Προγραμματισμός Ι

Συναρτήσεις

Δημήτρης Μιχαήλ

Τμήμα Πληροφορικής και Τηλεματικής
Χαροκόπειο Πανεπιστήμιο

Συναρτήσεις

Ως τώρα γράφαμε όλα τα προγράμματα μας μέσα στην **`main`**.

1. Τι θα γινόταν όμως εαν ένα πρόγραμμα μας είναι 200000 γραμμές κώδικα;
2. Στην περίπτωση που θέλουμε να εκτελέσουμε τον ίδιο ή παρόμοιο κώδικα πολλές φορές;

Συναρτήσεις

1. Χωρίζουμε το πρόγραμμα σε μέρη
2. Φιλοσοφία του ”διαίρει και βασίλευε” (divide and conquer)
3. Τα μέρη αντιστοιχούν σε επιμέρους και μικρότερα υποπροβλήματα

Αναλογία με τον πραγματικό κόσμο:

- Ο προϊστάμενος αναθέτει μια δουλειά σε κάποιον υφιστάμενο χωρίς να γνωρίζει τις λεπτομέρειες.

Συναρτήσεις

1. Τα υποπροβλήματα είναι πιο εύκολα διαχειρίσιμα
2. Δημιουργούμε μια συνάρτηση για κάθε υποπρόβλημα
3. Ο κώδικας είναι πιο ευανάγνωστος και διαχειρίσιμος
4. Χρησιμοποιούμε υπάρχουσες συναρτήσεις σε καινούρια προγράμματα για την λύση νέων προβλημάτων
5. Δεν επαναλαμβάνουμε ίδιο κώδικα πολλές φορές στα προγράμματα μας
6. Αφαίρεση (abstraction): Κρύβουμε λεπτομέρειες που δεν είναι απαραίτητες στον προγραμματιστή

Συναρτήσεις

1. Τα υποπροβλήματα είναι πιο εύκολα διαχειρίσιμα
2. Δημιουργούμε μια συνάρτηση για κάθε υποπρόβλημα
3. Ο κώδικας είναι πιο ευανάγνωστος και διαχειρίσιμος
4. Χρησιμοποιούμε υπάρχουσες συναρτήσεις σε καινούρια προγράμματα για την λύση νέων προβλημάτων
5. Δεν επαναλαμβάνουμε ίδιο κώδικα πολλές φορές στα προγράμματα μας
6. Αφαίρεση (abstraction): Κρύβουμε λεπτομέρειες που δεν είναι απαραίτητες στον προγραμματιστή

Συναρτήσεις

1. Τα υποπροβλήματα είναι πιο εύκολα διαχειρίσιμα
2. Δημιουργούμε μια συνάρτηση για κάθε υποπρόβλημα
3. Ο κώδικας είναι πιο ευανάγνωστος και διαχειρίσιμος
4. Χρησιμοποιούμε υπάρχουσες συναρτήσεις σε καινούρια προγράμματα για την λύση νέων προβλημάτων
5. Δεν επαναλαμβάνουμε ίδιο κώδικα πολλές φορές στα προγράμματα μας
6. Αφαίρεση (abstraction): Κρύβουμε λεπτομέρειες που δεν είναι απαραίτητες στον προγραμματιστή

Συναρτήσεις

1. Τα υποπροβλήματα είναι πιο εύκολα διαχειρίσιμα
2. Δημιουργούμε μια συνάρτηση για κάθε υποπρόβλημα
3. Ο κώδικας είναι πιο ευανάγνωστος και διαχειρίσιμος
4. Χρησιμοποιούμε υπάρχουσες συναρτήσεις σε καινούρια προγράμματα για την λύση νέων προβλημάτων
5. Δεν επαναλαμβάνουμε ίδιο κώδικα πολλές φορές στα προγράμματα μας
6. Αφαίρεση (abstraction): Κρύβουμε λεπτομέρειες που δεν είναι απαραίτητες στον προγραμματιστή

Συναρτήσεις

1. Τα υποπροβλήματα είναι πιο εύκολα διαχειρίσιμα
2. Δημιουργούμε μια συνάρτηση για κάθε υποπρόβλημα
3. Ο κώδικας είναι πιο ευανάγνωστος και διαχειρίσιμος
4. Χρησιμοποιούμε υπάρχουσες συναρτήσεις σε καινούρια προγράμματα για την λύση νέων προβλημάτων
5. Δεν επαναλαμβάνουμε ίδιο κώδικα πολλές φορές στα προγράμματα μας
6. Αφαίρεση (abstraction): Κρύβουμε λεπτομέρειες που δεν είναι απαραίτητες στον προγραμματιστή

Συναρτήσεις

1. Τα υποπροβλήματα είναι πιο εύκολα διαχειρίσιμα
2. Δημιουργούμε μια συνάρτηση για κάθε υποπρόβλημα
3. Ο κώδικας είναι πιο ευανάγνωστος και διαχειρίσιμος
4. Χρησιμοποιούμε υπάρχουσες συναρτήσεις σε καινούρια προγράμματα για την λύση νέων προβλημάτων
5. Δεν επαναλαμβάνουμε ίδιο κώδικα πολλές φορές στα προγράμματα μας
6. Αφαίρεση (abstraction): Κρύβουμε λεπτομέρειες που δεν είναι απαραίτητες στον προγραμματιστή

Συναρτήσεις της Βιβλιοθήκης της C

Η βιβλιοθήκη της C παρέχει πολλές συναρτήσεις. Μερικές τις έχουμε πετύχει ήδη:

1. **`printf`**() βοηθάει για να εκτυπώσουμε στην οθόνη
2. **`scanf`**() διαβάζει από το πληκτρολόγιο
3. **`getchar`**() διαβάζει ένα χαρακτήρα από το πληκτρολόγιο

Όταν γράφουμε προγράμματα δεν πρέπει να ανακαλύπτουμε τον τροχό! Χρησιμοποιούμε τις έτοιμες βιβλιοθήκες συναρτήσεων.

Ορισμός Συνάρτησης σε C

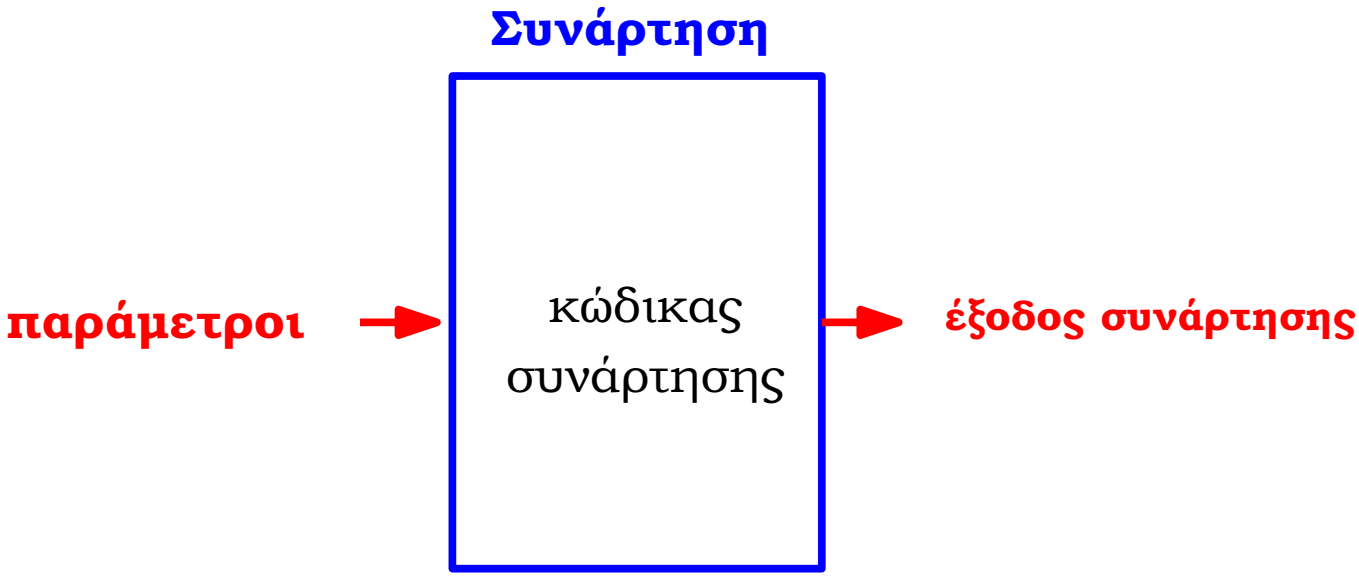
Παρακάτω βλέπουμε την γενική δομή ορισμού μιας συνάρτησης σε C:

```
1  return_value_type  name(comma_separated_parameter_list) {
2      declarations
3
4      statements
5  }
```

Κάθε συνάρτηση έχει:

1. ένα μοναδικό όνομα
2. μια λίστα παραμέτρων
3. μια τιμή επιστροφής

Συνάρτηση


Συνάρτηση
παράμετροι
κώδικας
συνάρτησης
έξοδος συνάρτησης

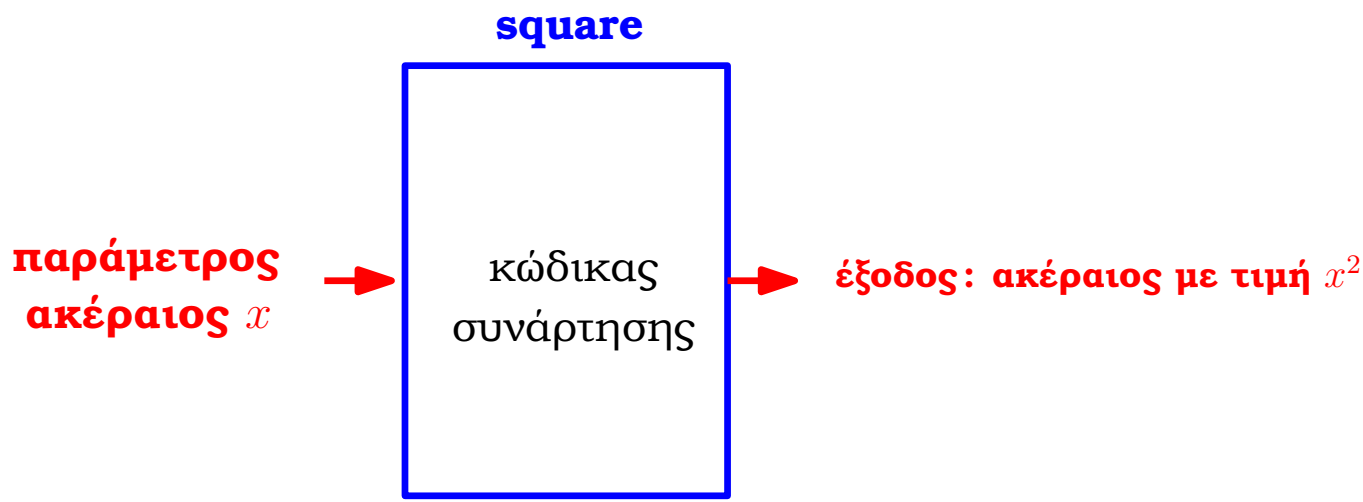
Παράδειγμα Ορισμού Συνάρτησης σε C

Παρακάτω βλέπουμε τον ορισμό μιας συνάρτησης σε C η οποία υπολογίζει το τετράγωνο ενός αριθμού:

```
1  int square(int x) {
2      return x * x;
3  }
```

- η συνάρτηση επιστρέφει έναν ακέραιο
- η συνάρτηση παίρνει μια παράμετρο με όνομα **x** και τύπο ακέραιο

Συνάρτηση Υπολογισμού Τετραγώνου



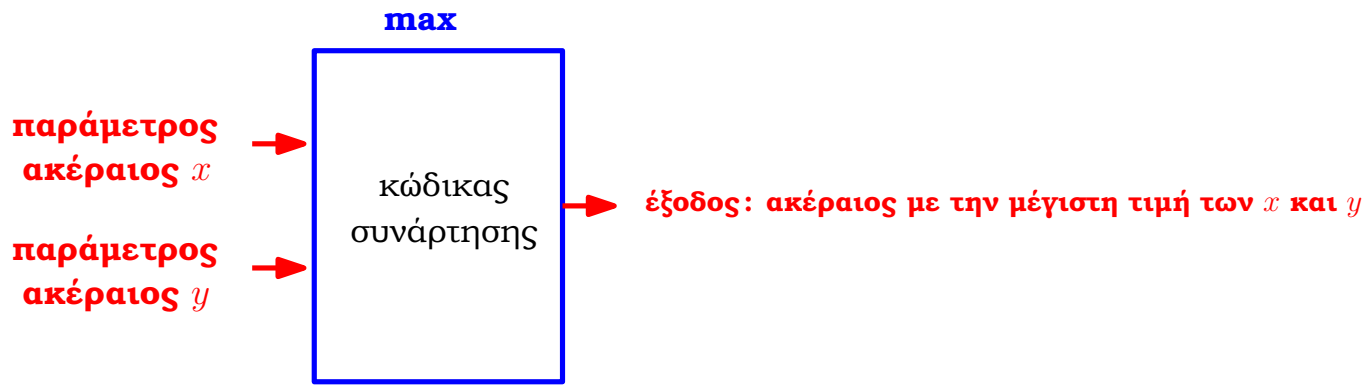
Παράδειγμα Ορισμού Συνάρτησης σε C

Ορισμός μιας συνάρτησης σε C η οποία υπολογίζει το μέγιστο δύο ακεραίων:

```
1  int max(int x, int y) {
2      if (x > y)
3          return x;
4      else
5          return y;
6  }
```

- η συνάρτηση επιστρέφει έναν ακέραιο
- η συνάρτηση παίρνει δύο ακέραιες παραμέτρους **`x`** και **`y`**

Συνάρτηση Υπολογισμού Μεγίστου 2 Ακεραίων



Κώδικας Συνάρτησης

Σε μια συνάρτηση γράφουμε κώδικα όπως ακριβώς γράφουμε στην συνάρτηση **main**().

```
1  #include <stdio.h>
2
3  int max(int a, int b, int c) {
4      int max;
5
6      max = a;
7      if (b > max)
8          max = b;
9      if (c > max)
10         max = c;
11     return max;
12 }
13
14 int main() {
15     printf("max of 1, 2 and 3 = %d\n", max(1,2,3));
16
17     return 0;
18 }
```

Μεταβλητές Συνάρτησης

```
1  int max(int a, int b, int c) {
2      int max;
3
4      max = a;
5      if (b > max)
6          max = b;
7      if (c > max)
8          max = c;
9
10     return max;
11 }
```

Η παραπάνω συνάρτηση έχει 4 τοπικές μεταβλητές.

- τις παραμέτρους **a**, **b** και **c**
- την μεταβλητή **max**

Η μεταβλητές αυτές είναι γνωστές μόνο μέσα στην συνάρτηση.

Κλήση Συνάρτησης

Καλούμε μια συνάρτηση με το όνομα της και τιμές για τις παραμέτρους μέσα σε παρενθέσεις.

```
1  int main() {
2      int i, j, k;
3      int m;
4
5      printf("Give 3 values\n");
6      scanf("%d%d%d", &i, &j, &k);
7
8      m = max(i, j, k);
9      printf("maximum = %d\n", m);
10
11     return 0;
12 }
```

Την τιμή επιστροφής πρέπει είτε να την αναθέσουμε σε κάποια μεταβλητή ή να την χρησιμοποιήσουμε απευθείας π.χ να την τυπώσουμε με την **`printf`**().

Τύπος Επιστροφής

Όταν ορίζουμε μια συνάρτηση πρέπει να δηλώσουμε και τι τύπο δεδομένων θα επιστρέψει, π.χ

```
1  int max(int x, int y) {
2      return (x>y)?x:y;
3  }
```

Τι κάνουμε όμως στην περίπτωση που δεν θέλουμε να επιστρέψουμε τίποτα;

Τύπος Επιστροφής

Η C μας παρέχει την δεσμευμένη λέξη **`void`** που υποδηλώνει πως μια συνάρτηση δεν επιστρέφει τίποτα.

```
1  void printstars(int n) {
2      int i;
3
4      for(i = 0; i < n; i++)
5          putchar('*');
6
7      return;    /* could be omitted */
8  }
```

Η παραπάνω συνάρτηση τυπώνει **`n`** αστερίσκους και δεν επιστρέφει τίποτα.

Επιστροφή από Συνάρτηση

Με την δεσμευμένη λέξη **`return`** τερματίζει η εκτέλεση μιας συνάρτησης.

```
1  void printstars(int n) {
2      int i;
3
4      for(i = 0; i < n; i++)
5          putchar('*');
6
7      return;    /* FUNCTION END HERE */
8
9      for(i = 0; i < n; i++)
10         putchar('*');
11 }
```

Ο κώδικας που ακολουθεί την **`return`** δεν θα εκτελεστεί ποτέ.

Ροή Προγράμματος

Η ροή του προγράμματος αλλάζει όταν συναντήσει μια κλήση συνάρτησης

- `f(a, b, c)`

1. πρώτα υπολογίζονται οι εκφράσεις που αντιστοιχούν στα ορίσματα `a`, `b` και `c`,
2. οι τιμές των εκφράσεων `a`, `b` και `c` αντιγράφονται στα ορίσματα της `f` με την ίδια σειρά,
3. ο έλεγχος μεταπηδά στην πρώτη εντολή της `f`,
4. οι εντολές της `f` εκτελούνται μέχρι να συναντηθεί
   - `return`
   - τέλος συνάρτησης, δηλαδή },
5. η κλήση `f(a,b,c)` αντικαθίσταται με την τιμή επιστροφής,
6. ο έλεγχος συνεχίζει με τον κώδικα που ακολουθεί την κλήση της συνάρτησης.

Κλήση Συνάρτησης

```
1  #include <stdio.h>
2
3  void printstars(int n) {
4      int i;
5      for(i = 0; i < n; i++)
6          putchar('*');
7  }
8
9  int main() {
10     int i;
11     for(i = 1; i <= 5; i++) {
12         printstars(i);
13         printf("\n");
14     }
15     return 0;
16 }
```

Μόλις καλέσουμε μια συνάρτηση το πρόγραμμα αρχίζει και εκτελεί τον κώδικα της συνάρτησης.

Κλήση Συνάρτησης

```
1  #include <stdio.h>
2
3  void printstars(int n) {
4      int i;
5      for(i = 0; i < n; i++)
6          putchar('*');
7  }
8
9  int main() {
10     int i;
11     for(i = 1; i <= 5; i++) {
12         printstars(i);
13         printf("\n");
14     }
15     return 0;
16 }
```

Πριν ξεκινήσει να εκτελεί τον κώδικα της συνάρτησης αναθέτει στην τοπική μεταβλητή `n` της συνάρτησης την τιμή που δώσαμε όταν καλέσαμε την συνάρτηση.

Κλήση Συνάρτησης

```
1  #include <stdio.h>
2
3  void printstars(int n) {
4      int i;
5      for(i = 0; i < n; i++)
6          putchar('*');
7  }
8
9  int main() {
10     int i;
11     for(i = 1; i <= 5; i++) {
12         printstars(i);
13         printf("\n");
14     }
15     return 0;
16 }
```

Η εκτέλεση μιας συνάρτησης τερματίζει είτε με την εντολή **return** είτε άμα φτάσει η εκτέλεση στο τέλος της συνάρτησης.

Κλήση Συνάρτησης

```
1  #include <stdio.h>
2
3  void printstars(int n) {
4      int i;
5      for(i = 0; i < n; i++)
6          putchar('*');
7  }
8
9  int main() {
10     int i;
11     for(i = 1; i <= 5; i++) {
12         printstars(i);
13         printf("\n");
14     }
15     return 0;
16 }
```

Μόλις τελειώσει η εκτέλεση της συνάρτησης, η εκτέλεση επιστρέφει στην γραμμή του προγράμματος που κάλεσε την συνάρτηση και συνεχίζει κανονικά.

Κλήση Συνάρτησης από Συνάρτηση

```
1  #include <stdio.h>
2
3  void printchar(char c, int s) {
4      int i;
5      for(i = 0; i < s; i++)
6          putchar(c);
7  }
8
9  void printstarspaces(int s, int n) {
10     printchar(' ', s);
11     printchar('*', n);
12     printchar(' ', s);
13 }
14
15 int main() {
16     int i, n = 9;
17     for(i = 1; i <= n; i+=2) {
18         printstarspaces((n-i)/2, i);
19         printf("\n");
20     }
21     return 0;
22 }
```

Μέσα από μια συνάρτηση μπορούμε να καλέσουμε άλλες συναρτήσεις.

Κλήση Συνάρτησης από Συνάρτηση

```
1  #include <stdio.h>
2
3  void printchar(char c, int s) {
4      int i;
5      for(i = 0; i < s; i++)
6          putchar(c);
7  }
8
9  void printstarspaces(int s, int n) {
10     printchar(' ', s);
11     printchar('*', n);
12     printchar(' ', s);
13 }
14
15 int main() {
16     int i, n = 9;
17     for(i = 1; i <= n; i+=2) {
18         printstarspaces((n-i)/2, i);
19         printf("\n");
20     }
21     return 0;
22 }
```

Τι τυπώνει αυτός ο κώδικας;

Αρχέτυπα και Ορισμοί

```
1  #include <stdio.h>
2
3  int max(int x, int y);
4
5  int main() {
6      int a,b;
7      printf("Enter two integers: ");
8      scanf("%d%d", &a, &b);
9      printf("Maximum is: %d\n", max(a,b));
10     return 0;
11 }
12
13 int max(int x, int y) {
14     if (x > y)
15         return x;
16     else
17         return y;
18 }
```

με την χρήση αρχέτυπου

- μπορούμε να καλέσουμε μια συνάρτηση χωρίς να την έχουμε πλήρως ορίσει

Αρχέτυπα και Ορισμοί

```
1  #include <stdio.h>
2
3  int max(int x, int y);
4
5  int main() {
6      int a,b;
7      printf("Enter two integers: ");
8      scanf("%d%d", &a, &b);
9      printf("Maximum is: %d\n", max(a,b));
10     return 0;
11 }
12
13 int max(int x, int y) {
14     if (x > y)
15         return x;
16     else
17         return y;
18 }
```

με την χρήση αρχέτυπου

- μπορεί μια συνάρτηση να είναι ορισμένη σε άλλο αρχείο κώδικα

Αρχέτυπα και Ορισμοί

```
1  #include <stdio.h>
2
3  int max(int x, int y);
4
5  int main() {
6      int a,b;
7      printf("Enter two integers: ");
8      scanf("%d%d", &a, &b);
9      printf("Maximum is: %d\n", max(a,b));
10     return 0;
11 }
12
13 int max(int x, int y) {
14     if (x > y)
15         return x;
16     else
17         return y;
18 }
```

με την χρήση αρχέτυπου

- μπορεί μια συνάρτηση να είναι ήδη μεταγλωττισμένη

Αρχέτυπα και Ορισμοί

```
1  #include <stdio.h>
2
3  int max(int x, int y);
4
5  int main() {
6      int a,b;
7      printf("Enter two integers: ");
8      scanf("%d%d", &a, &b);
9      printf("Maximum is: %d\n", max(a,b));
10     return 0;
11 }
12
13 int max(int x, int y) {
14     if (x > y)
15         return x;
16     else
17         return y;
18 }
```

με την χρήση αρχέτυπου

- ο μεταγλωττιστής έχει αρκετές πληροφορίες για να επικυρώσει πως η κλήση της συγκεκριμένης συνάρτησης είναι σωστή

Αρχεία Επικεφαλίδων

Από την αρχή του μαθήματος λέμε στον προ-επεξεργαστή να διαβάσει το αρχείο ”stdio.h”.

```
1  #include <stdio.h>
2
3  int main() {
4      /* code goes here */
5  }
```

Αυτό το αρχείο βρίσκεται σε κάποιο καλά ορισμένο μέρος στο σύστημα (διαφορετικό ανάλογα με το λειτουργικό σύστημα ή/και τον μεταγλωττιστή) και περιέχει αρχέτυπα συναρτήσεων για Είσοδο/Έξοδο (I/O).

Αρχεία Επικεφαλίδων

Τι είναι μια βιβλιοθήκη;

- συλλογή συναρτήσεων και τύπων δεδομένων
- συνήθως παρέχει κάποια εξειδικευμένη λειτουργία
- συνηθως ήδη μεταγλωττισμένος κώδικας
- παρέχει αρχεία επικεφαλίδων για να ξέρουμε τι δυνατότητες υπάρχουν (ονόματα συναρτήσεων, τύποι δεδομένων, κ.τ.λ)
- καλή τεκμηρίωση των δυνατοτήτων του κώδικα

Η βιβλιοθήκη της C (standard library) παρέχει πάρα πολλές δυνατότητες.

Βασικά Αρχεία Επικεφαλίδων Βιβλιοθήκης C

<stdio.h>	αρχέτυπα συναρτήσεων για είσοδο/έξοδο
<stdlib.h>	αρχέτυπα συναρτήσεων για μετατροπή αριθμών σε κείμενο και αντίστροφα, κατανομή μνήμης, τυχαίους αριθμούς και άλλες βοηθητικές συναρτήσεις
<math.h>	αρχέτυπα συναρτήσεων για μαθηματικές συναρτήσεις
<assert.h>	προσθήκη διαγνωστικών για διόρθωση προγράμματος
<ctype.h>	έλεγχος ιδιοτήτων χαρακτήρων και μετατροπές από πεζά γράμματα σε κεφαλαία και αντιστρόφως

Μαθηματικές Συναρτήσεις και ο Τύπος `double`

Ο τύπος **`double`** όπως και ο τύπος **`float`** είναι μια προσεγγιστική αναπαράσταση των πραγματικών αριθμών.

Έχει μεγαλύτερο μέγεθος και μπορεί να αναπαραστήσει περισσότερους αριθμούς.

Μαθηματικές Συναρτήσεις και ο Τύπος `double`

Η βιβλιοθήκη της C ορίζει μέσω του `<math.h>` διάφορες μαθηματικές συναρτήσεις, όπως

```
1  double sqrt(double x);
2
3  double exp(double x);
4
5  double log(double x);
6
7  double pow(double x, double y);
8
9  double sin(double x);
```

και άλλες παρόμοιες.

Πολλές από αυτές υπάρχουν και για τον τύπο `float`.

Κλήση Συναρτήσεων και Παράμετροι

Οι γλώσσες προγραμματισμού παρέχουν 2 τρόπους κλήσης μιας συνάρτησης:

1. κλήση μέσω τιμής (call by value)
2. κλήση μέσω αναφοράς (call by reference)

Κλήση Μέσω Τιμής

Call by Value

- Στην κλήση μέσω τιμής, πριν κληθεί μια συνάρτηση γίνεται ένα αντίγραφο όλων των μεταβλητών που έχουν δοθεί ως παράμετροι.
- Η συνάρτηση καλείται με παραμέτρους αυτά τα αντίγραφα.
- Με το τέλος της συνάρτησης τα αντίγραφα αυτά σβήνονται από την μνήμη.

Κλήση Μέσω Αναφοράς

Call by Reference

Στην κλήση μέσω αναφοράς η συνάρτηση καλείται με τις πραγματικές μεταβλητές ως παραμέτρους.

- Η γλώσσα C **δεν** υποστηρίζει κλήση μέσω αναφοράς, αλλά μπορεί να την προσομοιώσει.

Δεν έχουμε ακόμη τις απαραίτητες γνώσεις για να δούμε τις λεπτομέρειες.

Κλήση Μέσω Τιμής

Call by Value

```
1  #include <stdio.h>
2
3  void increment(int x) {
4      x++;
5  }
6
7  int main() {
8      int y = 5;
9
10     printf("y = %d\n", y);
11     increment(y);
12     printf("y = %d\n", y);
13 }
```

Κλήση Μέσω Τιμής

Call by Value

```
1  #include <stdio.h>
2
3  void increment(int x) {
4      x++;
5  }
6
7  int main() {
8      int y = 5;
9
10     printf("y = %d\n", y);
11     increment(y);
12     printf("y = %d\n", y);
13 }
```

Και τα δύο **printf**() τυπώνουν την τιμή 5 μιας και πριν κληθεί η συνάρτηση **increment**() δημιουργείτε ένα αντίγραφο της μεταβλητής **y**.

Κλήση Μέσω Αναφοράς

Call by Reference

Η γλώσσα C δεν υποστηρίζει κλήση μέσω αναφοράς, αλλά μπορεί να την προσομοιώσει

Δεν έχουμε ακόμη τις απαραίτητες γνώσεις για να δούμε τις λεπτομέρειες.

Το έχουμε δει όμως στην πράξη όταν καλούμε την συνάρτηση **`scanf`**():

```
1  #include <stdio.h>
2
3  int main() {
4      int x;
5
6      printf("Give a number\n");
7      scanf("%d", &x);
8
9      /* ... */
10 }
```

Κανόνες Εμβέλειας Μεταβλητών

Μία μεταβλητή είναι γνωστή στο μπλόκ που δηλώνεται:

```
1  #include <stdio.h>
2
3  void fun() {
4      int x = 4;
5      /* a is NOT present here */
6  }
7
8  void more_fun() {
9      int a = 5;
10     /* x is not present here */
11 }
12
13 int main() {
14     /* a or x are NOT present here */
15
16     /* ... */
17 }
```

Μία μεταβλητή έχει εμβέλεια μέχρι να κλείσει το μπλοκ ({ }).

Κανόνες Εμβέλειας Μεταβλητών

Οι παράμετροι των συναρτήσεων θεωρούνται τοπικές μεταβλητές των συναρτήσεων και άρα είναι γνωστές μόνο στο μπλοκ της εκάστοτε συνάρτησης:

```
1  #include <stdio.h>
2
3  int min(int x, int y) {
4      /* a or b are NOT present here */
5      return (x<y)?x:y;
6  }
7
8  int max(int a, int b) {
9      /* x or y are NOT present here */
10     return (a>b)?a:b;
11 }
12
13 int main() {
14     /* a,b,x or y are not present here */
15
16     /* ... */
17 }
```

Μία μεταβλητή έχει εμβέλεια μέχρι να κλείσει το μπλοκ ({ }).

Κανόνες Εμβέλειας Μεταβλητών

Τοπικές μεταβλητές με ίδιο όνομα είναι διαφορετικές μεταβλητές.

```
1  #include <stdio.h>
2
3  int fun(int x) {
4      int a = 1;
5
6      /* a is different from a of morefun */
7  }
8
9  int morefun(int y) {
10     int a = 2;
11
12     /* a is different from a of fun */
13 }
14
15 int main() {
16     /* no variable a is present here */
17 }
```

Κανόνες Εμβέλειας Μεταβλητών

Μπορούμε να δηλώσουμε μεταβλητές έξω από οποιαδήποτε συνάρτηση (εμβέλεια αρχείου).

```
1  #include <stdio.h>
2
3  void fun() {
4      /* a is NOT present here */
5  }
6
7  int a = 5;
8
9  void more_fun() {
10     /* a IS present here */
11 }
12
13 int main() {
14     /* a IS present here */
15     /* some code */
16 }
```

Μια μεταβλητή με εμβέλεια αρχείου είναι γνωστή από το σημείο που δηλώνεται και κάτω.

Κανόνες Εμβέλειας Μεταβλητών

Σε περίπτωση που δηλωθεί μια τοπική μεταβλητή με ίδιο όνομα με μια μεταβλητή που έχει εμβέλεια αρχείου (global variable) τότε η τοπική επικαλύπτει την καθολική μεταβλητή.

```
1  #include <stdio.h>
2
3  int a = 5;
4
5  void fun() {
6      int a = 200;
7      /* a here is 200 and different from global a */
8      a++;
9  }
10
11 void morefun() {
12     /* a here has value 5 */
13     a++;
14 }
15
16 int main() {
17     fun();
18     morefun();
19     /* a here has value 6 */
20 }
```

Στατικές Μεταβλητές

Λέξη κλειδί **`static`**

Μπορούμε να δηλώσουμε πως μια μεταβλητή είναι στατική:

```
1 static int x;
2 static float f;
```

Το αποτέλεσμα εξαρτάται από την εμβέλεια της δήλωσης μας.

Καθολικές Στατικές Μεταβλητές

Μπορούμε να δηλώσουμε πως μια καθολική μεταβλητή (εμβέλεια αρχείου) είναι στατική:

```
1  #include <stdio.h>
2
3  static int x;
4
5  int main() {
6      /* ... */
7  }
```

Το οποίο σημαίνει πως η μεταβλητή έχει εμβέλεια μόνο το αρχείο στο οποίο δηλώνεται.

Δεν έχουμε μιλήσει ακόμη για πολλαπλά αρχεία. Είναι γενικά δυνατό να ορίζουμε μια μεταβλητή σε ένα αρχείο C και να την χρησιμοποιούμε σε άλλο αρχείο.

Τοπικές Στατικές Μεταβλητές

Έχουν την τοπική τους εμβέλεια αλλά έχουν στατική διάρκεια αποθήκευσης. Μόλις αρχικοποιηθούν παραμένουν στη μνήμη για όλη την διάρκεια του προγράμματος.

```
1  #include <stdio.h>
2
3  void foo() {
4      static int x = 0;
5
6      printf("function foo has been called %d time(s)\n", ++x);
7  }
8
9  int main() {
10     int i;
11     for(i = 0; i < 100; i++) {
12         foo();
13     }
14     return 0;
15 }
```

Κάθε φορά που καλούμε την συνάρτηση `foo` χρησιμοποιείται η ίδια μεταβλητή `x` η οποία διατηρεί την τιμή της.

Αναδρομή

Αναδρομή είναι η διαδικασία που μια συνάρτηση καλεί τον εαυτό της.

Πολλές φορές μπορούμε να υλοποιήσουμε κάτι πολύ πιο εύκολα χρησιμοποιώντας αυτήν την τεχνική.

Υπολογισμός του Παραγοντικού

Ο μαθηματικός τύπος του παραγοντικού είναι αναδρομικός:

$$n! = \begin{cases} 1 & \text{if } n = 0, \\ n \cdot (n-1)! & \text{if } n > 0. \end{cases}$$

Μπορούμε να γράψουμε απευθείας αυτό τον τύπο σε C:

```
1  int factorial(int n) {
2      if (n <= 0)
3          return 1;
4      return n * factorial(n-1);
5  }
```

Σειρά Fibonacci

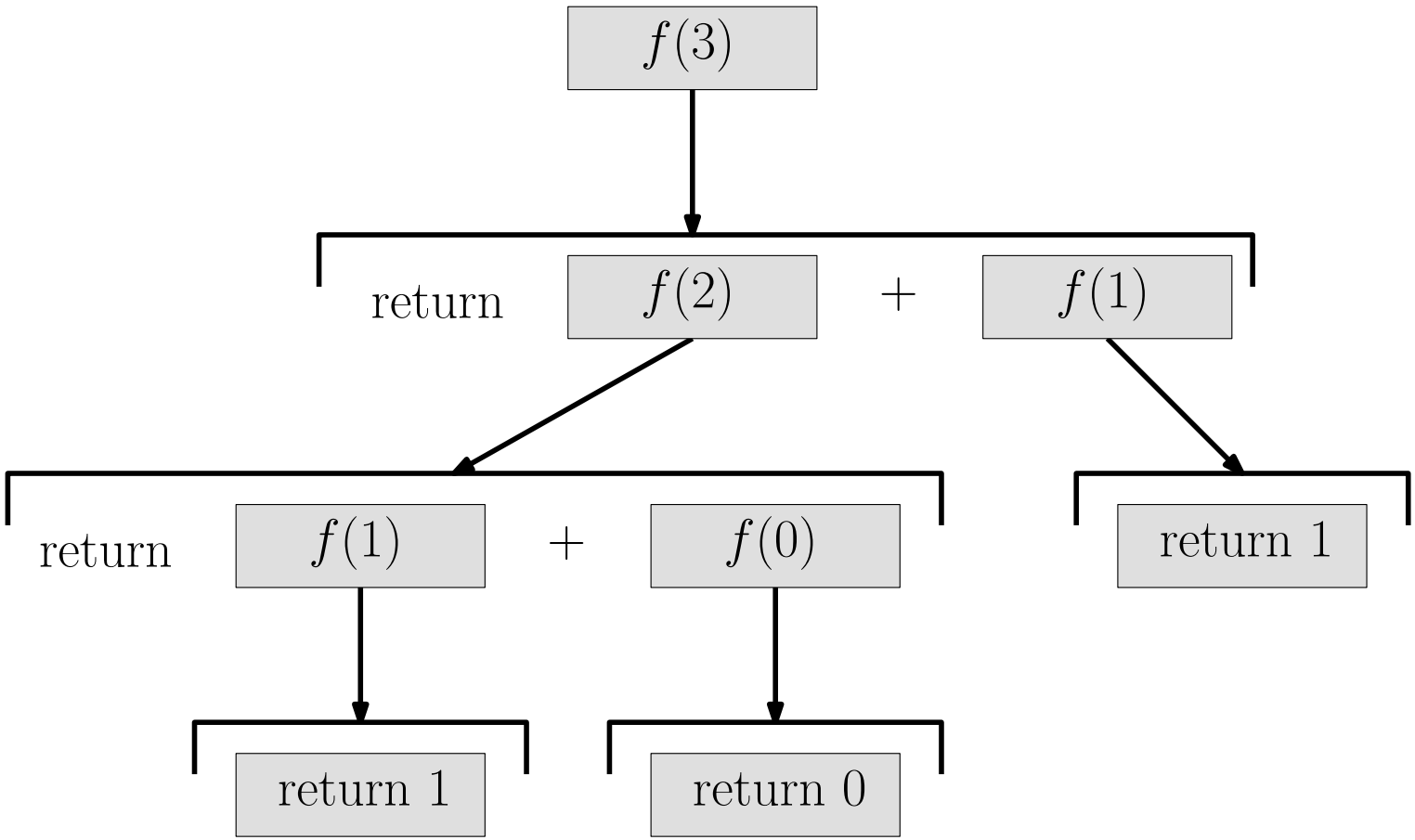
Η σειρά Fibonacci

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

αρχίζει με το 0 και το 1 και έχει την ιδιότητα πως κάθε επόμενος αριθμός είναι το άθροισμα των δύο προηγούμενων αριθμών Fibonacci.

```
1  #include <stdio.h>
2
3  long fibonacci(long);
4
5  int main() {
6     long result, number;
7
8     printf("Enter an integer: ");
9     scanf("%ld", &number);
10    result = fibonacci(number);
11    printf("Fibonacci( %ld ) = %ld\n", number, result);
12    return 0;
13 }
14
15 long fibonacci(long n) {
16     if (n == 0 || n == 1)
17         return n;
18     return fibonacci(n-1) + fibonacci(n-2);
19 }
```

Σειρά Fibonacci



Επανάληψη με Αναδρομή

Το γεγονός πως η κλήση των συναρτήσεων γίνεται με με call-by-value, επιτρέπει την χρήση της αναδρομής για να κάνουμε επανάληψη.

```
1  #include <stdio.h>
2
3  void backwards(int x) {
4      if (x < 0)
5          return;
6      printf("%d ", x);
7      backwards(x-1);
8  }
9
10 int main() {
11     backwards(100);
12     return 0;
13 }
```

Επανάληψη με Αναδρομή

Το γεγονός πως η κλήση των συναρτήσεων γίνεται με με call-by-value, επιτρέπει την χρήση της αναδρομής για να κάνουμε επανάληψη.

```
1  #include <stdio.h>
2
3  void backwards(int x) {
4      if (x < 0)
5          return;
6      printf("%d ", x);
7      backwards(x-1);
8  }
9
10 int main() {
11     backwards(100);
12     return 0;
13 }
```

Το παραπάνω πρόγραμμα τυπώνει

```
100 99 98 ... 2 1 0
```

Επανάληψη με Αναδρομή

Το γεγονός πως η κλήση των συναρτήσεων γίνεται με με call-by-value, επιτρέπει την χρήση της αναδρομής για να κάνουμε επανάληψη.

```
1  #include <stdio.h>
2
3  void backwards(int x) {
4      if (x < 0)
5          return;
6      printf("%d ", x);
7      backwards(x-1);
8      printf("%d ", x);
9  }
10
11 int main() {
12     backwards(100);
13     return 0;
14 }
```

Το παραπάνω πρόγραμμα τυπώνει

```
100 99 98 ... 2 1 0 0 1 2 ... 98 99 100
```

Επανάληψη με Αναδρομή

Έστω ο παρακάτω κώδικας που υπολογίζει το άθροισμα από 0 έως n.

```
1  #include <stdio.h>
2
3  int main() {
4      int i, n, sum;
5
6      scanf("%d", &n);
7
8      for(i = 0, sum = 0; i <= n; i++) {
9          sum += i;
10     }
11
12     printf("sum = %d\n", sum);
13     return 0;
14 }
```

Μπορούμε να το υλοποιήσουμε με την χρήση μίας αναδρομικής συνάρτησης, χωρίς επανάληψη;

Επανάληψη με Αναδρομή

Ο παρακάτω κώδικας υπολογίζει το άθροισμα από 0 έως n με την χρήση μιας αναδρομικής συνάρτησης.

```
1  #include <stdio.h>
2
3  int sum(int i) {
4      if (i == 0)
5          return i;
6      return sum(i-1) + i;
7  }
8
9  int main() {
10     int n;
11
12     scanf("%d", &n);
13
14     printf("sum = %d\n", sum(n));
15     return 0;
16 }
```

Ένα ακόμη παράδειγμα αναδρομής

```
1  #include <stdio.h>
2
3  void recursive_function(int n) {
4      if (n <= 0) {
5          printf("Called with non-positive n, stopping..\n");
6          return;
7      }
8      printf("Called with n = %d\n", n);
9      printf("Calling myself with n = %d\n", n-1);
10     recursive_function(n-1);
11     printf("Continuing call with n = %d\n", n);
12     printf("Call with n = %d finished, stopping..\n", n);
13 }
14
15 int main() {
16     recursive_function(5);
17     return 0;
18 }
```

Ένα ακόμη παράδειγμα αναδρομής

```
Called with n = 5
Calling myself with n = 4
Called with n = 4
Calling myself with n = 3
Called with n = 3
Calling myself with n = 2
Called with n = 2
Calling myself with n = 1
Called with n = 1
Calling myself with n = 0
Called with non-positive n, stopping..
Continuing call with n = 1
Call with n = 1 finished, stopping..
Continuing call with n = 2
Call with n = 2 finished, stopping..
Continuing call with n = 3
Call with n = 3 finished, stopping..
Continuing call with n = 4
Call with n = 4 finished, stopping..
Continuing call with n = 5
Call with n = 5 finished, stopping..
```

Γιατί έχουμε την παραπάνω έξοδο;