

Δομημένος Προγραμματισμός

**ΔΙΑΛΕΞΕΙΣ ΘΕΩΡΙΑΣ**

Δομημένος Προγραμματισμός

**#1**

**ΕΙΣΑΓΩΓΗ**

## #1.1 Υπολογιστής (Υλικό – Λογισμικό)

Ο **υπολογιστής** είναι μία μηχανή η οποία επεξεργάζεται δεδομένα τα οποία εισάγοντα σε αυτόν με κατάλληλο τρόπο και παράγει αποτελέσματα. Αυτό γίνεται με την καθοδήγηση ενός συνόλου εντολών που λέγεται **πρόγραμμα**.

Ο υπολογιστής αποτελείται από διάφορα τμήματα (τροφοδοτικό, μνήμη, CPU κλπ.) που όλα μαζί λέγονται **υλικό** (hardware).

Το πρόγραμμα λέγεται **λογισμικό** (software) και χωρίζεται σε πρόγραμμα ελέγχου της λειτουργίας του υπολογιστή (λειτουργικό) και σε πρόγραμμα επεξεργασίας δεδομένων (εφαρμογή).

## #1.2 Λογικές Μονάδες

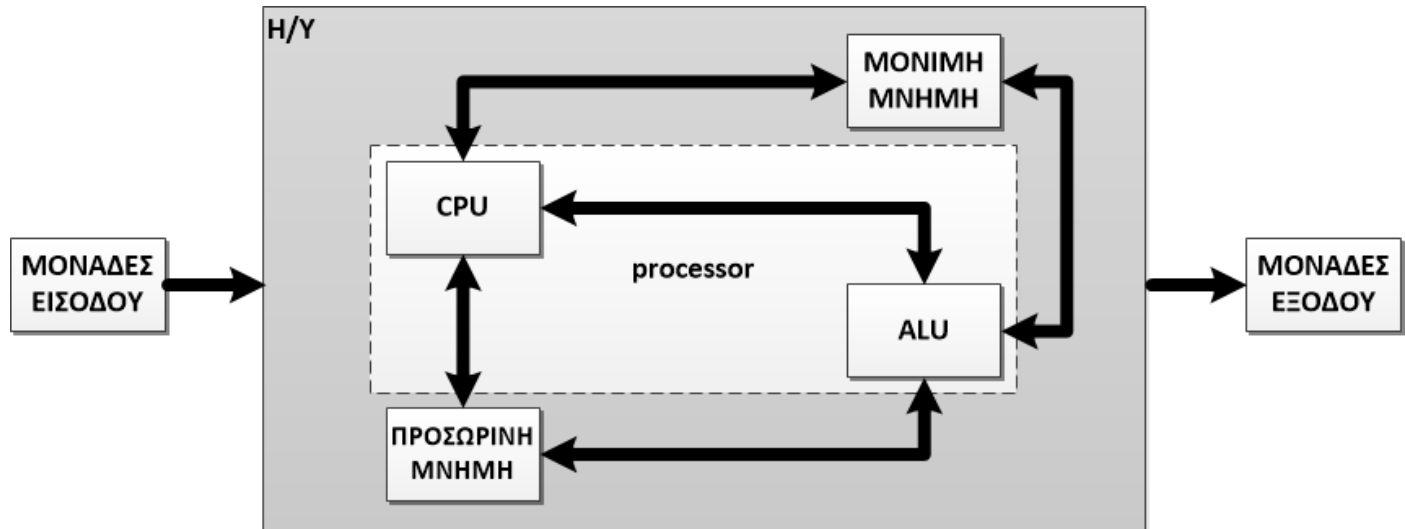
Ο υπολογιστής ανεξάρτητα από την ταχύτητα με την οποία δουλεύει ή ακόμα και τον τύπο ή το μέγεθός του, αποτελείται από λειτουργικά τμήματα που καλούνται **λογικές μονάδες**.

Οι λογικές μονάδες του υπολογιστή είναι:

- Μονάδα εισόδου (*Input Device*),
- Μονάδα εξόδου (*Output Device*),
- Πρωτεύουσα μνήμη ή προσωρινή (*RAM*),
- Αριθμητική και λογική μονάδα (*ALU*),
- Κεντρική μονάδα επεξεργασίας (*CPU*),
- Δευτερεύουσα μνήμη ή μόνιμη (*HDD, TAPE, CD, DVD*).

## #1.3 Σχηματική παράσταση των Λογικών Μονάδων του Η/Υ

Πιο κάτω φαίνονται οι λογικές μονάδες του υπολογιστή.



## #1.4 Μονάδες εισόδου

**Μονάδες εισόδου** (*Input Devices*) είναι οι συσκευές εκείνες οι οποίες εισάγουν τις πληροφορίες από το φυσικό περιβάλλον ή από άλλες υπολογιστικές μονάδες, στον υπολογιστή για να τις επεξεργαστεί.

Μονάδες εισόδου:

- πληκτρολόγιο (*keyboard*),
- ποντίκι (*mouse*),
- σαρωτής (*scanner*),
- αναγνώστης γραμμωτού κώδικα (*barcode reader*),
- μικρόφωνο (*microphone*),
- ψηφιακή φωτογραφική μηχανή (*camera*) κλπ.

## #1.5 Μονάδες εξόδου

**Μονάδες εξόδου** είναι οι συσκευές εκείνες στις οποίες καταλήγουν οι επεξεργασμένες πληροφορίες από τον υπολογιστή προς διάθεση ή περεταίρω επεξεργασία.

Μονάδες εξόδου:

- οθόνη (*monitor*),
- εκτυπωτής (*printer*),
- ηχεία (*speakers*),
- προβολέας (*projector*),
- δρομολογητής δικτύου (*network hub*) κλπ.

## #1.6 Πρωτεύουσα μνήμη

**Πρωτεύουσα μνήμη** είναι η μονάδα εκείνη στην οποία πηγαίνουν τα δεδομένα για να επεξεργαστούν, αλλά και τα επεξεργασμένα δεδομένα επίσης που περιμένουν για την διάθεσή τους προς άλλα συστήματα έξω από την υπολογιστή.

Η Πρωτεύουσα μνήμη είναι μονάδα σχετικά μικρού αποθηκευτικού μεγέθους αλλά πολύ γρήγορης πρόσβασης.

Επειδή τα δεδομένα δεν μένουν πολύ χρόνο σε αυτή, αλλά και επειδή χάνονται με την διακοπή παροχής τροφοδοσίας, λέγεται και **προσωρινή μνήμη**.



## #1.7 Αριθμητική και λογική μονάδα

Η **Αριθμητική και Λογική Μονάδα (ALU)** είναι η μονάδα που είναι υπεύθυνη για την εκτέλεση αριθμητικών πράξεων (πρόσθεση, αφαίρεση, πολλαπλασιασμός κλπ.) μεταξύ των τιμών αλλά και την σύγκριση των περιεχομένων των θέσεων της μνήμης.

Οι πράξεις αλλά και οι συγκρίσεις γίνονται στο δυαδικό αριθμητικό σύστημα.

Η Αριθμητική και Λογική Μονάδα (ALU) αποτελείται από καταχωρητές για την προσωρινή αποθήκευση των τιμών στην είσοδο αλλά και στην έξοδό της.

Συμπεριλαμβάνεται συνήθως μέσα στη Κεντρική Μονάδα Επεξεργασίας και Ελέγχου (CPU).

## #1.8 Κεντρική Μονάδα Επεξεργασίας

Η **Κεντρική Μονάδα Επεξεργασίας (CPU)** είναι η μονάδα καθολικού ελέγχου του υπολογιστή. Είναι η μονάδα που δίνει τις εντολές για την λειτουργία όλων των άλλων μονάδων.

Η κεντρική μονάδα επεξεργασίας καθορίζει :

- την προτεραιότητα επικοινωνίας με τις άλλες λογικές μονάδες,
- τη διαδικασία εισόδου και εξόδου των δεδομένων,
- τη θέση της μνήμης που αυτά θα αποθηκευτούν,
- το πως και το πότε θα επεξεργαστεί τα δεδομένα η Αριθμητική και Λογική Μονάδα ALU κλπ.

## #1.9 Δευτερεύουσα μνήμη

Η **δευτερεύουσα μνήμη** είναι η μονάδα στην οποία αποθηκεύονται τα δεδομένα για πολύ μεγάλο χρονικό διάστημα και ανασύρονται από αυτή όποτε χρειαστεί.

Η δευτερεύουσα μνήμη είναι μονάδα μεγάλου αποθηκευτικού μεγέθους αλλά πολύ αργής πρόσβασης σε σχέση με την προσωρινή μνήμη.

Επειδή τα δεδομένα μένουν πολύ χρόνο σε αυτή, αλλά και επειδή δεν χάνονται με την διακοπή παροχής τροφοδοσίας, λέγεται και **μόνιμη μνήμη**.

## #1.10 Γλώσσες προγραμματισμού

- **Γλώσσα μηχανής** είναι η γλώσσα που καταλαβαίνει ο υπολογιστής και αποτελείται από αριθμοσειρές του δυαδικού αριθμητικού συστήματος που περιέχουν τα ψηφία 0 και 1.
- **Γλώσσα *assembly*** είναι η γλώσσα του επεξεργαστή. Είναι πιο κοντά σε γλώσσα υψηλού επιπέδου γιατί αποτελείται από συντετμημένες λέξεις της φυσικής γλώσσας. Μειονέκτημά της ότι χρειάζονται πολλές εντολές για μία εργασία.
- **Γλώσσα υψηλού επιπέδου** είναι η γλώσσα του προγραμματιστή που βρίσκεται πιο κοντά στην φυσική γλώσσα, έχει συγκεκριμένο συντακτικό και κάνει την ζωή των προγραμματιστών πιο εύκολη. Μεγάλο της πλεονέκτημα ότι με μία εντολή μπορούν να πραγματοποιηθούν πολλές εργασίες.

### #1.11.1 Βήματα επίλυσης ενός προβλήματος

Για να λύσουμε οποιοδήποτε υπολογιστικό πρόβλημα πρέπει να ακολουθήσουμε τα παρακάτω βήματα:

#### 1. **Καθορισμός προβλήματος**

Προσπαθούμε με την χρήση της φυσικής γλώσσας να περιγράψουμε όσο το δυνατόν σαφέστερα το προς επίλυση πρόβλημα.

#### 2. **Ανάλυση προβλήματος**

Καθορίζουμε τις πληροφορίες που έχουμε ως *δεδομένα εισόδου* και τα απαιτούμενα αποτελέσματα ως *δεδομένα εξόδου*. Επίσης καθορίζουμε και τις μεταβλητές που θα χρησιμοποιήσουμε.

## #1.11.2 Βήματα επίλυσης ενός προβλήματος

### 3. **Αλγόριθμος**

Καθορίζουμε τον ακριβή αριθμό αλλά και την σαφή μορφή των βημάτων που θα χρειαστούν για την επίλυση του προβλήματος σε πεπερασμένο χρόνο.

### 4. **Μετατροπή αλγορίθμου σε κώδικα**

Μετατρέπουμε τον αλγόριθμο σε πρόγραμμα εντολών με την χρήση της επιθυμητής γλώσσας προγραμματισμού.

### 5. **Εκτέλεση και έλεγχος προγράμματος**

Εκτελούμε το πρόγραμμα εισάγοντας το πλήθος των δεδομένων που είναι ικανό για να παράγουμε το επιθυμητό αποτέλεσμα.

## #1.12 Σχηματική παράσταση βημάτων επίλυσης προβλήματος



### #1.13.1 Κύκλος ανάπτυξης ενός προγράμματος

Για την δημιουργία ενός προγράμματος εντολών, ακολουθούμε τα πιο κάτω βήματα:

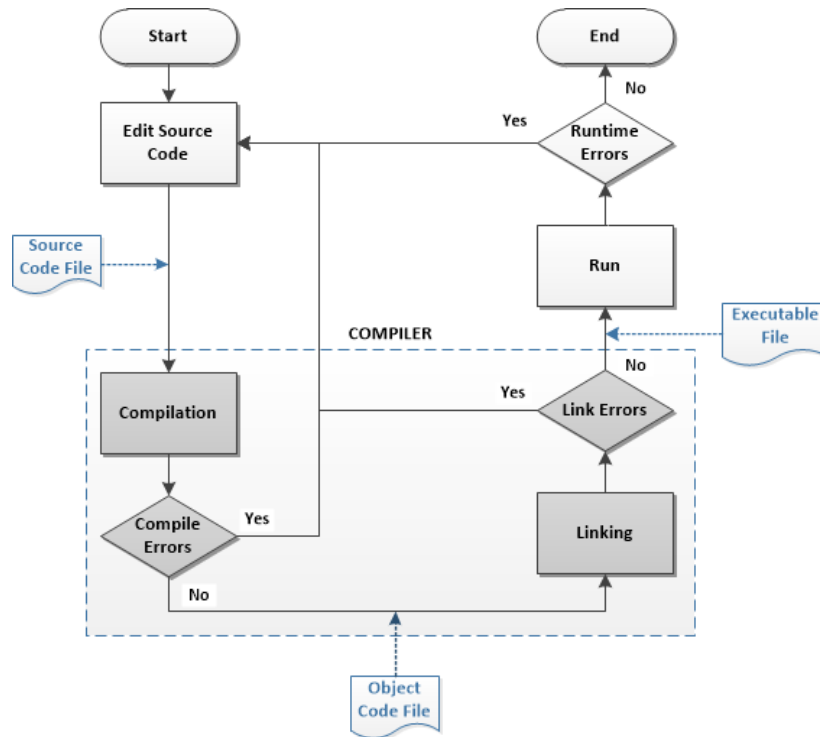
1. Χρησιμοποιούμε τον συντάκτη (*editor*) της γλώσσας προγραμματισμού ή οποιονδήποτε εξωτερικό συντάκτη, για την συγγραφή του **πηγαίου κώδικα** (*source code*).
2. Μεταγλωττίζουμε τον πηγαίο κώδικα σε **αντικειμενικό κώδικα** (*object code*) με την βοήθεια του μεταγλωττιστή (*compiler*), ελέγχοντας ταυτόχρονα για συντακτικά λάθη (*compile errors*). Ο αντικειμενικός κώδικας δεν θα δημιουργηθεί αν δεν έχουν διορθωθεί όλα τα συντακτικά λάθη.



### #1.13.2 Κύκλος ανάπτυξης ενός προγράμματος

3. Συνδέουμε (*linking*) με τον συνδέτη (*linker*) του compiler τον αντικειμενικό κώδικα, με τις βιβλιοθήκες συναρτήσεων της γλώσσας που χρησιμοποιήσαμε στον κώδικα και δημιουργούμε το **εκτελέσιμο αρχείο** (*executable file*).
4. Εκτελούμε (*τρέχουμε*) το πρόγραμμα (*run*) με την εισαγωγή ενός μικρού αλλά αντιπροσωπευτικού πλήθους δεδομένων, που θα μας βοηθήσει να πάρουμε τα προσδοκώμενα αποτελέσματα.
5. Εντοπίζουμε και διορθώνουμε τα **σημασιολογικά λάθη** (*run-time errors*) που παρουσιάζονται κατά τη διάρκεια της εκτέλεσης και που οφείλονται στη αστοχία σχεδιασμού και μπορούν να αλλοιώσουν τα προσδοκώμενα αποτελέσματα.

## #1.14 Σχηματική παράσταση Κύκλου Ανάπτυξης Προγράμματος



Δομημένος Προγραμματισμός

**#2**

**Ο ΑΛΓΟΡΙΘΜΟΣ**

### #2.1.1 Τι είναι αλγόριθμος

Η διαδικασία η οποία λύνει ένα συγκεκριμένο υπολογιστικό πρόβλημα σε πεπερασμένο χρόνο ονομάζεται **αλγόριθμος**.

Ο αλγόριθμος αποτελείται από έναν συγκεκριμένο αριθμό βημάτων ικανό να επιλύσει το πρόβλημα.

Τα βήματα πρέπει να είναι δομημένα με σαφή τρόπο, ώστε ο αλγόριθμος μετά το πέρας της διαδικασίας να παράξει σωστό αποτέλεσμα.

Ο αλγόριθμος έχει **είσοδο** που είναι τα δεδομένα που θα επεξεργαστεί, καθώς και **έξοδο** που είναι τα δεδομένα αυτά επεξεργασμένα μέσα σε ένα **συγκεκριμένο χρόνο**.

## #2.1.2 Τι είναι αλγόριθμος

Στο σχεδιασμό ενός αλγόριθμου πρέπει να λαμβάνουμε πάντα υπόψη το χρόνο περάτωσης της διαδικασίας επίλυσης, ο οποίος καθορίζει την ταχύτητα με την οποία ο αλγόριθμος θα επιλύσει το εκάστοτε πρόβλημα.

Άρα πάντοτε όταν καλούμαστε να σχεδιάσουμε έναν αλγόριθμο θα πρέπει να λαμβάνουμε υπόψη μας την είσοδο, τον ακριβή αριθμό βημάτων και τον χρόνο περάτωσης της όλης διαδικασίας για μία σωστή έξοδο.

## #2.2 Δομές αλγόριθμων

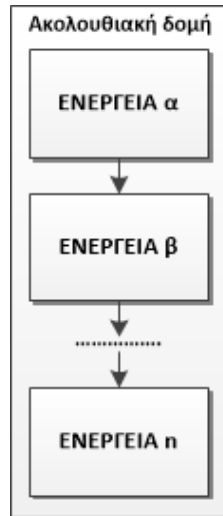
Στην επιστήμη των υπολογιστών έχουν καθοριστεί τρεις δομές ανάπτυξης αλγορίθμων. Οι δομές αυτές χρησιμοποιούνται είτε αυτόνομα είτε σε συνδυασμούς μέσα σε ένα πρόγραμμα.

Οι τρεις δομές αλγορίθμων είναι:

- Η **Ακολουθιακή** δομή
- Η Δομή **επιλογής**
- Η Δομή **επανάληψης**.

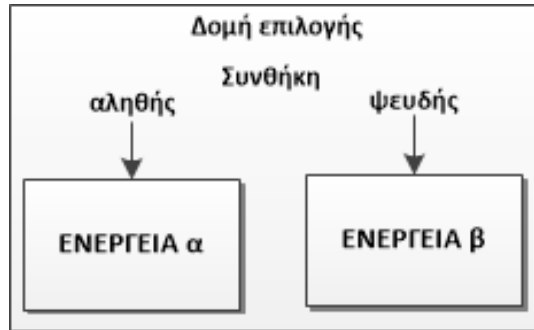
### #2.3.1 Ακολουθιακή δομή

Στην ακολουθιακή δομή ο αλγόριθμος αποτελείται από μια ακολουθία ενεργειών, που μπορεί να είναι είτε απλές εντολές που πρέπει να εκτελεστούν ή ακόμα και δομές επιλογής ή επανάληψης.



## #2.3.2 Δομή επιλογής

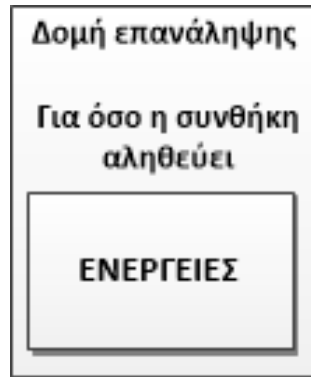
Στη δομή επιλογής θέτουμε εξ αρχής μία συνθήκη, η οποία πρέπει υποχρεωτικά να ελέγχεται για την ορθότητά της. Εάν το αποτέλεσμα της συνθήκης είναι αληθές τότε εκτελούμε μία ακολουθία ενεργειών ενώ εάν είναι ψευδές εκτελούμε μία άλλη ακολουθία ενεργειών. Και στις δύο περιπτώσεις μπορεί να επέρχεται και ο τερματισμός του αλγόριθμου.





### #2.3.3 Δομή επανάληψης

Στη δομή επανάληψης θέτουμε επίσης μία συνθήκη η οποία για όσο είναι αληθής θα εκτελείται μια ακολουθία ενεργειών. Η ακολουθία θα εκτελείται είτε για έναν συγκεκριμένο αριθμό επαναλήψεων ή μέχρι την εμφάνιση ενός συμβάντος. Η δομή επανάληψης λέγεται αλλιώς και βρόχος (*loop*).



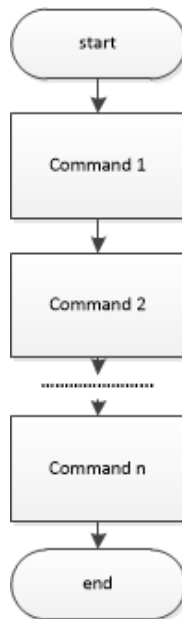
## #2.4 Αποτύπωση αλγόριθμου

Τους αλγόριθμους τους παριστάνουμε με δύο τρόπους.

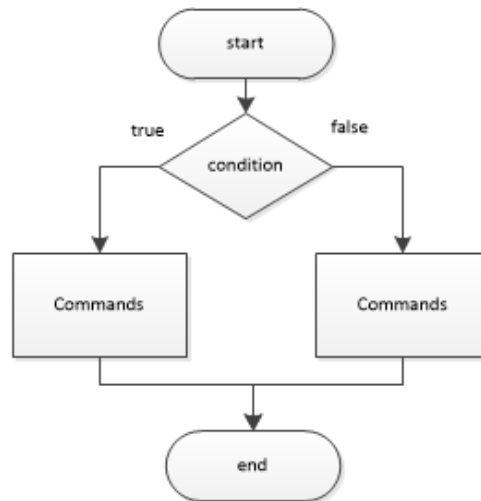
Με την **Ενοποιημένη Γλώσσα Μοντελοποίησης (UML)**, η οποία με την χρήση συγκεκριμένων γεωμετρικών σχημάτων μας δείχνει όλη την λειτουργία του αλγόριθμου.

Με την χρήση του **ψευδοκώδικα**, ο οποίος χρησιμοποιεί την φυσική γλώσσα με τέτοιο τρόπο ώστε, να γίνει κατανοητή η λειτουργία του αλγόριθμου. Ο ψευδοκώδικας είναι το τελευταίο βήμα πριν τη συγγραφή του πραγματικού κώδικα.

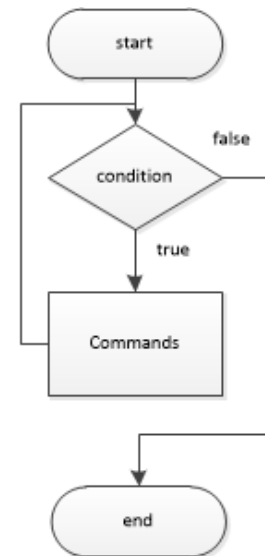
## #2.5.1 Οι τρεις δομές σε UML



sequence



decision



repetition

## #2.5.2 Οι τρεις δομές σε ψευδοκώδικα

```
Command a  
Command b  
Command c  
.....  
Command n
```

sequence

```
If (condition true)  
{  
  Commands  
}  
else (condition false)  
{  
  Commands  
}
```

decision

```
While (condition true)  
{  
  Commands  
}
```

repetition

## #2.6.1 Οι βασικοί αλγόριθμοι

Υπάρχουν κάποιοι αλγόριθμοι οι οποίοι έχουν ονομαστεί βασικοί λόγω του ότι χρησιμοποιούνται πάρα πολύ συχνά στην επίλυση προβλημάτων. Οι αλγόριθμοι αυτοί είναι οι εξής:

1. Αλγόριθμος άθροισης
2. Αλγόριθμος γινομένου
3. Αλγόριθμος ελάχιστου ή μέγιστου
4. Αλγόριθμος ταξινόμησης
  - a. με επιλογή
  - b. φυσαλίδας
  - c. με εισαγωγή

## #2.6.2 Οι βασικοί αλγόριθμοι

- 5. Αλγόριθμος αναζήτησης
  - a. Ακολουθιακή
  - b. Δυναμική

### #2.7.1 Αθροιστικός αλγόριθμος

Ο αλγόριθμος αυτός δέχεται στην είσοδο μια σειρά από ακέραιους και δίνει στην έξοδό του το άθροισμά τους.

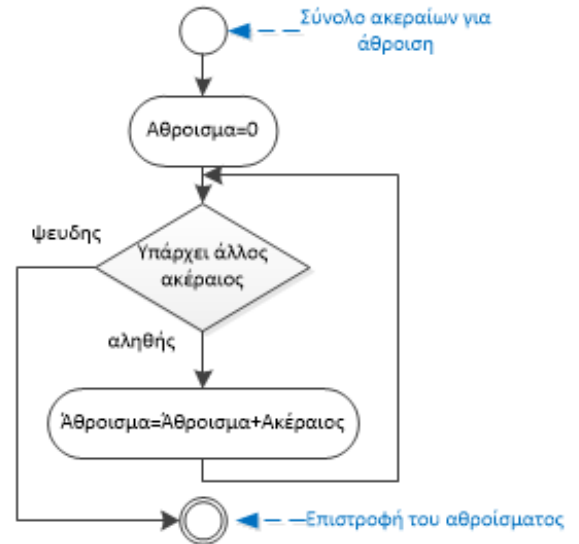
Αυτό γίνεται με την δημιουργία μιας μεταβλητής «**άθροισμα**» η οποία αρχικοποιείται στην τιμή **0**, και η οποία δέχεται κατά σειρά τις τιμές των ακεραίων μέσα σε μία δομή επανάληψης (οι τιμές των ακεραίων προστίθενται στην προηγούμενη τιμή του αθροίσματος).

## #2.7.2 Ψευδοκώδικας- διάγραμμα

Είσοδος: πλήθος Ακεραίων

Έξοδος: Άθροισμα Ακεραίων

```
{  
  Άθροισμα  $\leftarrow$  0  
  Για όσο(υπάρχει άλλος ακέραιος)  
    Άθροισμα  $\leftarrow$  Άθροισμα + Ακέραιος  
  Επέστρεψε Άθροισμα  
}
```





### #2.8.1 Αλγόριθμος γινόμενου

Ο αλγόριθμος αυτός δέχεται στην είσοδο μια σειρά από ακέραιους και δίνει στην έξοδό του το γινόμενο τους.

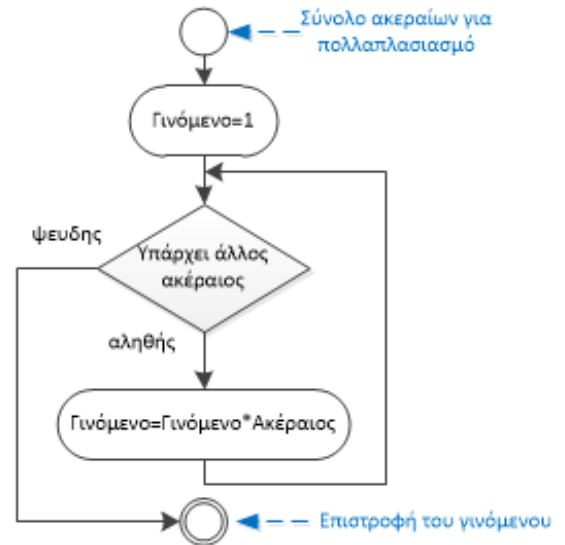
Αυτό γίνεται με την δημιουργία μιας μεταβλητής «**γινόμενο**» η οποία αρχικοποιείται στην τιμή **1**, και η οποία δέχεται κατά σειρά τις τιμές των ακεραίων μέσα σε μία δομή επανάληψης (οι τιμές των ακεραίων πολλαπλασιάζονται με την προηγούμενη τιμή του γινόμενου).

## #2.8.2 Ψευδοκώδικας- διάγραμμα

Είσοδος: πλήθος Ακεραίων

Έξοδος: Γινόμενο Ακεραίων

```
{  
  Γινόμενο  $\leftarrow$  1  
  Για όσο(υπάρχει άλλος ακέραιος)  
    Γινόμενο  $\leftarrow$  Γινόμενο * Ακέραιος  
  Επέστρεψε Γινόμενο  
}
```



## #2.9.1 Αλγόριθμος εύρεσης μεγαλύτερου

Ο αλγόριθμος αυτός δέχεται στην είσοδο μια σειρά από ακέραιους και δίνει στην έξοδό του τον μεγαλύτερο.

Αυτό γίνεται με την δημιουργία μιας μεταβλητής «**μεγαλύτερος**» στην οποία δίνουμε την τιμή ενός πολύ μικρού ακέραιου.

Εάν ο επόμενος ακέραιος είναι μεγαλύτερος από τον «**μεγαλύτερο**» τότε δίνουμε την τιμή του στον «**μεγαλύτερο**».

Αν όχι τότε ελέγχουμε τον επόμενο έως ότου ελεγχθούν όλοι οι ακέραιοι.

## #2.9.2 Ψευδοκώδικας- διάγραμμα

Είσοδος: πλήθος Ακεραίων

Έξοδος: ο μεγαλύτερος ακέραιος

{

Μεγαλύτερος  $\leftarrow -\infty$

Για όσο(υπάρχει άλλος ακέραιος)

{

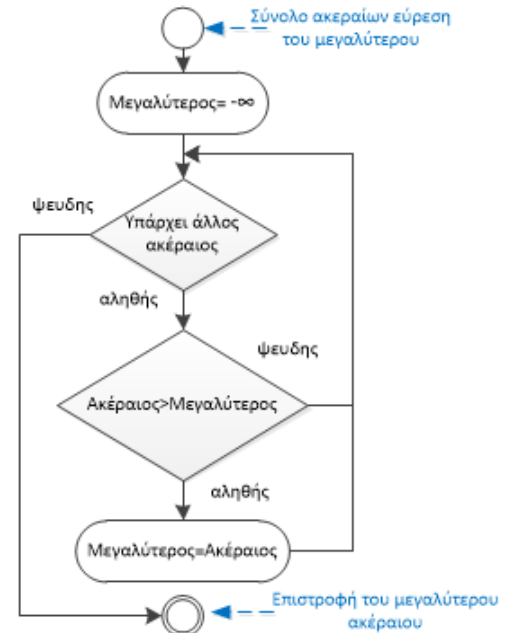
Αν(Ακέραιος > Μεγαλύτερος)

Μεγαλύτερος  $\leftarrow$  Ακέραιος

}

Επέστρεψε Μεγαλύτερος

}



### #2.10.1 Αλγόριθμος εύρεσης μικρότερου

Ο αλγόριθμος αυτός δέχεται στην είσοδο μια σειρά από ακέραιους και δίνει στην έξοδό του τον μικρότερο.

Αυτό γίνεται με την δημιουργία μιας μεταβλητής «**μικρότερος**» στην οποία δίνουμε την τιμή ενός πολύ μεγάλου ακέραιου.

Εάν ο επόμενος ακέραιος είναι μικρότερος από τον «μικρότερο» τότε δίνουμε την τιμή του στον «μικρότερο».

Αν όχι τότε ελέγχουμε τον επόμενο έως ότου ελεγχθούν όλοι οι ακέραιοι.

## #2.10.2 Ψευδοκώδικας- διάγραμμα

Είσοδος: πλήθος Ακεραίων

Έξοδος: ο μεγαλύτερος ακέραιος

{

Μικρότερος  $\leftarrow +\infty$

Για όσο(υπάρχει άλλος ακέραιος)

{

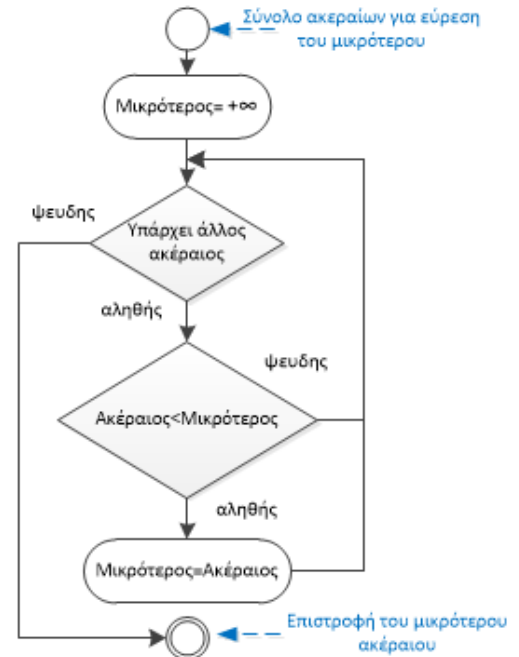
Αν(Ακέραιος < Μικρότερος)

Μικρότερος  $\leftarrow$  Ακέραιος

}

Επέστρεψε Μικρότερος

}

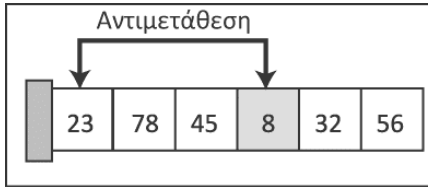


### #2.11.1 Ταξινόμηση με επιλογή

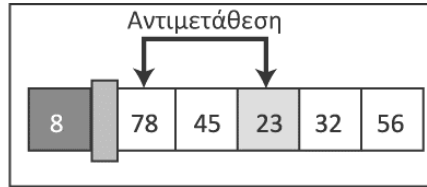
Εδώ το σύνολο των αριθμών χωρίζεται σε δύο υποσύνολα, ένα ταξινομημένο και ένα μη ταξινομημένο, τα οποία τα χωρίζει ένα νοητό τείχος. Στην αρχή βρίσκουμε το μικρότερο από τους ακεραίους που υπάρχουν στο μη ταξινομημένο υποσύνολο και το τοποθετούμε στην αρχή. Κατόπιν μετακινούμε το τείχος μια θέση δεξιά έτσι ώστε να χωρίσουμε τα δύο υποσύνολα. Ελέγχουμε για τον επόμενο μικρότερο του μη ταξινομημένου υποσυνόλου και αφού τον βρούμε τον τοποθετούμε στη πρώτη θέση του μη ταξινομημένου υποσυνόλου. Αυτές οι ενέργειες επαναλαμβάνονται έως ότου όλοι οι ακέραιοι ταξινομηθούν από τον μικρότερο στον μεγαλύτερο. Για ένα σύνολο  $n$  στοιχείων για να ολοκληρωθεί η αναζήτηση, κάνουμε  $n-1$  ελέγχους. Με τον ίδιο τρόπο δουλεύουμε και για την φθίνουσα ταξινόμηση.

## #2.11.2 Ταξινόμηση με επιλογή-παράδειγμα

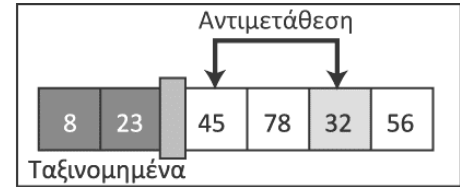
Παράδειγμα ταξινόμησης με επιλογή σε ένα σύνολο 5 ακεραίων.



Αρχική λίστα

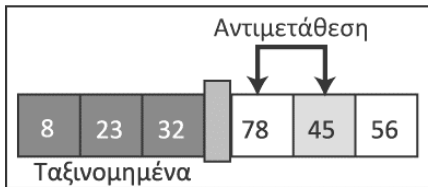


Μετά το πρώτο πέρασμα



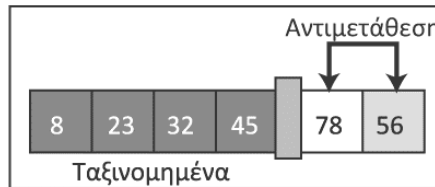
Ταξινομημένα

Μετά το δεύτερο πέρασμα



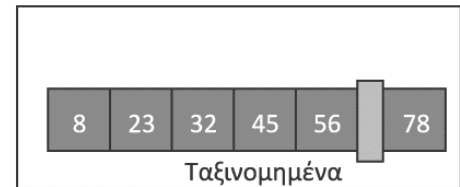
Ταξινομημένα

Μετά το τρίτο πέρασμα



Ταξινομημένα

Μετά το τέταρτο πέρασμα



Ταξινομημένα

Μετά το πέμπτο πέρασμα

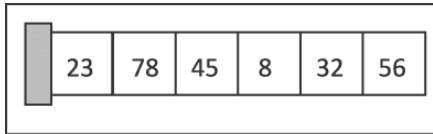


### #2.12.1 Ταξινόμηση φυσαλίδας

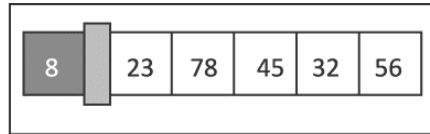
Εδώ το σύνολο των αριθμών χωρίζεται επίσης σε δύο υποσύνολα, ένα ταξινομημένο και ένα μη ταξινομημένο, τα οποία τα χωρίζει ένα νοητό τείχος. Το μικρότερο από τους ακεραίους που υπάρχουν στο μη ταξινομημένο υποσύνολο αναδύεται σαν φυσαλίδα και μετακινείται στο ταξινομημένο υποσύνολο. Κατόπιν μετακινούμε το τείχος μια θέση δεξιά έτσι ώστε να χωρίσουμε τα δύο υποσύνολα. Ελέγχουμε για τον επόμενο μικρότερο του μη ταξινομημένου υποσυνόλου και αφού τον βρούμε θα αναδυθεί και αυτός στο ταξινομημένο σύνολο. Εδώ η ανάδυση γίνεται με τη μέθοδο της επανάληψης. Για ένα σύνολο  $n$  στοιχείων για να ολοκληρωθεί η αναζήτηση, κάνουμε  $n-1$  ελέγχους. Με τον ίδιο τρόπο δουλεύουμε και για την φθίνουσα ταξινόμηση.

## #2.12.2 Ταξινόμηση φυσαλίδας-παράδειγμα

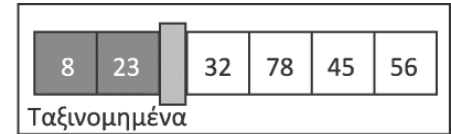
Παράδειγμα ταξινόμησης φυσαλίδας σε ένα σύνολο 5 ακεραίων.



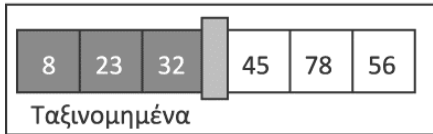
Αρχική λίστα



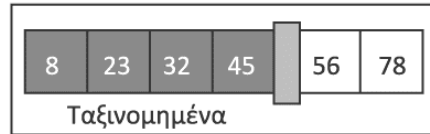
Μετά το πρώτο πέρασμα



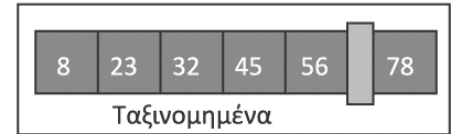
Μετά το δεύτερο πέρασμα



Μετά το τρίτο πέρασμα



Μετά το τέταρτο πέρασμα



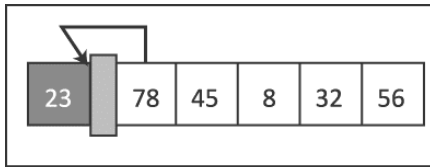
Μετά το πέμπτο πέρασμα

### #2.13.1 Ταξινόμηση με εισαγωγή

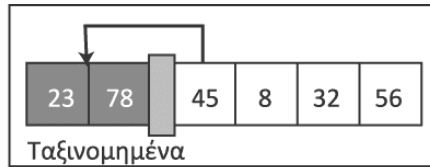
Εδώ το σύνολο των αριθμών χωρίζεται επίσης σε δύο υποσύνολα, ένα ταξινομημένο και ένα μη ταξινομημένο, τα οποία τα χωρίζει ένα νοητό τείχος. Πάντα το πρώτο στοιχείο του μη ταξινομημένου υποσυνόλου μεταφέρεται στην κατάλληλη θέση του ταξινομημένου υποσυνόλου. Χρησιμοποιούμε δύο βρόχους με εμφώλευση. Ο εξωτερικός βρόχος επιλέγει τους ακεραίους από το μη ταξινομημένο υποσύνολο και ο εσωτερικός τους τακτοποιεί στην σωστή θέση στο ταξινομημένο υποσύνολο. Για ένα σύνολο  $n$  στοιχείων για να ολοκληρωθεί η αναζήτηση, κάνουμε  $n-1$  ελέγχους. Με τον ίδιο τρόπο δουλεύουμε και για την φθίνουσα ταξινόμηση.

## #2.13.2 Ταξινόμηση με εισαγωγή-παράδειγμα

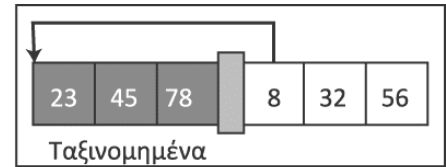
Παράδειγμα ταξινόμησης με εισαγωγή σε ένα σύνολο 5 ακεραίων.



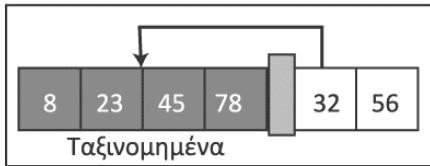
Αρχική λίστα



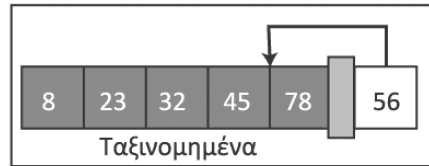
Μετά το πρώτο πέρασμα



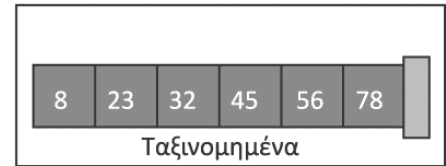
Μετά το δεύτερο πέρασμα



Μετά το τρίτο πέρασμα



Μετά το τέταρτο πέρασμα



Μετά το πέμπτο πέρασμα

### #2.14.1 Ακολουθιακή αναζήτηση

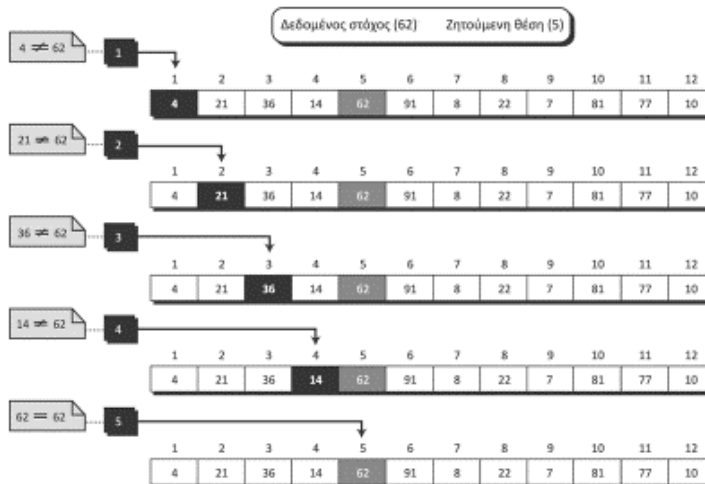
Σε αυτή την περίπτωση κάνουμε αναζήτηση ενός ακεραίου σε ένα μη ταξινομημένο σύνολο ακεραίων. Ξέροντας εξ αρχής την τιμή του ακεραίου ελέγχουμε από την αρχή του συνόλου έναν-έναν τους ακέραιους συγκρίνοντας την τιμή τους με τον γνωστό. Όταν βρεθεί ο ακέραιος είναι γνωστό σε μας πλέον και η θέση του από τα βήματα που έχουμε κάνει μέχρι να τον βρούμε.

Εννοείται ότι εάν ο ακέραιος δεν υπάρχει μπορεί να εξαντλήσουμε το σύνολο χωρίς αποτέλεσμα.

Για ένα σύνολο  $n$  ακεραίων θα χρειαστούμε  $n$  βήματα έως ότου τον εντοπίσουμε.

## #2.14.2 Ακολουθιακή αναζήτηση-παράδειγμα

Παράδειγμα ακολουθιακής αναζήτησης.

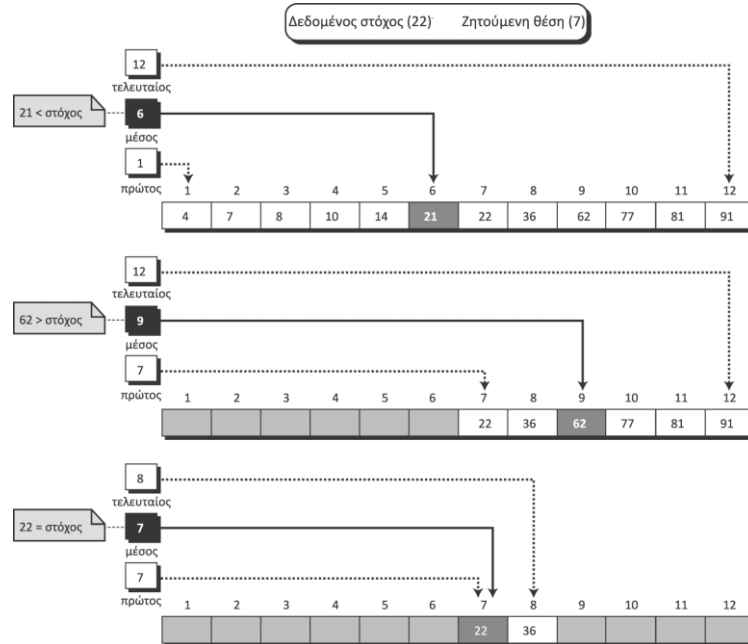


### #2.15.1 Δυαδική αναζήτηση

Σε αυτήν τη περίπτωση κάνουμε αναζήτηση σε ταξινομημένο σύνολο ακεραίων. Ελέγχουμε την τιμή του ακεραίου που βρίσκεται στη μέση του συνόλου. Εάν ο αριθμός βρίσκεται στο πρώτο μισό του συνόλου δεν ελέγχουμε το δεύτερο μισό και αντίστροφα. Κατόπιν ελέγχουμε το μισό που ενδέχεται να υπάρχει ο ακέραιος χωρίζοντας το εκ νέου στη μέση. Αυτό επαναλαμβάνεται έως ότου βρεθεί ο ακέραιος.

## #2.15.2 Δυναδική αναζήτηση-παράδειγμα

Παράδειγμα δυναδικής αναζήτησης.





## #2.16.1 Υποαλγόριθμοι

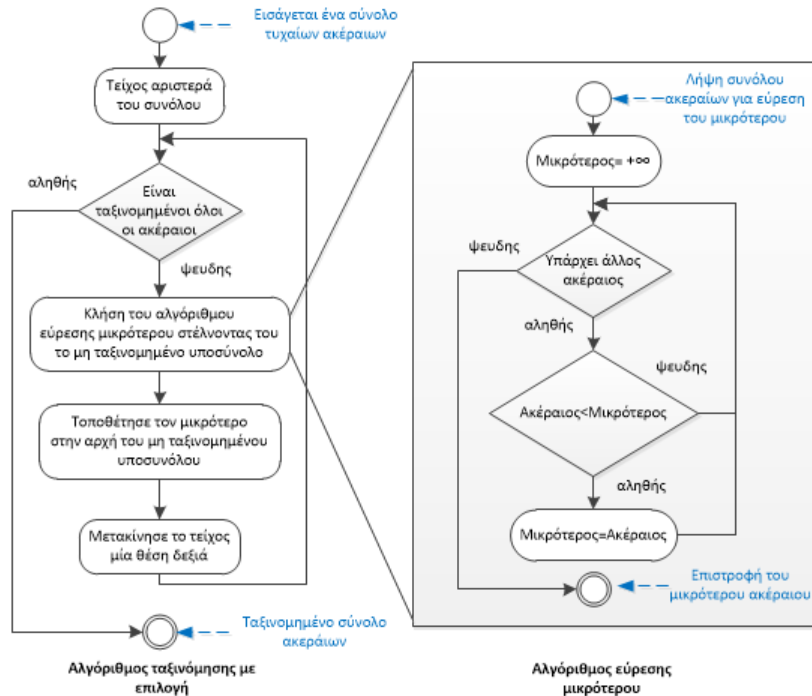
Για να λύσουμε οποιοδήποτε υπολογιστικό πρόβλημα χρησιμοποιούμε στον σχεδιασμό της γενικότερης λύσης τον αλγόριθμο.

Στον δομημένο προγραμματισμό όμως, έχει επικρατήσει η άποψη ότι για να λυθεί το πρόβλημα πρέπει να επιμεριστεί η γενικότερη λύση σε υπολύσεις τις οποίες υλοποιούν οι υπορουτίνες.

Οι υπορουτίνες σχεδιάζονται με τους υποαλγόριθμους.

Ένας υποαλγόριθμος υλοποιεί ένα από τα βήματα ενός αλγόριθμου και μπορούμε να τον καλέσουμε σε οποιοδήποτε βήμα του αλγόριθμου χωρίς να χρειαστεί να τον ξαναγράψουμε.

## #2.16.2 Παράδειγμα υποαλγόριθμου



## #2.17 Επανάληψη και αναδρομή

Για να λύσουμε οποιοδήποτε υπολογιστικό πρόβλημα χρησιμοποιούμε δύο διαφορετικές μεθόδους σύνταξης αλγορίθμων, την αναδρομή και την επανάληψη.

Ένας αλγόριθμος ονομάζεται **αναδρομικός** για την ολοκλήρωση της λύσης καλεί τον εαυτό του.

Ενώ **επαναληπτικός** όταν για την ολοκλήρωση της λύσης χρησιμοποιεί επαναλήψεις.

Ένα παράδειγμα χρήσης των δύο αυτών μεθόδων είναι ο υπολογισμός του παραγοντικού ενός ακεραίου αριθμού. (Σημ. Παραγοντικό ενός ακεραίου είναι το γινόμενο όλων των ακεραίων από το 1 έως και τον συγκεκριμένο ακεραίο.)

## #2.18 Παραγοντικό με επανάληψη και αναδρομή

Έστω ο ακέραιος αριθμός  $n$

Επαναληπτική μέθοδος

$$\text{Παραγοντικό}(n) \left[ \begin{array}{ll} \text{για } n=0 & n!=1 \\ \text{για } n>0 & n!=n*(n-1)*(n-2)*\dots*2*1 \end{array} \right]$$

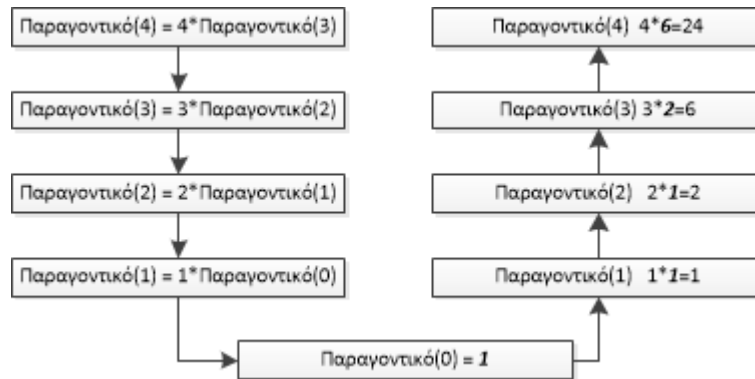
Αναδρομική μέθοδος

$$\text{Παραγοντικό}(n) \left[ \begin{array}{ll} \text{για } n=0 & n!=1 \\ \text{για } n>0 & n!=n*\text{παραγοντικό}(n-1) \end{array} \right]$$

## #2.18 Παράδειγμα με αναδρομή

Πιο κάτω φαίνεται η λειτουργία της αναδρομικής μεθόδου θεωρώντας ότι ο ακέραιος είναι ο αριθμός 4.

$$n! = n * \text{παραγοντικό}(n-1)$$



Δομημένος Προγραμματισμός

**#3**

**Η ΓΛΩΣΣΑ C**

### #3.1 Τι είναι η γλώσσα C

Η γλώσσα προγραμματισμού **C** είναι μια γλώσσα προγραμματισμού **υψηλού επιπέδου**.

Μπορεί δηλαδή να λειτουργήσει σε πολλές αρχιτεκτονικές υπολογιστών.

Με άλλα λόγια τα προγράμματα που κατασκευάζουμε με την γλώσσα προγραμματισμού C είναι συμβατά με γλώσσες μηχανής πολλών αρχιτεκτονικών υπολογιστών.

## #3.2 Τα χαρακτηριστικά της γλώσσας C

Η γλώσσα προγραμματισμού C :

- Χρησιμοποιείται και σαν γλώσσα χαμηλού επιπέδου,
- Υποστηρίζει διαδικαστικό (δομημένο) προγραμματισμό,
- Είναι εύκολη στην εκμάθηση,
- Παράγει γρήγορα (στην εκτέλεσή τους) προγράμματα,
- Είναι με την **C++** και την **Java** οι πιο διαδεδομένες και ευρύτερα χρησιμοποιούμενες γλώσσες.



### #3.3 Περισσότερα για τη γλώσσα C

- Η γλώσσα C υποστηρίζει όλες τις δομές αλγορίθμων, δηλαδή ακολουθιακή δομή, δομή επιλογής, αφαιρετική δομή με επιλογή περιπτώσεων χρήσης και δομή επανάληψης.
- Επίσης με την γλώσσα C μπορούμε να φτιάξουμε εκτός από μεταβλητές, απλούς άλλα και δυναμικούς πίνακες (arrays), συναρτήσεις (functions), δομές (structures).
- Η γλώσσα C δεν υποστηρίζει αντικειμενοστραφή προγραμματισμό.

### #3.4 Πρόγραμμα σε γλώσσα C

Πιο κάτω βλέπουμε ένα πρόγραμμα σε γλώσσα C, το οποίο θα τυπώσει στην οθόνη την πρόταση ***hello people***.

```
/* =====  
to proto mou programma se C  
===== */  
  
#include <stdio.h>  
void main()  
{  
    printf("hello people.\n");  
  
} //telos ths main
```

### #3.5.1 Ανάλυση παραδείγματος

Ξεκινώντας βλέπουμε ότι το πρόγραμμα αρχίζει ουσιαστικά με την συνάρτηση ***main()*** που είναι και η ***βασική συνάρτηση της γλώσσας C*** (και της C++).

Μέσα στα άγκιστρά ***{}*** αναπτύσσεται το ***κυρίως σώμα*** του προγράμματος που εδώ περιέχει μόνο την πρόταση:

```
printf("hello people.\n");
```

Η ***printf()*** είναι συνάρτηση μορφοποιούμενης εξόδου, που περιέχεται στην ***βιβλιοθήκη συναρτήσεων*** της γλώσσας C.

Πιο συγκεκριμένα ανήκει στο αρχείο επικεφαλίδας (ή κλάση) εισόδου-εξόδου ***stdio.h***

### #3.5.2 Ανάλυση παραδείγματος

Το αρχείο αυτό ονομάζεται **αρχείο επικεφαλίδας ή κεφαλίδας**, έχει κατάληξη **.h** και περιλαμβάνει συναρτήσεις που αφορούν τις συσκευές εισόδου-εξόδου.

Δηλώνεται ότι θα συμπεριληφθεί στον κώδικά που φτιάχνουμε, με την **προτροπή `#include`** που γράφεται πάντα στην αρχή του προγράμματος και έξω από την *main()*.

Η **`printf()`** είναι **συνάρτηση**, άρα έχει **ορίσματα** που περικλείονται μέσα στις παρενθέσεις της. Εδώ το όρισμα της *printf()* είναι η πρόταση *hello people* η οποία περικλείεται από **διπλά εισαγωγικά**.

Η *printf()* θα στείλει στην οθόνη την πιο πάνω πρόταση.

### #3.5.3 Ανάλυση παραδείγματος

Μέσα στην πρόταση βλέπουμε τον **χαρακτήρα διαφυγής \n** που λέει στο πρόγραμμα να αλλάξει γραμμή. Βλέπουμε επίσης ότι η συνάρτηση τελειώνει με το ελληνικό ερωτηματικό ; το οποίο ονομάζεται **τερματικό εντολής** και μπαίνει πάντα στο τέλος κάθε εντολής εκτός από το τέλος της *main* αλλά και τις προτροπές (*include*, *define*) στις οποίες μπαίνει στην αρχή τους η δέση #.

Το πρόγραμμα στην έξοδό του (όταν μεταγλωττιστεί και τρέξει) θα εμφανίσει στην οθόνη :

```
hello people
```

### #3.5.4 Ανάλυση παραδείγματος

Στο πρόγραμμα υπάρχουν επίσης και **σχόλια** τα οποία είναι χρήσιμα για την περιγραφή και κατανόηση των διαφόρων τμημάτων του κώδικά που φτιάχνουμε.

Τα σχόλια μπορούν να εκτείνονται σε μία ή και περισσότερες γραμμές.

Στο σχόλιο **μιας γραμμής** χρησιμοποιούμε τις δύο καθέτους στην αρχή του σχολίου.

```
// end of main
```

### #3.5.5 Ανάλυση παραδείγματος

Σε σχόλια **περισσοτέρων γραμμών** χρησιμοποιούμε στην αρχή του σχολίου τους χαρακτήρες `/*` και στο τέλος τους ίδιους χαρακτήρες με ανάποδη σειρά `*/`.

```
/* new source code  
for my PROGRAM */
```

Σημ. δεν ξεχνάμε ποτέ το κλείσιμο του σχολίου `*/` γιατί οτιδήποτε από εκεί και πέρα θεωρείται από την γλώσσα σχόλιο και δεν εκτελείται.

### #3.6.1 Αρχεία κεφαλίδας της γλώσσας C

Τα αρχεία κεφαλίδας περιέχουν έτοιμες συναρτήσεις που δίνουν λύσεις σε επιμέρους θέματα. Μερικά από αυτά με κάποιες από τις συναρτήσεις τους φαίνονται παρακάτω.

**stdio.h** : (C Standard Input and Output Library)

***scanf()*** Read formatted data, ***printf()*** Print formatted data,

***fopen()*** Open file, ***fclose()*** Close file,

***getc()*** Get character, ***putc()*** Write character.

**math.h** : (C Numerics Library)

***cos()*** Compute cosine, ***acos()*** Compute arc cosine,

***pow()*** Raise to power, ***sqrt()*** Compute square root,

***fmax()*** Maximum value, ***fmin()*** Minimum value.



## #3.6.2 Αρχεία κεφαλίδας της γλώσσας C

**stdlib.h** : (C Standard General Utilities Library)

**rand()** Generate random number, **calloc()** Allocate and zero-initialize array, **abort()** Abort current process.

**string.h** : (C Strings)

**strcpy()** Copy string, **strcmp()** Compare two strings, **strlen()** Get string length, **memchr()** Locate character in block of memory.

**time.h** : (C Time Library)

**time()** Get current time, **difftime()** Return difference between two times.

### #3.7.1 Κανόνες συντακτικού της γλώσσας C

Το συντακτικό της γλώσσας διέπεται από συγκεκριμένους κανόνες που μερικοί από αυτούς φαίνονται πιο κάτω:

1. Το πρόγραμμα αποτελείται από **προτάσεις** που εκτελούνται με την σειρά.
2. Κάθε εντολή τερματίζεται με το **ελληνικό ερωτηματικό** (ή αλλιώς τερματικό εντολής).
3. Χρησιμοποιούμε **μια γραμμή** ανά πρόταση.
4. Αφήνουμε **κενές γραμμές** στα διαφορετικά τμήματα του κώδικα.

### #3.7.2 Κανόνες συντακτικού της γλώσσας C

5. Η γλώσσα C κάνει διάκριση **πεζών** και **κεφαλαίων** γραμμάτων (είναι case-sensitive).
6. Οι **σταθερές** γράφονται με κεφαλαία.
7. Οι εντολές μιας **συνάρτησης** μπαίνουν πάντα μέσα σε **άγκιστρα**.
8. Αφήνουμε **κενά διαστήματα** για καλύτερη ανάγνωση του κώδικα.
9. Οι γραμμές δεν πρέπει να έχουν περισσότερους από 80 χαρακτήρες.

### #3.8.1 Λεξιλόγιο της γλώσσας C

Στην C υπάρχουν:

1. **Δεσμευμένες λέξεις** που δεν πρέπει να τις χρησιμοποιούμε για άλλο σκοπό, δηλαδή
  - Ονόματα συναρτήσεων βιβλιοθήκης (π.χ. ***printf, scanf***).
  - Μακροεντολές που περιέχονται σε αρχεία κεφαλίδας (π.χ. ***INT\_MAX, INT\_MIN***).
  - Ονόματα τύπων που περιέχονται σε αρχεία κεφαλίδας (π.χ. ***va\_list***).
  - Εντολές προς τον προεπεξεργαστή (π.χ. ***include, define***).

## #3.8.2 Λεξιλόγιο της γλώσσας C

2. **Λέξεις κλειδιά** που χρησιμοποιούνται από την γλώσσα.
  - Δηλαδή π.χ ο τύπος πραγματικών αριθμών ***double***,
  - ο έλεγχος ροής ***if-else***,
  - η επανάληψη ***for***
  - η εντολή διακοπής της ροής ***break*** και άλλες.
3. **Αναγνωριστές** που είναι ονόματα που δίνει ο προγραμματιστής στις μεταβλητές, τις συναρτήσεις κ.α.
4. **Τελεστές** που είναι σύμβολα που χρησιμοποιούνται για μαθηματικές ή λογικές πράξεις.

### #3.9.1 Μετατροπή αλγόριθμου σε κώδικα με τη γλώσσα C

Παρακάτω φαίνεται η μετατροπή ενός αλγόριθμου, σε κώδικα της γλώσσας C.

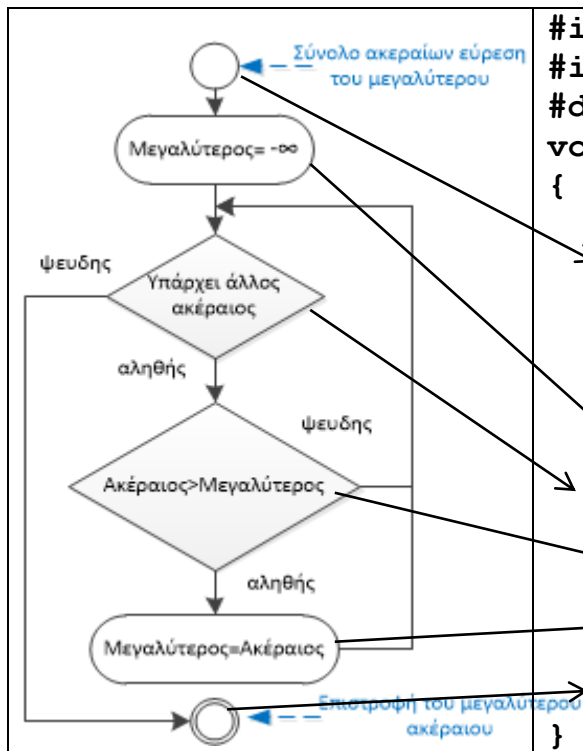
Πιο συγκεκριμένα πρόκειται για τον αλγόριθμο εύρεσης του μεγαλύτερου ακέραιου από ένα σύνολο 100 ακεραίων.

Ο κώδικας που γράφουμε περιέχει δήλωση μεταβλητών καθώς και μονοδιάστατου πίνακα ακεραίων.

Επίσης περιέχει μια δομή επανάληψης με έλεγχο στην είσοδο και μετρητή και μία δομή επιλογής.

## #3.9.2 Μετατροπή αλγόριθμου σε κώδικα με τη γλώσσα C

### Αλγόριθμος



### Κώδικας

```
#include <stdio.h> //Libr. Input-output
#include <limits.h> //Libr. oria akeraewn
#define R 100 //plh8os akeraewn
void main()
{
    int Ar[R], i, max; //dhlwsh metavlhtwn
    for(i=0; i<R; i++)
    { // eisagwgh akeraewn ston pinaka
        printf("\nSTOIXEIO[%d]:", i);
        scanf("%d", &Ar[i]);
    }
    max = INT_MIN; //anhkei sthn limits.h
    for(i=0; i<R; i++)
    {
        if(Ar[i] > max) //elegxos megalyterou
            max = Ar[i];
    }
    printf("\n max number is o:%d \n", max);
}
```

Δομημένος Προγραμματισμός

#4

**ΜΕΤΑΒΛΗΤΕΣ-ΤΥΠΟΙ**



### #4.1.1 Τι είναι μεταβλητή

**Μεταβλητή** είναι μία δεσμευμένη από εμάς θέση στην μνήμη στην οποία θα αποθηκευτούν δεδομένα εξ αρχής ή κατά την εκτέλεση του προγράμματος.

Η μεταβλητή έχει:

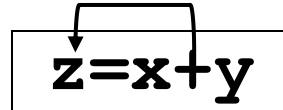
- Τύπο άρα και μέγεθος, που καθορίζεται από τον τύπο της,
- Όνομα και
- Τιμή.

Η μεταβλητή μπορεί να αλλάξει τιμή κατά την εκτέλεση του προγράμματος. Στην μεταβλητή αποθηκεύονται αριθμητικές αλλά και αλφαριθμητικές τιμές.

## #4.2 Πως χρησιμοποιείται

Οι μεταβλητές μαζί με τις σταθερές και τους τελεστές, χρησιμοποιούνται από τις γλώσσες προγραμματισμού μέσα σε **μαθηματικές εκφράσεις** για την εκτέλεση υπολογισμών.

Για παράδειγμα στην ακόλουθη έκφραση:


$$\mathbf{z = x + y}$$

στην μεταβλητή **z** θα **εκχωρηθεί** η τιμή που θα προκύψει από το άθροισμα των τιμών των μεταβλητών **x** και **y**.

Σημ. το σύμβολο **+** είναι ο **τελεστής** πρόσθεσης.

## #4.3 Δήλωση μεταβλητής

Η μεταβλητή δηλώνεται στον κώδικα του προγράμματος, με **πρόταση ορισμού** που τελειώνει πάντοτε με το ελληνικό ερωτηματικό ; που λέγεται και τερματικό εντολής.

Έστω η πρόταση:

τύπος όνομα_μεταβλητής;
-------------------------

Στην πιο πάνω πρόταση δηλώνουμε τον **τύπο** της μεταβλητής και κατά συνέπεια τον **χώρο** που θα καταλάβει στην μνήμη καθώς και το **όνομά** της.

Σημ. Δεν έχουμε περάσει ακόμα **τιμή** στην μεταβλητή.

### #4.4.1 Παράδειγμα δήλωσης

Στην πρόταση ορισμού που ακολουθεί:

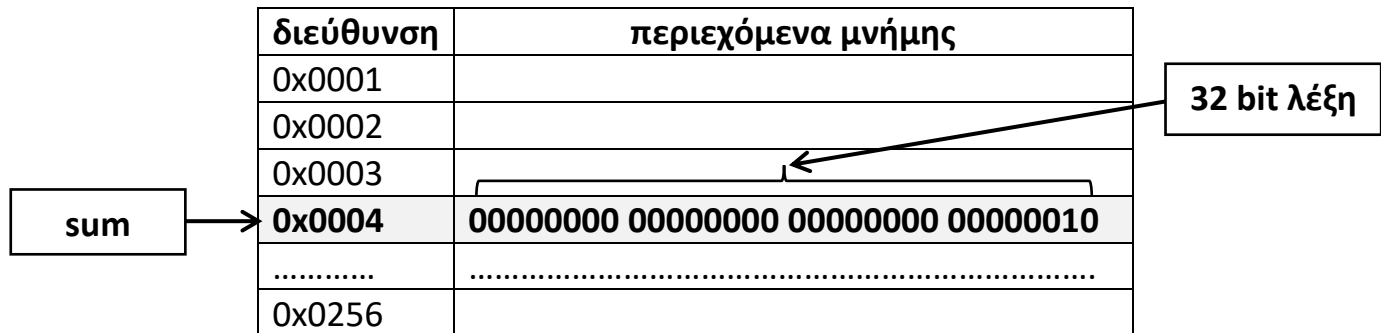
```
int sum = 2;
```

λέμε στον **compiler** να δεσμεύσει στην μνήμη χώρο για μια μεταβλητή **ακεραίου** τύπου (*int*) που το όνομά της είναι **sum** και η τιμή της είναι 2.

Οι μεταβλητές δηλώνονται πάντα στην αρχή της `main` και κάθε συνάρτησης (μετά το αριστερό άγκιστρο) και το όνομά τους είναι πάντα συναφές με το είδος της πραγματικής τιμής που καλούνται να λάβουν. Άρα εδώ η *sum* θα πάρει μία τιμή αθροίσματος.

## #4.4.2 Παράδειγμα δήλωσης

Η παραπάνω δήλωση μεταβλητής στην μνήμη ενός συστήματος **32bit αρχιτεκτονικής** φαίνεται στο παρακάτω σχήμα.



Που σημαίνει ότι σε μια συγκεκριμένη διεύθυνση της μνήμης (`0x0004`) που αντιστοιχεί στο όνομα της μεταβλητής **`sum`**, ο compiler αποθηκεύει την τιμή της μεταβλητής (σε 32 bits) στο δυαδικό αριθμητικό σύστημα.

### #4.5.1 Όνομα μεταβλητής

Για να ονομάσουμε τις μεταβλητές μας χρησιμοποιούμε :

- Τα **γράμματα** του αγγλικού αλφαβήτου, κεφαλαία και μικρά,
- Τους **αριθμούς** 0 έως και 9,
- Τον χαρακτήρα **υπογράμμισης** ( \_ ).

Σημ. Η γλώσσα C κάνει διάκριση πεζών και κεφαλαίων γραμμάτων (case sensitive).

Αυτό σημαίνει ότι με όποια ακολουθία χαρακτήρων δηλώσουμε μία μεταβλητή, με την ίδια ακολουθία θα την καλέσουμε οπουδήποτε μέσα στο πρόγραμμα.

## #4.5.2 Όνομα μεταβλητής

Για να ονομάσουμε τις μεταβλητές προσέχουμε τα εξής:

- Το όνομα να ξεκινά με *γράμμα* ή με *χαρακτήρα υπογράμμισης* αν ο επόμενος χαρακτήρας είναι γράμμα. Π.χ. *Temp* ή *\_temp*. Ποτέ με οποιοδήποτε άλλο χαρακτήρα ή αριθμό. Π.χ. *@Temp* ή *7temp*.
- Δεν χρησιμοποιούμε για όνομα μεταβλητής *δεσμευμένες λέξεις* από την γλώσσα.
- Το μήκος του ονόματος δεν πρέπει να ξεπερνά τους *31 χαρακτήρες*.
- Το όνομα των μεταβλητών είναι πάντα *συναφές* με το είδος της πραγματικής τιμής που καλούνται να λάβουν.

### #4.5.3 Όνομα μεταβλητής

Παραδείγματα δήλωσης μεταβλητών.

		Χαρακτήρας
λάθος	<b>char #var</b>	
σωστό	<b>char var ή char _var</b>	
		Αριθμός
λάθος	<b>float 5temp</b>	
σωστό	<b>float _5temp ή float temp5</b>	
		Δεσμευμένη λέξη
λάθος	<b>int main</b>	
σωστό	<b>int Main</b>	
		Κενό διάστημα
λάθος	<b>float max temp</b>	
σωστό	<b>float maxTemp ή float max_temp</b>	



## #4.6.1 Τύπος μεταβλητής

**Τύπος** μεταβλητής είναι αυτός ο οποίος καθορίζει, το τι είδους τιμές θα μπορέσουμε να καταχωρήσουμε στην μεταβλητή αλλά και το χώρο σε bytes που θα καταλάβει η μεταβλητή στην μνήμη του συστήματος.

Π.χ. Σε ένα σύστημα 32bit δεδομένων (στο δυαδικό αριθμητικό σύστημα), η μεταβλητή τύπου int θα καταλάβει στη μνήμη χώρο 4bytes (1byte = 8bits ή 4bytes x 8bits = 32bits) ή αλλιώς  $2^{32} = 4,294,967,295$ . Οπότε μπορεί να πάρει μία από τις τιμές

-2,147,483,648 έως 2,147,483,647 (στις τιμές συμπεριλαμβάνεται και το 0).

## #4.6.2 Τύπος μεταβλητής

Τύπος	16 bit	32 bit	Εύρος
unsigned short int	2 bytes	2 bytes	0 έως 65,535
short int	2 bytes	2 bytes	-32,768 έως 32,767
unsigned int	2 bytes	4 bytes	(16 bit): 0 έως 65,535 (32 bit): 0 έως 4,294,967,295
int	2 bytes	4 bytes	(16 bit): -32,768 έως 32,767 (32 bit): -2,147,483,648 έως 2,147,483,647
unsigned long int	4 bytes	4 bytes	0 έως 4,294,967,295
long int	4 bytes	4 bytes	-2,147,483,648 έως 2,147,483,647
char	1 byte	1 byte	256 χαρακτήρες
bool	1 byte	1 byte	αληθές ή ψευδές
float	4 bytes	4 bytes	1.2e-38 έως 3.4e+38
double	8 bytes	8 bytes	2.2e-308 έως 1.8e+308
long double	10 bytes	10 bytes	3.4e-4932 έως 1.1e+4932

## #4.7.1 Κατηγορίες τύπων

Βασικοί τύποι:

- **int** είναι ο τύπος ακεραίου αριθμού,
- **float** είναι ο τύπος πραγματικού αριθμού κινητής υποδιαστολής χαμηλής ακρίβειας ( $1.2e-38$  έως  $3.4e+38$ ),
- **double** είναι ο τύπος πραγματικού αριθμού κινητής υποδιαστολής υψηλής ακρίβειας ( $2.2e-308$  έως  $1.8e+308$ ),
- **char** είναι ο τύπος χαρακτήρα,

Απαριθμητικός τύπος:

- **enum** είναι ο αριθμητικός τύπος που παίρνει διακριτές τιμές ακεραίων σε όλο το πρόγραμμα.

## #4.7.2 Κατηγορίες τύπων

Κενός τύπος:

- **void** ο προσδιοριστής void δείχνει ότι δεν υπάρχει διαθέσιμη τιμή. Συνήθως χρησιμοποιείται από συναρτήσεις.

Παραγόμενοι τύποι

- **pointer** είναι ο δείκτης που δείχνει τη διεύθυνση μεταβλητής,
- **arrays** είναι οι πίνακες τιμών ίδιου πεδίου ορισμού,
- **struct** δομές είναι οι εγγραφές χαρακτηριστικών,
- **union** ενώσεις είναι ειδικές κατασκευές που περιέχουν μεταβλητές διαφόρων τύπων,
- **function** συναρτήσεις, ένα σύνολο εντολών που εκτελούνται μαζί.

## #4.8 Προσδιοριστές και τελεστής διεύθυνσης &

Στη γλώσσα C για να εισάγουμε από το πληκτρολόγιο αριθμητική τιμή ή χαρακτήρα χρησιμοποιούμε την συνάρτηση ***scanf()***, τους προσδιοριστές **%c** (χαρακτήρα), **%d** (ακεραίου), **%f** (πραγματικού) και τον τελεστή διεύθυνσης **&**.

Εδώ ο τελεστής διεύθυνσης **&** αναφέρεται στην διεύθυνση της μεταβλητής που ακολουθεί και τοποθετεί την τιμή που του δίνει ο εκάστοτε προσδιοριστής σε αυτήν.

Για να στείλουμε την τιμή στην οθόνη χρησιμοποιούμε την συνάρτηση ***printf()*** και τους προσδιοριστές **%c**, **%d**, **%f**, **%o** (οκταδικής μορφής), **%x** (δεκαεξαδικής μορφής), **%e** (εκθετικής μορφής), **%g** (επιλογή μορφής από το σύστημα).

### #4.9.1 Τύπος χαρακτήρα – char

Η μεταβλητή που δηλώνεται ως τύπος **char** δέχεται όλους τους ASCII χαρακτήρες, πάντα μέσα σε μονά εισαγωγικά (π.χ. 'A', 'c', '\$' κ.α.) και δηλώνεται με την πιο κάτω πρόταση:

```
char όνομα_μεταβλητής;
```

Μπορεί να πάρει εξ αρχής τιμή π.χ.:

```
char first_char='G' ;
```

Σημ. Εδώ η μεταβλητή **first\_char** παίρνει κατά τη δήλωσή της την τιμή **G**.

## #4.9.2 Ο Τύπος χαρακτήρα – char

Μπορεί να πάρει τιμή στην πορεία, με την συνάρτηση *scanf()* τον προσδιοριστή χαρακτήρα **%c** και τον τελεστή διεύθυνσης **&** όπως πιο κάτω:

```
scanf("%c", &first_char);
```

ο χαρακτήρας μπορεί να τυπωθεί στην οθόνη με την συνάρτηση *printf()* και τον προσδιοριστή χαρακτήρα **%c** όπως πιο κάτω:

```
printf("ο xarakthras einai o %c\n", first_char);
```

εάν στην συνάρτηση *scanf()* δώσουμε τον χαρακτήρα **G** τότε η έξοδος της πιο πάνω πρότασης θα είναι:

```
ο xarakthras einai ο G
```

## #4.10 Χαρακτήρες διαφυγής

Στο προηγούμενο παράδειγμα είδαμε μέσα στην συνάρτηση *printf()* τον χαρακτήρα **\n** που τελικά δεν τυπώθηκε στην έξοδο.

Αυτός:

- ανήκει στις **ακολουθίες διαφυγής**,
- δηλώνει **νέα γραμμή**,
- μπαίνει πάντα μέσα στα **διπλά εισαγωγικά** και
- δεν εκτυπώνεται.

Οι χαρακτήρες διαφυγής ονομάζονται και **σταθερές**.



## #4.11 Πίνακας με τους χαρακτήρες διαφυγής

Χαρακτήρας	Ακολουθία
Συναγερμός (κουδούνι)	\a
Οπισθοχώρηση	\b
Αλλαγή σελίδας	\f
Νέα γραμμή	\n
Επαναφορά κεφαλής	\r
Οριζόντιος στηλοθέτης	\t
Κατακόρυφος στηλοθέτης	\v
Πλάγια γραμμή	\\
Λατινικό ερωτηματικό	\?
Μονό εισαγωγικό	\'
Διπλό εισαγωγικό	\"
Οκταδικός αριθμός	\ooo
Δεκαεξαδικός αριθμός	\xhhh

### #4.12.1 Ο Τύπος ακεραίου – int

Με τον τύπο ακεραίου **int** δηλώνουμε τις μεταβλητές που θα δεχθούν **ακέραιοι** θετικούς και αρνητικούς αριθμούς.

Δηλώνεται με την πιο κάτω πρόταση:

```
int όνομα_μεταβλητής;
```

Μπορεί να πάρει εξ αρχής τιμή όπως πιο κάτω:

```
int sum=58;
```

Η στην πορεία με την συνάρτηση *scanf()* τον προσδιοριστή δεκαδικού **%d** και τον τελεστή διεύθυνσης **&** όπως πιο κάτω:

```
scanf ("%d", &num);
```

## #4.12.2 Ο Τύπος ακεραίου – int

Η τιμή της μεταβλητής ακεραίου μπορεί να τυπωθεί στην οθόνη με την συνάρτηση *printf()* και τον προσδιοριστή **%d** για δεκαδική μορφή, **%o** για οκταδική μορφή και **%x** για δεκαεξαδική μορφή:

```
printf("ο αριθμος είναι %d,%o,%x\n", num, num, num);
```

Η έξοδος θα είναι:

```
ο αριθμος είναι ο 58,72,3a
```



### #4.13.1 Ο τύπος πραγματικού αριθμού – float

Με τον τύπο πραγματικού αριθμού float δηλώνουμε τις μεταβλητές που θα δεχθούν **πραγματικούς** αριθμούς κινητής υποδιαστολής μικρής ακρίβειας. Δηλώνεται με την πιο κάτω πρόταση:

```
float όνομα_μεταβλητής;
```

Μπορεί να πάρει εξ αρχής τιμή όπως πιο κάτω:

```
float temp=5.8;
```

Η στην πορεία με την συνάρτηση *scanf()* τον προσδιοριστή **%f** και τον τελεστή διεύθυνσης **&** όπως πιο κάτω:

```
scanf ("%f", &temp);
```

## #4.13.2 Ο τύπος πραγματικού αριθμού – float

Η τιμή της μεταβλητής πραγματικού αριθμού μπορεί να τυπωθεί στην οθόνη με την συνάρτηση *printf()* και τον προσδιοριστή **%f** για δεκαδική μορφή, **%e** για εκθετική μορφή και **%g** για επιλογή μορφής από το σύστημα (την μικρότερη).

```
printf("ο αριθμος είναι %f, %e, %g\n", temp, temp, temp);
```

Η έξοδος θα είναι:

```
ο αριθμος είναι 5.800000, 5.800000e, 5.8
```



## #4.14 Οι σταθερές

Στον προγραμματισμό εκτός από τις μεταβλητές χρησιμοποιούμε και τις σταθερές τις οποίες συναντάμε σε τρία είδη.

- Τις **κυριολεκτικές** σταθερές. Π.χ. `x = 3;`  
*Η σταθερά είναι η τιμή 3*
- Τις σταθερές με **αντικατάσταση**. Π.χ. `#define M 10`  
*Αντικαθιστά στο πρόγραμμα το M με την τιμή 10*
- Τις σταθερές με την χρήση της εντολής **const**.  
Π.χ. `const int x = 7;`

*Εκχωρεί στην μεταβλητή x την τιμή 7 και δεν της επιτρέπει να αλλάξει τιμή σε όλο το πρόγραμμα.*

Δομημένος Προγραμματισμός

#5

**ΤΕΛΕΣΤΕΣ-ΕΚΦΡΑΣΕΙΣ**

## #5.1 Τι είναι ο τελεστής

**Τελεστής** (*operator*) είναι ένα σύμβολο το οποίο παριστάνει μία πράξη, που εκτελείται σε ένα ή περισσότερα δεδομένα.

Τα δεδομένα ονομάζονται **τελεστέοι** (*operands*) και μπορεί να είναι μεταβλητές, σταθερές, ή και συνθήκες.

Οι τελεστές με τους τελεστέους σχηματίζουν εκφράσεις, οι οποίες βρίσκονται σε προτάσεις που τερματίζονται πάντα με το **τερματικό εντολής (;)**.



## #5.2 Κατηγορίες τελεστών

- Ανάλογα με τον αριθμό τελεστέων στους οποίους δρουν:
  1. **Μοναδιαίοι** (*unary*) που επιδρούν σε έναν τελεστέο,
  2. **Δυαδικοί** (*binary*) που επιδρούν σε δύο τελεστέους (βρίσκονται ανάμεσα),
  3. **Τριαδικοί** (*ternary*) που επιδρούν σε τρεις τελεστέους.
- Ανάλογα με την διεργασία που εκτελούν:
  1. **Αριθμητικοί** οι οποίοι εκτελούν αριθμητικές πράξεις,
  2. **Λογικοί** οι οποίοι εκτελούν λογικές πράξεις,
  3. **Συσχετιστικοί** οι οποίοι εκτελούν συγκρίσεις τιμών,
  4. **Διαχείρισης bits** οι οποίοι διαχειρίζονται bit,
  5. **Διαχείρισης μνήμης** οι οποίοι διαχειρίζονται τη μνήμη.

### #5.3.1 Παραδείγματα ανάλογα με τον αριθμό τελεστών

#### Μοναδιαίος τελεστής:

`&a`

είναι ο τελεστής επιστροφής διεύθυνσης `&` που εφαρμόζεται πάνω στην μεταβλητή `a`.

`i++`

είναι ο τελεστής αύξησης κατά ένα `++` που εφαρμόζεται πάνω στην μεταβλητή `i`.

`!w`

είναι ο λογικός τελεστής του αντίθετου NOT `!` που εφαρμόζεται πάνω στην μεταβλητή `w`.

## #5.3.2 Παραδείγματα ανάλογα με τον αριθμό τελεστών

Δυαδικός τελεστής:

$$a + b$$

είναι ο τελεστής της πρόσθεσης  $+$  που εφαρμόζεται πάνω στις μεταβλητές  $a$  και  $b$ .

$$(x \geq y)$$

είναι ο συγκριτικός τελεστής μεγαλύτερο ή ίσο  $\geq$  που εφαρμόζεται πάνω στις μεταβλητές  $x$  και  $y$ .

### #5.3.3 Παραδείγματα ανάλογα με τον αριθμό τελεστών

Τριαδικός τελεστής:

(συνθήκη)?a:b

είναι η *υπό συνθήκη απόφαση* ? : όπου σύμφωνα με την αλήθεια της συνθήκης θα εκτελεστεί η πρόταση a ή η πρόταση b.

Σημ. Σε ορισμένες περιπτώσεις αντικαθιστά μια δομή απόφασης if/else.

## #5.4 Τελεστές ανάλογα με την διεργασία που εκτελούν

Οι τελεστές ανάλογα με την διεργασία που εκτελούν χωρίζονται σε:

**Αριθμητικούς:** πρόσθεσης  $+$ , αφαίρεσης  $-$ , πολλαπλασιασμού  $*$ , πηλίκου ακεραίων ή πραγματικών  $/$ , υπόλοιπο ακεραίων  $\%$ .

**Λογικούς:** λογικό ΚΑΙ (and)  $\&\&$ , λογικό Ή (or)  $||$ , λογικό ΟΧΙ (not)  $!$

**Συσχετιστικούς** ή συγκριτικούς: μεγαλύτερο  $>$ , μικρότερο  $<$ , ίσο  $==$ , διάφορο  $!=$ , μεγαλύτερο ή ίσο  $>=$ , μικρότερο ή ίσο  $<=$ .

**Διαχείρισης bits:** ολίσθηση προς τα δεξιά  $>>$  ή αριστερά  $<<$ , bit and  $\&$ , bit or  $|$ , bit not  $!$ , bit exclusive or  $\wedge$ .

**Διαχείρισης μνήμης:** δείκτης  $\rightarrow$ , επιστροφή διεύθυνσης  $\&$ .

## #5.5 Εκφράσεις

Οι **εκφράσεις** είναι ένας συνδυασμός τελεστών και τελεστέων που δημιουργούν προτάσεις και εξάγουν ένα αποτέλεσμα που έχει την μορφή αριθμητικής ή δυαδικής τιμής.

Οι εκφράσεις χωρίζονται σε:

- σταθερές εκφράσεις (μόνο σταθερές τιμές),
- ακέрайου τύπου ή τύπου κινητής υποδιαστολής ή μικτές,
- εκφράσεις δείκτη (τιμές διεύθυνσης,) δηλαδή με: δείκτες, τελεστή διεύθυνσης &, αλφαριθμητικά και πίνακες .

## #5.6 Υλοποίηση έκφρασης

Για κάθε έκφραση υπάρχει μια υπολογιζόμενη αριθμητική ή δυαδική τιμή. Η τιμή αυτή υπολογίζεται σύμφωνα με την **προτεραιότητα** των τελεστών.

Όταν έχουμε σε μία έκφραση τελεστές ίδιας προτεραιότητας, τότε επιβάλλεται η κατεύθυνση εφαρμογής τελεστών ή αλλιώς **προσεταιριστικότητα**. Συνήθως η κατεύθυνση είναι από **αριστερά προς τα δεξιά**. Δηλαδή στην έκφραση  $\alpha + 3 + z / \beta * \gamma$ , θα εκτελεστεί πρώτα ο πολλαπλασιασμός  $\beta * \gamma$ , μετά το πηλίκο ακεραίων  $z / \beta * \gamma$ , μετά οι προσθέσεις (προτεραιότητα) και από τις προσθέσεις, πρώτα αυτή που βρίσκεται αριστερά  $\alpha + 3$  και μετά αυτή που βρίσκεται δεξιά  $\alpha + 3 + z / \beta * \gamma$  (προσεταιριστικότητα).

## #5.7 Παρένθεση και φωλιασμένη έκφραση

Ο πρώτος σε προτεραιότητα αριθμητικός τελεστής είναι η παρένθεση. Κατά τον υπολογισμό μιας έκφρασης υπολογίζονται **πρώτα** οι εκφράσεις που βρίσκονται μέσα σε **παρενθέσεις**.

Δηλαδή άλλο αποτέλεσμα έχει η έκφραση:

$(5*6)-2=28$  και άλλο η:  $5*(6-2)=20$ .

**Φωλιασμένες** εκφράσεις είναι οι εκφράσεις που βρίσκονται μέσα σε μια άλλη έκφραση. Στις φωλιασμένες εκφράσεις προτεραιότητα έχει η εσωτερική παρένθεση. Πχ. Στην έκφραση  **$((a\%2)\leq 3)\&\&p$**  η σειρά εκτέλεσης είναι πρώτα η  $a\%2$ , μετά η  $(a\%2)\leq 3$  και τέλος η  $((a\%2)\leq 3)\&\&p$ .



## #5.8.1 Προτεραιότητα-Προσεταιριστικότητα

Σειρά	Όνομα	Τελεστής	Σειρά υπολογισμού
1	Ανάλυση εμβέλειας	::	Αριστερά στα Δεξιά
2	Επιλογή μέλους, δείκτης, κλήση συναρτήσεων, αύξηση και μείωση κατάληξης	. -> () ++ --	Αριστερά στα Δεξιά
3	sizeof, αύξηση και μείωση προθέματος, συμπληρωματικό, AND, NOT, μοναδιαίο μείον και συν, διεύθυνση και εμφάνιση διεύθυνσης, new, new[], delete, delete[], μετατροπή, sizeof()	++ -- ^ ! - + & * ()	Δεξιά στα Αριστερά
4	Επιλογή μέλους για δείκτη	. * -> *	Αριστερά στα Δεξιά
5	Πολλαπλασιασμός, διαίρεση, υπόλοιπο	* / %	Αριστερά στα Δεξιά
6	Πρόσθεση, αφαίρεση	+ -	Αριστερά στα Δεξιά
7	Μετατόπιση	<< >>	Αριστερά στα Δεξιά
8	Σχετική ανισότητα	<<= >>=	Αριστερά στα Δεξιά

## #5.8.2 Προτεραιότητα-Προσεταιριστικότητα

9	Ισότητα, ανισότητα	== !=	Αριστερά στα Δεξιά
10	AND σε επίπεδο bit	&	Αριστερά στα Δεξιά
11	Αποκλειστικό OR σε επίπεδο bit	^	Αριστερά στα Δεξιά
12	OR σε επίπεδο bit		Αριστερά στα Δεξιά
13	Λογικό AND	&&	Αριστερά στα Δεξιά
14	Λογικό OR		Αριστερά στα Δεξιά
15	Υπό συνθήκη	?:	Δεξιά στα Αριστερά
16	Τελεστές δήλωσης	= *= /= %= += .= <<= >>= &=  = ^=	Δεξιά στα Αριστερά
17	Τελεστής throw	throw	
18	κόμμα	,	Αριστερά στα Δεξιά

## #5.9 Τελεστής ανάθεσης τιμής

Ο **τελεστής ανάθεσης τιμής (=)** είναι ο πιο σημαντικός από τους τελεστές.

Παίρνει την τιμή που βρίσκεται στον δεξιό τελεστέο (ή την τιμή που προκύπτει από την έκφραση που βρίσκεται δεξιά του) και την αναθέτει στον αριστερό (ή τους αριστερούς) τελεστέο. Π.χ.

**a=10** η σταθερά 10 περνά ως τιμή στην μεταβλητή a.

**z=x+y** η τιμή που προκύπτει από το άθροισμα των μεταβλητών x και y, περνά ως τιμή στη μεταβλητή z.

**a=b=c=100** η σταθερά 100 περνά ως τιμή κατά σειρά στις μεταβλητές a, b, c.

## #5.10 Σύνθετος τελεστής ανάθεσης τιμής

Στην γλώσσα C για συντομία χρησιμοποιούμε τους σύνθετους τελεστές ανάθεσης τιμής. Π.χ.

Η έκφραση  $x=x+2$  μπορεί να γραφτεί  $x+=2$ . Εδώ χρησιμοποιούμε τον σύνθετο τελεστή πρόσθεσης  $+=$ .

Η έκφραση  $k=k*z$  γράφεται  $k*=z$ . Εδώ χρησιμοποιούμε τον σύνθετο τελεστή πολλαπλασιασμού  $*=$ .

Προσοχή στις εκφράσεις του τύπου:  $y*=x-3$

Εδώ η πράξη είναι η  $y=y*(x-3)$  και όχι η  $y=y*x-3$ . Αν θέσουμε τις τιμές,  $x=5$  και  $y=8$  τότε η στην πρώτη  $y=16$  ενώ στην δεύτερη  $y=37$ .

## #5.11 Τελεστής αύξησης – μείωσης κατά 1

Στην έκφραση  $x=x+1$ , η νέα τιμή της μεταβλητής  $x$  είναι αυτή που είχε, αυξημένη κατά 1.

Αυτό στη γλώσσα C το δείχνουμε με τον τελεστή `x++` που είναι ο τελεστής αύξησης κατά 1.

Το ίδιο ισχύει και για την μείωση κατά ένα όπου την έκφραση  $x=x-1$  την αντικαθιστούμε με τον τελεστή `x--` που είναι ο τελεστής μείωσης κατά 1.

Τους τελεστές αυτούς συχνά τους χρησιμοποιούμε ως μετρητές (counters) οι οποίοι καταμετρούν γεγονότα. Π.χ. το πόσες φορές άνοιξε μια πόρτα θα τις καταγράψει ο μετρητής `door++`.

### #5.12.1 Προπορευόμενος – παρελκόμενος τελεστής

Οι τελεστές αύξησης και μείωσης κατά ένα συναντώνται σε δύο μορφές.

Τον τελεστή που βρίσκεται πριν την μεταβλητή (αριστερά της) και οποίος ονομάζεται **προπορευόμενος (++a)** και αυτόν που βρίσκεται μετά την μεταβλητή (δεξιά της) και ονομάζεται **παρελκόμενος (a++)**.

Όταν συναντώνται μόνοι τους σε οποιοδήποτε σημείο του κώδικα λειτουργούν σαν τελεστές αύξησης ή μείωσης.

Όταν βρίσκονται όμως μέσα σε μια έκφραση λειτουργούν διαφορετικά.

## #5.12.1 Προπορευόμενος – παρελκόμενος τελεστής

### Παραδείγματα

Στην έκφραση  **$a = ++b + c$**  ο τελεστής αύξησης  **$++b$**  είναι προπορευόμενος, αυτό σημαίνει ότι **πρώτα αλλάζει την τιμή του** και κατόπιν χρησιμοποιείται η νέα του τιμή στην έκφραση. Αν δηλαδή οι μεταβλητές έχουν τις τιμές  $b=10$ ,  $c=20$  τότε πρώτα θα γίνει η  $b=11$  και μετά θα γίνει η πράξη της πρόσθεσης με την  $c$ . Το τελικό αποτέλεσμα της έκφρασης θα είναι  $a=31$ .

Στην έκφραση  **$a = b++ + c$**  ο τελεστής αύξησης  **$b++$**  είναι παρελκόμενος, δηλαδή **πρώτα θα χρησιμοποιηθεί η τιμή του στην πρόσθεση** και κατόπιν θα αλλάξει τιμή. Οπότε με τιμές  $b=10$ ,  $c=20$  πρώτα  $a=30$  ( $10+20$ ) και μετά  $b=11$ .

## #5.13 Δυαδικοί τελεστές

Δυαδικός τελεστής	Σύμβολο	Δυαδικός τελεστής	Σύμβολο
Μικρότερο	<	Πρόσθεση	+
Μικρότερο ή ίσο	<=	Αφαίρεση	-
Ίσο	=	Πολλαπλασιασμός	*
Διάφορο	!=	Διαίρεση πραγματικών	/
Μεγαλύτερο	>	Πηλίκο διαίρεσης ακεραίων	/
Μεγαλύτερο ή ίσο	>=	Υπόλοιπο διαίρεσης ακεραίων	%



## #5.14 Αριθμητικοί τελεστές

Τελεστής	Σύμβολο
Πρόσθεση	+
Αφαίρεση	-
Πολλαπλασιασμός	*
Διαίρεση πραγματικών	/
Πηλίκο διαίρεσης ακεραίων	/
Υπόλοιπο διαίρεσης ακεραίων	%

## #5.15 Συσχετιστικοί τελεστές

Τελεστής	Σύμβολο
Μικρότερο	<
Μικρότερο ή ίσο	<=
Ίσο	=
Διάφορο	!=
Μεγαλύτερο	>
Μεγαλύτερο ή ίσο	>=

## #5.16 Λογικοί τελεστές

Τελεστής	Δράση
<b>&amp;&amp;</b>	Λογικό AND
<b>  </b>	Λογικό OR
<b>!</b>	Λογικό NOT

Έστω δύο εκφράσεις  $p$  και  $q$  στις οποίες εφαρμόζουμε λογικούς τελεστές. Ο πίνακας αληθείας είναι (όπου  $T=$ True και  $F=False$ ):

<b>p</b>	<b>q</b>	<b>p&amp;&amp;q</b>	<b>p  q</b>	<b>!p</b>
T	T	T	T	F
T	F	F	T	
F	T	F	T	T
F	F	F	F	

## #5.17 Τελεστές διαχείρισης bits

Τελεστής	Δράση
<b>&amp;</b>	Bitwise AND
<b> </b>	Bitwise OR
<b>^</b>	Bitwise XOR
<b>&lt;&lt;</b>	Ολίσθηση αριστερά
<b>&gt;&gt;</b>	Ολίσθηση δεξιά
<b>~</b>	Συμπλήρωμα ως προς ένα

## #5.18 Σύνθετες εκφράσεις

Με τη χρήση πολλών ειδών τελεστών, μπορούμε να φτιάξουμε σύνθετες εκφράσεις, δηλαδή εκφράσεις που να περιέχουν και αριθμητικές πράξεις αλλά και συγκρίσεις. Οι τιμές των σύνθετων εκφράσεων είναι στην γλώσσα C αυστηρά 0 ή 1. Π.χ. έστω οι μεταβλητές  $a=5$ ,  $b=0$ ,  $x=1$ ,  $y=3$  και η έκφραση: **(b>a) && (x==1)**

Η **(b>a)** θα δώσει αποτέλεσμα 0 διότι δεν ισχύει  $0>5$ . Ενώ η έκφραση:

**(b>a) && (x==1)** θα δώσει αποτέλεσμα 0 και αυτό γιατί **(b>a)** δίνει 0, **(x==1)** δίνει 1. Υπάρχει όμως ο λογικός τελεστής **&&** ο οποίος σύμφωνα με τον πίνακα αληθείας του δίνει 1 μόνο εάν και οι δύο εκφράσεις δίνουν 1. Εδώ επειδή οι εκφράσεις δίνουν η μία 0 και η άλλη 1, το τελικό αποτέλεσμα της σύνθετης έκφρασης θα είναι 0.

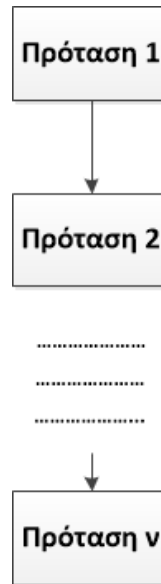
Δομημένος Προγραμματισμός

#6

ΕΛΕΓΧΟΣ ΡΟΗΣ

## #6.1 Ακολουθιακή εκτέλεση προτάσεων

Στον δομημένο προγραμματισμό το πρόγραμμα ολοκληρώνεται όταν εκτελεστούν μια-μια διαδοχικά οι προτάσεις-εντολές του.



## #6.2 Λήψη λογικής απόφασης

Τι γίνεται όμως όταν σε κάποιο βήμα του προγράμματος πρέπει να πάρουμε κάποια απόφασή η οποία εξαρτάται από την επαλήθευση κάποιας συνθήκης;

Τι γίνεται όταν φτάσουμε δηλαδή σε ένα **σημείο απόφασης**;

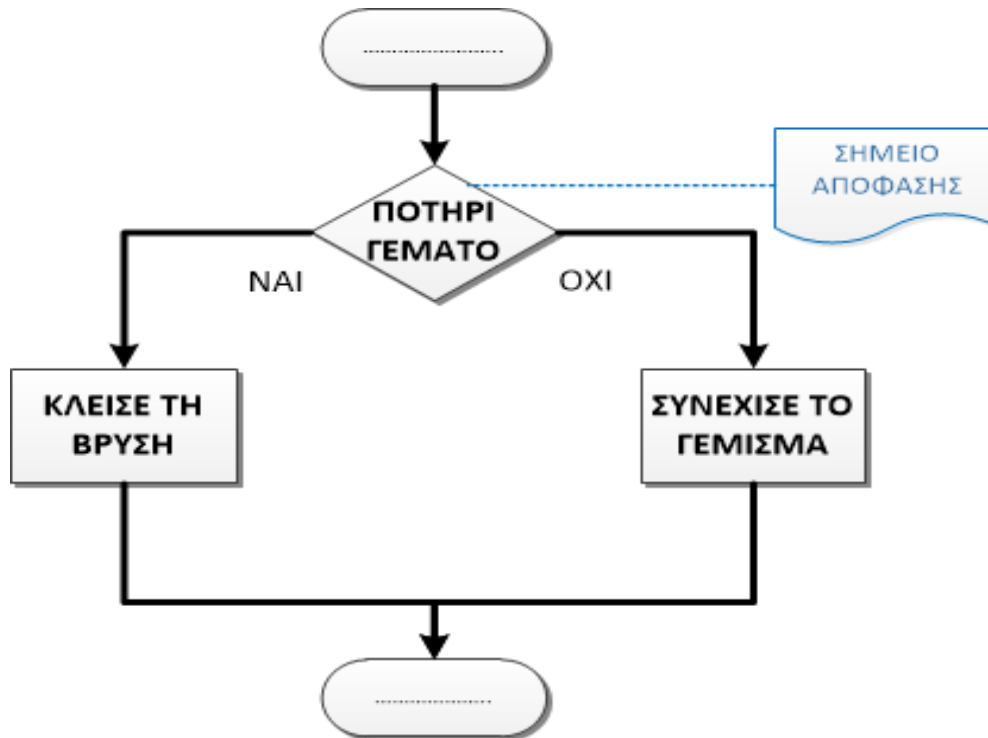
Πως θα πάρουμε την κατάλληλη **λογική απόφαση** έτσι ώστε να κατευθύνουμε κατάλληλα την ροή του προγράμματος;

Παράδειγμα:

**ΕΑΝ το ποτήρι γέμισε ΤΟΤΕ κλείσε την βρύση ΑΛΛΙΩΣ συνέχισε το γέμισμα.**



## #6.3 Σχηματική παράσταση λογικής απόφασης με δύο προτάσεις



## #6.4 Κατασκευές ελέγχου ροής

Για την διαφοροποίηση από την ακολουθιακή εκτέλεση απαιτούνται ειδικές κατασκευές.

Οι κατασκευές διακρίνονται σε δύο βασικές κατηγορίες.

- την **υπό συνθήκη διακλάδωση** (conditional branching) με την οποία οδηγούμε την ροή του προγράμματος σε προτάσεις σύμφωνα με την επαλήθευση κάποιας συνθήκης.
- την **επανάληψη** (looping) με την οποία επαναλαμβάνονται προτάσεις έως ότου πληρωθεί μια συνθήκη ή συμπληρωθεί ένας προκαθορισμένος αριθμός επαναλήψεων.

## #6.5 Υπό συνθήκη διακλάδωση στην γλώσσα C

Η υπό συνθήκη διακλάδωση στην γλώσσα C περιέχει την εντολή ***if*** και την εντολή ***else***. Η σύνταξή της φαίνεται πιο κάτω:

```
if (συνθήκη)
{
    προτάσεις;
}

else
{
    προτάσεις;
}
```

## #6.6 Τα τμήματα της *if*

Η εντολή *if* αποτελείται από τρία τμήματα:

- **Το τμήμα της συνθήκης** που ακολουθεί την λέξη *if*.
- **Το αληθές τμήμα** που βρίσκεται μέσα σε ένα ζεύγος αγκίστρων, ακολουθεί την λέξη *if* και εκτελείται όταν η συνθήκη είναι αληθής.
- **Το ψευδές τμήμα** - εφόσον υπάρχει - που βρίσκεται και αυτό μέσα σε ένα ζεύγος αγκίστρων, ακολουθεί την λέξη *else* και εκτελείται όταν η συνθήκη είναι ψευδής.

## #6.7 Παρατηρήσεις για την *if*

a) Μερικές φορές δεν υπάρχει `else`, δηλαδή ψευδές τμήμα:

```
if (PSIGEIO_adeio==TRUE) gemise_PSIGEIO() ;
```

Εάν η συνθήκη είναι ψευδής δεν γίνεται καμία ενέργεια.

b) Εάν υπάρχουν περισσότερα από δύο τμήματα και απαιτούνται ένθετες προτάσεις ***if/else*** τότε το:

```
else{if(συνθήκη) {προτάσεις;}}
```

Γίνεται:

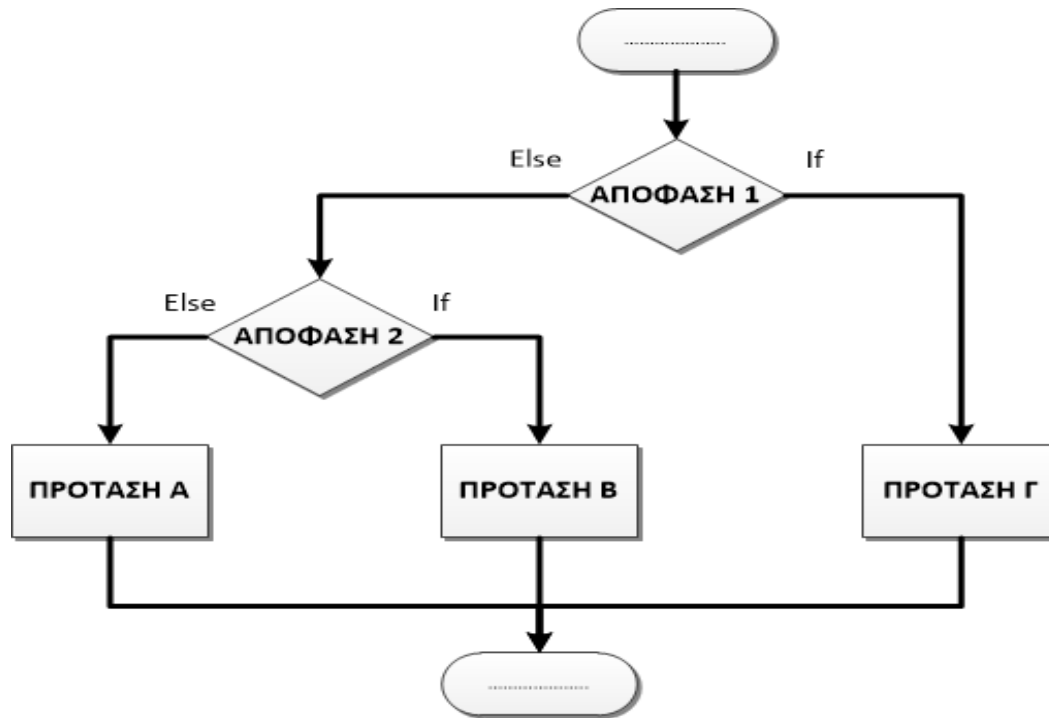
```
else if(συνθήκη) {προτάσεις;}
```

Η μορφή αυτή ονομάζεται κλίμακα ***if-else- if***.

## #6.8 Κλίμακα if-else-if

```
if (συνθήκη)
{
    προτάσεις;
}
else if (συνθήκη)
{
    προτάσεις;
}
. . . . .
else
{
    προτάσεις;
}
```

## #6.9 Σχηματική παράσταση κλίμακας *if-else-if*



### #6.10.1 Υποθετικός τελεστής

Όταν έχουμε να επιλέξουμε ανάμεσα σε δύο και μόνο προτάσεις τότε χρησιμοποιούμε τον τελεστή υπό όρους ή αλλιώς **υποθετικό τελεστή**.

Ο Υποθετικός τελεστής(?:) υλοποιεί μία υποθετική πρόταση.

Η έκφραση με υποθετικό τελεστή:

$\text{εκφρ1} \text{ ? } \text{εκφρ2} \text{ : } \text{εκφρ3}$
--

Σημαίνει ότι η τιμή της πιο πάνω έκφρασης είναι η τιμή της  $\text{εκφρ2}$  εάν η  $\text{εκφρ1}$  είναι αληθής, αλλιώς είναι η τιμή της  $\text{εκφρ3}$ . Η  $\text{εκφρ1}$  είναι η **συνθήκη ελέγχου**.



## #6.10.2 Υποθετικός τελεστής

Παράδειγμα:

$x > z \text{ ? } x : z$
--------------------------

Η πιο πάνω πρόταση σημαίνει ότι η έκφραση θα πάρει την τιμή της μεταβλητής  $x$  εάν το  $x > z$  είναι αληθές, αλλιώς θα πάρει την τιμή της μεταβλητής  $z$ .

## #6.11 Υπό συνθήκη διακλάδωση *switch*

Η εντολή ***switch*** ακολουθείται από μία συνθήκη της οποίας η τιμή αφού υπολογιστεί συγκρίνεται με μια από τις εκφράσεις των περιπτώσεων ***case*** που ακολουθούν.

Όταν η τιμή της συνθήκης είναι ίση με κάποια από τις εκφράσεις τότε ο έλεγχος οδηγείται σε αυτήν, εκτελείται και κατόπιν με την πρόταση ελέγχου ***break*** φεύγει από την *switch* και μεταφέρεται στις προτάσεις που υπάρχουν μετά από την ετικέτα ***default***.

Σημ. Η *switch* διαφέρει από την *if* γιατί ελέγχει μόνο ισότητα.

## #6.12 Σύνταξη της *switch*

```
switch(έκφραση)
{
    case (σταθ.-έκφρ. 1):
        προτάσεις;
    break;
    case (σταθ.-έκφρ. 2):
        προτάσεις;
    break;
    . . . . .
    case (σταθ.-έκφρ. N):
        προτάσεις;
    break;
    default:
        προτάσεις;
    break;
}
```

## #6.13 Κανόνες της *switch*

Οι κανόνες που διέπουν την λειτουργία της υπό συνθήκη διακλάδωσης *switch* είναι οι ακόλουθοι:

- Κάθε ***case*** πρέπει να έχει μία ***int*** ή ***char*** σταθερά έκφραση.
- Δύο ***case*** δεν μπορούν να έχουν την ίδια τιμή.
- Οι προτάσεις κάτω από την ετικέτα ***default*** εκτελούνται όταν δεν ικανοποιείται καμία από τις ***case*** ετικέτες.
- Η ***default*** δεν είναι απαραίτητα η τελευταία ετικέτα.

Δομημένος Προγραμματισμός

**#7**

**ΒΡΟΧΟΙ (I)**

## #7.1 Τι είναι

Ο **Βρόχος** (loop) είναι μια ειδική κατασκευή που αποτελείται από προτάσεις επανάληψης (σώμα προτάσεων επανάληψης).

Οι **προτάσεις επανάληψης** είναι οι προτάσεις που επαναλαμβάνουν ένα σύνολο εντολών, είτε για έναν επιθυμητό αριθμό επαναλήψεων ή έως ότου πληρωθεί μία συνθήκη τερματισμού.

Προσοχή: Όταν δεν υπάρχει δηλωμένη κάποια συνθήκη τερματισμού ή συγκεκριμένος αριθμός επαναλήψεων τότε ο βρόχος λέγεται **ατέρμων βρόχος** (infinity loop), εκτελείται συνέχεια και οδηγεί σε σφάλμα.

## #7.2.1 Κατηγοριοποίηση

Οι βρόχοι ως προς τον τρόπο που ολοκληρώνουν την λειτουργία τους κατηγοριοποιούνται σε δύο κατηγορίες.

- **Βρόχος οδηγούμενος από μετρητή**

είναι ο βρόχος που τερματίζεται μετά το τέλος ενός ορισμένου αριθμού επαναλήψεων.

- **Βρόχος οδηγούμενος από γεγονός**

είναι ο βρόχος που τερματίζεται με την εμφάνιση ενός γεγονότος ή αλλιώς με την πλήρωση ενός κριτηρίου τερματισμού (μιας συνθήκης).

## #7.2.2 Κατηγοριοποίηση

Η δεύτερη κατηγορία κατηγοριοποίησης, δηλαδή των βρόχων που οδηγούνται από γεγονός, χωρίζεται εκ νέου σε δύο κατηγορίες.

- **Βρόχος με συνθήκη εισόδου**

που ο έλεγχος του κριτηρίου τερματισμού (της συνθήκης) γίνεται στην αρχή του βρόχου.

- **Βρόχος με συνθήκη εξόδου**

όπου ο έλεγχος του κριτηρίου τερματισμού (της συνθήκης) γίνεται στο τέλος του βρόχου.



## #7.3 Σύνθετοι βρόχοι

Στον προγραμματισμό συναντάμε συνήθως δύο βρόχους οδηγούμενους και από μετρητή αλλά και από γεγονός και είναι:

- Ο βρόχος ***while-do*** είναι βρόχος με συνθήκη εισόδου, που οδηγείται τόσο από μετρητή όσο και από γεγονός.
- Ο βρόχος ***do-while*** είναι βρόχος με συνθήκη εξόδου, που οδηγείται και αυτός τόσο από μετρητή όσο και από γεγονός.

## #7.4 Βρόχοι στην γλώσσα C – Βρόχος *while*

Ο βρόχος ***while*** είναι βρόχος με συνθήκη εισόδου, οδηγούμενος από γεγονός.

Εκτελείται όσο η συνθήκη παραμένει αληθής.

Όταν η συνθήκη γίνει ψευδής, ο έλεγχος του προγράμματος παρακάμπτει το περιεχόμενο του βρόχου και προχωρά στην επόμενη εντολή που βρίσκεται εκτός βρόχου.

Ο βρόχος *while* έχει εφαρμογή εκεί που δεν είναι γνωστός από την αρχή ο αριθμός των επαναλήψεων.

## #7.5 Σύνταξη του βρόχου *while*

Ο βρόχος *while* συντάσσεται ως εξής:

```
while (συνθήκη) //είσοδος του βρόχου
{
    Πρόταση 1; /* Προτάσεις μέσα στις οποίες
    Πρόταση 2;
    .....
    Πρόταση n;      θα αλλάξει η συνθήκη */
}
```

## #7.6 Λειτουργία του βρόχου *while*

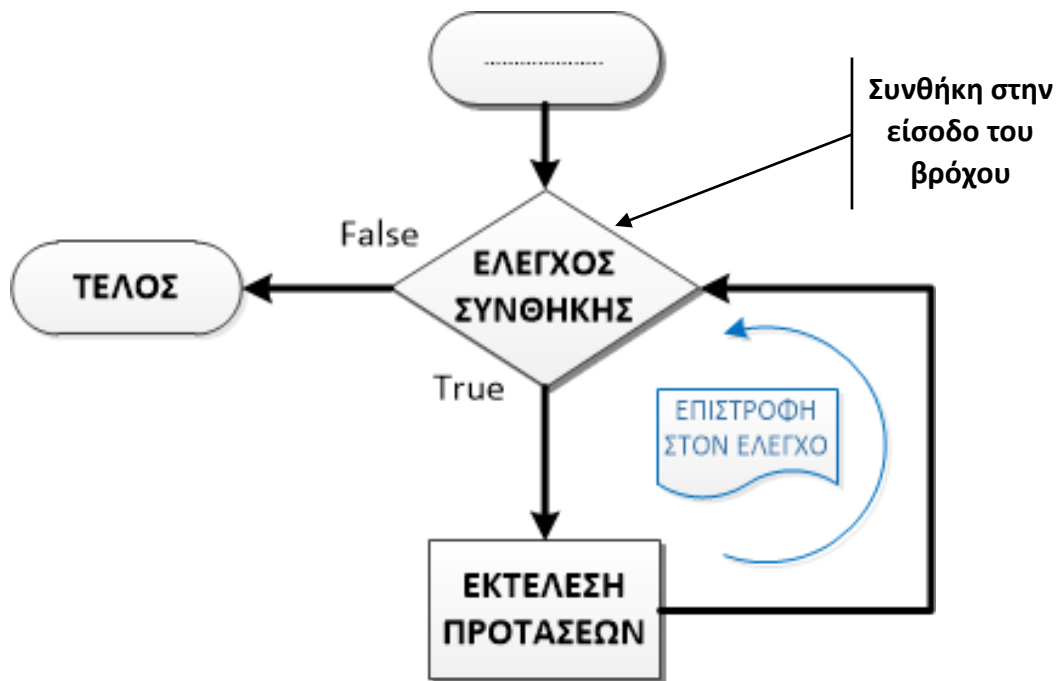
Ο βρόχος *while* λειτουργεί ως ακολούθως:

### Ψευδοκώδικας

- Έλεγε τη συνθήκη
- Εάν η συνθήκη είναι αληθής τότε
  - Εκτέλεσε τις προτάσεις
  - Γύρισε στον έλεγχο της συνθήκης
- Αλλιώς σταμάτησε

Σημ. εάν η συνθήκη είναι ψευδής εξαρχής, δεν θα εκτελεστούν ούτε μια φορά οι προτάσεις του βρόχου.

## #7.7 Σχηματική παράσταση λειτουργίας του βρόχου *while*



## #7.8 Βρόχος *for*

Ο βρόχος ***for*** είναι βρόχος με συνθήκη εισόδου, οδηγούμενος από μετρητή.

Εκτελείται όσο η συνθήκη εισόδου παραμένει αληθής και για έναν συγκεκριμένο αριθμό επαναλήψεων.

Όταν η συνθήκη γίνει ψευδής, ο έλεγχος του προγράμματος παρακάμπτει το περιεχόμενο του βρόχου και προχωρά στην επόμενη εντολή.

Έχει εφαρμογή εκεί που είναι γνωστός από την αρχή ο αριθμός των επαναλήψεων.

## #7.9 Σύνταξη του βρόχου *for*

Ο βρόχος *for* συντάσσεται ως εξής:

```
for (αρχική τιμή μετρητή; συνθήκη; βήμα μετρητή)
{
    Πρόταση 1;
    Πρόταση 2;
    .....
    Πρόταση n;
}
```

## #7.10 Λειτουργία του βρόχου *for*

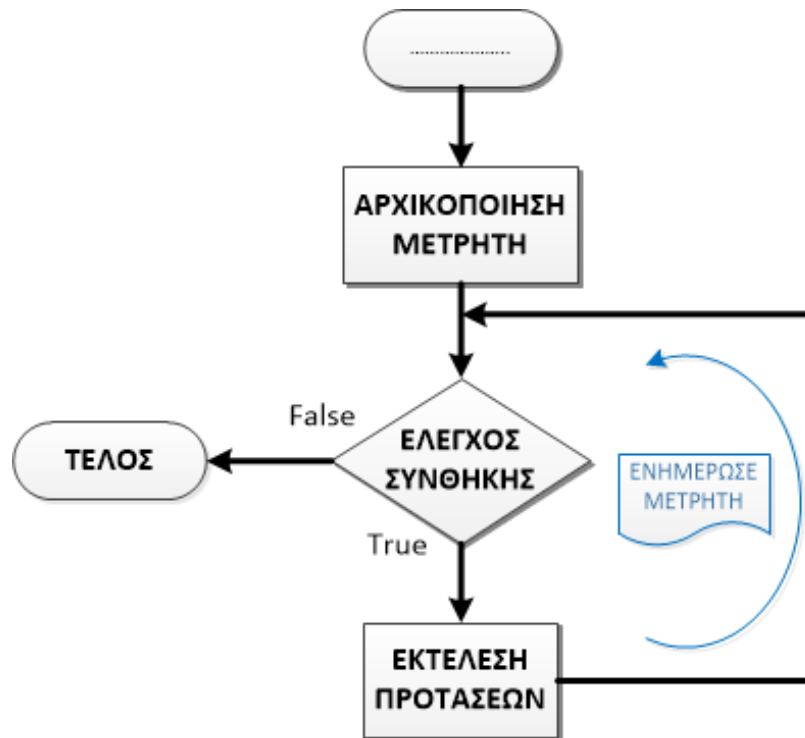
Ο βρόχος *for* λειτουργεί ως ακολούθως:

### Ψευδοκώδικας

- Αρχικοποίησε το μετρητή
- Έλεγε τη συνθήκη
- Εάν η συνθήκη είναι αληθής τότε
  - Εκτέλεσε τις προτάσεις
  - Ενημέρωσε το μετρητή
  - Γύρισε στον έλεγχο της συνθήκης
- Αλλιώς (συνθήκη ψευδής) ενημέρωσε το μετρητή και σταμάτησε.



## #7.11 Σχηματική παράσταση λειτουργίας του βρόχου *for*



## #7.12 Ο τελεστής κόμμα (,) μέσα σε βρόχο for

Ο **τελεστής κόμμα (,)** χρησιμοποιείται μέσα σε βρόχο for, για να μας επιτρέψει την εφαρμογή περισσοτέρων της μιας έκφρασης.

Συγκεκριμένα μέσα σε βρόχο for μπορούμε να έχουμε τέτοιον διαχωρισμό με την εξής μορφή:

```
for (x=0 , y=2; x<=y; x++ , y+=2)
{
    Προταση 1;
    .....
    Προταση n;
}
```

Εδώ έχουμε αρχικοποίηση και χρήση δύο μετρητών (x και y).

Δομημένος Προγραμματισμός

**#8**

**ΒΡΟΧΟΙ (II)**

### #8.1.1 Μετασχηματισμός βρόχου for σε while

Ο βρόχος for μπορεί να μετασχηματιστεί σε βρόχο while και αντίστροφα με τη ακόλουθη μορφή:

Από βρόχο for:

```
for (αρχική τιμή; συνθήκη; βήμα)
{
    προτάσεις;
}
```

## #8.1.2 Μετασχηματισμός βρόχου for σε while

Σε βρόχο while:

```
αρχική τιμή μετρητή;  
while (συνθήκη)  
{  
    προτάσεις;  
    βήμα μετρητή;  
}
```

## #8.2 Βρόχος *do-while*

Ο βρόχος ***do-while*** είναι βρόχος με συνθήκη εξόδου.

Συντάσσεται ως εξής:

```
do
{
    Πρόταση 1; /* Προτάσεις μέσα στις οποίες
    Πρόταση n;  θα αλλάξει η συνθήκη */
}
while (συνθήκη);
```

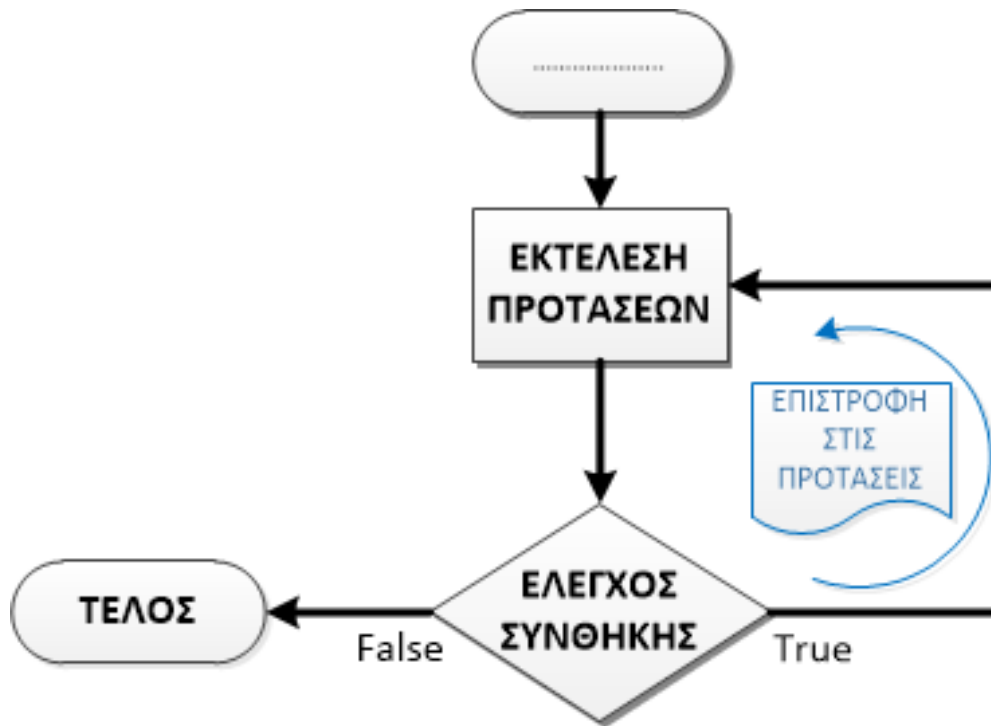
## #8.3 Λειτουργία του βρόχου *do-while*

Ο βρόχος do – while λειτουργεί ως ακολούθως:

### Ψευδοκώδικας

- Εκτέλεσε τις προτάσεις
- Έλεγε τη συνθήκη
- Εάν η συνθήκη είναι αληθής τότε
  - Ξεκίνησε από την αρχή
- Αλλιώς σταμάτησε.

## #8.4 Σχηματική παράσταση λειτουργίας του βρόχου *do-while*





## #8.5.1 Μετασχηματισμός βρόχου for σε do/while

Ο βρόχος for μπορεί να μετασχηματιστεί σε βρόχο do/while και αντίστροφα με τη ακόλουθη μορφή:

Από βρόχο for:

```
for (αρχική τιμή; συνθήκη; βήμα)
{
    προτάσεις;
}
```

## #8.5.2 Μετασχηματισμός βρόχου for σε do/while

Σε βρόχο while:

```
αρχική τιμή μετρητή;  
do  
{  
    προτάσεις;  
    βήμα μετρητή;  
}  
while (συνθήκη);
```

## #8.6 Φωλιασμένοι βρόχοι (*nested*)

**Φωλιασμένος** (Ένθετος) βρόχος ονομάζεται ο βρόχος που βρίσκεται μέσα σε έναν άλλο.

Ο εσωτερικός βρόχος είναι μία εσωτερική πρόταση άρα πρώτα εκτελείται ο εσωτερικός βρόχος και μετά εκτελείται η επόμενη επανάληψη του εξωτερικού.

Ο συνολικός αριθμός επαναλήψεων σε έναν πολλαπλό βρόχο είναι το γινόμενο του αριθμού των επαναλήψεων όλων των επιμέρους βρόχων.

Στην C δεν υπάρχει περιορισμός στην ένθεση των βρόχων, άρα μπορούμε να έχουμε πολλαπλή ένθεση.

## #8.7 Σύνταξη φωλιασμένου βρόχου

Οι φωλιασμένοι βρόχοι for συντάσσονται ως ακολούθως:

```
for (αρχική τιμή; συνθήκη; βήμα)
{
    for (αρχική τιμή; συνθήκη; βήμα)
    {
        προτάσεις; //προτάσεις φωλιασμένου βρόχου
    }
    προτάσεις; //προτάσεις εξωτερικού βρόχου
}
```

### #8.8.1 Διακόπτοντας τους βρόχους - Εντολή *break*

Κάποιες φορές θα πρέπει να διακόψουμε την λειτουργία ενός βρόχου με την εμφάνιση ενός γεγονότος. Γι' αυτό το λόγο έχουμε την εντολή *break* και την βοηθητική της *continue*.


Η εντολή ***break*** χρησιμοποιείται σε μία πρόταση για να διακόπτει την εκτέλεση ενός βρόχου, οδηγώντας τον έλεγχο του προγράμματος στην εντολή που βρίσκεται αμέσως μετά το βρόχο.

Όταν βρίσκεται μέσα σε φωλιασμένο βρόχο, τότε επηρεάζεται μόνο ο εσωτερικός βρόχος.

## #8.8.2 Εντολή *break* - σύνταξη

Η εντολή ***break*** χρησιμοποιείται ως ακολούθως:

```
while (συνθήκη)
{
    προτάσεις;
    if (συνθήκη)
        break; // η εντολή οδηγεί
                // στο τέλος του βρόχου
    .....
} // τέλος βρόχου
```



### #8.9.1 Μεταφορά στην αρχή του βρόχου - Εντολή *continue*

Η εντολή ***continue*** οδηγεί τον έλεγχο της ροής στην αρχή του βρόχου, παραλείποντας την εκτέλεση του υπόλοιπου τμήματος του σώματος του βρόχου και προχωρώντας στην επόμενη επανάληψη.

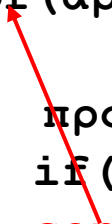
Στους βρόχους *while* και *do-while*, η *continue* υποχρεώνει τον έλεγχο του προγράμματος να περάσει κατευθείαν στη συνθήκη ελέγχου και να προχωρήσει κατόπιν στην επεξεργασία του βρόχου.

Στην *for* ο εκτελείται πρώτα το τμήμα του βρόχου και κατόπιν ελέγχεται η συνθήκη ελέγχου, προτού συνεχισθεί η εκτέλεση του βρόχου.

## #8.9.2 Εντολή *continue* - σύνταξη

Η εντολή ***continue*** χρησιμοποιείται ως ακολούθως:

```
for(αρχική τιμή; συνθήκη; βήμα)
{
    προτάσεις;
    if(συνθήκη)
        continue; // η εντολή οδηγεί
                   // στην αρχή του βρόχου
    .....
}
```





### #8.10.1 Μεταφορά σε συγκεκριμένο σημείο - Εντολή *goto*

Η εντολή ***goto*** οδηγεί τον έλεγχο της ροής σε συγκεκριμένο σημείο του κώδικα, χωρίς να επαληθεύεται κάποια συνθήκη.

Ο έλεγχος πηγαίνει σε συγκεκριμένη ετικέτα (*label*) που ακολουθεί την εντολή *goto*.

Χρησιμοποιήθηκε ευρέως στον προγραμματισμό των περασμένων δεκαετιών, δημιουργώντας όμως προβλήματα στα προγράμματα γι' αυτό και καταργήθηκε.

## #8.10.2 Εντολή *goto* – σύνταξη

Η εντολή ***goto*** χρησιμοποιείται ως ακολούθως:

```
void main()  
{  
    .....  
    label_1:  
    προτάσεις;  
    .....  
    if (συνθήκη)  
        πρόταση;  
    goto label_1; // η εντολή οδηγεί  
                 //σε συγκεκριμένο σημείο του κώδικα  
}
```

Δομημένος Προγραμματισμός

#9

ΠΙΝΑΚΕΣ

### #9.1.1 Μονοδιάστατοι πίνακες

**Πίνακας** (*array*) είναι ένα σύνολο από θέσεις μνήμης όπου η κάθε μία περιέχει δεδομένα του ίδιου τύπου. Κάθε θέση ονομάζεται στοιχείο του πίνακα. Τον πίνακα τον αντιμετωπίζουμε ως μεταβλητή πολλών θέσεων.

Δηλώνουμε έναν πίνακα γράφοντας τον τύπο, μετά το όνομα του πίνακα και τέλος έναν δείκτη (*subscript*) μέσα σε τετράγωνες αγκύλες που είναι ο αριθμός των στοιχείων (θέσεων) του πίνακα.

Δηλαδή:

```
int ArrayName[25] ;
```

### #9.1.2 Μονοδιάστατοι πίνακες

Ο μεταγλωττιστής (*compiler*) για αυτόν τον πίνακα δεσμεύει 100 bytes. (δηλ. τύπος ακεραίου `int` = 4 bytes, άρα 4 bytes \* 25 στοιχεία = 100 bytes).

Τα στοιχεία του πίνακα αριθμούνται από το 0, δηλαδή το πρώτο στοιχείο του πίνακα είναι `ArrayName[0]`, το δεύτερο `ArrayName[1]`, το τρίτο `ArrayName[2]` κοκ. Άρα εάν έχουμε έναν πίνακα με  $n$  στοιχεία, η αρίθμηση τους είναι:

**`όνομα_πίνακα[0], όνομα_πίνακα[1],..... όνομα_πίνακα [n-1]`**

Άρα ο πίνακας του παραδείγματος έχει 25 στοιχεία που αριθμούνται από το `ArrayName[0]` έως το `ArrayName[24]`.

### #9.1.3 Μονοδιάστατοι πίνακες

Ο *compiler* υπολογίζει που θα αποθηκεύσει την τιμή σε ένα στοιχείο του πίνακα ανάλογα με το μέγεθος και τον δείκτη.

Π.χ. θέλουμε να αποθηκεύσει μία τιμή στο στοιχείο *ArrayName[5]* (έκτο στοιχείο). Ο μεταγλωττιστής πολλαπλασιάζει την μετατόπιση επί το μέγεθος του κάθε στοιχείου **5X4bytes**. Μετακινείται από την αρχή **20bytes** και γράφει την τιμή του στοιχείου.

Προσοχή!! εάν του δώσουμε να γράψει στο *ArrayName[50]*, αγνοεί ότι δεν υπάρχει αυτό το στοιχείο στον πίνακα, υπολογίζει την απόσταση (200bytes) και πάει και γράφει εκεί που μπορεί να υπάρχουν άλλα δεδομένα.

## #9.2 Αρχικοποίηση πίνακα

Για να **αρχικοποιήσουμε** έναν μονοδιάστατο πίνακα γράφουμε, τον τύπο του, όνομά του, το πλήθος των στοιχείων του και κατόπιν τις τιμές που θέλουμε τα στοιχεία του να έχουν μέσα σε άγκιστρα.

Έστω ότι έχουμε τη δήλωση: **int intArray[5]={1,11,21,31,41};**

Αυτό σημαίνει ότι το στοιχείο:

*intArray*[0] θα πάρει την τιμή 1,

1	11	21	31	41
---	----	----	----	----

το *intArray*[1] την τιμή 11,

το *intArray*[2] την τιμή 21,

το *intArray*[3] την τιμή 31 και το *intArray*[4] την τιμή 41.

### #9.3.1 Επισημάνσεις

- Εάν παραλείψουμε το μέγεθος του πίνακα (τον δείκτη) αλλά αρχικοποιήσουμε με τιμές, ο compiler θα φτιάξει έναν πίνακα που να χωρέσει τις τιμές αυτές.
- Δεν πρέπει να αρχικοποιούμε έναν πίνακα με τιμές περισσότερες από το μέγεθός του διότι οι παραπάνω τιμές γράφονται έξω από τον πίνακα όπου μπορεί να υπάρχουν άλλες πληροφορίες.
- Εάν αρχικοποιήσουμε έναν πίνακα με λιγότερες τιμές, τότε ο compiler θα αρχικοποιήσει τα πρώτα στοιχεία με τις δοθείσες τιμές και τα υπόλοιπα με τιμές 0.



### #9.3.2 Επισημάνσεις

- Μπορούμε να αρχικοποιήσουμε έναν πίνακα με μηδενικές τιμές γράφοντας μόνο μια φορά την τιμή 0

```
int array[10]={0};
```

Ή μέσα σε ένα βρόχο:

```
for(int a=0; a<10; a++)  
    array[a]=0;
```

- Προσοχή στον δείκτη. Στη δήλωση `int intArray[5];` ο δείκτης 5 καθορίζει το μέγεθος του πίνακα, ενώ στο `intArray[4]=41;` ο δείκτης 4 δηλώνει ότι το 5<sup>ο</sup> στοιχείο του πίνακα που αναφερόμαστε, θα πάρει την τιμή 41.

## #9.4 Μέγεθος πίνακα

Μπορούμε να βρούμε το μέγεθος σε bytes ενός πίνακα χρησιμοποιώντας τον τελεστή ***sizeof()***.

Ο τελεστής αυτός επιστρέφει το μέγεθος σε bytes της μεταβλητής ή του πίνακα στον οποίο εφαρμόζεται. Π.χ.

- Η δήλωση: ***sizeof(ονομα\_πίνακα)*** μας δείχνει το μέγεθος του πίνακα σε bytes.
- Η δήλωση: ***sizeof(ονομα\_πίνακα[δ])*** μας δείχνει το μέγεθος σε bytes του συγκεκριμένου στοιχείου του πίνακα.
- Η διαίρεση: ***sizeof(ονομα\_πίνακα) / sizeof(ονομα\_πίνακα[δ])*** μας δείχνει το αριθμό των στοιχείων του πίνακα.

## #9.5 Δισδιάστατοι πίνακες

**Δισδιάστατοι** είναι οι πίνακες των οποίων τα στοιχεία των γραμμών τους είναι επίσης πίνακες.

Η πρόταση :

```
int array[3][10];
```

Δηλώνει την μεταβλητή *array* σαν πίνακα ακεραίων αριθμητικών τιμών 3 στοιχείων, που κάθε στοιχείο της είναι ένας πίνακας 10 στοιχείων.

Σημ. Στην C δεν υπάρχει περιορισμός στον αριθμό διαστάσεων ενός πίνακα.

## #9.6 Αναφορά στα στοιχεία του πίνακα

Για να αναφερθούμε σε ένα συγκεκριμένο στοιχείο ενός δισδιάστατου πίνακα θα πρέπει να καθοριστούν σωστά οι δείκτες.

Η αναφορά είναι της μορφής: **όνομα\_πίνακα[δ1][δ2]**,

όπου δ1=γραμμή και δ2=στήλη.

Αν λοιπόν χρησιμοποιήσουμε έναν δείκτη στην αναφορά ενός πίνακα, δηλ. **όνομα\_πίνακα[2]**, τότε αναφερόμαστε στην τρίτη γραμμή του πίνακα.

Ενώ αν χρησιμοποιήσουμε δύο δείκτες δηλ. **όνομα\_πίνακα[2][4]**, τότε αναφερόμαστε στο πέμπτο στοιχείο της τρίτης γραμμής του πίνακα.

### #9.7.1 Αρχικοποίηση δισδιάστατου πίνακα

Για να αρχικοποιηθεί ένας δισδιάστατος πίνακας θα πρέπει η δήλωση κάθε γραμμής να περικλείεται από άγκιστρα, οι γραμμές να χωρίζονται μεταξύ τους με κόμματα και όλες οι γραμμές να περικλείονται εκ νέου με άγκιστρα. Αν δεν υπάρχουν σε κάποια στοιχεία τιμές τότε τα στοιχεία αυτά παίρνουν την τιμή 0. Στην δήλωση:

```
int array[3][3]={ {1,2,3},{4,5,6},{7,8,9} };
```

ο πίνακας που δημιουργείται είναι ο:

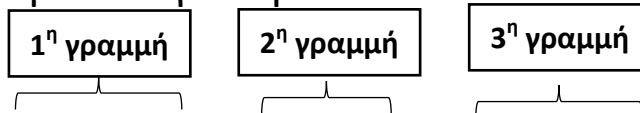


1	2	3
4	5	6
7	8	9

## #9.7.2 Αρχικοποίηση δισδιάστατου πίνακα

Προσέχουμε πάντα που βάζουμε τα άγκιστρα.

Η δήλωση:



```
int Arr[4][3]={ {3,2,1}, {6,4}, {7,8,9} }
```

Θα δημιουργήσει έναν πίνακα της μορφής:

Σημ. Βλέπουμε ότι ο compiler το 3<sup>ο</sup> στοιχείο της

3<sup>η</sup> γραμμή που λείπει το συμπληρώνει με την τιμή 0

και στην 4<sup>η</sup> που λείπει ολόκληρη από την

αρχικοποίηση επίσης με την τιμή 0.

3	2	1
6	4	0
7	8	9
0	0	0

### #9.7.3 Αρχικοποίηση δισδιάστατου πίνακα

Στον ίδιο πίνακα η δήλωση χωρίς τα εσωτερικά άγκιστρα:

1 <sup>η</sup> γραμμή	2 <sup>η</sup> γραμμή	3 <sup>η</sup> γραμμή
-----------------------	-----------------------	-----------------------

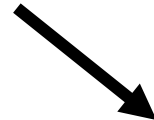
  

`int Arr[4][3]={3,2,1, 6,4,7, 8,9}`

θα δημιουργήσει έναν πίνακα της μορφής:

Σημ. εδώ ο *compiler* θα καταχωρήσει

κατά σειρά τις τιμές που του δίνουμε και τις υπόλοιπες κενές θέσεις θα τις θέσει στο 0.



3	2	1
6	4	7
8	9	0
0	0	0

### #9.8.1 Πολυδιάστατοι πίνακες

Ένας πολυδιάστατος πίνακας δηλώνεται όπως παρακάτω:

**τύπος όνομα\_πίνακα [δ1][δ2][δ3]**

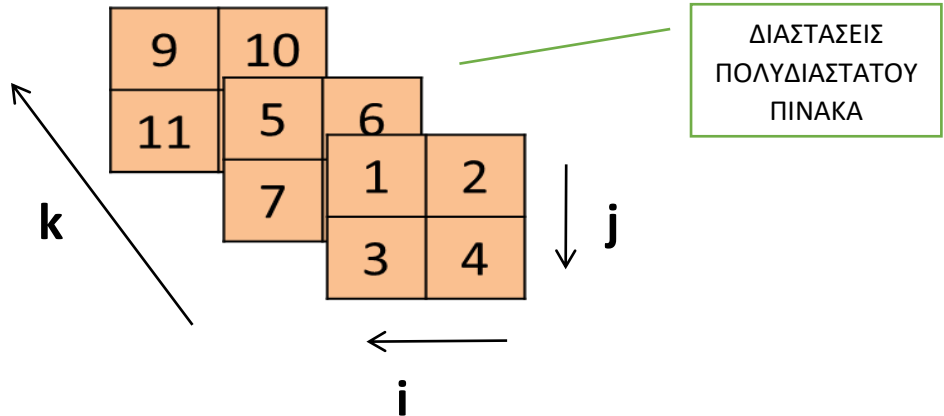
Δήλωση και αρχικοποίηση ενός ακέραιου πίνακα 2X2X2 γίνεται:

```
int Array[2][2][2]=  
{{{1,2},{3,4}},{5,6},{7,8}},{9,10},{11,12}}};
```



## #9.8.2 Πολυδιάστατοι πίνακες

Ο πίνακας που θα προκύψει θα είναι ο ακόλουθος:



Η προσπέλαση του πίνακα γίνεται με 3 βρόχους *for* και τρεις δείκτες.  
Δηλαδή: `Array[i][j][k]`.

## #9.9 Πίνακες χαρακτήρων – Αλφαριθμητικά

Για να αποθηκεύσουμε αλφαριθμητικά σε ένα πρόγραμμα χρησιμοποιούμε τους πίνακες χαρακτήρων.

Οι **πίνακες χαρακτήρων** είναι μονοδιάστατοι πίνακες που τερματίζονται με τον χαρακτήρα **\0**, που είναι το μηδέν του κώδικα ASCII.

Επειδή πάντα το αλφαριθμητικό τερματίζεται με το **\0**, ο πίνακας πρέπει να δηλώνεται με έναν στοιχείο περισσότερο από τον αριθμό των χαρακτήρων που θα καταχωρήσουμε.

## #9.10 Δήλωση πίνακα χαρακτήρων

Για να γράψουμε την λέξη Hello, δηλώνουμε και αρχικοποιούμε έναν πίνακα χαρακτήρων ως εξής:

```
char array[6]={ 'H' , 'e' , 'l' , 'l' , 'o' , '\0' } ;
```

Αλλιώς θα πρέπει να γράψουμε κατά σειρά τις εξής προτάσεις:

```
char array[5]; //δήλωση πίνακα χαρακτήρων  
array[0]='H'; //εισαγωγή χαρακτήρα στο 1° στοιχείο  
array[1]='e'; //.....  
array[2]='l'; //.....  
array[3]='l'; //.....  
array[4]='o'; // εισαγωγή χαρακτήρα στο 5° στοιχείο
```

## #9.11 Είσοδος – Έξοδος Αλφαριθμητικού

Για να εισάγουμε ή να διαβάσουμε ένα αλφαριθμητικό σε έναν πίνακα χαρακτήρων χρησιμοποιούμε τις συναρτήσεις *scanf()* και *printf()* μαζί με τον προσδιοριστή *%s* και χωρίς την χρήση του τελεστή διεύθυνσης **&** πριν από όρισμα που είναι το όνομα του πίνακα, γιατί το όνομα του πίνακα ουσιαστικά δείχνει το πρώτο του στοιχείο.

Δηλαδή γράφουμε:

```
scanf("%s", table_name)
printf("%s", table_name)
```

## #9.12 Άλλος τρόπος εισόδου – εξόδου αλφαριθμητικού

- Για την εισαγωγή ενός αλφαριθμητικού, καλούμε την συνάρτηση **gets()** με όρισμα το όνομα του πίνακα χωρίς δείκτη:

```
gets (όνομα_πίνακα)
```

Το αλφαριθμητικό θα καταχωρηθεί εωσότου πατήσουμε το *enter*.

- Για την εκτύπωση του αλφαριθμητικού στην οθόνη καλούμε την συμπληρωματική συνάρτηση της *gets()*, την **puts()**:

```
puts (όνομα_πίνακα)
```

Το μειονέκτημα με την *puts* είναι ότι δεν μπορούμε να μορφοποιήσουμε την έξοδο όπως με την *printf()*.

Σημ. Οι συναρτήσεις *gets()* και *puts()* βρίσκονται στο αρχείο επικεφαλίδας *<stdio.h>*.

## #9.13 Συναρτήσεις αλφαριθμητικού

Για τον χειρισμό των αλφαριθμητικών χρησιμοποιούμε συναρτήσεις που βρίσκονται στο αρχείο κεφαλίδας **<string.h>**.

Αυτές είναι:

<b><i>strlen()</i></b>	για εύρεση του μήκους ενός string
<b><i>strcpy()</i></b> και <b><i>strncpy()</i></b>	για αντιγραφή ενός string και αντιγραφή των <b><i>n</i></b> πρώτων χαρακτήρων
<b><i>strcat()</i></b> και <b><i>strncat()</i></b>	για συνένωση 2 strings και συνένωση των <b><i>n</i></b> πρώτων χαρακτήρων
<b><i>strcmp()</i></b> και <b><i>strncmp()</i></b>	για σύγκριση 2 strings και σύγκριση των <b><i>n</i></b> πρώτων χαρακτήρων
<b><i>strchr()</i></b>	για εύρεση χαρακτήρα σε string
<b><i>strstr()</i></b>	για εύρεση string σε string

Δομημένος Προγραμματισμός

**#10**

**ΣΥΝΑΡΤΗΣΕΙΣ**

### #10.1.1 Τι είναι συνάρτηση

Στο δομημένο προγραμματισμό προσπαθούμε την επίλυση ενός σύνθετου προβλήματος να την μοιράσουμε σε μικρότερες λειτουργικές και πιο απλές μονάδες.

Η **Συνάρτηση** είναι μία από αυτές τις λειτουργικές μονάδες, είναι δηλαδή ένα κομμάτι του προγράμματος, που εκτελεί μία συγκεκριμένη λειτουργία.

Μέσα σε ένα πρόγραμμα της C μπορεί να υπάρχουν αυτόνομες συναρτήσεις καθώς και η `main()` που είναι η βασική συνάρτηση του προγράμματος .



## #10.1.2 Τι είναι συνάρτηση

Η συνάρτηση έχει ένα συγκεκριμένο

- **όνομα** (συναφές με την δουλειά που καλείται να κάνει) με το οποίο την καλούμε,
- **εισόδους** (παραμέτρους ή αλλιώς μεταβλητές),
- **έξοδο** (μια επιστρεφόμενη τιμή), αλλά και
- **εντολές** (ή μπλοκ εντολών) με τις οποίες επιτυγχάνεται η λύση.

Η συνάρτηση καλείται μέσα στην βασική συνάρτηση της C (την *main*) ή και από άλλες συναρτήσεις.

Η συνάρτηση μπορεί επίσης με την σειρά της, να καλέσει άλλες συναρτήσεις.

## #10.2 Σχηματική παράσταση λειτουργίας κλήσεων συναρτήσεων

### ΠΡΟΓΡΑΜΜΑ

```
main()
```

```
{
```

```
Δήλωση;
```

```
Συνάρτηση1();
```

```
Δήλωση;
```

```
Συνάρτηση2();
```

```
Δήλωση;
```

```
Συνάρτηση4();
```

```
}
```

Συνάρτηση1

Εντολή1;

.....;

return;

Συνάρτηση2

Εντολή1;

Δήλωση;

Συνάρτηση3();

return;

Συνάρτηση3

Εντολή;

.....;

return;

Συνάρτηση4

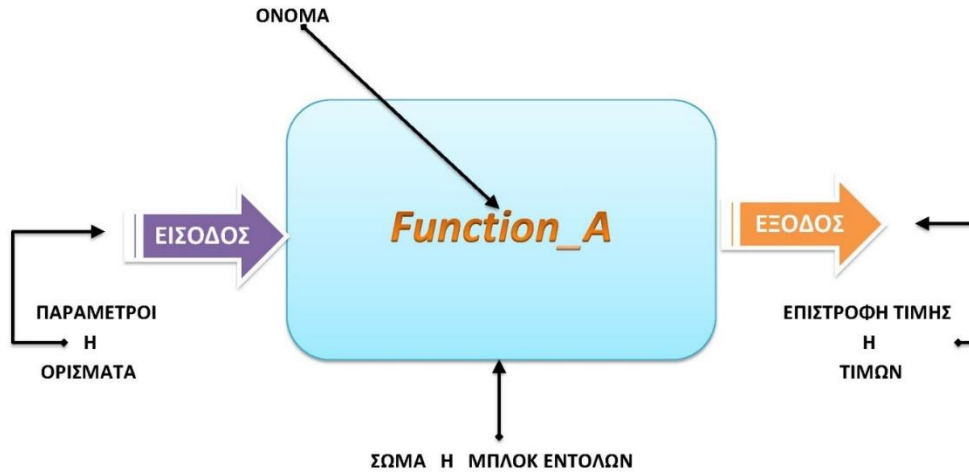
Εντολή1;

.....;

return;

## #10.3 Απεικόνιση συνάρτησης

### ΑΠΕΙΚΟΝΙΣΗ ΣΥΝΑΡΤΗΣΗΣ



## #10.4 Η συνάρτηση μέσα στο πρόγραμμα

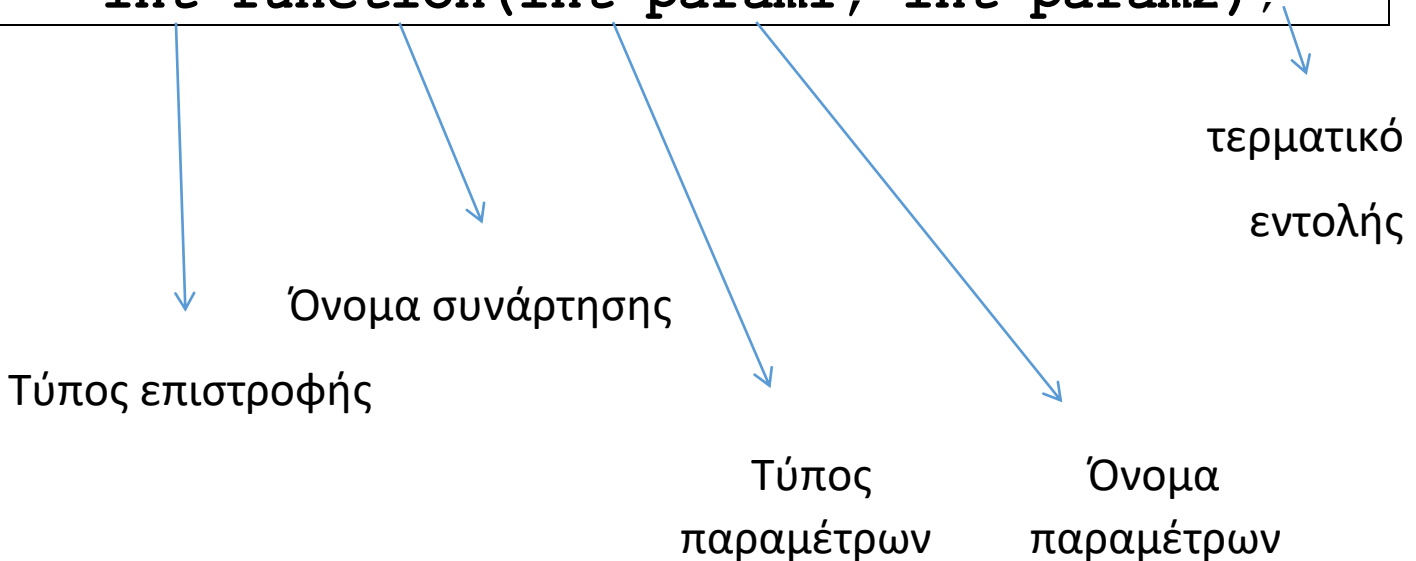
Οι συναρτήσεις μέσα σε ένα σημείο του προγράμματος καλούνται αφού πρώτα έχουν δηλωθεί στην αρχή του προγράμματος και έχουν οριστεί συνήθως μετά από την `main()`. Άρα για να χρησιμοποιήσουμε μια συνάρτηση πρέπει να κάνουμε τα ακόλουθα:

- **Δήλωση:** Επισημαίνουμε στον Compiler ότι θα χρησιμοποιήσουμε στο πρόγραμμα τη συνάρτηση. Η δήλωση λέγεται και *Πρωτότυπο*.
- **Ορισμός:** Περιγράφεται η λειτουργία της συνάρτησης με άλλα λόγια γράφουμε τον κώδικα της συνάρτησης.
- **Κλήση:** Καλούμε την συνάρτηση ή αλλιώς εκτελούμε τις εντολές της (τον κώδικα).

## #10.5.1 Δήλωση συνάρτησης

Η δήλωση (Πρότυπο) μιας συνάρτησης στο πρόγραμμα είναι η εξής:

```
int function(int param1, int param2);
```



## #10.5.2 Δήλωση συνάρτησης

Το **Πρότυπο** (ή αλλιώς επικεφαλίδα της συνάρτησης) έχει όνομα, εισόδους (παράμετροι με τύπο) αλλά και έξοδο (τύπος επιστρεφόμενης τιμής)

Το *Πρότυπο* της συνάρτησης είναι μία εντολή άρα θα τελειώνει με **τερματικό εντολής (;).**

Στη *δήλωση* (Πρότυπο) και τη *κλήση* μιας συνάρτησης χρησιμοποιούμε **πάντα** τερματικό εντολής.

Στον *ορισμό* της ποτέ.

## #10.6.1 Ορισμός συνάρτησης

επικεφαλίδα συνάρτησης

```
int function(int param1, int param2)
```

{  
 εντολή1;  
 εντολή2;  
 return(value) ;  
}

σώμα συνάρτησης

Λέξη κλειδί

Τιμή επιστροφής

The diagram illustrates the components of a C function definition. The function signature `int function(int param1, int param2)` is shown with a bracket underneath it, and an arrow points from the label 'επικεφαλίδα συνάρτησης' (function header) to this bracket. The function body is enclosed in curly braces `{ ... }`, with an arrow pointing from the label 'σώμα συνάρτησης' (function body) to the right brace. Inside the body, the `return(value) ;` statement is highlighted with two arrows: one from 'Λέξη κλειδί' (keyword) pointing to `return`, and another from 'Τιμή επιστροφής' (return value) pointing to `value`.

## #10.6.2 Ορισμός συνάρτησης

Στον ορισμό της συνάρτησης -δηλαδή στην περιγραφή της λειτουργίας της συνάρτησης- χρησιμοποιούμε την επικεφαλίδα της συνάρτησης, δηλαδή: α) το όνομά της, β) τις παραμέτρους στο όρισμά της και γ) μέσα σε ένα ζεύγος αγκίστρων, τις εντολές της συνάρτησης καθώς και τις τοπικές μεταβλητές που χρησιμοποιεί. Επίσης την λέξη **return** αν η συνάρτηση πρέπει να επιστρέφει μία τιμή.

Στον ορισμό της συνάρτησης δεν χρησιμοποιούμε το ερωτηματικό.

Σημ. Εάν δεν χρησιμοποιήσουμε πρωτότυπο τότε θα πρέπει να ορίζουμε την συνάρτηση πριν από την `main()` (πριν από την κλήση της).




## #10.7 Κλήση συνάρτησης

Κατά την κλήση μιας συνάρτησης μέσα σε ένα πρόγραμμα χρησιμοποιούνται: α) το όνομά της, β) οι πραγματικές παράμετροι εάν υπάρχουν και γ) το τερματικό εντολής.

Δεν χρησιμοποιούμε τον τύπο επιστροφής.

```
void main()  
{   Εντολή1;  
    Εντολή2;  
    function(param1, param2) ;  
    Εντολή3;  
}
```

κλήση συνάρτησης



## #10.8 Συνάρτηση χωρίς επιστρεφόμενη τιμή

Οι συναρτήσεις εάν δεν επιστρέφουν τιμή τότε ο τύπος επιστροφής τιμής δηλώνεται **void** και δεν υπάρχει μέσα στο σώμα της συνάρτησης η λέξη κλειδί return. Στο παράδειγμα που ακολουθεί, βλέπουμε τον ορισμό της συνάρτησης *tempMeasure()* που δεν επιστρέφει κάποια τιμή.

```
void tempMeasure()  
{  
    εντολή1;  
    εντολή2;  
    .....}
```

## #10.9 Συνάρτηση με επιστρεφόμενη τιμή

Οι συναρτήσεις εάν επιστρέφουν τιμή, τότε ο τύπος επιστροφής τιμής δηλώνεται ως *int* ή *float* και μέσα στο σώμα της συνάρτησης υπάρχει η λέξη κλειδί **return** η οποία μας επιστρέφει μια τιμή. Στο παράδειγμα που ακολουθεί, βλέπουμε τον ορισμό της συνάρτησης *tempMeasure()* που επιστρέφει την τιμή που προκύπτει, από το γινόμενο των παραμέτρων *x* και *y* που είναι και τα ορίσματα της συνάρτησης.

```
int tempMeasure(int x, int y)
{
    εντολή1;
    return x*y;
}
```

## #10.10 Επιστρεφόμενες τιμές

Μια συνάρτηση μπορεί να επιστρέφει:

a) μία **τιμή** δηλαδή: **return 10 ;**

b) μία **παράσταση** δηλαδή: **return (x>3) ;**

(Εδώ επιστρέφεται **0** εάν η παράσταση στην παρένθεση είναι αληθής ή **1** αν είναι ψευδής και όχι την τιμή του **x**).

c) Η ακόμα την **τιμή μιας συνάρτησης** δηλαδή:

**return (FunctionOne()) ;**

Μπορούμε να έχουμε μέσα σε μία συνάρτηση **περισσότερες από μία επιστροφές** τιμών δηλαδή:

**(return 5 else return -1)**

## #10.11 Τοπικές μεταβλητές

**Τοπικές μεταβλητές** είναι οι μεταβλητές που δηλώνονται και έχουν εμβέλεια μόνο μέσα στην ίδια τη συνάρτηση. Όταν η συνάρτηση επιστρέφει τότε οι μεταβλητές αυτές δεν είναι πλέον διαθέσιμες.

Από αυτό συμπεραίνουμε ότι οι μεταβλητές αυτές δεσμεύουν μνήμη μόνο κατά την εκτέλεση της συνάρτησης. Άρα μετά το τέλος της συνάρτησης την αποδεσμεύουν.

Οι μεταβλητές αυτές δηλώνονται όπως και κάθε άλλη μεταβλητή.

Επίσης τοπικές μεταβλητές θεωρούμε και τις **παραμέτρους** (ή αλλιώς **τα ορίσματα**) της συνάρτησης. Τοπικές θεωρούνται και οι μεταβλητές που δηλώνονται μέσα στην *main()*.

## #10.12 Καθολικές μεταβλητές

**Καθολικές μεταβλητές** λέγονται αυτές οι μεταβλητές που δηλώνονται έξω από μία συνάρτηση (έξω και από την `main`) και έχουν καθολική εμβέλεια.

Οι μεταβλητές αυτές δεσμεύουν μνήμη καθ' όλη την διάρκεια εκτέλεσης του προγράμματος.

Είναι διαθέσιμες δηλαδή για οποιαδήποτε συνάρτηση μέσα στο πρόγραμμα αλλά και για την `main()`.

Καλό είναι να αποφεύγονται γιατί μπερδεύουν τον κώδικα του προγράμματος και οδηγούν πολλές φορές σε σφάλματα.

## #10.13 Συναρτήσεις με πίνακες ως παραμέτρους

Μπορούμε να χρησιμοποιήσουμε στην κλήση μιας συνάρτησης σαν πραγματική παράμετρο το όνομα ενός πίνακα.

Στην προκειμένη περίπτωση δεν στέλνουμε στην συνάρτηση όλο τον πίνακα αλλά την διεύθυνση του πρώτου του στοιχείου.

Στην δήλωση της συνάρτησης ορίζουμε και τον τύπο του τοπικού πίνακα χωρίς το πλήθος των στοιχείων του.

Άρα στον μεταγλωττιστή στέλνουμε ουσιαστικά το πρώτο byte και τον αριθμό bytes των στοιχείων του δηλαδή το μέγεθός του.

## #10.14 Δήλωση – κλήση – ορισμός με πίνακα

```
τύπος func_name (τύπος Arr_name[]);
```

**Δήλωση** συνάρτησης με πίνακα ως παράμετρο:

Τύπος συνάρτησης

Τύπος πίνακα

**Κλήση** συνάρτησης με πίνακα ως παράμετρο:

```
func_name (Array);
```

**Ορισμός** συνάρτησης με πίνακα ως παράμετρο:

```
τύπος func_name (τύπος Arr_name[])  
{  
    Προταση_1;  
}
```

Όνομα πίνακα της main() που  
στέλνουμε στη συνάρτηση



## #10.15 Συναρτήσεις με αναδρομή και επανάληψη

**Αναδρομικές** λέγονται οι συναρτήσεις που καλούν τον εαυτό τους.

Χρησιμοποιούνται σε επίλυση πολύπλοκων προβλημάτων.

Λόγω του ότι το πρόγραμμα στο οποίο χρησιμοποιούμε αναδρομικές συναρτήσεις γίνεται αργό, καλό θα είναι είτε να χρησιμοποιούμε με μέτρο την αναδρομή ή να χρησιμοποιούμε συναρτήσεις με **επανάληψη** (loop).

Οι συναρτήσεις με επανάληψη καλούνται να λύσουν το πρόβλημα με την χρήση δομών επανάληψης (π.χ. for).

## #10.16 Υπερφόρτωση συναρτήσεων

Πολλές φορές πρέπει να χρησιμοποιήσουμε πολλές διαφορετικές συναρτήσεις που κάνουν παρόμοιες (συναφείς) δουλειές, αλλά έχουν διαφορετικά ονόματα που πρέπει να θυμόμαστε για να τις χρησιμοποιούμε μέσα στο πρόγραμμα.

Για να κάνουμε την ζωή μας πιο εύκολη, χρησιμοποιούμε το ίδιο όνομα για όλες τις όμοιες (συναφείς) συναρτήσεις, με τα ορίσματα τις κάθε μιας ξεχωριστά και στην δήλωση και στην κλήση αλλά και στον ορισμό.

Αυτή η διαδικασία στον αντικειμενοστραφή προγραμματισμό ονομάζεται **υπερφόρτωση συναρτήσεων**.

## #10.17 Συναρτήσεις **INLINE**

Όταν ορίσουμε μία συνάρτηση ***inline*** τότε ο *compiler* αντιγράφει τις εντολές της συνάρτησης στην καλούσα συνάρτηση.

Με αυτό τον τρόπο το πρόγραμμα γίνεται πιο γρήγορο στις περιπτώσεις που πρέπει να κληθεί μία συνάρτηση πολλές φορές μέσα από μία άλλη.

Είναι σαν να έχουμε γράψει τις εντολές της συνάρτησης μέσα στην καλούσα συνάρτηση.

Αυτό το κάνουμε όταν η συνάρτηση που καλούμε έχει συνήθως μία εντολή.

Δομημένος Προγραμματισμός

**#11**

**ΔΟΜΕΣ**

## #11.1 Τι είναι δομή

**Δομή** είναι ένα σύνολο από μεταβλητές, την οποία και καλούμε με ένα όνομα (πχ. πελάτες). Η δομή μπορεί να ονομαστεί και νέος τύπος.

Τα μέλη της δομής είναι:

- βασικοί τύποι,
- πίνακες
- ή και άλλες δομές.

Οι δομές είναι σαν τους πίνακες στις βάσεις δεδομένων όπου οι στήλες του πίνακα είναι οι μεταβλητές της δομής.

Από τις δομές παράγονται οι μεταβλητές της δομής (πχ. πελάτης\_1, πελάτης\_2 κλπ).

## #11.2 Ορισμός της δομής

Για τον ορισμό μιας δομής χρησιμοποιούμε την λέξη ***struct***, κατόπιν το όνομα της δομής και μέσα σε ένα ζεύγος αγκίστρων τα μέλη της δομής.

Ο ορισμός κλείνει με την δήλωση των μεταβλητών της δομής και το τερματικό εντολής ;

```
struct όνομα_δομής
{
    τύπος μέλος_1;
    τύπος μέλος_2;
    τύπος μέλος_3;
    .....
} [μεταβλητές_δομής] ;
```

## #11.3 Αρχικοποίηση των μεταβλητών της δομής

Μπορούμε να κάνουμε αρχικοποίηση των μεταβλητών της δομής κατά τη δήλωσή της.

```
struct όνομα_δομής
{
    τύπος μέλος_1;
    τύπος μέλος_2;
    τύπος μέλος_3;
    .....
}μεταβλητή_1(τιμή_1, τιμή_2, τιμή_3, .....);
```

Σε ένα μέλος μιας δομής αναφερόμαστε με το όνομα μεταβλητής, τον τελεστή τελεία και το όνομα του μέλους της δομής.

### #11.4.1 Παράδειγμα

Θέλουμε να φτιάξουμε μια δομή για να καταχωρούμε τα στοιχεία των φοιτητών σε ένα μάθημα (κωδικός, όνομα, επώνυμο, βαθμός).

Ο κώδικας θα είναι ο πιο κάτω:

```
struct foithtes
{
    int foit_code;
    char foit_name[20];
    char foit_lastname[20];
    float vathmos;
};
```



## #11.4.2 Παράδειγμα

Εάν θέλουμε να αρχικοποιήσουμε έναν φοιτητή (π.χ τον foithth1)κατά τον ορισμό της δομής θα γράψουμε:

```
struct foithtes
{
    int foit_code;
    char foit_name[20];
    char foit_lastname[20];
    float vathmos;
} foithths1={001,'Γιώργος','Πάνου',7.5};
```

Καταχωρούμε μία ολόκληρη εγγραφή που περιέχει όλα τα στοιχεία του φοιτητή1. Όπως θα λέγαμε και στις βάσεις δεδομένων καταχωρούμε μία πλειάδα.

### #11.4.3 Παράδειγμα

Όταν θέλουμε να καταχωρήσουμε τιμές στα μέλη μιας δομής, μετά τον ορισμό της δομής, καλούμε με τον τελεστή τελεία ένα-ένα τα μέλη.

```
struct foithtes
{
    int foit_code;
    char foit_name[20];
    char foit_lastname[20];
    float vathmos;
}foithths1;
foithths1.foit_code=001;
strcpy(foithths1.foit_name='Γιώργος');
```

## #11.4.4 Παράδειγμα

Για να εισάγουμε τιμές από το πληκτρολόγιο χρησιμοποιούμε προτροπές προς τον χρήστη με την συνάρτηση *printf* και εισάγουμε με τις συναρτήσεις *scanf* και *gets*.

```
printf("dose kwdiko foithth");  
scanf("%d",&foithths1.foit_code);  
printf("dose onoma foithth");  
gets(foithths1.foit_name);
```

## #11.5 Δομές και Συναρτήσεις

Στα προηγούμενα θεωρήσαμε ότι ο ορισμός της δομής αλλά και η δηλώσεις των μεταβλητών της γίνονται μέσα στην `main()`. Μπορούμε ωστόσο να δηλώνουμε την δομή μας έξω από την `main` και να χρησιμοποιούμε τις μεταβλητές της δομής από συναρτήσεις. Να στέλνουμε δηλαδή μία μεταβλητή της δομής ως τιμή της παραμέτρου μιας συνάρτησης.

Παράδειγμα. Έστω η μεταβλητή δομής ***foithths1*** και η συνάρτηση: ***foithtes show\_foit(struct foithtes)*** τύπου *foithtes*.

Κλήση συνάρτησης με όρισμα την μεταβλητή δομής:

```
show_foit(foithths1); //klhsh synarthshs me orisma  
                      //to melos ths domhs foithths
```

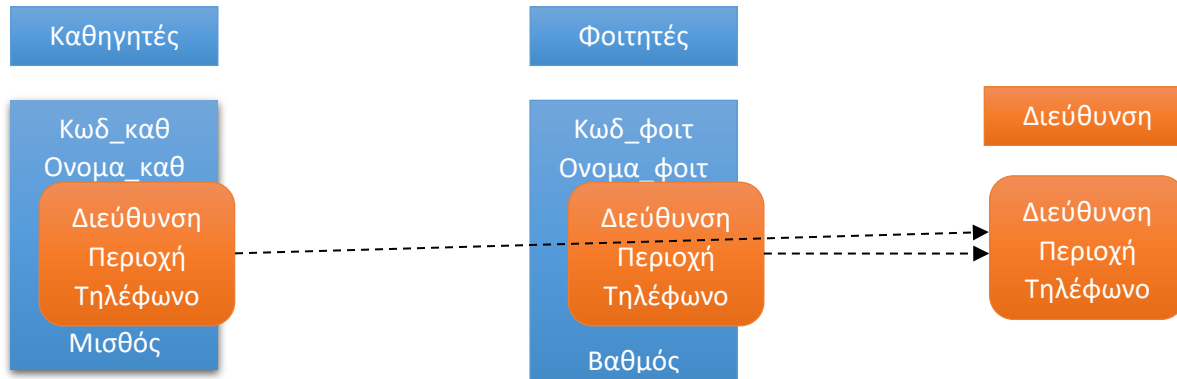
## #11.6 Ένθετες δομές

Μέσα στον ορισμό μιας δομής μπορεί να εμφανίζεται ως μέλος μια άλλη δομή. Όταν περισσότερες της μιας δομής έχουν ίδια μέλη, τότε ορίζουμε μια νέα δομή με τα κοινά μέλη και εισάγουμε πλέον την νέα δομή σαν μέλος των άλλων.

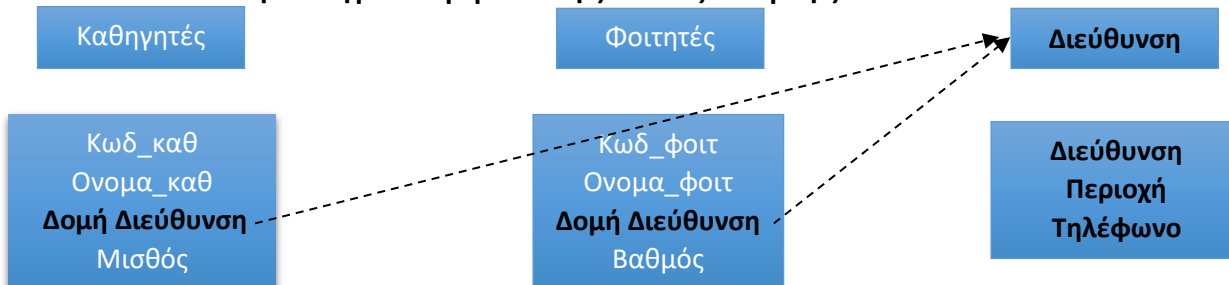
Ένα παράδειγμα είναι οι δομές καταγραφής των στοιχείων των καθηγητών και των φοιτητών. Εκτός από τα μοναδικά μέλη οι δομές αυτές έχουν και κοινά μέλη που μπορούν να γίνουν μία νέα δομή η οποία θα συμπεριλαμβάνεται στις αρχικές δομές. Αυτό θα μας βοηθήσει στο να κάνουμε οικονομία στις γραμμές κώδικα. Πιο κάτω βλέπουμε την σχηματική παράσταση του παραδείγματος.

## #11.7 Σχηματική παράσταση ένθετων δομών

- Αρχικές δομές



- Μετά την δημιουργία της νέας δομής



## #11.8 Δομές και πίνακες

Είδαμε την δήλωση μιας μεταβλητής από μία δομή αλλά και την εισαγωγή τιμών στα μέλη της.

Τι γίνεται όμως όταν θέλουμε να δηλώσουμε μαζικά περισσότερες μεταβλητές; Δηλαδή περισσότερους από έναν φοιτητές;

Τότε χρησιμοποιούμε έναν **πίνακα μεταβλητών της δομής**.

Ο πίνακας δηλώνεται με τον γνωστό τρόπο δήλωσης πινάκων, με τύπο την δομή και δείκτη τον αριθμό των μεταβλητών που θέλουμε να φτιάξουμε.

## #11.9 Παράδειγμα

Για παράδειγμα ένας πίνακας της δομής `foithtes` που είδαμε στα προηγούμενα με 20 στοιχεία δηλώνεται στην `main` ως εξής:

```
struct foithtes foithths[20];
```

Το πρώτο στοιχείο του πίνακα είναι το `foithths[0]`, στο οποίο για να καταχωρήσουμε βαθμό θα γράψουμε:

```
foithths[0].Vathmos=9.5;
```

Επίσης μπορούμε να μεταφέρουμε όλη την δομή του στοιχείου 2 στο στοιχείο 0 του πίνακα γράφοντας:

```
foithths[2]= foithths[2];
```



## #11.10 Αρχικοποίηση πίνακα

Για να αρχικοποιήσουμε μαζικά πολλές μεταβλητές ενός πίνακα ακολουθούμε την τακτική αρχικοποίησης πινάκων.

```
struct foithtes
{
    int foit_code;
    char foit_name[20];
    char foit_lastname[20];
    float vathmos;
} foithths[]={ {1,"GRHGORHS","ΙΟΑΝΝΟΥ",5.0},
{2,"MARIOS","TSOUKALAS",3.5},{3,"ΝΙΚΟΛΑΟΣ","ΠΑΥΛΟΥ",7.0},
};
```

Εδώ δεν βάζουμε δείκτη στον πίνακα. Τα στοιχεία που θα έχει καθορίζονται από τον αριθμό εγγραφών.

## #11.11 Συνώνυμα τύπων

Μπορούμε να χρησιμοποιήσουμε συνώνυμα για να δηλώνουμε τύπους αλλά και δομές. Εάν θέλουμε να φτιάξουμε ένα συνώνυμο για τον τύπο ακεραίου *int* γράφουμε:

```
typedef int akereos;
```

Μπορούμε τώρα να δηλώνουμε πίνακες ή και μεταβλητές με το συνώνυμο του τύπου.

```
akereos Array[10] ;  
akereos x=5;
```

Παραπάνω δηλώσαμε έναν πίνακα και μία μεταβλητή τύπου *aker*.

## #11.12 Συνώνυμα δομής

Μπορούμε επίσης να φτιάξουμε και συνώνυμο δομής, αφού και αυτήν την αντιμετωπίζουμε ως τύπο.

```
typedef struct όνομα_δομής
{
    μέλος1;
    μέλος2;
    .....
} συνώνυμο;
```

Μπορούμε με αυτόν τον τρόπο, να δηλώνουμε πλέον μεταβλητές του συνώνυμου της δομής, που αντιστοιχεί βεβαίως στην αρχική δομή.

### #11.13.1 Απαρίθμηση

Μπορούμε να φτιάξουμε ένα σύνολο από σταθερές δίνοντάς τους συμβολικά ονόματα και με την λέξη `enum` να ορίσουμε τον τύπο του συνόλου. Π.χ. το σύνολο των μηνών του έτους:

```
enum month {JAN=1, FEB, MAR, APR, MAI, JUN,  
JUL, AUG, SEP, OCT, NOV, DEC};
```

Εδώ φτιάχνουμε τον τύπο απαρίθμησης *month* ο οποίος θα έχει ως πρώτη τιμή το 1 που αντιστοιχεί στο συμβολικό όνομα *JAN*. Τα υπόλοιπα ονόματα θα παίρνουν την αμέσως μεγαλύτερη τιμή. Μέσα στο πρόγραμμα θα χρησιμοποιούνται οι αριθμητικές τιμές των ονομάτων.

## #11.13.2 Απαρίθμηση

Άρα μπορούμε να τυπώσουμε όλους τους μήνες του έτους χρησιμοποιώντας τον ακόλουθο βρόχο.

```
enum month mo;  
for (mo=JAN; mo<=DEC; mo++)  
    printf("%d", mo);
```

Εδώ δηλώνουμε μια μεταβλητή *mo* του τύπου *month* και αφού την αρχικοποιήσουμε μέσα στο βρόχο με την τιμή *JAN* τυπώνουμε την αντίστοιχη αριθμητική τιμή.

### #11.13.3 Απαρίθμηση

Μπορεί στη δήλωση του απαριθμητικού τύπου να έχουμε διαφορετικές αριθμήσεις. Πάντα το επόμενο όνομα θα παίρνει την επόμενη τιμή του προηγούμενου ονόματος και όχι την τιμή του πρώτου στην σειρά.

```
enum month {JAN=1, FEB, MAR, APR=10, MAI, JUN,  
JUL, AUG, SEP, OCT, NOV, DEC};
```

Εδώ τα τρία πρώτα ονόματα θα πάρουν τις τιμές JAN=1, FEB=2, MAR=3 και τα υπόλοιπα τις τιμές APR=10, MAI=11, JUN=12, JUL=13, AUG=14, SEP=15, OCT=16, NOV=17, DEC=18.

## #11.14 Ένωση

**Ένωση** είναι μια θέση μνήμης που μπορεί να περιέχει μεταβλητές διαφορετικού τύπο σε κάθε δεδομένη χρονική στιγμή. Με άλλα λόγια αναφερόμαστε πάντα σε μια μεταβλητή της ένωσης. Το μέγεθος της ένωσης είναι το μέγεθος του μεγαλύτερου από τους τύπος κάθε φορά. Ορίζεται όπως η δομή αλλά με την λέξη **union**. Π.χ η παρακάτω ένωση αλλά και οι τιμές των μελών:

<pre>union several {     int a;     float b;     char c; }sev;</pre>	<pre>sev.a=5; sev.b=2.9; sev.c='M' ;</pre>
--	--

Η ένωση αυτή την στιγμή έχει μόνο την τιμή της τελευταίας κατά σειρά μεταβλητής. Οι άλλες μεταβλητές έχουν τυχαίες τιμές.

## Βιβλιογραφία

- **Η γλώσσα προγραμματισμού C** δεύτερη έκδοση-Kernighan, Ritchie- Εκδόσεις Κλειδάριθμος 1998.
- **Τεχνολογία Λογισμικού** Θεωρία και Πράξη, Δεύτερη Αμερικάνικη έκδοση – Shari Lawrence Pfleeger - Εκδόσεις Κλειδάριθμος 2007.
- **Εισαγωγή στην Πληροφορική** - Σ. Τσιτμηδέλης Ε., Τικτοπούλου - Εκδόσεις Δεμερντζής Παντελής 2009.
- **Εισαγωγή στην Επιστήμη των Υπολογιστών** - Behrouz Forouzan, Εκδόσεις Κλειδάριθμος 2015.
- **Μαθαίνετε εύκολα C** – Δημήτριος Καρολίδης – Εκδόσεις Καρολίδη 2013.