ΕΙΣΑΓΩΓΗ ΣΤΟ ΔΟΜΗΜΕΝΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ

<u>Τρίτη Διάλεξη</u>
Εντολές Επιλογής και Επανάληψης

Εντολές επιλογής
Εντολή if
Η πιο απλή μορφή της if συντάσσεται ως εξής:

```
if ( συνθήκη ) {
     εντολή ;
     εντολή ;
     ...
     εντολή ;
}
```

Οι εντολές μέσα στα άγκιστρα αποτελούν το *σώμα* της if. Το νόημα της παραπάνω εντολής είναι ότι οι εντολές στο σώμα της if θα εκτελεστούν μόνο αν η συνθήκη μέσα στις παρενθέσεις είναι αληθής.
Μέσα σε μια if μπορούμε να βάλουμε και άλλες if.

Η πιο απλή περίπτωση της παραπάνω μορφής είναι όταν έχουμε μόνο μία εντολή. Τότε, δε χρειάζονται άγκιστρα:

```
if ( συνθήκη ) εντολή ;
```

Στην περίπτωση αυτή, συνηθίζεται και πάλι η εντολή να γράφεται στην επόμενη σειρά και προς τα μέσα για να ξεχωρίζει το σώμα από τη συνθήκη (αν και δεν είναι υποχρεωτικό):

```
if ( συνθήκη )
     εντολή ;
```

Εφόσον οι παραστάσεις έχουν λογική τιμή, μία παράσταση μπορεί να χρησιμοποιηθεί και ως συνθήκη (αυτό ισχύει και στις εντολές επανάληψης που θα δούμε μετά). Δηλαδή:

`if (x == 0 )` ισοδυναμεί με `if ( !x )`

και γενικότερα

`if (`*παράσταση* `== 0 )` ισοδυναμεί με `if ( !`*παράσταση* `)`

όπως επίσης:

`if (x )` ισοδυναμεί με `if ( x != 0 )`

και γενικότερα

`if (`*epxression* `)` ισοδυναμεί με `if (` *παράσταση* `!= 0 )`

ΠΡΟΣΟΧΗ. Όπως φαίνεται από τα παραπάνω, μετά από τις παρενθέσεις που περικλείουν τη συνθήκη, ΔΕΝ βάζουμε ελληνικό ερωτηματικό. Αυτό μπαίνει μόνο μετά το τέλος των εντολών που ορίζουν το σώμα της if. Ωστόσο, αν βάζαμε ελλ. ερωτηματικό αμέσως μετά από τις παρενθέσεις, αυτό θα ήταν συντακτικά σωστό, δηλαδή η παρακάτω εντολή

```
if ( συνθήκη )  ;
```

μεταγλωττίζεται και εκτελείται χωρίς πρόβλημα παρ' οτι πιθανότατα δεν είναι αυτό που θέλαμε να κάνουμε. Κι αυτό γιατί με τον τρόπο αυτό είναι σα να γράψαμε μία “κενή εντολή” με αποτέλεσμα ο υπολογιστής να καταλαβαίνει το εξής: “όταν η συνθήκη ισχύει μην κάνεις τίποτε εδώ και συνέχισε με το υπόλοιπο πρόγραμμα”.
'Ετσι, αν γράψουμε

```
if ( συνθήκη ); /* και εδώ ελληνικό ερωτηματικό */
     εντολή ;
```

είναι διαφορετικό απ' ο,τι αν γράφαμε:

```
if ( συνθήκη )
     εντολή ;
```

Στην πρώτη περίπτωση το if τελειώνει με το ερωτηματικό αμέσως μετά από τις παρενθέσεις οπότε η εντολή που ακολουθεί θα εκτελείται πάντα, ανεξαρτήτως της συνθήκης, ενώ στη δεύτερη θα εκτελείται μόνο αν η συνθήκη είναι αληθής (που μάλλον είναι και αυτό που θα θέλαμε να κάνουμε).

Αν θέλουμε να εκτελέσουμε συγκεκριμένες ενέργειες τόσο για την περίπτωση κατά την οποία ισχύει μία συνθήκη όσο και για την περίπτωση να μην ισχύει, δηλαδή να έχουμε μία δομή επιλογής με δύο σκέλη, μπορούμε να επεκτείνουμε την if με τη βοήθεια της λέξης else, ως εξής:

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

Τέλος, μπορούμε να κάνουμε δομές if που να χειρίζονται πολλές περιπτώσεις. Π.χ. η παρακάτω if χειρίζεται δύο περιπτώσεις και με την else χειρίζεται κάθε άλλο ενδεχόμενο:

```
if ( συνθήκη_1 ) {
     εντολή ;
     εντολή ;
     ...
     εντολή ;
} else if ( συνθήκη_2 ){
     εντολή ;
     εντολή ;
     ...
     εντολή ;
} else {
     εντολή ;
     εντολή ;
     ...
     εντολή ;
}
```

Σε όλες αυτές τις περιπτώσεις μπορεί στο σώμα κάθε σκέλους να υπάρχει μόνο μία εντολή, π.χ.

```
if ( συνθήκη )
     εντολή ;
else
     εντολή ;
```

Εντολή switch

Η παρακάτω εντολή	ισοδυναμεί με

`switch ( παράσταση )`
`/* ΠΡΟΣΟΧΗ! Οχι συνθήκη! */`
`{`
`  case constA :`
`  {`
`  ..........`
`  break;`
`  }`
`  case constB :`
`  {`
`  .........`
`  break;`
`  }`
`  default :`
`  {`
`  .........`
`  break;`
`  }`
`}`
	`if ( παράσταση == constA )`
	`{`
	`  .............`
	`} else if ( παράσταση == constB ) {`
	`  ...............`
	`}`
	`else {`
	`.........`
	`}`

Παρατηρούμε τη χρήση της εντολής break; η οποία εξασφαλίζει ότι μετά από κάθε επιλογή ο έλεγχος μεταβιβάζεται στην πρώτη εντολή αμέσως μετά από το τέλος της switch. Η εντολή αυτή υπάρχει αυτοτελώς και χρησιμοποιείται και σε δομές επανάληψης για να τερματίζει τον εσωτερικό βρόχο. - βλ. επόμενη ενότητα.

Τελεστής υπό συνθήκη (συνθήκηal operator):
Αυτός είναι ο μόνος τελεστής που συνδυάζει τρεις τελεστέους για να δώσει ένα αποτέλεσμα και η γενική μορφή είναι η εξής:

```
συνθήκη ? παράσταση_για_Αληθή : παράσταση_για_Ψευδή ;
```

Ωραίο παράδειγμα: το πρόγραμμα

```
if(x>0)
     printf("1\n");
else {
     if(x<0)
          printf("-1\n");
     else
          printf("0\n");
}
```

μπορεί να γραφεί και ως:

```
     printf("%d\n",(x>0) ? 1 : ((x<0) ? -1 : 0));
```

Εντολή goto:
Σύνταξη:
go to *label*;

Παράδειγμα:

```
int x;
...
start: printf("\n%d", x);
...
if ( x < 5 )
{
    x++;
    goto start;
}
```

Αυτή είναι καλή για παράκαμψη μεγάλων κομματιών του κώδικα στη φάση της ανάπτυξης κυρίως, καθώς και για έξοδο από δομές επανάληψης (βλ. μετά) όταν η break δε μπορεί να βοηθήσει με απλό τρόπο επειδή μας βγάζει μόνο από τον εσωτερικό βρόχο, αλλά καλύτερα να αποφεύγεται όσο είναι δυνατό.
Ο δομημένος προγραμματισμός με την πιο αυστηρή έννοια του όρου ***αποκλείει*** την εντολή αυτή επειδή δεν υπάρχει κάτι που μπορεί να γίνει με τη goto και να μη μπορεί να γίνει με "δομημένες" εντολές επιλογής ή επανάληψης.

Εντολές επανάληψης.
Οι εντολές επανάληψης, που αναφέρονται και ως βρόχοι επανάληψης (loops) είναι τριών ειδών:

- Υπό συνθήκη με έλεγχο στην αρχή (while)
- Υπό συνθήκη με έλεγχο "στο τέλος" (do...while)
- Για ορισμένο αριθμό φορών (for)
-

Εντολή while
Η γενική μορφή της είναι η εξής:

```
while ( συνθήκη )  {
     εντολή;
     εντολή;
     ....
}
```

Παρατηρήσεις:

- Αν αρχικά η συνθήκη δεν ισχύει, τότε μπορεί να μην εκτελεστεί ούτε μία φορά. Συνεπώς προσοχή κατά πόσον είναι αυτό που θελουμε. Αν δε θέλουμε κάτι τέτοιο, πρέπει να εξασφαλίσουμε ότι η συνθήκη ή παράσταση ελέγχου έχει αληθή τιμή πριν ξεκινήσει ο βρόχος (loop). Διαφορετικά μπορούμε να χρησιμοποιήσουμε και do ... while( ) - βλ. παρακάτω.
- Προσοχή επίσης στην αέναη επανάληψη (endless loop)! Π.χ., τι κάνει το παρακάτω loop ανάλογα με το σύμβολο της ανισότητας στην συνθήκη ελέγχου;

`x = 1;`
`while( x <= 0)`
`{`
`  ......`
`  x++;`
`}`
	`x = 1;`
	`while( x > 0)`
	`{`
	`  ......`
	`  x++;`
	`}`

Αυτό δε θα εκτελεστεί ποτέ!	Αυτό δε θα τελειώσει ποτέ! (εκτός αν έχει break , goto ή return)

Για να εκτελεστεί η while πεπερασμένο αριθμό φορών θα πρέπει μέσα στο σώμα της (ανάμεσα στα άγκιστρα δηλαδή) να υπάρχει κάποια μεταβαλλόμενη ποσότητα που θα καταστήσει τη συνθήκη ψευδή.
Παράδειγμα: υπολογισμός αθροίσματος και γινομένου ακέραιων αριθμών από 1 έως 10.

```
n = 1;
sum = 0;
prod = 1;
while (n < 11) {
     sum += n;
     prod *= n++;
}
```

Μέσα στη while, το sum αυξάνεται κατά n, το prod πολλαπλασιάζεται επί n και μετά το n αυξάνεται κατά 1 (υπενθυμίζουμε ότι το prod *= n++; μπορεί να γραφεί και ως prod = prod * n; n = n + 1;). Έτσι, όταν το n γίνει 10 θα μεταβάλλει άλλη μία φορά τα sum και prod και μετά θα πάρει την τιμή 11 οπότε θα πάψει να ισχύει η συνθήκη n < 11 και η while δε θα εκτελεστεί άλλη φορά.

Μπορούμε να γράψουμε και εντολές while με μόνο μία απλή εντολή. Τότε, δε χρειάζονται άγκιστρα:

```
while(συνθήκη) εντολή;
```

Συνήθως, όπως και με την if, σε αυτή την περίπτωση γράφουμε την εντολή από κάτω, ως εξής:

```
while(συνθήκη)
     εντολή;
```

Έτσι, στο προηγούμενο παράδειγμα, ο υπολογισμός του αθροίσματος μπορεί να γραφτεί χωριστά από αυτόν του γινομένου (αν και σε αυτό το συγκεκριμένο παράδειγμα δεν υπάρχει λόγος που να επιβάλλει κάτι τέτοιο):

```
while (n < 11) sum += n++;
while (n < 11) prod *= n++;
```

(Να προσεχτεί ότι βάλαμε τον τελεστή ++ και στις δύο περιπτώσεις για να εξασφαλίσουμε τον τερματισμό της κάθε while!).
Όπως και με την if, το ελληνικό ερωτηματικό αμέσως μετά από τις παρενθέσεις σημαίνει την εκτέλεση της "κενής εντολής" που ίσως δεν είναι αυτό που θέλουμε! Έτσι, αν γράψουμε:

```
n = 1;
sum = 0;
while (n < 11) ; /* και εδώ ελληνικό ερωτηματικό */
     sum += n++;
```

το μόνο που θα καταφέρουμε θα είναι να "κρεμάσει" το πρόγραμμα λόγω του ατέρμονου βρόχου `while (n < 11) ;` αφού το n θα είναι < 11 και δε μεταβάλλεται από το while.

Τέλος, αξίζει να σημειώσουμε ότι μέσα σε μία while μπορούμε να τοποθετήσουμε άλλες εντολές while ή ακόμη και άλλου τύπου επαναλήψεις ή εντολές επιλογής.
Για να διακόψουμε μία while μπορούμε να συνδυάσουμε μία συνθήκη με την εντολή break, αλλά αν έχουμε δύο (ή περισσότερα while) το ένα μέσα στο άλλο και τη break στο πιο εσωτερικό, ο έλεγχος δε βγαίνει τελείως από τις while αλλά πάει στο αμέσως πιο εξωτερικό while. Για να βγούμε τελείως από όλα τα ένθετα while χρειάζεται μία εντολή goto.
Η εντολή continue μπορεί να χρησιμοποιηθεί για να διακόψουμε την τρέχουσα επανάληψη σε κάποιο σημείο πριν τελειώσει (πριν εκτελεστούν όλες οι εντολές στο σώμα της while και να περάσουμε αμέσως στην επόμενη επανάληψη).

Εντολή do ... while
Η γενική μορφή της είναι

```
do
{
     εντολή;
     εντολή;
     ...
     εντολή;
}
while( συνθήκη ) ;
```

όπου πρέπει να προσέξουμε ότι στο τέλος, μετά από τη συνθήκη, μπαίνει ελληνικό ερωτηματικό. Ως προς αυτό το σημείο διαφέρει από όλες τις άλλες εντολές επιλογής και επανάληψης.
Αυτή η εντολή εκτελείται πάντα τουλάχιστον μία φορά – αντίθετα από την while.

Η do ... while προσφέρεται για περιπτώσεις όπου ζητείται είσοδος από το χρήστη και θέλουμε να επαναλαμβάνεται ένα "ερώτημα" μέχρι να εισαχθεί η σωστή απάντηση.
<u>Παράδειγμα</u>: μενού επιλογών χρήστη.

```
int choice;
...
do {
    printf("\n 1. Εισαγωγή στοιχείων ");
    printf("\n 2. Εκτύπωση στοιχείων ");
    printf("\n 3. Διόρθωση στοιχείων ");
    printf("\n 4. Διαγραφή στοιχείων ");
    printf("\n 5. Έξοδος");
    printf("\nΕπιλέξτε")
    scanf("%d", &choice);
} while (choice != 5);
```

Εδώ, ο χρήστης πρέπει να δώσει μία από τις τιμές 1 έως 5 που αντιστοιχούν σε κάποιες λειτουργίες του προγράμματος. Η εκτύπωση των επιλογών που προσφέρονται θα επαναληφθεί αν ο χρήστης δώσει άλλες τιμές και μέχρι να δώσει κάποια από 1 έως 5.

Εντολή for
Η εντολή for μας επιτρέπει να καθορίσουμε ακριβώς και εκ των προτέρων πόσες φορές θέλουμε να γίνει μία επανάληψη.
Μια από τις πιο συνηθισμένες μορφές της εντολής for είναι και η παρακάτω:

```
for (i = αρχική τιμή ; i < οριακή τιμή ; ++i)  {
     εντολή;
     εντολή;
     ....
}
```

Για παράδειγμα, ας υπολογίσουμε και πάλι το άθροισμα και το γινόμενο των ακέραιων αριθμών από 1 έως 10.

```
sum = 0;
prod = 1;
for (n = 1; n < 11; n++) {
     sum += n;
```

```
        prod *= n;
    }
```

Άλλος τρόπος θα ήταν να γράψουμε:

```
    for (n = 10; n > 0; n--) {
        sum += n;
        prod *= n;
    }
```

Γενικά, μπορούμε να κάνουμε διάφορους συνδυασμούς αυτού του είδους για να πετύχουμε το ίδιο αποτέλεσμα. Η γενική μορφή της for είναι

```
    for(Αρχική Συνθήκη ; Συνθήκη Ελέγχου; Παράσταση Μεταβολής) {
        εντολή;
        εντολή;
        ....
    }
```

Όπως και με την if και τη while, μπορούμε να γράψουμε και for με μόνο μία απλή εντολή, την οποία συνήθως γράφουμε από κάτω:

```
    for(Αρχική Συνθήκη ; Συνθήκη Ελέγχου; Παράσταση Μεταβολής) {
        εντολή;
```

Έτσι, στο προηγούμενο παράδειγμα, ο υπολογισμός του αθροίσματος μπορεί να γραφτεί χωριστά από αυτόν του γινομένου (αν και σε αυτό το συγκεκριμένο παράδειγμα δεν υπάρχει λόγος που να επιβάλλει κάτι τέτοιο):

```
    for (n = 10; n > 0; n--)
        sum += n;
    for (n = 10; n > 0; n--)
        prod *= n;
```

Όπως και με τις if και while, το ελληνικό ερωτηματικό αμέσως μετά από τις παρενθέσεις σημαίνει την εκτέλεση της "κενής εντολής" που ίσως δεν είναι αυτό που θέλουμε! Έτσι, αν γράψουμε:

```
    sum = 0;
    for (n = 10; n > 0; n--) ; /* και εδώ ελληνικό ερωτηματικό */
        sum += n;
```

το μόνο που θα καταφέρουμε θα είναι να μειωθεί το n από το 10 στο 1 και μετά να προστεθεί στο sum που έτσι, θα γίνει 1.

Πάντως, μερικοί προγραμματιστές εκμεταλλεύονται αυτή τη δυνατότητα για να πετύχουν ένα πιο συνοπτικό στυλ γραφής. Π.χ. το άθροισμα από το 1 έως το 10 μπορεί να υπολογιστεί και έτσι:

```
    for (n = 10; n > 0; sum += n--) ;
```

Τέλος, αξίζει να σημειώσουμε ότι μέσα σε μία for μπορούμε να τοποθετήσουμε άλλες εντολές for ή ακόμη και άλλου τύπου επαναλήψεις ή εντολές επιλογής.

Για να διακόψουμε μία for μπορούμε να συνδυάσουμε μία συνθήκη με την εντολή break, αλλά αν έχουμε δύο (ή περισσότερα for) το ένα μέσα στο άλλο και τη break στο πιο εσωτερικό, ο έλεγχος δε βγαίνει τελείως από τις for αλλά πάει στο αμέσως πιο εξωτερικό for. Για να βγούμε τελείως από όλα τα ένθετα for χρειάζεται μία εντολή goto.

Εντολή continue

Σύνταξη:

```
    continue;
```

Η εντολή continue μπορεί να χρησιμοποιηθεί για να διακόψουμε την τρέχουσα επανάληψη σε κάποιο σημείο πριν τελειώσει (πριν εκτελεστούν όλες οι εντολές σε μια for ή while) και να περάσουμε αμέσως στην επόμενη επανάληψη. Συνήθως συνδυάζεται με κάποια συνθήκη.