

ΦΥΣ 347

Υπολογιστικές Μέθοδοι στη Φυσική //

Φθινόπωρο 2018

Διδάσκων: Φώτης Πτωχός

e-mail: fotis@ucy.ac.cy

Τηλ: 22.89.2837

Γραφείο: B235 – ΘΕΕ02

Γενικές Πληροφορίες

□ Ώρες/Αίθουσα διδασκαλίας:

- Δευτέρα 14:00 – 18:00
- Εργαστήριο Η/Υ: 012

Διαλέξεις/Εργαστήρια

 **Εργαστήρια** 

ΥΠΟΧΡΕΩΤΙΚΑ



Το πολύ μια απουσία

Απορίες/Διαλέξεις/Εργαστήρια

- ❑ Διακόπτεται για απορίες κατά την διάρκεια της διάλεξης
- ❑ Προγράμματα θα γράφονται παράλληλα με την διάλεξη ώστε να υπάρχει αμεσότητα με τις εντολές που χρησιμοποιούνται
- **Ώρες γραφείου:** Πέμπτη είναι η καλύτερη μέρα
Φυσικά μπορείτε να έρθετε οποιαδήποτε άλλη μέρα και ώρα....
κυρίως αργά το απόγευμα...

Βιβλιογραφία

Δεν θα ακολουθήσω κάποιο συγκεκριμένο βιβλίο.

Οι διαλέξεις/σημειώσεις θα είναι στο web.

Οι ασκήσεις/οδηγίες των εργαστηρίων όπως και οι εργασίες που θα δουλεύετε σπίτι θα τις βρίσκετε επίσης στο web.

(<http://www2.ucy.ac.cy/~fotis/phy347.html>)

Χρήσιμη βιβλιογραφία/παραδείγματα μπορούν να βρεθούν στο web.

Προσπαθήστε και μόνοι σας τις πρώτες δύο βδομάδες να ασχοληθείτε και να εξοικειωθείτε με

Λειτουργικό σύστημα: **Linux**

Γλώσσα προγραμματισμού: **C++**

Λογισμικό Γραφικών αναπαραστάσεων: **ROOT**

Μπορείτε να χρησιμοποιήσετε τα προσωπικά σας laptops αλλά θα πρέπει αν έχετε Windows να εγκαταστήσετε **Virtual Machine** και **Ubuntu**

Βιβλιογραφία

- ❑ Για C++ θα μπορούσατε να χρησιμοποιήσετε πάμπολες πληροφορίες που υπάρχουν στο διαδίκτυο
- ❑ Για Linux θα μπορούσατε να χρησιμοποιήσετε τα ακόλουθα:
 - [Linux in a nutshell](#) από Ellen Siever, Stephen Spainhour, Stephen Figgins and Jessica P. Hekman , ed. O'Reilly
 - [Running Linux](#) από Matt Welsh, Matthias Kalle Dalheimer, and Lar Kaufman, ed. O'Reilly
- ❑ Για το λογισμικό ROOT:
 - Από την επίσημη ιστοσελίδα του ROOT (<http://root.cern.ch>)

Βιβλιογραφία

□ Για το κύριο μέρος του μαθήματος: Υπολογιστική φυσική

- **Numerical Recipes** από William H. Press, Saul A. Teukolsky, William T. Vetterling and Brian P. Flannery. Μπορείτε να το βρείτε επίσης **δωρεάν** με κώδικα σε C.
- **Numerical Methods for Physics** (2nd edition) από A. Garcia (εκδ. Prentice Hall)
- **A First Course in Computational Physics** από P. DeVries εκδ. Wiley & Sons
- **An Introduction to Computer Simulation Methods** H. Gould and J. Tobochnik, Addison-Wesley, 2nd edition
- **Computational Physics**, από N. Giordano, Prentice Hall
- Την ιστοσελίδα του μαθήματος ΦΥΣ 145 (Υπολογιστικές μέθοδοι στην Φυσική Ι – www2.ucy.ac.cy/~phy145/phy145.htm

Βαθμολογία

➤ Η βαθμολογία θα βασιστεί στα ακόλουθα:

- ✓ **10%** ασκήσεις εργαστηρίων
- ✓ **20%** ασκήσεις κατ' οίκον
- ✓ **10%** ένα project
- ✓ **25%** ενδιάμεση εξέταση στο 7ο εργαστήριο (Σάββατο 20 Οκτώβρη)
- ✓ **35%** τελική εξέταση



project

- ❑ Κάποιο πρόβλημα που θα χρειαστεί περισσότερη σκέψη και θα χρησιμοποιήσει αυτά που έχετε μάθει.
- ❑ Θα είναι ομαδικά και ο καθένας θα πρέπει να είναι σε θέση να εξηγήσει και να παρουσιάσει το τι έκανε η ομάδα του
- ❑ Το project θα ανακοινωθεί τέλος Οκτώβρη και θα έχετε ένα μήνα για να το προετοιμάσετε

Περιεχόμενο Μαθήματος

- Προγραμματισμός σε C++
- Προγραμματισμός/Χρήση λογισμικού ROOT
- Λύση συνήθων διαφορικών εξισώσεων και εφαρμογές
- Λύση μερικών διαφορικών εξισώσεων
- Γραμμικά συστήματα – Ιδιοτιμές/Ιδιοσυναρτήσεις
- Αριθμητική Ολοκλήρωση
- Τυχαίοι Αριθμοί
- Στοχαστικές Μέθοδοι
- Προσομοίωση
- Αλυσίδες Markov
- Αλγόριθμος Metropolis
- Μοντέλο Ising

Το πρώτο πρόγραμμα σε C++

Χρησιμοποιείτε τον επεξεργαστή κειμένου (emacs) για να γράψετε το ακόλουθο πρόγραμμα σε ένα αρχείο το οποίο να ονομάσετε **hello.cc**

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

int main()
{
    cout << "Hello world. \n" ;
}
```

Προσέξτε ώστε να μη σας ξεφύγουν τα διάφορα σημεία στήξης.

Μεταφράστε το πρόγραμμά σας σε γλώσσα μηχανής (**compilation**) με την εντολή:

```
g++ -o hello.x hello.cc
```

Αν παραλήψετε το **-o hello.x** τότε αυτόματα θα γραφεί το file **a.out**

Τρέξτε το πρόγραμμά σας δίνοντας την εντολή: `./hello.x` προσοχή στην τελεία .

Επεξήγηση του προγράμματος `hello.cc`

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

iostream είναι το όνομα ενός αρχείου το οποίο ονομάζεται *header file* ή *include file* και προσφέρεται από τον μεταφραστή της γλώσσας. Περιέχει κώδικα για διάφορες συναρτήσεις τις οποίες μπορούμε να χρησιμοποιήσουμε για είσοδο/έξοδο δεδομένων

#include επιτρέπει την εισαγωγή του κώδικα του `iostream` αρχείου στο δικό μας κώδικα και στο συγκεκριμένο σημείο

➤ Τέτοιου είδους γραμμές εμφανίζονται πάντα στην αρχή ενός προγράμματος

```
int main()
{
    cout << "Hello world. \n" ;
}
```

Όλα τα προγράμματα πρέπει να περιέχουν το τμήμα ενός κώδικα το οποίο ονομάζεται **main** και είναι εκεί που ξεκινά η εκτέλεση του προγράμματος.

Η δήλωση αυτή (`int main()`) προσδιορίζει ότι όλες οι γραμμές του κώδικα που περιέχονται ανάμεσα στις { ... } καθορίζουν το σύνολο του προγράμματος.

➤ Ο compiler θεωρεί το κύριο πρόγραμμα σαν υποπρόγραμμα συνάρτησης.

□ Για συναρτήσεις και τους τύπους συναρτήσεων θα μιλήσουμε αργότερα

Επεξήγηση του προγράμματος `hello.cc`

```
cout << "Hello world. \n" ;
```

Η εντολή αυτή προκαλεί την εκτύπωση της γραμματοσειράς "Hello world" να εκτυπωθεί στην οθόνη του υπολογιστή,

Παρατηρήστε ότι όλες οι απλές εντολές πρέπει να τερματίζονται με ";"

Ο ειδικός χαρακτήρας `\n` προκαλεί το τέλος της γραμμής στην οποία γράφεται κάτι στην οθόνη και δεν εμφανίζεται σαν πραγματικός χαρακτήρας.

Οι διπλές " " δηλώνουν την αρχή και το τέλος μια γραμματοσειράς.

Το όνομα ***cout*** δηλώνει output στην οθόνη (περιέχεται στο header file *iostream*)

Ο τελεστής `<<` σε συνεργασία με το `cout` προκαλεί την εκτύπωση στην οθόνη ότι βρίσκεται στα δεξιά του ("Hello world.") στην προκειμένη περίπτωση

Σημείωση: αν δεν είχε χρησιμοποιηθεί η δήλωση *using namespace std*; Τότε μπροστά από το `cout` θα έπρεπε να χρησιμοποιηθεί το πρόθεμα ***std::***

```
std::cout << "Hello world. \n" ;
```

Το δεύτερο πρόγραμμά μας σε C++: **whoRyou.cc**

Θέλουμε να γράψουμε ένα πρόγραμμα το οποίο:

- 1) να μας ρωτά το όνομά μας και την ηλικία μας
- 2) να διαβάσει τα δεδομένα από το πληκτρολόγιο
- 3) να αναγράφει στην οθόνη τα δεδομένα αυτά

Το πρόγραμμά μας θα μοιάζει ως εξής:

```
#include <iostream>
using namespace std;
int main()
{
    int age;
    char name[10];
    cout << "Enter the name and age: \n" ;
    cin >> name >> age;
    cout << "Your name is" << name
         << " and age is" << age << "\n";
}
```

Η στοίχιση της κάθε γραμμής δεν είναι απαραίτητη αλλά είναι καλή πρακτική να ακολουθείται γιατί το πρόγραμμα είναι πιο ευανάγνωστο και μπορεί εύκολα να ξεχωρίσουν τα διάφορα blocks εντολών

Ο emacs ακολουθεί τη πρακτική αυτή με απλό tabbing

Προσέξτε ότι μέχρι να βρεθεί ";" ο compiler θεωρεί συνέχιση εντολής ανεξάρτητα από το πλήθος των γραμμών

Κάντε compile το πρόγραμμά σας: `g++ -o whoRyou.x whoRyou.cc`

Αν το compilation αποτύχει τότε προσπαθήστε να δείτε τι γράφει τα μηνύματα των λαθών και να τα διορθώσετε

Εκτέλεση προγράμματος whoRyou.x

Τρέξτε το πρόγραμμα με την εντολή: `./whoRyou.x`

Θα πρέπει να σας ζητηθεί το όνομα και η ηλικία

Δώστε το ονοματεπώνυμο σαν μια λέξη ακολουθούμενη από την ηλικία σε μορφή ακεραίου και πιάστε *Enter*

Δοκιμάστε διαφορετικές μορφές εισόδου δεδομένων:

Ονοματεπώνυμο σε μία γραμμή και ηλικία σε δεύτερη γραμμή
ή δοκιμάστε διπλό *Enter* μεταξύ των δεδομένων

Δεδομένα εισαγωγής ερμηνεύονται από την κλάση ***cin*** είναι «λέξεις».

Οι λέξεις χωρίζονται από κενά (white spaces, tabs, end-of-line)

Σαν αποτέλεσμα αν εισάξετε το ονοματεπώνυμό σας σαν όνομα «κενό» επίθετο τότε το όνομα θα ληφθεί σαν το ονοματεπώνυμο και το επίθετο θα ληφθεί σαν την ηλικία.

Επεξήγηση του προγράμματος whoRyou.cc

```
int age;  
char name[10];
```

Οι εντολές αυτές είναι οι δηλώσεις των μεταβλητών *age* και *name*

Όλες οι μεταβλητές στη C++ θα πρέπει υποχρεωτικά να δηλωθούν πριν χρησιμοποιηθούν ώστε να κρατηθεί ο απαραίτητος χώρος μνήμης.

Η δήλωσή τους μπορεί να γίνει οπουδήποτε στο κώδικα, ακόμα και ακριβώς πριν τη χρησιμοποίησή τους.

Ιδιαίτερα προσεκτικοί σε αυτό το σημείο. Ανάλογα με το που δηλώνονται, οι μεταβλητές μπορεί να είναι “τοπικές” ή “γενικές”. Δηλαδή μπορεί να υπάρχουν σε ένα τμήμα του προγράμματος και να μην υπάρχουν/δηλωθεί σε άλλο τμήμα του προγράμματος.

Οι συνηθισμένοι τύποι μεταβλητών και ο αριθμός των αντίστοιχων bits είναι:

Τύπος	Περιγραφή
int	Integer (32 bits)
float	Real (32 bits)
double	Floating point (64 bits)
char	ένας χαρακτήρας (8 bits=1 byte)
bool	Boolean (8 bits = 1 byte)

Μεταβλητές ενός C++ προγράμματος

- ✓ Τα ονόματα μεταβλητών θα πρέπει να ξεκινούν από **γράμμα** ή **_** και θα πρέπει να περιέχουν **γράμματα, νούμερα ή _**
- Τα ονόματα μεταβλητών μπορεί να περιέχουν μικρά ή κεφαλαία γράμματα
Προσοχή: η μεταβλητή ***Isreal*** είναι διαφορετική από την ***isreal***
- ✓ Κάποια ονόματα δεν μπορούν να χρησιμοποιηθούν σαν μεταβλητές γιατί αποτελούν στοιχεία της γλώσσας όπως `int`, `char`
- ✓ Μετά την δήλωση του είδους της, η αριθμητική μεταβλητή μπορεί να αποκτήσει τιμή ή να εκτιμηθεί το αποτέλεσμα που ανατίθεται στην μεταβλητή

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

int main()
{
    int a;
    cout << "Give me a number\n";
    cin >> a;
    cout << "a = " << a << "\n";
    return 0;
}
```

Τύποι μεταβλητών

➤ **Ακέραιοι τύποι:** `int i;`

`int j=5;`

`int k(5);`

➤ **Κινητής Υποδιαστολής:** `float a;`

`float a=10; float b(20);`

➤ **Διπλής ακρίβειας:** `double a;`

`double a=10; double b(20);`

➤ **Boolean:**

`bool d;`

`d = true;`

`bool a = false; bool b = true;`

`bool c(true);`

Δήλωση και ανάθεση τιμής ταυτόχρονα
Εναλλακτικός τρόπος ανάθεσης τιμής

Είδη μεταβλητών στη C++

➤ Character:

char c;

char d = 'a';

char f ('a');

Προσοχή η διπλή απόστροφος αντιστοιχεί σε διαφορετικού είδους μεταβλητή που ονομάζεται C-string (είναι τύπου πίνακα χαρακτήρων)

Υπάρχουν ειδικοί χαρακτήρες που χρησιμοποιούνται στο σύστημα για συγκεκριμένες διεργασίες όπως:

\?	Ερωτηματικό	\a	κουδούνι
\'	Απόστροφος	\b	διαγραφή χαρακτήρα
\"	Εισαγωγικά	\f	αλλαγή σελίδας
\\	Ανάποδη κάθετο	\n	αλλαγή γραμμής
\r	return	\t	οριζόντιο tab

Τι θα τυπώσει η εντολή:

std::cout<< "This\nis\na\nTest\nShe said , \"How are you? \"\n\"";

This

is

a

Test

She said, "How are you?"

Τύποι μεταβλητών – character

Για να έχουμε περισσότερους από έναν χαρακτήρες θα πρέπει μετά το όνομα της μεταβλητής χαρακτήρων να γράψουμε τον αριθμό χαρακτήρων μέσα σε αγκύλη

```
char name[10];
```

Στην περίπτωση αυτή ζητούμε 10 χαρακτήρες αλλά μόνο οι 9 χρησιμοποιούνται ενώ ο 10^{ος} χρησιμοποιείται για να δηλώσει το τέλος της γραμματοσειράς

Ουσιαστικά μετατρέπουμε την μεταβλητή σε μορφή πίνακα 10 στοιχείων

Τύποι μεταβλητών

➤ Μιγαδικός τύπος:

Θα πρέπει να χρησιμοποιηθεί με συμπερίληψη ενός header file

#include <complex>

Η δήλωση μιας τέτοιας μεταβλητής μιγαδικού τύπου με πραγματικό και μιγαδικό μέρος διπλής ακρίβειας δίνεται από την σύνταξη:

```
std::complex<double> z;           // z = 0.0 + 0.0i
std::complex<double> z1(4.5);     // z1 = 4.5 + 0.0i
std::complex<double> z1(3.0,2.0); // z1 = 3.0 + 2.0i
std::complex<double> z2(z1);      // z2 = z1
```

Για τους μιγαδικούς υπάρχουν επίσης οι συναρτήσεις:

std::abs(z);	// όπου z μιγαδικός – επιστρέφει	$z = \sqrt{a^2 + b^2}$
std::norm(z);	// όπου z μιγαδικός – επιστρέφει	$zz^* = a^2 + b^2$
std::arg(z);	// όπου z μιγαδικός – επιστρέφει	$\arctan(b/a)$
std::conj(z);	// όπου z μιγαδικός – επιστρέφει τον συζυγή	
std::real(z);	std::imag(z)	// όπου z μιγαδικός – επιστρέφει real και μιγαδικό μέρος του z

Σταθερές ποσότητες

Σταθερές ποσότητες σε ένα πρόγραμμα μπορούν να δηλωθούν ως εξής:

```
double const pi (3.14159265358979);
```

```
int const my(10);
```

Αν προσπαθήσουμε να αλλάξουμε την τιμή της σταθεράς ο compiler θα μας ειδοποιήσει για λάθος.

Προσοχή θέλουν δηλώσεις μεταβλητών με ίδιο όνομα σε διαφορετικά blocks.

```
#include <iostream>

int main()
{
    double a(3.2);
    {
        int a = 5;
        std::cout << a;      Θα τυπώσει α=5
    }
    std::cout << a;          Θα τυπώσει α=3.2
    return 0;
}
```

Επεξήγηση του προγράμματος whoRyou.cc

```
cout << "Enter the name and age: \n" ;
```

Είναι εν γένει καλή ιδέα να τυπώνουμε κάτι ώστε να ξέρουμε τι περιμένει το πρόγραμμα να δοθεί σαν input

```
cin >> name >> age;
```

Αυτός είναι ο τρόπος με τον οποίο το *iostream* δίνει τη δυνατότητα να εισαχθούν δεδομένα από το πληκτρολόγιο.

Η εντολή μπορεί ισοδύναμα να χωριστεί σε 2 εντολές:

```
cin >> name ;  
cin >> age;
```

Ο τελεστής **>>** προκαλεί τις “λέξεις” από το πληκτρολόγιο να μετατραπούν και αποθηκευτούν ανάλογα με τον προκαθορισμένο τύπο.

Μπορεί να θεωρηθεί σαν ο τελεστής στη διεύθυνση της ροής των δεδομένων

Η κλάση των γραμματοσειρών - **strings**

Η standard βιβλιοθήκη που συνοδεύει τους compilers της C++ απλουστεύει το τρόπο χρήσης της ακολουθίας χαρακτήρων, παρακολουθώντας τον αριθμό χαρακτήρων σε μια γραμματοσειρά και βοηθώντας σε πράξεις και χειρισμούς.

Παρακάτω είναι και πάλι το πρόγραμμα `whoRyou.C` χρησιμοποιώντας την κλάση ***string*** για την μεταβλητή *name*

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    int age;
    string name;
    cout << "Enter the name and age: \n" ;
    cin >> name >> age;
    cout << "Your name is" << name
        << " and age is" << age << "\n";
}
```

Προσέξτε ότι θα πρέπει να εισαχθεί το header file *string* και δεν χρειάζεται να δηλωθεί ο αριθμός των χαρακτήρων της μεταβλητής *name*.

Λύσεις δευτεροβάθμιας εξίσωσης

Το παρακάτω παράδειγμα χρησιμοποιείται για την εισαγωγή της εντολής υπόθεσης:
if then ...else

Αλλά και για τον τρόπο που μπορούμε να αποφύγουμε προβλήματα στρογγυλοποίησης

Έστω ότι έχουμε να λύσουμε την εξίσωση: $ax^2 + bx + c = 0$

Με λύσεις:
$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Το αποτέλεσμα ενώ μαθηματικά είναι σωστό, εισάγει προβλήματα στις περιπτώσεις περιορισμένης ακρίβειας αριθμών:

Ως γνωστόν οι αριθμοί απλής ακρίβειας, *float*, έχουν ακρίβεια 8 σημαντικών δεκαδικών ψηφίων ενώ οι αριθμοί διπλής ακρίβειας, *double*, ακρίβεια 15 σημαντικών ψηφίων.

Επομένως ένας αριθμός που μπορεί να έχει άπειρο αριθμό ψηφίων μπορεί να αναπαρασταθεί στον υπολογιστή μόνο με περιορισμένο αριθμό.

Αφαιρώντας δύο αριθμούς με παρόμοια ακρίβεια χάνονται σημαντικά ψηφία.
Για παράδειγμα η αφαίρεση $0.12345678 - 0.12345676 = 0.00000002 = 0.2 \times 10^{-7}$
με ένα μόνο σημαντικό ψηφίο.

Αν οι δύο αφαιρούμενες τιμές χρειάζονται να στρογγυλοποιηθούν για να απεικονιστούν με 32bits, η διαφορά τους είναι ιδιαίτερα μή ακριβής.

Λύσεις 2-βάθμιας εξίσωσης – στρογγυλοποίηση

Στη περίπτωση της 2-βάθμιας, αν ο όρος b είναι θετικός και πολύ μεγαλύτερος από τον όρο $4ac$ τότε ο αριθμητής οδηγεί στο αποτέλεσμα της αφαίρεσης 2 σχεδόν ίδων αριθμών και επομένως απώλεια ακρίβειας.

Υπάρχει ωστόσο μια απλή λύση στο πρόβλημα: Μπορούμε να διαιρέσουμε τη 2-βάθμια εξίσωση με x^2 :

$$a + \frac{b}{x} + \frac{c}{x^2} = 0$$

Η οποία είναι 2-βάθμια ως προς $1/x$ και θα μπορούσαμε να τη λύσουμε ως προς $1/x$ και να αναστρέψουμε τη λύση παίρνοντας:

$$x_{1,2} = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}}$$

Η λύση με το αρνητικό πρόσημο στη μία συνδέεται με τη λύση με το αντίθετο πρόσημο στην άλλη. Η λύση που δεν έχει πρόβλημα στρογγυλοποίησης στη μία προσέγγιση αντιστοιχεί σε αυτή με το πρόβλημα στρογγυλοποίησης στην άλλη.

➤ Χρειάζεται να χρησιμοποιηθούν και οι δύο προσεγγίσεις

Κώδικας λύσης 2-βάθμιας εξίσωσης

1. Εισαγωγή των τιμών a , b και c από το πληκτρολόγιο
2. Λύση για τις δύο ρίζες
3. Εκτύπωση αποτελέσματος


Περισσότερο αναλυτικά θα χρειαστεί να κάνουμε:

1. Εισαγωγή των τιμών a , b και c από το πληκτρολόγιο
2. Εύρεση του πρόσημου του b
3. Για τη μία λύση, χρησιμοποίηση του standard τύπου με το πρόσημο μπροστά από τη ρίζα να είναι ίδιο με το πρόσημο του b
4. Για τη δεύτερη λύση, χρησιμοποίηση του ανάστροφου τύπου με το πρόσημο μπροστά από τη ρίζα να είναι ίδιο με το πρόσημο του b
5. Εκτύπωση αποτελέσματος

Το πρόγραμμα λύσης της 2-βαθμιας – quadratic.cc

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

int main()
{
    float a, b, c, x1, x2 ;
    float s, d, signb;
    // Get a, b and c
    cout << "Enter a, b, and c" << "\n";
    cin >> a >> b >> c ;
    d = b*b - 4*a*c;
    signb = 1;
    if (b < 0) signb = -signb;
    s = -(b + sqrt(d) * signb)/2;
    x1 = s/a;
    x2 = c/s;
    cout << "Roots are \n" << x1 << " " << x2 << "\n";
}
```

 // δηλώνει σχόλιο – μη εκτελέσιμη εντολή

Κάντε compile τον κώδικα με `g++ -o quadratic.x quadratic.cc -lm`

Το `-lm` σημαίνει link με την βιβλιοθήκη (το l library) μαθηματικών (m) για τη sqrt

Εξήγηση – quadratic.cc

```
#include <cmath>
```

Header file για τις συναρτήσεις μαθηματικών – θα χρησιμοποιηθεί στα περισσότερα προγράμματα που θα γράψετε.

```
d = b*b - 4*a*c;  
signb = 1;
```

Εντολές ανάθεσης και όχι εξισώσεις. Η εντολή δηλώνει ότι εκτελείται η πράξη στα δεξιά της ισότητας και το αποτέλεσμα αποθηκεύεται στη θέση μνήμης της μεταβλητής

Όλες οι μεταβλητές θα πρέπει να έχουν δηλωθεί από πριν

Προσέξτε ότι η C++ δεν έχει συμβολισμό για δυνάμεις – Το τετράγωνο γράφεται $b*b$

Θα μπορούσε να χρησιμοποιηθεί η συνάρτηση βιβλιοθήκης **pow(x,y)** που είναι x^y

Μπορούμε να χρησιμοποιήσουμε σύνθετη ανάθεση:

```
x += y;      Που είναι ισοδύναμη με      x = x + y;
```

Άλλες σύνθετες αναθέσεις χρησιμοποιούν: **-=** ***=** και **/=**

Έλεγχος υπόθεσης

`if (b < 0) signb = -signb;` Απλή περίπτωση ελέγχου λογικής και υπόθεσης

Η σύνταξη είναι:

`if (logical-expression) statement;`

Το αποτέλεσμα της λογικής έκφρασης είναι αληθές ή ψευδές

Οι λογικοί τελεστές είναι:

< μικρότερο

<= μικρότερο ή ίσο

> μεγαλύτερο

>= μεγαλύτερο ή ίσο

== ίσο

!= όχι ίσο

Προσοχή: `if (b=a)` είναι τελείως διαφορετικό από το `if (b==a)` !!!

Στη 1^η περίπτωση έχουμε ανάθεση της τιμής του `a` στη `b` και εξέταση αν είναι ίση με 0 (ψευδές) και διαφορετική από 0 (αληθές)

Λογικές εκφράσεις

Λογικές εκφράσεις μπορούν να συνδυαστούν με παρενθέσεις και λογικούς τελεστές για το σχηματισμό λογικών εκφράσεων:

```
If ( (a==0) && (b ==0) ) {  
    cout << “ Yes a and b are both zero” ;  
}
```

Οι λογικοί τελεστές είναι:

&& AND

|| OR

! NOT

Στον κώδικα που γράψαμε προηγουμένως θα μπορούσαμε να έχουμε ένα απλό if then ... else με σύνταξη:

```
If (b < 0) ) signb = -1;  
else signb =1;
```

Θα μπορούσαμε να έχουμε άγκιστρα για να εισάξουμε διάφορες εντολές:

```
If (b < 0) ) {  
    signb = -1;  
    cout << “ b is negative” ;}  
else {  
    signb =1;  
    cout << “b is positive”; }
```

Απλή έκφραση συνθήκης – **ternary operator**

Πέρα από τις σύνθετες εκφράσεις ελέγχου *if* (expression) {...} *else* {...} υπάρχει και μια απλή έκφραση ελέγχου συνθήκης που ουσιαστικά είναι ένας **τελεστής ελέγχου**

(expression1) ? expression2 : expression3

Η έκφραση αυτή είναι ισοδύναμη με την *if* (...) {} *else* {}

Αν η έκφραση 1 είναι αληθής (δηλαδή η τιμή της είναι μή μηδενική) τότε θα υπολογιστεί η έκφραση 2 και θα γίνει η τιμή της υπόθεσης της έκφρασης 1 διαφορετικά θα υπολογιστεί η έκφραση 3.

`(j < 5) ? 12 : -6`

Η έκφραση παίρνει την τιμή 12 εάν $j < 5$, διαφορετικά θα πάρει την τιμή -6

`k = (i < 0) ? n : m`

Η τιμή της μεταβλητής n θα ανατεθεί στη μεταβλητή k εάν $i < 0$ διαφορετικά θα ανατεθεί η τιμή της μεταβλητής m

Symbolic names

Σε αρκετές περιπτώσεις χρειαζόμαστε να ορίσουμε το όνομα μιας σταθεράς που ενεργεί σαν ψευδώνυμο για την τιμή που αντιπροσωπεύει. Η τιμή μπορεί να είναι μια ακολουθία χαρακτήρων και μπορεί να είναι είτε νούμερα ή γραμματοσειρά.

Η αντικατάσταση του ονόματος με την τιμή γίνεται κατά την διάρκεια του compilation σε όλα τα σημεία του προγράμματος που υπάρχει το όνομα αυτό.

Τέτοια συμβολικά ονόματα συνήθως ορίζονται με την εντολή **define** και δίνεται έξω από το κύριο πρόγραμμα.

Το symbolic name δίνεται σε κεφαλαία γράμματα ακολουθούμενο από την τιμή που αντιπροσωπεύει.

#define NAME text

Επειδή δεν είναι εκτελέσιμη εντολή δεν τελειώνει με ";" στο τέλος της εντολής.

```
#include <iostream>
#define PI 3.141593
using namespace std;
int main() {
    double area, radius=1.0;
    area = PI * radius * radius;
    cout << area << endl;
}
```

Το όνομα PI αντικαθίσταται από τον compiler με την τιμή 3.141593 στην εντολή που εμφανίζεται το όνομα PI

Διαφορά μεταξύ #define και const data member

Η βασική διαφορά είναι ότι σταθερά που ορίζεται από την εντολή #define είναι ανεξάρτητη του πεδίου εφαρμογής στον κώδικα ενώ η const data member μπορεί να οριστεί οπουδήποτε στο κώδικα και έτσι μπορεί να είναι local μεταβλητή σε αντίθεση με την define που είναι global.

Η σταθερά που ορίζεται από την #define μπορεί να επανα-οριστεί οπουδήποτε στο πρόγραμμα αλλά η const δεν μπορεί να επανα-δηλωθεί ή επανα-οριστεί

```
#include <iostream>
using namespace std;
#define X 30
const int Y = 10;
int main()
{
    const int Z = 30;
    cout << "Value of X: " << X << endl;
    cout << "Value of Y: " << Y << endl;
    cout << "Value of Z:" << Z << endl;
    return 0;
}
```

```
#include <iostream>
using namespace std;
#define X 30
int main()
{
    cout << "Value of X: " << X << endl;
    #undef X
    #define X 300
    cout << "Value of X: " << X << endl;
    return 0;
}
```

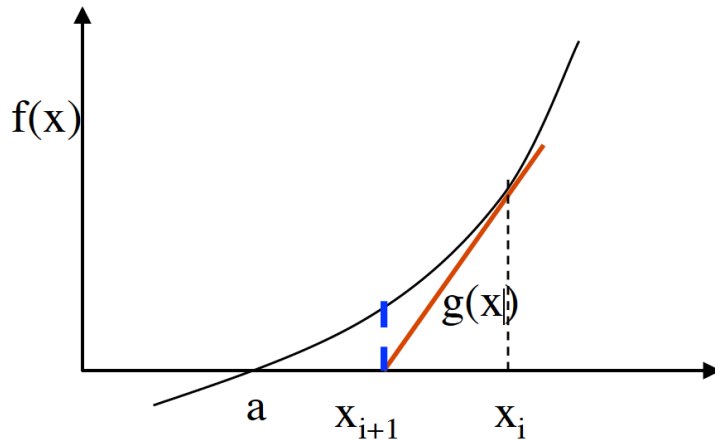
Για την αλλαγή μιας const θα είχαμε σφάλμα σε compilation

Δομή βρόχου επανάληψης

Στο παράδειγμα αυτό θα μελετήσουμε την μέθοδο εύρεσης ριζών μίας μη γραμμικής εξίσωσης με τη χρήση της μεθόδου Newton – Rapshon. Παράλληλα θα μελετήσουμε τις δομές βρόχου επανάληψης που εμφανίζονται στην C++/C.

Η μέθοδος Newton-Rhapson για εύρεση ρίζας μή γραμμικής εξίσωσης

Η μέθοδος στηρίζεται στην χρησιμοποίηση των παραγώγων $f'(x)$ της συνάρτησης $f(x)$ για ταχύτερη σύγκλιση στην εύρεση των ριζών της $f(x) = 0$



Η βασική ιδέα: Μια συνεχής παραγωγίσιμη, συνάρτηση $f(x)$ που έχει συνεχή δεύτερη παράγωγο, μπορεί να αναπτυχθεί κατά Taylor ως προς ένα σημείο x_n που είναι κοντά στη ρίζα.

Έστω ότι η πραγματική ρίζα αυτή είναι a

$$f(a) = f(x_n) + (a - x_n)f'(x_n) + (a - x_n)^2 \frac{f''(x_n)}{2!} + \dots$$

Εφόσον $x=a$ είναι ρίζα της f , τότε $f(a) = 0$ οπότε από τους 2 πρώτους όρους του αναπτύγματος έχουμε:

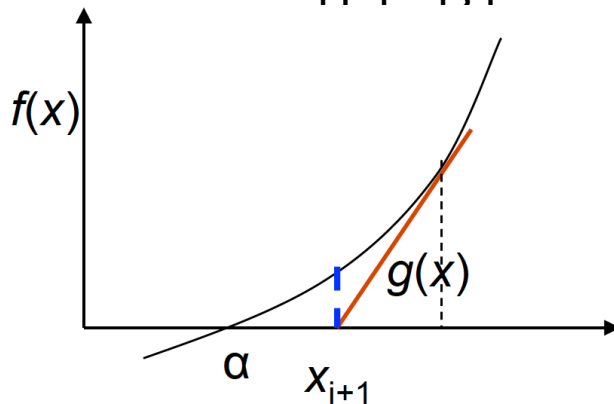
$$0 = f(a) = f(x_n) + (a - x_n)f'(x_n) \Rightarrow a = x_n - \frac{f(x_n)}{f'(x_n)}$$

Η μέθοδος Newton-Rhapson για εύρεση ρίζας μή γραμμικής εξίσωσης

Επομένως χρειαζόμαστε $f(x)$ και $f'(x)$ για την μέθοδο.

Κάθε επανάληψη της μεθόδου βρίσκει:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$



$$\text{Σφάλμα: } x_{k+1} - a = x_k - a - \frac{f(x_k)}{f'(x_k)} \Rightarrow \varepsilon_{k+1} = \varepsilon_k + \frac{f(x_k)}{f'(x_k)} \quad (\text{A})$$

Χρησιμοποιώντας τους 3 πρώτους όρους από Taylor:

$$0 = f(a) = f(x_k) + (a - x_k)f'(x_k) + (a - x_k)^2 \frac{f''(x_k)}{2!}$$

$$\Rightarrow f(x_k) = -(a - x_k)f'(x_k) - (a - x_k)^2 \frac{f''(x_k)}{2!} \Rightarrow f(x_k) = -\varepsilon_k f'(x_k) - \varepsilon_k^2 \frac{f''(x_k)}{2!}$$

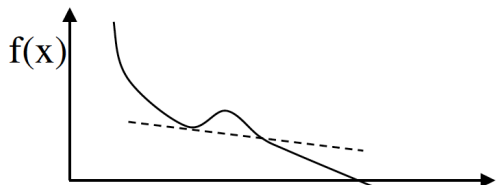
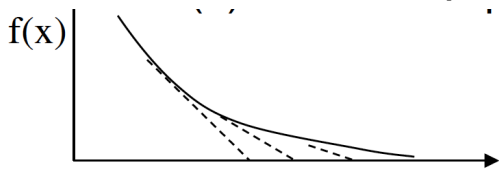
Αντικαθιστώντας στην (A)

Ρυθμός σύγκλισης:

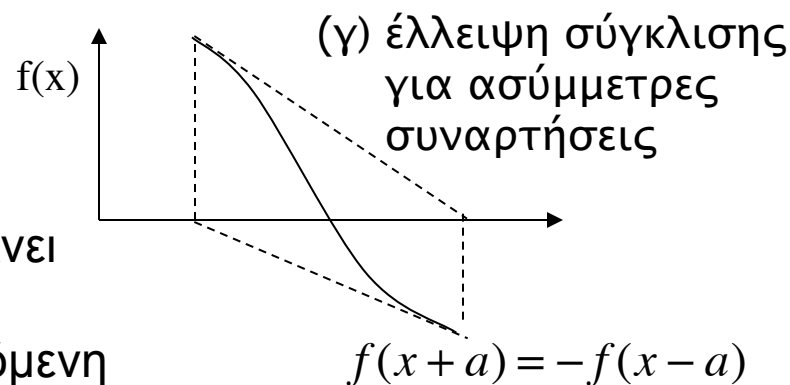
$$\varepsilon_{k+1} = -\frac{\varepsilon_k^2 f''(x_k)}{2f'(x_k)}$$

Προβλήματα της μεθόδου:

(α) Αργή σύγκλιση προς τη ρίζα
όταν $f'(x) \sim 0$ κοντά στη ρίζα



(b) Τοπικό ακρότατο στέλνει
μακριά την τιμή x_{k+1} που
χρησιμοποιείται στην επόμενη
επανάληψη



Ο κώδικας Newton-Raphson

Η ιδέα του κώδικα αρχικά:

Πρέπει να ορίσουμε την ακρίβεια της τιμής της ρίζας που επιθυμούμε για να σταματήσουμε τις επαναλήψεις εύρεσης ρίζας. Για παράδειγμα $|a - x|$

Εισαγωγή της αρχικής τιμής της ρίζας

Επαναλήψεις για εύρεση της ρίζας. Προσδιορισμός του μέγιστου αριθμού επαναλήψεων πριν θεωρηθεί ότι δεν υπάρχει σύγκλιση

Υπολογισμός της νέας λύσης σύμφωνα με τη σχέση Newton

Αν η διαφορά της νέας λύσης από την προηγούμενη είναι μικρότερη της ακρίβειας τότε θεωρούμε ότι βρέθηκε η λύση - Διαφορετικά συνεχίζουμε τις επαναλήψεις.

Δήλωση αν η μέθοδος συνέκλινε ή όχι μετά από N αριθμό επαναλήψεων

Ο κώδικας Newton-Raphson

```
//=====
// Newton-Raphson method for finding roots
//  $4x = \cos(x)$ 
//=====
#include <iostream>
#include <cmath>
using namespace std;
#define N 100 // max number of iterations
int main(){
    double p, pnew, f, dfdx, tol;
    cout << "What is the tolerance?" ;
    cin >> tol;
    cout << "Enter starting value of x:"
    cin >> pnew;
//main loop
    for (int i = 0; i < N; i++) {
        p = pnew; //anathesi tis proigoumenis lysis
        // Evaluate the derivative and the function
        f = 4*p - cos(p);
        dfdx = 4 + sin(p); //paragwgos tis 4x-cos(x)
        pnew = p - f/dfdx;
        if (abs(pnew - p) < tol) {
            cout << "root is " << pnew << " to within " << tol << "\n";
            return 0; } // end of if
    } // end of loop
    cern << "Failed to converge after " << N << " iterations." << endl; return 1;}

```

Compilation με:

g++ -o newton.x newton.cpp -lm

Ο κώδικας Newton-Raphson – Εξήγηση

```
#define N 100
```

Έχουμε αναφέρει ήδη τη χρήση της ψευδο-εντολής αυτής. Η N δεν είναι μεταβλητή, δεν έχει συγκεκριμένο τύπο και δεν αποθηκεύει χώρο στη μνήμη γιατί αντικαθιστά τη τιμή της σε επίπεδο compiler.

```
for (int i=0; i< N; i =  
i+1)  
{  
...  
}
```

Η βασική δομή βρόχου επανάληψης.
Οι εντολές που βρίσκονται στο περικλείονται από τα άγκιστρα θα εκτελεστούν συνεχώς εφόσον η συνθήκη $i < N$ ικανοποιείται.

Ξεκινώντας από τη τιμή $i = 0$ και αυξάνοντας το μετρητή i κατά 1 κάθε φορά που φθάνει στην τελευταία εντολή της δομής αυτής.

Η γενική έκφραση της δομής βρόχου είναι:

```
for (αρχική τιμή μετρητή; λογική έκφραση; αύξηση μετρητή)  
{  
    block εντολών  
}
```

Να σημειωθεί ότι η λογική έκφραση ελέγχεται πριν την είσοδο στο block εντολών.
π.χ. εάν $N=0$ τότε η συνθήκη $i < N$ είναι ψευδής και δεν θα εκτελεστεί καμία εντολή.
Στη C++ η αύξηση μιας μεταβλητής κατά μία μονάδα ($i=i+1$) χρησιμοποιεί τον λεγόμενο **unirary** τελεστή ++ οπότε μπορούμε να γράψουμε **for (int i=0; i<N; i++)**

while δομή βρόχου

Θα μπορούσαμε να επιτύχουμε το ίδιο αν χρησιμοποιούσαμε βρόχο while

Στην περίπτωση αυτή το αντίστοιχο τμήμα του κώδικα θα ήταν:

```
i=0;  
while (i < N) {  
    ....  
    i++;  
}
```

Παρατηρήστε ότι θα πρέπει να φροντίσουμε να κάνουμε την ανάθεση της αρχικής τιμής καθώς και την αύξηση της τιμής του μετρητή μόνοι μας. Αυτός είναι ο βασικός λόγος που δεν προτιμάται η χρήση της δομής αυτής βρόχου.

Η γενική σύνταξη της εντολής while είναι:

```
while (λογική έκφραση) {  
    block εντολών  
}
```

Οι εντολές εκτελούνται μέσα στο block των εντολών εφόσον η συνθήκη ικανοποιείται.

Εναπόκειται στο χρήστη να θέσει τη σωστή συνθήκη και να εξασφαλίσει την έξοδο από μία ατέρμονη επανάληψη

Η δομή do ... while

Η δομή αυτή είναι λιγότερο χρησιμοποιούμενη. Η γενική σύνταξη είναι:

```
do {  
    block εντολών  
} while (λογική έκφραση) ;
```

Η σύνταξη της δομής αυτής είναι παρόμοια με αυτή της while δομής με τη διαφορά ότι η λογική έκφραση εξετάζεται αφού έχει εκτελεστεί μία φορά το block των εντολών

Η σύνταξη της while δομής μπορεί να μην επιτρέψει την εκτέλεση του block των εντολών αν η συνθήκη δεν ικανοποιείται. Και στις δύο περιπτώσεις οι εντολές εκτελούνται τόσο όσο υποστηρίζει η λογική έκφραση.

Ο κώδικας Newton-Raphson – Εξήγηση

```
if (abs(p-pnew) < tol) {  
    ...  
}
```

Ο έλεγχος που εκτελείται στο στάδιο αυτό αποτελεί τη λεγόμενη συνθήκη διακοπής εκτέλεσης της μεθόδου. Η μέθοδος σταματά όταν η απόσταση της νέας λύσης είναι πολύ κοντά στην προηγούμενη σηματοδοτώντας την επίτευξη σύγκλισης

```
return 0;
```

Η εντολή αυτή προκαλεί έξοδο από το main πρόγραμμα. Η τιμή 0 είναι η τιμή εξόδου του κώδικα. Συνήθως η τιμή 0 χρησιμοποιείται για να δηλώσει ότι η έξοδος από το πρόγραμμα δεν συνοδεύεται από κάποιο πρόβλημα.

Παρατηρήστε ότι στο τέλος του main χρησιμοποιείται η τιμή 1 για να δηλώσει ότι δεν επιτεύχθει σύγκλιση.

Με τον τερματισμό του προγράμματος μπορείτε να ελέγξετε το κωδικό επιστροφής δίνοντας την εντολή `echo $status` (για τον `tcsch` φλοιό εντολών) στο terminal window στο οποίο τρέχате το πρόγραμμα (για φλοιό εντολών `bash` η εντολή είναι `echo $?`)

Θα μπορούσατε να επιστρέψετε το κωδικό εξόδου με την εντολή **exit (0)**; Το νούμερο στην παρένθεση είναι ο επιθυμητός κωδικός ελέγχου.

Έξοδος από την ακολουθία του loop βρόχου

Υπάρχουν και άλλοι τρόποι για έξοδο από τις εντολές βρόχου χωρίς την έξοδο από το κύριο πρόγραμμα.

Η εντολή `break` αυτόματα σταματά την εκτέλεση των εντολών του loop.

Θα μπορούσαμε να γράψουμε:

```
for (int i=0; i< N; i = i+1) {  
    ...  
    if (abs(p - pnew) < tol) break;}
```

Η εντολή `break` αυτόματα σταματά την εκτέλεση των εντολών και μεταφέρει τη ροή του προγράμματος απευθείας στην πρώτη εκτελέσιμη εντολή μετά το τέλος του block βρόχου.

Το πρόβλημα που υφίσταται στην περίπτωση αυτή είναι ότι καταλήγουμε στις ίδιες εντολές όπως και στην περίπτωση της μή σύγκλισης και θα πρέπει να αποφασίσει ο χρήστης ελέγχοντας το μετρητή αν υπήρξε σύγκλιση. Για παράδειγμα θα είχαμε:

```
for (int i=0; i< N; i = i+1) {  
    ...  
    if (abs(p - pnew) < tol) break;  
}  
if (i>= N) {  
    cerr << "Failed to converge after "<< N << " iterations"  
    return 1;  
}  
cout <<"Root is " << pnew << " to within" << tol << "\n";  
return 0;
```

Έξοδος από την ακολουθία του loop βρόχου

Εκτός της εντολής break υπάρχει και η εντολή **continue** που προκαλεί αλλαγή στη ροή εκτέλεσης του προγράμματος.

Σε αντίθεση με την εντολή break που σταματά την εκτέλεση των εντολών του βρόχου και μεταφέρει τη ροή του προγράμματος στο τέλος του βρόγχου, η εντολή continue σταματά την εκτέλεση των εντολών μόνο για την συγκεκριμένη επανάληψη του loop

```
for (int i=0; i< N; i = i+1) {  
    ...  
    if (abs(p – pnew) > tol) continue; }  
    cout << "Root is " << pnew << " to within" << tol << "\n";  
    return 0;  
}  
cerr << "Failed to converge after " << N << " iterations"  
return 1;
```

Παρατηρήστε ότι η ανισότητα έχει αντιστραφεί ώστε να επιτευχθεί η συνέχιση της εκτέλεσης του προγράμματος και μετά την εντολή continue.

cerr

Στο τέλος του προγράμματος χρησιμοποιείται το αντικείμενο **cerr** που περιέχεται στη μέθοδο input/output **iostream** και στέλνει την έξοδο του προγράμματος στην μονάδα **stderr** αντί της μονάδας **stdout**.

Ο λόγος που χρησιμοποιείται είναι για να κρατηθούν τυχόν λάθη ή προβληματικές περιπτώσεις του κώδικα που θα μπορούσαν να αγνοηθούν σε άλλες περιπτώσεις.

Πίνακες

Ένας πίνακας στην C++ ορίζεται σαν: **τύπος όνομα [πλήθος στοιχείων]**

Η προηγούμενη περίπτωση αντιστοιχεί σε πίνακα μιας διάστασης

```
float temp[365];
```

```
int const N[200]; double test[N]; //δήλωση με παράμετρο
```

Η αρίθμηση των στοιχείων του πίνακα ξεκινά από το 0-στοιχείο: **temp[0]=10;**

Μπορούμε να δώσουμε τις τιμές ενός πίνακα ως εξής:

```
int test[5] = {1,2,3,4,5};
int testit[] = {0,1,2,3,4,5,6,7,8,9}; //μέγεθος πίνακα 10
char alpha[5] = {'a', 'b', 'c', 'd', 'e'};
int a[5] = {12,5,4}; // οι τιμές θα είναι α=={12,5,4,0,0}
```

Πίνακας δυο διαστάσεων γράφεται: **τύπος όνομα [πλήθος1][πλήθος2]**

```
float temp[10][5];
```

Ένα στοιχείο του πίνακα δίνεται **temp[3][1];**

Αρχικές τιμές δίνονται ανά γραμμή ως εξής: **int temp[2][3]= { {0,1,2}, {3,4,5} };**

➤ Στη C++ οι πίνακες αποθηκεύονται κατά γραμμές σε αντίθεση με την Fortran

Πίνακες μεγαλύτερων διαστάσεων ορίζονται ανάλογα: **τύπος όνομα [πλ1][πλ2][πλ3]**

Πίνακες

Χρειάζεται να ξεχωρίσουμε δύο είδη αριθμών που δίνουμε για να καθορίσουμε το μέγεθος ενός πίνακα :

- (α) Το μέγεθος το οποίο θα πρέπει να δεσμευθεί στη μνήμη του υπολογιστή για να αποθηκεύσει τα στοιχεία του πίνακα
- (β) Το μέγεθος που χρησιμοποιείται στο πρόγραμμα όταν τρέχει και πιθανόν να είναι μικρότερο από το πραγματικό μέγεθος του πίνακα.

Έν γένει κάποιος πίνακας μπορεί να χρησιμοποιηθεί σαν μεταβλητού μεγέθους αλλά στην περίπτωση των 2-Δ πινάκων θα πρέπει να δοθεί οπωσδήποτε το μέγεθος του δείκτη που αλλάζει πιο γρήγορα.

Στην C++ τα στοιχεία αποθηκεύονται κατά γραμμές και επομένως θα πρέπει να δοθεί οπωσδήποτε το μέγεθος των στηλών διαφορετικά ο compiler δεν γνωρίζει που τελειώνει μια γραμμή και που αρχίζει η επόμενη.

Διανύσματα

Στην C++ ορίζεται μια νέα δομή πολυ εύχρηστη που ονομάζεται **vector**

Πρέπει να δοθεί η κλάση αυτή με το αντίστοιχο header file : **#include <vector>**

Ο ορισμός ακολουθεί τη σύμβαση: **std::vector<τύπος> όνομα(πλήθος);**

std::vector<double> v(30); // 1-Δ πίνακας διπλής ακρίβειας, v με 30 στοιχεία

```
#include <vector>
#include <iostream>

int main() {
    int N;
    std::cin >> N;
    std::vector<double> v(N);
    //v[0], v[1], ..., v[N-1];
}
```

- Μπορούμε να δώσουμε στοιχεία στο διάνυσμα με την εντολή ***vectorname.push_back(number);***
- Μπορούμε να βρούμε το μέγεθος του διανύσματος (πλήθος στοιχείων) ***vectorname.size();***