Διαδικαστικός Προγραμματισμός

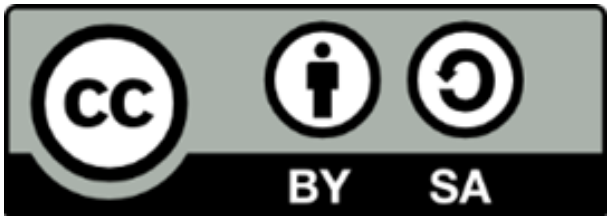
Ενότητα 3: Εντολές ελέγχου→ επανάληψη

Καθηγήτρια Μαρία Σατρατζέμη

Τμήμα Εφαρμοσμένης Πληροφορικής

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Μακεδονίας» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Σκοποί ενότητας

- Να μάθετε τις λεπτομέρειες των εντολών **while** και **for** και τις περιπτώσεις στις οποίες χρησιμοποιούμε την κάθε μια.

- Να καταλάβετε πώς να χρησιμοποιείτε τις δυνατότητες μορφοποίησης της συνάρτησης **printf**.

- Να εκτιμήσετε τη σπουδαιότητα της σχεδίασης των προγραμμάτων σας με τέτοιον τρόπο ώστε να είναι κατανοητά από άλλους προγραμματιστές.

Επίλυση προβλημάτων σε μεγαλύτερη κλίμακα

```
/*
 * File: add2.c
 * ------------
 * Αυτό το πρόγραμμα «διαβάζει» δύο αριθμούς,
 * τους προσθέτει και εμφανίζει το άθροισμά τους.
 */

#include <stdio.h>
#include "genlib.h"
#include "simpio.h"

main()
{
    int n1, n2, total;

    printf("This program adds two numbers.\n");
    printf("1st number? ");
    n1 = GetInteger();
    printf("2nd number? ");
    n2 = GetInteger();
    total = n1 + n2;
    printf("The total is %d.\n", total);
}
```

Υποθέστε ότι θέλετε να τροποποιήσετε το πρόγραμμα άθροισης 2 αριθμών για να προσθέτει 10 ή 100 ή και 1000 αριθμούς.

Θα χρησιμοποιούσατε την ίδια στρατηγική;

Δηλαδή,

Θα δηλώνατε 10 μεταβλητές

Θα διαβάζατε τις τιμές τους

θα τις προσθέτατε στο τέλος

ή μήπως δεν χρειάζεται να «θυμόμαστε» (αποθηκεύουμε σε μια μεταβλητή) τον κάθε αριθμό;

Μήπως θα μπορούσαμε να χρησιμοποιούμε μια μόνο μεταβλητή, έστω **value**, για να διαβάζουμε τους αριθμούς και να τους προσθέτουμε στην πορεία (καθώς δηλαδή τους διαβάζουμε) σε μια μεταβλητή, έστω **total**;

Άθροιση 10 αριθμών

- Δήλωση μεταβλητών

  int value, total;

  όπου:

  value: είναι η μεταβλητή όπου διαβάζεται κάθε φορά ο αριθμός που εισάγει ο χρήστης

  total: είναι η μεταβλητή στην οποία υπολογίζεται (σταδιακά) το άθροισμα.

- Αλγόριθμος

  1. Εμφάνιση προτροπής για την εισαγωγή αριθμού
  2. Διάβασμα και αποθήκευση της τιμής που εισάγει ο χρήστης στη μεταβλητή **value**
  3. Πρόσθεση της νέας τιμής στο άθροισμα

```
printf("number? ");
value = GetInteger();
total += value;
```

Πως όμως μπορούμε να εκτελέσουμε τα παραπάνω βήματα 10 φορές;

Ο βρόχος for (1)

Για την εκτέλεση μιας λειτουργίας (ομάδας εντολών) ένα συγκεκριμένο πλήθος φορών χρησιμοποιείται η εντολή ή αλλιώς ο **βρόχος for**, η οποία παριστάνει τον **ιδιωματισμό επανάληψης Ν-φορές** (repeat-N-times idiom):

```
for (i = 0; i < N; i++)
{
    εντολές προς επανάληψη
}
```

όπου ***N*** το πλήθος των επαναλήψεων.

Για το παράδειγμα μας λοιπόν θα έχουμε τον εξής κώδικα:

```
for (i = 0; i < 10; i++)
{
    printf("number? ");
    value = GetInteger();
    total += value;
}
```

Ο βρόχος for (2)

Η γενική μορφή της εντολής for είναι η εξής:

```
for (έκφραση_αρχικοποίησης; έκφραση_ελέγχου; έκφραση_ενημέρωσης)
{
      εντολές
}
```

όπου:

- έκφραση_αρχικοποίησης = μια έκφραση που αρχικοποιεί τη μεταβλητή αριθμοδείκτη (index variable), η οποία είναι μια μεταβλητή που λειτουργεί ως μετρητής και χρησιμοποιείται στη συνθήκη ελέγχου του βρόχου.

  Το γράμμα i χρησιμοποιείται ως όνομα της μεταβλητής αριθμοδείκτη για ιστορικούς λόγους.

  Η μεταβλητή αυτή πρέπει να δηλωθεί όπως και κάθε άλλη μεταβλητή.

  Συνήθως σε αυτή τη μεταβλητή δίνουμε ως αρχική τιμή το 0, εκτός και αν χρειαζόμαστε την τιμή της i στους υπολογισμούς.
- έκφραση_ελέγχου = αντιπροσωπεύει τη συνθήκη ελέγχου του βρόχου, η οποία καθορίζει τη συνέχιση ή τον τερματισμό του βρόχου
- έκφραση_ενημέρωσης = μια έκφραση που αυξάνει (ή μειώνει) την τιμή της μεταβλητής ελέγχου

Ο βρόχος for (3)

Λειτουργία/σημασία της εντολής:

Αρχικοποιείται η μεταβλητή αριθμοδείκτη, και στη συνέχεια, σε κάθε επανάληψη:

Ελέγχεται η συνθήκη ελέγχου και

αν είναι αληθής (**TRUE**):

- *εκτελούνται οι εντολές* του σώματος του βρόχου, και στη συνέχεια
- *αυξάνεται* (ή μειώνεται) *η μεταβλητή αριθμοδείκτη* σύμφωνα με την **έκφραση_ενημέρωσης**

αν είναι ψευδής (**FALSE**):

- *τερματίζεται η εκτέλεση του βρόχου*.

Ο βρόχος for (4)

Και οι 3 εκφράσεις-παραστάσεις σε ένα βρόχο for είναι προαιρετικές:

- Αν λείπει η *έκφραση-αρχικοποίησης* δεν γίνεται ανάθεση τιμής.
- Αν λείπει η *έκφραση-ελέγχου* θεωρείται ότι η συνθήκη είναι **TRUE**.
- Αν λείπει η *έκφραση-ενημέρωσης* δεν συμβαίνει καμία αλλαγή μεταξύ των κύκλων (επαναλήψεων) του βρόχου.

Έτσι, η γραμμή ελέγχου

for(;;)

είναι ισοδύναμη όσον αφορά στη λειτουργία της με την

while (TRUE)

Ο βρόχος for (5)

Όλες οι εντολές ελέγχου της C αποτελούνται από δύο μέρη:

- Τη **γραμμή ελέγχου** (control line): που είναι η 1η γραμμή μιας εντολής ελέγχου και αρχίζει με μια λέξη-κλειδί που προσδιορίζει τον τύπο της εντολής. Στην περίπτωση που εξετάζουμε, η γραμμή ελέγχου είναι:

  for (i = 0; i < N; i++)

  και καθορίζει το πλήθος των επαναλήψεων.
- Το **σώμα** (body): που αποτελείται από της εντολές που περικλείονται στα άγκιστρα.

Η γραμμή ελέγχου και το σώμα είναι εννοιολογικά ανεξάρτητα.

Παράδειγμα:

```
for (i = 0; i < 2; i++)
{
    printf("a rose is ");
}
printf("a rose.\n ");
```

a rose is a rose is a rose

Ανάθεση αρχικών τιμών

```
/*
 * Αρχείο: add10.c
 * -------------
 * Αυτό το πρόγραμμα προσθέτει μια λίστα δέκα αριθμών,
 * εμφανίζοντας στο τέλος το συνολικό άθροισμα. Αντί να διαβάζει
 * τους αριθμούς σε χωριστές μεταβλητές, το πρόγραμμα διαβάζει κάθε
 * αριθμό και τον προσθέτει στο τρέχον άθροισμα
 */

#include <stdio.h>
#include "genlib.h"
#include "simpio.h"

main()
{
    int i, value, total;

    printf("This program adds a list of ten numbers.\n");



    for (i = 0; i < 10; i++) {
        printf(" ? ");
        value = GetInteger();
        total += value;
    }
    printf("The total is %d\n", total);

}
```

Η χρήση μιας εντολής ανάθεσης προκειμένου να εξασφαλιστεί ότι μια μεταβλητή θα έχει την κατάλληλη αρχική τιμή ονομάζεται **ανάθεση αρχικής τιμής** (initialization).

Η παράλειψη της ανάθεσης αρχικών τιμών στις μεταβλητές αποτελεί μια πολύ συνηθισμένη αιτία σφάλματος.

Άγνωστο πλήθος επαναλήψεων

Όταν η επίλυση ενός προβλήματος απαιτεί την επανάληψη μιας ομάδας εντολών ένα προκαθορισμένο πλήθος φορών μπορούμε να χρησιμοποιήσουμε, όπως είδαμε, την εντολή **for**.

Τι γίνεται όμως στην περίπτωση που θέλουμε να επαναλάβουμε μια ομάδα εντολών ένα πλήθος φορών που δεν είναι προκαθορισμένο;

Για παράδειγμα, υποθέστε ότι σας έχει ζητηθεί να γράψετε ένα πρόγραμμα που να προσθέτει τα ψηφία ενός θετικού ακεραίου που δίνει ο χρήστης από το πληκτρολόγιο.

Εφόσον δεν είναι γνωστό το πλήθος των ψηφίων είναι προφανές ότι δεν μπορούμε να χρησιμοποιήσουμε την εντολή **for**.

Σε τέτοιες περιπτώσεις χρησιμοποιούμε την εντολή **while**.

Ο βρόχος while (1)

Η γενική μορφή της *εντολής* ή αλλιώς του **βρόχου while** είναι η εξής:

```
while (συνθήκη)
{
    εντολές
}
```

Λειτουργία/σημασία της εντολής:

Όσο η συνθήκη ελέγχου αποτιμάται σε TRUE οι εντολές που υπάρχουν στο σώμα της while εκτελούνται κατ' επανάληψη.

Όταν η συνθήκη ελέγχου αποτιμηθεί σε FALSE τερματίζεται η εκτέλεση του while - χωρίς να εκτελεστούν οι εντολές που υπάρχουν στο σώμα της – και η ροή εκτέλεσης του προγράμματος μεταβαίνει στην εντολή που ακολουθεί αμέσως μετά από τη while.

Ο βρόχος while (2)

Παρατηρήσεις:

- Στην επαναληπτική δομή ***while*** πρώτα γίνεται ο έλεγχος της συνθήκης (αποτίμηση της συνθήκης) και στη συνέχεια, εφόσον η συνθήκη είναι αληθής (αποτιμηθεί σε **TRUE**), εκτελούνται οι εντολές που υπάρχουν στο σώμα της. Επομένως, είναι πιθανό οι εντολές μιας δομής ***while*** να μην εκτελεστούν καμία φορά:

```
limit = 10;
while (limit < 10)
{
  εντολές
}
```

- Στις ***εντολές*** που υπάρχουν στο σώμα της ***while*** πρέπει να υπάρχει κάποια εντολή που επηρεάζει κατάλληλα την έκφραση, έτσι ώστε κάποια στιγμή αυτή να αποτιμηθεί σε **FALSE**. Διαφορετικά, η εκτέλεση των εντολών της ***while*** θα επαναλαμβάνεται επ' άπειρον, ή όπως συνηθίζεται να λέμε θα έχουμε έναν **ατέρμονα βρόχο** (**infinite loop**):

```
limit = 2;
while (limit > 1)
{
  printf("%d", limit);
  limit = limit + 2;
}
```

Χρήση του βρόχου while (1)

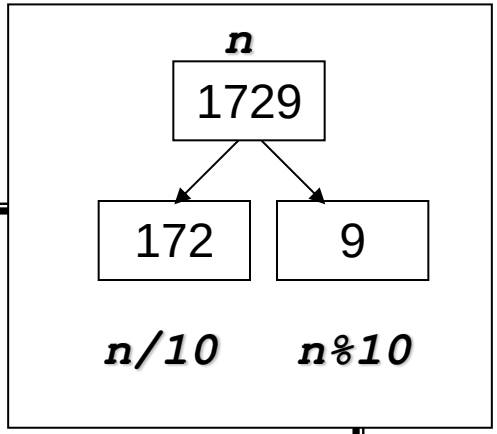
Πρόβλημα: να γράψετε ένα πρόγραμμα που να προσθέτει τα ψηφία ενός θετικού ακεραίου που δίνει ο χρήστης από το πληκτρολόγιο.

- Για παράδειγμα, αν ο χρήστης δώσει τον αριθμό **1729** τότε το αποτέλεσμα θα είναι **19** (**1+7+2+9**).
- Για το πρόβλημα αυτό μπορούμε να χρησιμοποιήσουμε τη στρατηγική που εφαρμόσαμε για την πρόσθεση των 10 αριθμών:

```
main()
{
    int n, dsum;
    printf("This program sums the digits in an integer.\n");
    printf("Enter a positive integer: ");
    n = GetInteger();
    dsum = 0;
    Για κάθε ψηφίο του αριθμού, πρόσθεσε αυτό το ψηφίο στην
   dsum
    printf("The sum of the digits is %d\n", dsum);}
}
```

Το πρόβλημα πλέον έγκειται στον προσδιορισμό ενός τρόπου με τον οποίο μπορούμε να διαχωρίσουμε ένα αριθμό στα ψηφία που τον απαρτίζουν.

Χρήση του βρόχου while (2)




```
main()
{
    int n, dsum;
    printf("This program sums the digits in an integer.\n");
    printf("Enter a positive integer: ");
    n = GetInteger();
    dsum = 0;
    while (n > 0)
    {
        dsum += n % 10;
        n /= 10;
    }
    printf("The sum of the digits is %d\n", dsum);
}
```

Επίλυση του προβλήματος του «μισού βρόχου» (1)

Πρόβλημα: να γράψετε ένα πρόγραμμα που θα προσθέτει ένα άγνωστο πλήθος θετικών ακέραιων που δίνει ο χρήστης από το πληκτρολόγιο. Το τέλος της εισόδου σηματοδοτείται με την καταχώρηση του 0.

Το πρόβλημα αυτό είναι ειδική περίπτωση μιας κατηγορίας προβλημάτων, στα οποία η συνέχιση της επανάληψης βασίζεται σε μια τιμή-φρουρό. Η δομή ενός βρόχου που βασίζεται σε μια **τιμή-φρουρό** συνίσταται στην επανάληψη των ακόλουθων βημάτων:

1. Διάβασε μια τιμή.
2. Αν η τιμή είναι ίση με το φρουρό, βγες από το βρόχο.
3. Εκτέλεσε τις εντολές επεξεργασίας που απαιτούνται για τη συγκεκριμένη τιμή.

Η παραπάνω περίπτωση όπου ένας βρόχος περιέχει ορισμένες λειτουργίες που πρέπει να εκτελεστούν πριν από τον έλεγχο για τερματισμό του βρόχου αποκαλείται από τους προγραμματιστές **πρόβλημα του «μισού βρόχου»** (loop-and-a-half-problem).

Επίλυση του προβλήματος του «μισού βρόχου» (2)

Ένα πρόβλημα «μισού βρόχου» μπορεί να λυθεί με ένα από τους παρακάτω τρόπους:

```
while (TRUE)
{
    προτροπή χρήστη και ανάγνωση τιμής
  if (τιμή == φρουρός) break;
  επεξεργασία τιμής δεδομένων
}
```

```
προτροπή χρήστη και ανάγνωση τιμής
while (τιμή != φρουρός)
{
    επεξεργασία τιμής δεδομένων προτροπή χρήστη
    και ανάγνωση τιμής
}
```

Η 2η στρατηγική έχει δύο μειονεκτήματα:

- Η σειρά των λειτουργιών μέσα στο βρόχο δεν είναι η αναμενόμενη/συνηθισμένη.
- Απαιτούνται δύο αντίγραφα των εντολών που διαβάζουν ένα αριθμό.

Άθροισμα αγνώστου πλήθους αριθμών

Η λύση στο πρόβλημα άθροισης των θετικών ακεραίων είναι η εξής:

```
main()
{
   int value, total;

   printf("This program adds a list of
    numbers.\n");
   printf("Signal end of list with a 0.\n");
   total = 0;
   while (TRUE) {
        printf(" ? ");
        value = GetInteger();
        if (value == 0) break;
        total += value;
   }
   printf("The total is %d\n", total);
}
```

```
main()
{
   int value, total;

   printf("This program adds a list of
    numbers.\n");
   printf("Signal end of list with a 0.\n");
   total = 0;
   printf(" ? ");
   value = GetInteger();
   while (value != 0) {
      total += value;
      printf(" ? ");
      value = GetInteger();
   }
   printf("The total is %d\n", total);
}
```

Ο βρόχος do (1)

Η γενική μορφή της *εντολής* ή αλλιώς του **βρόχου *do*** είναι η εξής:

```
do
{
    εντολές
}while (συνθήκη);
```

***Λειτουργία/σημασία της εντολής*:**

*Όσο η συνθήκη ελέγχου αποτιμάται σε **TRUE*** οι ***εντολές*** που υπάρχουν στο σώμα της ***do*** εκτελούνται κατ' επανάληψη.

*Όταν η συνθήκη ελέγχου αποτιμηθεί σε **FALSE*** τερματίζεται η εκτέλεση του ***do*** - χωρίς να εκτελεστούν οι εντολές που υπάρχουν στο σώμα της – και η ροή εκτέλεσης του προγράμματος μεταβαίνει στην εντολή που ακολουθεί αμέσως μετά από τη ***do***.

Ο βρόχος do (2)

Παρατηρήσεις:

- Στην επαναληπτική δομή ***do*** πρώτα εκτελούνται οι εντολές που υπάρχουν στο σώμα της και στη συνέχεια ο έλεγχος της συνθήκης (αποτίμηση της συνθήκης), και εφόσον η συνθήκη είναι αληθής (αποτιμηθεί σε **TRUE**) συνεχίζεται η εκτέλεση του σώματος των εντολών. Επομένως, οι εντολές μιας δομής do θα εκτελεσθούν τουλάχιστον μια φορά:

```
limit = 10;
do
{
  εντολές
} while (limit < 10)
```

- Στην επαναληπτική δομή ***do*** οι εντολές που υπάρχουν στο σώμα του βρόγχου εκτελούνται ΤΟΥΛΑΧΙΣΤΟΝ μια φορά ακόμη και αν η συνθήκη ελέγχου έχει από την αρχή την τιμή FALSE, καθώς ο έλεγχος της συνθήκης γίνεται στο τέλος του σώματος των εντολών, γι' αυτό θα πρέπει να είμαστε ιδιαίτερα προσεκτικοί.

```
// μια φορά
limit = 10;
do
{
      εντολές
} while (limit < 10)
```

```
// καμία φορά
limit = 10;
while (limit < 10)
{
      εντολές
}
```

Μορφοποιημένη έξοδος (1)

Μια κλήση της συνάρτησης **printf** έχει την ακόλουθη υποδειγματική μορφή:

printf("αλφαριθμητικό ελέγχου", παράσταση1, παράσταση2, …);

Κωδικοί μορφοποίησης:

- **%d**: ακέραιος δεκαδικός αριθμός
- **%ld**: ακέραιος δεκαδικός αριθμός (long)
- **%f**: αριθμός κινητής υποδιαστολής
- **%e**: εκθετικός αριθμός
- **%g**: γενική μορφή αριθμών
- **%s**: αλφαριθμητικό
- **%%**: σύμβολο ποσοστού

Μορφοποιημένη έξοδος (2)

Η συνάρτηση **printf** παρέχει σημαντικό έλεγχο σε ό,τι αφορά τη μορφοποίηση της εξόδου και σας επιτρέπει να χειρίζεστε:

- το πλάτος
- τη στοίχιση
- την ακρίβεια των δεδομένων εξόδου

με τη συμπερίληψη πρόσθετων πληροφοριών στο κωδικό μορφοποίησης.

Αυτή η πρόσθετη πληροφορία που τοποθετείται μεταξύ του συμβόλου του ποσοστού και του κωδικού μορφοποίησης μοιάζει με αριθμό κινητής υποδιαστολής αλλά στην πραγματικότητα αποτελείται από τα παρακάτω μέρη, τα οποία είναι όλα προαιρετικά:

Μορφοποιημένη έξοδος (3)

- Ένα **σύμβολο πλην**: υποδηλώνει ότι τα δεδομένα αυτού του πεδίου πρέπει να είναι ευθυγραμμισμένα αριστερά. Αν δεν υπάρχει στοιχίζονται δεξιά.

- Ένα **αριθμητικό πλάτος πεδίου**: καθορίζει το ελάχιστο πλήθος χαρακτήρων που πρέπει να χρησιμοποιηθούν για το πεδίο εξόδου.

- Μια **υποδιαστολή που ακολουθείται από μια προδιαγραφή αριθμητικής ακρίβειας**: στην περίπτωση του κωδικού **%g** η ακρίβεια καθορίζει το μέγιστο πλήθος σημαντικών ψηφίων, ενώ στην περίπτωση του κωδικού **%s** καθορίζει το μέγιστο πλήθος χαρακτήρων του αλφαριθμητικού που θέλετε να εμφανιστούν.

π.χ. **%.2f, %-14s, %-14.14s, %6d, %4.1g**

*printf**(“%-14.14s %6d %6d %4.1f%%\n”**, state, totalArea, forestArea, percent);*

Στυλ προγραμματισμού

- Να χρησιμοποιείτε σχόλια για να πληροφορείτε τους αναγνώστες σας γι’ αυτά που πρέπει να γνωρίζουν.
- Να χρησιμοποιείτε εσοχές για να επισημαίνετε τα διάφορα επίπεδα ελέγχου του προγράμματος.
- Να χρησιμοποιείτε ονόματα με νόημα.
- Να ακολουθείτε μια σύμβαση για τα ονόματα μεταβλητών που θα βοηθά τους αναγνώστες να αναγνωρίζουν τη λειτουργία τους (π.χ. τα ονόματα των μεταβλητών ξεκινούν με πεζό γράμμα, ενώ οι συναρτήσεις που αναπτύσσουμε οι ίδιοι με κεφαλαίο).
- Να χρησιμοποιείτε καθιερωμένους ιδιωματισμούς και συμβάσεις όπου χρειάζεται.
- Να αποφεύγετε τη μη αναγκαία πολυπλοκότητα.

Ο μηχανισμός define

- Για τη διευκόλυνση συγκεντρωτικών αλλαγών η C διαθέτει τον μηχανισμό **#define** για τον ορισμό σταθερών:

```
#define σύμβολο τιμή
```

- Το ***σύμβολο*** παριστάνει ένα όνομα το οποίο ακολουθεί τους ίδιους κανόνες που χρησιμοποιούνται για τις μεταβλητές.

- Η ***τιμή*** παριστάνει μια σταθερά της C.

- Κάθε φορά που το ***σύμβολο*** εμφανίζεται οπουδήποτε μέσα στο πρόγραμμα – μετά τη γραμμή ***#define*** -, αντικαθίσταται από την καθορισμένη τιμή.

Τέλος Ενότητας

Διαδικαστικός Προγραμματισμός
Τμήμα Εφαρμοσμένης Πληροφορικής