Προγραμματισμός Ι (ECE115)

#6 εκτέλεση σε επανάληψη

Σπύρος Λάλης
lalis@uth.gr

Γιατί επανάληψη;

- Συχνά υπάρχει ανάγκη εκτέλεσης της ίδιας ή πρακτικά παρόμοιας εντολής ή ομάδας εντολών, **πολλές** φορές

Άκομψη λύση(;)

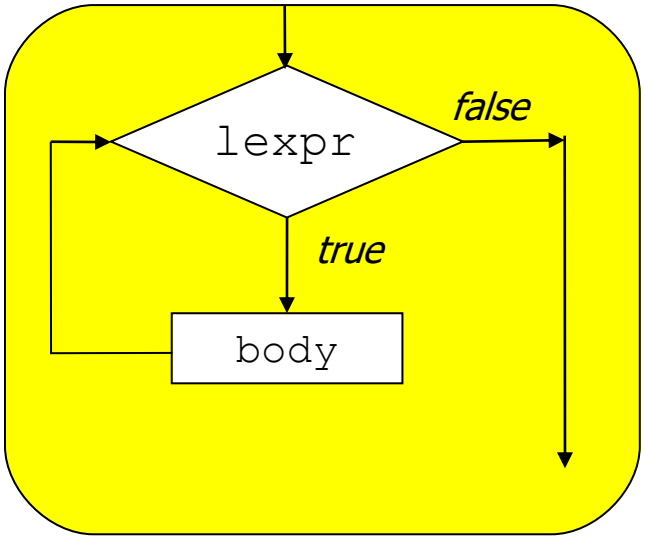
- Γράφουμε το ίδιο κομμάτι κώδικα πολλές φορές
- Τι γίνεται αν κάποιες εντολές πρέπει να επαναληφθούν 100 ή 1000 ή 1000000 φορές;
- Τι κάνουμε αν **δεν** γνωρίζουμε εκ των προτέρων το πόσες φορές πρέπει να εκτελεστεί το κομμάτι κώδικα;
- Τι γίνεται αν κάποια κομμάτια κώδικα πρέπει να εκτελούνται «για πάντα»;

Πραγματική λύση

- Δομές επανάληψης

Εκτέλεση σε επανάληψη: `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 εκτελεστεί **καθόλου**
  - αν η `lexpr` αποτιμηθεί σε 0 ήδη την πρώτη φορά
- ... ή ενδεχομένως επ' άπειρο
  - αν η `lexpr` δεν αποτιμηθεί ποτέ σε 0 (προγραμματιστικό λάθος;)

Υπολογισμός 1+2+…+100

```
#include <stdio.h>

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

  sum = 0;
  sum = sum + 1;
  sum = sum + 2;
  sum = sum + 3;
  …
  sum = sum + 100;

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

  return(0);
}
```

εντολή που επαναλαμβάνεται

αρχίζει να αθροίζει από το 1

έως και το 100

Υπολογισμός 1+2+...+100

```
#include <stdio.h>

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

   sum = 0;
   i = 1;
   while (i <= 100) {
      sum = sum + i;
      i = i + 1;
   }

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

   return(0);
}
```

μεταβλητή μετρητής

αρχίζει να αθροίζει από το 1

έως το 100

αυξάνοντας κάθε φορά κατά 1

Υπολογισμός 1+2+...+n

```
#include <stdio.h>

int main(int argc, char *argv[]) {
  int n, sum, i;

  scanf("%d", &n);

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

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

  return(0);
}
```

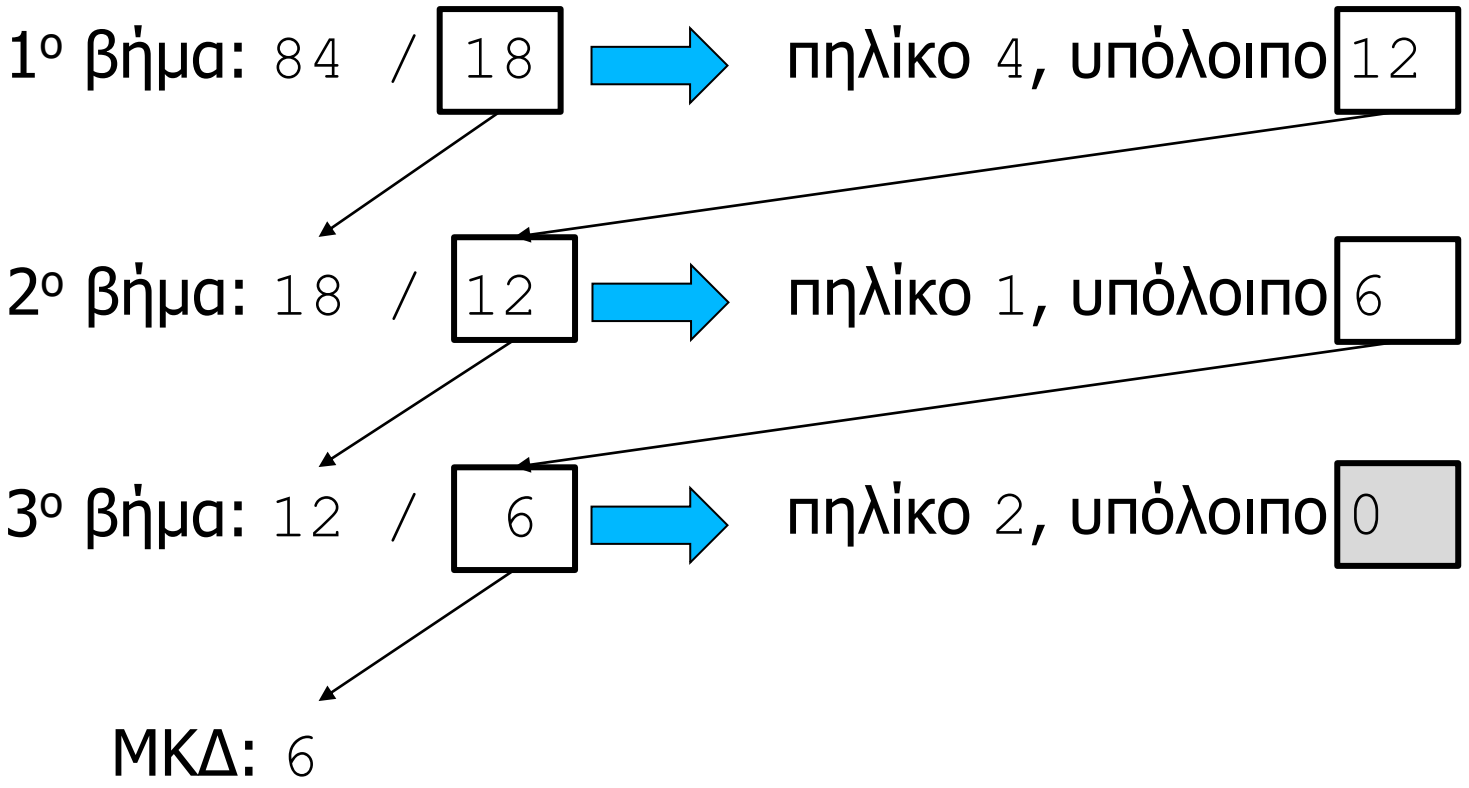
Πολλαπλασιασμός με πρόσθεση

```
/* υπολογίζει το x*y, για y>=0 */

#include <stdio.h>

int main(int argc, char *argv[]) {
  int x, y, res;

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

  res = 0;
  while (y > 0) {
    res = res + x;
    y--;
  }

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

  return(0);
}
```

```c
/* n! */

#include <stdio.h>

int main(int argc, char *argv[]) {
  int n, factorial, i;

  scanf("%d", &n);

  i = 2;
  factorial = 1;
  while (i <= n) {
    factorial = factorial * i;
    i = i+1;
  }

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

  return(0);
}
```

Μέγιστος κοινός διαιρέτης

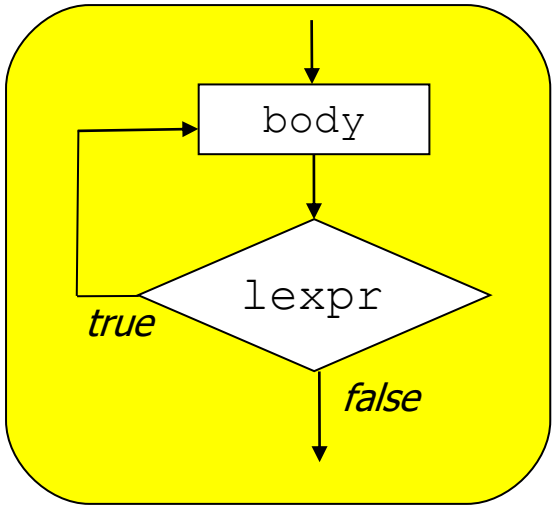
Π.χ. υπολογισμός ΜΚΔ `84, 18`



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

#include <stdio.h>

int main(int argc, char *argv[]) {
 int diaireteos, diaireths, ypoloipo;

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

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

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

  return(0);
}
```

Εκτέλεση σε επανάληψη: `do-while`

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

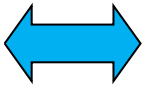


- **Πρώτα** εκτελείται το body και μετά αποτιμάται η λογική συνθήκη επανάληψης `lexpr`
  - αν η τιμή της είναι διάφορη του `0` τότε το `body` εκτελείται ξανά
- Το body θα εκτελεσθεί **τουλάχιστον** μια φορά ...
- ... και ενδεχομένως επ' άπειρο
  - αν η lexpr δεν αποτιμηθεί ποτέ σε 0 (πιθανό προγραμματιστικό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

```c
/* ανάγνωση αριθμητικής τιμής παραλείποντας
όλους τους χαρακτήρες που δεν είναι ψηφία */

#include <stdio.h>

int main(int argc, char *argv[]) {
  char c;
  int val=0;

  do {
    c = getchar();
  } while ((c < '0') || (c > '9'));

  do {
    val = val*10 + c - '0';
    c = getchar();
  } while ((c >= '0') && (c <= '9'));

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

  return(0);
}
```

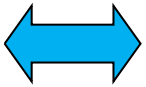
Ισοδυναμίες

```
<body>
while <lexpr>
  <body>
```



```
do
  <body>
while <lexpr>
```

```
while <lexpr>
  <body>
```



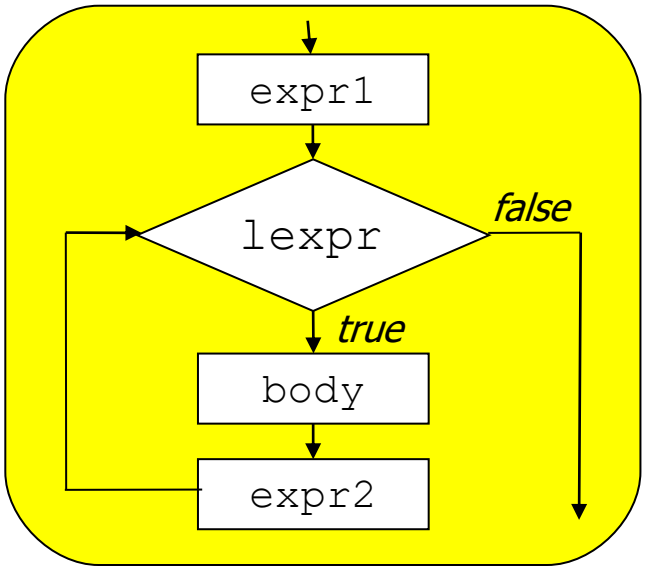
```
if <lexpr>
  do
    <body>
  while <lexpr>
```

κάθε `while` μπορεί να μετασχηματιστεί σε ένα **ισοδύναμο** `do-while`, και το αντίστροφο


while (not edge) {
run();
}
do {
run();
} while (not edge);

Εκτέλεση σε επανάληψη: `for`

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



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

Υπολογισμός 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);
}
```

Υπολογισμός 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);
}
```

Επανάληψη χωρίς σώμα

- Ειδικά στην δομή ελέγχου `for` μπορεί να **μην** χρειάζεται (πάντα) να βάλουμε σώμα εντολών
  - οι εντολές του «κανονικού» σώματος μπορεί να ενσωματωθούν στις εκφράσεις ελέγχου / προόδου

- Η C **δεν** υποστηρίζει την απουσία σώματος

- Υπάρχει η επιλογή ανάμεσα στην χρήση
  - της «κενής» εντολής `;` (που δεν κάνει τίποτα)
  - του «άδειου» σώματος `{}` (που δεν περιέχει καμία εντολή)

- Το αποτέλεσμα είναι το ίδιο (δεν εκτελείται τίποτα)

Υπολογισμός 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);
}
```

το «άδειο» σώμα χωρίς εντολές

Υπολογισμός 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);
}
```

η «κενή» εντολή
που δεν κάνει τίποτα

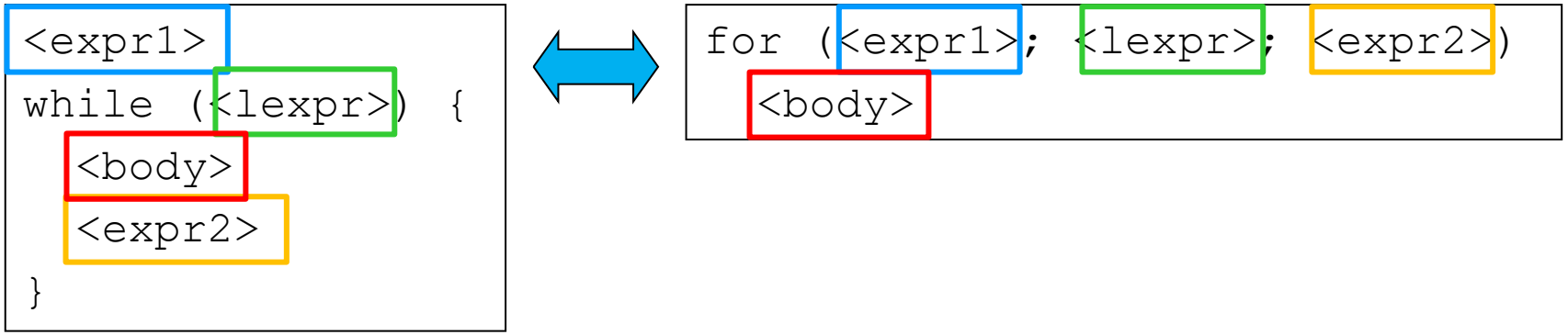
Υπολογισμός 1+2+...+n

```
#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);
}
```

```
/* υπολογισμός x*y, για y>=0 */

#include <stdio.h>

int main(int argc, char *argv[]) {
  int x, y, res, i;

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

  res  = 0;
  for ( ; y>0; y--) {
    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)", i, j);
    }
    printf("\n");
  }

  return(0);
}
```

Ισοδυναμίες



κάθε `while` μπορεί να μετασχηματιστεί σε ένα **ισοδύναμο** `for`, και το αντίστροφο

η δομή `for` μπορεί να επιτύχει καλύτερη αναγνωσιμότητα: ξεχωρίζει με ρητό τρόπο, **στο συντακτικό επίπεδο**, τις εκφράσεις «αρχικοποίησης» και «προόδου» από τον υπόλοιπο κώδικα της επανάληψης – συχνά οδηγεί και σε ταχύτερο κώδικα μηχανής (ευκολότερη βελτιστοποίηση από το μεταγλωττιστή)

Γιατί τόσες πολλές δομές επανάληψης;

- Κάθε δομή ελέγχου έχει τα πλεονεκτήματα της, κυρίως όσον αφορά την αναγνωσιμότητα του κώδικα
  - εξαρτάται από την λογική που θέλουμε να υλοποιήσουμε
- Μερικές δομές διευκολύνουν τον μεταφραστή στην παραγωγή καλύτερου κώδικα μηχανής
- Συχνά η επιλογή γίνεται με βάση το προσωπικό στυλ του καθενός

- 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 πραγματικά χρειάζεται κάτι τέτοιο

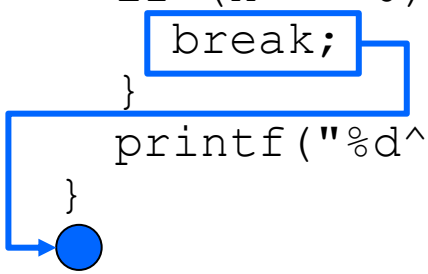
εντολές αλμάτων σε δομές επανάληψης

Οι εντολές `break` και `continue`

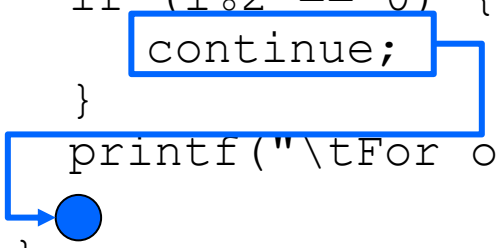
- Κανονική έξοδος από μια δομή επανάληψης γίνεται όταν η λογική συνθήκη ελέγχου αποτιμάται σε `0`
- Αυτό μπορεί να είναι περιοριστικό
  - σε κάποιες περιπτώσεις μπορεί να κάνει τον κώδικα πιο πολύπλοκο και δυσανάγνωστο

- Με `break` επιτυγχάνεται **άμεση** έξοδος από **οποιοδήποτε** σημείο του κώδικα της επανάληψης
- Με `continue` **παρακάμπτονται** οι (υπόλοιπες) εντολές του κώδικα του σώματος της επανάληψης, αλλά **χωρίς** να γίνεται έξοδος από την επανάληψη
  - στη `for` **δεν** παρακάμπτεται η έκφραση «προόδου»

```c
/* ανάγνωση ενός αριθμού και εκτύπωση του τετραγώνου,
   μέχρι να δοθεί μια τιμή ίση με 0  */

#include <stdio.h>

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

  while (1) {

    x = 0;
    printf("Enter int value or 0 to stop: ");
    scanf("%d", &x);

    if (x == 0) {
      break;
    }
    printf("%d^2 is %d\n", x, x*x);
  }

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



```
/* εκτύπωση τετραγώνου και κύβου 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++) {

    printf("Counter: %d\n", i);
    printf("\tFor every value: %d^2 is %d\n", i, i*i);
    if (i%2 == 0) {
       continue;
    }
    printf("\tFor odd values: %d^3 is %d\n", i, i*i*i);
  }

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



Η εντολή `goto`

- `goto <label>` : γίνεται **άλμα** και η εκτέλεση συνεχίζεται από το σημείο με την ετικέτα `<label>`
- Η `goto` δίνει μεγάλη ευελιξία
  - επιτρέπει τη μεταφορά του ελέγχου έξω από κάθε δομή και σε **οποιοδήποτε** σημείο του προγράμματος

- **Προσοχή:** πρόχειρη/κακή χρήση της `goto` συνήθως οδηγεί σε ιδιαίτερα **δυσνόητα** προγράμματα

- Η `goto` χρησιμοποιείται ως τελευταία λύση
  - όταν όλοι οι υπόλοιποι συνδυασμοί δομών και εντολών ελέγχου κάνουν τον κώδικα λιγότερο ευανάγνωστο
  - π.χ., άμεση έξοδος μέσα από πολλά επίπεδα επανάληψης

```
/* 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');
```

```
/* μια πιο ενδεδειγμένη χρήση της goto */

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

abort: …
```