Προγραμματισμός Ι (ΗΥ120)

Διάλεξη 7:

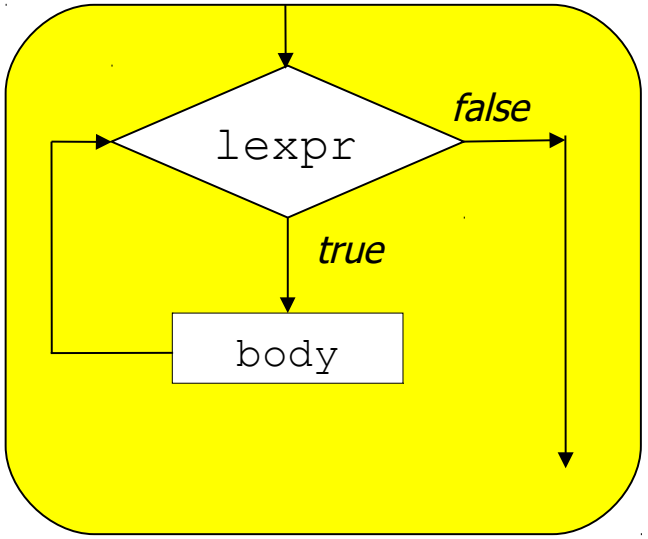
Δομές Επανάληψης - Αναγνωσιμότητα

Χρήστος Δ. Αντωνόπουλος
12/11/2018

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

Επανάληψη εκτέλεσης: `while`


```
while (<lexpr>)
  <body>
```



- Όσο η λογική συνθήκη επανάληψης `lexpr` αποτιμάται σε μια τιμή διάφορη του `0` τότε εκτελείται το `body`, διαφορετικά η εκτέλεση συνεχίζεται μετά το `while`.
- Το `body` μπορεί να μην εκτελεσθεί **καθόλου** αν την πρώτη φορά η `lexpr` αποτιμηθεί σε `0`…
  - … ή επ' άπειρο, αν η `lexpr` δεν αποτιμηθεί ποτέ σε `0` (κάτι που μπορεί να συμβεί λόγω προγραμματιστικού λάθους).

Παράδειγμα: υπολογισμός x!

- Θέλουμε το factorial `= 1*2*3*…*(x-1)*x;`
- **Πρόβλημα:** Αν το `x` λαμβάνει τιμή από το χρήστη, τότε **δεν** γνωρίζουμε την τιμή του την ώρα που γράφουμε τον κώδικα μας, συνεπώς δεν γνωρίζουμε μέχρι ποιά τιμή να συνεχίσουμε τον πολλαπλασιασμό.
- Λύση: Μετασχηματίζουμε την παραπάνω έκφραση ως επανάληψη των εντολών:

  factorial `= factorial * i; i = i + 1;`

- Η επανάληψη πρέπει να γίνει τόσες φορές όσες η τιμή της μεταβλητής `x` δηλαδή μέχρι η μεταβλητή `i` να περάσει την τιμή της

  `Όσο i <= x`

- Οι αρχικές τιμές είναι: `factorial = 1; i = 2;`


```
/* y=x! */

#include <stdio.h>

int main(int argc, char *argv[]) {

  int x, factorial, i;

  scanf("%d", &x);

  factorial = 1; i = 2;

  while (i <= x) {
    factorial = factorial * i;
    i = i+1;
  }

  printf("%d\n", factorial);

  return(0);
}
```

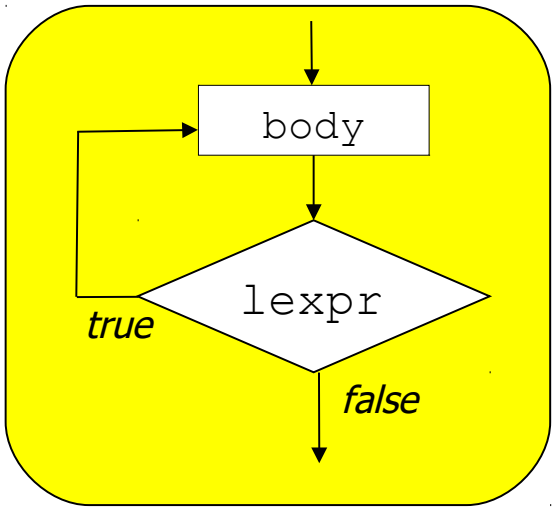
Εύρεση Μέγιστου Κοινού Διαιρέτη: Επέκταση Αλγορίθμου Ευκλείδη

- Π.χ. υπολογισμός ΜΚΔ 84, 18
  - 84 / 18: Πηλίκο 4, Υπόλοιπο 12
  - 18 / 12: Πηλίκο 1, Υπόλοιπο 6
  - 12 / 6: Πηλίκο 2, Υπόλοιπο 0
  - ΜΚΔ : 6


```
/* μέγιστος κοινός διαιρέτης x,y */
/* Επέκταση του Ευκλείδιου Αλγόριθμου */

#include <stdio.h>

int main(int argc, char *argv[]) {

  int diaireteos, diaireths, temp, ypoloipo;

  scanf("%d %d", &diaireteos, &diaireths);

  while (diaireths != 0) {
    ypoloipo = diaireteos % diaireths;
    diaireteos = diaireths;
    diaireths = ypoloipo;
  }

  printf("%d\n", diaireteos);

  return(0);
}
```

Επανάληψη εκτέλεσης: do-while


```
do
   <body>
while (<lexpr>)
```

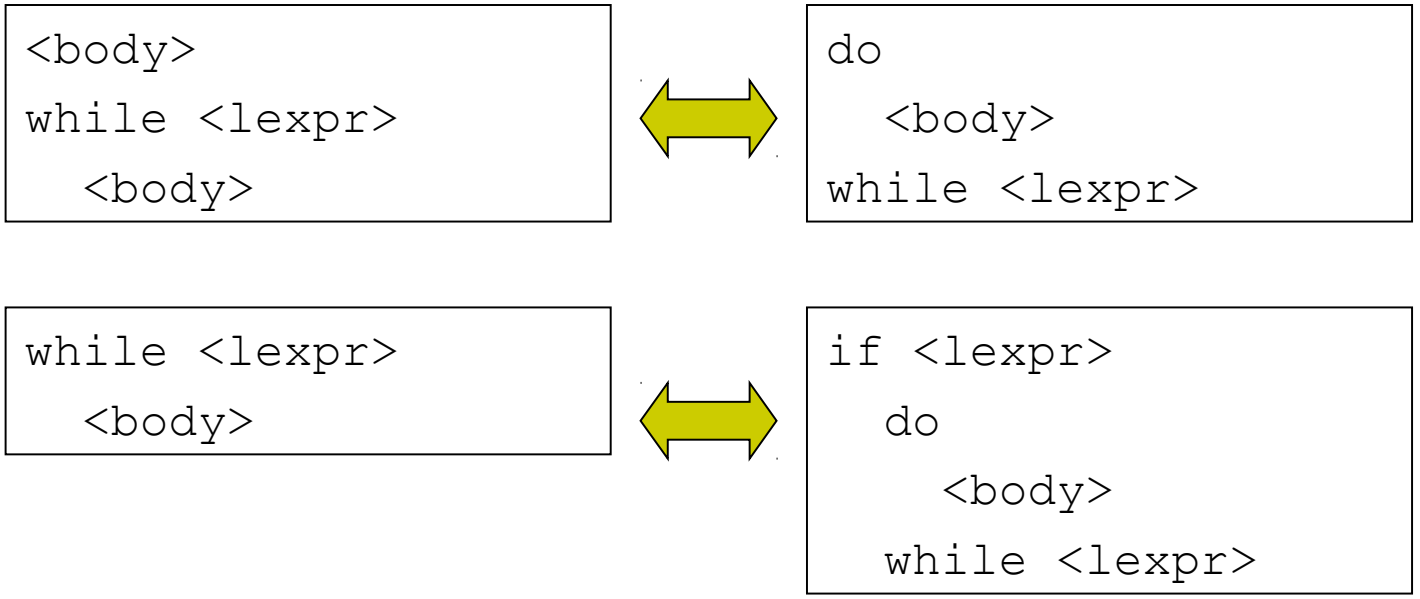


- Η λογική συνθήκη επανάληψης `lexpr` αποτιμάται αφού εκτελεσθεί πρώτα το `body`, και αν η τιμή της είναι διάφορη του `0` τότε το `body` εκτελείται ξανά.
- Το `body` θα εκτελεσθεί **τουλάχιστον μια φορά …**
  - … και ενδεχομένως επ' άπειρο αν η `lexpr` δεν αποτιμηθεί ποτέ σε `0` (συνήθως από προγραμματιστικό λάθος).


```
/* αντιγραφή χαρακτήρων από είσοδο σε έξοδο
   μέχρι να διαβαστεί ο χαρακτήρας '~' */

#include <stdio.h>

int main(int argc, char *argv[]) {

  char c;

  do {
    c = getchar();
    putchar(c);
  } while (c != '~');

  return(0);
}
```


```
/* ανάγνωση αριθμητικής τιμής σύμφωνα με την
   έκφραση number={space}{digit}space */

#include <stdio.h>

int main(int argc, char *argv[]) {

  char c;
  int res;

  do {
    c = getchar();
  } while (c == ' ');

  res = 0;
  do {
    res = res * 10 + c - '0';
    c = getchar();
  } while (c != ' ');

  printf("%d\n", res);
  return(0);
}
```

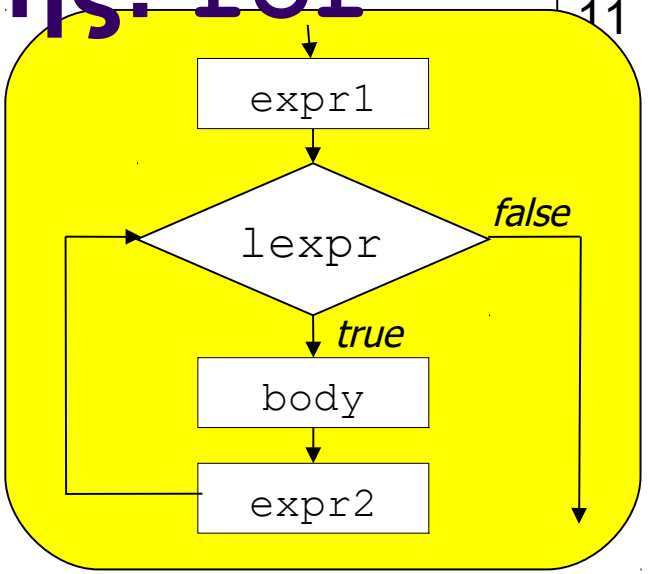
Ισοδυναμία `while` <-> `do-while`



- Για κάθε πρόγραμμα που χρησιμοποιεί `while` μπορεί πάντα να φτιαχτεί ένα ισοδύναμο πρόγραμμα που χρησιμοποιεί `do-while`, και το αντίστροφο.

Επανάληψη εκτέλεσης: `for`


```
for (<expr1>;<lexpr>;<expr2>)
  <body>
```



- Η έκφραση `expr1` αποτιμάται μια μοναδική φορά, και όσο η λογική συνθήκη επανάληψης `lexpr` αποτιμάται σε τιμή διάφορη του `0` τότε εκτελείται το `body` και μετά η έκφραση `expr2`.
  - Οι εκφράσεις `expr1` και `expr2` χρησιμοποιούνται συνήθως για την «αρχικοποίηση» και «πρόοδο» των μεταβλητών της συνθήκης επανάληψης `lexpr`.


```
/* sum=1+2+…+n */

#include <stdio.h>

int main(int argc, char *argv[]) {

  int n, sum, i;

  scanf("%d", &n);

  sum = 0;
  for (i=1; i<=n; i++) {
    sum = sum + i;
  }

  printf("%d\n", sum);

  return(0);
}
```


```
/* s=1+2+…+n */

#include <stdio.h>

int main(int argc, char *argv[]) {

  int n, sum, i;

  scanf("%d", &n);

  for (sum=0, i=1; i<=n; i++) {
    sum = sum + i;
  }

  printf("%d\n", sum);

  return(0);
}
```

```
/* s=1+2+…+n */

#include <stdio.h>

int main(int argc, char *argv[]) {

  int n, sum, i;

  scanf("%d", &n);

  for (sum=0, i=1; i<=n; sum=sum+i, i++) { }

  printf("%d\n", sum);

  return(0);
}
```


```
/* s=1+2+…+n */

#include <stdio.h>

int main(int argc, char *argv[]) {

  int n, sum, i;

  scanf("%d", &n);

  for (sum=0, i=1; i<=n; sum=sum+i++) { }

  printf("%d\n", sum);

  return(0);
}
```


```
/* πολλαπλασιασμός με πρόσθεση z=x*y, y>=0 */

#include <stdio.h>

int main(int argc, char *argv[]) {

  int x, y, res, i;

  scanf("%d %d", &x, &y);

  res = 0;
  for (i=0; i<y; i++) {
    res = res + x;
  }

  printf("%d\n", res);

  return(0);
}
```

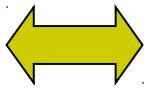

```
/* συνδυασμοί <i,j> με i:[0,n) και j:[0,m) */

#include <stdio.h>

int main(int argc, char *argv[]) {

  int n, m, i, j;

  scanf("%d %d", &n, &m);

  for (i=0; i<n; i++) {
    for (j=0; j<m; j++) {
      printf("%d,%d\n", i, j);
    }
  }

  return(0);
}
```


```
for (i=0; i<n; i++) {
  for (j=0; j<m; j++) {
    printf("%d,%d\n", i, j);
  }
}
```

```
for (i=0; i<n; i++)
  for (j=0; j<m; j++)
    printf("%d,%d\n", i, j);
```

Ισοδυναμία `while` <-> `for`


```
<expr1>
while <lexpr> {
  <body>
  <expr2>
}
```



```
for (expr1;lexpr;expr2)
  <body>
```

- Για κάθε πρόγραμμα που χρησιμοποιεί `while` μπορεί πάντα να φτιαχτεί ένα ισοδύναμο πρόγραμμα που χρησιμοποιεί `for`, και το αντίστροφο.
- Η δομή `for` επιτυγχάνει καλύτερη αναγνωσιμότητα καθώς ξεχωρίζει τις εκφράσεις «αρχικοποίησης» και «προόδου» από τον υπόλοιπο κώδικα της επανάληψης.
  - Συχνά πετυχαίνει και ταχύτερο κώδικα (ευκολότερη βελτιστοποίηση από το μεταγλωττιστή).

Η εντολές `break` και `continue`

- Κανονικά έξοδος από δομή επανάληψης: Όταν η συνθήκη ελέγχου αποτιμάται (κάποια στιγμή) σε `0`, **πριν** ή **μετά** την εκτέλεση του αντίστοιχου «σώματος» / «μπλοκ».
  - Σε κάποιες περιπτώσεις αυτό μπορεί να είναι αρκετά περιοριστικό και να κάνει τον κώδικα πολύπλοκο.
- Με την εντολή **`break`** επιτυγχάνεται «έξοδος» από οποιοδήποτε σημείο του κώδικα της επανάληψης.
- Με την εντολή **`continue`** παρακάμπτονται οι (υπόλοιπες) εντολές του κώδικα της επανάληψης, χωρίς έξοδο από την επανάληψη.
  - Στη δομή `for` το `continue` **δεν** παρακάμπτει την έκφραση «προόδου».


```
/* εκτύπωση ζυγών αριθμών στο διάστημα [1..n) */

#include <stdio.h>

int main(int argc, char *argv[]) {

  int n, i;

  scanf("%d", &n);

  for (i=1; i<n; i++) {
    if (i%2 != 0)
      continue;
    printf("%d ", i);
  }

  printf("\n");
  return(0);
}
```


```
/* ανάγνωση και πρόσθεση δύο θετικών αριθμών,
   μέχρι να δοθεί μια τιμή μικρότερη-ίση 0  */

#include <stdio.h>

int main(int argc, char *argv[]) {

  int a, b;

  while (1) {

    printf("enter 2 ints > 0 or 0 to stop: ");
    scanf("%d",&a);
    scanf("%d",&b);

    if ((a <= 0) || (b <= 0))
        break;
    printf("%d plus %d is %d\n", a, b, a+b);
  }

  return(0);
}
```

Η εντολή goto

- `goto <label>` : η εκτέλεση συνεχίζεται από το σημείο με την ετικέτα `<label>`.
- Η `goto` δίνει μεγάλη ευελιξία. Επιτρέπει τη μεταφορά του ελέγχου σε οποιοδήποτε σημείο του προγράμματος, με άμεση έξοδο «μέσα από» πολλά επίπεδα επανάληψης.
  - Με πρόχειρη χρήση της `goto` μπορεί να δημιουργηθούν δυσνόητα προγράμματα. Χρειάζεται **προσοχή!**
  - Η `goto` χρησιμοποιείται ως τελευταία λύση, όταν όλοι οι υπόλοιποι συνδυασμοί δομών και εντολών ελέγχου κάνουν τον κώδικα λιγότερο ευανάγνωστο.

```
/* αντιπαράδειγμα */
…
get:    c = getchar();
        goto check1;
cont1:  goto check2;
cont2:  putchar(c);
        goto get;
…
check1: if (c == '\n') {
          goto theend;
        } else {
          goto cont1;
        }
…
check2: if (( c>= 'a') && (c <= 'z'))
          c = c - ('a' - 'A');
        goto cont2;
theend: putchar('\n');
```


```
…
do {
  c = getchar();
  if ((c >= 'a') && (c <= 'z')) { c = c - ('a' - 'A'); }
  putchar(c);
} while (c != '\n');
```

```
/* και μια πιο ενδεδειγμένη χρήση */

while (…) {
  …
  for (…) {
    …
      do {
        …
        if (…) { goto abort; }
        …
      } while (…);
    …
  }
  …
}

abort: …
```

Δομές επανάληψης χωρίς σώμα

- Στις δομές ελέγχου `while`, `do-while` και `for` μπορεί να μην χρειάζεται να βάλουμε κάποιο σώμα εντολών
  - Πως και γιατί;
- Η C **δεν** υποστηρίζει την **απουσία** σώματος.
  - Σε αυτή την περίπτωση έχουμε την επιλογή ανάμεσα στην χρήση
    - της «κενής» εντολής `;` (που δεν κάνει τίποτα)
    - του «άδειου» σώματος εντολών `{}` (που δεν περιέχει καμία εντολή)
  - Το αποτέλεσμα είναι το ίδιο (δεν εκτελείται τίποτα).

```
/* υπολογισμός s=1+2+…N */

int i, s;

for (i=1, s=0; i<=N; s=s+i++) {}
```

το «άδειο» σώμα χωρίς εντολές

```
/* υπολογισμός s=1+2+…N */

int i, s;

for (i=1, s=0; i<=N; s=s+i++) ;
```

η «κενή» εντολή που δεν κάνει τίποτα

Γιατί τόσες δομές ελέγχου;

- Κάθε δομή ελέγχου έχει τα πλεονεκτήματα της, κυρίως όσον αφορά την αναγνωσιμότητα του κώδικα.
- Μερικές δομές διευκολύνουν τον μεταφραστή στην παραγωγή καλύτερου κώδικα μηχανής.
- Πολλές φορές η επιλογή γίνεται με βάση το προσωπικό στυλ του καθενός –φυσικά, διαφορετικοί άνθρωποι έχουν και διαφορετικά γούστα ...
- Πρωταρχικός στόχος για εσάς: **αναγνωσιμότητα**!

- Η όποια «βελτιστοποίηση» του κώδικα γίνεται **αφού** σιγουρευτούμε ότι το πρόγραμμα είναι **σωστό** …
  - … και **αφού** γίνουν κατάλληλες μετρήσεις που θα δείξουν το σημείο όπου χρειάζεται κάτι τέτοιο.

Προϋποθέσεις Αναγνωσιμότητας

- Διάταξη
  - Στοίχιση
  - Κενά
  - Κενές γραμμές
- Ονομασία αναγνωριστικών
- Σχόλια
- Χρήση σταθερών
- Μη χρήση καθολικών μεταβλητών

Αναγνωσιμότητα

- Μην προσπαθείτε να φανείτε υπερβολικά (και προκαταβολικά) «έξυπνοι» όταν γράφετε κώδικα.
- Να φροντίζετε να γράφετε ένα πρόγραμμα με πρώτο κριτήριο την **αναγνωσιμότητα** του
  - Δυστυχώς είναι εύκολο να γράψει κανείς ακατανόητο κώδικα.
- Βάζετε **σχόλια** στον κώδικα, και **όταν** χρειάζεται.
  - Τυπικές περιπτώσεις όπου ένα σχόλιο βοηθάει:
    - Περιγραφή λειτουργικότητας σε ψηλό επίπεδο
    - Τεκμηρίωση «περίεργου» κώδικα με παρενέργειες
    - Χρησιμότητα μεταβλητών του προγράμματος
- Συχνά, η (σωστή) μορφοποίηση του κειμένου είναι από μόνη της η πιο χρήσιμη περιγραφή του κώδικα.

Διάταξη: Στοίχιση

- Η στοίχιση πρέπει να δείχνει τη λογική δομή του προγράμματος
- Κάθε εντολή πρέπει να είναι σε δική της γραμμή
- Το σώμα σύνθετης εντολής πρέπει να είναι ένα tab πιο μέσα από τη σύνθετη εντολή.
    - Βέλτιστο μέγεθος tab = 4 κενά

Διάταξη: Στοίχιση

- Υπάρχουν δύο μέθοδοι τοποθέτησης αγκίστρων

The One True Brace Style (1TBS)

```
for (i=0; i<10; i++) {
    printf("%d ", i);
    if (size>5)
        printf("Too big");
}
```

Allman Style

```
for (i=0; i<10; i++)
{
    printf("%d ", i);
    if (size>5)
        printf("Too big");
}
```

ΧΡΗΣΙΜΟΠΟΙΕΙΤΕ ΠΑΝΤΑ ΤΗΝ ΙΔΙΑ ΜΕΘΟΔΟ ΜΕΣΑ ΣΤΟ ΠΡΟΓΡΑΜΜΑ!

Κακή Στοίχιση

```
/* αντιπα
    ρά
δειγμα */ #include <stdio.h>
int main(int argc, char *argv[]) {

int          i;/* μεταβλητή ακεραίος i */
int s,n;/*άλλες δύο τέτοιες μεταβλητές*/
/* αρχικοποίηση μεταβλητών */
i=1; /* i γίνεται 1 */
s=0;       /* s γίνεται 0 */
scanf("%d", &n); /* διάβασε τιμή */
/* και τώρα αρχίζουμε τον υπολογισμό μας */
while (i<=n)  /* δεν έχουμε τελειώσει */  {
  s=s+i;/* αύξησε s κατά i */ i++;/* αύξησε i
κατά 1 */}
printf("%d\n", s);
/* τέλος */
```

```
/* κάπως καλύτερα */
#include <stdio.h>
int main(int argc, char *argv[]) {
        int i;    /* μεταβλητή ακεραίος i */
        int s,n;  /*άλλες δύο τέτοιες μεταβλητές*/

        /* αρχικοποίηση μεταβλητών */
        i=1;                    /* i γίνεται 1 */
        s=0;                    /* s γίνεται 0 */
        scanf("%d", &n);        /* διάβασε τιμή */

        /* και τώρα αρχίζουμε τον υπολογισμό μας */
        while (i<=n) {   /* δεν έχουμε τελειώσει */
            s=s+i;      /* αύξησε s κατά i */
            i++;        /* αύξησε i κατά 1 */
        }

        printf("%d\n", s);
        /* τέλος */
        return(0);
}
```

Κενά / Κενές Γραμμές

- Κενά:
  - Ανάμεσα σε τελεστές και τελεσταίους
  - Σε for loops
  - Ανάμεσα στις παραμέτρους συναρτήσεων
- Κενές γραμμές:
  - Ανάμεσα σε διαφορετικές συναρτήσεις
  - Για να ξεχωρίσετε τμήματα κώδικα με διαφορετικές λειτουργίες
    - π.χ. Αρχικοποίηση μεταβλητών από υπολογισμό
- Μη χρησιμοποιείτε πάνω από 80 στήλες

Ονομασία Μεταβλητών

- ΝΑΙ
  - Περιγραφικά ονόματα
- ΟΧΙ
  - υπερβολικές συντομογραφίες
  - μεταβλητές ενός χαρακτήρα
  - παρεμφερή ονόματα (ακουστικά ή σημασιολογικά)
  - χρήση `l` (el ή ένα?)
  - χρήση `O` (μηδέν ή κεφαλαίο όμικρον?)

Ονομασία Μεταβλητών

- Μεταβλητές = ουσιαστικά
- Συναρτήσεις = ρηματικές φράσεις

- Μεταβλητές, συναρτήσεις, τύποι: πεζά
- Σταθερές: κεφαλαία

Ονομασία Μεταβλητών

- Συχνά το όνομα περιέχει δύο (ή τρεις) λέξεις
- Υπάρχουν δύο μέθοδοι φορμαρίσματος:
  - Η δεύτερη (και τρίτη) λέξη αρχίζει με κεφαλαίο
    - πχ. calcSalesTax, studentName
  - Οι λέξεις χωρίζονται με underscore
    - πχ. calc_sales_tax, student_name
- Χρησιμοποιείτε πάντα την ίδια μέθοδο μέσα στο ίδιο πρόγραμμα.

Ονομασία Μεταβλητών

- Οι μετρητές συνήθως ονομάζονται i ή j ή k
  - Μοναδική εξαίρεση στον κανόνα που απαγορεύει χρήση μονού χαρακτήρα ως όνομα
  - Αν ένας μετρητής πρόκειται να χρησιμοποιηθεί πέραν του loop => περιγραφικό όνομα.
- Ονόματα τύπων αρχίζουν από t_ ή τελειώνουν σε T
- Ονόματα δεικτών αρχίζουν από p_ ή τελειώνουν σε ptr

Καθολικές (global) Μεταβλητές

- Απαγορεύεται να τις χρησιμοποιείτε εκτός αν δε γίνεται διαφορετικά
  - Πάντα γίνεται διαφορετικά στα προγράμματα που θα γράψετε σε αυτό το μάθημα.

Αρχικοποίηση Σταθερών

- Χρησιμοποιείστε #define ή const για σταθερές
  - Το const είναι καλύτερο γιατί σας επιτρέπει να ελέγξετε την τιμή της μεταβλητής κατά το debugging

Σχόλια

- Περιγράφουν πάντα το σκοπό του κώδικα
- Συχνά περιέχουν επιπλέον χρήσιμες πληροφορίες που δεν είναι αμέσως προφανείς από ανάγνωση του κώδικα
- Ακολουθούν τη στοίχιση του κώδικα

- Μην το παρακάνετε...

Πολλά λόγχη άγνοια

```
/* αντιπαράδειγμα */

#include <stdio.h>

int main(int argc, char *argv[]) {

        int i;          /* μεταβλητή ακεραίος i */
        int sum,num;  /*άλλες δύο τέτοιες μεταβλητές*/

        /* αρχικοποίηση μεταβλητών */
        i = 1;          /* i γίνεται 1 */
        sum = 0;        /* sum γίνεται 0 */

        scanf("%d", &num);  /* διάβασε τιμή */

        /* αρχίζουμε τον υπολογισμό μας */
        while (i<=num) {          /* δεν έχουμε τελειώσει */
            sum = sum + i;        /* αύξησε sum κατά i */
            i++;                  /* αύξησε i κατά 1 */
        }

        printf("%d\n", sum);          /* τύπωσε αποτέλεσμα */
        /* τέλος */
        return(0);
}
```