Δ.Π.Θ Δομημένος Προγραμματισμός

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΑΡΑΓΩΓΗΣ & ΔΙΟΙΚΗΣΗΣ

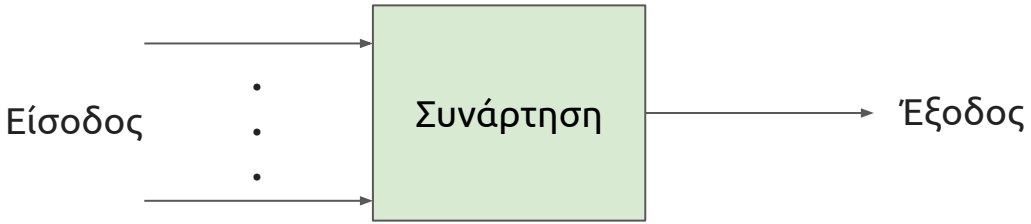
Συναρτήσεις

Dr. Αθανάσιος Μπαλαφούτης
Εργαστηριακό Διδακτικό Προσωπικό
Τομέας Συστημάτων Παραγωγής
Εργαστήριο Ρομποτικής και Αυτοματισμών
abalafou@pme.duth.gr
Γραφείο 304, τηλ.: 25410 – 79892

Ορισμός Συνάρτησης

Ένα ομαδοποιημένο σύνολο εντολών που:

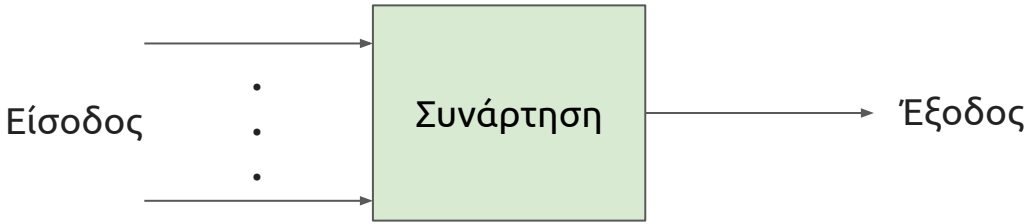
- Δέχεται καμία, μια ή περισσότερες εισόδους
- Εκτελεί υπολογισμούς
- Παράγει μια έξοδο



Ορισμός Συνάρτησης

Ένα ομαδοποιημένο σύνολο εντολών που:

- Δέχεται καμία, μια ή περισσότερες εισόδους
- Εκτελεί υπολογισμούς
- Παράγει μια έξοδο



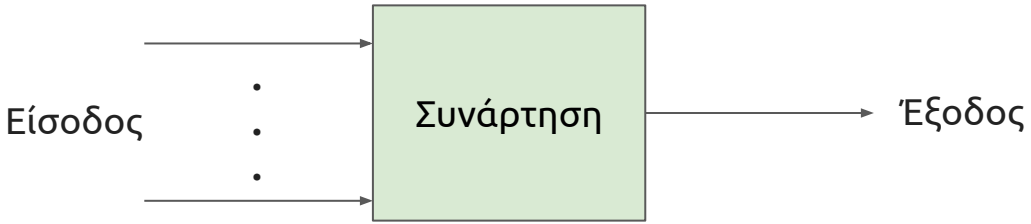
Σύνταξη Συνάρτησης:

```
Return_type function_name(set_of_inputs);
```

Ορισμός Συνάρτησης

Ένα ομαδοποιημένο σύνολο εντολών που:

- Δέχεται καμία, μια ή περισσότερες εισόδους
- Εκτελεί υπολογισμούς
- Παράγει μια έξοδο



Σύνταξη Συνάρτησης:

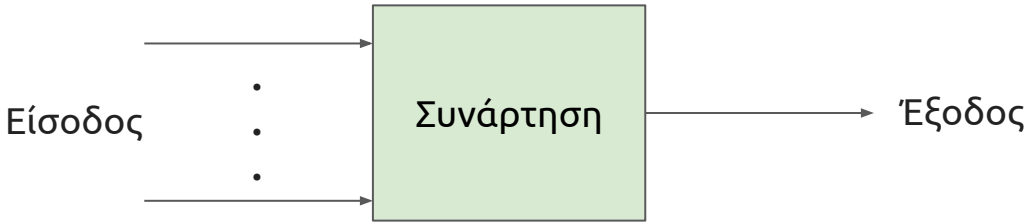
```
Return_type function_name(set_of_inputs);
```

Ο τύπος (int, float, double) της εξόδου που θα επιστρέψει η συνάρτηση

Ορισμός Συνάρτησης

Ένα ομαδοποιημένο σύνολο εντολών που:

- Δέχεται καμία, μια ή περισσότερες εισόδους
- Εκτελεί υπολογισμούς
- Παράγει μια έξοδο



Σύνταξη Συνάρτησης:

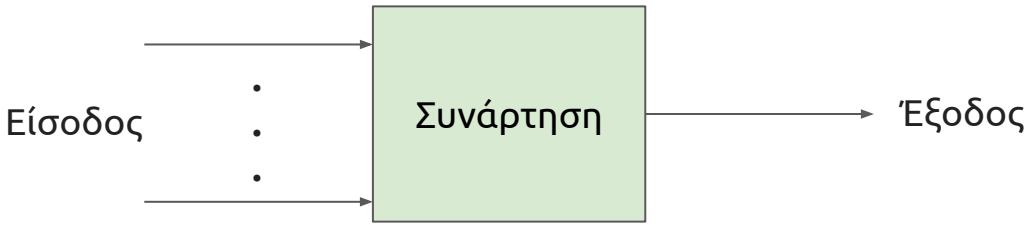
```
Return_type function_name(set_of_inputs);
```

Το όνομα της συνάρτησης

Ορισμός Συνάρτησης

Ένα ομαδοποιημένο σύνολο εντολών που:

- Δέχεται καμία, μια ή περισσότερες εισόδους
- Εκτελεί υπολογισμούς
- Παράγει μια έξοδο



Σύνταξη Συνάρτησης:

```
Return_type function_name(set_of_inputs);
```

Οι μεταβλητές που θα δεχτεί η συνάρτηση ως είσοδο

Τύποι Συναρτήσεων

Οι Συναρτήσεις, με βάση τα ορίσματα που δέχονται μπορούν να έχουν μια από τις παρακάτω μορφές:

- Συναρτήσεις χωρίς ορίσματα και χωρίς επιστρεφόμενη τιμή
- Συναρτήσεις χωρίς ορίσματα και επιστρεφόμενη τιμή
- Συναρτήσεις με ορίσματα και χωρίς τιμή επιστροφής
- Συναρτήσεις με ορίσματα και με τιμή επιστροφής

Τύποι Συναρτήσεων

Οι Συναρτήσεις, με βάση τα ορίσματα που δέχονται μπορούν να έχουν μια από τις παρακάτω μορφές:

- Συναρτήσεις χωρίς ορίσματα και χωρίς επιστρεφόμενη τιμή `exit()`
- Συναρτήσεις χωρίς ορίσματα και επιστρεφόμενη τιμή
- Συναρτήσεις με ορίσματα και χωρίς τιμή επιστροφής
- Συναρτήσεις με ορίσματα και με τιμή επιστροφής

Τύποι Συναρτήσεων

Οι Συναρτήσεις, με βάση τα ορίσματα που δέχονται μπορούν να έχουν μια από τις παρακάτω μορφές:

- Συναρτήσεις χωρίς ορίσματα και χωρίς επιστρεφόμενη τιμή `exit()`
- Συναρτήσεις χωρίς ορίσματα και επιστρεφόμενη τιμή `int random_number = rand()`
- Συναρτήσεις με ορίσματα και χωρίς τιμή επιστροφής
- Συναρτήσεις με ορίσματα και με τιμή επιστροφής

Τύποι Συναρτήσεων

Οι Συναρτήσεις, με βάση τα ορίσματα που δέχονται μπορούν να έχουν μια από τις παρακάτω μορφές:

- Συναρτήσεις χωρίς ορίσματα και χωρίς επιστρεφόμενη τιμή `exit()`
- Συναρτήσεις χωρίς ορίσματα και επιστρεφόμενη τιμή `int random_number = rand()`
- Συναρτήσεις με ορίσματα και χωρίς τιμή επιστροφής `printf("Hello!\n")`
- Συναρτήσεις με ορίσματα και με τιμή επιστροφής

Τύποι Συναρτήσεων

Οι Συναρτήσεις, με βάση τα ορίσματα που δέχονται μπορούν να έχουν μια από τις παρακάτω μορφές:

- Συναρτήσεις χωρίς ορίσματα και χωρίς επιστρεφόμενη τιμή `exit()`
- Συναρτήσεις χωρίς ορίσματα και επιστρεφόμενη τιμή `int random_number = rand()`
- Συναρτήσεις με ορίσματα και χωρίς τιμή επιστροφής `printf("Hello!\n")`
- Συναρτήσεις με ορίσματα και με τιμή επιστροφής `double result = pow(base, exponent)`

Γιατί χρησιμοποιούμε Συναρτήσεις;

- **Επαναχρησιμοποίηση κώδικα**

  Όταν οριστεί μια συνάρτηση, μπορεί να χρησιμοποιηθεί (κληθεί) σε πολλά διαφορετικά σημεία του ίδιου προγράμματος ή ακόμη και σε διαφορετικά προγράμματα (ως βιβλιοθήκη)

- **Αφαιρετικότητα**

  Μας επιτρέπει να αποκρύψουμε τις λεπτομέρειες υλοποίησης μιας ιδέας ή αλγορίθμου, και να επικεντρωθούμε στη μεγάλη εικόνα του συνολικού προβλήματος που προσπαθούμε να επιλύσουμε.

Π.χ θυμηθείτε τις συναρτήσεις: **`scanf, printf, abs, pow`**

Παράδειγμα

Να γράψετε πρόγραμμα που να διαβάζει τα μήκη των πλευρών ενός παραλληλογράμμου και να εμφανίζει το εμβαδόν του.

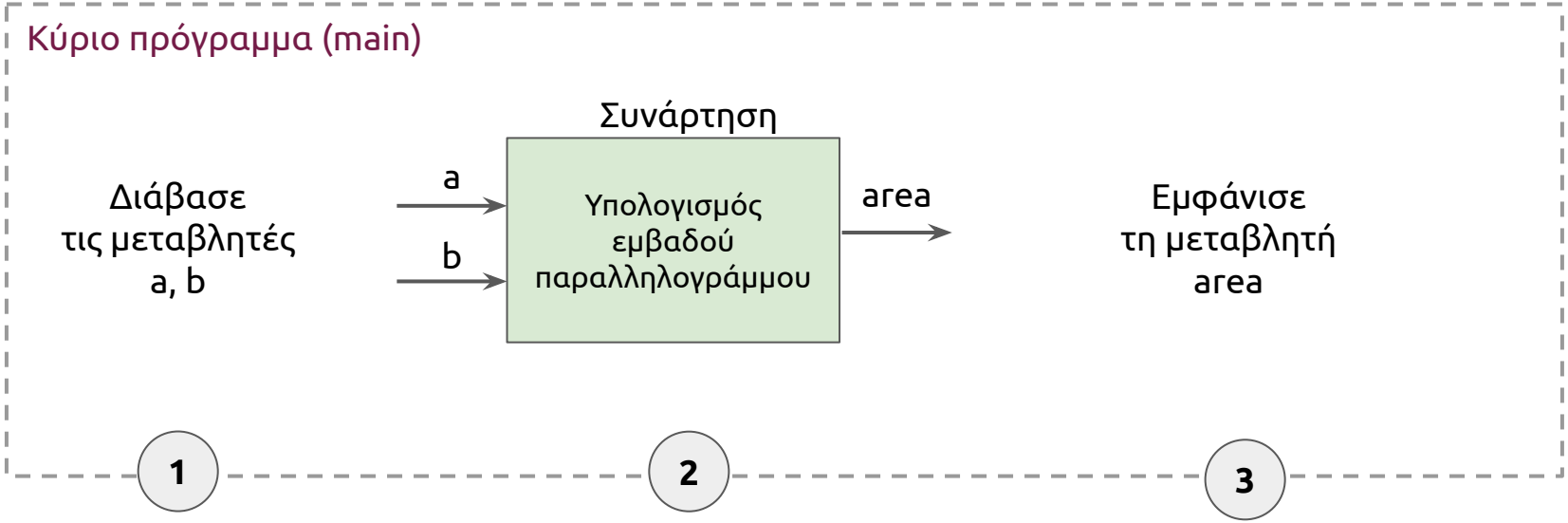
Παράδειγμα

Να γράψετε πρόγραμμα που να διαβάζει τα μήκη των πλευρών ενός παραλληλογράμμου και να εμφανίζει το εμβαδόν του.

```c
#include <stdio.h>

int main() {
	float a, b, area;

	printf("Δώσε τα μήκη των πλευρών του παραλληλογράμμου:");
	scanf("%f %f", &a, &b);

	area = a * b;

	printf("Το εμβαδό του παραλληλογράμμου είναι: %.2f \n", area);
	return 0;
}
```

Παράδειγμα

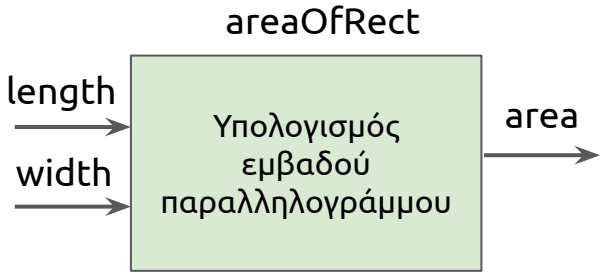
Θα λύσουμε το ίδιο πρόβλημα αξιοποιώντας τις συναρτήσεις



Παράδειγμα - Ορισμός Συνάρτησης

Ας γράψουμε πρώτα τη γενική συνάρτηση

```c
#include <stdio.h>

float areaOfRect(float length, float width){
   float area;
   area = length * width;
   return area;
}
```



Παράδειγμα - Κλήση Συνάρτησης


```c
#include <stdio.h>

float areaOfRect(float length, float width){
   float area;
   area = length * width;
   return area;
}

int main() {
       float a, b, area;

       printf("Δώσε τα μήκη των πλευρών του παραλληλογράμμου:");
       scanf("%f %f", &a, &b);
       area = areaOfRect(a, b);
       printf("Το εμβαδό του παραλληλογράμμου είναι: %.2f \n", area);
       return 0;
}
```

Δήλωση Συναρτήσεων

Όπως ήδη γνωρίζουμε, η **δήλωση μιας μεταβλητής** σε ένα πρόγραμμα, αποσαφηνίζει στον **μεταφραστή** (compiler) τις **ιδιότητες** που θέλουμε να έχει αυτή η μεταβλητή .

Παράδειγμα δήλωσης μεταβλητής:

```
int var;
```

Ιδιότητες μεταβλητής:

1. Όνομα μεταβλητής: `var`
2. Τύπος μεταβλητής: `int`

Δήλωση Συναρτήσεων

Αντίστοιχα, η **δήλωση μιας Συνάρτησης** (function prototype) σε ένα πρόγραμμα, αποσαφηνίζει στον **μεταφραστή** (compiler) τις **ιδιότητες** που θέλουμε να έχει αυτή η Συνάρτηση .

Παράδειγμα δήλωσης Συνάρτησης:

```
int fun(int, char);
```

Ιδιότητες μεταβλητής:

1. Όνομα Συνάρτησης: `fun`
2. Τύπος επιστρεφόμενης τιμής: `int`
3. Πλήθος παραμέτρων εισόδου: `2`
4. Τύπος 1ης παραμέτρου: `int`
5. Τύπος 2ης παραμέτρου: `char`

Δήλωση Συναρτήσεων

Αντίστοιχα, η **δήλωση μιας Συνάρτησης** (function prototype) σε ένα πρόγραμμα, αποσαφηνίζει στον **μεταφραστή** (compiler) τις **ιδιότητες** που θέλουμε να έχει αυτή η Συνάρτηση .

Παράδειγμα δήλωσης Συνάρτησης:

```
int fun(int, char);
```

Ιδιότητες μεταβλητής:

1. Όνομα Συνάρτησης: `fun`
2. Τύπος επιστρεφόμενης τιμής: `int`
3. Πλήθος παραμέτρων εισόδου: `2`
4. Τύπος 1ης παραμέτρου: `int`
5. Τύπος 2ης παραμέτρου: `char`

Η δήλωση μιας συνάρτησης, μπορεί να παραληφθεί αν η συνάρτηση οριστεί **πριν** από τη main()

Παράδειγμα - Δήλωση Συνάρτησης


```c
#include <stdio.h>
float areaOfRect(float, float);

int main() {
	float a, b, area;

	printf("Δώσε τα μήκη των πλευρών του παραλληλογράμμου:");
	scanf("%f %f", &a, &b);
	area = areaOfRect(a, b);
	printf("Το εμβαδό του παραλληλογράμμου είναι: %.2f \n", area);
	return 0;
}
float areaOfRect(float length, float width){
   float area;
   area = length * width;
   return area;
}
```

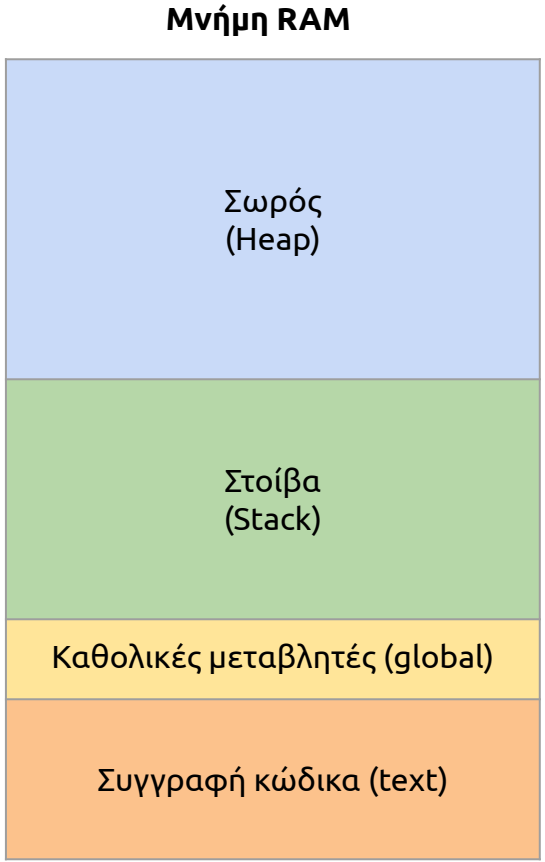
Τυπικές και Πραγματικές Παράμετροι


```c
#include <stdio.h>
float areaOfRect(float, float);

int main() {
        float a, b, area;

        printf("Δώσε τα μήκη των πλευρών του παραλληλογράμμου:");
        scanf("%f %f", &a, &b);
        area = areaOfRect(a, b);
        printf("Το εμβαδό του παραλληλογράμμου είναι: %.2f \n", area);
        return 0;
}
float areaOfRect(float length, float width){
   float area;
   area = length * width;
   return area;
}
```

Πραγματικές

Τυπικές

Διαχείριση Κύριας Μνήμης



Διαχείριση Κύριας Μνήμης


```c
#include <stdio.h>

int total;

int Square(int x){
    return x * x;
}

int SquareOfSum(int x, int y){
    int z = Square(x + y);
    return z;
}

int main(){
    int a = 4;
    int b = 8;
    total = SquareOfSum(a, b);
    printf("%d\n", total);
}
```

Στοίβα (Stack)

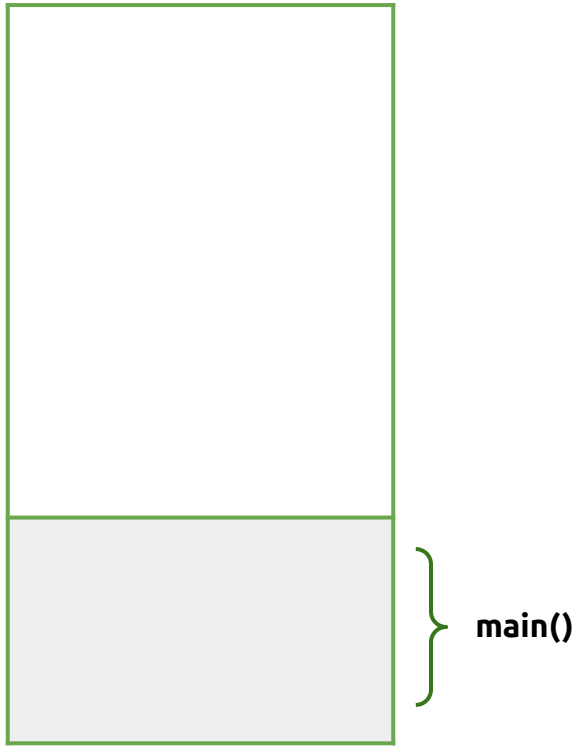
Καθολικές Μεταβλητές (Global)

total = ?

Διαχείριση Κύριας Μνήμης


```c
#include <stdio.h>

int total;

int Square(int x){
    return x * x;
}

int SquareOfSum(int x, int y){
    int z = Square(x + y);
    return z;
}

int main(){
    int a = 4;
    int b = 8;
    total = SquareOfSum(a, b);
    printf("%d\n", total);
}
```

Στοίβα (Stack)



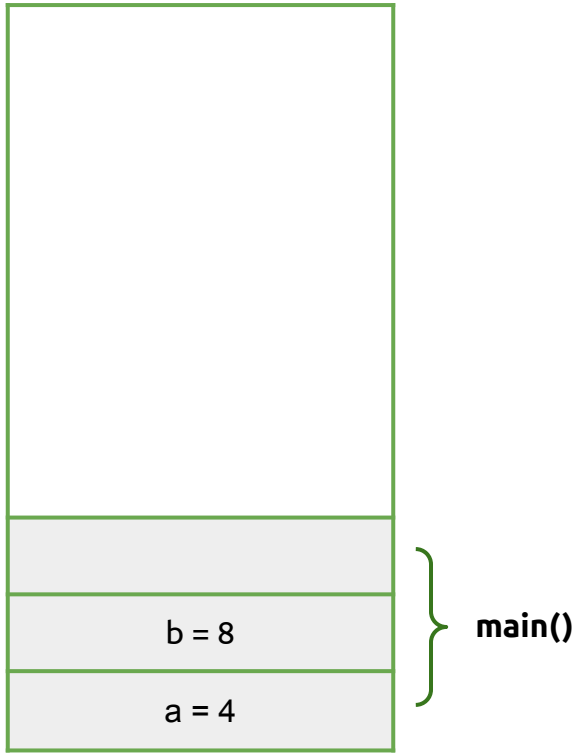
Καθολικές Μεταβλητές (Global)

total = ?

Διαχείριση Κύριας Μνήμης


```c
#include <stdio.h>

int total;

int Square(int x){
    return x * x;
}

int SquareOfSum(int x, int y){
    int z = Square(x + y);
    return z;
}

int main(){
    int a = 4;
    int b = 8;
    total = SquareOfSum(a, b);
    printf("%d\n", total);
}
```

Στοίβα (Stack)



Καθολικές Μεταβλητές (Global)

total = ?

Διαχείριση Κύριας Μνήμης

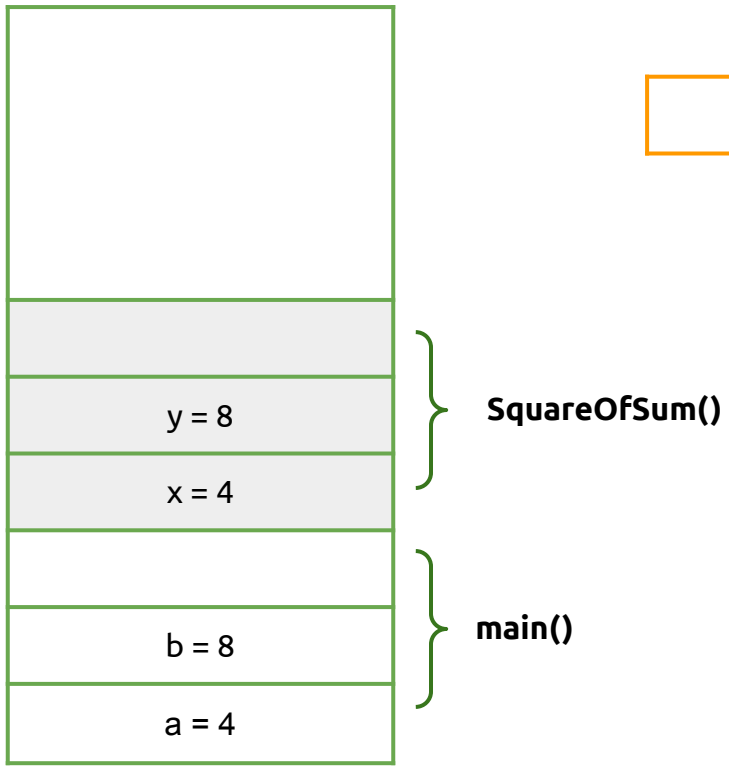

```c
#include <stdio.h>

int total;

int Square(int x){
    return x * x;
}

int SquareOfSum(int x, int y){
    int z = Square(x + y);
    return z;
}

int main(){
    int a = 4;
    int b = 8;
    total = SquareOfSum(a, b);
    printf("%d\n", total);
}
```


Καθολικές Μεταβλητές (Global)

total = ?

Διαχείριση Κύριας Μνήμης


```c
#include <stdio.h>

int total;

int Square(int x){
    return x * x;
}

int SquareOfSum(int x, int y){
    int z = Square(x + y);
    return z;
}

int main(){
    int a = 4;
    int b = 8;
    total = SquareOfSum(a, b);
    printf("%d\n", total);
}
```

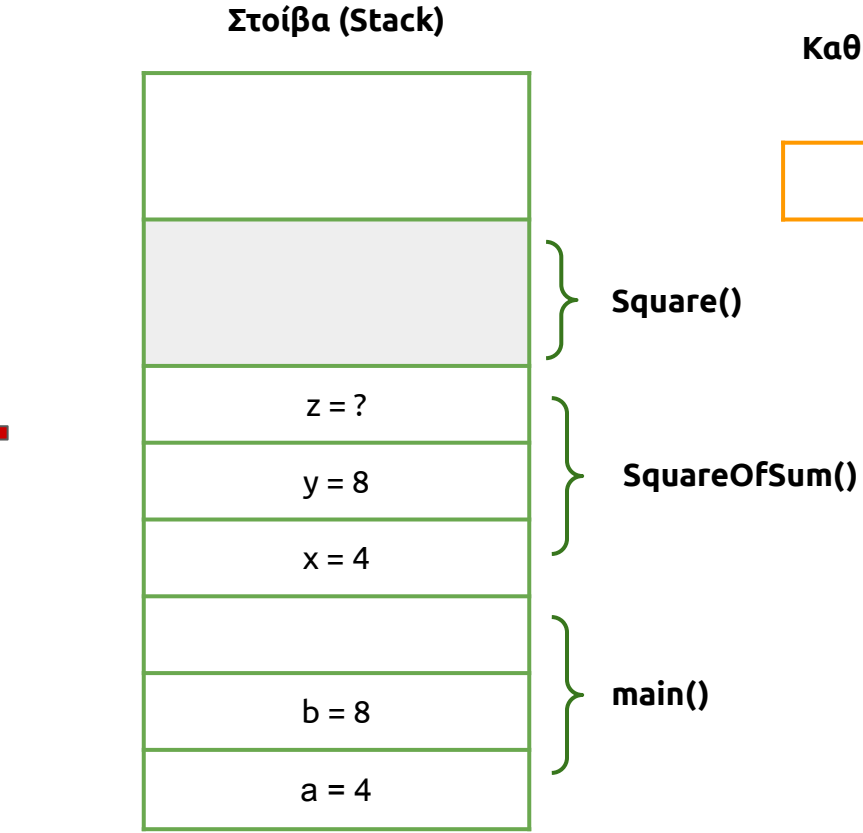


Καθολικές Μεταβλητές (Global)

total = ?

Διαχείριση Κύριας Μνήμης


```c
#include <stdio.h>

int total;

int Square(int x){
    return x * x;
}

int SquareOfSum(int x, int y){
    int z = Square(x + y);
    return z;
}

int main(){
    int a = 4;
    int b = 8;
    total = SquareOfSum(a, b);
    printf("%d\n", total);
}
```

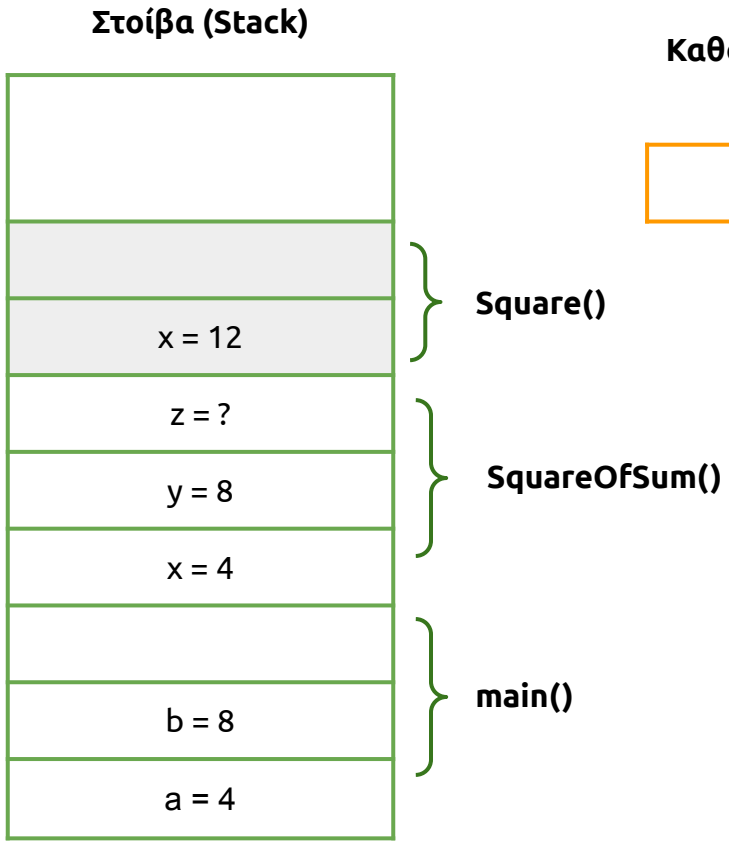


Καθολικές Μεταβλητές (Global)

total = ?

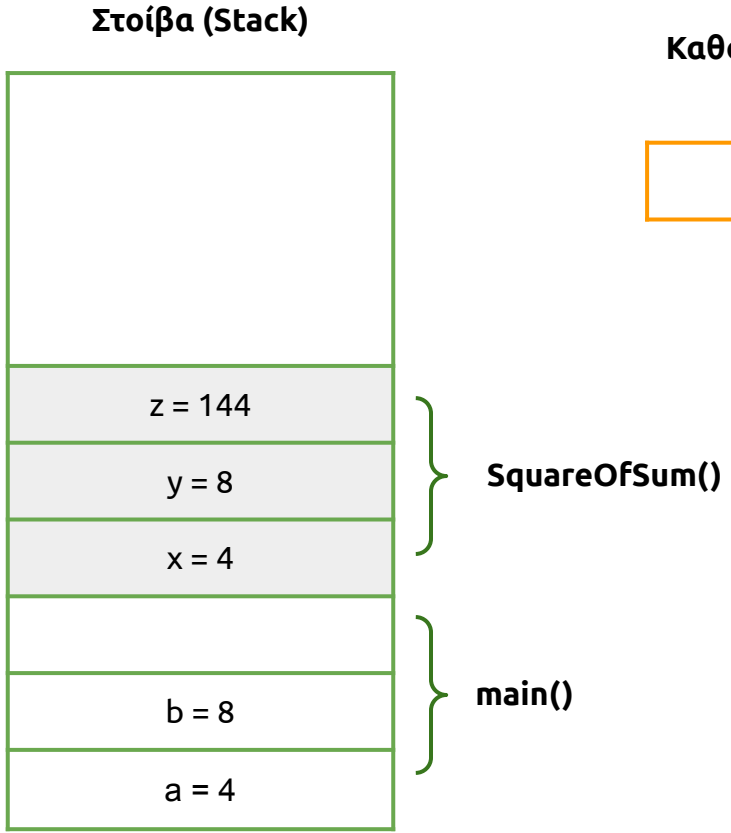
Διαχείριση Κύριας Μνήμης


```c
#include <stdio.h>

int total;

int Square(int x){
    return x * x;
}

int SquareOfSum(int x, int y){
    int z = Square(x + y);
    return z;
}

int main(){
    int a = 4;
    int b = 8;
    total = SquareOfSum(a, b);
    printf("%d\n", total);
}
```



Διαχείριση Κύριας Μνήμης

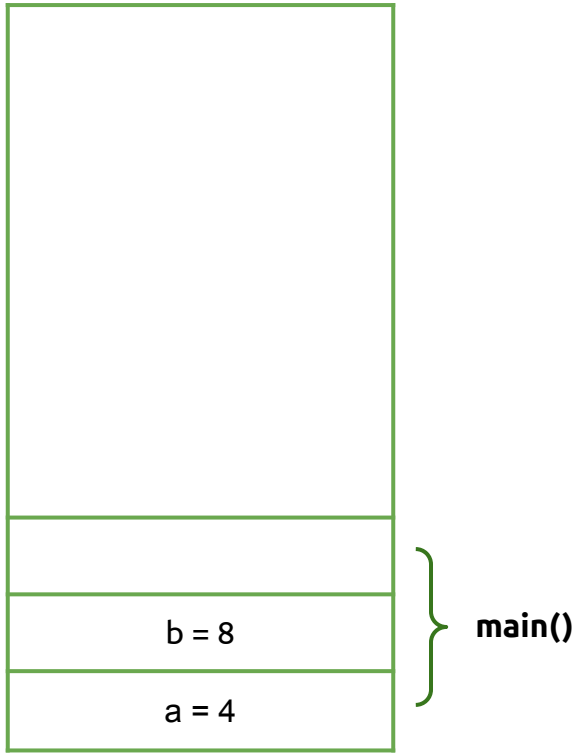

```c
#include <stdio.h>

int total;

int Square(int x){
    return x * x;
}

int SquareOfSum(int x, int y){
    int z = Square(x + y);
    return z;
}

int main(){
    int a = 4;
    int b = 8;
    total = SquareOfSum(a, b);
    printf("%d\n", total);
}
```



Διαχείριση Κύριας Μνήμης


```c
#include <stdio.h>

int total;

int Square(int x){
    return x * x;
}

int SquareOfSum(int x, int y){
    int z = Square(x + y);
    return z;
}

int main(){
    int a = 4;
    int b = 8;
    total = SquareOfSum(a, b);
    printf("%d\n", total);
}
```

Στοίβα (Stack)

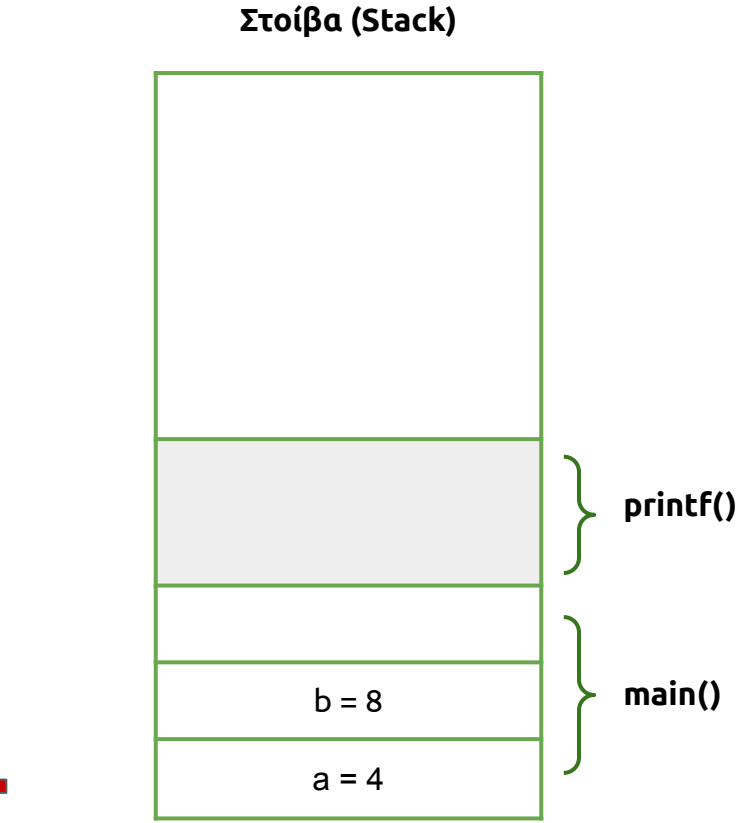


Καθολικές Μεταβλητές (Global)

total = 144

Διαχείριση Κύριας Μνήμης


```c
#include <stdio.h>

int total;

int Square(int x){
    return x * x;
}

int SquareOfSum(int x, int y){
    int z = Square(x + y);
    return z;
}

int main(){
    int a = 4;
    int b = 8;
    total = SquareOfSum(a, b);
    printf("%d\n", total);
}
```



Καθολικές Μεταβλητές (Global)

total = 144

Διαχείριση Κύριας Μνήμης


```c
#include <stdio.h>

int total;

int Square(int x){
    return x * x;
}

int SquareOfSum(int x, int y){
    int z = Square(x + y);
    return z;
}

int main(){
    int a = 4;
    int b = 8;
    total = SquareOfSum(a, b);
    printf("%d\n", total);
}
```

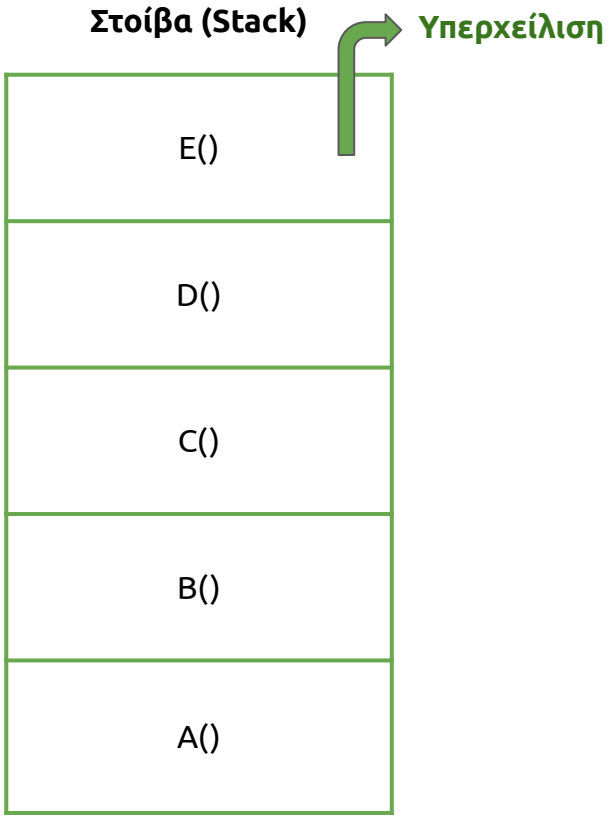
Στοίβα (Stack)

Καθολικές Μεταβλητές (Global)

Υπερχείλιση Στοίβας (Stack overflow)

Κατά την εκτέλεση ενός προγράμματος, το λειτουργικό σύστημα δεσμεύει σταθερή ποσότητα μνήμης για τη Στοίβα. (π.χ. 1MB).

Αν στο πρόγραμμά υπάρχουν πολλές ενεργές κλήσεις συναρτήσεων (π.χ. Αναδρομικές συναρτήσεις), μπορεί να συμβεί υπερχείλιση της στοίβας και το πρόγραμμά μας να σταματήσει να εκτελείται.



Συναρτήσεις που δεν επιστρέφουν τιμές

Η μεταβλητή sum, επιστρέφει στη main()

```c
#include <stdio.h>

int addNumbers(int a, int b) {
    int sum ;
    sum = a + b;
    return sum;
}

int main() {
    int sum;
    sum = addNumbers(5, 7);
    printf("Το άθροισμα είναι: %d\n", sum);
    return 0;
}
```

Συναρτήσεις που δεν επιστρέφουν τιμές

Η μεταβλητή sum, επιστρέφει στη main()

```c
#include <stdio.h>

int addNumbers(int a, int b) {
    int sum;
    sum = a + b;
    return sum;
}

int main() {
    int sum;
    sum = addNumbers(5, 7);
    printf("Το άθροισμα είναι: %d\n", sum);
    return 0;
}
```

Η μεταβλητή sum, **ΔΕΝ** επιστρέφει στη main()

```c
#include <stdio.h>

void addNumbers(int a, int b) {
    int sum = a + b;
    printf("Το άθροισμα είναι: %d\n", sum);
}

int main() {
    addNumbers(5, 7);
    return 0;
}
```

Τι θα εκτυπώσει το πρόγραμμα;


```c
#include <stdio.h>

void Increment(int a) {
   a = a + 1;
}

int main() {
   int a;
   a = 10;
   Increment(a);
   printf("a = %d\n", a);
   return 0;
}
```

a = ?

Τι θα εκτυπώσει το πρόγραμμα;


```c
#include <stdio.h>

void Increment(int a) {
   a = a + 1;
}

int main() {
   int a;
   a = 10;
   Increment(a);
   printf("a = %d\n", a);
   return 0;
}
```

a = **10**

Τι θα εκτυπώσει το πρόγραμμα;

Η μεταβλητή **a** παραμένει 10

```c
#include <stdio.h>

void Increment(int a) {
   a = a + 1;
}

int main() {
   int a;
   a = 10;
   Increment(a);
   printf("a = %d\n", a);
   return 0;
}
```

Η μεταβλητή **a** γίνεται 11

```c
#include <stdio.h>

int Increment(int a) {
  return a + 1;
}

int main() {
  int a;
  a = 10;
  a = Increment(a);
  printf("a = %d\n", a);
  return 0;
}
```

Συναρτήσεις χωρίς παραμέτρους


```c
#include <stdio.h>

void printHello() {
  printf("Hello\n");
}

int main() {
  printHello(); // Κλήση Συνάρτησης
  return 0;
}
```

Άσκηση 1.1 - Τυχαίοι Αριθμοί

Να γράψετε συνάρτηση με όνομα myRand, που να δέχεται ως παραμέτρους, δύο ακέραιους αριθμούς. Οι παράμετροι αυτοί αντιστοιχούν στον μικρότερο και μεγαλύτερο τυχαίο ακέραιο αριθμό που θέλουμε να παράγει η συνάρτηση αυτή.

Η συνάρτηση θα επιστρέφει έναν ακέραιο αριθμό στο επιθυμητό εύρος.

Στη συνέχεια, να γράψετε πρόγραμμα που να διαβάζει 10 τυχαίους αριθμούς στο διάστημα [10, 99], καλώντας τη συνάρτηση myRand και να εμφανίζει το μεγαλύτερο από αυτούς.

Άσκηση 1.1 - Τυχαίοι Αριθμοί


```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int myRand(int lower, int upper) {
   int rand_num;

   rand_num = rand() % (upper - lower + 1) + lower;

   return rand_num;
}
```

Άσκηση 1.1 - Τυχαίοι Αριθμοί


```
int main() {
  int r, max = -1;
  srand(time(0));

  for(int i = 0; i < 10; i++){
    r = myRand(10, 99);
    printf("%d \n", r);
    if (r > max)
       max = r;
  }
  printf("Ο μεγαλύτερος τυχαίος αριθμός είναι: %d \n", max);
  return 0;
}
```

Άσκηση 1.1 - Τυχαίοι Αριθμοί

Τι θα συμβεί αν αρχικοποιήσω τη γεννήτρια τυχαίων αριθμών μέσα στη συνάρτηση;

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int myRand(int lower, int upper) {
   int rand_num;

   rand_num = rand() % (upper - lower + 1) + lower;

   return rand_num;
}
```

```c
srand(time(0));
```

Άσκηση 1.1 - Τυχαίοι Αριθμοί

Τι θα συμβεί αν αρχικοποιήσω τη γεννήτρια τυχαίων αριθμών μέσα στη συνάρτηση;

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int myRand(int lower, int upper) {
   int rand_num;

   rand_num = rand() % (upper - lower + 1) + lower;

   return rand_num;
}
```

```c
srand(time(0));
```

Άσκηση 1.1 - Τυχαίοι Αριθμοί

Τι θα συμβεί αν αρχικοποιήσω τη γεννήτρια τυχαίων αριθμών μέσα στη συνάρτηση;

Πρόβλημα

- Κάθε φορά που καλείται η συνάρτηση **myRand**, θα επαναρχικοποιείται η γεννήτρια τυχαίων αριθμών με βάση τον χρόνο.
- Δεδομένου ότι οι διαδοχικές κλήσεις στη **myRand** γίνονται πολύ γρήγορα (συνήθως εντός του ίδιου δευτερολέπτου), η συνάρτηση **time(0)** θα επιστρέφει την ίδια τιμή.
- Ως αποτέλεσμα, οι τυχαίοι αριθμοί δεν θα είναι πραγματικά τυχαίοι, αλλά θα επαναλαμβάνονται.

Δημιουργία αρχείου Βιβλιοθήκης

Μπορώ να επαναχρησιμοποιήσω τη συνάρτηση **myRand**, σε άλλα προγράμματα, χωρίς να πρέπει να την ορίσω ξανά;

Δημιουργία αρχείου Βιβλιοθήκης

Μπορώ να επαναχρησιμοποιήσω τη συνάρτηση **myRand**, σε άλλα προγράμματα, χωρίς να πρέπει να την ορίσω ξανά;

Βήμα 1: Δημιουργία αρχείου myRand .h (αρχείο επικεφαλίδας - header file)

```
#ifndef MYRAND_H
#define MYRAND_H

// Δήλωση της συνάρτησης
int myRand(int min, int max);

#endif
```

Δημιουργία αρχείου Βιβλιοθήκης

Μπορώ να επαναχρησιμοποιήσω τη συνάρτηση **myRand**, σε άλλα προγράμματα, χωρίς να πρέπει να την ορίσω ξανά;

Βήμα 1: Δημιουργία αρχείου myRand .h (αρχείο επικεφαλίδας - header file)

```
#ifndef MYRAND_H
#define MYRAND_H

// Δήλωση της συνάρτησης
int myRand(int min, int max);

#endif
```

Μηχανισμός Προφύλαξης πολλαπλών ορισμών

Χωρίς αυτήν την προστασία, αν το αρχείο myRand.h εισαχθεί περισσότερες από μία φορές (π.χ., από διαφορετικά αρχεία που το περιλαμβάνουν), θα προκαλούσε **σφάλματα πολλαπλών ορισμών** (multiple definition errors), όπως:

- Πολλαπλή δήλωση ή υλοποίηση της ίδιας συνάρτησης.
- Πολλαπλή δήλωση της ίδιας μεταβλητής ή σταθεράς.

Δημιουργία αρχείου Βιβλιοθήκης

Μπορώ να επαναχρησιμοποιήσω τη συνάρτηση **myRand**, σε άλλα προγράμματα, χωρίς να πρέπει να την ορίσω ξανά;

Βήμα 2: Δημιουργία αρχείου myRand.c (αρχείο υλοποίησης συνάρτησης)

```c
#include "myRand.h"
#include <stdlib.h>

// Υλοποίηση της συνάρτησης
int myRand(int min, int max) {
   return min + rand() % (max - min + 1);
}
```

Δημιουργία αρχείου Βιβλιοθήκης

Τεχνικές λεπτομέρειες, διασύνδεσης των αρχείων - μεταγλώττιση - εκτέλεση…

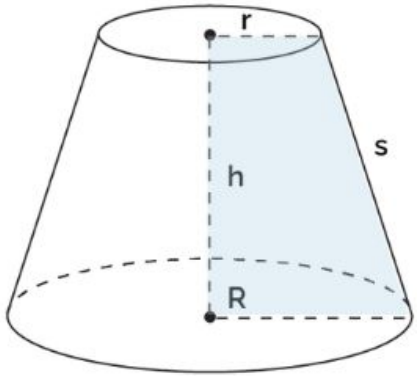
… στο Εργαστήριο

Άσκηση 1.2

Να γράψετε 2 συναρτήσεις για τον υπολογισμό της επιφάνειας και του όγκου του σχήματος:

$$S = \pi r^2 + \pi R^2 + \pi(R+r)s \quad \text{, όπου} \quad s = \sqrt{h^2 + (R-r)^2}$$

$$V = \frac{1}{3}\pi h \left(R^2 + Rr + r^2\right)$$



Στη συνέχεια να γράψετε ένα πρόγραμμα σε γλώσσα C που θα καλεί τις συναρτήσεις και θα εμφανίζει τα αποτελέσματα που επιστρέφει η κάθε συνάρτηση.

Άσκηση 1.2 - Συνάρτηση surface

$$S = \pi r^2 + \pi R^2 + \pi(R+r)s \quad \text{, όπου} \quad s = \sqrt{h^2 + (R-r)^2}$$

```c
#include <stdio.h>
#include <math.h>
#define PI 3.14159

float surface(float R, float r, float h) {
  float s, S;
  s = sqrt(pow(h, 2) + pow(R - r, 2));
  S = PI * pow(r, 2) + PI * pow(R, 2) + PI * (R + r) * s;

  return S;
}
```

Άσκηση 1.2 - Συνάρτηση volume

$$V = \frac{1}{3}\pi h \left(R^2 + Rr + r^2\right)$$

```
float volume(float R, float r, float h) {
  float V;

  V = 1 / 3 * PI * h * (pow(R, 2) + R * r + pow(r, 2));

  return V;
}
```

Βρείτε το λάθος !

Άσκηση 1.2 - Συνάρτηση volume

$$V = \frac{1}{3}\pi h \left(R^2 + Rr + r^2\right)$$

```
float volume(float R, float r, float h) {
  float V;

  V = 1.0 / 3.0 * PI * h * (pow(R, 2) + R * r + pow(r, 2));

  return V;
}
```

Άσκηση 1.2 - Συνάρτηση main


```c
int main() {
  float R, r, h, S, V;

  printf("Δώσε τα R, r, h:");
  scanf("%f %f %f", &R, &r, &h);

  S = surface(R, r, h);
  V = volume(R, r, h);

  printf("Επιφάνεια: %.2f\n", S);
  printf("Όγκος: %.2f\n", V);

  return 0;
}
```

Άσκηση 1.3

Σε μια βιομηχανική μονάδα παραγωγής, το κόστος κατασκευής ενός προϊόντος εξαρτάται από:

- Το κόστος πρώτων υλών
- Το κόστος εργασίας
- Τα γενικά έξοδα παραγωγής (π.χ. ηλεκτρικό ρεύμα, μηχανήματα, συντήρηση, κτλ.)

Η τιμή πώλησης του προϊόντος καθορίζεται από το κόστος παραγωγής του, προσθέτοντας ένα ποσοστό κέρδους. Να γράψετε ένα πρόγραμμα που θα υλοποιεί τις εξής συναρτήσεις:

1. `calcProductionCost`:
    - Δέχεται ως ορίσματα το κόστος πρώτων υλών, το κόστος εργασίας και τα γενικά έξοδα.
    - Επιστρέφει το συνολικό κόστος παραγωγής.
2. `calcSellingPrice`:

- Δέχεται ως ορίσματα το κόστος παραγωγής και το ποσοστό κέρδους.
- Καλεί τη συνάρτηση `calcProductionCost` για να υπολογίσει το κόστος παραγωγής.
- Επιστρέφει την τελική τιμή πώλησης του προϊόντος.

Το κύριο πρόγραμμα θα διαβάζει τα απαραίτητα δεδομένα από τον χρήστη, θα καλεί τις συναρτήσεις και θα εμφανίζει την τελική τιμή πώλησης του προϊόντος.

Άσκηση 1.3 - Συναρτήσεις


```c
#include <stdio.h>


double calcProductionCost(double rawMaterials, double laborCost, double overhead) {
   return rawMaterials + laborCost + overhead;
}
```

Άσκηση 1.3 - Συναρτήσεις


```c
#include <stdio.h>


double calcProductionCost(double rawMaterials, double laborCost, double overhead) {
    return rawMaterials + laborCost + overhead;
}

double calcSellingPrice(double rawMaterials, double laborCost, double overhead,
double profit)
{
    // Κλήση της συνάρτησης calcProductionCost
    double productionCost = calcProductionCost(rawMaterials, laborCost, overhead);

    // Υπολογισμός τιμής πώλησης
    return productionCost * (1 + profit / 100);
}
```

Άσκηση 1.3 - Main (1/2)


```c
int main() {
    double rawMaterials, laborCost, overhead, profit;

    // Είσοδος δεδομένων από τον χρήστη
    printf("Εισαγάγετε το κόστος πρώτων υλών (€): ");
    scanf("%lf", &rawMaterials);

    printf("Εισαγάγετε το κόστος εργασίας (€): ");
    scanf("%lf", &laborCost);

    printf("Εισαγάγετε τα γενικά έξοδα (€): ");
    scanf("%lf", &overhead);

    printf("Εισαγάγετε το ποσοστό κέρδους (%%): ");
    scanf("%lf", &profit);
```

Άσκηση 1.3 - Main (2/2)


```
    // Υπολογισμός κόστους παραγωγής
    double productionCost = calcProductionCost(rawMaterials, laborCost, overhead);
    printf("Συνολικό κόστος παραγωγής: %.2f€\n", productionCost);

    // Υπολογισμός τιμής πώλησης
    double sellingPrice = calcSellingPrice(rawMaterials, laborCost, overhead, profit);
    printf("Τελική τιμή πώλησης: %.2f€\n", sellingPrice);

    return 0;
}
```

Άσκηση 1.4

Για τον υπολογισμό της ημέρας της εβδομάδας μιας δεδομένης ημερομηνίας χρησιμοποιείται ο τύπος Zeller's congruence:

$$w = \left(d + \left\lfloor \frac{13(m+1)}{5} \right\rfloor + K + \left\lfloor \frac{K}{4} \right\rfloor + \left\lfloor \frac{J}{4} \right\rfloor + 5J\right) \mod 7$$

Όπου:

- **d:** Ημέρα του μήνα.
- **m:** Μήνας (Μάρτιος=3, ..., Δεκέμβριος=12, Ιανουάριος=13, Φεβρουάριος=14).
- **K:** Τα δύο τελευταία ψηφία του έτους (για το 2024, είναι 24).
- **J:** Τα πρώτα δύο ψηφία του έτους (για το 2024, είναι 20).
- **w:** Ημέρα της εβδομάδας (0 = Σάββατο, 1 = Κυριακή, ..., 6 = Παρασκευή).

Αν ο μήνας είναι **Ιανουάριος (1)** ή **Φεβρουάριος (2)**, τότε θεωρείται ότι **ανήκει στο προηγούμενο έτος.** Δηλαδή, για **Ιανουάριο** γράφουμε **m = 13** και για **Φεβρουάριο** γράφουμε **m = 14**. Ταυτόχρονα, μειώνουμε το έτος κατά 1.

Άσκηση 1.4

1. Υλοποιήστε μια συνάρτηση **zeller(int d, int m, int y)** που θα δέχεται:
   - **d:** Την ημέρα του μήνα.
   - **m:** Τον μήνα (με τροποποίηση: 3=Μάρτιος, 4=Απρίλιος, ..., 13=Ιανουάριος, 14=Φεβρουάριος).
   - **y:** Το έτος.
   - Η συνάρτηση θα επιστρέφει έναν ακέραιο αριθμό w, όπου: 0 = Σάββατο, 1 = Κυριακή, 2 = Δευτέρα, ..., 6 = Παρασκευή.

Άσκηση 1.4 - Συνάρτηση zeller


```c
#include <stdio.h>

int zeller(int d, int m, int y) {
    if (m == 1 || m == 2) {
        m += 12; // Μετατροπή Ιανουαρίου & Φεβρουαρίου
        y--;     // Ανήκει στο προηγούμενο έτος
    }

    int K = y % 100; // Τα δύο τελευταία ψηφία του έτους
    int J = y / 100; // Τα πρώτα δύο ψηφία του έτους

    int w = (d + (13 * (m + 1)) / 5 + K + (K / 4) + (J / 4) + (5 * J)) % 7;

    return w; // 0 = Σάββατο, 1 = Κυριακή, ..., 6 = Παρασκευή
}
```

Άσκηση 1.4

2. Υλοποιήστε μια συνάρτηση **printDayOfWeek(int w)** που θα δέχεται την τιμή w και θα εμφανίζει την αντίστοιχη ημέρα της εβδομάδας.

Άσκηση 1.4 - printDayOfWeek

2. Υλοποιήστε μια συνάρτηση **printDayOfWeek(int w)** που θα δέχεται την τιμή w και θα εμφανίζει την αντίστοιχη ημέρα της εβδομάδας.

```
void printDayOfWeek(int w) {
   printf("Η ημέρα της εβδομάδας είναι: ");
   switch (w) {
       case 0: printf("Σάββατο\n"); break;
       case 1: printf("Κυριακή\n"); break;
       case 2: printf("Δευτέρα\n"); break;
       case 3: printf("Τρίτη\n"); break;
       case 4: printf("Τετάρτη\n"); break;
       case 5: printf("Πέμπτη\n"); break;
       case 6: printf("Παρασκευή\n"); break;
       default: printf("Μη έγκυρη τιμή!\n");
   }
}
```

Άσκηση 1.4

2. Στη συνάρτηση **main()**, το πρόγραμμα θα:
   - ζητά από τον χρήστη να εισάγει ημέρα, μήνα και έτος. Ο μήνας θα εισάγεται χωρίς τροποποίηση (1=Ιανουάριος, 2=Φεβρουάριος, ..., 12=Δεκέμβριος)
   - καλεί τη συνάρτηση **zeller()** για να υπολογίσει την ημέρα της εβδομάδας.
   - εμφανίζει το αποτέλεσμα με την **printDayOfWeek()**.

Άσκηση 1.4 - main


```c
int main() {
    int day, month, year;

    printf("Δώστε ημέρα: ");
    scanf("%d", &day);
    printf("Δώστε μήνα: ");
    scanf("%d", &month);
    printf("Δώστε έτος: ");
    scanf("%d", &year);

    int w = zeller(day, month, year);
    printDayOfWeek(w);

    return 0;
}
```

Άσκηση 1.5

Να γραφούν τρεις συναρτήσεις για τις παρακάτω λειτουργίες:

- συνάρτηση που να δέχεται ως όρισμα έναν ακέραιο αριθμό και να τον εμφανίζει αντίστροφα.
- συνάρτηση που να μετρά το πλήθος των ψηφίων ενός ακεραίου.
- συνάρτηση που να μετρά το πλήθος των bit της δυαδικής αναπαράστασης ενός ακεραίου (π.χ. ο αριθμός 9 στο δυαδικό σύστημα γράφεται 1001, άρα πλήθος bit = 4)

Να γραφεί πρόγραμμα που να διαβάζει έναν ακέραιο αριθμό, να καλεί τις παραπάνω συναρτήσεις και να εμφανίζει τα αποτελέσματα.

Άσκηση 1.5 - reverseNumber


```c
#include <stdio.h>

void reverseNumber(int n) {
    printf("Αριθμός αντίστροφα: ");
    while (n > 0) {
        printf("%d", n % 10);
        n /= 10;
    }
    printf("\n");
}
```

Άσκηση 1.5 - countDigits


```
int countDigits(int n) {
    if (n == 0) return 1; //  Ειδική περίπτωση για το 0
    int count = 0;
    while (n > 0) {
        count++;
        n /= 10;
    }
    return count;
}
```

Άσκηση 1.5 - countBits


```
int countBits(int n) {
    if (n == 0) return 1; //  Ειδική περίπτωση για το 0
    int count = 0;
    while (n > 0) {
        count++;
        n >>= 1; //  Δεξιά ολίσθηση (διαίρεση με 2)
    }
    return count;
}
```

bit3	bit2	bit1	bit0
1	1	0	1
$1x2^3$	$1x2^2$	$0x2^1$	$1x2^0$

$1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 = \mathbf{13}$

Δεξιά ολίσθηση →

bit3	bit2	bit1	bit0
0	1	1	0
$0x2^3$	$1x2^2$	$1x2^1$	$0x2^0$

$0x2^3 + 1x2^2 + 1x2^1 + 0x2^0 = \mathbf{6}$

Άσκηση 1.5 - main


```c
int main() {
    int number;

    printf("Δώσε έναν ακέραιο αριθμό: ");
    scanf("%d", &number);

    reverseNumber(number);
    printf("Πλήθος ψηφίων: %d\n", countDigits(number));
    printf("Πλήθος bit στη δυαδική αναπαράσταση: %d\n", countBits(number));

    return 0;
}
```