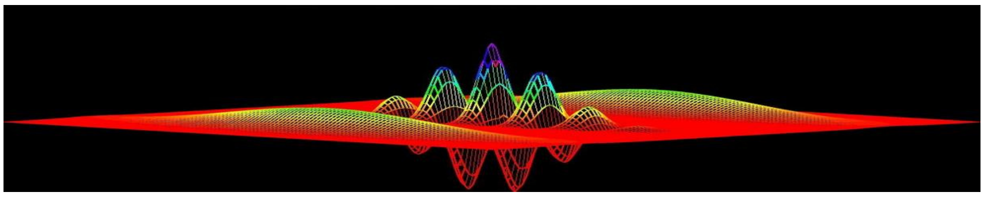
Computational Physics

numerical methods with C++ (and UNIX)



Fernando Barao
Instituto Superior Tecnico, Dep. Fisica
email: barao@lip.pt

C++ general rules

- ✔ C++ is case sensitive
- ✔ A C++ statement may begin at any place in the line and can continue into the next line
- ✔ The end of the statement is indicated by a semicolon **;**
- ✔ There can be multiple staements in a line `int a=5 ; int b=10 ;`
- ✔ Comments to code can be inserted by using // `int a=5 ; //...`
- ✔ A large part of the code can be commented using /* ...*/
- ✔ The name of a variable must start with a letter and shal contain only letters, numbers and underscore _
- ✔ Every C++ program has a main function

```
#define PRINT
#include <iostream>
  int main()   {
  int a = 5;
  std::cout << a << std::endl;
  return 0; //successful return (can be omitted)
  }
```

C++ data types

✔ A variable has allways to be declared in order the appropriate space is reserved in memory by the compiler

✔ Once declared, a numerical variable can be initialized or evaluated

```cpp
// integers
int a = 5;
int a; a=5;
int a(5);
unsigned int year; //positive integer

// characters
char a = 66; // 'B' (66 = int code)
char a = 'B'; //single quotes

// constants
const int a = 5; //cannot be modified

// reals
float b = −10.50; //single precision
float b = −1.05e+1;
double pi = 3.141592....; //double prec
```

```cpp
// boolean vars
bool flag = true; //or false

// strings (C++ std lib)
string name = "alberto";
string name("alberto");

// character strings
char word[20] = "four";
/* word[4]='\0' (null character)
   the null character is automatically
   added to the end of the character
   string enclosed in double quotes */
```

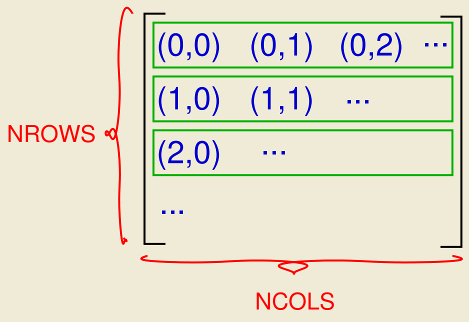
C++ arrays storage

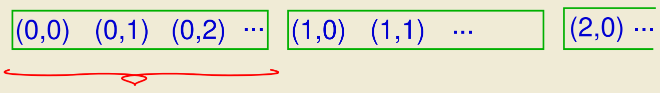
✔ multi-dimensional arrays creation :
- as *rectangular sequential arrays* where rows are sequentially stored in memory
- as *arrays of arrays* (to be seen later on !)

```cpp
// 1−dim array
double v[30] = {0}; //init to zero all
cout << "1st element='' << v[0] << endl;

// matrices
// 2−dim array declaration and init
// m[ROWS][COLS] : 10 ROWS * 5 COLS elements
int m[10][5] = {
                {0, 1, 2, 3, 4}, //row 0
                {5, 6, 7, 8, 9}, //row 1
                {10, 11, 12, 13, 14}, //row 2
                ...
                {45, 46, 47, 48, 49} //last row
              };

// print elements in memory order (see pointers section)
for (int i=0; i<50; i++) {
  printf("%d ",*m);
  m++;
}
```





C++ data types (cont.)

Type	Description	Byte size
short int	short integer	2
short	ranges from -32768 to 32767	
signed short int	ranges from -32768 to 32767	
unsigned short int	ranges from 0 to 65535	
int	integer	4
signed int	ranges from -2147483648 to 2147483647	
unsigned int	ranges from 0 to 4294967295	
float	floating point number, single precision	4
double	floating point number, double precision	8
long double	floating point number, long double precision	12
bool	boolean value, *true* or *false*	1
char	character	1
signed char	one byte integer from -128 to 127	
unsigned char	one byte integer from 0 to 255	

C++ data structures

✔ A data structure groups a set of characteristics of a given object (it is the prelude of a *class* in C++)

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

// define structure
struct alunoIST {
  string name; // nome
  float mark; // nota
};

int main() {
  alunoIST A;
  A.name = "Joao";
  A.mark = 20.0;
}
```

C++ pointers

```
// declare pointer to an integer variable and set it to NULL
int *p = NULL;

// assign address of an integer number to pointer
int a = 5; p = &a; // p points to a variable

// deassign pointer: get value pointed to
int c = *p; // c=5

// arrays and pointers: the array name is a pointer to the 1st element
    of the array
float v[10];
float a = *v; //retrieves 1st element of the array (float a=v[0];)
float *p2 = &v[1]; //pointer to 2nd element (similar to float *p2 = v+1)
    ;

// passing array to a function by reference/pointer
float a[100];
function(a); //function prototype: void function(float [])

// character string pointer
char *word = "four"; //similar to: char word[5]="four"
```

C++ pointers (cont.)

✔ memory allocation in C++ : the *new* and *delete* operators

memory allocated dynamically by the user - **heap memory region**

```
//array of strings
string s[10];
string *s = new string[10]; //allocating memory!!!

// matrice defined as arrays of arrays (pointer to pointers!)
// Define matrice of 10 ROWS * 5 COLS
int **m = new int*[10]; // pointer to an array of 10 pointers to
    integers
for (int i=0; i<10; i++) {//ROW arrays
  m[i] = new int[5]; // m[i] is a pointer to 5 elements 1-dim array
}

// setting values to the 50 allocated memory positions
for (int i=0; i<10; i++) {
   for (int j=0; j<5; j++) {
     m[i][j] = i*5 + j;
   }
}
```

C++ pointers (cont.)

✔ After memory dinamically allocated in the program through the *new* operator we shall at the end of the program free the memory with the *delete* operator

```
// accessing and print sequential elements in memory
for (int i=0; i<10; i++) {
  int *p = m[i];
  for (int j=0; j<5; j++) {
    cout << *p << " " << flush;
    p++;
  }
}
cout << endl;

//free arrays memory
delete[] s;
for (int i=0; i<10; i++) { delete[] m[i]; }
```

C++ control statements

```
// if-else
int a = 10;
if ( a < 5) {
  true statement;
} else {
  false statement;
}

// while
double dx=1., eps=1.e-6;
while (dx > eps) {
  statements;
}

// do-while
do {
} while (dx > eps);

//for loop
for (int i=0; i< 10; i++) {
   statements;
}
```

C++ operators

arithmetic

$+$	sum
$-$	subtraction
$*$	multiplication
$/$	division
%	modulo (remainder)

compound assignation

$a+=b$	$a=a+b$
$a-=b$	$a=a-b$
$a*=b$	$a=a\times b$
$a/=b$	$a=a/b$
$a*=b+c$	$a=a\times(b+c)$
$a++$	$a=a+1$
$++a$	$a=a+1$
$a--$	$a=a-1$
$--a$	$a=a-1$

logical

$a==b$	equal to
$a!=b$	not equal to
$a<b$	less than
$a<=b$	less than or equal to
$a>b$	greater than
$a>=b$	greater than or equal to
$a\&\&b$	AND
$a\|\|b$	OR
$!a$	boolean opposite

bitwise

$<<\ >>$	left and right bit shit
$\&\|$	bit AND OR

others

sizeof(a)	byte size

C++ operators (cont.)

✔ Arithmetic operators **(*) and (/)** have precedence over **(+) and (-)**

```
What C++ code to evaluate:
a + b/c +d
```

Unary operators (only act on single operands) like **(++), (–) and signs (+), (-)** have precedence over arithmetic operators

```
  What does this C++ code:
     int a,  b= 5, c;
     b = a++; // b=?
     c = ++a; // c=?
```

C++ functions

- ✔ A function is a self contained program segment that carries out some specific, well defined task.
- ✔ Every C++ program consists of several functions, one of them mandatory : *main()*
- ✔ A function can return a value, values (arrays) or nothing.
- ✔ A function needs to be declared before being used ; *function prototyping* is needed if function come after

```

#include <cstdlib> // exit()
#include <cstdio> // printf

//function prototyping
double factorial(int);

///////////////////////////////////////////////
int main() {
  for (int i=0; i<=20; i++) {
    printf("factorial(%d)=%12.3e\n",i,factorial(i));
  }
  return 0;
}

///////////////////////////////////////////////
double factorial(int n) {
  double fact=1.;
  if (n<0) {exit(1);} //abort prog if n negative
  for (int count=n; count > 0; --count)
     fact *= (double)count;
  return fact;
}
```

C++ variables

✔ **Global variables**

They are defined outside the main function and user defined functions. They are available to the program and user functions.

```
int n; // global variable
double factorial(); //function prototyping
int main() {
 for (n=0; n<=20; n++)
   {printf("factorial=%12.3e\n",n,factorial());}
 return 0;
}
```

✔ **Local variables**

Variables defined inside the functions and private to them or within C++ code blocks { ... }.

The return from the function frees the local variable locations (lost) !

C++ variables (cont.)

- ✔ **Static variables**

  Variables defined inside the functions can be declared as *static* and therefore their value is preserved between calls to the function.
  Mechanism that can be used to run code only once.

```
double F(int n) { //function code
  static int initflag = 0;
  if (!initflag) {
    do initialization statements;
    initflag++;
  } // just run once
}
```