτω, κάνουμε τη διάκριση μεταξύ φυσικών και τεχνητών γλωσσών.

Φυσικές γλώσσες	Τεχνητές γλώσσες
Χρησιμοποιούνται για την επικοινωνία μεταξύ ανθρώπων.	Χρησιμοποιούνται για επικοινωνία του ανθρώπου με τον υπολογιστή.
Αναπτύχθηκαν και εξελίχθηκαν στη διάρκεια αιώνων.	Αναπτύχθηκαν σε εργαστήρια, με βάση πάντα τις αρχές της γλωσσολογίας, ώστε να έχουν όμοια χαρακτηριστικά με τις φυσικές γλώσσες κι έτσι η εκμάθησή τους να είναι εύκολη.
Έχουν μεγάλες δυνατότητες εξέλιξης. Νέες λέξεις μπορεί να εισαχθούν, κανόνες γραμματικής και σύνταξης να αλλάξουν κλπ.	Χαρακτηρίζονται από στασιμότητα επειδή κυρίως κατασκευάζονται για ένα συγκεκριμένο σκοπό. Παρόλα αυτά, κατά καιρούς, οι δημιουργοί τις βελτιώνουν, τις μεταβάλλουν και τις επεκτείνουν (π.χ. νέες εντολές) ώστε να διορθωθούν αδυναμίες και να παρακολουθούν τις νέες εξελίξεις.
Έχουν Αλφάβητο, Λεξιλόγιο, Γραμματική, Σημασιολογία.	Παρόμοια.
Παραδείγματα: Ελληνικά, Αγγλικά, Γαλλικά κ.α.	Γλώσσες προγραμματισμού υψηλού επιπέδου. Παραδείγματα: C, Pascal, Basic, Python, Java κ.α.

Τεχνικές σχεδίασης προγραμμάτων

Δεν αρκεί κάποιος να γνωρίζει απλά μία γλώσσα προγραμματισμού. Θα πρέπει να ακολουθήσει και μία **τεχνική σχεδίασης του προγράμματός του**.

Για την σύνταξη **σωστών**, **κατανοητών** και **εύκολα συντηρήσιμων προγραμμάτων** ακολουθήθηκαν διάφορες τεχνικές-μεθοδολογίες ανάπτυξης.

Αυτές οι **τεχνικές** συνοψίζονται στις εξής:

- **Ιεραρχική σχεδίαση προγράμματος**. Ονομάζεται και ιεραρχικός προγραμματισμός (top-down program design).
- **Τμηματικός προγραμματισμός** (modular programming)
- **Δομημένος προγραμματισμός** (structured programming)

Στη συνέχεια, περιγράφουμε συνοπτικά, την κάθε τεχνική.

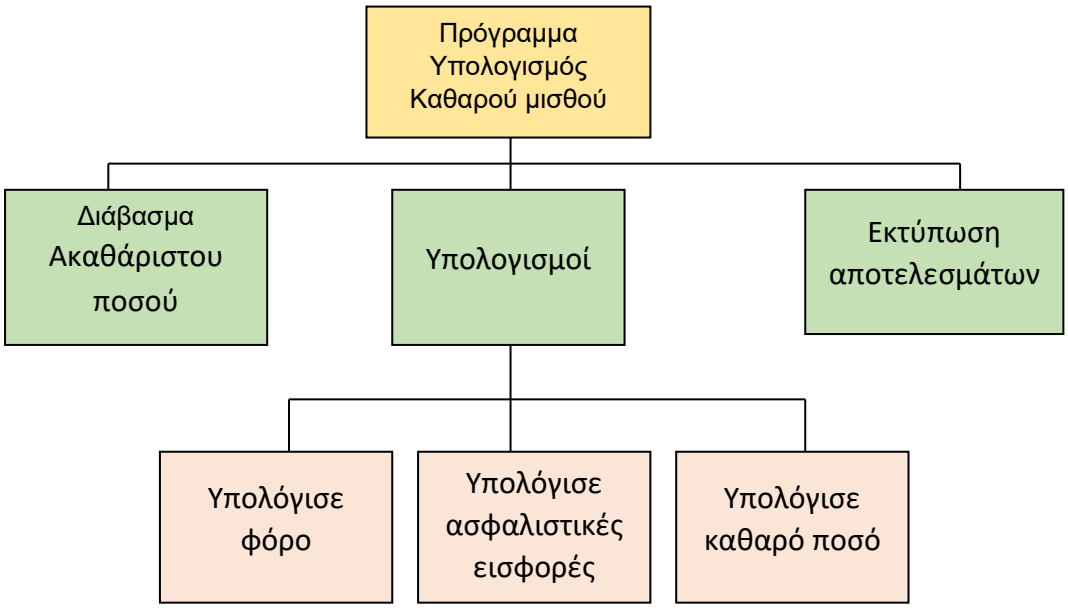
Ιεραρχική σχεδίαση (Ιεραρχικός προγραμματισμός)

Η τεχνική αυτή συνιστά στον καθορισμό των βασικών λειτουργιών του προγράμματος σε ανώτερο επίπεδο και στη συνέχεια τη διάσπαση καθεμιάς σε μικρότερες και απλούστερες μέχρι του σημείου να είναι τόσο απλές που μπορούν να επιλυθούν άμεσα.

Συνεπώς, σκοπός της ιεραρχικής σχεδίασης είναι η διάσπαση του προβλήματος σε μικρότερα κι απλούστερα υπο-προβλήματα τα οποία είναι ευκολότερο να επιλυθούν.

Για την Top-down σχεδίαση χρησιμοποιούμε το ***ιεραρχικό διάγραμμα***.

Παράδειγμα : Πρόγραμμα Υπολογισμός Καθαρού Μισθού



Βλέπουμε πώς το αρχικό πρόβλημα αναλύεται σε επιμέρους υπο-προβλήματα. Ιδιαίτερα, παρατηρούμε το υπο-πρόβλημα «`Υπολογισμοί`» αναλύεται σε 3 επιμέρους υπο-προβλήματα.

Η διάσπαση μπορεί να συνεχιστεί σε περισσότερα επίπεδα μέχρι του σημείου όπου είναι ξεκάθαρο πώς το υπο-πρόβλημα θα επιλυθεί. Για παράδειγμα, αν ο φόρος είναι το 15% του ακαθάριστου εισοδήματος τότε είναι ξεκάθαρο στο υπο-πρόβλημα «`Υπολόγισε φόρο`» τι ακριβώς υπολογισμός θα γίνει.

Τμηματικός προγραμματισμός

Η ιεραρχική σχεδίαση υλοποιείται με τον τμηματικό προγραμματισμό.

Μετά την ανάλυση του προβλήματος σε μικρότερα και απλούστερα υπο-προβλήματα, κάθε υπο-πρόβλημα αποτελεί μία ξεχωριστή και ανεξάρτητη **ενότητα** (module). Τώρα, για κάθε ενότητα θα γραφτεί το κατάλληλο πρόγραμμα ή τμήμα προγράμματος[3].

Η ανάπτυξη προγραμμάτων κατά τμήματα, διευκολύνει την επίλυση σύνθετων προβλημάτων, ιδιαίτερα μεγάλης κλίμακας (π.χ. ένα πληροφοριακό σύστημα ενός

[3] Στον προγραμματισμό, αυτά τα ξεχωριστά τμήματα προγράμματος που επιτελούν συγκεκριμένες λειτουργίες ονομάζονται *υποπρογράμματα* και αποτελούν θέμα ξεχωριστού Κεφαλαίου.

οργανισμού, όπως είναι το myschool), μειώνει τα λάθη, επιτρέπει την ευκολότερη παρακολούθηση της ανάπτυξης του προγράμματος, την κατανόηση και τη συντήρησή τους.

Δομημένος προγραμματισμός

Είναι η μεθοδολογία σύνταξης προγραμμάτων που έχει επικρατήσει σήμερα. **Εμπεριέχει τις αρχές τις ιεραρχικής σχεδίασης και του τμηματικού προγραμματισμού**. Επιπλέον, αναφέρει τα εξής βασικά στοιχεία:

- **Για τη δημιουργία σωστών προγραμμάτων χρησιμοποιούμε μόνο τις 3 στοιχειώδες αλγοριθμικές δομές και συνδυασμό αυτών: Ακολουθίας, Επιλογής και Επανάληψης**[4]. Όλα τα προγράμματα, οσοδήποτε μεγέθους, μπορούν να γραφτούν στηριζόμενα στη χρήση μόνο αυτών των δομών, αποφεύγοντας πλήρως τη χρήση τη δομής GOTO.

- **Κάθε πρόγραμμα ή ενότητα προγράμματος έχει μόνο μία είσοδο και μόνο μία έξοδο**.

Πλεονεκτήματα του δομημένου προγραμματισμού:

- Άμεση υλοποίηση των αλγορίθμων σε πρόγραμμα.
- Διευκόλυνση της ανάλυσης του προγράμματος σε τμήματα (ενότητες). Το κάθε τμήμα μπορεί να γραφτεί από διαφορετικές ομάδες προγραμματιστών.
- Ευκολότερη και συντομότερη ανάπτυξη προγραμμάτων.
- Περιορισμός των λαθών κατά την ανάπτυξη του προγράμματος.
- Διευκόλυνση στην ανάγνωση και κατανόηση του προγράμματος από τρίτους.
- Ευκολότερη διόρθωση και συντήρηση του προγράμματος.

Θα παρουσιάσουμε ένα παράδειγμα απλού αλγορίθμου σε ψευδογλώσσα με GOTO (αδόμητος) και χωρίς GOTO (δομημένος).

[4] Παραδείγματα χρήσης των 3 αλγοριθμικών δομών με ψευδογλώσσα είδαμε στο Κεφάλαιο 2.

Παράδειγμα 1ο: Να γραφτεί αλγόριθμος που διαβάζει δύο αριθμούς και να τυπώνει ποιός είναι ο μεγαλύτερος. Αν είναι ίσοι να μας εμφανίζει, επίσης, σχετικό μήνυμα.

Με GOTO (μη δομημένος τρόπος):

```
Αλγόριθμος Μεγαλύτερος_αριθμός_με_GOTO
    Διάβασε α, β
    Αν α > β τότε GOTO 1
    Αν α < β τότε GOTO 2
    Αν α = β τότε GOTO 3
    GOTO 4
    1: Εκτύπωσε "Ο πρώτος είναι μεγαλύτερος"
    GOTO 4
    2: Εκτύπωσε "Ο δεύτερος είναι μεγαλύτερος"
    GOTO 4
    3: Εκτύπωσε "Είναι ίσοι!"
    4: Εκτύπωσε "Τέλος"
Τέλος Μεγαλύτερος_αριθμός_με_GOTO
```

Χωρίς GOTO (δομημένος τρόπος):

```
Αλγόριθμος Μεγαλύτερος_αριθμός_χωρίς_GOTO
    Διάβασε α, β
    Αν α > β τότε Εκτύπωσε "Ο πρώτος είναι μεγαλύτερος"
    αλλιώς_αν α < β τότε Εκτύπωσε "Ο δεύτερος είναι
                                   μεγαλύτερος"
    αλλιώς Εκτύπωσε "Είναι ίσοι!"
    Τέλος_αν

    Εκτύπωσε "Τέλος"
Τέλος Μεγαλύτερος_αριθμός_χωρίς_GOTO
```

Παράδειγμα 2ο: Να γραφτεί αλγόριθμος που τυπώνει όλους τους ζυγούς ακέραιους αριθμούς από το 2 μέχρι το 100.

Με GOTO (μη δομημένος τρόπος):

```
Αλγόριθμος Εκτύπωση_ζυγών_αριθμών_με_GOTO
    X ← 2
    1: Αν X > 100 τότε GOTO 4
    Αν X MOD 2 = 0 τότε GOTO 3
    2: X ← X + 1
    GOTO 1
    3: Εκτύπωσε X
    GOTO 2
    4: Εκτύπωσε "Τέλος"
Τέλος Εκτύπωση_ζυγών_αριθμών_με_GOTO.
```

Χωρίς GOTO (δομημένος τρόπος)[5]:

```
Αλγόριθμος Εκτύπωση_ζυγών_αριθμών_χωρίς_GOTO
    X ← 2
    Όσο X < 100 επανάλαβε
        Αν X MOD 2 = 0 τότε
            Εκτύπωσε X
        Τέλος_αν
        X ← X + 1
    Τέλος_επανάληψης
    Εκτύπωσε "Τέλος"
```

[5] Φυσικά, υπάρχουν κι άλλοι τρόποι υλοποίησης (π.χ. με βήμα αύξησης κατά 2 ή με τη δομή Για…από…μέχρι)

```
Τέλος Εκτύπωση_ζυγών_αριθμών_χωρίς_GOTO.
```

Προφανώς, στον δομημένο τρόπο του αλγορίθμου, η ροή εκτέλεσης είναι ομαλότερη συν το ότι ο αλγόριθμος είναι πιο κατανοητός και μπορεί να διορθωθεί ευκολότερα.

Στις δεκαετίες του 1950-60-70 γίνονταν εκτενή χρήση μίας εντολής GOTO η οποία οδηγούσε στην ανάπτυξη προγραμμάτων που ήταν δυσνόητα και δύσκολα συντηρήσιμα επειδή η ροή του προγράμματος διακλαδωνόταν συνέχεια σε διαφορετικά σημεία με αποτέλεσμα να προκαλεί «πονοκέφαλο» σε όποιον τρίτο ήθελε να ασχοληθεί με αυτό. Ακόμα και το πιο απλό πρόγραμμα ήταν δυσνόητο!

Η τεχνική του δομημένου προγραμματισμού εξοστρακίζει πλήρως τη δομή GOTO και ο σύγχρονος προγραμματισμός τη θεωρεί μία ανάμνηση-εφιάλτη! Στο διαδίκτυο, μπορεί κάποιος να βρει σχετικά παραδείγματα, κυρίως από παλαιές γλώσσες προγραμματισμού (π.χ. GWBASIC).

Αντικειμενοστραφής προγραμματισμός

Η φιλοσοφία του αντικειμενοστραφούς προγραμματισμού είναι η εξής:

Να θεωρήσουμε τα δεδομένα και τις αποδεκτές ενέργειες που γίνονται πάνω σε αυτά ως ένα ενιαίο αντικείμενο (object). Αυτό το αντικείμενο θα μπορεί να χρησιμοποιηθεί πολύ εύκολα οπουδήποτε αλλού, δηλαδή θα είναι *επαναχρησιμοποιήσιμο*.

Έτσι, σε ένα πρόγραμμα, δημιουργούμε αρκετά αντικείμενα που συσχετίζονται μεταξύ τους ώστε να επιτευχθεί ο στόχος του προγράμματος. Η χρήση αντικειμένων κάνει τα προγράμματα περισσότερο ευέλικτα.

Για παράδειγμα, θα μπορούσαμε να θεωρήσουμε μία στοίβα ως ένα αντικείμενο : Ένας πίνακας `Π[1:20]` που θα δέχεται ακεραίους. Οι μόνες ενέργειες που θα ήταν επιτρεπτές θα ήταν η *ώθηση* (push) ενός στοιχείου στη στοίβα και η *αφαίρεση* (pop) ενός στοιχείου από τη στοίβα. Αυτό το αντικείμενο θα μπορούσαμε να το δώσουμε και σε όποιον άλλο το είχε ανάγκη.

Ο αντικειμενοστραφής προγραμματισμός ακολουθεί τις αρχές του ιεραρχικού, τμηματικού και φυσικά του δομημένου προγραμματισμού.

Στο παράρτημα θα βρείτε και μερικές άλλες έννοιες σχετικά με τις τεχνικές σχεδίασης προγραμμάτων.

Προγραμματιστικά περιβάλλοντα

Ένα προγραμματιστικό περιβάλλον είναι **ειδικό λογισμικό που παρέχει όλα εκείνα τα εργαλεία (tools) για τη συγγραφή, διόρθωση, μετάφραση και παραγωγή του προγράμματος**.

Ένα πρόγραμμα που φτιάχνεται σε μία γλώσσα υψηλού επιπέδου δεν είναι άμεσα κατανοητό από τον Η/Υ. Θα πρέπει να μεταφραστεί σε ισοδύναμο πρόγραμμα σε γλώσσα μηχανής (δυαδική μορφή). Την διαδικασία μετάφρασης την πραγματοποιούν τα μεταφραστικά προγράμματα. Είναι δύο ειδών:

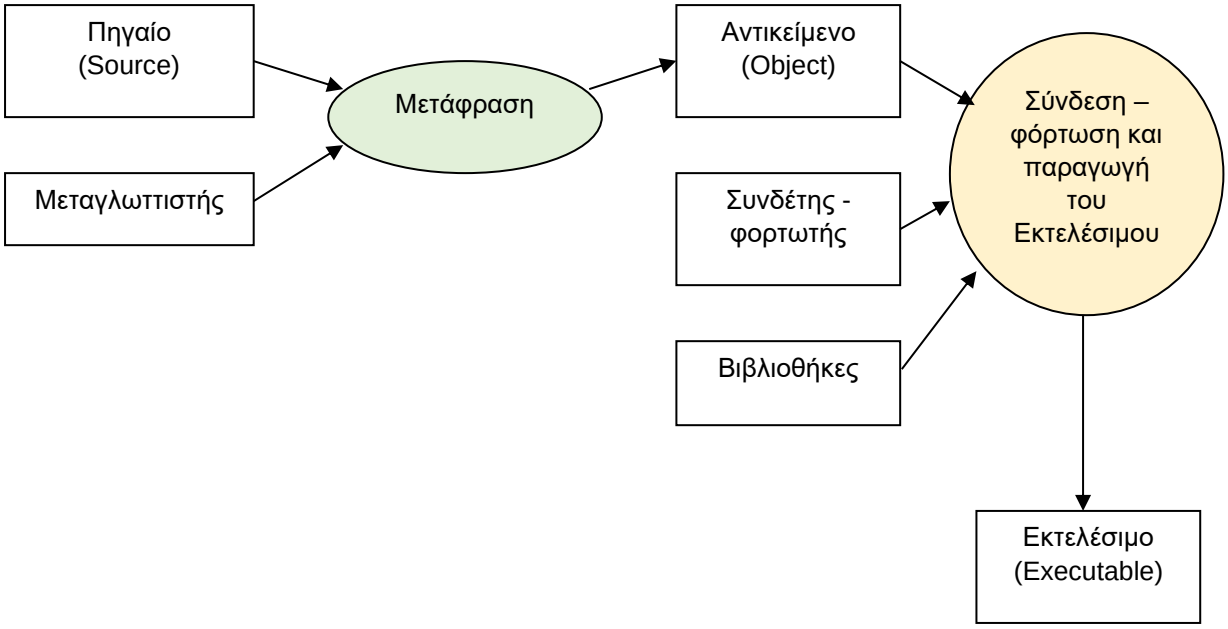
- Μεταγλωττιστές (Compilers)
- Διερμηνευτές (Interpreters)

Στον παρακάτω πίνακα επεξηγούνται κάποιες βασικές έννοιες, που σχετίζονται με το προγραμματιστικό περιβάλλον.

Έννοια	Περιγραφή
Πηγαίο πρόγραμμα (Source program)	**Είναι αυτό που συγγράφει ο προγραμματιστής σε μία γλώσσα υψηλού επιπέδου** (π.χ. C, Java, Basic, Python κλπ.). Είναι κατανοητό από τον άνθρωπο, όχι όμως από τη μηχανή.
Μεταφραστής: • **Μεταγλωττιστής** (Compiler) • **Διερμηνευτής** (Interpreter)	Ειδικό λογισμικό που μετατρέπει το πηγαίο πρόγραμμα σε μορφή δυαδική, και συνεπώς κατανοητή από τη μηχανή. • Ο **μεταγλωττιστής (compiler)** λαμβάνει ως είσοδο όλο το πηγαίο πρόγραμμα και το μετατρέπει σε ισοδύναμο σε γλώσσα μηχανής (δυαδική μορφή). • Ο **διερμηνευτής (interpreter)** λαμβάνει ως είσοδο μία εντολή του πηγαίου προγράμματος, τη μεταφράζει και εκτελεί

	αμέσως μία ισοδύναμη ακολουθία εντολών σε γλώσσα μηχανής.
Αντικείμενο πρόγραμμα (Object program)	**Μετά τη μεταγλώττιση του πηγαίου προγράμματος, το ισοδύναμο πρόγραμμα σε γλώσσα μηχανής ονομάζεται αντικείμενο (object program).** Συνήθως, δεν είναι σε θέση να εκτελεστεί και χρειάζεται να *συνδεθεί* με άλλα τμήματα προγράμματος που ονομάζονται *βιβλιοθήκες της γλώσσας*.
Βιβλιοθήκες (Libraries)	**Έτοιμες ενότητες (modules) *αντικείμενου προγράμματος* της γλώσσας**, απαραίτητες για την παραγωγή του εκτελέσιμου προγράμματος.
Συνδέτης-Φορτωτής (Linker-Loader)	Ειδικό λογισμικό που **συνδέει το αντικείμενο πρόγραμμα με τις βιβλιοθήκες της γλώσσας**. Το αποτέλεσμα της σύνδεσης είναι το εκτελέσιμο πρόγραμμα (executable program).
Εκτελέσιμο πρόγραμμα (Executable program)	**Μετά τη μεταγλώττιση και σύνδεση το παραγόμενο πρόγραμμα είναι πλέον σε θέση να εκτελεστεί από τον υπολογιστή**.
Συντάκτης (Editor)	**Επεξεργαστής κειμένου, εξειδικευμένος για τη συγγραφή του πηγαίου προγράμματος.**

Η **διαδικασία μεταγλώττισης και σύνδεσης προγράμματος** φαίνεται σχηματικά παρακάτω:



Συντακτικά – Λογικά λάθη

Κατά τη δημιουργία ενός προγράμματος σχεδόν πάντα ενυπάρχουν **λάθη**. Τα λάθη τα χωρίζουμε σε **δύο κατηγορίες** :

- Τα **συντακτικά λάθη** **ανιχνεύονται κατά την διαδικασία της μεταγλώττισης ή διερμήνευσης. Αφορούν παραβιάσεις του τυπολογικού και συντακτικού της γλώσσας**. (π.χ. μία εντολή έχει γραφτεί συντακτικά λάθος ή έχει ξεχαστεί μία παρένθεση).
  Ο μεταφραστής υποδεικνύει τα λάθη με κατάλληλα μηνύματα και ο προγραμματιστής πρέπει να επιστρέψει στο πηγαίο πρόγραμμα, να διορθώσει τα λάθη και να το επανυποβάλλει για μεταγλώττιση.

- Τα **λογικά λάθη** **είναι και τα πλέον δύσκολα στην ανίχνευσή τους. Έχουν να κάνουν με σφάλματα στη λογική επίλυσης του προβλήματος ή λανθασμένης διατύπωσης του αλγορίθμου** (π.χ. το πρόγραμμα παράγει άλλα αποτελέσματα κι όχι τα ζητούμενα!).
  Δεν ανιχνεύονται από τον μεταφραστή και πρέπει ο προγραμματιστής να κοπιάσει (με τη βοήθεια κατάλληλων εργαλείων και μεθόδων) για να τα εντοπίσει και να τα διορθώσει.

Ένα παραγόμενο εκτελέσιμο πρόγραμμα είναι σίγουρο ότι δεν έχει συντακτικά λάθη. Όμως, δεν είναι σίγουρο αν δεν έχει λογικά λάθη, παρ' όλους τους ελέγχους που μπορεί να κάνει ο προγραμματιστής.

Όσο πιο πολύπλοκο είναι ένα πρόγραμμα τόσο περισσότερη είναι η πιθανότητα να ενυπάρχουν, χωρίς να έχουν εντοπιστεί, λογικά λάθη, ακόμα και μήνες ή χρόνια μετά τη λειτουργία του! Γι' αυτό κατά καιρούς ή σε τακτά χρονικά διαστήματα οι εταιρείες παραγωγής λογισμικού διορθώνουν τα προγράμματά τους (π.χ. σκεφτείτε το λειτουργικό σύστημα των Windows).

*Η **διαδικασία εντοπισμού των λογικών λαθών από τον προγραμματιστή ονομάζεται εκσφαλμάτωση (debugging).***

Τα σύγχρονα προγραμματιστικά περιβάλλοντα περιέχουν οπωσδήποτε 3 ειδών προγράμματα:

- Τον συντάκτη,
- Τον μεταγλωττιστή και
- Τον συνδέτη - φορτωτή.

Πέραν αυτών, περιέχουν όλα εκείνα τα εργαλεία που διευκολύνουν την εύκολη, ταχύτατη σύνταξη, διόρθωση και συντήρηση των προγραμμάτων γι' αυτό και πολλές φορές ονομάζονται ως **RAD περιβάλλοντα** (**R**apid **A**pplication **D**evelopment)[6].

Όλα αυτά παρέχονται με έναν ενιαίο και λειτουργικό τρόπο στον προγραμματιστή.

Ερωτήσεις κατανόησης

1. Τί είναι ο προγραμματισμός;

2. Ποιά είναι τα κοινά χαρακτηριστικά μεταξύ των φυσικών και τεχνητών γλωσσών;

[6] Ονομάζονται, επίσης και *ολοκληρωμένα προγραμματιστικά περιβάλλοντα* (**I**ntegrated **D**evelopment **E**nvironments – **IDE**). Μάλιστα, αυτός ο όρος είναι πιο δημοφιλής.

3. Ποιές είναι οι διαφορές μεταξύ των φυσικών και τεχνητών γλωσσών;

4. Τί είναι η ιεραρχική σχεδίαση ενός προγράμματος (ιεραρχικός προγραμματισμός). Πώς γίνεται η παράσταση της ιεραρχικής σχεδίασης;

5. Σχεδιάστε ένα ιεραρχικό διάγραμμα που να δείχνει τις λειτουργίες που πρέπει να γίνουν και τις σχέσεις μεταξύ τους ώστε να υπολογίσουμε το τελικό αποτέλεσμα (προάγεται-απορρίπτεται) ενός μαθητή στο τέλος του σχολικού έτους.
(Λάβετε υπόψη βαθμούς και απουσίες).

6. Τί είναι ο τμηματικός προγραμματισμός; Τί πλεονεκτήματα προσφέρει;

7. Περιγράψτε τις αρχές και τα πλεονεκτήματα του δομημένου προγραμματισμού.

8. Δίνεται ο παρακάτω αλγόριθμος (μη δομημένος):

```
Αλγόριθμος Μεγαλύτερος_του_100
    Διάβασε α
    Αν α > 100 τότε GOTO 1
    Αν α < 100 τότε GOTO 2
    Αν α = 100 τότε GOTO 3
    GOTO 4
    1: Εκτύπωσε "Είναι μεγαλύτερος του 100"
    GOTO 4
    2: Εκτύπωσε "Είναι μικρότερος του 100"
    GOTO 4
    3: Εκτύπωσε "Είναι ίσος με το 100!"
    4: Εκτύπωσε "Τέλος"
Τέλος Μεγαλύτερος_του_100
```

Ξαναγράψτε τον αλγόριθμο με βάση τις αρχές του δομημένου προγραμματισμού.

9. Δίνεται ο παρακάτω αλγόριθμος (μη δομημένος):

```
Αλγόριθμος Εκτύπωση_των_100_πρώτων_ζυγών_αριθμών
    X ← 1
    1: Αν X > 100 τότε GOTO 4
    Αν X MOD 2 = 0 τότε GOTO 2
    3: X ← X + 1
    GOTO 1
    2: Εκτύπωσε X
    GOTO 3
    4: Εκτύπωσε "Τέλος εκτύπωσης"
Τέλος Εκτύπωση_των_100_πρώτων_ζυγών_αριθμών
```

Προφανώς, πρόκειται για μία επαναληπτική διαδικασία, αλλά εκφράζεται με αδόμητο τρόπο. Επαναδιατυπώστε τον αλγόριθμο με βάση τις αρχές του δομημένου προγραμματισμού.

10. Τί είναι ο αντικειμενοστραφής προγραμματισμός;

11. Τί είναι ένα προγραμματιστικό περιβάλλον;

12. Ποια η διαφορά μεταξύ του μεταγλωττιστή (compiler) και του διερμηνευτή (interpreter);

13. Απαντήστε με Σωστό/Λάθος στις παρακάτω προτάσεις:

    a. Ο μεταγλωττιστής δέχεται ως είσοδο το αντικείμενο πρόγραμμα και παράγει το πηγαίο.
    b. Ο προγραμματιστής γράφει το πηγαίο πρόγραμμα στον συντάκτη.
    c. Μετά την μεταγλώττιση του πηγαίου προγράμματος παράγεται αμέσως το εκτελέσιμο.
    d. Ο συνδέτης-φορτωτής είναι ειδικό λογισμικό που διασυνδέει το αντικείμενο πρόγραμμα με τις βιβλιοθήκες της γλώσσας.
    e. Η εκτέλεση του προγράμματος με διερμηνευτή είναι πιο αργή σε σχέση με την εκτέλεση του προγράμματος που παράγεται από τον μεταγλωττιστή.
    f. Το πηγαίο πρόγραμμα είναι κατανοητό και από τον άνθρωπο και από τη μηχανή.
    g. Το εκτελέσιμο πρόγραμμα είναι μεταφρασμένο σε γλώσσα μηχανής.

14. Περιγράψτε σχηματικά τη διαδικασία μετάφρασης και σύνδεσης του πηγαίου προγράμματος ώστε να παραχθεί το εκτελέσιμο.

15. Ποιά είναι τα 3 είδη προγραμμάτων που πρέπει να έχει ένα σύγχρονο προγραμματιστικό περιβάλλον;

16. Ποιές οι διαφορές μεταξύ συντακτικών και λογικών λαθών.

17. Ο φίλος ισχυρίζεται: «Είναι αδύνατον ένα εκτελέσιμο πρόγραμμα να έχει συντακτικά λάθη, μπορεί όμως να έχει λογικά». Έχει δίκιο;

Παράρτημα

Κατηγορίες γλωσσών προγραμματισμού

Στον παρακάτω πίνακα παρουσιάζονται, με συνοπτικό τρόπο, οι κατηγορίες των γλωσσών προγραμματισμού.

Κατηγορία	Περιγραφή

Γλώσσα μηχανής	**Το πρόγραμμα περιέχει εντολές που είναι σε δυαδική μορφή, άμεσα κατανοητή από τον Η/Υ** (όχι όμως από τον άνθρωπο). **Δηλαδή, το πρόγραμμα αποτελείται από ακολουθίες 0 και 1** .
	
	Παράδειγμα:
	`10101000 00001010`
	`11000000 00000001`
	……
	
	Πλεονεκτήματα :
	▪ Ταχύτατη εκτέλεση των εντολών.
	▪ Δεν απαιτείται μεταφραστικό πρόγραμμα.
	
	Μειονεκτήματα :
	▪ Το γράψιμο του προγράμματος είναι μία ιδιαίτερα επίπονη και χρονοβόρα διαδικασία.
	▪ Απαιτείται βαθιά γνώση της αρχιτεκτονικής του Η/Υ.
	▪ Το πρόγραμμα «τρέχει» μόνο σε συγκεκριμένο τύπο του Η/Υ (*δηλαδή, με συγκεκριμένο τύπο επεξεργαστή*).

Γλώσσες χαμηλού επιπέδου ή Συμβολικές γλώσσες	**Οι εντολές που είναι σε μορφή 0 και 1 αντικαθίστανται από μνημονικά (συμβολικά) ονόματα**. Για παράδειγμα, η εντολή `100001100` αντικαθίστανται από το μνημονικό όνομα `ADD`.

	Ένα δείγμα χρήσης θα ήταν : `INDEX =$01` {*βάλε στην INDEX την τιμή 1*} `ADD INDEX` {*πρόσθεσε την τιμή της INDEX στον συσσωρευτή*} `LDA #10` {*φόρτωσε στο συσσωρευτή την τιμή 10*} `CLA` {*καθάρισε το συσσωρευτή*} ……. **Πλεονεκτήματα** : ▪ Ταχύτατη εκτέλεση των εντολών. ▪ Η μορφή του προγράμματος είναι καλύτερα κατανοητή από τον άνθρωπο σε σχέση με τη γλώσσα μηχανής. **Μειονεκτήματα** : ▪ Η αντιστοιχία 1 προς 1 με τις εντολές της γλώσσας παρέμενε. ▪ Απαιτείται η χρήση ενός μεταφραστικού προγράμματος ώστε οι συμβολικές εντολές να μετατραπούν στις αντίστοιχες δυαδικές. Το ειδικό αυτό πρόγραμμα ονομάζεται ***συμβολομεταφραστής (assembler).*** ▪ Το γράψιμο του προγράμματος εξακολουθεί να είναι μία ιδιαίτερα επίπονη και χρονοβόρα διαδικασία. ▪ Απαιτείται βαθιά γνώση της αρχιτεκτονικής του Η/Υ. ▪ Το πρόγραμμα «τρέχει» μόνο στο συγκεκριμένο τύπο του Η/Υ (*δηλαδή, με συγκεκριμένο τύπο επεξεργαστή*).

Γλώσσες υψηλού επιπέδου	**Λέγονται έτσι διότι τα προγράμματα διατυπωμένα σε μία τέτοια γλώσσα είναι άμεσα κατανοητά από τον άνθρωπο** (αλλά όχι από τον Η/Υ) **αφού χρησιμοποιείται μία γλώσσα που είναι αρκετά περιγραφική όπως μία φυσική γλώσσα**.
	
	Παράδειγμα (σε γλώσσα BASIC),
	
	`INPUT "Δώσε την τελική τιμή" ; N`
	`SUM = 0`
	
	`For INDEX = 1 to N`
	`SUM = SUM + INDEX`
	`Next`
	
	Πλεονεκτήματα:
	▪ Η μορφή του προγράμματος είναι εύκολα κατανοητή από τον άνθρωπο σε σχέση με τη γλώσσα μηχανής ή τη συμβολική γλώσσα.

	▪ Το γράψιμο του προγράμματος δεν είναι πλέον μία ιδιαίτερα επίπονη και χρονοβόρα διαδικασία όπως συμβαίνει με τη γλώσσα μηχανής ή τη συμβολική γλώσσα. ▪ Δεν απαιτείται σχεδόν καμία γνώση της αρχιτεκτονικής του Η/Υ. Συνεπώς, είναι ανεξάρτητα από την αρχιτεκτονική του Η/Υ. ▪ Το πρόγραμμα «τρέχει» σε όλους τους τύπους Η/Υ αρκεί να υπάρχει το κατάλληλο μεταφραστικό πρόγραμμα. Συνεπώς ένα χαρακτηριστικό τους είναι η ***μεταφερσιμότητα***, δηλαδή ένα πρόγραμμα υψηλού επιπέδου να εκτελείται, με ελάχιστες μετατροπές, σε πολλούς τύπους Η/Υ. ▪ Η εκμάθηση της γλώσσας είναι εύκολη. ▪ Η διόρθωση λαθών και η συντήρηση των προγραμμάτων είναι ευκολότερη. **Μειονεκτήματα** : ▪ Απαιτείται η χρήση ενός μεταφραστικού προγράμματος ώστε οι εντολές να μετατραπούν σε πολλές δυαδικές εντολές (δεν έχουμε εδώ αντιστοιχία 1 προς 1). Έχουμε δύο ειδών μεταφραστικά προγράμματα: τους ***μεταγλωττιστές (compilers)*** και τους ***διερμηνείς (interpreters)***. ▪ Το πρόγραμμα «τρέχει» πιο αργά σε σχέση με τα προγράμματα των συμβολικών γλωσσών ή της γλώσσας μηχανής.

Κατηγορίες γλωσσών προγραμματισμού υψηλού επιπέδου

- **Διαδικασιακές ή αλγοριθμικές γλώσσες (Procedural)** : λέγονται έτσι διότι επιτρέπουν την εύκολη υλοποίηση αλγορίθμων π.χ. Pascal, Basic, C.
- **Αντικειμενοστραφείς γλώσσες (object – oriented) :** λέγονται έτσι διότι ακολουθούν την τεχνική του αντικειμενοστραφούς προγραμματισμού π.χ. C++, C#, Java, Visual Basic. Αυτή η τεχνική σχεδίασης προγραμμάτων είναι πολύ δημοφιλής και ως εκ τούτου και οι αντίστοιχες γλώσσες.
- **Συναρτησιακές γλώσσες (functional)** π.χ. LISP
- **Μη-διαδικασιακές γλώσσες** π.χ. PROLOG

- **Γλώσσες ερωταπαντήσεων (Query languages) ή 4ης γενιάς** π.χ. SQL

Μία άλλη κατηγοριοποίηση των γλωσσών υψηλού επιπέδου γίνεται **με κριτήριο τον τομέα στον οποίο είναι κατάλληλες, δηλαδή για ποιό είδος προβλημάτων είναι κατάλληλες να επιλύουν**:

- **Γλώσσες γενικής χρήσης** : Σκοπός τους είναι να επιλύουν πάσης φύσεως προβλήματα (αριθμητικά, εμπορικά, επιστημονικά). Τέτοιες είναι η Basic, Pascal. Μερικές γλώσσες, όμως, έχουν δημιουργηθεί αποκλειστικά για να επιλύουν ευκολότερα συγκεκριμένους τύπους προβλημάτων όπως :
  - **Γλώσσες επιστημονικής κατεύθυνσης** : π.χ. FORTRAN.
  - **Γλώσσες εμπορικής κατεύθυνσης** : π.χ. COBOL.
- **Γλώσσες προγραμματισμού συστημάτων** π.χ. C.
- **Γλώσσες τεχνητής νοημοσύνης** π.χ. LISP, PROLOG.
- **Γλώσσες ειδικής χρήσης.** Σκοπός τους είναι να επιλύουν ειδικού τύπου προβλήματα όπως διαχείριση Βάσεων Δεδομένων κ.α. π.χ. SQL .

Διαδικασιακές γλώσσες κατά χρονολογική σειρά εμφάνισης.

FORTRAN (1957)	Κατάλληλη για επίλυση επιστημονικών προβλημάτων (αριθμητικές εφαρμογές).
COBOL (1960)	Κατάλληλη για επίλυση εμπορικών προβλημάτων (εφαρμογές μισθοδοσίας, λογιστικές κλπ.).
ALGOL (1960)	Δημιουργήθηκε με σκοπό την ανάπτυξη προγραμμάτων κυρίως για επιστημονικές εφαρμογές ως ανταγωνιστική της Fortran. Δεν χρησιμοποιήθηκε, όμως, ευρέως στην πράξη. Πρωτοπαρουσίασε τις αρχές του δομημένου προγραμματισμού.

PL/1 (μέσα ’60)	Προσπάθησε να συνδυάσει τις δυνατότητες των γλωσσών προσανατολισμένων για εμπορικές και επιστημονικές εφαρμογές χωρίς όμως να γνωρίσει επιτυχία.
BASIC (μέσα ’60)	Δημιουργήθηκε με σκοπό την εκπαίδευση των αρχάριων στον προγραμματισμό. Σκοπός της BASIC είναι να γράφονται μικρά προγράμματα που κατόπιν εκτελούνται με τη βοήθεια διερμηνέα (interpreter). Στις μέρες μας αποτελεί μία πανίσχυρη, γενικής χρήσης, γλώσσα στην αντικειμενοστραφής της έκδοση Visual Basic.
PASCAL (1970)	Γλώσσα γενικής χρήσης. Στηρίχθηκε στην ALGOL. Είναι η καταλληλότερη γλώσσα για να μάθει κάποιος δομημένο προγραμματισμό.
C (αρχές ’70)	Περιέχει αρκετά κοινά χαρακτηριστικά με τη Pascal για την ανάπτυξη δομημένων εφαρμογών αλλά παράλληλα ενσωματώνει και χαρακτηριστικά γλώσσας χαμηλού επιπέδου. Θεωρείται κατάλληλη για την ανάπτυξη λειτουργικών συστημάτων (π.χ. Unix). Στις μέρες μας αποτελεί μία πανίσχυρη, γενικής χρήσης, γλώσσα στην αντικειμενοστραφής της έκδοση C#, C++.
ADA (1979)	Γλώσσα γενικής χρήσης που δίνει έμφαση στο θέμα της αξιοπιστίας των προγραμμάτων. Γι’ αυτό, χρησιμοποιήθηκε πρωτίστως για στρατιωτικές εφαρμογές.

Αντικειμενοστραφείς γλώσσες κατά χρονολογική σειρά εμφάνισης.

Smalltalk (αρχές ’80)	Η πρώτη αντικειμενοστραφής γλώσσα με ολοκληρωμένο μάλιστα περιβάλλον ανάπτυξης προγραμμάτων.
C++ (τέλη ’80)	Αποτελεί μία μετεξέλιξη της C στο χώρο του αντικειμενοστραφούς προγραμματισμού και χρησιμοποιείται αρκετά στην ανάπτυξη λειτουργικών συστημάτων (π.χ. Windows) αλλά και άλλου τύπου εφαρμογών. Θεωρείται σήμερα μία κορυφαία γλώσσα.
Python (τέλη ’80)	Απλή γλώσσα που υποστηρίζει διαδικασιακό και αντικειμενοστραφή προγραμματισμό. Κατάλληλη για αρχάριους επειδή έχει απλό συντακτικό. Παρέχει πολλές επεκτάσεις μέσω των αντίστοιχων βιβλιοθηκών.
JAVA (μέσα ’90)	Γλώσσα ειδικά σχεδιασμένη για την ανάπτυξη εφαρμογών στο Internet. Σκοπός της είναι να γράφονται προγράμματα που θα εκτελούνται, χωρίς μετατροπές, σε Η/Υ με διαφορετικά λειτουργικά συστήματα. Περιέχει αρκετά στοιχεία από τη C++.
C# (2002)	Δημιουργήθηκε ως ανταγωνιστική της JAVA. Σκοπός της είναι να συνδυάσει την ευχρηστία της Basic και τη δυναμική της C++ για την ανάπτυξη εφαρμογών που θα εκτελούνται σε Η/Υ με διαφορετικά λειτουργικά συστήματα.

Συναρτησιακές γλώσσες κατά χρονολογική σειρά εμφάνισης.

LISP (μέσα ’60)	Δημιουργήθηκε για την ανάπτυξη προγραμμάτων στο χώρο της τεχνητής νοημοσύνης.

Μη – διαδικασιακές γλώσσες κατά χρονολογική σειρά εμφάνισης.

PROLOG (αρχές ’70)	Δημιουργήθηκε για την ανάπτυξη προγραμμάτων στο χώρο της τεχνητής νοημοσύνης.

Γλώσσες 4ης γενιάς ή ερωταπαντήσεων

SQL (Structured Query Language)	Δεν απευθύνεται μόνο σε προγραμματιστές αλλά και χρήστες. Ο χρήστης μπορεί, σχετικά εύκολα, να υποβάλει ερωτήσεις στο σύστημα ή να αναζητά πληροφορίες από μία Βάση Δεδομένων. Παράδειγμα, SELECT Επώνυμο, Όνομα FROM Μαθητές WHERE Τάξη = ‘’Γ2’’ Η παραπάνω πρόταση της γλώσσας SQL θα κάνει μία αναζήτηση στη βάση δεδομένων των μαθητών και θα εμφανίσει τα ονοματεπώνυμα των μαθητών του Γ2 μόνο.

Οπτικός προγραμματισμός.

Είναι η δυνατότητα να δημιουργούμε, με γραφικό τρόπο, ολόκληρο το περιβάλλον της εφαρμογής όπως για παράδειγμα τα μενού και τα πλαίσια διαλόγου και άλλα παράθυρα της εφαρμογής.

Τα ολοκληρωμένα περιβάλλοντα ανάπτυξης προγραμμάτων (**IDE**) μας παρέχουν τα εργαλεία για να σχεδιάσουμε το γραφικό περιβάλλον (**GUI - Graphical User Interface**) της εφαρμογής μας. Ένα τέτοιο IDE είναι το Microsoft Visual Studio.

Ο οπτικός προγραμματισμός εκμεταλλεύεται τις δυνατότητες των γραφικών περιβαλλόντων επικοινωνίας (π.χ. Windows, Android, MacOs κ.λπ.).

Καθοδηγούμενος από γεγονότα προγραμματισμός

Είναι η δυνατότητα να εκτελούνται οι διάφορες λειτουργίες του προγράμματος με την ενεργοποίηση ενός γεγονότος (ή αλλιώς συμβάντος)[7]. Για παράδειγμα, αν κάνουμε κλικ σε κάποια εντολή ενός μενού ή σε κάποιο κουμπί σε ένα παράθυρο της εφαρμογής τότε θα εκτελεστεί μία λειτουργία.

Τα σύγχρονα προγραμματιστικά περιβάλλοντα (IDE) είναι κτισμένα πάνω στις αρχές του οπτικού και καθοδηγούμενου από γεγονότα προγραμματισμού.

Παράλληλος προγραμματισμός

Σήμερα υπάρχουν μεγάλοι Η/Υ που διαθέτουν στο εσωτερικό τους πολλούς επεξεργαστές. Οι επεξεργαστές αυτοί μοιράζονται την ίδια μνήμη και λειτουργούν παράλληλα. Έτσι, την ίδια χρονική στιγμή, μπορούν να εκτελούνται διαφορετικές εντολές του προγράμματος.

Για να εκμεταλλευτούμε αυτήν την ιδιαίτερη ισχύ των Η/Υ θα πρέπει **τα προγράμματα να είναι φτιαγμένα με τέτοιον τρόπο ώστε διαφορετικά τμήματά του να εκτελούνται**

[7] Ο αγγλικό όρος είναι event και συνολικά ο προγραμματισμός καθοδηγούμενος από γεγονότα ως event-driven programming.

παράλληλα. Για τον σκοπό αυτό έχουν αναπτυχθεί ιδιαίτερες γλώσσες προγραμματισμού όπως η OCCAM. Επίσης, παραδοσιακές γλώσσες προγραμματισμού, όπως C# και Visual Basic, περιλαμβάνουν επεκτάσεις που υποστηρίζουν την παράλληλη εκτέλεση τμημάτων του προγράμματος.

Από τι εξαρτάται η επιλογή μίας γλώσσας προγραμματισμού;

- Από το είδος του προβλήματος (εμπορικό, αριθμητικό κ.α.)
- Από τον Η/Υ στον οποίο θα εκτελεστεί το πρόγραμμα.
- Από τη διαθέσιμη γλώσσα ή προγραμματιστικό περιβάλλον στο οποίο θα αναπτυχθεί το πρόγραμμα.
- Από τις γνώσεις και την εμπειρία του προγραμματιστή.

Με την ευρεία διάδοση των γραφικών περιβαλλόντων επικοινωνίας (π.χ. Windows, MacOS κ.λπ.) δημιουργήθηκαν παραλλαγές κάποιων γλωσσών που απευθύνονται σε αυτά. Τέτοιες είναι η Visual Basic, Visual C++, Delphi (Visual Pascal), C# κ.α. Αυτές οι γλώσσες ακολουθούν τη φιλοσοφία του *οπτικού* και του *καθοδηγούμενου-από-γεγονότα προγραμματισμού* χωρίς να απορρίπτουν τις αρχές του δομημένου προγραμματισμού.

ΤΕΛΟΣ ΣΗΜΕΙΩΣΕΩΝ ΚΕΦΑΛΑΙΟΥ 6