C++ “Guided” Tutorials

Brian Krar, Meng

EIEIOO

Intro

Goal: Getting comfortable with C++

- Build up a foundation/some familiarity

Introductory look at some of the features of C++

- If you have already used some C++, some of these tutorials are probably straightforward, but others might be worth a refresher

Will start with a few “mini-exercises” just to get warmed up

- Slides should be uploaded, so follow along as you see fit
- Can jump ahead to some of the exercises at the back, if you find the pace is too slow

Mini-Exercise 1 – Input/Output

Write a script called sum_int.cc that prints the sum of two ints (inputted from the keyboard)

```
//Code that finds the sum of 2 ints (from keyboard input)
#include <iostream>
using namespace std;

int main(){









    return 0;
}
```

How to Run (Compile!) Your C++ Code

Method 1) [Canonical Way] – g++ compiler

g++ -o sum_int sum_int .cc

Generates an executable file named sum_int.out in the local directory. Run program as an executable (like any other linux command):

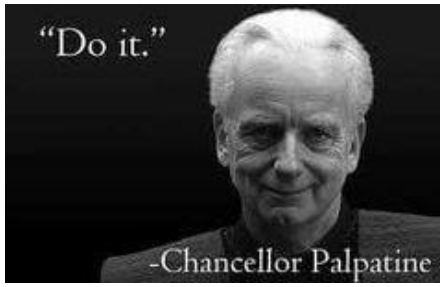
./sum_int

Method 2) [The ROOT way] - use the ROOT compiler (ACLiC for CINT).

.L sum_int.cc+

main()

- **<u>Always</u> Compile your code!**



Mini-Exercise 1 – Input/Output

Write a script called sum_int.cc that prints the sum of two ints (inputted from the keyboard)

Code should look something like this:

```
//Code that finds the sum of 2 ints (from keyboard input)
#include <iostream>
using namespace std;

int main(){
int num1, num2, sum;
    cout << "\n Sum of two numbers :\n";
    cout << "-------------------------\n";
    cout << " Input 1st number : ";
    cin >> num1 ;
    cout << " Input 2nd number : ";
    cin >> num2;
    sum = num1 + num2;
    cout <<" The sum of the numbers is : " << sum << endl;
    cout << endl;
    return 0;
}
```

Mini-Exercise 2 – Write Your Own Script Containing a Function

Write a script called multiply_numb.cc with an “multiply” function that will return the product of two integers a and b

In your main, print the result

Mini-Exercise 2 – Solution

Write a script called multiply_numb.cc with an "multiply" function that will return the product of two integers a and b

In your main, print the result

Code should look something like this:

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

int multiply (int a, int b){
    double p = a * b;
    return p;
}

int main()
{
    int a = 2;
    int b = 5;
    int prod = multiply(a, b) ;
    cout << " Numbers are  a      :     " << a << "  b : " << b <<endl;
    cout << " Product (a x b)     :     " << prod << endl;

}
```

Mini-Exercise 2.5 – Arrays and Loops

Modify script such that it fills an array of size n from i = 0,1,…n,
The value in each entry should be = i*p, where p = a * b (from code before)

- **Print the value of each entry in the array**
- **Save output of each entry to a file called “prod_output.txt”**

Notes:

- In C++, arrays are indexed starting at 0
- One way to create the .txt file is using “ofstream” (requires the <fstream> header)

Mini-Exercise 2.5 – Arrays and Loops - Solution

Fill an array of size n from i = 0,1,…n
The value in each entry should be = i*p, where p = a * b (from code before)

- **Print the value of each entry in the array**
- **Save output of each entry to a file called “prod_output.txt”**

Code should look something like this:

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

int multiply (int a, int b){
    double p = a * b;
    return p ;
}

int main()
{
    ofstream file_out;
    string file_name = "prod_output.txt";
    file_out.open (file_name.c_str());

    int n = 100;
    int arr[n];
    int a = 5;
    int b = 2;

    int prod = multiply(a, b) ;


    for(int i=0;i<n;i++) //for loop
    {
      arr[i] = i*prod;
      cout << " Arr =   " << arr[i] << " prod = " << prod<<endl;

      // File name //
      file_out << " Arr =   " << arr[i] << " prod = " << prod<<endl;

    }


}
```

Worked Example #1 – Using Structures in C++

Suppose we are writing a program involving particles, and that for each particle, we are interested in its mass and momentum

Brute force: Could write a variable for each quantity:

```
double mass_1;
double mass_2;
double mass_3;
double momentum_1;
double momentum_2;
double momentum_3;
```

This can get quite tedious -- in C++ we can find a better way

- We know that mass_1 and momentum_1 are for the same particle , same for mass_2 and momentum_2, etc

Solution: Use a 'struct', which is a data structure that can link together related variables

```
//Define a struct which contains two variables: mass and momentum

struct particle {
  double mass;
  double p;
}; // note the semicolon after the closing brace of the struct here
// this is important! -- if omitted, it can cause compilation issues

// create an object which is an instance of this struct type:

particle p1;

// Can assign the values of the variables for this object, using the syntax:
// object.variable = value;

p1.mass = 0.982;
p1.p = 0.0;

// anytime we want to access the variables within the struct,
// we do it by writing object.variable
```

This can be now used, just like you would other basic data types (like int, double etc.)

Worked Example #1 – Using Structures in C++

Suppose we are writing a program involving particles, and that for each particle, we are interested in keeping track of its mass and momentum

Option 1 (Brute force):

```
double mass_1;
double mass_2;
double mass_3;
double momentum_1;
double momentum_2;
double momentum_3;
```

But what if we were considering:
100 particles? 10000?

Option 2 (using 'struct') :

```
//Solution (in C++) -- struct -- a data structure that can link together related variables

//Define a struct which contains two variables: mass and momentum

struct particle {
  double mass;
  double p;
}; // note the semicolon after the closing brace of the struct here
// this is important! -- if omitted, it can cause compilation issues

// now that we have defined this 'particle' struct, we can use it just like the basic data types
// you learned about earlier like (int, double)!

// create an object which is an instance of this struct type:

particle p1;

// Can assign the values of the variables for this object, using the syntax:
// object.variable = value;

p1.mass = 0.982;
p1.p = 0.0;

// anytime we want to access the variables within the struct,
// we do it by writing object.variable

cout << "Particle 1: mass = " << p1.mass;
cout << "; momentum = " << p1.p <<endl;

// and we can make a second 'particle' using the same defined struct

particle p2;
p2.mass = 0.938;
p2.p = 25.0;
cout << "Particle 2: mass = " << p2.mass;
cout << "; momentum = " << p2.p <<endl;

// This is much nicer than making a bunch of variables
// Also now with the structure,  we know that p1.mass and p1.momentum are both properties of particle p1
```

Better, but …

Reminders From Yesterday

Objects and classes

- Class
  - Can define a custom data type
  - Blueprint/details to define properties and behaviour that an object should have
  - Contains data (properties) and functions (behaviour)
    - Data: data members, attributes
    - Functions: methods, function methods
- Object
  - Specific instance created from the class (blueprint)
  - General structure of every object is the same
  - Properties may be different

Classes in C++

- Classes can have data members and/or methods (functions)
  - If a data member or function is *private*, it can only be accessed/called from within the class
  - If a data member or function is *public*, it can be accessed from outside of the class
    - It is bad practice for data members to be public
    - Not the case for methods – this will depend though
  - If a data member or function is *protected*, it can be accessed by classes that inherit from it
- Classes have two methods that must exist
  - The constructor: runs when the object is instantiated
    - Used to initialize data members
    - Has the same name as the class itself

Worked Example #2 – Using Classes in C++

Suppose we are writing a program involving particles, and that for each particle, we are interested in keeping track of its mass and momentum

For additional flexibility, we could create our own 'Particle' class in C++

- Need to make a 'header file' Particle.h and an implementation file Particle.cc

Header File - Particle.h

- Contains the declaration of the class
- Anytime we want to use this class in some program, we must include the header file in the program
- Tells the program what the Particle class is built of, and what it can do

The 'Particle' class consists of two *members*, -- variables for mass and momentum

- Like the struct we created earlier, but more powerful

A class can also contain functions (these are the *methods* of the class)

- For example, we can calculate the energy of a particle from its mass and momentum. So we could make that function be part of the class, say GetEnergy()
- Then if we created a particle object, we could get the energy of that particle within our program just by doing: *particle.GetEnergy()*

Worked Example #2 – Using Classes in C++

Suppose we are writing a program involving particles, and that for each particle, we are interested in keeping track of its mass and momentum

Let's try coding this!

Make your own header file (Particle.h) and implementation file (Particle.cc) for the "Particle" class

Should contain functions for getting the momentum of a particle and setting the mass and momentum of a particle.

- **GetMass()**
- **GetMomentum()**
- **SetMass()**
- **SetMomentum()**

Should also contain a function that can calculate the particle energy from the mass and momentum

- **GetEnergy()**

For your reference, I'll attach commented code solutions to these after the talk

- As an exercise, you could try extending what these can do (e.g. keep track of particle charge and x,y,z components of momentum and calculate the magnitude of the momentum etc.)

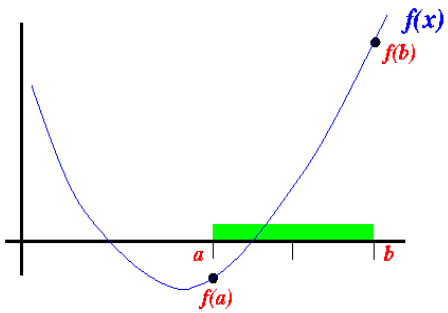
Worked Example #3 – "ROOT" Finding

The Bisection Method is a numerical method in Mathematics to find a root of a given function f(x)

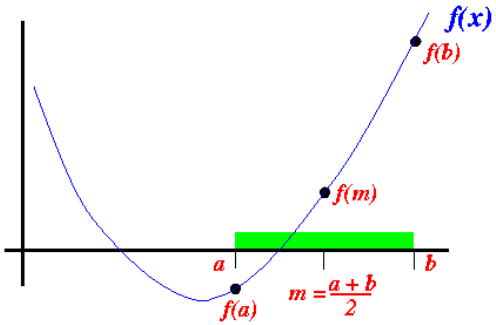
The Bisection Method is a successive approximation method that narrows down an interval that contains a root of the function f(x)

Given an initial interval [a,b] that contains a root:

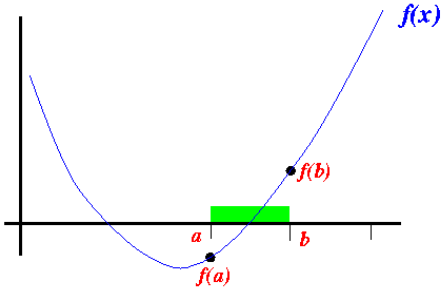
- **The Bisection Method will cut the interval into 2 halves and check which half interval contains a root of the function**
- **The Bisection Method will keep cut the interval in halves until the resulting interval is extremely small**
- **The root is then approximately equal to any value in the final (very small) interval**



We cut the interval [*a,b*] in the middle: *m = (a+b)/2*

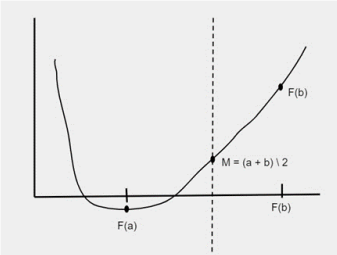


Because sign of *f(m)* ≠ sign of *f(a)* , we *proceed* with the search in the *new* interval [*a,b*]

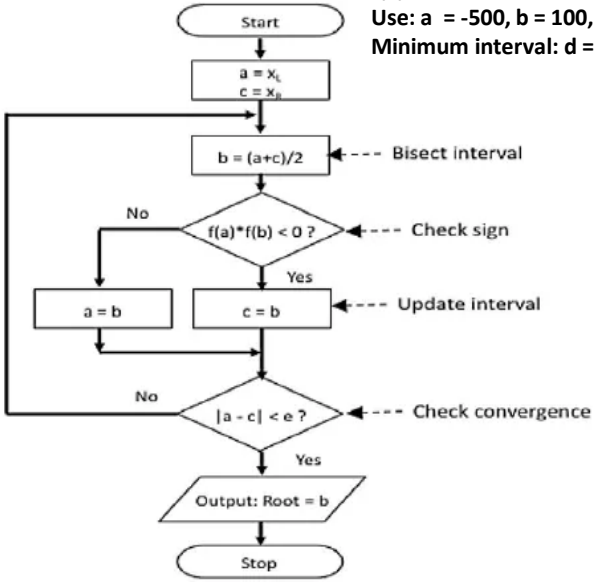


Worked Example #3 – "ROOT" Finding

Task: Numerically find the value of the root that lies between the interval [a,b] = [-500,100] in function: f(x) = x^3 -2x^2 +3 using the bisection method



f(x) = x^3 -2*x^2 +3
Use: a = -500, b = 100,
Minimum interval: d = 0.01



Some Suggested Skeleton Code:

```
#include <iostream>
using namespace std;
#define EP 0.01 // Define EPSILON (min interval) here

//Create function for f(x) here//
//:::::::::::::::::::::::::::://
//:::::::::::::::::::::::::::://

// Prints root of f(x) with error in EPSILON
void bisection(double a, double b) {

   //Should include a condition that checks a,b are opposite signs

   //Define mid-point

   //Loop until midpoint = root. Recal that (b-a) >= 0.01 (min_interval)

   //::::::::::::    Loop     ::::::::::::::::::::::://
    {
      // Find mid-point

      // Check if mid-point is root

      // If not, decide which side to repeat steps


    }
    //::::::::::::::::::::::::::::::::::::::::::::://

    // Print the Value of the root
    // And the # of iterations it took to get it
}
 // main function
int main() {
   double a =-500, b = 100;
   bisection(a, b);
   return 0;
}
```

Suggested Approach:

- Input the equation and the value of intervals a,b
- Divide the intervals as : m = (a + b) / 2
  - Print m is the root
- If f(m) ≠ 0
  - Check if f(a) * f(m) < 0
  - Then root will lie between a and m
  - Check if f(b) * f(m) < 0
  - Then root will lie between b and m

Answer: Should find a Root at = -0.991821 after 16 iterations of the bisection method

Now It's Your Turn!

- G.I. Joe, programmer

Try some of the following exercises yourself

- Solutions can/will be posted as needed
- They are only there if you want to practice more C++ coding (so you can ignore any exercises as you see fit)

Also look through some of the additional C++ resources I've listed

- Can also search for yourself online. There is no shortage of good references for C++

The real fun starts tomorrow with ROOT

Data Analysis Framework

I would recommend working through ROOT tutorials/problems over strictly C++ ones: Since most of you will be using ROOT specifically this summer

Self Directed 1 – Algorithms

Quadratic Formula

- **Write a program that calculates the roots of a quadratic (using the quadratic formula). Recall that there are three different cases depending on the value of the discriminant**

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- Depending on the value of the discriminant $b^2 - 4ac$ we have 3 cases:
  1. $b^2 - 4ac > 0$ two real values for x
  2. $b^2 - 4ac = 0$ one real value for x
  3. $b^2 - 4ac < 0$ no real values for x

Self Directed 2 – Arrays,Loops, I/O

Quick Sort

- Write a C++ program to sort a collection of integers using the Quick sort algorithm

Most Common Element in an Array

- Write a program that finds the most occurring entry in an array of integers

Diamonds are Forever

- Write a program in C++ to display a diamond pattern with asterisks of central width “n”

String Concatenation

- Write a program that, for a given string, will run the string concatenate method n times based on the value of n

Printing Times Tables

- Write a program in C++ to display the multiplication table vertically from 1 to n

```
 Display the multipliaction table vertically from 1 to n:
------------------------------------------------------------
Input the number multiplicated upto: 5
Multiplication table from 1 to 5
1x1=  12x1=  23x1=  34x1=  45x1=  5
1x2=  22x2=  43x2=  64x2=  85x2=  10
1x3=  32x3=  63x3=  94x3=  125x3=  15
1x4=  42x4=  83x4=  124x4=  165x4=  20
1x5=  52x5=  103x5=  154x5=  205x5=  25
1x6=  62x6=  123x6=  184x6=  245x6=  30
1x7=  72x7=  143x7=  214x7=  285x7=  35
1x8=  82x8=  163x8=  244x8=  325x8=  40
1x9=  92x9=  183x9=  274x9=  365x9=  45
1x10=  102x10=  203x10=  304x10=  405x10=  50
```

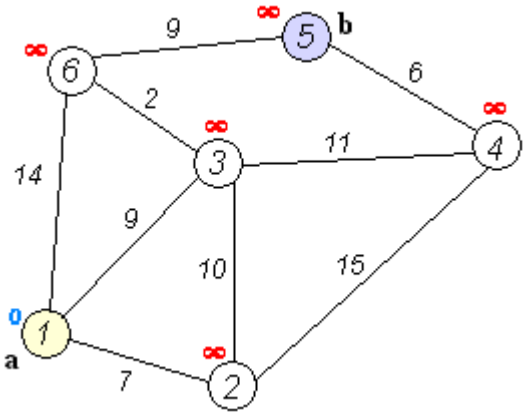
```
Input number of rows (half of the diamond): 5

    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

Self Directed 3 – Algorithms Cont.

Not necessary a super useful example for what we plan on doing with C++, but a pretty common/'standard' one done in intro C++ classes:

Coding Dijkstras's (Shortest Path First) Algorithm

- An algorithm for finding the shortest paths between nodes in a graph
- Creates a tree of shortest paths from the starting vertex, the source, to all other points in the graph
- Finds a shortest path tree from a single source node, by building a set of nodes that have minimum distance from the source
- Refer to this link for more detail
- https://www.educative.io/edpresso/how-to-implement-dijkstras-algorithm-in-cpp



Write a program that Performs Dijkstra's algorithm for the following input:

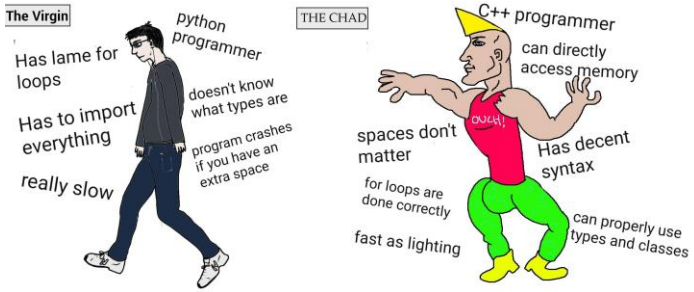
Input:

```
//Input//
G[max][max]={{0,1,0,3,10},
    {1,0,5,0,0},
    {0,5,0,2,1},
    {3,0,2,0,6},
    {10,0,1,6,0}}
n=5
u=0
```

Expected Output:

```
//Output//
Distance of node1=1
Path=1<-0
Distance of node2=5
Path=2<-3<-0
Distance of node3=3
Path=3<-0
Distance of node4=6
Path=4<-2<-3<-0
```

Some C++ Resources For More Exercises



http://www.whbell.net/resources/HepCppIntro/
Nice C++ guide, designed for HEP students in mind

https://en.wikibooks.org/wiki/C%2B%2B_Programming
eBook that goes over C++, some worked examples for beginners

https://www.w3resource.com/cpp-exercises/
Large repository of C++ example problems, particularly simple ones
(Good for beginners – I even pulled some of the self-directed problems from here)

http://www.stroustrup.com/Programming/Solutions/exercise_solutions.html
Some solutions to select problems, from the man who invented the language

http://www.cplusplus.com/doc/tutorial/
Not exercises, but a very readable tutorial, straight from the source

Again, this is not an exhaustive list – mostly from a surface level online search.
Plenty of other resources exist, don't be afraid to do a little digging yourself!