Υποπρογράμματα

Συναρτήσεις (Functions)

Εκπαιδευτής: Άριστος Χατζησοφοκλέους

Ειδικότητα: Μηχανικής Ηλεκτρονικών Υπολογιστών

ΤΕΣΕΚ Πάφου

Τάξη Β – Μέση Τεχνική και Επαγγελματική Εκπαίδευση

Μάθημα: Εφαρμογές Προγραμματισμού

1

Σημερινό μάθημα

Να γραφεί πρόγραμμα το οποίο για 3 διαφορετικές περιπτώσεις να ζητά 3 ακέραιους αριθμούς και να υπολογίζει και εμφανίζει τον μεγαλύτερο από αυτούς.

```
#include <iostream>
using namespace std;

int main() {
    int a, b, c, max;
    cout<<"Periptosi 1 : "<<endl;
    cout<<" dose 3 akereous arithmous:";
    cin>>a>>b>>c;
    max=a;
    if (b>max)
        max=b;
    if (c>max)
        max=c;
    cout<<endl;
    cout<<"Periptosi 1 :O megaliteros arithmos einai:"<<max<<endl;
    cout<<endl;
    cout<<"Periptosi 2 : "<<endl;
    cout<<" dose 3 akereous arithmous:";
    cin>>a>>b>>c;
    max=a;
    if (b>max)
        max=b;
    if (c>max)
        max=c;
    cout<<endl;
    cout<<"Periptosi 2 :O megaliteros arithmos einai:"<<max<<endl;
    cout<<endl;
    cout<<"Periptosi 3 : "<<endl;
    cout<<" dose 3 akereous arithmous:";
    cin>>a>>b>>c;
    max=a;
    if (b>max)
        max=b;
    if (c>max)
        max=c;
    cout<<endl;
    cout<<"Periptosi 3 :O megaliteros arithmos einai:"<<max<<endl;
    return 0;
}
```

```
I:\C++\21-02-2020\askisi1_aris.exe
Periptosi 1 :
 dose 3 akereous arithmous:10 20 30

Periptosi 1 :O megaliteros arithmos einai:30

Periptosi 2 :
 dose 3 akereous arithmous:55 6 15

Periptosi 2 :O megaliteros arithmos einai:55

Periptosi 3 :
 dose 3 akereous arithmous:64 200 3

Periptosi 3 :O megaliteros arithmos einai:200

Process returned 0 (0x0)   execution time : 27.528 s
Press any key to continue.
```

2

```
#include <iostream>
using namespace std;

int megalyteros(int x, int y, int z)
{
int max1;
max1=x;
    if (y>max1)
        max1=y;
    if (z>max1)
        max1=z;
    return max1;
        }

int main() {
    int a, b, c, max;
    cout<<"Periptosi 1 : "<<endl;
    cout<<" dose 3 akereous arithmous:";
    cin>>a>>b>>c;
    max=megalyteros(a,b,c);
    cout<<endl;
    cout<<"Periptosi 1 :O megaliteros arithmos einai:"<<max<<endl;
    cout<<endl;
    cout<<"Periptosi 2 : "<<endl;
    cout<<" dose 3 akereous arithmous:";
    cin>>a>>b>>c;
    max=megalyteros(a,b,c);
    cout<<endl;
    cout<<"Periptosi 2 :O megaliteros arithmos einai:"<<max<<endl;
    cout<<endl;
    cout<<"Periptosi 3 : "<<endl;
    cout<<" dose 3 akereous arithmous:";
    cin>>a>>b>>c;
    max=megalyteros(a,b,c);
    cout<<endl;
    cout<<"Periptosi 3 :O megaliteros arithmos einai:"<<max<<endl;
    return 0;
}
```

Υποπρόγραμμα

Κλήση υποπρογράμματος

Κλήση υποπρογράμματος

Κλήση υποπρογράμματος

Σημερινό μάθημα

Να γραφεί πρόγραμμα το οποίο για 3 διαφορετικές περιπτώσεις να ζητά 3 ακέραιους αριθμούς και να υπολογίζει και εμφανίζει τον μεγαλύτερο από αυτούς.

Λύση με χρήση υποπρογράμματος

```
I:\C++\21-02-2020\askisi1_aris.exe
Periptosi 1 :
 dose 3 akereous arithmous:10 20 30

Periptosi 1 :O megaliteros arithmos einai:30

Periptosi 2 :
 dose 3 akereous arithmous:55 6 15

Periptosi 2 :O megaliteros arithmos einai:55

Periptosi 3 :
 dose 3 akereous arithmous:64 200 3

Periptosi 3 :O megaliteros arithmos einai:200

Process returned 0 (0x0)   execution time : 27.528 s
Press any key to continue.
```


Γιατί υποπρόγραμμα ;

Αρκετά συχνά κάποια ενέργεια ή ένα ολόκληρο τμήμα προγράμματος, είναι αναγκαίο να επαναλαμβάνεται σε πολλά διαφορετικά σημεία του προγράμματος αλλά να επεξεργάζεται διαφορετικά δεδομένα.

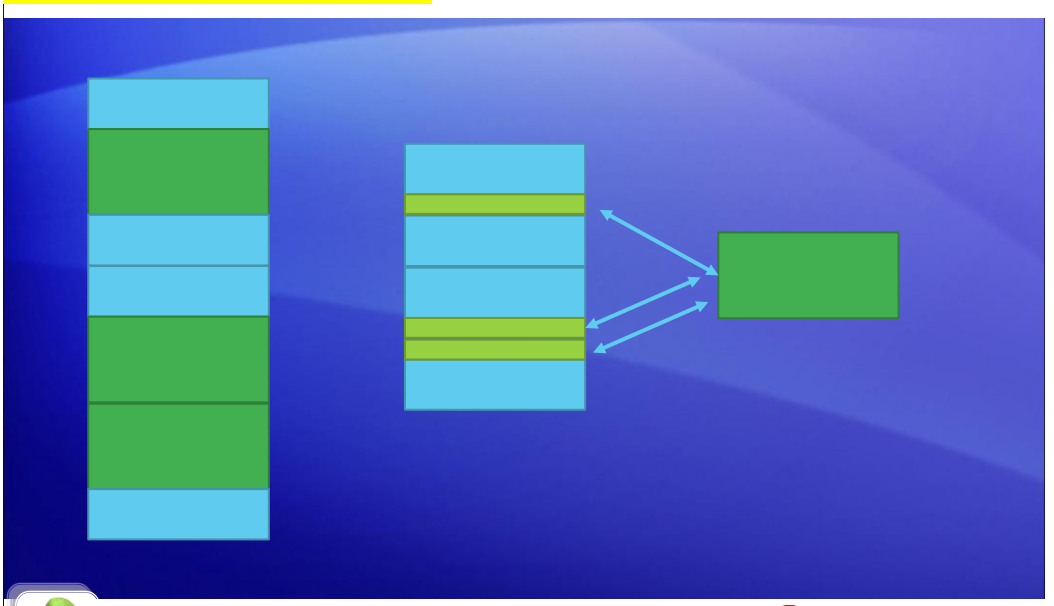
Είναι προγραμματιστικά (και πρακτικά) σωστό να επαναλαμβάνουμε το τμήμα κώδικα αυτούσιο (με άλλα δεδομένα φυσικά) στα διάφορα σημεία του προγράμματος που το χρειαζόμαστε; **Μάλλον όχι.**

Μήπως είναι καλύτερο, η γλώσσα προγραμματισμού να προσφέρει τη δυνατότητα δημιουργίας μιας ομάδας (block) εντολών η οποία:

- να συμβολίζεται με ένα όνομα,
- να έχει εισόδους για την είσοδο δεδομένων και
- εξόδους για την έξοδο των αποτελεσμάτων

και στην συνέχεια με μία απλή αναφορά του ονόματος της στα διάφορα σημεία του προγράμματος να εκτελείται αυτούσιο όλο το τμήμα με διαφορετικά όμως δεδομένα;

Γιατί υποπρόγραμμα ;



Στόχοι μαθήματος

1. Να αναγνωρίζουμε προβλήματα που χρειάζονται για την επίλυσή τους υποπρογράμματα / συναρτήσεις (functions)
2. Να αναφέρουμε τι είναι υποπρόγραμμα / συνάρτηση και ποιον σκοπό εξυπηρετεί σε ένα πρόγραμμα
3. Να συντάσσουμε την επικεφαλίδα και το σώμα μίας συνάρτησης (επιλογή τύπου επιστρεφόμενης τιμής, τύπων δεδομένων των τυπικών παραμέτρων, τοπικών μεταβλητών, εντολών και επιστροφή τιμής).
4. Να συντάσσουμε τη κλήσης μίας συνάρτησης (πραγματικές παράμετροι, αντιστοίχιση με τις τυπικές παραμέτρους και επιστρεφόμενη τιμή) και τον ρόλο της εντολής επιστροφής τιμής (return)
5. Να υλοποιούμε τον σχεδιασμό του προβλήματος με χρήση υποπρογράμματος, σε πρόγραμμα, με τη χρήση του προγραμματιστικού περιβάλλοντος, ώστε να επιλυθεί το πρόβλημα.

Ορισμοί

Τμηματικός προγραμματισμός

Τμηματικός προγραμματισμός (Modular programming) είναι μία τεχνική ανάπτυξης προγραμμάτων, της οποίας κύριο χαρακτηριστικό είναι η διάσπαση ενός προβλήματος σε πιο απλά τμήματα προγραμμάτων, έτσι ώστε να διευκολυνθεί ο έλεγχος και να μειωθεί η έκταση του κώδικά.

Υποπρόγραμμα

θεωρείται ένα αυτόνομο κομμάτι κώδικα, το οποίο εκτελεί ένα συγκεκριμένο έργο και ορίζεται ξεχωριστά από το κυρίως πρόγραμμα.

Καλές πρακτικές

1. Πριν ξεκινήσουμε να γράφουμε ένα πρόγραμμα, μελετάμε πώς αυτό μπορεί να αναλυθεί σε επιμέρους τμήματα και αποφασίζουμε για τα αντίστοιχα υποπρογράμματα.
2. Εξετάζουμε αν κάποια υποπρογράμματα, τα οποία έχουμε ήδη γράψει στο παρελθόν ή υπάρχουν σε έτοιμες βιβλιοθήκες προγραμμάτων, μπορούν να χρησιμοποιηθούν, για να κερδίσουμε χρόνο και να αποφύγουμε λάθη.
3. Προσπαθούμε κάθε υποπρόγραμμα να είναι όσο το δυνατόν πιο ανεξάρτητο από τα άλλα. Αυτό μας προφυλάσσει από λάθη στο πρόγραμμά μας και επιτρέπει τη χρήση του σε άλλα προγράμματα αργότερα.

Η τεχνική του τμηματικού προγραμματισμού προσφέρει καλύτερο έλεγχο και υψηλό επίπεδο αφαίρεσης και σαν τεχνική δεν χρησιμοποιείται μόνο στον προγραμματισμό αλλά σε πολλούς τομείς της ανθρώπινης δραστηριότητας.

Έτοιμες Συναρτήσεις

Σε όλες τις γλώσσες προγραμματισμού υπάρχουν έτοιμες συναρτήσεις που μπορεί να χρησιμοποιήσει ο χρήστης.

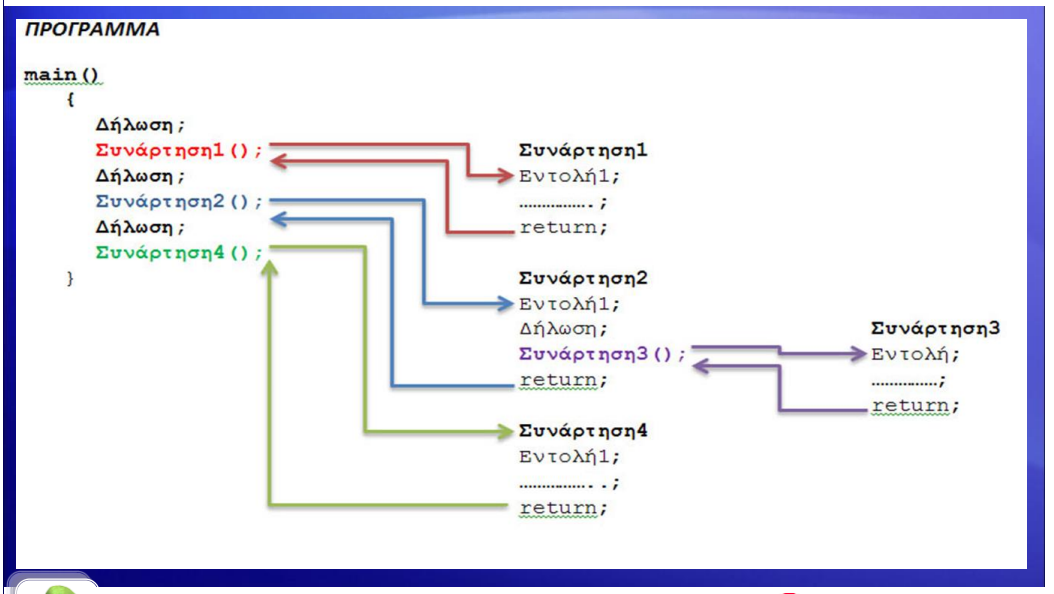
Στη C++ απαιτείται να ορίσουμε τις κατάλληλες βιβλιοθήκες μέσα στις οποίες βρίσκονται αυτές οι συναρτήσεις.

π.χ. **<cmath>** αν θέλουμε να χρησιμοποιήσουμε την συνάρτηση **sqrt(…)**

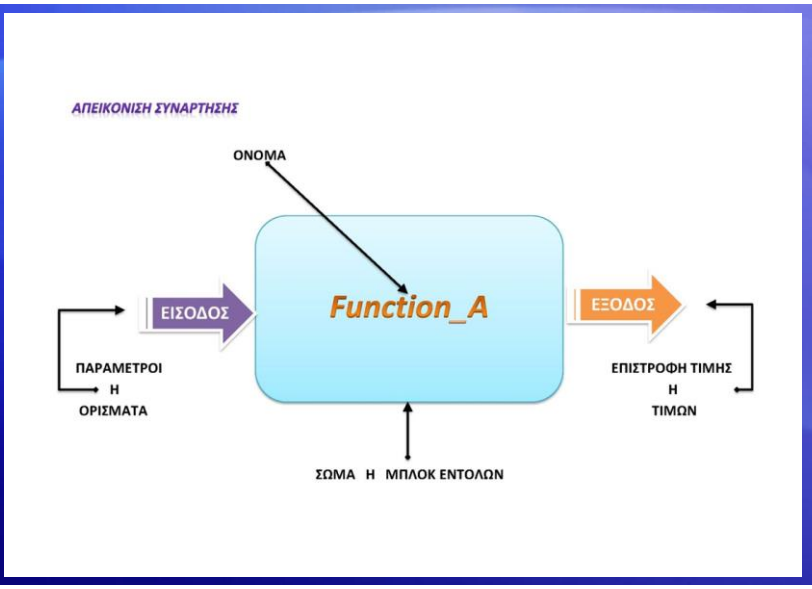
<iomanip> αν θέλουμε να χρησιμοποιήσουμε την συνάρτηση setw(…)

Μπορούμε να ορίσουμε τις δικές μας συναρτήσεις για λειτουργίες που επαναλαμβάνονται στο πρόγραμμά μας.

ΣΧΗΜΑΤΙΚΗ ΠΑΡΑΣΤΑΣΗ ΛΕΙΤΟΥΡΓΙΑΣ ΚΛΗΣΕΩΝ ΣΥΝΑΡΤΗΣΕΩΝ



ΑΠΕΙΚΟΝΙΣΗ ΣΥΝΑΡΤΗΣΗΣ



ΑΠΟ ΤΙ ΑΠΟΤΕΛΕΙΤΑΙ

Η συνάρτηση αποτελείται από τα πιο κάτω στοιχεία:

- Τύπος επιστρεφόμενης τιμής
- Το όνομα της συνάρτησης,
- Το σώμα της (ή αλλιώς μπλοκ εντολών),
- Την είσοδο που είναι ένα σύνολο τιμών ή αλλιώς ορισμάτων και
- Την έξοδο που είναι μία επιστρεφόμενη τιμή.

Ορισμός και κλήση συναρτήσεων

Η συνάρτηση (function) είναι ένας τύπος υποπρογράμματος που υπολογίζει και επιστρέφει μόνο μία τιμή στο κυρίως πρόγραμμα.

```
#include <iostream>
using namespace std;

int add_2_nums(int a, int b){          // Ορισμός συνάρτησης
      return a + b;                    // Επιστροφή αθροίσματος
}                                      // Τέλος σώματος συνάρτησης

int main() {                           // Κύρια συνάρτηση (main)
      cout << add_2_nums(65,97);       // Κλήση συνάρτησης add_2_nums
return 0;                              // μέσα στην κύρια συνάρτηση
}                                      // Τέλος κύριας συνάρτησης
```

Ορισμός και κλήση συναρτήσεων

Για κάθε συνάρτηση πρέπει να ορίζουμε τα εξής:

(α) **Τύπος επιστρεφόμενης τιμής:** Η συνάρτηση μπορεί να επιστρέψει όλους τους βασικούς τύπους που γνωρίζουμε (int, double, char, boolean), ακόμα και συμβολοσειρές (string). Αν η συνάρτηση δεν θα επιστρέφει κάποια τιμή, ορίζεται με τύπο void. Αν επιστρέφει κάποια τιμή, αυτή θα εμφανίζεται μετά τη λέξη return.

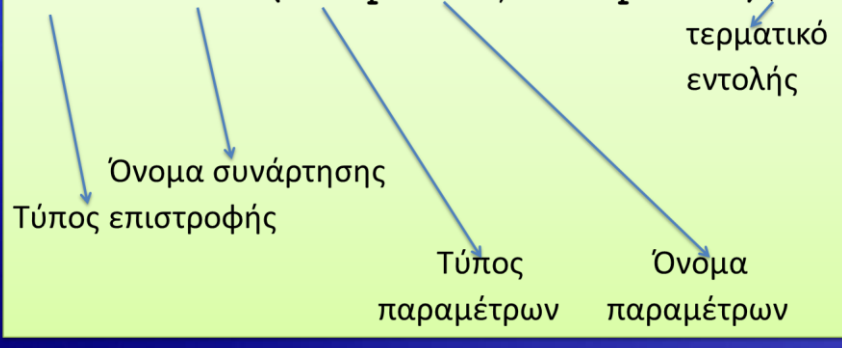
(β) **Όνομα συνάρτησης**: Ισχύουν οι ίδιοι κανόνες ονοματολογίας με τις μεταβλητές.

(γ) **Παράμετροι**: Μεταβλητές που επιτρέπουν το πέρασμα της τιμής τους από ένα τμήμα προγράμματος σε άλλο. Μία συνάρτηση μπορεί να έχει μηδέν ή περισσότερες παραμέτρους.

Ορισμός και κλήση συναρτήσεων

Η δήλωση συντάσσεται ως εξής:

```
int function(int param1, int param2);
```



15

Ορισμός και κλήση συναρτήσεων

```
#include <iostream>
using namespace std;

int add_2_nums(int a, int b){          // Ορισμός συνάρτησης
      return a + b;                    // Επιστροφή αθροίσματος
}                                      // Τέλος σώματος συνάρτησης

int main() {                           // Κύρια συνάρτηση (main)
      cout << add_2_nums(65,97);       // Κλήση συνάρτησης add_2_nums
return 0;                              // μέσα στην κύρια συνάρτηση
}                                      // Τέλος κύριας συνάρτησης
```

Στο πιο πάνω παράδειγμα η **επικεφαλίδα της συνάρτησης** είναι η εξής:

```
int add_2_nums(int a, int b)
```

Από την επικεφαλίδα συμπεραίνουμε τα εξής:

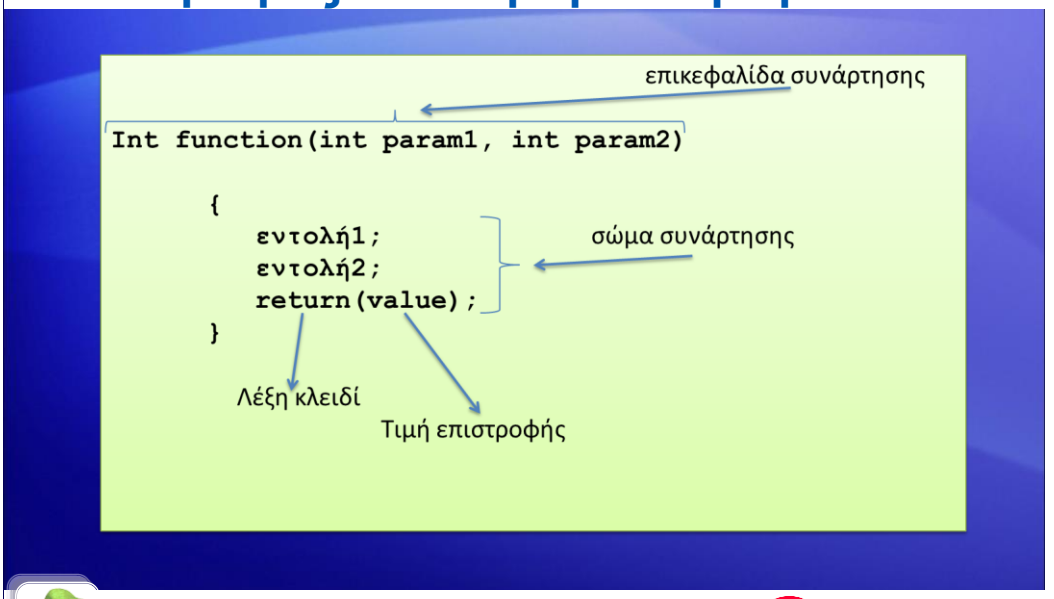
- Η συνάρτηση θα επιστρέφει έναν ακέραιο αριθμό γιατί είναι τύπου integer
- Το όνομα της συνάρτησης είναι **add_2_nums**
- Η συνάρτηση δέχεται δύο παραμέτρους (a, b) ως ακέραιες τιμές

Η κλήση της συνάρτησης γίνεται στο κυρίως πρόγραμμα ως εξής:

```
cout << add_2_nums(65,97);
```


16

Ορισμός και κλήση συναρτήσεων



Παράδειγμα 1

Να δημιουργήσετε πρόγραμμα το οποίο να καλεί μία συνάρτηση η οποία να δέχεται παραμετρικά δύο ακέραιους αριθμούς και να επιστρέφει τον μεγαλύτερο από τους δύο. Να θεωρήσετε ότι οι δύο ακέραιοι που θα δοθούν δεν θα είναι ίσοι.

```
#include <iostream>
using namespace std;

int max_num(int a, int b) {                 // Ορισμός συνάρτησης
if (a>b)                                    // Αν a>b η συνάρτηση θα επιστρέψει
return a;                                   // το a και θα τερματίσει
return b;                                   // Αλλιώς θα επιστρέψει το b. Το
}                                           // else μπορούμε να το παραβλέψουμε

int main() {                                // Κύρια συνάρτηση (main)
int num1, num2;
cin >> num1 >> num2;                        // Καταχώριση τιμών
cout << max_num(num1,num2);                 // Εμφάνιση μέγιστου
return 0;
}
```

Παράδειγμα 1 – 2ος τρόπος κλησης υποπρογράμματος

Να δημιουργήσετε πρόγραμμα το οποίο να καλεί μία συνάρτηση η οποία να δέχεται παραμετρικά δύο ακέραιους αριθμούς και να επιστρέφει τον μεγαλύτερο από τους δύο. Να θεωρήσετε ότι οι δύο ακέραιοι που θα δοθούν δεν θα είναι ίσοι.

```
#include <iostream>
using namespace std;

int max_num(int a, int b) {                    // Ορισμός συνάρτησης
if (a>b)                                       // Αν a>b η συνάρτηση θα επιστρέψει
return a;                                      // το a και θα τερματίσει
return b;                                      // Αλλιώς θα επιστρέψει το b. Το
}                                              // else μπορούμε να το παραβλέψουμε

int main() {                                   // Κύρια συνάρτηση (main)
int num1, num2;
cin >> num1 >> num2;                           // Καταχώριση τιμών
z= max_num(num1,num2);
cout << z;                    // Εμφάνιση μέγιστου
return 0;
}
```

Φύλλο εργασίας

Να κάνετε από το φύλλο εργασίας 1

Τι μάθαμε σήμερα.

1. Να αναγνωρίζουμε προβλήματα που χρειάζονται για την επίλυσή τους υποπρογράμματα / συναρτήσεις (functions)
2. Να αναφέρουμε τι είναι υποπρόγραμμα / συνάρτηση και ποιον σκοπό εξυπηρετεί σε ένα πρόγραμμα
3. Να συντάσσουμε την επικεφαλίδα και το σώμα μίας συνάρτησης (επιλογή τύπου επιστρεφόμενης τιμής, τύπων δεδομένων των τυπικών παραμέτρων, τοπικών μεταβλητών, εντολών και επιστροφή τιμής).
4. Να συντάσσουμε τη κλήσης μίας συνάρτησης (πραγματικές παράμετροι, αντιστοίχιση με τις τυπικές παραμέτρους και επιστρεφόμενη τιμή) και τον ρόλο της εντολής επιστροφής τιμής (return)
5. Να υλοποιούμε τον σχεδιασμό του προβλήματος με χρήση υποπρογράμματος, σε πρόγραμμα, με τη χρήση του προγραμματιστικού περιβάλλοντος, ώστε να επιλυθεί το πρόβλημα.