Μεταβλητές, σταθερές, τύποι και δηλώσεις στην C

- Μεταβλητές
  - Συμβολικά ονόματα για θέσεις μνήμης
  - Αποθήκευση και ανάκτηση δεδομένων
  - Ανήκουν σε τύπους
- Σταθερές
  - Συγκεκριμένες τιμές
  - Ανήκουν σε τύπους
- Τύποι δεδομένων
  - Για ακεραίους
    - `short int` ή `short` (συνήθως 2 bytes)
    - `int` (συνήθως 4, σπανιότερα 2 ή 8, bytes, ανάλογα με το μέγεθος λέξης του επεξεργαστή)
    - `long int` ή `long` (συνήθως 4, μερικές φορές 8, bytes)
    - `long long int` ή `long long` (συνήθως 8 bytes)
  - Για χαρακτήρες, αλλά και ακεραίους
    - `char` (1 byte)
  - Για πραγματικούς αριθμούς (κινητής υποδιαστολής)
    - `float` (απλής ακρίβειας, συνήθως 4 bytes)
    - `double` (διπλής ακρίβειας, συνήθως 8 bytes)
    - `long double` (εκτεταμένης ακρίβειας, συνήθως 16 bytes)

- Παραδείγματα σταθερών
  - Ακέραιες σταθερές (θεωρούνται τύπου `int`):
    `237`, `-12345`, `22431L` (το `L` συμβολίζει ότι επιθυμούμε να θεωρηθεί η σταθερά τύπου `long`), `0224` (ο οκταδικός αριθμός 224), `0x1C` (ο δεκαεξαδικός αριθμός `1C`)
  - Σταθερές χαρακτήρα:
    `'A'` (ο ASCII κωδικός του χαρακτήρα `A`, δηλαδή 65)
    `'\n'` (ο ASCII κωδικός για την αλλαγή γραμμής)
    `'\t'` (ο ASCII κωδικός για τον στηλογνώμονα, tab)
    `'0'` (ο ASCII κωδικός του χαρακτήρα `0`, δηλαδή 48)
    `'\0'` (ο χαρακτήρας με ASCII κωδικό 0)
    `'\145'` (ο χαρακτήρας με ASCII κωδικό τον οκταδικό αριθμό `145`)
    `'\xA2'` (ο χαρακτήρας με ASCII κωδικό τον δεκαεξαδικό αριθμό `A2`)
  - Σταθερές κινητής υποδιαστολής (θεωρούνται τύπου `double`):
    `23.78`, `-1.24e-14` (δηλαδή $-1.24 \cdot 10^{-14}$)
- Μία ειδική κατηγορία σταθερών είναι τα αλφαριθμητικά ή συμβολοσειρές (strings). Μία συμβολοσειρά είναι μία ακολουθία από χαρακτήρες μέσα σε ", για παράδειγμα `"Hello world\n"`. Πρόκειται, ουσιαστικά, για έναν πίνακα χαρακτήρων, χωρίς τα ", με τελευταίο στοιχείο του πίνακα το `'\0'` (παρόλο που δεν σημειώνεται στη διατύπωση της συμβολοσειράς μέσα στα ").

- Με μία δήλωση ενημερώνεται ο μεταγλωττιστής για την πρόθεσή μας να χρησιμοποιήσουμε στο πρόγραμμά μας κάποια ή κάποιες μεταβλητές συγκεκριμένου τύπου, τις οποίες μπορούμε να αρχικοποιήσουμε και με σταθερές του τύπου αυτού, αν θέλουμε, και αντιστοιχεί σ΄ αυτές κατάλληλες θέσεις μνήμης. Παραδείγματα:

```
int i, j, k;
char ch;
float x, y = 2.34;
long count, cost;
double pi = 3.14159;
```

- Τα ονόματα των μεταβλητών μπορούν να περιλαμβάνουν, μέχρι 31 συνολικά, πεζούς και κεφαλαίους χαρακτήρες, αριθμούς και τον χαρακτήρα _.

- Δεν επιτρέπεται η χρήση δεσμευμένων λέξεων της C (π.χ. `if`, `float`, `for`, κλπ.) για ονόματα μεταβλητών.

- Δεν είναι καλό να ορίζουμε μεταβλητές στα προγράμματά μας που αρχίζουν από _, γιατί τέτοια ονόματα χρησιμοποιούνται για τις μεταβλητές από τις βιβλιοθήκες και ενδέχεται να δημιουργηθεί πρόβλημα.

- Παρότι είναι επιτρεπτό, δεν είναι καλό ονόματα μεταβλητών να αποτελούνται μόνο από κεφαλαίους χαρακτήρες, γιατί συνήθως δίνουμε τέτοια ονόματα στις συμβολικές σταθερές που ορίζουμε με `#define`, για να τις διαχειριστεί ο προεπεξεργαστής.

- Στη δήλωση μίας ακέραιας μεταβλητής ή μίας μεταβλητής χαρακτήρα, μπορεί να προηγηθεί ο προσδιοριστής `unsigned`, που δηλώνει ότι οι αριθμοί που φυλάσσονται στη μεταβλητή πρέπει να ερμηνευθούν σαν θετικοί (ή μηδέν).

- Αν υπάρχει στη δήλωση ο προσδιοριστής `signed`, ή δεν υπάρχει κανένας, το περιεχόμενο της μεταβλητής θεωρείται ακέραιος με πρόσημο (θετικό ή αρνητικό).

- Παραδείγματα:

```
signed char ch;
unsigned char uch;
short i;
unsigned long int k;
```

  - Στη μεταβλητή `ch` φυλάσσονται αριθμοί από -128 $(= -2^7)$ έως 127 $(= 2^7 - 1)$, ενώ στην `uch` από 0 έως 255 $(= 2^8 - 1)$
  - Στη μεταβλητή `i` φυλάσσονται αριθμοί από -32768 $(= -2^{15})$ έως 32767 $(= 2^{15} - 1)$, ενώ στην `k` από 0 έως 4294967295 $(= 2^{32} - 1)$

- Σε δήλωση μεταβλητής, με ταυτόχρονη αρχικοποίηση, μπορεί να εφαρμοσθεί ο προσδιοριστής `const`, που εξασφαλίζει ότι η τιμή της μεταβλητής δεν θα αλλάξει. Παράδειγμα:

```
const double e = 2.71828;
```

- Για μεταβλητές, σταθερές, τύπους και δηλώσεις, αναλυτικότερα στις παραγράφους §2.1 έως §2.4 του [KR]. [α΄]

[α΄] Για τη συνέχεια, ο συμβολισμός [KR] θα αναφέρεται στο βιβλίο: Brian W. Kernighan, Dennis M. Ritchie, "Η Γλώσσα Προγραμματισμού C".

Εντολές αντικατάστασης, τελεστές και παραστάσεις

- Οι εντολές σ΄ ένα πρόγραμμα C, όπως και οι δηλώσεις, τελειώνουν με ένα ελληνικό ερωτηματικό (;).
- Ένα βασικό είδος εντολής είναι η εντολή αντικατάστασης (ή κατά την πιο συνήθη ορολογία, εντολή ανάθεσης). Είναι της μορφής: ⟨μεταβλητή⟩ = ⟨παράσταση⟩
- Μία εντολή αντικατάστασης έχει σαν αποτέλεσμα τον υπολογισμό της τιμής της παράστασης στο δεξί μέλος του = και την ανάθεση της τιμής αυτής στη μεταβλητή στο αριστερό μέλος του =.
- Πολύ συχνές είναι οι αριθμητικές παραστάσεις που περιγράφουν πράξεις μεταξύ ακέραιων ή πραγματικών μεταβλητών και σταθερών, μέσω των αριθμητικών τελεστών, δηλαδή +, −, *, / και % (υπόλοιπο διαίρεσης).
- Προσοχή! Διαίρεση μεταξύ ακεραίων δίνει σαν αποτέλεσμα το πηλίκο της διαίρεσης.

- Παραστάσεις που εμπλέκουν μεταβλητές και σταθερές διαφορετικών τύπων δίνουν σαν αποτέλεσμα τιμή του "ευρύτερου" τύπου. Για παράδειγμα, κάποια αριθμητική πράξη μεταξύ `int` και `double` δίνει αποτέλεσμα `double`.
- Ανάθεση μίας τιμής ενός τύπου σε μεταβλητή άλλου τύπου έχει σαν αποτέλεσμα τη μετατροπή της τιμής στον τύπο της μεταβλητής, είτε χωρίς απώλεια πληροφορίας, είτε με απώλεια πληροφορίας, ανάλογα με το αν ο τύπος της μεταβλητής έχει τη δυνατότητα να αναπαραστήσει πλήρως την ανατιθέμενη τιμή.
- Ρητή μετατροπή τύπου μπορεί να γίνει μέσω προσαρμογής (cast), για παράδειγμα, αν η μεταβλητή `i` είναι ακέραιου τύπου, η έκφραση `(double) i` αναφέρεται στην τιμή της `i` σαν πραγματικό αριθμό διπλής ακρίβειας.

- Δύο πολύ χρήσιμοι τελεστές είναι οι τελεστές μοναδιαίας αύξησης (++) και μείωσης (--). Όταν εφαρμόζονται επάνω σε μία μεταβλητή, είτε στα αριστερά είτε στα δεξιά της, έχουν σαν αποτέλεσμα την αύξηση, ή μείωση αντίστοιχα, της τιμής της μεταβλητής κατά 1. Για παράδειγμα, η ακολουθία εντολών

```
x = 2;
y = 9;
x++;
--y;
++x;
x--;
y--;
++y;
```

  θα έχει σαν τελικό αποτέλεσμα οι μεταβλητές x και y να έχουν τελικά σαν τιμές τις 3 και 8, αντίστοιχα. Στο παράδειγμα αυτό, δεν είχε κάποια σημασία η τοποθέτηση των τελεστών αριστερά ή δεξιά της μεταβλητής.

- Εκτός από αυτόνομη εντολή, η εφαρμογή των τελεστών μοναδιαίας αύξησης σε μία μεταβλητή, μπορεί να παίξει και το ρόλο παράστασης (ή τμήματος παράστασης). Στην περίπτωση αυτή, έχει σημασία αν ο μοναδιαίος τελεστής είναι προθεματικός (αριστερά της μεταβλητής) ή μεταθεματικός (δεξιά της μεταβλητής).

- Ένας προθεματικός τελεστής μοναδιαίας αύξησης ή μείωσης σε μία παράσταση υπονοεί ότι πρώτα πρέπει να γίνει η τροποποίηση της τιμής της μεταβλητής (αύξηση ή μείωση, ανάλογα) και μετά να χρησιμοποιηθεί η τροποποιημένη τιμή για τον υπολογισμό της τιμής της παράστασης.
- Ένας μεταθεματικός τελεστής μοναδιαίας αύξησης ή μείωσης σε μία παράσταση υπονοεί ότι πρώτα πρέπει να χρησιμοποιηθεί η τρέχουσα τιμή της μεταβλητής για τον υπολογισμό της τιμής της παράστασης και μετά να γίνει η τροποποίηση της τιμής της μεταβλητής (αύξηση ή μείωση, ανάλογα).
- Παράδειγμα:

```
x = 4;
y = 7;
z = ++y;
y = z - (x++);
z = x - (--y);
```

  Τι τιμή θα έχουν οι μεταβλητές x, y και z μετά από αυτές τις εντολές; [α΄]

[α΄] 5, 3 και 2, αντίστοιχα

- Πολύ συχνά, στην C, θέλουμε να ελέγξουμε κάποια συνθήκη και ανάλογα με το αποτέλεσμα του ελέγχου (αληθές ή ψευδές) να προβούμε σε κάποια ενέργεια.
- Οι συνθήκες συνήθως είναι συγκρίσεις τιμών αριθμητικών παραστάσεων, μέσω των συσχετιστικών τελεστών (ή τελεστών σύγκρισης), που είναι οι >, <, >=, <=, == (ίσο) και != (διάφορο).
- Πιο σύνθετες συνθήκες μπορούν να κατασκευασθούν από απλούστερες μέσω των λογικών τελεστών && (ΚΑΙ), || (Ή) και ! (ΟΧΙ).
- Ουσιαστικά, οι συνθήκες δεν είναι κάτι διαφορετικό από τις παραστάσεις. Μία αληθής συνθήκη είναι μία παράσταση με τιμή ίση με 1, ενώ μία ψευδής συνθήκη θεωρείται ότι έχει τιμή 0. Αντίστοιχα, μία παράσταση με τιμή διάφορη του 0 μπορεί να παίξει τον ρόλο μίας αληθούς συνθήκης, ενώ αν η τιμή της είναι 0, είναι ισοδύναμη με μία ψευδή συνθήκη. Για παράδειγμα, γράφοντας

  ```
  if (myvar != 0) { ........
  ```

  είναι ακριβώς το ίδιο με το

  ```
  if (myvar) { ........
  ```

- Μερικές φορές, είναι πολύ χρήσιμο να κάνουμε πράξεις σε επίπεδο bit μεταξύ ακεραίων τιμών (μεταβλητών και σταθερών). Αυτό επιτυγχάνεται μέσω των τελεστών πράξεων με bit, που είναι οι:
  - `&`: ΚΑΙ
  - `|`: Ή
  - `^`: αποκλειστικό Ή
  - `<<`: ολίσθηση αριστερά
  - `>>`: ολίσθηση δεξιά
  - `~`: συμπλήρωμα ως προς ένα
- Παράδειγμα:

```
unsigned char x;
x = 178;
x = x & 074;
x = x | 0xD6;
x = x ^ 104;
x = x >> 3;
x = x << 2;
x = ~x;
```

  Ποια θα είναι η τελική τιμή της μεταβλητής `x`; [α΄]

[α΄] 179

- Κάθε διμελής τελεστής (+, −, *, /, %, &, |, ^, <<, >>) μπορεί να χρησιμοποιηθεί και με ένα σύντομο τρόπο σε μία εντολή αντικατάστασης, ως εξής:
  ⟨μεταβλητή⟩ ⟨τελεστής⟩= ⟨παράσταση⟩
  Αυτή η εντολή είναι ισοδύναμη με την:
  ⟨μεταβλητή⟩ = ⟨μεταβλητή⟩ ⟨τελεστής⟩ ⟨παράσταση⟩
  Για παράδειγμα, η εντολή

  ```
  sum += x;
  ```

  είναι ισοδύναμη με την

  ```
  sum = sum + x;
  ```

- Μερικές φορές, είναι χρήσιμες και οι παραστάσεις υπό συνθήκη, σαν την
  ⟨συνθήκη⟩ ? ⟨παράσταση⟩$_1$: ⟨παράσταση⟩$_2$
  Η τιμή αυτής της παράστασης είναι αυτή που έχει η ⟨παράσταση⟩$_1$, αν η ⟨συνθήκη⟩ είναι αληθής (δηλαδή, αν έχει τιμή διάφορη του 0, θεωρώντας την ως παράσταση και αυτή), ή αυτή που έχει η ⟨παράσταση⟩$_2$, αν η ⟨συνθήκη⟩ είναι ψευδής (δηλαδή ίση με 0). Παράδειγμα:

  ```
  z = (a > b) ? a : b;
  ```

  Με την εντολή αυτή, η τιμή της z θα γίνει ίση με τη μεγαλύτερη τιμή από τις a και b.

- Η χρήση διαφόρων τελεστών μέσα σε μία παράσταση μπορεί να προκαλέσει σύγχυση στον αναγνώστη ενός προγράμματος για το πώς ακριβώς θα πρέπει να υπολογισθεί η τιμή της παράστασης, δηλαδή για τη σειρά με την οποία θα πρέπει να εφαρμοσθούν οι τελεστές, όχι όμως και στον μεταγλωττιστή.
- Υπάρχουν συγκεκριμένες προτεραιότητες μεταξύ των τελεστών, άλλες περισσότερο και άλλες λιγότερο αναμενόμενες, οπότε η σειρά εφαρμογής των τελεστών είναι σαφής.
- Σε περίπτωση που θέλουμε να επιβάλουμε συγκεκριμένο τρόπο εφαρμογής των τελεστών, πρέπει να χρησιμοποιούμε παρενθέσεις, ακόμα και αν είμαστε βέβαιοι ότι σε κάποιες περιπτώσεις δεν είναι απαραίτητες λόγω των προτεραιοτήτων των τελεστών.
- Φυσικά, πρέπει να αποφεύγονται υπερβολές. Στην εντολή

  ```
  x = y + z*w;
  ```

  δεν χρειάζονται παρενθέσεις αν θέλουμε η παράσταση στο δεξιό μέλος να είναι η `y + (z*w)` (λόγω της αναμενόμενης μεγαλύτερης προτεραιότητας του `*` έναντι του `+`, που όντως ισχύει). Βέβαια, η παράσταση `(y+z) * w` είναι διαφορετική από την προηγούμενη.

- Κάθε εντολή αντικατάστασης μπορεί να θεωρηθεί και σαν παράσταση με τιμή αυτήν την τιμής της παράστασης που αναθέτουμε στη μεταβλητή. Έτσι, ο κορμός του προγράμματος `capitalize.c`

```
ch = getchar();
while (ch != EOF) {
  if (ch >= 'a' && ch <= 'z')
    ch = ch - ('a'-'A');
  putchar(ch);
  ch = getchar();
}
```

  θα μπορούσε να γραφεί πιο συμπυκνωμένα ως εξής:

```
while ((ch = getchar()) != EOF) {
  if (ch >= 'a' && ch <= 'z')
    ch = ch - ('a'-'A');
  putchar(ch);
}
```

  Προσέξτε, στη συμπυκνωμένη εκδοχή, τις παρενθέσεις γύρω από την εντολή αντικατάστασης `ch = getchar()`. Είναι απαραίτητες γιατί ο τελεστής `!=` έχει μεγαλύτερη προτεραιότητα από τον `=`. Αν δεν τις βάζαμε, τι θα γινόταν;

- Για εντολές αντικατάστασης, τελεστές και παραστάσεις στην C, υπάρχει εκτεταμένη ανάλυση στις παραγράφους §2.5 έως §2.12 του [KR], που συνοδεύεται από πάρα πολλά ενδιαφέροντα παραδείγματα.
- Επίσης, το Κεφάλαιο 1 από το [KR] (σελ. 19–58) είναι μία πολύ περιεκτική προπαρασκευαστική εισαγωγή στην C, που εμπλέκει πολλές έννοιες οι οποίες αναλύονται εκτενέστερα στη συνέχεια του βιβλίου. Είναι εξαιρετικά χρήσιμη η εισαγωγή αυτή, αν μη τι άλλο για να προσπαθήσει κανείς να καταλάβει τα προγράμματα που περιλαμβάνονται εκεί και, γιατί όχι, να δοκιμάσει να λύσει και τις ασκήσεις του.

Εύρεση Μ.Κ.Δ. και Ε.Κ.Π. τυχαίων αριθμών

```
/* File: gcdlcm.c */
#include <stdio.h>
#include <stdlib.h>      /* Header file for srand and rand functions */
#include <time.h>                    /* Header file for time function */
#define COMPUTATIONS 8    /* Pairs of numbers to compute GCD and LCM */
#define MAXNUMB 1000                                 /* Maximum number */

main()
{ int i, m, n, large, small, gcd, rem;
  long curtime, lcm;
  curtime = time(NULL);  /* Current time (in seconds since 1/1/1970) */
  printf("Current time is %ld\n\n", curtime);
  srand((unsigned int) curtime); /* Seed for random number generator */
  for (i=0 ; i < COMPUTATIONS ; i++) {
    m = rand() % MAXNUMB + 1;    /* Select 1st number in [1,MAXNUMB] */
    n = rand() % MAXNUMB + 1;    /* Select 2nd number in [1,MAXNUMB] */
    if (m > n) {                           /* Which number is larger? */
      large = m;                                      /* Larger is m */
      small = n;
    }
    else {
      large = n;                                      /* Larger is n */
      small = m;
    }
    while (small) {                          /* While small is not 0 */
      rem = large % small;          /* Find remainder of large/small */
      large = small;                  /* New large is previous small */
      small = rem;                   /* New small is remainder found */
    }
    gcd = large;    /* GCD of m and n equals to final value of large */
    lcm = (m*n)/gcd;                  /* LCM(m,n) * GCD(m,n) = m * n */
    printf("GCD(%3d,%3d) = %2d  LCM(%3d,%3d) = %6ld\n",
           m, n, gcd, m, n, lcm);
  }
}
```

```
% gcc -o gcdlcm gcdlcm.c
% ./gcdlcm
Current time is 1130666309

GCD(855,405) = 45  LCM(855,405) =   7695
GCD(125,139) =  1  LCM(125,139) =  17375
GCD(996,467) =  1  LCM(996,467) = 465132
GCD(191,890) =  1  LCM(191,890) = 169990
GCD(548,612) =  4  LCM(548,612) =  83844
GCD(378,531) =  9  LCM(378,531) =  22302
GCD(837,127) =  1  LCM(837,127) = 106299
GCD(938,656) =  2  LCM(938,656) = 307664
%
```

- Για την εύρεση τυχαίων ζευγαριών θετικών ακεραίων, χρησιμοποιείται η συνάρτηση `rand`, η οποία κάθε φορά που καλείται επιστρέφει ένα (ψευδο)τυχαίο αριθμό. Η αρχικοποίηση της γεννήτριας (ψευδο)τυχαίων αριθμών γίνεται καλώντας τη συνάρτηση `srand`, δίνοντάς της σαν παράμετρο την τρέχουσα χρονική στιγμή, όπως αυτή επιστρέφεται από τη συνάρτηση `time` (η παράμετρος `NULL` που δίνουμε στην `time` θα συζητηθεί στο μέλλον).

- Για την εύρεση του Μέγιστου Κοινού Διαιρέτη (Μ.Κ.Δ.) δύο αριθμών, χρησιμοποιούμε τον αλγόριθμο του Ευκλείδη. Δηλαδή, βρίσκουμε το υπόλοιπο της διαίρεσης του μεγαλύτερου με τον μικρότερο από τους αριθμούς και αντικαθιστούμε τον μεγαλύτερο με τον μικρότερο, τον μικρότερο με το υπόλοιπο της διαίρεσης και επαναλαμβάνουμε τη διαδικασία μέχρι ο μικρότερος να γίνει ίσος με 0. Τότε ο ζητούμενος Μ.Κ.Δ. είναι ο μεγαλύτερος.

- Για την εύρεση του Ε.Κ.Π., χρησιμοποιούμε τη σχέση $\text{ΕΚΠ}(m,n) \cdot \text{ΜΚΔ}(m,n) = m \cdot n$.

Η ροή του ελέγχου στην C

- Όπως σε κάθε γλώσσα προγραμματισμού, έτσι και στην C, υπάρχουν εντολές που επηρεάζουν τη ροή του ελέγχου στα προγράμματα (`if`, `switch`, `while`, `for`, κλπ.).
- Στις εντολές αυτές είναι πολύ συχνή η χρήση ενός μπλοκ εντολών. Ένα μπλοκ εντολών αποτελείται από εντολές που είναι κλεισμένες μέσα σε άγκιστρα ({ και }). Αν σ΄ ένα μπλοκ εντολών περιέχεται μία μόνο εντολή, αυτή δεν είναι απαραίτητο να περικλείεται μέσα σε άγκιστρα.
- Είναι εξαιρετικά χρήσιμο να μελετηθεί όλο το Κεφάλαιο 3 από το [KR] (σελ. 85–100), στο οποίο περιγράφονται αναλυτικά όλες οι δομές ελέγχου που προσφέρει η C. Ακολουθεί μία πιο συνοπτική περιγραφή των δομών αυτών.

Εντολή `if`

- Η σύνταξη της εντολής `if` είναι:
  `if` (⟨παράσταση⟩)
  ⟨μπλοκ εντολών⟩$_1$
  `else`
  ⟨μπλοκ εντολών⟩$_2$
- Αν η ⟨παράσταση⟩ είναι αληθής (έχει τιμή διάφορη από το μηδέν), θα εκτελεσθεί το ⟨μπλοκ εντολών⟩$_1$, αλλιώς το ⟨μπλοκ εντολών⟩$_2$.
- Το τμήμα από το `else` και μετά είναι προαιρετικό. Αν δεν υπάρχει και η ⟨παράσταση⟩ είναι ψευδής, ο έλεγχος θα πάει στην επόμενη εντολή του προγράμματος, δηλαδή στην πρώτη εντολή μετά το `if`.
- Η προαιρετικότητα του `else` μπορεί να προκαλέσει προβλήματα στην αναγνωσιμότητα προγραμμάτων με εμφωλευμένες εντολές `if`, αν δεν υπάρχει σαφής ομαδοποίηση με άγκιστρα. Αν δεν υπάρχουν άγκιστρα που να επιβάλλουν συγκεκριμένη δομή, κάθε `else` αντιστοιχεί στην αμέσως προηγούμενη `if` που δεν έχει `else`.
- Τι ακριβώς σημαίνει το παρακάτω;

```
if (n > 0)
   if (a > b)
      z = a;
else
   z = b;
```

  Προσοχή! Ο μεταγλωττιστής δεν ασχολείται με τη στοίχιση.

- Αυτό:

```
if (n > 0) {
   if (a > b)
      z = a;
   else
      z = b;
}
```

- Αν θέλαμε το `else` να αντιστοιχεί στο πρώτο `if`, έπρεπε να το γράψουμε έτσι:

```
if (n > 0) {
   if (a > b)
      z = a;
}
else
   z = b;
```

- Μία συνηθισμένη χρήση της εντολής `if` είναι όταν θέλουμε να ελέγξουμε μία σειρά από επάλληλες συνθήκες, και ανάλογα με το ποια θα βρεθεί ότι ισχύει πρώτη, να εκτελέσουμε ένα μπλοκ εντολών. Αυτό γίνεται ως εξής:

`if (`$\langle$παράσταση$\rangle_1$`)`
    $\langle$μπλοκ εντολών$\rangle_1$
`else if (`$\langle$παράσταση$\rangle_2$`)`
    $\langle$μπλοκ εντολών$\rangle_2$
.....
`else if (`$\langle$παράσταση$\rangle_{n-1}$`)`
    $\langle$μπλοκ εντολών$\rangle_{n-1}$
`else` $\langle$μπλοκ εντολών$\rangle_n$

- Η προηγούμενη εντολή `if` θα μπορούσε να γραφεί σε μία εκδοχή με στοίχιση ως εξής:

  `if` (⟨παράσταση⟩$_1$)
  ⟨μπλοκ εντολών⟩$_1$
  `else`
  `if` (⟨παράσταση⟩$_2$)
  ⟨μπλοκ εντολών⟩$_2$
  .....
  `else`
  `if` (⟨παράσταση⟩$_{n-1}$)
  ⟨μπλοκ εντολών⟩$_{n-1}$
  `else`
  ⟨μπλοκ εντολών⟩$_n$

  Η συγκεκριμένη στοίχιση, όμως, μάλλον κάνει πιο δυσανάγνωστη την εντολή, οπότε είναι καλό να μην την ακολουθήσει κάποιος.

- Ένα παράδειγμα:

```
if (x > 0) {
   sign = 1;
   printf("Number is positive\n");
}
else if (x < 0) {
   sign = -1;
   printf("Number is negative\n");
}
else {
   sign = 0;
   printf("Number is zero\n");
}
```

Εντολή `switch`

- Η σύνταξη της εντολής `switch` είναι:
  `switch` (⟨παράσταση⟩) {
  `case` ⟨σταθερά⟩$_1$:
  ⟨εντολές⟩$_1$
  `case` ⟨σταθερά⟩$_2$:
  ⟨εντολές⟩$_2$
  .....
  `default`:
  ⟨εντολές⟩
  }
  Η `switch` είναι μία διακλαδωμένη εντολή απόφασης που λειτουργεί όπως περιγράφεται στη συνέχεια.

- Η ⟨παράσταση⟩ πρέπει να είναι ακέραια. Υπολογίζεται η τιμή της.

- Κάθε ⟨σταθερά⟩$_i$ πρέπει να είναι ακέραια σταθερά και δεν πρέπει να υπάρχουν δύο `case` με την ίδια σταθερά.

- Αν η τιμή που έχει η ⟨παράσταση⟩ ισούται με κάποια ⟨σταθερά⟩$_i$, τότε ο έλεγχος μεταφέρεται στις ⟨εντολές⟩$_i$. Αν όχι, ο έλεγχος μεταφέρεται στις ⟨εντολές⟩ (μετά το `default`).

- Οι ⟨εντολές⟩$_i$ (και ⟨εντολές⟩) δεν είναι απαραίτητο να συνιστούν μπλοκ εντολών, δηλαδή να είναι κλεισμένες μέσα σε { και }.

- Μετά την εκτέλεση των εντολών σε μία `case`, ο έλεγχος μεταφέρεται στις εντολές της επόμενης `case`, εκτός αν τελευταία εντολή της προηγούμενης `case` είναι η `break`. Δεν είναι καλή πρακτική οι εντολές μίας `case` να συνεχίζουν με αυτές της επόμενης, εκτός αν ισχύει το επόμενο.

- Είναι δυνατόν για περισσότερες τιμές της μίας για την ⟨παράσταση⟩ να θέλουμε να εκτελεσθεί η ίδια ομάδα εντολών. Σ′ αυτήν την περίπτωση, τοποθετούμε τα αντίστοιχα `case` συνεχόμενα και βάζουμε την ομάδα εντολών στο τελευταίο απ′ αυτά.

- Η περίπτωση `default` είναι προαιρετική. Αν δεν υπάρχει και δεν ταιριάζει καμία `case`, ο έλεγχος μεταφέρεται μετά την εντολή `switch`.

- Παράδειγμα:

```
switch (x) {
   case 1:  printf("one\n");
            break;
   case 2:
   case 3:  printf("two or three\n");
            break;
   case 4:  printf("four\n");
            break;
   default: printf("other\n");
            break;
}
```

  Τι θα γινόταν αν έλειπαν κάποιες `break`; Γιατί βάλαμε `break` και στην `default` περίπτωση;

Εντολές βρόχου `while`

- Μία πιθανή σύνταξη της εντολής `while` είναι:
  `while (`⟨παράσταση⟩`)`
  ⟨μπλοκ εντολών⟩

- Αρχικά υπολογίζεται η ⟨παράσταση⟩. Αν έχει τιμή διάφορη του μηδενός (είναι αληθής) εκτελείται το ⟨μπλοκ εντολών⟩. Υπολογίζεται πάλι η ⟨παράσταση⟩ και ενόσω είναι αληθής, θα γίνεται ο κύκλος εκτέλεσης του ⟨μπλοκ εντολών⟩, μέχρι να γίνει η ⟨παράσταση⟩ ψευδής (δηλαδή ίση με το μηδέν).

- Παράδειγμα:

```
f = 1;
while (--n)
   f = f*(n+1);
```

  Τι υπολογίζει αυτό το `while`;

- Επίσης, θυμηθείτε και την εντολή `while` στο `http://www.di.uoa.gr/~ip/cprogs/capitalize.c`.

- Άλλη πιθανή σύνταξη της εντολής `while` είναι:
  `do`
  ⟨μπλοκ εντολών⟩
  `while (`⟨παράσταση⟩`)`

- Αρχικά εκτελείται το ⟨μπλοκ εντολών⟩. Μετά, υπολογίζεται η ⟨παράσταση⟩. Αν έχει τιμή διάφορη του μηδενός (είναι αληθής) εκτελείται πάλι το ⟨μπλοκ εντολών⟩ και συνεχίζεται ο κύκλος εκτέλεσής του, μέχρι να γίνει η ⟨παράσταση⟩ ψευδής (δηλαδή ίση με το μηδέν).

- Για λόγους καλύτερης αναγνωσιμότητας του προγράμματος, ακόμα και αν το ⟨μπλοκ εντολών⟩ αποτελείται από μία μόνο εντολή, συνήθως τη βάζουμε μέσα σε { και }.

- Παράδειγμα:

```
f = 1;
do {
   f = f*n;
} while (--n);
```

  Τι υπολογίζει αυτό το `while`;

- Επίσης, θυμηθείτε και την εντολή `do`/`while` στο `http://www.di.uoa.gr/~ip/cprogs/picomp.c`.

- Η βασική διαφορά των δύο εκδοχών της εντολής `while` είναι ότι στην πρώτη περίπτωση το ⟨μπλοκ εντολών⟩ μπορεί και να μην εκτελεσθεί καμία φορά, αν η ⟨παράσταση⟩ είναι αρχικά ψευδής, ενώ στην εκδοχή `do`/`while`, το ⟨μπλοκ εντολών⟩ θα εκτελεσθεί τουλάχιστον μία φορά, αφού η ⟨παράσταση⟩ ελέγχεται στο τέλος.

Εντολή βρόχου `for`

- Η σύνταξη της εντολής `for` είναι:
  `for (`⟨παράσταση⟩$_1$ `;` ⟨παράσταση⟩$_2$ `;` ⟨παράσταση⟩$_3$`)`
  ⟨μπλοκ εντολών⟩

- Διαδικαστικά, ισοδυναμεί με τις εξής εντολές:
  ⟨παράσταση⟩$_1$`;`
  `while (`⟨παράσταση⟩$_2$`) {`
  ⟨μπλοκ εντολών⟩
  ⟨παράσταση⟩$_3$`;`
  }

- Δηλαδή, αρχικά υπολογίζεται η ⟨παράσταση⟩$_1$. Συνήθως, πρόκειται για την αρχικοποίηση ενός δείκτη που ελέγχει μία επαναληπτική διαδικασία. Στη συνέχεια, εκτελείται επαναλαμβανόμενα το ⟨μπλοκ εντολών⟩, που αποτελεί το σώμα της διαδικασίας, ενόσω η ⟨παράσταση⟩$_2$, που συνήθως ελέγχει την τιμή του δείκτη ελέγχου, είναι αληθής. Πριν τη διεξαγωγή νέας επανάληψης, εκτελείται και η ⟨παράσταση⟩$_3$, που συνήθως μεταβάλλει τον δείκτη ελέγχου.

- Παράδειγμα:

```
f = 1;
for (i=1 ; i <= n ; i++)
   f = f*i;
```

  Τι υπολογίζει αυτό το `for`;

- Επίσης, θυμηθείτε και τις εντολές `for` στα
  `http://www.di.uoa.gr/~ip/cprogs/easter.c` και
  `http://www.di.uoa.gr/~ip/cprogs/magic.c`.

- Είναι δυνατόν κάποια από τις $\langle$παράσταση$\rangle_{1,2\ \acute{\eta}\ 3}$ να μην υπάρχει, συνήθως η πρώτη ή/και η τρίτη. Αν δεν υπάρχει η δεύτερη, τότε δεν γίνεται έλεγχος για τερματισμό του βρόχου, οπότε πρέπει αυτό να γίνει βιαίως με κάποιο τρόπο μέσα από το σώμα της επανάληψης. Σε κάθε περίπτωση, όποια $\langle$παράσταση$\rangle_i$ και να λείπει, τα `;` υπάρχουν κανονικά.

- Η εντολή
  `for ( ; ; )`
  $\langle$μπλοκ εντολών$\rangle$
  είναι η κλασική υλοποίηση ενός ατέρμονος βρόχου.

- Επίσης, στην C υπάρχει και ο τελεστής `,` (κόμμα), με τον οποίο μπορούμε να κατασκευάσουμε μία σύνθετη παράσταση που θα υπολογισθεί από αριστερά προς τα δεξιά. Η τιμή της σύνθετης παράστασης ισούται με την τιμή της δεξιότερης από τις απλές που την συνιστούν.

- Μερικές φορές, θέλουμε να γράψουμε μία εντολή `for` που θα ελέγχεται από περισσότερους του ενός δείκτες. Αυτό γίνεται χρησιμοποιώντας τον τελεστή `,` στην $\langle$παράσταση$\rangle_1$ και στην $\langle$παράσταση$\rangle_3$. Παράδειγμα:

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

Εντολές `break` και `continue`

- Μία χρήση της εντολής `break` είναι αυτή που είδαμε στην εντολή `switch`, για τον τερματισμό της ακολουθίας εντολών που θα εκτελεσθούν όταν ταιριάξει κάποια συγκεκριμένη `case`.

- Επίσης, με την εντολή `break`, μπορούμε να τερματίσουμε βιαίως τις επαναλήψεις ενός βρόχου (`while`, `do/while` ή `for`), πριν ισχύσει η συνθήκη τερματισμού (ή όταν ο βρόχος είναι ατέρμων). Παράδειγμα:

```
while (!finished) {
   ......................
   if (bad_data)
      break;
   ......................
}
```

- Με την εντολή `continue`, ο έλεγχος στο εσωτερικό ενός βρόχου μεταφέρεται στο τέλος του, για να αρχίσει η επόμενη επανάληψη. Παράδειγμα:

```
for (i=0 ; i < n ; i++) {
   if (a[i] < 0)  /* Ignore negative elements */
      continue;
   ............  /* Process positive elements */
}
```

Εντολή goto και ετικέτες

- Κάθε γλώσσα προγραμματισμού, έτσι και η C, έχει μία εντολή για ρητή μεταφορά του ελέγχου σε κάποιο άλλο σημείο (ετικέτα) του προγράμματος. Στην C, η εντολή αυτή είναι η goto.

- Η goto μπορεί να χρησιμοποιηθεί σε πολύ εξειδικευμένες περιπτώσεις, για παράδειγμα όταν θέλουμε να γίνει βίαιη έξοδος από εμφωλευμένους βρόχους, όταν ο έλεγχος βρίσκεται σε κάποιο εσωτερικό βρόχο. Αυτό δεν μπορεί να γίνει με την εντολή break. Παράδειγμα:

```
for (i=0 ; i < n ; i++)
   for (j=0 ; j < m ; j++)
      if (a[i] == b[j])
         goto found;
..............   /* Didn't find common element */
found:
..............   /* Found a[i] == b[j] */
```

- Η εντολή goto βλάπτει σοβαρά την ιδέα του δομημένου προγραμματισμού, γιατί μπορεί να οδηγήσει σε προγράμματα δυσανάγνωστα και δύσκολα συντηρήσιμα, και γι΄ αυτό πρέπει να χρησιμοποιείται σπάνια, αν όχι καθόλου. [α΄]

[α΄] Στα πλαίσια του συγκεκριμένου μαθήματος, απλά ΑΠΑΓΟΡΕΥΕΤΑΙ η χρήση της.