Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Σχολή Θετικών Επιστημών
Διατμηματικό Πρόγραμμα Μεταπτυχιακών Σπουδών (ΔΠΜΣ)
ΜΔΕ στον Ηλεκτρονικό Αυτοματισμό (ΗΑ)

Εισαγωγή στη C++

Γ. Λάτσας

Εισαγωγή στη C++

- Πρωτοαναπτύχθηκε από τον Bjarne Stroustrup στα Bell Labs το 1982. Αποτελεί εξέλιξη (υπερσύνολο) της γλώσσας C.
- Είναι low level γλώσσα, όπως η C, αλλά αρκετά πιο σύνθετη και περίπλοκη.
- Ενσωματώνει πληθώρα βιβλιοθηκών, ενώ επιτρέπει την εύκολη ανάπτυξη και ενσωμάτωση νέων βιβλιοθηκών.
- Επιτρέπει την επαναχρησιμοποίηση συμβόλων, ονομάτων, κλπ.
- Υποστηρίζει αντικειμενοστραφή προγραμματισμό (object-oriented programming).
- Έχει προτυποποιηθεί, ενώ συνεχίζει να αναπτύσσεται και να προτυποποιείται συνεχώς. Τελευταίο πρότυπο το 2021 (ISO/IEC TS 23619:2021).
- Υψηλή συμβατότητα μεταξύ των διαφόρων compilers.
- Ιδιαίτερα διαδεδομένη. Χρησιμοποιείται σε μεγάλη γκάμα εφαρμογών.

Εισαγωγή στη C++ (2)

- **Βασικά πλεονεκτήματα:**
  - υπερφόρτωση συναρτήσεων
  - χώροι ονομάτων
  - κλάσεις
  - ενθυλάκωση δεδομένων (encapsulation)

    → ενσωμάτωση δεδομένων και κώδικα ώστε να αποτελούν αυτόνομο "μαύρο κουτί"
  - πολυμορφισμός

    → χρήση ενός interface με διαφορετικές μεθόδους (function overloading, templates)
  - κληρονομικότητα

    → δημιουργία νέων κλάσεων από ήδη υπάρχουσες

Βιβλιογραφία

Ενδεικτικά:

- Γ. Σ. Τσελίκης, C++ Από τη θεωρία στην εφαρμογή
- P. Dietel, H. Dietel, C++ How to program
- H. Schildt, C++ from the ground up
- www.cplusplus.com

Compilers – IDEs

- Οι πιο διαδεδομένοι compilers:
  - GNU compiler: g++ (για περιβάλλον Linux)
  - MinGW (έκδοση των gcc, g++ για Windows)
  - Microsoft Visual Studio (Microsoft C++ compiler)
  - Intel C++ compiler

- Γραφικά περιβάλλοντα IDEs
  - Microsoft Visual Studio
  - Code::Blocks
  - Dev-C++

Ένα απλό πρόγραμμα – Σύγκριση με C

Γλώσσα C (hello.c)

```c
#include <stdio.h>

int main()
{
    printf("Hello World\n");
    return 0;
}
```

Γλώσσα C++ (hello.cpp)

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

int main()
{
    cout << "Hello World" << endl;
    // ή εναλλακτικά
    cout << "Hello World\n";
    return 0;
}
```

header files

- Περιέχουν ορισμούς:
  - σταθερών
  - μεταβλητών
  - συναρτήσεων
  - macros

- Δεν περιέχουν (εν γένει):
  - κώδικα υλοποίησης συναρτήσεων.
    - Ο κώδικας υλοποίησης βρίσκεται είτε στις αντίστοιχες βιβλιοθήκες, είτε στα αρχεία κώδικα του προγράμματος.

- Όταν σε ένα αρχείο .cpp υπάρχει η οδηγία `#include <header.h>`, τότε στο σημείο αυτό ενσωματώνεται ολόκληρο το αρχείο **`header.h`** προτού σταλεί το αρχείο .cpp στον compiler.

Βασικά στοιχεία προγραμματισμού σε C++

- Η οδηγία `#include`
  - `#include <header.h>`: Το αρχείο θα αναζητηθεί στους προκαθορισμένους φακέλους του συστήματος για τα header files.
  - `#include "header.h"`: Το αρχείο θα αναζητηθεί στον τρέχοντα φάκελο. Μπορεί να χρησιμοποιηθεί επίσης σχετικό ή απόλυτο path.
- Σχόλια
  - `// Σχόλιο μιας γραμμής`
  - `/* Αρχή σχολίου…`

    `……`

    `…τέλος σχολίου */`
- Τα ονόματα (εντολές, μεταβλητές, κλπ.) είναι case sensitive
  - `x=3;`
  - `X=5; //διαφορετική μεταβλητή από τη x`

Βασικά στοιχεία προγραμματισμού σε C++ (2)

- Στο τέλος κάθε εντολής τοποθετείται άνω τελεία ";". Αυτό επιτρέπει μια εντολή να χωρίζεται σε περισσότερες σειρές χωρίς πρόβλημα.
- Μπορούν να χρησιμοποιούνται εσοχές κατά το δοκούν. Συνήθως χρησιμοποιούνται εσοχές για να βοηθούν την ανάγνωση του κώδικα (να οριοθετούν νοητά ενότητες ή block του κώδικα).
- Ένα block κώδικα οριοθετείται από brackets { }. Ένα block περιλαμβάνει ένα πλήθος εντολών.
- Μπορούν να χρησιμοποιηθούν κενά για να κάνουν πιο ευανάγνωστο τον κώδικα.

```
int main()
{ …
   if (a < b) //ή if(a<b)
   {
      A = 10 * 2 + 3; //ή A=10*2+3;
   }
   cout << "Some text …" << a << " text " << A*sin(2.0*PI*frequency*time)+
                                             (10*T)/2;
   …
}
```

Η συνάρτηση main()

- Η συνάρτηση main αποτελεί το κυρίως πρόγραμμα. Είναι η συνάρτηση που θα εκτελεστεί.
- Μπορεί να παίρνει ορίσματα, τα οποία ορίζονται μέσα στις παρενθέσεις.
- Μπορεί να επιστρέφει κάποια τιμή ή όχι.
- Ο κώδικας της main περιλαμβάνεται σε brackets { … }.
  - `void main() { … }`
  - `int main(int argc, char *argv[])`

```
{   …
    return 0;
}
```

- Τα ορίσματα **`argc`**, **`argv[]`** περιέχουν το πλήθος και τις παραμέτρους της εντολής εκτέλεσης του προγράμματος. Πχ. Αν η εντολή εκτέλεσης είναι:

```
program param1 param2 param3
```

τότε: `argc=4, *argv[]={"program", "param1", "param2", "param3"}`

Χώροι ονομάτων (namespaces)

- Για να μην υπάρχει περιορισμός στο πλήθος των διαφορετικών ονομάτων που μπορούν να οριστούν (πχ. ως μεταβλητές, σταθερές, συναρτήσεις, κλπ.), καθώς και για να μην υπάρχει πρόβλημα αν ένα όνομα έχει οριστεί σε μια βιβλιοθήκη και οριστεί και στο πρόγραμμα, η C++ ορίζει χώρους ονομάτων.
- Όλα τα ονόματα των βασικών βιβλιοθηκών της C++ είναι ορισμένα στον χώρο std.
- Κάποια βιβλιοθήκη μπορεί να ορίσει δικό της χώρο ονομάτων (π.χ. customspace)
- Η χρήση ενός ονόματος "name" σε κάποιο namespace γίνεται ως `namespace::name` (π.χ. `std::cout`).
- Η οδηγία `using namespace std;` σημαίνει ότι στο εξής θα χρησιμοποιείται ο χώρος `std`, οπότε η κλήση σε οποιαδήποτε συνάρτηση μπορεί να γίνει κατευθείαν (π.χ. `cout`) .

Δεσμευμένα ονόματα

```
alignas
alignof
and
and_eq
asm
auto
bitand
bitor
bool
break
case
catch
char
char16_t
char32_t
class
compl
const
constexpr
const_cast
continue
decltype
default
delete
do
double
dynamic_cast
else
enum
explicit
export
extern
false
float
for
friend
goto
if
inline
int
long
mutable
namespace
new
noexcept
not
not_eq
nullptr
operator
or
or_eq
private
protected
public
register
reinterpret_cast
return
short
signed
sizeof
static
static_assert
static_cast
struct
switch
template
this
thread_local
throw
true
try
typedef
typeid
typename
union
unsigned
using
virtual
void
volatile
wchar_t
while
xor
xor_eq
override*
final*
```

Οδηγίες (directives)

- #include (ενσωμάτωση ενός header file)
- #define (ορισμός ενός macro)
- #undef (αναίρεση προηγούμενου ορισμού)
- #ifdef , #ifndef (έλεγχος αν έχει οριστεί κάτι)
- #if , #elif , #else , #endif (γενικός έλεγχος if, else if)
- #if defined , #if !defined (έλεγχος αν έχει οριστεί κάτι)

Παραδείγματα:

- `#define PI 3.14159265`
- `#define MIN(a,b) (((a)<(b)) ? a : b)`

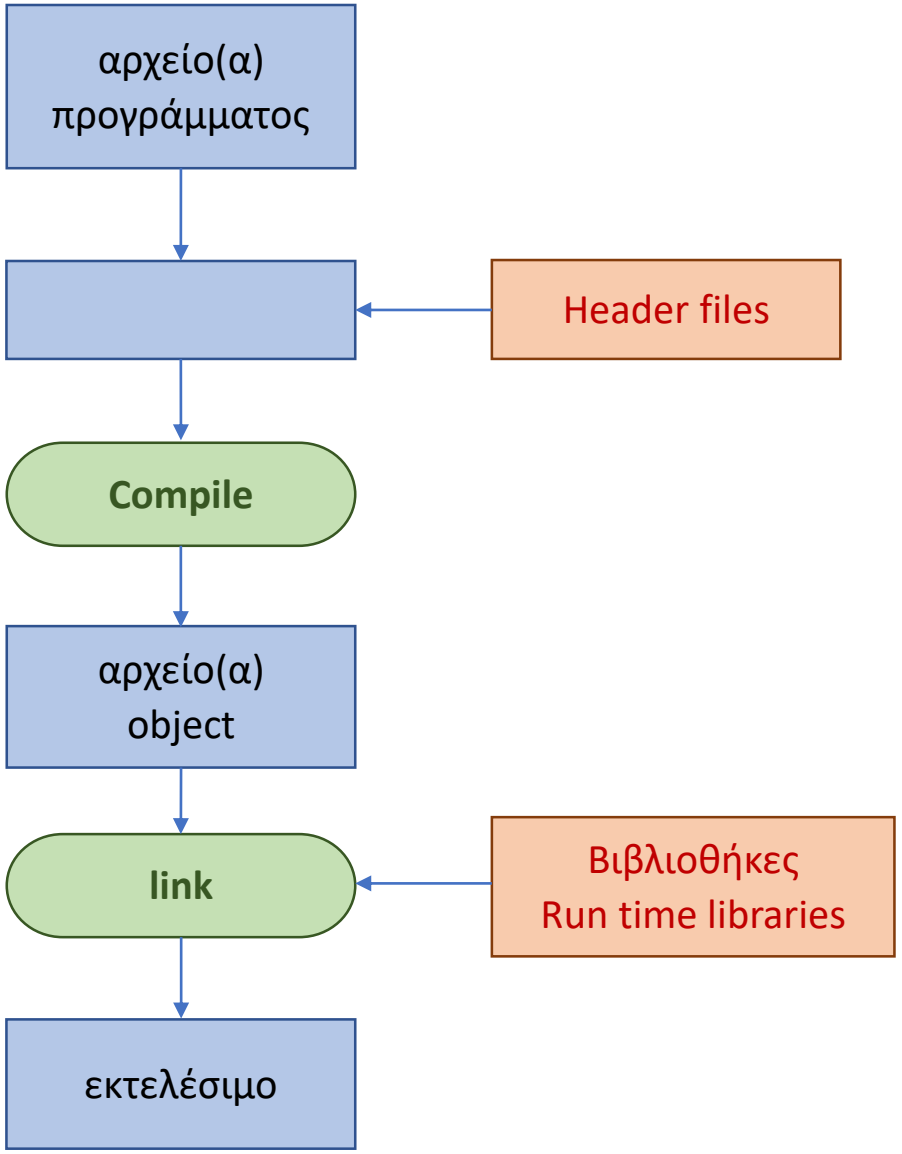
  - Οι οδηγίες απευθύνονται στον preprocessor, ο οποίος προετοιμάζει τα source αρχεία προτού τα στείλει στον compiler.
  - Όταν ο compiler συναντήσει το PI ή το MIN, θα τα αντικαταστήσει με την ποσότητα που έχει οριστεί στην αντίστοιχη οδηγία `#define`.

Compiling and linking

GCC (ή MinGW)

```
g++ -o out.exe source1.cpp source2.cpp
```

- Ο compiler ενσωματώνει στα source αρχεία (.cpp) τα header files και κάνει compile, παράγοντας αρχεία object (.o).
- Ο linker συνδέει τα αρχεία object, καθώς και τυχόν βιβλιοθήκες που χρησιμοποιεί το πρόγραμμα και παράγει το εκτελέσιμο.



Βασικοί τύποι δεδομένων

Τύπος	Μέγεθος (bytes)	Τιμές
bool	1	false/true
unsigned short int	2	0…65535
unsigned int	4	0…4294967295
unsigned long int	4	0…4294967295
short int	2	-32768…32767
int	4	-2147483648…2147483647
long int	4	-2147483648…2147483647
float	4	±1.17549e-38…±3.40282e+38
double	8	±2.22507e-308…±1.79769e+308
long double	10, 16	±2.22507e-308…±1.79769e+308
char	1	-128…127
unsigned char	1	0…255
wchar_t	2	-32768…32767

Δήλωση μεταβλητών

- Οι μεταβλητές πρέπει να έχουν δηλωθεί προτού χρησιμοποιηθούν.
- Μπορεί να είναι global ή ορισμένες τοπικά.

```
#include <iostream>
#include <iomanip>

using namespace std;
int a;        //global μεταβλητή

int main() {
     int i, j, k;          //Τοπικές (local) μεταβλητές στη main.
     float var1, var2;
     …
     return 0;
}
```

Σταθερές

20	**int**
20u	**unsigned int**
20l	**long**
20ul	**unsigned long**
0115	**οκταδικό**
0x1A	**δεκαεξαδικό**
0b10101010	**δυαδικό (gcc)**
3.14159	**double**
3.14159L	**long double**
'A'	**char**

- Με την οδηγία define:

```
#define ELECTRON_CHARGE 1.6e-19
```

→ Δεν είναι μεταβλητή αλλά σταθερά του προγράμματος. Είναι global.

- Ως σταθερή μεταβλητή:

```
const int Nsize=100;
```

→ Είναι μεταβλητή, αρχικοποιείται κατά τη δήλωση και δεν μπορεί να αλλάξει στη συνέχεια. Μπορεί να είναι global ή local.

Μετατροπές τύπων

Έμμεσες μετατροπές

```
float x;
int y,k;
double z;
z=x+y;          //η y μετατρέπεται σε float για να γίνει η πράξη.
k=z+x;          //η z+x μετατρέπεται σε int. Ο compiler πιθανό να βγάλει μήνυμα: warning
                  C4244: '=' : conversion from 'double' to 'float', possible loss of data.
```

Σειρά μετατροπής:

long double ← double ← float ← unsigned long ← long ← unsigned int ← int

Ρητές μετατροπές

```
k=(int)x;       //Ρητή μετατροπή της τιμής του x σε ακέραιο (C).
k=int(z);       //Ρητή μετατροπή της τιμής του z σε ακέραιο (C++).
```

Σημείωση: Οι μεταβλητές διατηρούν τον τύπο τους. Δηλαδή μετατρέπονται προσωρινά μόνο οι τιμές τους.

Είσοδος – Έξοδος

```
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
    double A;
    int a;
    float x,y;
    a = 10;
    x = 125258.73;

    cout << "Enter a number: ";
    cin >> y;
    cout << setw(5) << a << endl;
    cout << setprecision(3) << x <<endl;
    cout << setprecision(8) << x <<endl;
    cout << "y=" << y << endl;
    cout << "y=" << setprecision(4) <<
                y << endl;
    cout << scientific << "y=" << y <<
                "=" << fixed << y << '\n';
    return 0;
}
```

setprecision(n): ορίζει το πλήθος σημαντικών ή δεκαδικών ψηφίων.
setw(n): ορίζει το εύρος της εξόδου σε χαρακτήρες.
fixed/scientific: ορίζει τη μορφή εξόδου.

Έξοδος προγράμματος

```
Enter a number: 299792458
___10
1.25e+005
125258.73
y=2.9979245e+008
y=2.998e+008
y=2.9979e+008=299792448.0000
```

Τελεστές

Αριθμητικοί τελεστές

=	Ανάθεση τιμής
+	Πρόσθεση
-	Αφαίρεση
*	Πολλαπλασιασμός
/	Διαίρεση
%	Modulo (υπόλοιπο ακέραιας διαίρεσης)
++	Αύξηση κατά ένα
--	Μείωση κατά ένα

Σχεσιακοί τελεστές

>	Μεγαλύτερο
<	Μικρότερο
>=	Μεγαλύτερο ή ίσο
<=	Μικρότερο ή ίσο
==	Έλεγχος ισότητας
!=	Έλεγχος ανισότητας

Τελεστές (2)

Λογικοί τελεστές

&&	AND
||	OR
!	NOT

Δυαδικοί τελεστές

&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT

Συνδυαστικοί Τελεστές

+=	`y+=x` ⇔ `y=y+x`
-=	`y-=x` ⇔ `y=y-x`
*=	`y*=x` ⇔ `y=y*x`
/=	`y/=x` ⇔ `y=y/x`
%=	`y%=x` ⇔ `y=y%x`

Παραδείγματα τελεστών

```
int x=3, y=2;
y=x++;          // το y θα πάρει την τιμή του x και μετά το x θα αυξηθεί κατά ένα.
                // x=4, y=3
y=++x;          // το x θα αυξηθεί κατά ένα και το αποτέλεσμα θα ανατεθεί στο y.
                // x=5, y=5

float x=3.1;
x+=2.2;         // x=5.3
x*=2;           // x=10.6

(a >= 5) && (a <= 10)          // 5 ≤ a ≤ 10   (δηλ. a∈[5,10])
(a < 5) || (a > 10)            // a<5  ή   a > 10 (δηλ. a∉[5,10])
!((a >= 5) && (a <= 10))       //  a ∉[5,10]
```

Προτεραιότητα τελεστών

1	::	Left-to-right
2	a++ a-- . ->	
3	++a --a ! ~ *a &a	Right-to-left
4	.* ->*	Left-to-right
5	a*b a/b a%b	
6	a+b a-b	
7	<< >>	
8	< <= > >=	
9	== !=	
10	a&b	
11	^	
12	|	
13	&&	
14	||	
15	= += -= *= /= %=	Right-to-left

- Χρήση παρενθέσεων

$(a+b)\frac{k+m}{f+g}$ → `(a+b)*(k+m)/(f+g)`

$\frac{a+\frac{b+d}{c\,j}\,q}{k+m}$ → `(a+((b+d)/(c*j))*q))/(k+m)`

Η παρένθεση με κόκκινο είναι περιττή, αλλά δεν βλάπτει. Μπορεί να χρησιμοποιηθεί για ασφάλεια.

Η βιβλιοθήκη <cmath>

Ορίζει χρήσιμες μαθηματικές συναρτήσεις. Ενδεικτικά:

- `abs(x)` → απόλυτη τιμή
- `exp(x)` → εκθετική συνάρτηση
- `log(x)` → φυσικός λογάριθμος
- `pow(x,y)`, `y`: float, double ή int → δύναμη x^y
- `sqrt(x)` → τετραγωνική ρίζα
- `sin(x)`, `cos(x)`, `tan(x)` → ημίτονο, συνημίτονο, εφαπτομένη
- `asin(x)`, `acos(x)`, `atan(x)`, `atan2(y,x)` → αντίστροφο ημίτονο, συνημίτονο, εφαπτομένη, `y`: float ή double
- `ceil(x)` → ο αμέσως μεγαλύτερος ακέραιος του `x`
- `floor(x)` → ο αμέσως μικρότερος ακέραιος του `x`

  - `x`: float ή double

Μιγαδικοί αριθμοί

- `#include <complex>`
- Οι μιγαδικοί αριθμοί είναι ορισμένοι ως κλάση με πρότυπες συναρτήσεις για κάθε τύπο (float, double, long double).
- Ορίζονται όλες οι συνήθεις μαθηματικές πράξεις και συναρτήσεις και έχει γίνει υπερφόρτωση όλων των απαιτούμενων τελεστών.

```
complex<double> a,b,ci=complex<double>(0.0,1.0);
double x,y,m,f;

a=complex<double>(1.2,3.4);     //a=1.2 + i3.4
b=complex<double>(5.0,2.1);     //b=5.0 + i2.1
x=real(a); //πραγματικό μέρος του a
y=imag(b); //φανταστικό μέρος του a
m=abs(a);  //μέτρο του a
f=arg(b);  //γωνία του b
b=conj(a); //συζυγής του a
b=polar(m,f);     //ορισμός τιμής σε πολική μορφή
```

Οι εντολές if - else if - else

```
if(συνθήκη1)
    {… εντολές…}
else if(συνθήκη2)
    {…εντολές…}
else
    {…εντολές…}
```

- Τα `else if` και `else` είναι προαιρετικά.
- Τα { } μπορεί να λείπουν αν περιέχουν μόνο μία εντολή
- Μετά τα `if()`,`else if()`,`else()` δεν μπαίνει άνω τελεία ';'.
- Εναλλακτική σύνταξη:

```
var=(cond) ? ifTrue : ifFalse;
```

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

int main()
{
  int a=-1, b=0;
  if(a<b && a<0)
  {
      cout<<"Yes";
  }
  else
  {
      cout<<"No";
  }
  return 0;
}
```

Παράδειγμα if - else

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

int main() {
   int tmp1,tmp2,code=1234;
   cout << "Enter PIN: ";
   cin >> tmp1;

   if(tmp1 == code)
   {
      cout << "Correct PIN.";
      cout << "Enter new PIN";
      cin >> tmp1;
      cout << "Enter new PIN again";
      cin >> tmp2;

      if(tmp1 == tmp2)
      {
         code=tmp2;
         cout << "New PIN stored\n";
      }
      else
         cout << "PIN mismatch\n";
   }
   else
      cout << "Wrong PIN\n";
   return 0;
}
```

Σημείωση:

- Στον έλεγχο ισότητας χρησιμοποιούμε τον τελεστή '==', και όχι τον '='
- Αν γράψουμε "*tmp1=code*" θα γίνει ανάθεση της τιμής *code* στην *tmp1*.
- Το αποτέλεσμα της ανάθεσης θα είναι ***true*** εφόσον η ανάθεση γίνει επιτυχώς. Οπότε η συνθήκη στο if() θα ικανοποιείται πάντα!

Παράδειγμα if - else if - else

```
#include <iostream>
using namespace std;
int main() {
   double x,y,z;
   char oper;
   bool success=true;
   cout << "Enter operand1 operator operand2 :";
   cin >> x >> oper >> y;
   if(oper == '+') z = x+y;
   else if(oper == '-') z = x-y;
   else if(oper == '*') z = x*y;
   else if(oper == '/' && y != 0) z = x/y;
   else
      success = false;
   if(success)
      cout << x << oper << y << " = " << z << endl;
   return !success;
}
```

Η εντολή for

- Σύνταξη: `for(init;expr;incr) {…}`
  - *init*: αρχικοποίηση μεταβλητής, π.χ. i=1
  - *expr*: συνθήκη τερματισμού, π.χ. i<=10
  - *incr*: μεταβολή μεταβλητής σε κάθε επανάληψη, π.χ. i=i+1, i++, i=i-2, κλπ.
- Η μεταβλητή i μπορεί να οριστεί μέσα στην έκφραση αρχικοποίησης.

```
for (int i=0;i<100;i++) {…}
```

- Η μεταβλητή δεν είναι απαραίτητα ακέραιος.

```
for (double x=0;x<10;x+=0.5) {…}
```

- Η μεταβλητή μπορεί να αυξάνεται ή να μειώνεται.

```
for (i=100;i>0;i-=10) {…}
```

- Αν στο βρόχο περιέχεται μία εντολή, τα { } μπορούν να παραλειφθούν.

```
for (i=0;i<10;i++) cout<<i<<endl;
```

Η εντολή for (2)

- Μπορεί να χρησιμοποιηθούν δύο ή περισσότερες μεταβλητές και η συνθήκη να είναι περίπλοκη:

```
for (i=0, k=1; i+k<=100; i++,k+=2)
{
   cout<<i<<" "<<k<<" "<<i+k<<endl;
}
```

ή

```
for (i=0, k=10 ; i<=10 && k>2; i++, k-=2)
   cout<<i<<" "<<endl;
```

while() {...} και do {...} while()

```
int main()
{
    int i=10;
    while(i>0)
    {
        cout << i << endl;
        i--;
    }
    return 0;
}
```

```
int main()
{
    int i=10;
    do
    {
        cout << i << endl;
        i--;
    } while(i>0);
    return 0;
}
```

- Ενώ φαινομενικά τα δυο προγράμματα λειτουργούν με τον ίδιο τρόπο, έχουν μια ουσιώδη διαφορά: στο αριστερό ο έλεγχος της συνθήκης γίνεται στην αρχή, ενώ στο δεξιό στο τέλος. Αν η συνθήκη δεν ικανοποιείται εξ αρχής, το δεξιό θα εκτελεστεί τουλάχιστον μία φορά, ενώ το αριστερό όχι.

Η εντολή switch

```
switch(expression)
{
   case const1:
      εντολές;
      break;
   case const2:
      εντολές;
      break;
   …
   default:
      εντολές;
}
```

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

int main()
{
   int a;
   cout<<"Enter a number:";
   cin>>a;
   switch(a)
   {
      case 1:
         cout<<"One\n";
         break;
      case 2:
         cout<<"Two\n";
         break;
      default:
         cout<<"Other\n";
   }
return 0;
}
```

Εντολές break και continue

- Η εντολή break μπορεί να χρησιμοποιηθεί για να τερματιστεί πρώιμα ένας βρόχος, πχ. εφόσον ικανοποιηθεί κάποια συνθήκη

```
for(int i=1;i<100;i++)
{…
   if(some condition) break;
}
```

- Η εντολή continue δίνει οδηγία να εκτελεστεί η επόμενη επανάληψη του βρόχου. Οι εντολές που απομένουν δεν εκτελούνται στην παρούσα επανάληψη.

```
for(int x=0;x<=100;x++)
{
   if(x%2) continue; //Αν ο x είναι περιττός…
   cout << x << ' '; //Τυπώνονται μόνο οι άρτιοι αριθμοί
}
```

Υπολογισμός του π

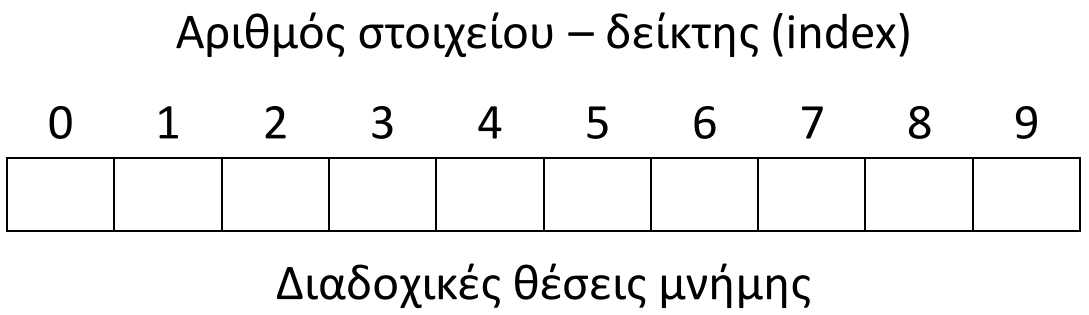
Γινόμενο Wallis:

$$\frac{\pi}{2} = \prod_{i=1}^{n} \frac{2i}{2i-1} \frac{2i}{2i+1}$$

```
int main()
{
   double product1=1.0;
   double x;
   for (int i=1;i<20000;i++)
   {
      x=(double)(2*i);
      product1*=(x/(x-1.0))*(x/(x+1.0));
   }
   cout<<"Product = "<<product1<<endl;
   cout<<"Pi = "<<product1 * 2.0<<endl;
   return 0;
}
```

Μονοδιάστατοι πίνακες

```
#define M 10
const int n=10;
int k = 10;
int a[10];
int b[n];
int c[M];
int d[k];        //ΛΑΘΟΣ: δεν επιτρέπεται γιατί το k δεν είναι σταθερό.
                 //(Επιτρέπεται όμως για local πίνακες).
```



- Οι πιο πάνω πίνακες έχουν κελιά αριθμημένα από το 0 έως το 9. Η C++ δεν ελέγχει τα όρια. Έτσι θέλει προσοχή να μην βγούμε εκτός του χώρου μνήμης του πίνακα.

Αρχικοποίηση πίνακα:

```
int x[10]={9, 8, 7, 6, 5, 4, 3, 1, 2, 3}
int y[10]={0}      //όλα τα στοιχεία μηδενικά
```

Πίνακες πολλών διαστάσεων

- Δήλωση πίνακα

```
const int row=3,col=5;
int x[row][col];
```

Γραμμή 0					Γραμμή 1					Γραμμή 2				
0	1	2	3	4	0	1	2	3	4	0	1	2	3	4
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

- Αρχικοποίηση στοιχείων:

```
int x[3][5]={{1,2,8,33,51},{4,12,3,21,9},{7,5,2,45,6}}
```

- Πίνακες 3 διαστάσεων

```
double y[5][6][3]={0};
```

- Κλήση στοιχείων πίνακα:

```
x[5][4], y[3][4][2]
```

Παράδειγμα με πίνακες

Πρόγραμμα υπολογισμού ελάχιστης, μέγιστης, μέσης τιμής και τυπ. απόκλισης.

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
   const int MAX=100;
   double x[MAX];
   int n;
   do
   {
      cout << "Enter number of data points [2 to " << MAX << "] : ";
      cin >> n;
      cout << endl;
   } while ( (n < 2) || (n > 100) );
   for (int i=0 ; i < n ; i++)
   {
      cout << "x["<< i+1 << "] = ";
      cin >> x[i];
      cout << endl;
   }
```

Παράδειγμα με πίνακες (2)

```
   double min_value=x[0],max_value=x[0],mean_value=0.0;
   for(int j=0 ; j < n ;j++)
   {
      if (x[j] < min_value) min_value = x[j];
      if (x[j] > max_value) max_value = x[j];

      mean_value += x[j];
   }
   mean_value = mean_value/n;

   double standard_deviation = 0.0;
   for(int j=0 ; j < n ; j++)
      standard_deviation += pow(x[j]-mean_value,2);

   standard_deviation = sqrt(standard_deviation/n);
   cout << "Minimun = " << min_value << endl;
   cout << "Maximum = " << max_value << endl;
   cout << "Mean value = " << mean_value << endl;
   cout << "Standard deviation = " << standard_deviation << endl;
   return 0;
}
```

Δείκτες

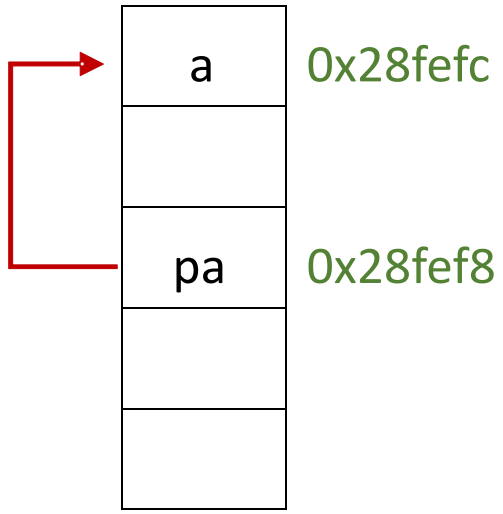
- Μεταβλητή που αποθηκεύει διεύθυνση μνήμης.

  `int *pa, a;`

  - `*pa` → το περιεχόμενο της μνήμης στη θέση pa
  - `&a` → η διεύθυνση μνήμης της a

  - `*pa=a` → στη διεύθυνση μνήμης που δείχνει ο pa τοποθετείται η τιμή της a
  - `pa=&a` → ο pa δείχνει στη θέση μνήμης του a

- Ο δείκτης πρέπει να είναι του ίδιου τύπου με τη μεταβλητή. Έτσι ο compiler γνωρίζει το εύρος της θέσης μνήμης ανάλογα με τον τύπο.



Δείκτες και μονοδιάστατοι πίνακες

- Όταν ορίζουμε έναν πίνακα a, το **a** είναι στην ουσία pointer που δείχνει στη διεύθυνση του πρώτου στοιχείου το πίνακα.
- Μπορούμε ισοδύναμα να χρησιμοποιήσουμε pointer για να προσπελάσουμε τα στοιχεία του.

```
int *p, a[10];
p=a;      //ο p δείχνει στο πρώτο στοιχείο του a
```

- Η προσπέλαση του στοιχείου 3 μπορεί να γίνει με τον δείκτη p ως:

  `*(p+3)` ή ισοδύναμα ως

  `p[3]`

- Οι δείκτες προσφέρουν μεγάλη δύναμη και ελευθερία στον προγραμματισμό.
- Θέλουν πολύ προσοχή γιατί επιτρέπεται να δείχνουν σχεδόν οπουδήποτε, οπότε είναι εύκολο να γίνουν λάθη που είναι δύσκολα εντοπίσιμα.

Δείκτες και πίνακες δύο διαστάσεων

- Ένας πίνακας δύο διαστάσεων (πχ. a[10][10]) στη C++ είναι σας ένας μονοδιάστατος πίνακας με στοιχεία μονοδιάστατους πίνακες των δέκα στοιχείων έκαστο.
- Αποθηκεύεται σε συνεχόμενες θέσεις μνήμης (στοιχεία της 1ης γραμμής, μετά της 2ης γραμμής, κοκ.).

```
int (*p)[3], a[3][3]; // ο p είναι pointer τύπου μονοδιάστατου
                      //πίνακα τριών στοιχείων
p=a;  //ο p δείχνει στο πρώτο στοιχείο του a
```

- Προσπέλαση του στοιχείου i,j

```
int i=2,j=1;
cout<<p[i][j];
cout<<*(p[i]+j);
cout<<*(*(p+i)+j);
```

	Στήλη 0	Στήλη 1	Στήλη 2
Γραμμή 0	a[0][0]	a[0][1]	a[0][2]
Γραμμή 1	a[1][0]	a[1][1]	a[1][2]
Γραμμή 2	a[2][0]	a[2][1]	a[2][2]

Πρότυπη κλάση vector

- Πρότυπη κλάση για τον ορισμό διανυσμάτων (πινάκων) με δυναμική διαχείριση μνήμης και δυνατότητα μεταβολής μεγέθους.
- Εκτελεί αρχικοποίηση (μηδενισμό) των στοιχείων.
- Περιέχει πλήθος συναρτήσεων, και έχουν υπερφορτωθεί οι τελεστές "=" και "[ ]".
- Δήλωση:

```
vector<type> name(size);
```

- Μερικές ενδεικτικές συναρτήσεις:
  - `push_back(value)` : προσθήκη στοιχείου στο τέλος και ανάθεση σε αυτό της τιμής value.
  - `pop_back()` : διαγραφή τελευταίου στοιχείου.
  - `resize(size)` : αλλαγή μεγέθους-πλήθους στοιχείων
  - `size()` : επιστρέφει το μέγεθος (το πλήθος των στοιχείων)
  - `clear()` : αδειάζει το διάνυσμα αποδεσμεύοντας τη μνήμη.

Παράδειγμα με vector

```
#include <iostream>
#include <vector> //header file που ορίζει την κλάση vector
using namespace std;
int main() {
   int isize=10;
   vector<int> myints;      //Δήλωση διανύσματος ακεραίων

   cout<<myints.size()<<endl;      //μέγεθος: 0 στοιχεία

   myints.resize(10);              //Αλλάζει το μέγεθος σε 10 στοιχεία

   myints.insert(myints.end(),10,0);     //Προσθέτει στο τέλος 10 μηδενικά
                                         //στοιχεία (μέγεθος: 20)
   myints.push_back(0);      //Προσθέτει στο τέλος ένα μηδενικό στοιχείο
                             //(μέγεθος: 21)
   myints.pop_back();        //Διαγράφει το τελευταίο στοιχείο (μεγεθος: 20)

   myints.clear();           //Διαγράφει όλα τα στοιχεία (μεγεθος: 0)
   return 0;
}
```

Πρότυπη κλάση vector – Διδιάστατοι πίνακες

- Ένας διδιάστατος πίνακας μπορεί να οριστεί ως διάνυσμα διανυσμάτων, δηλαδή vector που έχει ως στοιχεία vectors.

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
   int isize=2, jsize=3;
   vector<vector<double>> dx(isize, vector<double>(jsize));
// Το dx είναι διάνυσμα διανυσμάτων τύπου double.

   dx={ {1,2,3}, {4,5,6} };        //Αρχικοποίηση στοιχείων

   for (int i=0; i<isize; i++)
      for (int j=0; j<jsize; j++)
          cout << dx[i][j] << endl;

   return 0;
}
```

Χαρακτήρες και strings

- Όπως και στη C, ο βασικός τύπος για αλφαριθμητικούς χαρακτήρες είναι ο char. Για strings χρησιμοποιείται πίνακας χαρακτήρων char *.
- Ο πίνακας χαρακτήρων τερματίζεται πάντα (αυτόματα) με τον μηδενικό χαρακτήρα (null character, '\0').

```
char str1[]="This is a text.";
```
→

T	h	i	s		i	s		a		t	e	x	t	.	\0

```
char str2[18]="Another text.";
```
→

A	n	o	t	h	e	r		t	e	x	t	.	\0				

```
cout << str2;
```

```
cin >> str2;
```
→ Διαβάζει ένα string μέχρι το πρώτο κενό (space).

```
cin.getline(str2,sizeof(str2));
```
→ Διαβάζει ολόκληρη γραμμή, με μέγιστο πλήθος χαρακτήρων το μέγεθος του str2.

Χαρακτήρες και strings

Βασικές συναρτήσεις διαχείρισης (#include <cstring>) (ή <string.h>)

- **char* strcpy**(*s1*, *s2*) → Αντιγραφή από το *s2* στο *s1*.
- **char* strncpy**(*s1*,*s2*, *N*) → Αντιγραφή από το *s2* στο *s1 τους πρώτους N χαρακτήρες*.
- **char* strcat**(*s1*, *s2*) → Προσθήκη του *s2* στο τέλος του *s1*.
- **int strcmp**(*s1*, *s2*) → Σύγκριση δύο αλφαριθμητικών.
- **int strncmp**(*s1*, *s2*,*N*) → Συγκρίνει του πρώτους N χαρακτήρες δύο αλφαριθμητικών.
- **size_t strlen**(*s*) → Μήκος του s σε χαρακτήρες.

Παράδειγμα με strings

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
   char c,s1[50],s2[50];
   cout<<"Enter a string:\n";
   cin.getline(s2,sizeof(s2));

   cout << "You entered " << strlen(s2) << " characters" << endl;
   strcpy(s1,s2);
   cout << "s1 = " << s1 << endl << "s2 = "<< s2 << endl;
   cout << "compare s1 , s2 : " << strcmp(s1,s2) << endl;
   strcat(s1,"B");
   strcat(s2,"C");
   cout << "s1 = " << s1 << endl << "s2 = " << s2 << endl;
   cout << "compare s1 , s2 : " << strcmp(s1,s2) << endl;
   strcat(s2,"ABC");
   cout << "s1 = " << s1 << endl << "s2 = " << s2 << endl;
   cout << "compare s1 , s2 : " << strcmp(s1,s2) << endl;

   return 0;
}
```

Πίνακες δεικτών

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

int main()
{
   char* month[]={"JANUARY","FEBRUARY","MARCH","APRIL","MAY","JUNE",
      "JULY","AUGUST","SEPTEMBER","OCTOBER","NOVEMBER","DECEMBER"};
   for(int i=0; i<12; i++)
      cout << *month[i] << endl;

   for(int j=0; j<12 ;j++)
      cout << month[j] << " "
               << strlen(month[i]) << endl;

   return 0;
}
```

J	A	N	U	A	R	Y	\0	
F	E	B	R	U	A	R	Y	\0
M	A	R	C	H	\0			
A	P	R	I	L	\0			
M	A	Y	\0					
J	U	N	E	\0				
J	U	L	Y	\0				
A	U	G	U	S	T	\0		
S	E	P	T	E	M	B	E	R	\0
O	C	T	O	B	E	R	\0	
N	O	V	E	M	B	E	R	\0
D	E	C	E	M	B	E	R	\0

Η πρότυπη κλάση string

- Η C++ περιέχει τη βιβλιοθήκη string όπου έχει οριστεί η πρότυπη κλάση string.
- Παρέχει μεγάλη ευκολία και ευελιξία στη διαχείριση αλφαριθμητικών.
- Κάνει αυτόματα διαχείριση της μνήμης και περιορίζονται τα λάθη.
- Υπερφορτωμένοι τελεστές: =, +, ==, !=, <, <=, >, >= <<, >>

```
#include <iostream>
#include <string>
using namespace std;
int main() {
   string s1 = "This is a text.", s2 = "Another text.", s3;
   s3 = s1;                    //s3: "This is a text."
   s3 = s1 + " " + s2;         //s3: "This is a text. Another text."
   s3 += "!!!";                //s3: "This is a text. Another text.!!!"
   getline(cin,s1);  //ανάγνωση ολόκληρης γραμμής.
   cout << s3 << endl;
   cout << s1 << endl;
   return 0;
}
```

Συναρτήσεις

- Η συνάρτηση είναι ένα ανεξάρτητο τμήμα κώδικα με όνομα.
- Μπορεί να παίρνει παραμέτρους (ορίσματα).
- Μπορεί να επιστρέφει κάποια τιμή.
- Παράδειγμα συνάρτησης είναι η main.
- Μια συνάρτηση πρέπει να έχει δηλωθεί προτού χρησιμοποιηθεί. Η δήλωση γίνεται με το αντίστοιχο πρωτότυπο της συνάρτησης (ορίζει τον τύπο της και τα ορίσματα που παίρνει).
- Ο ορισμός της συνάρτησης (κώδικας της συνάρτησης) γίνεται οπουδήποτε στον κώδικα του προγράμματος, στο ίδιο ή σε διαφορετικό αρχείο. Μπορεί να έχει ήδη γίνει compile, είτε σε αρχείο .o είτε σε βιβλιοθήκη και να ενσωματωθεί στον κώδικα κατά το linking.

Παράδειγμα με συνάρτηση

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

double aver(double a, double b);  //Πρωτότυπο συνάρτησης

int main()
{
    double z,x1,x2;
    cout << "Give x1:";
    cin >> x1;
    cout << "Give x2:";
    cin >> x2;
    z=aver(x1,x2);                //Κλήση συνάρτησης
    cout << "Average is: " << z;
    return 0;
}

double aver(double a, double b)   //Ορισμός συνάρτησης. Τα a, b είναι ήδη
{                                 // ορισμένα ως τοπικές μεταβλητές.
    return (a+b)/2.0;             //Επιστρέφεται η τιμή τύπου double.
}
```

Συναρτήσεις με όρισμα πίνακα

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

int max1(int a[10]);
int max2(int *a, int N);

int main()
{
     int z,x[10];
     for(int i=0;i<10;i++)
     {
          cout<<"Give x["<<i<<"]:";
          cin>>x[i];
     }
     z=max1(x);
     cout<<"Maximum is: "<<z;
     z=max2(x);
     cout<<"Maximum is: "<<z;
     return 0;
}
```

```
int max1(int a[10])
{
     int max=0;
     for(int i=0;i<10;i++)
        if(max<a[i]) max=a[i];
     return max;
}

int max2(int *a,int N)
{
     int max=0;
     for(int i=0;i<N;i++)
        if(max<a[i]) max=a[i];
     return max;
}
```

Παρατηρήσεις για τα ορίσματα συναρτήσεων

- Κατά την κλήση μιας συνάρτησης με όρισμα μια μεταβλητή, στη συνάρτηση περνάει μόνο η τιμή της μεταβλητής. Αυτή αποθηκεύεται σε τοπική μεταβλητή της συνάρτησης. Δεν μπορεί η συνάρτηση να αλλάξει την τιμή της μεταβλητής του κυρίως προγράμματος.

```
int func(int a, double b) //Τα a, b είναι τοπικές
{                         //μεταβλητές της συνάρτησης.
}
```

- Κατά την κλήση μιας συνάρτησης με όρισμα πίνακα, στη συνάρτηση η διεύθυνση (δείκτης) του πρώτου στοιχείου του πίνακα. Η συνάρτηση μπορεί να αλλάξει την τιμή των στοιχείων του πίνακα.

```
int func(int *a, int N) //Το a είναι pointer στον
                        //πίνακα με τον οποίο έγινε
                        //η κλήση της συνάρτησης.
```

Κλήση με pointers ως όρισμα

- Πολλές φορές είναι επιθυμητό να γίνει επιστροφή περισσότερων τιμών, ή να μπορεί η συνάρτηση να αλλάξει τις τιμές κάποιων ορισμάτων.
- Αυτό μπορεί να επιτευχθεί με χρήση pointers (ή αναφορών) στο όρισμα.

```
void swap(int *x, int *y)        //Τα ορίσματα είναι pointers στις
{                                //μεταβλητές
   int temp;
   temp=*x;
   *x=*y;
   *y=temp;
}

int main() {
…
swap(&x, &y);          //Καλούμε με τις διευθύνσεις των μεταβλητών
…                      //ως ορίσματα.
}
```

Κλήση με αναφορές ως όρισμα

```
int i;
int &j = i;
```

- Το `&j` είναι αναφορά στην `i`, δεν είναι η διεύθυνση του `j`. Τότε αν γράψουμε `j=10`, η τιμή του `i` θα γίνει 10.

```
void swap(int &x, int &y)        //Τα ορίσματα είναι αναφορές.
{
   int temp;
   temp=x;
   x=y;
   y=temp;
}

int main() { …
   swap(x, y);
… }
```

Pointer σε συνάρτηση ως όρισμα συνάρτησης

- Πολλές φορές χρειάζεται να περάσουμε ως παράμετρο σε συνάρτηση μία άλλη συνάρτηση, ώστε να χρησιμοποιηθεί από την πρώτη.
- Αυτό επιτυγχάνεται περνώντας pointer στη συνάρτηση.
- Αν υποθέσουμε ότι έχουμε ορίσει τη συνάρτηση func1 που δέχεται ως ορίσματα έναν πραγματικό και έναν ακέραιο και επιστρέφει πραγματικό, δηλαδή:

```
float func1(float x, int n)
{ … }
```

τότε μπορούμε να ορίσουμε μια συνάρτηση somefunc που να παίρνει ως όρισμα pointer στη func1:

```
void somefunc(int (*fnc)(float, int), int a)
{ …      fnc(y, m);  … }
```

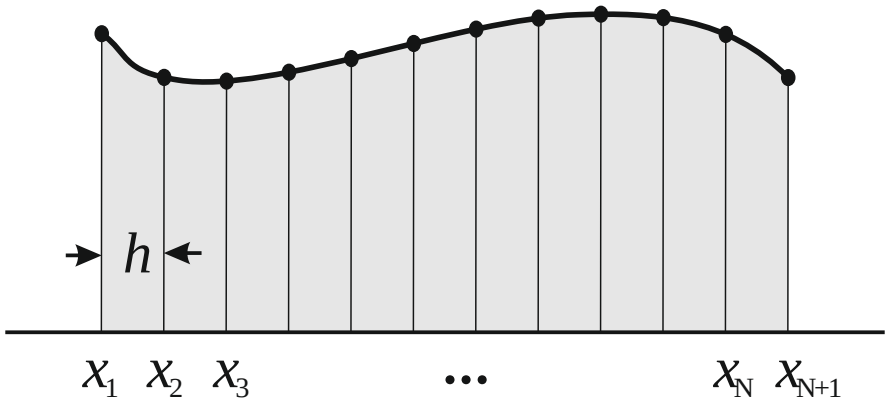
Παράδειγμα: ολοκλήρωση με μέθοδο τραπεζίου

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

double trapzd(double (*func)(double), double a, double b, int n);
double func1(double x);

int main()
{
   double a,b,I;
   int n=20;
   a=1.0;
   b=2.0;

   I=trapzd(func1,a,b,n);
   cout<<"Integral is: "<<I<<endl;
   return 0;

}

double func1(double x) {
   return x*x;
}
```

Παράδειγμα: ολοκλήρωση με μέθοδο τραπεζίου (2)

```
double trapzd(double (*func)(double), double a, double b, int n)
{
   double x,sum,del;
   int j;

   del=(b-a)/n;
   x=a;
   for (sum=0.0,j=1;j<=n;j++,x+=del)
   {
      sum += func(x);
      sum += func(x+del);
   }
   sum*=0.5*del;
   return sum;
}
```

$$\int_a^b f(x)dx \approx \sum_{i=1}^{N} \frac{1}{2} h[f(x_i) + f(x_{i+1})] \text{ ,όπου } h = x_{i+1} - x_i$$



Παράδειγμα: ταξινόμηση με bubblesort

```
#include <iostream>
#include <cstdlib>
#include <ctime>        //header file που ορίζει συναρτήσεις χρόνου
using namespace std;

void bubble(int* arr, int N);
void swap(int &x, int &y);

int main()
{
    int nums[10], t, size=10;
    srand(time(NULL)); //ορίζει το seed για τη συνάρτηση rand με βάση τον τρέχοντα
                       //χρόνο.
    for(t=0; t<size; t++) nums[t]=rand()%100+1; //παράγει τυχαίους αριθμούς (1-100)
    cout<<"Original array is: \n";
    for(t=0;t<size;t++) cout<<nums[t]<<'\n';
    bubble(nums,size);
    cout<<"\nSorted array: \n";
    for(t=0;t<size;t++) cout<<nums[t]<<'\n';
    return 0;
}
```

Παράδειγμα: ταξινόμηση με bubblesort (2)

```
void bubble(int* arr, int N) //υλοποιεί την bubblesort
{
    int a,b,t;
    for(a=1; a<N;a++)
        for(b=N-1; b>=a;b--)
            if(arr[b-1]>arr[b])
            {
                swap(arr[b-1],arr[b]);
            }
}

void swap(int &x, int &y)      //Συνάρτηση αντιμετάθεσης.
{
        int temp;
        temp=x;
        x=y;
        y=temp;
}
```

Ενδεικτική έξοδος προγράμματος bubblesort

```
Original array is:
44
68
49
43
17
18
7
75
7
38

Sorted array:
7
7
17
18
38
43
44
49
68
75

Process returned 0 (0x0)   execution time : 0.037 s
Press any key to continue.
```

Υπερφόρτωση συναρτήσεων

- Μπορούμε να ορίσουμε διαφορετικές συναρτήσεις με ίδιο όνομα αλλά διαφορετικό τύπο ή/και πλήθος ορισμάτων.
- Καλείται η κατάλληλη ανάλογα με τα ορίσματα που χρησιμοποιούμε κατά την κλήση.

```
int f(int i);
int f(int i, int j);
int f(double a);
…
f(10);      //Καλείται η 1η
f(1,3);     //Καλείται η 2η
f(5.3);     //Καλείται η 3η
```

Παράδειγμα υπερφόρτωσης συναρτήσεων

```
void swap(int &a, int &b)
{
   int temp;
   temp = a;
   a = b;
   b = temp;
}

void swap(double &a, double &b)
{
   double temp;
   temp = a;
   a = b;
   b = temp;
}

void swap(char &a, char &b)
{
   char temp;
   temp = a;
   a = b;
   b = temp;
}
```

```
int main()
{
   int ia=1,ib=2;
   double da=1.1, db=2.2;
   char ca='a', cb='b';
   …
   swap(ia,ib);
   swap(da,db);
   swap(ca,cb);
   …
   return 0;
}
```

Προκαθορισμένες τιμές ορισμάτων

- Είναι δυνατό να δώσουμε αρχικές τιμές στις παραμέτρους μιας συνάρτησης, ώστε να μην είναι απαραίτητο να δοθούν τιμές κατά την κλήση.
- Οι προαιρετικές παράμετροι θα πρέπει να είναι στο τέλος της λίστας παραμέτρων.

```
int fun(int a=1, int b=2, int c=3)
{ … }
```

Τρόποι κλήσης της συνάρτησης:

```
fun();          //a=1, b=2, c=3
fun(10);        //a=10, b=2, c=3
fun(10,20);     //a=10, b=20, c=3
fun(10,20,30); //a=10, b=20, c=30
```

Παρατήρηση:

- Η ταυτόχρονη υπερφόρτωση συνάρτησης με βάση το πλήθος των παραμέτρων και δήλωση αρχικών τιμών παραμέτρων, μπορεί να δημιουργήσει ασάφεια.

Αναδρομικές συναρτήσεις

- Αναδρομική λέγεται η συνάρτηση που καλεί τον εαυτό της.
- Χρήσιμη για υπολογισμό αναδρομικών τύπων.

```
int factr(int n)    //Αναδρομική συνάρτηση υπολογισμού παραγοντικού
{
   int ans;
   if(n==1) return(1);
   ans=factr(n-1)*n;
   return(ans);
}
```

- Κάθε κλήση της συνάρτησης δημιουργεί αντίγραφο στη μνήμη δεσμεύοντας νέα μνήμη.
- Πολλές αναδρομικές κλήσεις μπορεί να οδηγήσουν σε υπερχείλιση της μνήμης stack.
- Πολλές αναδρομικές κλήσεις καθυστερούν πολύ την εκτέλεση του προγράμματος.

Δυναμική δέσμευση μνήμης

- Με τη δυναμική δέσμευση μνήμης, γίνεται καλύτερη χρήση της διαθέσιμης μνήμης.
  - Δεσμεύεται μόνο όση μνήμη χρειάζεται σε κάθε στιγμή για την εκτέλεση του προγράμματος.
  - Απελευθερώνεται η μνήμη όταν πλέον δεν χρειάζεται.

C

```
double *dyn_x,*dyn_ar;
dyn_x = malloc(sizeof(double));
//Ένας αριθμός double

int n=10;
dyn_ar = malloc(n*sizeof(double));
//πίνακας 1D με n στοιχεία

free(dyn_x);  //αποδέσμευση μνήμης
free(dyn_ar);
```

C++

```
double *dyn_x,*dyn_ar;
dyn_x = new double;
//Ένας αριθμός double

int n=10;
dyn_ar = new double[n];
//πίνακας 1D με n στοιχεία

delete dyn_x; //αποδέσμευση μνήμης
delete[] dyn_ar;
```

Χρήση πολλών αρχείων κώδικα

file1.cpp

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

int a,b; //global μεταβλητές
int c;  //global για το file1
int func1(int n);
float func2(int x);

int main()
{
   int c1;      //local στη main
   for(…)
   {
      int f1=10;  //local στο block
      res=func1(f1);
      …
   }
   d=func2(a);
   …
   return 0;
}
```

file2.cpp

```
extern int a,b; //εξωτερικές global
int g;   //global για το file2

int func1(int n)
{ … }

float func2(int x)
{ … }
```

- Εντολή για compile:

```
g++ -o out.exe file1.cpp file2.cpp
```

Δομές - Structures

Η συλλογή πολλών μεταβλητών διαφορετικών τύπων σε μία ενιαία δομή.

```
struct όνομα_δομής
{
  τύπος όνομα;
  τύπος όνομα;
  …
}ονόματα_μεταβλητών;

όνομα_δομής όνομα_μεταβλητής;
```

Παράδειγμα

```
struct address //Δήλωση δομής.
{ //Δεν δεσμεύεται μνήμη
   char street[20];
   int number;
   char city[20];
   int code;
   char country[20];
};
```

```
int main()
{
   address work_address;
   address home_address;
   cout << sizeof(address);
   cin >> home_address.street;
   cin >> home_address.number;
   cin >> home_address.city;
   cin >> home_address.code;
   cin >> home_address.country;

   address work_address=
      {"Panepistimiou",1000,
         "Athens",12345,"Greece"};
   return 0;
}
```

Object Oriented Programming

Κλασσικός προγραμματισμός

- Έμφαση στη διαδικασία επίλυσης του προβλήματος.
  - Δεδομένα προβλήματος.
  - Αλληλουχία πράξεων και λογικών βημάτων για επεξεργασία των δεδομένων και την επίλυση του προβλήματος.

Αντικειμενοστραφής προγραμματισμός

- Έμφαση στα δεδομένα και τις ιδιότητές τους.
- Αναγνώριση οντοτήτων στο πρόβλημα και δημιουργία αντικειμένων.
- Αντικείμενα:
  - Ιδιότητες
  - Ενέργειες, συμπεριφορά

Η κλάση

- Στην ουσία αποτελεί (ορίζει) τον τύπο (το είδος) ενός αντικειμένου.
- Η κλάση έχει:
  - Δεδομένα – μεταβλητές μέλη (properties, fields, attributes)
  - Συμπεριφορά – συναρτήσεις μέλη (methods)

- Τα δεδομένα και οι συναρτήσεις της κλάσης μπορεί να είναι public (προσβάσιμα από το υπόλοιπο πρόγραμμα) ή private (προσβάσιμα μόνο εντός της κλάσης).
- Οι public συναρτήσεις και μεταβλητές αποτελούν το "interface" επικοινωνίας της κλάσης με το κυρίως πρόγραμμα.
- Συνήθως υπάρχει συνάρτηση για τη δημιουργία αντικειμένου (constructor) και την καταστροφή του (destructor). Αυτά χρησιμοποιούνται για τη δέσμευση/αποδέσμευση μνήμης και αρχικοποίηση των δεδομένων του αντικειμένου.

Ένα παράδειγμα κλάσης

```
class Box       //Ορισμός κλάσης για την περιγραφή ενός κουτιού
{
  private:      //Μπορούσε να παραλειφθεί
    double m_length;   //μεταβλητές για τις διαστάσεις
    double m_width;
    double m_height;

  public:
    int BoxDimensions(double length, double width, double height)
//Συνάρτηση ορισμού των διαστάσεων
    {
      if ((length >0.0) && (width >0.0) && (height >0.0)) {
         m_length = length;
         m_width = width;
         m_height = height;
      } else {
         return -1;
      }
    }
    double BoxVolume() //Συνάρτηση υπολογισμού και επιστροφής του όγκου
    { return m_length * m_width * m_height;  }
};
```

Ένα παράδειγμα κλάσης (2)

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

int main( )
{
   double V;
   int err;
   Box Box1; //δήλωση αντικειμένου τύπου box
   err=Box1.BoxDimensions(5.0 , 6.0 , 7.0);  //καταχώριση των διαστάσεων
   if(err<0)
   {
      cout<<"Error in dimensions!\n";
      exit(-1);
   }
   V=Box1.BoxVolume();          //Κλήση της συνάρτησης επιστροφής του όγκου
   cout << "Volume of Box1 : " << V << endl;
   return 0;
}
```

Απλή ουρά με κλάση

```
class queue
{
   int *q;
   int sloc, rloc, max;
 public:
   queue(int max_size); //constructor
   ~queue();            //destructor
   void qput(int i);
   int qget();
};

queue::queue(int max_size) //constructor
{
   sloc=rloc=0;
   q=new int[max_size];
   max=max_size;
   cout<<"Queue initialized\n";
}

queue::~queue() {      //destructor
    delete[] q;
    cout<<"Queue destroyed.\n";
}
```

```
void queue::qput(int i)
{
    if(sloc==max) {
        cout<<"Queue is full!\n";
        return;
    }
    sloc++;
    q[sloc]=i;
}

int queue::qget()
{
    if(rloc==sloc) {
        cout<<"Queue is empty!\n";
        return q[rloc];
    }
    rloc++;
    return q[rloc];
}
```

Constructors – Destructors

- Μία κλάση μπορεί να περιέχει πολλές υπερφορτωμένες συναρτήσεις constructors.
- Ο constructor χωρίς παραμέτρους λέγεται default constructor.
- Copy constructor:
  - Constructor που χρησιμοποιείται για τη δημιουργία ενός αντικειμένου με αντιγραφή ενός ήδη υπάρχοντος.
  - Δέχεται ως όρισμα αντικείμενο της ίδιας κλάσης.

```
class myclass {
…
        myclass(int i);
        myclass(char* s);
        myclass();      //default constructor
        myclass(const myclass &d);    //copy constructor
};
```

Στατικά μέλη κλάσης – Φιλικές συναρτήσεις

Static member

- Όταν μία μεταβλητή μέλος μιας κλάσης θέλουμε να έχει σταθερή τιμή για όλα τα αντικείμενα της κλάσης, αυτή δηλώνεται ως static.
- Η static μεταβλητή μοιράζεται μεταξύ των αντικειμένων της ίδιας κλάσης.

Friend function

- Οι συναρτήσεις που δεν είναι μέλη μιας κλάσης δεν έχουν πρόσβαση στα ιδιωτικά μέλη της κλάσης.
- Αν μία συνάρτηση δηλωθεί ως φιλική (friend function), τότε έχει κανονικά πρόσβαση σε αυτά.

Παράδειγμα με υπερφόρτωση constructors

```
class Box {
private:
   double m_length,m_width,m_height;
   static int BoxNumber;       //Πλήθος κουτιών - στατική μεταβλητή
   int BoxID;
public:
   Box() { m_length=m_width=m_height=1.0; BoxID=BoxNumber++; } //default constructor
   Box(double length,double width,double height){
      m_length = length;
      m_width = width;
      m_height = height;
      BoxID = BoxNumber++;
   }
   Box (const Box &BoxCopy) {  //copy constructor
      m_length = BoxCopy.m_length;
      m_width = BoxCopy.m_width;
      m_height = BoxCopy.m_height;
      BoxID = BoxNumber++;
   }
   void BoxDimensions(double length, double width, double height)
   { m_length = length; m_width = width; m_height = height;}
   double BoxVolume() { return m_length * m_width * m_height; }
   int GetBoxID() { return BoxID; }
};
```

Δείκτες σε αντικείμενα – Πίνακες αντικειμένων

- Αν έχουμε ορίσει μία κλάση, μπορούμε να ορίσουμε πίνακα αντικειμένων όπως θα κάναμε με οποιοδήποτε άλλο τύπο. Ομοίως για να ορίσουμε πίνακα αντικειμένων.

```
class Box       //Ορισμός κλάσης Box
{ …  };

int main()
{ …
   Box Boxarray1[3];            //Δήλωση πίνακα τύπου Box
   Box Box1, *Box2;              //Δήλωση pointer σε Box

   Box *Box_array = new Box[5]; //Δήλωση πίνακα και δέσμευση μνήμης
   Box2 = new Box;              //Δέσμευση μνήμης του Box2

   Box1.BoxDimensions(1,2,3);          //Κλήση της συνάρτησης BoxDimensions
   Box_array[1].BoxDimensions(4,5,6);
   Box2->BoxDimensions(10,20,30);      //->: τελεστής μέλους για δείκτες
…
   delete Box2;
   delete[] Box_array;
   return 0;
}
```

Ο δείκτης this

- Είναι ένας δείκτης που δημιουργεί αυτόματα ο compiler της C++.
- Δείχνει στη διεύθυνση ενός αντικειμένου, στιγμιότυπου μιας κλάσης.
- Ο δείκτης **this** προστίθεται στις παραμέτρους μίας συνάρτησης μέλους μιας κλάσης και δείχνει στο αντικείμενο για το οποίο έγινε η κλήση της συνάρτησης.
- Ο δείκτης **this** δεν είναι μέλος του αντικειμένου και δεν συμπεριλαμβάνεται στη μνήμη που δεσμεύει το αντικείμενο.
- Δεν ισχύει για στατικές συναρτήσεις.
- Συνηθέστερες χρήσεις:
  - Η επιστροφή από μία συνάρτηση μέλος ενός pointer στο αντικείμενο.
  - Η χρήση ονομάτων σε τοπικές μεταβλητές μιας συνάρτησης-μέλους που είναι ίδια με τις μεταβλητές-μέλη του αντικειμένου. Τότε η χρήση του this->var αναφέρεται στη μεταβλητή μέλος, ενώ η var στην τοπική μεταβλητή της συνάρτησης.

Υπερφόρτωση τελεστών

- Η C++ παρέχει τη δυνατότητα να γίνει υπερφόρτωση των τελεστών όπως ακριβώς και με τις συναρτήσεις. Αυτό είναι ιδιαίτερα χρήσιμο, καθώς μπορούν να οριστούν και να χρησιμοποιηθούν τελεστές όπως + - * / = ++ ==, κλπ. για τα αντικείμενα μιας κλάσης.
- Η βασική σύνταξη είναι:

```
τύπος επιστροφής operator#(παράμετροι) { … }
```

όπου # το σύμβολο του τελεστή.

- Περιορισμοί:
  - Τουλάχιστον ένας τελεστέος πρέπει να έχει οριστεί από το πρόγραμμα.
  - Δεν μπορεί να αλλάξει ο τρόπος σύνταξης του τελεστή.
  - Δεν αλλάζει η προτεραιότητα του τελεστή.
  - Δεν μπορούν να δημιουργηθούν νέοι τελεστές.

Υπερφόρτωση τελεστών (2)

```
#include <iostream>
using namespace std;
class dVector {
public:
   double x,y;
   dVector () {};
   dVector (double,double);
   dVector operator+ (dVector);
   double operator* (dVector);
};

dVector::dVector (double a, double b)
{  x = a;  y = b;  }

dVector dVector::operator+ (dVector par)
   //πρόσθεση διανυσμάτων
{
      dVector temp;
      temp.x = x + par.x;
      temp.y = y + par.y;
      return (temp);
}
```

```
double dVector::operator* (dVector par)
{  return (x * par.x + y * par.y); }
//εσωτερικό γινόμενο


int main()
{
   dVector a(3.2,1.1), b(2.3,5.5), c;
   double d;
   c=a+b;
   cout<<c.x<<","<<c.y<<endl;
   d=a*b;
   cout<<d<<endl;
   return 0;
}
```

Ο τελεστής ανάθεσης =

- Αν **a** και **b** δύο αντικείμενα μιας κλάσης και χρησιμοποιηθεί ο τελεστής ανάθεσης = (**b=a**), τότε οι μεταβλητές μέλη του **b** λαμβάνουν τις τιμές αυτών του **a**.
- Αν όμως το αντικείμενο **a** περιέχει δείκτες ή χρησιμοποιεί δυναμική μνήμη, τότε μετά την ανάθεση **b=a**, οι δείκτες του **b** δείχνουν στην ίδια θέση μνήμης με αυτούς του **a**.
- Όταν τα αντικείμενα **a**, **b** βγαίνουν εκτός εμβέλειας (πχ. στον τερματισμό του προγράμματος), καταστρέφονται και τα δύο αντικείμενα και απελευθερώνεται η μνήμη τους. Τότε όμως, η κοινή τους μνήμη θα πρέπει να απελευθερωθεί 2 φορές. Αυτό οδηγεί σε μήνυμα λάθους κατά την καταστροφή του 2^{ου} αντικειμένου.
- Για να λυθεί το πρόβλημα, απαιτείται η δημιουργία ένας constructor αντιγραφής που να εξασφαλίζει ότι τα αντικείμενα χρησιμοποιούν διαφορετικό χώρο μνήμης, καθώς και η υπερφόρτωση του τελεστή ανάθεσης.

Ο τελεστής ανάθεσης = (2)

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

class T
{
private:
     char *s;
public:
     T(const char str[]);
     ~T();
     void show() const {cout<<s<<endl;}
};

T::T(const char str[])
{
     s=new char[strlen(str)+1];
     strcpy(s,str);
}
T::~T() {
delete[] s;
}
```

```
int main()
{
     T t1("Peter"), t2("Mike");

     t2=t1;  //Δημιουργεί σφάλμα.
     t2.show();
     return 0;
}
```

Παρατηρήσεις:

- Με την αντιγραφή, ο δείκτης t2.s δείχνει στη ίδια μνήμη με τον t1.s.
- Με τον τερματισμό του προγράμματος, καταστρέφονται ο t1.s και ο t2.s, δηλαδή επιχειρείται η απελευθέρωση της ίδιας μνήμης δύο φορές!

Υπερφόρτωση του τελεστή ανάθεσης =

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

class T
{
private:
    char *s;
public:
    T(const char str[]);
    ~T();
    void show() const {cout<<s<<endl;}
    T& operator=(const T &t);
};

T::T(const char str[])
{
    s=new char[strlen(str)+1];
    strcpy(s,str);
}
T::~T() {
delete[] s;
}

T& T::operator=(const T &t)
{
    delete[] s;
    s=new char[strlen(t.s)+1];
    strcpy(s,t.s);
    return *this;
}

int main()
{
    T t1("Peter"), t2("Mike");

    t2=t1;
    t2.show();
    return 0;
}
```

Ο νέος τελεστής ανάθεσης εξασφαλίζει ότι το νέο αντικείμενο θα έχει δικό του χώρο μνήμης.

Υπερφόρτωση τελεστών << και >>

- Οι τελεστές << και >> που χρησιμοποιούνται για είσοδο και έξοδο, έχουν υπερφορτωθεί στις κλάσεις ostream και istream που ορίζονται στη βιβλιοθήκη <iostream>. Μπορούμε να τους υπερφορτώσουμε σε μία δική μας κλάση για να εκτελείται η είσοδος/έξοδος της κλάσης μας.

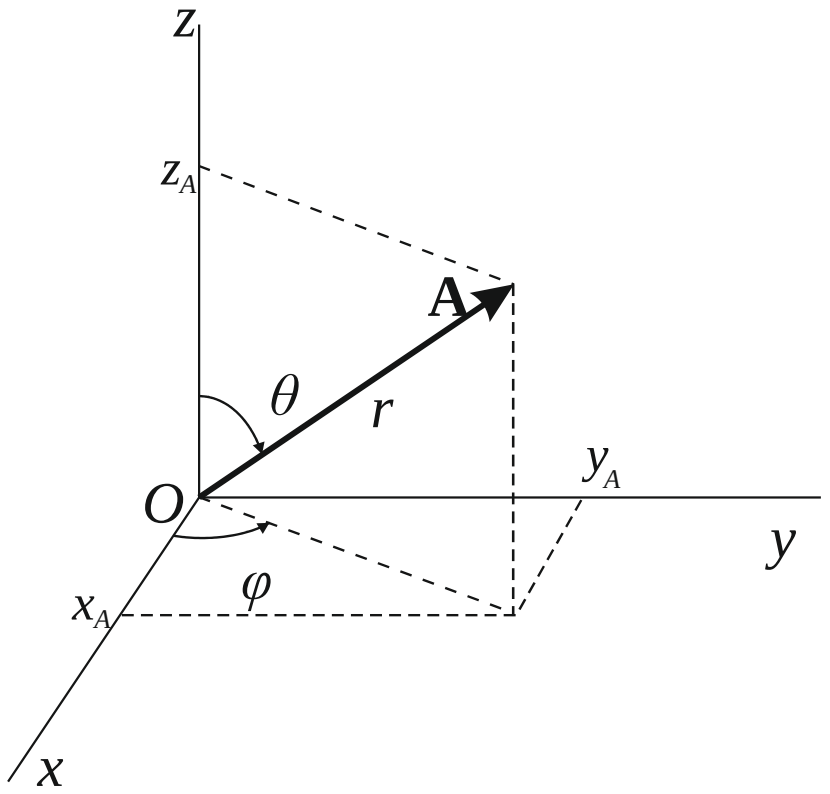
```
class T
{
 private:
    char *s;
 public:
    T(const char str[]);
    ~T();
    T& operator=(const T &t);
    void show() const {cout<<s<<endl;}
    friend ostream& operator<<(ostream& out, const T& t);
};
ostream& operator<<(ostream& out, const T &t)
{ out<<t.s; }
```

- Τώρα μπορούμε να γράψουμε `cout << t2 << endl` στο πρόγραμμα της προηγούμενης διαφάνειας.

Παράδειγμα με διανυσματικά μεγέθη (3D)

- Ένα διανυσματικό μέγεθος περιγράφεται από ένα διάνυσμα στο χώρο $\mathbf{A}(x,y,z)$ σε καρτεσιανό, σφαιρικό ή άλλο σύστημα συντεταγμένων.
- Εσωτερικό γινόμενο: $\mathbf{A}\cdot\mathbf{B}=(x_A x_B + y_A y_B + z_A z_B)$
- Εξωτερικό γινόμενο: $\mathbf{A}\times\mathbf{B} = \begin{vmatrix} \hat{x} & \hat{y} & \hat{z} \\ x_A & y_A & z_A \\ x_B & y_B & z_B \end{vmatrix}$

Καρτεσιανές συντεταγμένες	**Σφαιρικές συντεταγμένες**
x y z	r [0,∞) φ [0,2π) θ [0,π]
$x = r\cos\varphi\sin\theta$ $y = r\sin\varphi\sin\theta$ $z = r\cos\theta$	$r = \sqrt{x^2+y^2+z^2}$ $\varphi = \tan^{-1}\left(\frac{y}{x}\right)$ $\theta = \tan^{-1}\left(\frac{\sqrt{x^2+y^2}}{z}\right)$



Παράδειγμα με διανυσματικά μεγέθη (3D) (2)

```
#ifndef __VECTOR3D
#define __VECTOR3D
#define _SIZEMAX 256
#define TINY 1.e-200
#include <iostream>
#include <cmath>
#include <cstring>
using namespace std;

class vector3d {
 private:
   char m_loc[_SIZEMAX];
   double m_x; //x coordinate
   double m_y; //y coordinate
   double m_z; //z coordinate
 public:
   vector3d() { m_x=0; m_y=0; m_z=0; }
   vector3d(double xt, double yt, double
     zt) { m_x=xt; m_y=yt; m_z=zt; }

   double magnitude() {return
        sqrt(m_x*m_x+m_y*m_y+m_z*m_z);}
   double azimuth();
   double zenith();
   double x() {return m_x;}
   double y() {return m_y;}
   double z() {return m_z;}
   void spherical(double rt,double
        azim,double zen);
   void name(char *str);
        string get_name();

   vector3d operator+(vector3d op2);
   vector3d operator-(vector3d op2);
   vector3d operator=(vector3d op2);
   double operator*(vector3d op2);
   vector3d operator*(double op2);

   friend ostream &operator<<(ostream
       &stream, vector3d op2);
   friend istream &operator>>(istream
       &stream, vector3d &op2);
};
```

Παράδειγμα με διανυσματικά μεγέθη 3D (3)

```
void vector3d::spherical(double rt,
          double azim, double zen)
//Function to define the spherical
   coordinates of the vector
{
   if((rt>=0) && (azim>=0) &&
      (azim<2.0*M_PI) && (zen>=0) &&
      (zen<=M_PI))
   {
      m_x=rt*cos(azim)*sin(zen);
      m_y=rt*sin(azim)*sin(zen);
      m_z=rt*sin(zen);
   } else {
      m_x=m_y=m_z=0.0;
      cout<<"Illegal coordinates!\n";
   }
}

double vector3d::azimuth() {
   if(m_x==0.0) m_x=TINY;
   double temp = atan2(m_y,m_x);
   if(temp<0) temp += 2.0*M_PI;
   return temp;
}
```

```
double vector3d::zenith()
{
   if(m_z==0) m_z=TINY;
   return atan2(sqrt(m_x*m_x+
         m_y*m_y),m_z);
}

vector3d vector3d::operator=(vector3d
op2) {
    m_x=op2.m_x;
    m_y=op2.m_y;
    m_z=op2.m_z;
    return *this;
}

vector3d vector3d::operator+(vector3d
op2) {
    vector3d temp;
    temp.m_x=m_x+op2.m_x;
    temp.m_y=m_y+op2.m_y;
    temp.m_z=m_z+op2.m_z;
    return temp;
}
```

Παράδειγμα με διανυσματικά μεγέθη 3D (4)

```
vector3d vector3d::operator-(vector3d
op2) //subtraction between 2 vectors
{
    vector3d temp;
    temp.m_x=m_x-op2.m_x;
    temp.m_y=m_y-op2.m_y;
    temp.m_z=m_z-op2.m_z;
    return temp;
}

double vector3d::operator*(vector3d op2)
        //dot product
{
    double temp;
    temp=m_x*op2.m_x;
    temp+=m_y*op2.m_y;
    temp+=m_z*op2.m_z;
    return temp;
}

vector3d vector3d::operator*(double op2)
{       //scalar product
    vector3d temp;
    temp.m_x=m_x*op2;
    temp.m_y=m_y*op2;
    temp.m_z=m_z*op2;
    return temp;
}

ostream &operator<<(ostream &stream,
vector3d op2) {//output serialization
{
    stream <<"("<<op2.m_x <<","<<op2.m_y
       <<","<<op2.m_z<<")";
    return stream;
}

istream &operator>>(istream &stream,
vector3d &op2) //input serialization
{
    double tempx, tempy, tempz;
    stream >> tempx >> tempy >> tempz;
    op2=vector3d(tempx,tempy,tempz);
    return stream;
}
```

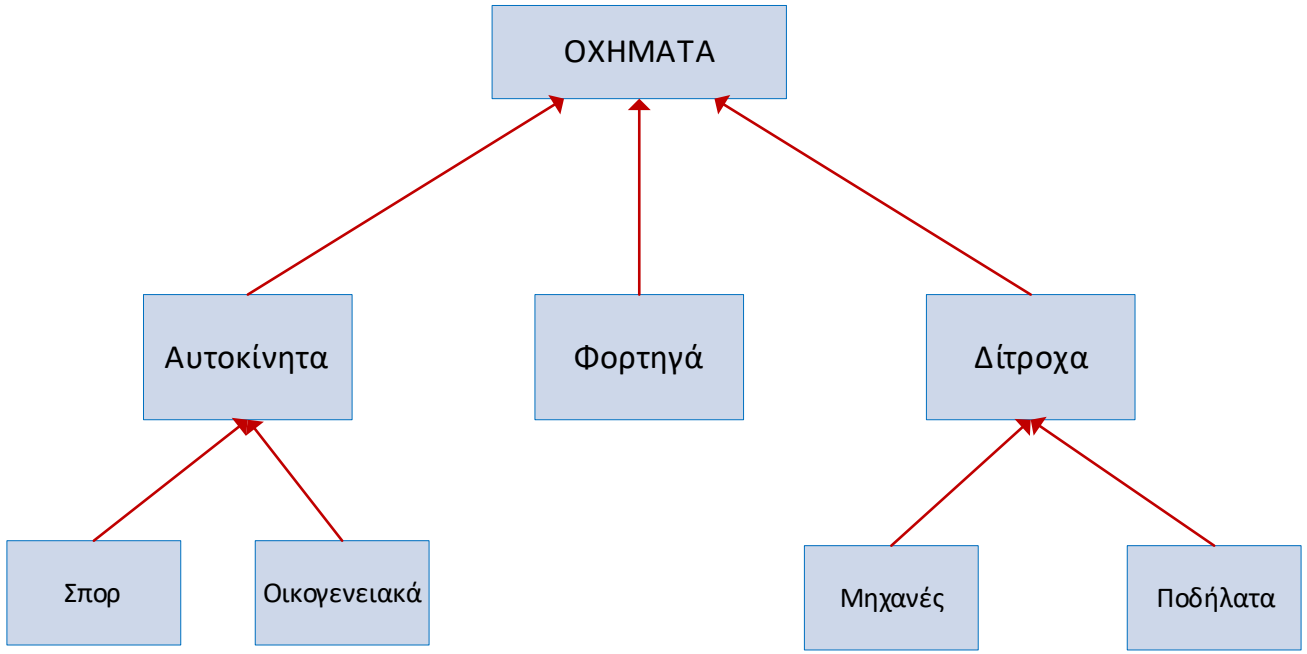
Παράδειγμα με διανυσματικά μεγέθη 3D (5)

```
void vector3d::name(char *str)
{ strcpy(m_loc,str);}

string vector3d::get_name() {
   char temp[_SIZEMAX] ;
   strcpy(temp,m_loc);
   return temp;
}

vector3d Xproduct(vector3d a, vector3d b)
{  //External product
   double x,y,z;

   x=a.y()*b.z()-b.y()*a.z();
   y=a.z()*b.x()-a.x()*b.z();
   z=a.x()*b.y()-a.y()*b.x();

   return vector3d(x,y,z);
}

#endif
```

```
int main()
{
   vector3d  a=vector3d(1.0, 2.0, 3.0),
             b=vector3d(4.0, 5.0, 6.0);

   cout<<"|a|= "<<a.magnitude()<<endl;
   cout<<"azimuth(a)= " << a.azimuth()
       <<endl;
   cout<<"zenith(a)= " << a.zenith()
       <<endl;
   cout<<"|a|= " << a.magnitude() <<endl;
   cout<<"azimuth(a) (deg)= " <<
       a.azimuth()*180.0/M_PI <<endl;
   cout<<"zenith(a) (deg)= " <<
       a.zenith()*180.0/M_PI <<endl;
   cout<<"a+b= " << a+b <<endl;
   cout<<"a-b= " << a-b <<endl;
   cout<<"a.b= " << a*b <<endl;
   cout<<"(a_x_b)= " << Xproduct(a,b)
       <<endl;
return 0;
}
```

Κληρονομικότητα

- Η C++ παρέχει τη δυνατότητα να οριστεί μία κλάση (παραγόμενη κλάση) με τρόπο ώστε να εμπεριέχει μια ήδη υπάρχουσα κλάση (κλάση βάσης).
- Η παραγόμενη κλάση έχει πρόσβαση στα μέλη της κλάσης βάσης σαν να είχαν οριστεί εντός της παραγόμενης.
- Χρησιμοποιείται για να οριστούν κλάσεις με ιεραρχική δομή από τη γενικότερη προς την ειδικότερη.



Templates – Πρότυπες συναρτήσεις και κλάσεις

- Στη C++ μπορούμε να ορίσουμε πρότυπες συναρτήσεις ή κλάσεις με γενικό τρόπο ώστε να δέχονται διαφορετικούς τύπους δεδομένων (πχ. int, float, double).
- Με τον τρόπο αυτό αποφεύγεται η ανάγκη για πολλαπλό ορισμό (και υπερφόρτωση) συναρτήσεων ή κλάσεων για κάθε τύπο δεδομένων.
- Δήλωση πρότυπης συνάρτησης:
  - `template <typename T> T func(T a)`
- Δήλωση πρότυπης κλάσης:
  - `template <classtype T> class classname { … }`
- Παράδειγμα η πρότυπη κλάση vector
  - `template <class T, class Alloc = allocator<T>> class vector;`

Αρχεία

- Για την είσοδο και έξοδο στη C++ έχουν οριστεί κλάσεις ρευμάτων (streams) στη βιβλιοθήκη <iostream>. Ένα ρεύμα εισόδου αποτελεί το αντικείμενο cin και ένα ρεύμα εξόδου το cout.
- Για τη διαχείριση αρχείων ορίζονται αντίστοιχες κλάσεις στη βιβλιοθήκη <fstream>.
  - `ifstream in;` : ορισμός αντικειμένου αρχείου για ανάγνωση
  - `ofstream out;` : ορισμός αντικειμένου αρχείου για εγγραφή
  - `fstream both;` : ορισμός αντικειμένου αρχείου για ανάγνωση και εγγραφή
- Άνοιγμα αρχείου με χρήση της συνάρτησης open της κλάσης:
  - `in.open("filename", mode);`

  ή κατά τη δήλωση του αντικειμένου με χρήση του constructor:
  - `ifstream in("filename");`
- Κλείσιμο αρχείου:
  - `in.close();`

Προσπέλαση αρχείων κειμένου

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

int main()
{
   char str[80], fname[]="test.txt";
   int i;
   float f;
   ofstream out(fname);
   if(!out) {
      cout<<"Error opening file
      for writing.\n";
      return -1;
   }
   out <<"This is a test txt file.\n";
   out << 10 << " " << 123.33 << endl;
   out.close();

   ifstream in(fname);
   if(!out) {
      cout<<"Error opening file
         for reading.\n";
      return -2;
   }

   in.getline(str,sizeof(str));
   in>>i;
   in>>f;
   cout<<str<<"\n";
   cout<<i<<"\n"<<f<<"\n";
   return 0;
}
```

test.txt:

```
This is a test txt file.
10 123.33
```

Output:

```
This is a test txt file.
10
123.33
```

Παράμετροι ανοίγματος αρχείων

- `fp.open("filename", mode, access);`

  - ***mode***: τρόπος ανοίγματος αρχείου
    - `ios::in` : άνοιγμα για είσοδο (ανάγνωση)
    - `ios::out` : άνοιγμα για έξοδο (εγγραφή)
    - `ios::app` : append – προσθήκη στο τέλος (για αρχεία εξόδου)
    - `ios::binary` : άνοιγμα αρχείου σε δυαδική μορφή
    - `ios::trunc` : άδειασμα υπάρχοντος αρχείου και εγγραφή από την αρχή
    - `ios::nocreate` : Άνοιγμα αρχείου μόνο αν υπάρχει
    - `ios::noreplace` : Άνοιγμα αρχείου μόνο αν δεν υπάρχει

Χρήσιμες συναρτήσεις προσπέλασης αρχείων

- get(*char ch*) : ανάγνωση ενός χαρακτήρα
- put(*char ch*) : εγγραφή ενός χαρακτήρα
- read(*unsigned char* ch, streamsize N*) : Ανάγνωση N bytes
- write(*const unsigned char* ch, streamsize N*) : Εγγραφή N bytes
- getline(*char* s, streamsize N*) : Ανάγνωση ολόκληρης γραμμής με μέγιστο πλήθος χαρακτήρων N
- flush() : Άδεισμα της μνήμης buffer (εγγραφή του περιεχομένου της).
- eof() : Έλεγχος αν ο δείκτης προσπέλασης έχει φτάσει στο τέλος του αρχείου
- setf(*fmtflags fmtfl*) : Καθορισμός μορφοποίησης εξόδου
- precision(*prec*) : Καθορισμός ακρίβειας εγγραφής αριθμού
- width(w) : Καθορισμός εύρους αριθμού σε χαρακτήρες