## Δομή ενός προγράμματος C – Συναρτήσεις

- Μία συνάρτηση C είναι ένα αυτόνομο, πακεταρισμένο τμήμα προγράμματος που φέρει σε πέρας μία διαδικασία η οποία έχει σαφείς προδιαγραφές εισόδου και εξόδου και συγκεκριμένο όνομα.
- Συνήθως, κωδικοποιούμε σε μία συνάρτηση έναν αλγόριθμο που πρόκειται να χρησιμοποιήσουμε στο πρόγραμμά μας αρκετές φορές ή εκτιμούμε ότι είναι γενικότερης χρησιμότητας και θα μπορούσε να φανεί χρήσιμος και σε προγράμματα που θα γράψουμε στο μέλλον.
- Σε γενικές γραμμές, είναι καλή πρακτική να γράφουμε προγράμματα C που αποτελούνται από πολλές και μικρές συναρτήσεις, παρά από λίγες και μεγάλες.
- Ακριβώς μία από τις συναρτήσεις ενός προγράμματος C πρέπει να έχει το όνομα `main`. Είναι η συνάρτηση από την οποία θα ξεκινήσει η εκτέλεση του προγράμματος.
- Ορισμός συνάρτησης:
  ⟨τύπος⟩ ⟨όνομα⟩ (⟨τυπικές παράμετροι⟩)
  {
  ⟨δηλώσεις και εντολές⟩
  }
- Για τον ορισμό κάθε συνάρτησης, δηλώνουμε το ⟨όνομα⟩ που έχει και, μέσα σε παρενθέσεις, τα ονόματα και τους τύπους που έχουν οι, λεγόμενες, ⟨τυπικές παράμετροι⟩, εφ΄ όσον υπάρχουν.

- Οι τυπικές παράμετροι μίας συνάρτησης είναι μεταβλητές, στις οποίες δίνονται συγκεκριμένες τιμές κατά την κλήση της συνάρτησης και οι οποίες χρησιμοποιούνται μέσα στο σώμα της συνάρτησης (⟨δηλώσεις και εντολές⟩) για να επιτευχθεί ο στόχος της συνάρτησης.
- Το σώμα μίας συνάρτησης περικλείεται μέσα σε { και }.
- Μία συνάρτηση μπορεί να επιστρέψει κάποια τιμή στο όνομά της, η οποία να χρησιμοποιηθεί από την συνάρτηση που την κάλεσε. Ο ⟨τύπος⟩ της τιμής που επιστρέφει μία συνάρτηση δίνεται μπροστά από το όνομά της, στον ορισμό της.
- Αν δεν ορίζεται τύπος επιστροφής για μία συνάρτηση, θεωρείται ότι επιστρέφει `int`.
- Αν δεν ενδιαφερόμαστε να επιστρέψει μία συνάρτηση κάποια τιμή, πρέπει να ορίζουμε ότι επιστρέφει `void`, τον "κενό" τύπο. Ακόμα και αν έχουμε ορίσει μία συνάρτηση να επιστρέφει κάποιο τύπο, όχι `void`, δεν είναι τελικά υποχρεωτικό να το κάνει, και αν το κάνει, δεν είναι υποχρεωτικό για την καλούσα συνάρτηση να χρησιμοποιήσει την επιστρεφόμενη τιμή.
- Η επιστροφή τιμής από συνάρτηση γίνεται με την εντολή `return` ⟨παράσταση⟩. Υπολογίζεται η τιμή που έχει η ⟨παράσταση⟩, επιστρέφεται και η συνάρτηση τερματίζει. Με μία απλή εντολή `return` τερματίζει χωρίς να επιστραφεί κάποια τιμή.

- Η συνάρτηση `main` μπορεί να ορίζεται
  1. είτε χωρίς τυπικές παραμέτρους
  2. είτε με δύο παραμέτρους συγκεκριμένων τύπων, μέσω των οποίων μπορεί να γνωρίζει η `main` αν το πρόγραμμα εκτελέσθηκε με ορίσματα στη γραμμή εντολής και, αν ναι, με ποια. Περισσότερα, γι΄ αυτό το θέμα, αργότερα.
- Η `main` επιστρέφει τιμή τύπου `int`, η οποία είναι διαθέσιμη στο περιβάλλον του λειτουργικού συστήματος από το οποίο εκτελέσθηκε το πρόγραμμα. Τελικά, ο απολύτως σωστός τρόπος για να ορίσουμε τη συνάρτηση `main` χωρίς τυπικές παραμέτρους, τον οποίο θα υιοθετήσουμε από εδώ και πέρα, είναι ο εξής (το `void` υποδηλώνει την έλλειψη τυπικών παραμέτρων):

  ```
  int main(void)
  ```

- Επίσης, πάντα θα πρέπει να επιστρέφουμε μία τιμή τερματισμού από τη `main`, με την εντολή `return` (τυπικά το 0, σε περίπτωση επιτυχούς τερματισμού του προγράμματος).
- Ο μεταγλωττιστής απαιτεί πριν την κλήση μίας συνάρτησης να γνωρίζει τους τύπους των τυπικών παραμέτρων της και τον τύπο της επιστρεφόμενης τιμής. Συνεπώς, ή πρέπει να προηγείται ο ορισμός μίας συνάρτησης από την κλήση της, ή, αν έπεται, να έχει προηγηθεί απλώς μία προαναγγελία της συνάρτησης (τύπος επιστρεφόμενης τιμής, όνομα και τύποι τυπικών παραμέτρων), το λεγόμενο πρωτότυπο της συνάρτησης.
- Περισσότερα για συναρτήσεις, στις παραγράφους §4.1 και §4.2 του [KR].

# Συνάρτηση ύψωσης σε δύναμη

```
/* File: powers.c */
#include <stdio.h>
#define MAXM 4          /* Maximum base of powers to compute */
#define MAXN 6                            /* Maximum exponent */

int power(int, int);                    /* Prototype of power */

int main(void)
{ int m, n, p;
  for (m=2 ; m <= MAXM ; m++)            /* Start from base 2 */
    for (n=2 ; n <= MAXN ; n++) {    /* Start from exponent 2 */
       p = power(m,n);                 /* Call power function */
       printf("%d^%d = %d\n", m, n, p);
    }
  return 0;
}

int power(int base, int n)
{ int p;
  for (p=1 ; n > 0 ; n--)                 /* base^n equals to */
    p *= base;          /* base * base * ... * base (n times) */
  return p;                                  /* Return result */
}
```

```
% gcc -o powers powers.c
% ./powers
2^2 = 4
2^3 = 8
2^4 = 16
2^5 = 32
2^6 = 64
3^2 = 9
3^3 = 27
3^4 = 81
3^5 = 243
3^6 = 729
4^2 = 16
4^3 = 64
4^4 = 256
4^5 = 1024
4^6 = 4096
%
```

# Συνάρτηση υπολογισμού παραγοντικού

```
/* File: factorial.c */
#include <stdio.h>
#define MAXN 12             /* Compute factorials up to MAXN */

int factorial(int n)                  /* No prototype needed */
{ int f;
  for (f=1 ; n > 0 ; n--)       /* Initialize factorial to 1 */
    f *= n;         /* Multiply into factorial n, n-1, ..., 1 */
  return f;                                  /* Return result */
}

int main(void)
{ int n;                          /* For all n from 1 to MAXN */
  for (n=1 ; n <= MAXN ; n++)           /* Call function and */
    printf("%2d! = %d\n", n, factorial(n));  /* print result */
  return 0;
}
```

```
% gcc -o factorial factorial.c
% ./factorial
 1! = 1
 2! = 2
 3! = 6
 4! = 24
 5! = 120
 6! = 720
 7! = 5040
 8! = 40320
 9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
%
```

## Εμβέλεια και χρόνος ζωής μεταβλητών

- Οι μεταβλητές που ορίζονται μέσα σε μία συνάρτηση λέγονται αυτόματες ή τοπικές. Ο λόγος είναι ότι οι μεταβλητές αυτές έχουν περιορισμένη εμβέλεια και χρόνο ζωής. Ισχύουν μόνο μέσα στη συνάρτηση που έχουν οριστεί και για όσο χρόνο εκτελείται η συνάρτηση.
- Εκτός από τις αυτόματες μεταβλητές, μπορούμε να ορίσουμε σ΄ ένα πρόγραμμα C και εξωτερικές ή καθολικές μεταβλητές. Οι μεταβλητές αυτές ορίζονται όπως και οι αυτόματες (με τον τύπο τους, το όνομά τους και με πιθανή αρχικοποίησή τους), μόνο που ορίζονται έξω από συναρτήσεις, στο ίδιο επίπεδο με αυτές.
- Οι εξωτερικές μεταβλητές έχουν το προτέρημα ότι είναι ορατές από κάθε συνάρτηση, μπορούν δηλαδή να χρησιμοποιηθούν στα σώματα των συναρτήσεων, και διατηρούνται καθ΄ όλη τη διάρκεια της εκτέλεσης του προγράμματος.
- Μέσω των εξωτερικών μεταβλητών, μπορεί να επιτευχθεί επικοινωνία μεταξύ συναρτήσεων χωρίς να χρειάζεται δεδομένα που κάποια συνάρτηση πρέπει να περάσει σε κάποια άλλη να το κάνει αυτό μέσω παραμέτρων.

- Το πλεονέκτημα που προσφέρουν οι εξωτερικές μεταβλητές μπορεί να μετατραπεί σε μειονέκτημα, αν δεν χρησιμοποιούνται με φειδώ. Προγράμματα που χρησιμοποιούν ευρέως εξωτερικές μεταβλητές μπορεί να είναι πολύ δυσανάγνωστα και δύσκολα συντηρήσιμα. Επίσης, συναρτήσεις που βασίζονται σε εξωτερικές μεταβλητές δεν είναι γενικής χρήσης.

- Αν ένα πρόγραμμα είναι διασπασμένο σε πολλά αρχεία και θέλουμε να έχουμε μία εξωτερική μεταβλητή που να είναι ορατή από τις συναρτήσεις σε όλα τα αρχεία, τότε πρέπει σε ακριβώς ένα αρχείο να δώσουμε τον ορισμό της μεταβλητής, για παράδειγμα

  ```
  int sp;
  ```

  ενώ σε καθένα από τα άλλα αρχεία πρέπει να κάνουμε μία δήλωση της μεταβλητής, προτάσσοντας τον προσδιοριστή `extern`, για παράδειγμα

  ```
  extern int sp;
  ```

  Ουσιαστικά, ο μεταγλωττιστής δεσμεύει μνήμη για μία εξωτερική μεταβλητή όταν δει τον ορισμό της (γι΄ αυτό πρέπει να υπάρχει αυτός ακριβώς μία φορά) και ενημερώνεται για τον τύπο μίας μεταβλητής όταν δει μία δήλωσή της με `extern` (θυμηθείτε ότι αν ένα πρόγραμμα είναι διασπασμένο σε πολλά αρχεία, τα αρχεία αυτά μεταγλωττίζονται ξεχωριστά).

- Αν θέλουμε η εμβέλεια μίας εξωτερικής μεταβλητής να είναι μόνο το αρχείο στο οποίο ορίζεται και να μην συγχέεται με κάποια εξωτερική μεταβλητή με το ίδιο όνομα, αλλά σε άλλο αρχείο, μπορούμε να προτάξουμε του ορισμού της τον προσδιοριστή `static`, για παράδειγμα

```
static double sum;
```

- Ο προσδιοριστής `static` μπορεί να εφαρμοσθεί και σε μεταβλητές που ορίζονται μέσα σε συνάρτηση, όμως με άλλη σημασία. Η εμβέλεια μίας τοπικής μεταβλητής που έχει ορισθεί σαν `static` μέσα σε συνάρτηση, είναι η συνάρτηση αυτή, αλλά η μεταβλητή δεν είναι αυτόματη. Διατηρείται, όπως και η τιμή της, και μετά τον τερματισμό της συνάρτησης. Σε νέα κλήση της συνάρτησης, η μεταβλητή έχει την τιμή που πήρε μετά την προηγούμενη κλήση.
- Μη αρχικοποιημένες εξωτερικές μεταβλητές θεωρούνται ότι έχουν τιμή 0.
- Μη αρχικοποιημένες αυτόματες μεταβλητές έχουν απροσδιόριστες τιμές.
- Οι τυπικές παράμετροι μίας συνάρτησης είναι αυτόματες μεταβλητές για τη συνάρτηση.

- Οι αυτόματες μεταβλητές σε μία συνάρτηση μπορούν να ορίζονται, εκτός από το επίπεδο του σώματος της συνάρτησης, και σε πιο εσωτερικά μπλοκ εντολών (για παράδειγμα στο σώμα μίας εντολής `while`). Αν υπάρχουν πολλαπλοί ορισμοί για το ίδιο όνομα μεταβλητής σε διάφορα μπλοκ, μέσα σε καθένα απ΄ αυτά ισχύει ο ορισμός που έγινε σ΄ αυτό, ή, αν δεν υπάρχει, αυτός που έγινε στο αμέσως πιο εξωτερικό του, ή, αν δεν υπάρχει, στο πιο εξωτερικό, κ.ο.κ.
- Αν υπάρχει σύγκρουση ονόματος μεταξύ μίας εξωτερικής μεταβλητής και μίας αυτόματης μέσα σ΄ ένα μπλοκ, ισχύει η αυτόματη.
- Παράδειγμα:

```
int count;
float y;

double myfun(int y)
{  int i;
   ..............
   while (......) {
      int i, j;
      ..........
      for (.......) {
         int count;
         ..........
      }
   }
}
```

Ποιες μεταβλητές ισχύουν πού;

- Όταν μεταγλωττίζεται ένα πρόγραμμα, δεσμεύεται ο απαιτούμενος χώρος για να φυλαχθούν όλες οι εξωτερικές μεταβλητές του προγράμματος (αλλά και οι τοπικές μεταβλητές που έχουν δηλωθεί σαν `static` μέσα σε συναρτήσεις). Ο χώρος αυτός είναι των στατικών (καθολικών) δεδομένων και είναι εκ των προτέρων γνωστός και δεσμευμένος μέσα στο εκτελέσιμο πρόγραμμα και αντιστοιχεί στο χώρο που θα καταληφθεί στη μνήμη όταν φορτωθεί το πρόγραμμα για να εκτελεσθεί.
- Οι αυτόματες μεταβλητές μίας συνάρτησης, ακριβώς επειδή έχουν περιορισμένο χρόνο ζωής, έχουν ένα διαφορετικό τρόπο φύλαξης, μέσω της, λεγόμενης, στοίβας.
- Η στοίβα για ένα πρόγραμμα είναι μία δομή τύπου LIFO (Last In First Out), δηλαδή μπορούμε να εισάγουμε διαδοχικά στοιχεία σ΄ αυτήν, αλλά αν θέλουμε να εξαγάγουμε κάποιο, θα εξαχθεί αυτό που εισήχθη πιο πρόσφατα.
- Όταν καλείται μία συνάρτηση, ο έλεγχος μεταφέρεται σ΄ αυτήν, αλλά πριν αρχίσει η εκτέλεσή της, δεσμεύεται χώρος στη στοίβα για τις αυτόματες μεταβλητές της (συμπεριλαμβανομένων και των τυπικών παραμέτρων της). Αν η συνάρτηση αυτή καλέσει στο σώμα της άλλη συνάρτηση, ακολουθείται η ίδια διαδικασία.

- Όταν τερματίζει μία συνάρτηση, επιστρέφει, αν χρειάζεται, κάποια τιμή στη συνάρτηση που την κάλεσε και ελευθερώνεται ο χώρος στη στοίβα που είχε κρατηθεί για το περιβάλλον (αυτόματες μεταβλητές) της συνάρτησης.
- Ο μηχανισμός φύλαξης των αυτόματων μεταβλητών των συναρτήσεων στη στοίβα επιτρέπει τη χρήση μίας πολύ ισχυρής προγραμματιστικής δυνατότητας, της *αναδρομής* (recursion).
- Μία συνάρτηση είναι αναδρομική όταν καλεί τον εαυτό της. Λόγω της στοίβας, δεν υπάρχει κίνδυνος σύγχυσης με τις μεταβλητές της συνάρτησης και τις τιμές τους, αφού σε κάθε κλήση δημιουργείται νέο περιβάλλον στη στοίβα για τις μεταβλητές της συγκεκριμένης ενεργοποίησης.
- Φυσικά, πρέπει η αναδρομική κλήση μίας συνάρτησης να γίνεται υπό συνθήκη, γιατί αλλιώς η εκτέλεση του προγράμματος δεν θα μπορούσε να γίνει με πεπερασμένο τρόπο.
- Για εξωτερικές και αυτόματες μεταβλητές, αλλά και για την αναδρομή, δείτε τις παραγράφους §4.3, §4.4 και §4.6 έως §4.10 του [KR].

# Υπολογισμός παραγοντικού με αναδρομή

```
/* File: recfact.c */
#include <stdio.h>
#define MAXN 12             /* Compute factorials up to MAXN */

int recfact(int);                    /* Prototype of recfact */

int main(void)
{ int n;                          /* For all n from 0 to MAXN */
  for (n=0 ; n <= MAXN ; n++)           /* Call function and */
    printf("%2d! = %d\n", n, recfact(n));    /* print result */
  return 0;
}

int recfact(int n)
{ if (!n)                                        /* If n == 0 */
    return 1;                                       /* 0! = 1 */
  else
    return n*recfact(n-1);                /* n! = n * (n-1)! */
}
```

```
% gcc -o recfact recfact.c
% ./recfact
 0! = 1
 1! = 1
 2! = 2
 3! = 6
 4! = 24
 5! = 120
 6! = 720
 7! = 5040
 8! = 40320
 9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
%
```

# Μετατροπές μεταξύ δεκαδικών και δυαδικών αριθμών

```
/* File: convdecbin.c */
#include <stdio.h>
#define ERROR -1       /* Return value for illegal character */

int getinteger(int base)
{ char ch; /* No need to declare ch as int - no EOF handling */
  int val = 0;                    /* Initialize return value */
  while ((ch = getchar()) != '\n')    /* Read up to new line */
    if (ch >= '0' && ch <= '0'+base-1)   /* Legal character? */
      val = base*val + (ch-'0');      /* Update return value */
    else
      return ERROR;                /* Illegal character read */
  return val; /* Everything OK - Return value of number read */
}

int main(void)
{ int n, digs[32], ind = 0;
  printf("Please, give a binary number:  ");
  n = getinteger(2);                   /* Read binary number */
  if (n == ERROR) {
    printf("Sorry, illegal character\n");  return 1; }
  else                        /* Print decimal value of number */
    printf("Decimal equivalent is:         %d\n", n);
  printf("Please, give a decimal number: ");
  n = getinteger(10);                 /* Read decimal number */
  if (n == ERROR) {
    printf("Sorry, illegal character\n");  return 1; }
  else {                    /* Convert number to binary digits */
    while (n) {             /* Repeat until number equals to 0 */
      digs[ind++] = n%2;         /* New digit is number mod 2 */
      n = n/2;                      /* New number is number/2 */
    }
    printf("Binary equivalent is:          ");
    while (ind--)             /* Print digits in reverse order */
      printf("%d", digs[ind]);
    printf("\n");
  }
  return 0;
}
```

```
% gcc -o convdecbin convdecbin.c
% ./convdecbin
Please, give a binary number:  100010010111101000100100 1
Decimal equivalent is:         18019401
Please, give a decimal number: 18019401
Binary equivalent is:          100010010111101000100100 1
% ./convdecbin
Please, give a binary number:  10011101
Decimal equivalent is:         157
Please, give a decimal number: 65535
Binary equivalent is:          1111111111111111
%
```

- Αν θεωρούσαμε ότι η συνάρτηση `getinteger` είναι γενικής χρησιμότητας, τι ακριβώς θα έπρεπε να βάλουμε σ΄ ένα πηγαίο αρχείο `getinteger.c` και τι σ΄ ένα αρχείο επικεφαλίδας `getinteger.h`, ώστε να χρησιμοποιούμε τη συνάρτηση αυτή και σε άλλα προγράμματα;

- Και ένα πρόβλημα:
  Η ακολουθία Fibonacci ορίζεται ως εξής:

$$a_0 = 1$$

$$a_1 = 1$$

$$a_n = a_{n-1} + a_{n-2}, \quad \forall\ n \geq 2$$

  Γράψτε πρόγραμμα C που να υπολογίζει, για δεδομένο $k$, τον $(k+1)$-οστό όρο $a_k$ της ακολουθίας Fibonacci. Υλοποιήστε δύο διαφορετικές εκδοχές του προγράμματος, μία χωρίς αναδρομή και μία με αναδρομή. Έχετε να σχολιάσετε κάτι;