Διαδικαστικός Προγραμματισμός σε C++

Συνέχεια

Short-circuit evaluation

Οι σύνθετες λογικές εκφράσεις με δύο μέλη υπολογίζονται από αριστερά προς τα δεξιά και σταματά ο υπολογισμός μόλις προσδιοριστεί η τελική τιμή της συνολικής έκφρασης.
Οι εκφράσεις μπορεί να είναι χρονοβόρες ή να έχουν «παρενέργειες».
Όταν δεν υπολογίζονται, αποφεύγουμε τις συνέπειές τους.

Παράδειγμα

```
int a, b{3};
std::cin >> a;
bool c = a > 5 &&++b < a; // c = (a>5) && ((++b)<a);
std::cout <<b <<'\n';
```

Πόσο είναι το b;

Αν a>5 **τότε** b=4**, αλλιώς** b=3**.**

Εντολή if (1/2)

```
if (λογική_έκφραση) {
        ... // τμήμα A
}else{
        ... // τμήμα B
}
```

Αν η λογική_έκφραση είναι αληθής, εκτελούνται οι εντολές στο τμήμα Α αλλιώς εκτελούνται οι εντολές στο τμήμα Β.

Παράδειγμα

Αν η πραγματική ποσότητα x **είναι μεγαλύτερη από** 0 **να τυπώσουμε «θετική» αλλιώς να τυπώσουμε «αρνητική ή 0»:**

```
if (x >0.0){
    std::cout <<u8"Θετική“
} else {
    std::cout <<u8"Αρνητική ή 0";
}
```

Εντολή if (2/2)

Παρατηρήσεις

- Μπορούμε να παραλείψουμε τα άγκιστρα που περιβάλλουν μία εντολή.
- Μπορούμε να παραλείψουμε το τμήμα από το else και μετά, αν είναι κενό το block B.
- Ως λογική έκφραση μπορούμε να έχουμε ποσότητα τύπου bool.

Παράδειγμα

Η μεταβλητή y να αλλάξει πρόσημο αν η πραγματική ποσότητα x είναι μεταξύ −1 και 1:

```
bool a = -1.0 <= x && x <= 1.0;
if (a)
    y = -y;
```

Τριαδικός τελεστής ? : (1/2)

Τελεστής με *τρία* ορίσματα: μια λογική συνθήκη και δυο εκφράσεις οποιουδήποτε είδους.

Σύνταξη

Η έκφραση

λογική_έκφραση ?έκφρασηΑ :έκφρασηΒ

έχει την τιμή «έκφρασηΑ» όταν η «λογική_έκφραση» είναι αληθής, ενώ έχει την τιμή «έκφρασηΒ» όταν η λογική_έκφραση είναι ψευδής. Ο υπολογισμός της κατάλληλης έκφρασης γίνεται μετά την επιλογή της.

Τριαδικός τελεστής ? : (2/2)

Παράδειγμα

Το

```
val = (συνθήκη ? value1 : value2);
```

ισοδυναμεί με

```
if (συνθήκη) {
    val = value1;
} else {
    val = value2;
}
```

Μπορεί να εμφανιστεί και στο *αριστερό μέρος* **του τελεστή εκχώρησης (με παρενθέσεις):**

```
(k == 5 ? a : b) = 3;
```

Εντολή switch (1/3)

```
switch (i) {
   case v1:
   ...
   case v2:
   ...
   case vN:
   ...
   default:
   ...
}
```

Ελέγχεται το i.
Αν είναι ίσο με v1 εκτελούνται όλες οι εντολές μετά το case v1.
Αν είναι ίσο με v2 εκτελούνται όλες οι εντολές μετά το case v2, κλπ.
Αν δεν είναι ίσο με κανένα, εκτελούνται όλες οι εντολές μετά το default.

Παρατηρήσεις

- **Το i είναι ακέραιος τύπος (int και παραλλαγές char, bool).**
- **τα v1, v2, .. είναι υποχρεωτικά σταθερές τιμές.**
- **Σχεδόν πάντα είναι απαραίτητο το break στο τέλος κάθε block εντολών ώστε να διακόπτεται η εκτέλεση του switch.**

Εντολή switch. Α' Παράδειγμα (2/3)

Αν ο ακέραιος k είναι 1,2,3,4 να τυπώσουμε "up", "down", "left", "right" αντίστοιχα. Για άλλη τιμή να γράψουμε "error":

```
switch (k) {
case 1:
   std::cout <<"up\n";
   break;
case 2:
   std::cout <<"down\n";
   break;
case 3:
   std::cout <<"left\n";
   break;
case 4:
   std::cout <<"right\n";
   break;
default:
   std::cout <<"error\n";
}
```

Εντολή switch. Β' Παράδειγμα (3/3)

Ομαδοποίηση τιμών

Αν ο ακέραιος k **είναι** 2, 4, 6, 8 **να τυπώσουμε "ζυγός".**
Αν είναι 1, 3, 5, 7, 9 **να τυπώσουμε "μονός".**
Αν είναι οτιδήποτε άλλο να τυπώσουμε "λάθος":

```
switch (k) {
case 2: case 4:
case 6: case 8:
    std::cout << u8"ζυγός\n";
    break;
case 1: case 3:
case 5: case 7: case 9:
    std::cout << u8"μονός\n";
    break;
default:
   std::cout << u8"λάθος\n";
}
```

Μιγαδικός Τύπος (1/3)

Υποστηρίζεται ο μιγαδικός τύπος αφού συμπεριληφθεί ο header <complex>**. Αν γράψουμε και την εντολή**
using namespace std::complex_literals;
μπορούμε να γράψουμε φανταστικούς αριθμούς με απλό τρόπο.

Δηλώσεις (με απόδοση αρχικής τιμής)

- std::complex<**double**> z{3.4,2.8}; *// z = 3.4 + 2.8 i*
- std::complex<**double**> z{3.4}; *// z = 3.4 + 0.0 i*
- std::complex<**double**> z = 3.4; *// z = 3.4 + 0.0 i*
- std::complex<**double**> z = 2.8i; *// z = 0.0 + 2.8 i*
- std::complex<**double**> z{}; *// z = 0.0 + 0.0 i*
- std::complex<**double**> z; *// z = 0.0 + 0.0 i*
- std::complex<**double**> w{z}; *// w = z*

Μιγαδικός Τύπος (2/3)

Πράξεις και συναρτήσεις μιγαδικών ποσοτήτων

- **Οι αριθμητικοί τελεστές '+', '-', '*', '/' και οι συντμήσεις '+=', '-=', '*=', '/=' μεταξύ μιγαδικών αριθμών ίδιου τύπου ή ενός μιγαδικού και ενός πραγματικού αριθμού με ίδιο βασικό τύπο εκτελούν τις αναμενόμενες και γνωστές πράξεις από τα μαθηματικά και έχουν μιγαδικό αποτέλεσμα.**
- **Οι μαθηματικές συναρτήσεις που έχουν νόημα για μιγαδικούς αριθμούς, δέχονται μιγαδικά ορίσματα και επιστρέφουν το αντίστοιχο μιγαδικό αποτέλεσμα (εκτός από τη συνάρτηση** std:: abs()**).**

Μιγαδικός Τύπος (3/3)

Επιπλέον συναρτήσεις για μιγαδικούς

- std::abs(z) → $\sqrt{zz^{\star}}$.
- std::polar(r,t) → re^{it}. **Αν παραληφθεί το** t **θεωρείται** 0.0.
- std::norm(z) → $zz^{\star}$.
- std::arg(z) → $\tan^{-1}(\beta/\alpha)$, **αν** $z = \alpha + i\beta$.
- std::conj(z) → $z^{\star}$.
- std::real(z) → α, **αν** $z = \alpha + i\beta$.
- std::imag(z) → β, **αν** $z = \alpha + i\beta$.

Για να αποδώσουμε νέα τιμή στο πραγματικό ή φανταστικό μέρος μιας μιγαδικής μεταβλητής z, μπορούμε να χρησιμοποιήσουμε τις συναρτήσεις–μέλη real() **και** imag() **με ένα όρισμα, τη νέα τιμή:**

```
std::complex<double> z{3.0,1.5}; // z = 3.0+1.5i
z.real(4.2);  // z = 4.2+1.5i
z.imag(-0.9); // z = 4.2−0.9i
```

Είσοδος/Εξοδος μιγαδικών δεδομένων (1/2)

Η εκτύπωση μιγαδικών δεδομένων με τον τελεστή '<<' γίνεται με τη μορφή

(πραγματικό μέρος,φανταστικό μέρος)

Η ανάγνωση μιγαδικών δεδομένων με τον τελεστή '>>' γίνεται με μια από τις παρακάτω διαμορφώσεις:

(πραγματικό μέρος,φανταστικό μέρος)
(πραγματικό μέρος)
πραγματικό μέρος

Είσοδος/Εξοδος μιγαδικών δεδομένων(2/2)

Παράδειγμα

Οι εντολές

```
std::complex<double> z{3.2,1.5};
std::cout << z;
```

εμφανίζουν στην οθόνη

(3.2,1.5)

Οι εντολές

```
std::complex<double> w;
std::cin >> w;
```

περιμένουν κάτι σαν

(2.3, 1.5) ή (2.3) ή 2.3

Εντολή using

Μπορούμε να ορίσουμε μια άλλη, ισοδύναμη αλλά ίσως πιο σύντομη *ονομασία* για τύπο με τη βοήθεια της εντολής using.

Σύνταξη

using *νέο_όνομα_τύπου =παλαιό_όνομα_τύπου;*

Δεν δημιουργείται νέος τύπος αλλά αποκτά νέο όνομα ο αρχικός.

Παράδειγμα

```
using complex = std::complex<double>;

std::complex<double> z{3.4, 1.4} ;
complex  w{2.8, -1.2};
```

Αναφορά (1/2)

Μπορούμε να ορίσουμε μια άλλη, ισοδύναμη αλλά ίσως πιο σύντομη, *ονομασία* για ποσότητα (μεταβλητή, σταθερή, συνάρτηση, κλπ.):

Σύνταξη

τύπος όνομαΑ;
τύπος & όνομαΒ{όνομαΑ};
auto *& όνομαΒ = όνομαΑ;*

Δεν δημιουργείται νέα ποσότητα αλλά μόνο αποκτά νέο όνομα η αρχική.

Παράδειγμα

```
int  a{3} ;
auto b = a;  // b = 3
auto  &c  =  a;
b=5;  // a = 3
c=7;  // a = 7
```

Αναφορά (2/2)

Αν η αρχική ποσότητα είναι σταθερή (const ή constexpr) πρέπει και η αναφορά να είναι const:

Παράδειγμα

```
int  constexpr a{5} ;
int  &p{a};          // Λάθος
int  const &q{a};
/* Σωστό. Δεν μπορεί να αλλάξει το a μέσω του q */
```

Μπορούμε να ορίσουμε (σταθερή) αναφορά σε σταθερή, μεταβλητή ή έκφραση κατάλληλου τύπου:

Παράδειγμα

```
int  const &p{4};
int  a{3};
int  const &q{a};
int  const &r{2 * a} ;
```

Χώρος ονομάτων (1/2)

Μπορούμε να οργανώσουμε τις μεταβλητές, σταθερές, συναρτήσεις, κλάσεις, κλπ. σε χώρους ονομάτων(namespaces**):**

```
namespace my{
int a;
}
namespace your {
double a;
}
```

Οι δύο μεταβλητές με το ίδιο όνομα είναι διαφορετικές. Εκτός των δύο χώρων ονομάτων, η πρώτη ονομάζεται my::a **και η δεύτερη** your::a**. Εντός κάθε χώρου η "τοπική" μεταβλητή λέγεται απλώς** a**.**

Χώρος ονομάτων (2/2)

- **Όλες οι ποσότητες που παρέχονται από τη Standard Library ορίζονται στο χώρο ονομάτων** std**. Δεν μπορούμε να προσθέσουμε τίποτε σε αυτόν.**
- **Αν χρειαστεί να χρησιμοποιήσουμε πολλές φορές ποσότητες από ένα χώρο ονομάτων π.χ. το std, μπορούμε να δώσουμε την εντολή**

  using namespace std;

  στο block που περικλείει τον κώδικά μας. Μετά από αυτή, (όλα) τα ονόματα των μεταβλητών, συναρτήσεων κλπ. δεν χρειάζονται το std:: **.**
- **Αν χρησιμοποιούμε πολλές φορές συγκεκριμένη ποσότητα από ένα χώρο ονομάτων, μπορούμε να παραλείπουμε το όνομα του** namespace **για αυτή μετά από εντολή σαν κι αυτή:**

  using std::cout;

Ανώνυμος χώρος ονομάτων

Χρήσιμος είναι ο ανώνυμος χώρος ονομάτων:

```
namespace {
    …
}
```

Οι ποσότητες που ορίζονται σε αυτόν χρησιμοποιούνται απευθείας με το όνομά τους μόνο στο αρχείο που ορίζεται ο ανώνυμος namespace **(και σε όσα το συμπεριλαμβάνουν με** #include**). Δεν μπορούν να χρησιμοποιηθούν σε άλλο αρχείο και επομένως να «συγκρουστούν» με ποσότητες που ορίζονται εκεί.**

1η εφαρμογή

```
#include <iostream>
using namespace std;
int main ( )
{
	int a, b; // Οι δύο αριθμοί
	int max; // Ο μεγαλύτερος
	cout << "Εισάγετε δύο ακεραίους ";
	cin >> a >> b;
	if ( a > b )
		max = a;
	else
		max = b;
	cout << "Ο μεγαλύτερος είναι ο " << max << endl;
}
```

2η εφαρμογή

```
#include <iostream>
using namespace std;
int main ( )
{
	double a, b; // Οι συντελεστές
	double x; // Η λύση
	cout << "Εισάγετε τους συντελεστές \n";
	cin >> a >> b;
	if ( a != 0 ) {
		x = -b/a;
		cout << "Η λύση είναι " << x << endl;
	} else
		cout << "Δεν υπάρχει λύση \n";
}
```

3η εφαρμογή

```
#include <iostream>
#include <cmath>
using namespace std;
int main ( )
{
        double a, b, c; // Οι συντελεστές
        double d; // Η διακρίνουσα
        double x1, x2; // Οι λύσεις
        cout << "Εισάγετε τα a,b,c ";
        cin >> a >> b >> c;
        if ( a != 0 ) {  // Είναι εξίσωση 2ου βαθμού
            d = b*b-4*a*c;
            if ( d > 0 ) {
                x1 = (-b+sqrt(d))/(2*a);
                x2 = (-b-sqrt(d))/(2*a);
                cout << "Υπάρχουν δύο λύσεις:\n";
                cout << x1 << endl;
                cout<< x2 << endl;
            } else if ( d == 0 ) {
                x1 = -b/(2*a);
                cout << "Υπάρχει μία λύση:\n";
                cout << x1 << endl;
            } else
                cout << "Δεν υπάρχουν λύσεις\n";
        } //if (a !=0)
} //main
```

4η εφαρμογή

```
#include <iostream>
using namespace std;
int main ( )
{
      int year;
      cout << "Εισάγετε το έτος\n";
      cin >> year;
      if ( year%4 == 0 )
           if ( year%100 == 0 )
              if ( year%400 == 0 )
                   cout << "Είναι δίσεκτο\n";
              else
                   cout << "Δεν είναι δίσεκτο\n";
           else
              cout << " Είναι δίσεκτο \n";
      else
           cout << " Δεν είναι δίσεκτο \n";
}
```

Εντολή επανάληψης for

for *(αρχική_εντολή; συνθήκη; τελική_εντολή) {*
...
}

Εκτέλεση

1. Εκτελείται η «αρχική_εντολή».
2. Ελέγχεται η «συνθήκη».
   - Αν είναι ψευδής, η ροή συνεχίζει με την πρώτη εντολή μετά το σύμπλεγμα **for**.
   - Αν είναι αληθής, εκτελείται το block εντολών μεταξύ των αγκίστρων '{}'. Αν δεν υπάρξει αλλαγή της ροής στο block εκτελείται η «τελική_εντολή».
3. Αν εκτελέστηκε το block χωρίς αλλαγή ροής, επαναλαμβάνεται το βήμα 2.

Παραδείγματα εντολής for (1/2)

- Εκτύπωση στην οθόνη των αριθμών 0-9.

```
for (int i{0}; i < 10; ++i) {
   std::cout << i << '\n';
}
```

- Εκτύπωση των αριθμών 99, 97, 95,…, 3, 1:

```
for (int i{99}; i > 0; i-=2) {
   std::cout << i << ' ';
}
```

- Πλήθος των ακεραίων που είναι πολλαπλάσιοι του 2 ή του 3 στο διάστημα [5, 108]:

```
std::size_t k{0};
for (int i{5}; i < 109; ++i) {
   if (i%2 == 0 || i%3 == 0) {
      ++k;
   }
}
```

Παραδείγματα εντολής for (2/2)

- Άθροισμα περιττών αριθμών μεταξύ 1 και 9.

```
int s{0};
for (int i{1}; i < 10; i+=2) {
    s += i;
}
```

- Λάθος άθροισμα των αριθμών 0.1, 0.2, 0.3:

```
double s{0.0};
for (double x{0.1}; x <= 0.3; x += 0.1) {
    s += x;
}
```

- Σωστό άθροισμα των αριθμών 0.1, 0.2, 0.3:

```
double s{0.0};
for (int i{1}; i <= 3; ++i) {
    s += 0.1 * i;
}
```

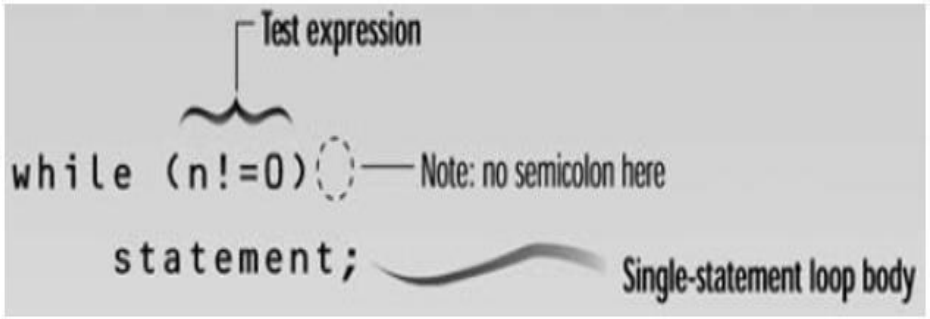
Εντολή επανάληψης while

while *(συνθήκη) {*
…
}

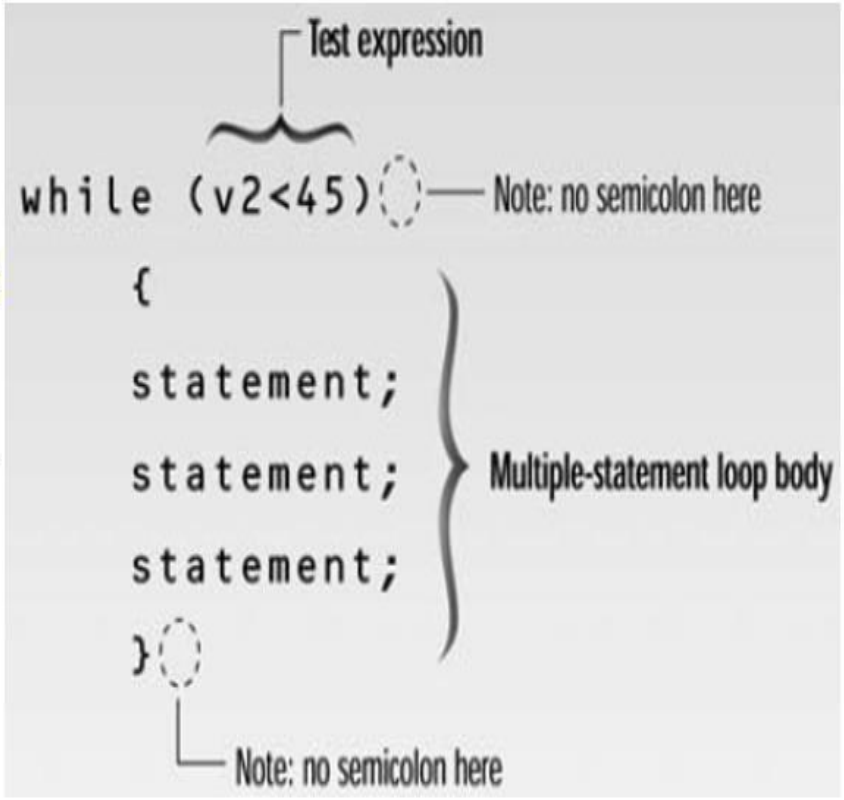
Εκτέλεση

1. Ελέγχεται η «συνθήκη»:
   - Αν είναι ψευδής, η ροή συνεχίζει με την πρώτη εντολή μετά το σύμπλεγμα **while**.
   - Αν είναι αληθής εκτελείται το block εντολών μεταξύ των αγκίστρων '{}'.
2. Αν δεν υπάρξει αλλαγή της ροής στο block επαναλαμβάνεται η διαδικασία από το βήμα 1. Η τιμή της «συνθήκης» μπορεί να μεταβληθεί κατά την εκτέλεση του block εντολών.

Loop body contain single statement



Loop body contain Multiple statement



Εντολή επανάληψης while

Θέλουμε να υπολογίσουμε στο s το άθροισμα των ακέραιων που δίνει ο χρήστης από το πληκτρολόγιο έως ότου δώσει τον αριθμό 0:

```
int i;

std::cin >> i;


int s{0};
while (i != 0)
{
    s += i;
    std::cin >> i;
}
```

Εντολή επανάληψης do ... while

do *{*

...

} ***while*** *(συνθήκη);*

Εκτέλεση

1. Εκτελείται το block εντολών μεταξύ των αγκίστρων '{}'.
2. Αν δεν υπήρξε αλλαγή ροής ελέγχεται η «συνθήκη»:
   - Αν είναι ψευδής, η ροή συνεχίζει με την πρώτη εντολή μετά την εντολή.
   - Αν είναι αληθής, επαναλαμβάνεται η διαδικασία από το βήμα 1 (εκτέλεση του block).

Η τιμή της «συνθήκης» μπορεί να μεταβάλλεται σε κάθε επανάληψη.

Εντολή επανάληψης do ... while

Παράδειγμα

Θέλουμε να εξασφαλίσουμε ότι ένας ακέραιος που το πρόγραμμά μας θα διαβάζει από το πληκτρολόγιο είναι θετικός. Αν ο χρήστης δώσει αρνητικό ή μηδέν, το πρόγραμμα να επαναλαμβάνει το διάβασμα.

```
int i;
do {
  std::cout << u8"\n Δώσε θετικό ακέραιο: ";
  std::cin >> i;
} while (i <= 0);
```

Εντολή range for

Βρόχος που διατρέχει σύνολο στοιχείων (έστω a):

```
for (auto & x : a) {
... // use x (read/write)
}
```

ή

```
for (auto const & x : a) {
... // use x (read)
}
```

Το x γίνεται διαδοχικά από το πρώτο στο τελευταίο στοιχείο, αναφορά (σταθερή ή όχι) στις τιμές των στοιχείων του a και χρησιμοποιείται στο σώμα. Π.χ.

```
for (auto const & x : {4,5,8,6}){
   std::cout << x << '\n';
}
```

Εντολή break

- Μπορεί να εμφανιστεί μόνο σε σώμα εντολής **switch** ή βρόχου.
- Η εκτέλεσή της προκαλεί διακοπή της εκτέλεσης της εντολής στην οποία βρίσκεται.

Παράδειγμα

Ανάγνωση θετικού ακέραιου.

```
int i;
while (true) {
    std::cout << u8"\n Δώσε θετικό ακέραιο: ";
    std::cin >> i;
    if (i > 0) {
        break;
    }
    std::cout << u8"Έδωσες αρνητικό\n";
}
// ... use i
```

Εντολή continue

- Μπορεί να εμφανιστεί μόνο σε σώμα βρόχου.
- Η εκτέλεσή της προκαλεί μετακίνηση της ροής εκτέλεσης στο τέλος του βρόχου.

Παράδειγμα

Θέλουμε να τυπώσουμε τις τετραγωνικές ρίζες των πρώτων 10 αριθμών εισόδου, αγνοώντας τους αρνητικούς.

```
for (int i{0}; i < 10; ++i) {
    double x;
    std::cin >> x;
    if (x < 0.0) {
      continue;
    }
    std::cout << u8"Η τετραγωνική ρίζα είναι " << std::sqrt(x) << '\n';
}
```

Συναρτήσεις

- Με την χρήση των συναρτήσεων ένα πρόγραμμα μπορεί να χωριστεί σε άλλα μικρότερα κομμάτια, δομές.
- Ο δομημένος προγραμματισμός βασίζεται στην αρχή της δόμησης μίας εφαρμογής προγραμματισμού κάνοντας χρήση μικρότερων δομημένων στοιχείων.
- Με την χρήση συναρτήσεων επιτυγχάνεται η διάσπαση ενός προβλήματος σε μικρότερα.
- Η συνάρτηση είναι ένα σύνολο εντολών που καλείται και χρησιμοποιείται από ένα σημείο ενός προγράμματος. Στη C++ ακόμη και το main θεωρείται ότι είναι μία συνάρτηση.

Συναρτήσεις της C++

- Η στανταρντ βιβλιοθήκη C++ περιλαμβάνει ένα πλήθος συναρτήσεων που μπορεί να χρησιμοποιήσει ο προγραμματιστής. Τα αρχεία επικεφαλίδων (header files) περιέχουν τους ορισμούς-δηλώσεις συναρτήσεων που μπορεί να χρησιμοποιηθούν.
- Πολλές μαθηματικές συναρτήσεις (sin, cos, tan, sqrt κλπ.) περιέχονται στο αρχείο επικεφαλίδων math.h ή cmath.h ανάλογα με τον compiler.

  Π.χ.

```
for (k=10; k>=0; k--)
    cout<<sqrt(k)<<endl;
```

Συναρτήσεις που ορίζονται από το χρήστη (1)

- Μία τέτοια συνάρτηση πρέπει να δηλωθεί και να οριστεί πριν γίνει χρήση της συνάρτησης από το πρόγραμμα. Ο ορισμός μίας συνάρτησης έχει την παρακάτω μορφή:

```
Τύπος_επιστροφής συνάρτηση (τύπος παράμετρος1,
                             τύπος παράμετρος2,…)
 {
    εντολές
 }
```

- Η παρακάτω συνάρτηση βρίσκει το άθροισμα δύο ακεραίων αριθμών

```
int athroisma(int a, int b)
{
      return a+b;
}
```

Συναρτήσεις που ορίζονται από το χρήστη (2)

- Στο παράδειγμα διακρίνονται

```
int athroisma(int a, int b)
{
        return a+b;
}
```

- Το **όνομα** της συνάρτησης
- Τον **τύπο** της τιμής που επιστρέφει
- Τις **παραμέτρους** και τον **ορισμό**τους
- Την **τιμή** που επιστρέφει με την εντολή return. Η συνάρτηση επιστρέφει μία και μόνο τιμή.

Παράδειγμα

```
using namespace std;
int addition (int a, int b)
{
    int p;
    p=a+b;
    return (p);
}

int main()
{
     int z;
     z = addition (15, 8);
     cout << "The result is " << z<<"\n";
     int k=40,l=50;
     z=addition (k, l);
     cout << "The result is " << z<<"\n";
     system("PAUSE");
     return 0;
}
```

Παρατηρήσεις

- Υπάρχει αντιστοιχία μεταξύ του τύπου και του αριθμού των παραμέτρων που χρησιμοποιούνται στον ορισμό και στην κλήση τηςσυνάρτησης.
- Οι παράμετροι που χρησιμοποιούνται για να ορίσουν τη συνάρτηση λέγονται τυπικές **παράμετροι** (formal parameters). Οι παράμετροι που χρησιμοποιούνται στην κλήση μίας συνάρτησης λέγονται **ορίσματα** (arguments). Τα ορίσματα αναφέρονται στην βιβλιογραφία και σαν παράμετροι κλήσης ή πραγματικές παράμετροι (actual parameters).
- Μόλις γίνει κλήση της συνάρτησης, οι τιμέςτων ορισμάτων μεταφέρονται στις παραμέτρους που βρίσκονται στον ορισμό της συνάρτησης. Η μεταβλητή p που ορίζεται στη συνάρτηση χρησιμοποιείται για να υπολογιστεί το άθροισμα και να γίνει επιστροφή του αθροίσματος από τη συνάρτηση στο main.
- Εάν η συνάρτηση δεν επιστρέφει καμία τιμή τότε ορίζεται σαν τύπου **void** και δεν υπάρχει **return** ή υπάρχει χωρίς τιμή (**return;**) Η τιμή που επιστρέφεται είναι του ιδίου τύπου με τη συνάρτηση.

Πρωτότυπα Συναρτήσεων

- Για να γίνει κλήση μίας συνάρτησης (δηλαδή για να χρησιμοποιηθεί η συνάρτηση) πρέπει προηγουμένως να έχει δηλωθεί.

- Η δήλωση γίνεται μέσω του λεγόμενου πρωτότυπου της συνάρτησης (function prototype) και επιτρέπει στον compiler να γνωρίζει τα απαραίτητα στοιχεία που αφορούν την συνάρτηση. Αυτά τα στοιχεία είναι ο τύπος και το όνομα της συνάρτησης καθώς και ο τύπος και ο αριθμός των παραμέτρων της συνάρτησης.

Παράδειγμα

```
int addition(int , int ); //prototypo sinartisis
int main( )
{
    int z;
    z = addition (15,8);
    cout << "The result is " << z<<"\n";

    int k=40,l=50;
    z=addition (k,l);
    cout << "The result is " << z<<"\n";
    system("PAUSE");
    return 0;
}
int addition (int a, int b)
{
    int p;
    p=a+b;
    return (p);
}
```

Σημείωση: Τα ονόματα των παραμέτρων της συνάρτησης μπορούν να παραληφθούν, όπως γίνεται στο παραπάνω πρωτότυπο

Τύπος Αναφοράς

- Το σύμβολο & χρησιμοποιείται σαν **τελεστής διεύθυνσης** (address-of operator) για να αποδώσει τιμή σε ένα δείκτη.
- Το ίδιο σύμβολο επίσης χρησιμοποιείται για να δηλώσει μία μεταβλητή που λέγεται **αναφορά** (reference) και είναι ένα εναλλακτικό όνομα μίας άλλης μεταβλητής
- Π.χ

  int x;

  int & ax=x;
- Με την δήλωση αυτή η μεταβλητή ax είναι ένα συνώνυμο της x. Οι δύο μεταβλητές έχουν το ίδιο περιεχόμενο.

Κλήση δια τιμής και κλήση με αναφορά

- Όταν καλείται μία συνάρτηση οι τιμές των ορισμάτων περνούν στην συνάρτηση με χρήση δύο παραμέτρων κλήσης. Οι τιμές των ορισμάτων αντιγράφονται σαν τιμές στις παραμέτρους της συνάρτησης δηλαδή δημιουργούνται αντίγραφα των μεταβλητών κλήσης. Η χρήση της συνάρτησης με τον τρόπο αυτό λέγεται **κλήση δια τιμής (call by value).** Τυχόν αλλαγές των παραμέτρων της συνάρτησης δεν επηρεάζουν τις αρχικές μεταβλητές στο κυρίως πρόγραμμα γιατί μόνο οι τιμές των μεταβλητών μεταφέρθηκαν στην συνάρτηση.

- Ένας δεύτερος τρόπος για το πέρασμα τιμών προς την συνάρτηση είναι με την **κλήση διά αναφοράς (call by reference)**. Με τον τρόπο αυτό η διεύθυνση του ορίσματος αντιγράφεται στην παράμετρο. Μέσα στην συνάρτηση η διεύθυνση χρησιμοποιείται για πρόσβαση στο όρισμα. Τυχόν αλλαγή στην τιμή της παραμέτρου επηρεάζουν και το όρισμα επειδή οι δύο μεταβλητές αναφέρονται στην ίδια διεύθυνση.

Παράδειγμα (χρήση παραμέτρων τιμής)

```
void func2values (int, int );

int main ( )
{           int a=10, b=30;
            cout << "The result is " << a<< b <<"\n";
            func2values(a, b);
            cout << "The result is " << a<< b <<"\n";
            system("PAUSE");
            return 0;                                   }
void func2values (int x, int  y )
{           x=2*x; y=2*y;
            cout<<"oi times x kai y edo einai "<< x <<" "<< y <<"\n“;    }
```

- Η κλήση μέσω τιμής δεν μπορεί να επιστρέψει την αλλαγμένη τιμή της παραμέτρου στο κυρίως πρόγραμμα.
- Ακόμη και με χρήση return μπορώ να επιστρέψω μία και μόνο τιμή.
- Για να γίνει επιστροφή της τιμής κάποιων μεταβλητών χρησιμοποιώ για κάθε μία από τις μεταβλητές κλήση με αναφορά

Παράδειγμα (χρήση παραμέτρων αναφοράς)

```
void func2values (int, int &);
// Stin defteri parametro ginetai klisi me anafora
int main ( )
{           int a=10, b=30;
            cout << "The result is " << a<< b <<"\n";
            func2values(a, b);
            cout << "The result is " << a<< b <<"\n";
            system("PAUSE");
            return 0;                                          }
void func2values (int x, int & y )
{           x=2*x; y=2*y;
            cout<<"oi times x kai y edo einai "<< x <<" "<< y <<"\n";      }
```

- Η κλήση με αναφορά είναι ένας αποτελεσματικός τρόπος επιστροφής περισσοτέρων της μίας τιμής συνάρτηση.
- Για να γίνει επιστροφή της τιμής κάποιων μεταβλητών χρησιμοποιώ για κάθε μία από τις μεταβλητές κλήση με αναφορά

Παράδειγμα

Να εμφανίζονται οι προηγούμενη και επόμενη τιμή μίας ακεραίας μεταβλητής

```
void func2(int, int &, int &);
// Dio  parametroi opou ginetai klisi me anafora
int main ()
{
        int a=200, prin, meta;
        cout << "The result is " << a<<"\n";
        func2(a, meta, prin);
        cout << "The result is " << a<<" "<<prin<<" "<<meta<<"\n";
        system("PAUSE");return 0;                                    }

void func2(int z , int & x, int & y )
{
        x=z+1; y=z-1;
        cout<<"oi times x kai y einai "<<x<<" "<<y<<"\n";            }
```

Εμβέλεια Μεταβλητών

- **Εμβέλεια (scope)** μίας μεταβλητής είναι ο χώρος σε ένα πρόγραμμα όπου μία μεταβλητή έχει οριστεί και επομένως μπορεί να χρησιμοποιηθεί.
- Μία μεταβλητή έχει εμβέλεια μόνο μέσα στο μπλοκ των εντολών που έχει οριστεί. Μπλοκ εντολών είναι μία ομάδα εντολών που περιέχεται σε αγκύλες { }. Μία τέτοια μεταβλητή θεωρείται τοπική (local) μεταβλητή.
- Αντίθετα μία **καθολική (global)** μεταβλητή είναι ορατή από οποιοδήποτε μέρος ενός προγράμματος. Μια καθολική μεταβλητή ορίζεται εκτός της συνάρτησης ή του main προγράμματος.

Παράδειγμα

```
int k=100; cout<<k<<endl;

{int k; k=200;
   cout<<k<<endl;
} // telos block entolon

cout<<"\nH timi is metablitis einai "<<k<<"\n";

int m=33;
{for(int m=2;m<=6;m++)
   cout<<m<<" ";
cout<<endl<<m<<" ";
} // telos block entolon


cout<<"\nH timi is metablitis einai "<<m<<"\n";
system("PAUSE");
return 0;
```

```
100
200
H timi is metablitis einai 100
2 3 4 5 6
33
H timi is metablitis einai 33
Press any key to continue . . .
```

Εμβέλεια Μεταβλητών (2)

- Οι μεταβλητές που ορίζονται μέσα σε επαναλήψεις ισχύον μόνο για την διάρκεια εκτέλεσηςτων επαναλήψεων
- Σε ένα μπλοκ εντολών μία τοπική μεταβλητή υπερισχύει μία καθολικής που έχει το ίδιο όνομα. Επίσης να σημειωθεί ότι Μετά το πέρας της κλήσης μίας συνάρτησης τυχόν τοπικές μεταβλητές που έχουν οριστεί μέσα στην συνάρτηση παύουν να υπάρχουν.

Παραδείγματα (εμβέλεια)

```
#include <iostream>
using namespace std;
int a=10, b=30;

//katholikes metablites
// me tis katholikes allazo dio times sto main

int func2values ( );

int main ()
{
   cout<<"Oi timi tou a="<<a<<endl;
   func2values();
   cout << "The result is " << a<<" "<<b<<"\n";
   system("PAUSE");
   return 0;
}

int func2values ( )
{
   a=50;
   a=2*a; b=2*b;
   cout<<"To a kai b einai "<<a<<" "<<b<<"\n";
   return (a);
}
```

```
Oi timi tou a=10
To a kai b einai 100 60
The result is 100 60
Press any key to continue . . .
```

```
int func2values (int , int );

int main ()
{
   int a=40,b=50;
   cout << "The result is "<< func2values(a,b)<<b<<"\n";
   system("PAUSE");
   return 0;
}

int func2values (int a, int b)
{

   a=2*a; b=2*b;
   int t=a+b;
   cout<<"oi times a  b  "<<a<<" "<<b<<"\n";
   cout<<"To t="<<t<<"\n";
   return (a);
}
```

```
oi times a  b  80 100
To t=180
The result is 8050
Press any key to continue . . .
```

Λυμένα Παραδείγματα Χρήσης Συναρτήσεων (1)

Λυμένο Παράδειγμα 1

Να γραφεί πρόγραμμα που να βρίσκει τον μεγαλύτερο από 2 ακέραιους. Η εισαγωγή των αριθμών καθώς και η εύρεση του μεγαλύτερου αριθμού να γίνονται με συναρτήσεις

```
using namespace std;

int max2(int ,int );

int dose1()                 //function
{
   int a;
   cout<<"dose 1 akereo ";
   cin>>a;
   return a;
}
//=========================================
int main(void)
{
   int k,l;
   k=dose1();
   l=dose1();
   cout<<"o megalyros apo tous dio einai ="<<max2(k,l)<<endl;
   system("PAUSE");
   return EXIT_SUCCESS;

}
//=========================================
int max2(int x,int y)
{
   if (x>y)
      return x;
   else
      return y;
}
```

Λυμένα Παραδείγματα Χρήσης Συναρτήσεων (2)

Λυμένο Παράδειγμα 2

Να γραφεί πρόγραμμα που να καλεί συνάρτηση που μετατρέπει τα δευτερόλεπτα σε ισοδύναμο χρόνο που εκφράζεται σε λεπτά και δευτερόλεπτα. Να εισάγεται ο αριθμός που αντιπροσωπεύει δευτερόλεπτα στο κύριο πρόγραμμα (πχ 65 sec = 1 min & 5 sec).

```
using namespace std;

//=========================================
int dose1()                    //function
{
   int xr;
   cout<<"dose ton xrono ";
   cin>>xr;
   return xr;
}
//=========================================
//definition of function
void sectomin(int x, int & a,int & b)
{
   a=x/60;
   b=x%60;
}
//============
int main( )
{
   int xronos,min,sec;
   xronos=dose1();
   sectomin(xronos,min,sec);
   cout<<"Ta "<<xronos<< " deftera einai "<<min;
   cout<<" min & "<<sec<<" deftera \n";
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

```
dose ton xrono 67
Ta 67 deftera einai 1 min & 7 deftera
Press any key to continue . . .
```