Numerical Methods
Lesson 3

Dr. Jose Feliciano Benitez

Universidad de Sonora

Review analysis technique so far

- Code management
- Generation of data
- Data management
- Data integrity check
- Data visualization/publishing
- Data modeling/fitting
- Proper graph labeling

Goals for this lesson

Deeper understanding of c/c++

- Variables: bool, short, int, long, float, double, char, string
- Arrays
- loops
- Functions
- Classes
- Memory management
- Debugging

Integer variables

- `bool`
  - values: `0 (false) , 1 (true)`
  - Common usage:
    ```
    bool x=true;
    if(x){ //do something }
    ```
  - Negation : `!x` , returns opposite value
- `short`
  - values : `0, +/-1, +/-2,` …
- `int and long`: values same as short but accept larger numbers. **Important** to know the size of the numbers in your dataset to avoid truncation.

Basic computer memory understanding:
All objects are stored in physical memory called bits
- bool : 1 byte = 8 bits (smallest block)
- short : 2 bytes = 16 bits
- int : 4 bytes = 32 bits
- long : 8 bytes = 64 bits

Check with command, eg. `sizeof(bool)`

1 byte storing the number 2:

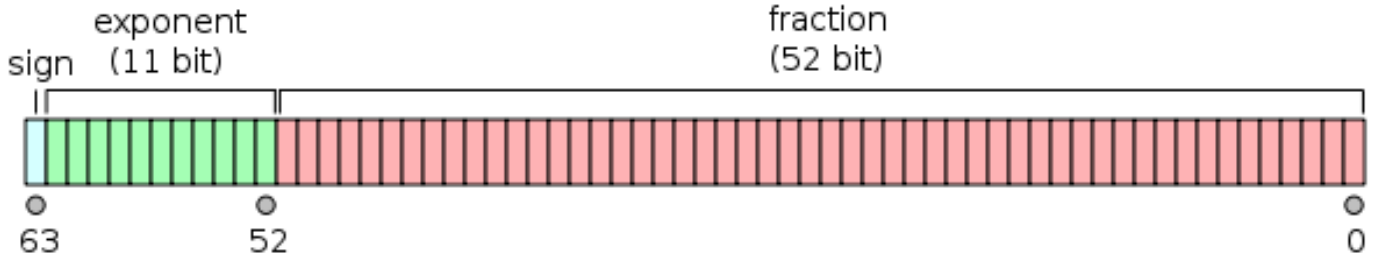
0	0	0	0	0	0	1	0

A bit is a physical element with magnetic orientation up or down (1 or 0)
How are numbers stored?
1 bit can store two values: 0=down, 1=up
2 bits can store 4: 0=00, 1=01, 2=10, 3=11
3 bits : 0=000, 1=001, 2=010, …. , 7=111
Second notation (eg. 001) is called the binary representation of the number. In general n bits can store 2^n numbers, largest value is 2^n -1

Float and Double

- float uses 4 bytes, store decimals up to 6 significant figures. Eg. 100000 or 0.00001
- double uses 8 bytes, store decimals with up to 16 significant digits: 1,000,000,000,000,000

Note 1: It is possible to store larger numbers, but not more precision (significant figures). Larger numbers are stored using an 11 bit exponent:



Note 2: it is possible to cast a float or double into a integer:

```
int d = 3.14159;
```

but d will loose the decimal part, after this operation d=3

Char and string

- `char` is 4 byte object
  - Values are single characters. Eg: `char c='a';`
  - Each char value can be mapped to a number.
- `string` is a complex object/structure, don't confuse with the simple variables. It uses 24 bytes(!).
  - Values are long text strings. Eg:

  `string text="metodos";`

  But can perform operations:

  `text += "numericos";`

  After this operation `text="metodosnumericos"`

Pointers

- All c++ objects can be handled via pointers. A pointer is created using the * symbol. Example:

```
int * x = NULL;
```

- A pointer is not a variable, cannot assign a number to it. A pointer is an address to the location of an actual object. Example:

```
int  y = 10;  // y is the actual variable
int * x = &y;  // x is a pointer with the address of y
cout<<y<<endl;  // will print 10
cout<<*x<<endl;  // will print 10
cout<<x<<endl;  // will print the address of y
```

- **NOTE:** Pointers are important part of coding because copying or passing actual objects from one part of the code to another is an expensive operation. Instead, we pass just the address of existing objects.
- **NOTE**: if you try to use a NULL pointer the program will crash. Very common coding bug, because we loose track of the values of the pointers in the code. You need to develop good logic skills.

Arrays

- Arrays are relatively simple things once the basic varibles known. Eg.:
  - `int  X[2] ; //contains 2 float variables which are accessed by the indices 0 and 1 : X[0] and X[1]`
- How to set the values:

  ```
  X[0] = 1;
  X[1] = 2;
  ```

  Or more compactly: `X = {1,2};`
- **Interesting note**: if you use an index larger than the array, for example X[3] in this example, the code will not crash, you will read the memory allocated after the array. Which will give you some unknown value. Common coding problem.
- Arrays are important because we can store a list of numbers (data) and use a in loop.

Loops

- Example **for loop**:

```
for(int i=0;i<10;i++){
  cout<<i<<endl; // code block will print 0 to 9
}
```

`i` is an internal variable to the loop, can be used to change the value of other quantities in a systematic way. Each iteration the condition `i<10` is checked. First value of i is 0, at end of code block i is automatically incremented by one (`i++`) **.**

- Example **while loop** (same behavior as for loop above):

```
int i=0;//initial value
while(i<10){//condition that will terminate the loop
   cout<<i<<endl;
   i++; //here i requires manual increment
}
```

- Choose type of loop to simplify code depending on the situation. While loops are more appropriate when an external quantity is already existing and can be used to test the condition.

functions

- Example: a function with an int argument input returning an int result

```
int f(int x=0){
    int y = x*10; //code doing something with the input
    return y;  //return int value
}
```

- Multiple functions can be defined in the same program

```
int f(int x){
  return x*10;
}

void print(int x){
  cout<<x<<endl;
}

int main(){// f and g are helper functions of main
  int y=f(10);
  print(y);
  return 1;// return value for main just says all is ok.
}
```

- Note for ROOT scripts there is no main() function. Instead we must give the same name as the script file name.

c++ class

- Example code defining a class and using it inside main function

```
//class definition
class particle {

public:
  particle(int x, int y);  // constructor
  int type;//public data member
  int f(int i);//public method (operation)

private:
  int mass;//private data member

}


// definition of class constructor and methods
particle::particle(int x, int y):
type(x),
mass(y){
 cout<<"Particle was defined with type="<<type<<" and mass="<<mass<<endl;
}


int particle::f(int i=0){ //define the particle f method
  return type*mass*i;
}



/// code using the above class, usually a separate file
int main(){

 particle p(1,10); // create an object of type particle and initialize data members
 cout<<p.f(100)<<endl;  // print value 1*10*100
 cout<<p.type<<endl;// type is public, so can be accessed.

 return 1;
}
```

Exercises

- Create your own code for each one of the above topics:
  `variables.C`
  `pointers.C`
  `arrays_loops.C`
  `functions_class.C`
  code complexity is whatever you want. Can even use the data from the previous lessons.
- Then publish it in the web Results area for Lesson3.