Η γλώσσα προγραμματισμού C

- Μεταβλητές, σταθερές, τύποι και δηλώσεις
- Εντολές αντικατάστασης, τελεστές και παραστάσεις
- Η ροή του ελέγχου
- Δομή προγράμματος, συναρτήσεις και εξωτερικές μεταβλητές
- Εμβέλεια και χρόνος ζωής μεταβλητών
- Αναδρομή
- Διευθύνσεις θέσεων μνήμης, δείκτες και πίνακες
- Δυναμική δέσμευση μνήμης
- Συμβολοσειρές
- Πίνακες δεικτών, δείκτες σε δείκτες και πολυδιάστατοι πίνακες
- Δείκτες σε συναρτήσεις
- Ορίσματα γραμμής εντολών
- Απαριθμήσεις, δομές, αυτο-αναφορικές δομές (λίστες, δυαδικά δέντρα), ενώσεις, πεδία bit και δημιουργία νέων ονομάτων τύπων
- Είσοδος και έξοδος
- Χειρισμός αρχείων
- Προεπεξεργαστής της C και μακροεντολές

Καλημέρα κόσμε της C

```
/* File: helloworld.c */
#include <stdio.h>

main()
{ printf("Hello world\n");
}
```

```
% gcc -o helloworld helloworld.c
% ./helloworld
Hello world
%
```

- Ό,τι περικλείεται μέσα σε `/*` και `*/` θεωρείται σχόλιο και δεν λαμβάνεται υπόψη από τον μεταγλωττιστή.
- Η γραμμή `#include <stdio.h>` είναι οδηγία προς τον προεπεξεργαστή της C να συμπεριλάβει σ΄ εκείνο το σημείο τα περιεχόμενα του *αρχείου επικεφαλίδας* (header file) `stdio.h`, το οποίο περιέχει χρήσιμες δηλώσεις για τις συναρτήσεις εισόδου/εξόδου.
- Κάθε πρόγραμμα C πρέπει να περιέχει ακριβώς μία συνάρτηση με όνομα `main`, που είναι αυτή από την οποία θα αρχίσει να εκτελείται το εκτελέσιμο πρόγραμμα που θα προκύψει από τη μεταγλώττιση.
- Η `printf` είναι μία συνάρτηση εξόδου, που όταν κληθεί, εκτυπώνει ό,τι της έχει δοθεί μέσα στις παρενθέσεις, στην προκείμενη περίπτωση το *αλφαριθμητικό ή συμβολοσειρά* (string) `"Hello world"` (χωρίς τα `"`), ακολουθούμενο από μία αλλαγή γραμμής (`\n`).

Πόσο είναι το π;

```
/* File: picomp.c */
#include <stdio.h>            /* Header file for standard I/O library */
#include <math.h>                     /* Header file for math library */

main()
{ long i;
  double sum, current, pi;
  i = 1;                               /* Denominator of current term */
  sum = 0.0;                                          /* So far sum */
  do {
    current = 1/(((double) i)*((double) i));          /* Current term */
    sum = sum+current;                    /* Add current term to sum */
    i++;                                             /* Next term now */
  } while (current > 1.0e-15);  /* Stop if current term is very small */
  pi = sqrt(6*sum);                     /* Compute approximation of pi */
  printf("Summed %8ld terms, pi is %10.8f\n", i-1, pi);
}
```

```
% gcc -o picomp picomp.c -lm
% ./picomp
Summed 31622777 terms, pi is 3.14159262
%
```

- Το πρόγραμμα υπολογίζει το π με βάση τη σειρά:

$$\frac{\pi^2}{6} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \frac{1}{6^2} + \frac{1}{7^2} + \frac{1}{8^2} + \ldots$$

- Το `#include <math.h>` χρειάζεται επειδή χρησιμοποιείται στο πρόγραμμα η μαθηματική συνάρτηση `sqrt`, για την εύρεση τετραγωνικής ρίζας. Προσοχή: Στην εντολή μεταγλώττισης και σύνδεσης (`gcc`) χρειάζεται και το `-lm`, για να συμπεριληφθεί στο εκτελέσιμο πρόγραμμα και η μαθηματική βιβλιοθήκη της C.

- Το `i` συμβολίζει μία ακέραια μεταβλητή στην οποία θα φυλάσσεται, δηλαδή στην περιοχή της μνήμης που της αντιστοιχεί ο μεταγλωττιστής, η τιμή του τρέχοντος παρονομαστή κατά τον υπολογισμό του αθροίσματος. Δηλώνεται με μέγεθος `long`, ώστε να μπορούν να φυλαχθούν σ΄ αυτήν ακέραιοι μεγέθους τουλάχιστον 4 bytes, ήτοι τουλάχιστον μέχρι το $2^{31} - 1 = 2147483647$. Αν είχε δηλωθεί απλώς σαν `int`, μπορεί να περιοριζόμασταν σε 2 bytes για τη φύλαξη, δηλαδή έως το $2^{15} - 1 = 32767$.

- Τα `sum`, `current` και `pi` είναι μεταβλητές για τη φύλαξη πραγματικών αριθμών, του μέχρι στιγμής υπολογισμένου αθροίσματος, του τρέχοντος όρου της σειράς και της τελικής προσεγγιστικής τιμής του π, αντίστοιχα. Δηλώθηκαν σαν `double` για να έχουμε εσωτερική αναπαράσταση των αριθμών αυτών με μεγαλύτερη ακρίβεια (διπλή) σε δεκαδικά ψηφία, απ΄ αυτήν που θα είχαμε αν είχαν δηλωθεί σαν `float`.

- Με τις εντολές αντικατάστασης `i = 1` και `sum = 0.0`, δίνουμε αρχικές τιμές στον τρέχοντα παρονομαστή και στο μέχρι στιγμής άθροισμα.

- Η ομάδα εντολών μέσα στο μπλοκ `do {.....} while` θα εκτελείται συνεχώς, όσο ο τρέχων όρος του αθροίσματος είναι μεγαλύτερος από κάποια ελάχιστη τιμή, την 10^{-15} (= `1.0e-15`).

- Σε κάθε επανάληψη του μπλοκ `do {.....} while`, υπολογίζουμε αρχικά τον τρέχοντα όρο του αθροίσματος (το πρόθεμα `(double)` στο `i` υπάρχει για να μετατραπεί η τιμή του ακεραίου `i` σε πραγματική τιμή διπλής ακρίβειας, πριν συμμετάσχει στην αριθμητική πράξη). Μετά προστίθεται ο τρέχων όρος του αθροίσματος στο μέχρι στιγμής άθροισμα, με το `sum = sum+current` (συνήθως το γράφουμε και σαν `sum += current`), και τέλος αυξάνουμε τον τρέχοντα παρονομαστή κατά 1, με το `i++` (ισοδύναμο με το `i = i+1`).
- Όταν τελειώσει το μπλοκ `do {.....} while`, στη μεταβλητή `sum` υπάρχει το άθροισμα όλων των όρων της σειράς που είναι μεγαλύτεροι από 10^{-15}. Αυτό είναι περίπου ίσο με το $\frac{\pi^2}{6}$. Οπότε, μετά υπολογίζουμε προσεγγιστικά το π με την εντολή `pi = sqrt(6*sum)` (δηλαδή $\sqrt{6 \times sum}$).
- Με την κλήση της συνάρτησης `printf` εκτυπώνεται το αποτέλεσμα. Συγκεκριμένα, εκτυπώνεται ό,τι υπάρχει μέσα στα ", αλλά στη θέση του `%8ld` εκτυπώνεται η τιμή του `i-1`, σαν δεκαδικός (`d`) ακέραιος μεγέθους `long` (`l`) σε 8 τουλάχιστον θέσεις, στοιχισμένος δεξιά. Επίσης, στη θέση του `%10.8f` εκτυπώνεται η υπολογισμένη τιμή του π, σαν πραγματικός αριθμός (`f`) σε 10 τουλάχιστον θέσεις, με δεξιά στοίχιση, από τις οποίες οι 8 για το κλασματικό μέρος.

Γράψτε προγράμματα C που να υπολογίζουν με αρκετή ακρίβεια τα παρακάτω αθροίσματα. Μπορείτε να βρείτε με ποιους ενδιαφέροντες αριθμούς ισούνται;

$$S_1 = \frac{1}{1^2} - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + \frac{1}{5^2} - \frac{1}{6^2} + \frac{1}{7^2} - \frac{1}{8^2} + \dots$$

$$S_2 = \frac{1}{1} - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \frac{1}{6} + \frac{1}{7} - \frac{1}{8} + \frac{1}{9} - \dots$$

$$S_3 = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} - \frac{1}{15} + \frac{1}{17} - \dots$$

$$S_4 = \frac{1}{1 \times 3} + \frac{1}{3 \times 5} + \frac{1}{5 \times 7} + \frac{1}{7 \times 9} + \frac{1}{9 \times 11} + \dots$$

$$S_5 = \frac{1}{2 \times 3 \times 4} - \frac{1}{4 \times 5 \times 6} + \frac{1}{6 \times 7 \times 8} - \frac{1}{8 \times 9 \times 10} + \dots$$

$$S_6 = \frac{1}{1^4} + \frac{1}{2^4} + \frac{1}{3^4} + \frac{1}{4^4} + \frac{1}{5^4} + \frac{1}{6^4} + \frac{1}{7^4} + \frac{1}{8^4} + \dots$$

Το ίδιο και για το γινόμενο:

$$P = \frac{2}{1} \times \frac{2}{3} \times \frac{4}{3} \times \frac{4}{5} \times \frac{6}{5} \times \frac{6}{7} \times \frac{8}{7} \times \frac{8}{9} \dots$$

Πότε είναι το Ορθόδοξο Πάσχα;

```
/* File: easter.c */
#include <stdio.h>
#define STARTYEAR 2000
#define ENDYEAR   2015

main()
{ int year, a, b, c, d, e;
  for (year = STARTYEAR ; year <= ENDYEAR ; year++) {
    a = year % 19;
    b = year % 4;    /* Gauss method for Easter date computation */
    c = year % 7;
    d = (19*a+15) % 30;
    e = (2*b+4*c+6*d+6) % 7;
    printf("Easter in the year %d: ", year);
    if (d+e+4 > 30)                              /* Easter in May */
      printf("  May %d\n", d+e-26);
    else                                        /* Easter in April */
      printf("April %d\n", d+e+4);
  }
}
```

```
% gcc -o easter easter.c
% ./easter
Easter in the year 2000: April 30
Easter in the year 2001: April 15
Easter in the year 2002:   May 5
Easter in the year 2003: April 27
Easter in the year 2004: April 11
Easter in the year 2005:   May 1
Easter in the year 2006: April 23
Easter in the year 2007: April 8
Easter in the year 2008: April 27
Easter in the year 2009: April 19
Easter in the year 2010: April 4
Easter in the year 2011: April 24
Easter in the year 2012: April 15
Easter in the year 2013:   May 5
Easter in the year 2014: April 20
Easter in the year 2015: April 12
%
```

- Το πρόγραμμα εφαρμόζει τη μέθοδο του Gauss για τον υπολογισμό της ημερομηνίας του Πάσχα το έτος $year$. Σύμφωνα με αυτήν, το Πάσχα είναι στις $(d + e + 4)$ Απριλίου (αν η ημερομηνία αυτή είναι μεγαλύτερη από 30, τότε πέφτει μέσα στον Μάιο), όπου $d = (19 \cdot a + 15) \bmod 30$, $e = (2 \cdot b + 4 \cdot c + 6 \cdot d + 6) \bmod 7$, $a = year \bmod 19$, $b = year \bmod 4$ και $c = year \bmod 7$.

- Με τις δηλώσεις `#define .....` ζητάμε από τον προεπεξεργαστή της C να αντικαταστήσει μέσα στο αρχείο, όπου βρει το σύμβολο που ακολουθεί το `#define` με το σύμβολο που είναι αμέσως μετά. Στην προκείμενη περίπτωση, ζητάμε να αντικατασταθεί το `STARTYEAR` με 2000 και το `ENDYEAR` με 2015.

- Το μπλοκ `for (` E_1 `;` E_2 `;` E_3 `) {.....}` περιγράφει έναν βρόχο που θα πρέπει να εκτελεσθεί επαναλαμβανόμενα. Πριν αρχίσει η εκτέλεσή του, θα εκτελεσθεί η εντολή E_1. Εφ΄ όσον η συνθήκη E_2 είναι αληθινή, θα γίνει η τρέχουσα επανάληψη στο βρόχο. Με το τέλος κάθε επανάληψης, εκτελείται και η εντολή E_3.

- Στις εντολές αντικατάστασης, ο τελεστής `%` παριστάνει το υπόλοιπο διαίρεσης (mod).

- Η δομή `if (` C `)` E_1 `else` E_2 σημαίνει ότι θέλουμε να εκτελεσθεί η εντολή (ή μπλοκ εντολών) E_1, αν η συνθήκη C είναι αληθής, ή η εντολή (ή μπλοκ εντολών) E_2, αν η συνθήκη C είναι ψευδής.

Να κατασκευασθεί μαγικό τετράγωνο 5×5

```
/* File: magic.c */
#include <stdio.h>
#define SIZE 5

main()
{ int i, j, k;
  int x[SIZE][SIZE];                        /* The board to be filled */
  for (i=0 ; i < SIZE ; i++)
    for (j=0 ; j < SIZE ; j++)
      x[i][j] = 0;                       /* Mark all squares as empty */
  i = (SIZE-1)/2;                                    /* Go to middle */
  j = SIZE-1;                                        /* right square */
  for (k=1 ; k <= SIZE*SIZE ; k++) {
    if (x[i][j] != 0) {                      /* If current not empty */
      i = i-1;                                 /* Move one square up */
      j = j-2;                                /* and two squares left */
      if (i < 0) i = i+SIZE;                   /* If you are outside */
      if (j < 0) j = j+SIZE;  /* go to the respective square inside */
    }
    x[i][j] = k;                /* Fill current square with number k */
    i++;                                      /* Move one square down */
    if (i == SIZE) i = 0;                    /* If outside get inside */
    j++;                                     /* Move one square right */
    if (j == SIZE) j = 0;                    /* If outside get inside */
  }
  for (i=0 ; i < SIZE ; i++) {
    for (j=0 ; j < SIZE ; j++)
      printf("%4d ", x[i][j]);                    /* Print out board */
    printf("\n");
  }
}
```

```
% gcc -o magic magic.c
% ./magic
  11   10    4   23   17
  18   12    6    5   24
  25   19   13    7    1
   2   21   20   14    8
   9    3   22   16   15
%
```

- Το πρόγραμμα εφαρμόζει τη μέθοδο του De la Loubère για να κατασκευάσει ένα μαγικό τετράγωνο 5×5 (η μέθοδος εφαρμόζεται για περιττό μήκος πλευράς), δηλαδή ένα πλαίσιο με 25 θέσεις σε διάταξη τετραγώνου μέσα στις οποίες είναι τοποθετημένοι οι αριθμοί από το 1 έως το 25, μία φορά ο καθένας, έτσι ώστε σε κάθε γραμμή, κάθε στήλη και κάθε μία από τις δύο διαγωνίους να έχουμε το ίδιο άθροισμα (αλήθεια, πόσο;). Για την περιγραφή της μεθόδου σε μία ψευδογλώσσα, δείτε τον αλγόριθμο 118, στη δεύτερη σελίδα του `http://www.di.uoa.gr/~ip/magic-sq.pdf`. Ο αλγόριθμος 117, στην πρώτη σελίδα του κειμένου αυτού, μπορεί να εφαρμοσθεί για την κατασκευή μαγικών τετραγώνων άρτιας πλευράς.

- Με τη δήλωση `int x[SIZE][SIZE]` ζητάμε να δεσμευθεί χώρος για ένα διδιάστατο πίνακα ακεραίων `x` με `SIZE` ($= 5$) γραμμές και `SIZE` στήλες. Προσοχή: Οι γραμμές και οι στήλες αριθμούνται από το 0 έως το `SIZE-1` ($= 4$). Με την έκφραση `x[3][1]`, μπορούμε να αναφερθούμε στο στοιχείο που βρίσκεται στην 4η γραμμή και 2η στήλη. Φυσικά, στην C μπορούμε να ορίσουμε και μονοδιάστατους πίνακες, αλλά και πίνακες με διάσταση μεγαλύτερη από 2.

- Παρατηρήστε τη χρήση των εντολών `if`. Δεν υπάρχει τμήμα `else`. Αν ισχύει η συνθήκη, εκτελείται η εντολή (ή μπλοκ εντολών) που ακολουθεί. Αλλιώς, ο έλεγχος πηγαίνει στην επόμενη εντολή. Ο τελεστής `!=` στη συνθήκη της πρώτης εντολής `if` σημαίνει "διάφορο" (έλεγχος αν το τρέχον τετραγωνίδιο είναι γεμάτο).

Μετατροπή πεζών γραμμάτων σε κεφαλαία

```
/* File: capitalize.c */
#include <stdio.h>

main()
{ int ch; /* Be careful! Declare ch as int because of getchar() and EOF */
  ch = getchar();                                  /* Read first character */
  while (ch != EOF) {           /* Go on if we didn't reach end of file */
    if (ch >= 'a' && ch <= 'z')                    /* If lower case letter */
      ch = ch - ('a'-'A'); /* Move 'a'-'A' positions in the ASCII table */
    putchar(ch);                                    /* Print out character */
    ch = getchar();                                 /* Read next character */
  }
}
```

```
% gcc -o capitalize capitalize.c
% cat capinp.txt
This is a text file with 2 lines to
test the C program "capitalize.c".
% ./capitalize < capinp.txt
THIS IS A TEXT FILE WITH 2 LINES TO
TEST THE C PROGRAM "CAPITALIZE.C".
%
```

- Η συνάρτηση `getchar` παρέχεται από την πρότυπη (standard) βιβλιοθήκη εισόδου/εξόδου της C. Διαβάζει ένα χαρακτήρα από την είσοδο και επιστρέφει στο όνομά της την ακέραια τιμή του ASCII κωδικού του χαρακτήρα. Αν δεν υπάρχει άλλος χαρακτήρας να διαβαστεί, δηλαδή φτάσαμε στο τέλος της εισόδου, τότε επιστρέφει στο όνομά της μία ειδική τιμή, το `EOF`, που είναι `#define`'d σε μία συγκεκριμένη ακέραια τιμή μέσα στο αρχείο επικεφαλίδας `stdio.h`.

- Η ομάδα εντολών μέσα στο μπλοκ `while` {.....} θα εκτελείται συνεχώς, όσο υπάρχουν και άλλοι χαρακτήρες για να διαβάσουμε από την είσοδο.
- Η συνθήκη μέσα στο `if` είναι μία λογική σύζευξη (τελεστής `&&`). Τα `'a'` και `'A'` είναι σταθερές χαρακτήρα και ισούνται με την αριθμητική τιμή των κωδικών των αντίστοιχων χαρακτήρων στον ASCII πίνακα, δηλαδή `'a'`=97 και `'A'`=65.
- Το συγκεκριμένο πρόγραμμα βασίζεται στην υπόθεση ότι στον ASCII πίνακα οι κεφαλαίοι λατινικοί χαρακτήρες είναι σε συνεχόμενες θέσεις και ότι το ίδιο ισχύει και για τους πεζούς. Αυτό όντως ισχύει, γι΄ αυτό και το πρόγραμμα δουλεύει σωστά. Για την μετατροπή ενός πεζού χαρακτήρα σε κεφαλαίο, δεν χρειάζεται να γνωρίζουμε τις ακριβείς τιμές των `ASCII` κωδικών, αρκεί να μετατοπίσουμε τον πεζό χαρακτήρα κατά `'a'-'A'` θέσεις.
- Η συνάρτηση `putchar` παρέχεται επίσης από την πρότυπη βιβλιοθήκη εισόδου/εξόδου της C (είναι η "αδελφή" συνάρτηση της `getchar`) και η λειτουργία της είναι να εκτυπώσει στην έξοδο τον χαρακτήρα του οποίου τον ASCII κωδικό δίνουμε μέσα στις παρενθέσεις.