ECE 250 Data Structures and Algorithms

C++ Tutorial

Hany Samuel and Douglas Wilhelm Harder
Department of Electrical and Computer Engineering
University of Waterloo

The purpose of this quick tutorial is discussing some topics of C++. It is recommended to download the source code so you can track the programs. These programs were written using Dev C++ .

1- Write a C++ program that calculates the area of the circle.

The idea here is simple. We let the user enter the radius, then we calculate the area using the formula Area = ΠR^2

```
// this program is found in the problem_1.cpp
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
  /* this program caculate the circle area given the radius entered by the user
    *ECE 250
    *Hany Samuel*/
   double radius;
   const double PI=3.14;
   cout<<"please enter the radius \n";
                              // can be written as cout<<"please enter the radius"<<endl;
   cin>>radius;
   double area=PI*(radius*radius);
   cout<<"the area is = "<<area<<"\n";

   system("PAUSE");
   return EXIT_SUCCESS;
}
```

Notice:

- Like the C#, the program starting point is the ***main*** function. But unlike C# the ***main*** function is not inside a class. In C++ you can define functions outside any class.
- This program shows how to declare a variable (e.g. ***double radius***).
- This program shows how to declare and initialize a variable at the same time (e.g. ***double PI=3.14***).
- We use ***cin / cout*** for input / output operations. To be able to use it we should include ***iostream***.
- As we see due to the fact that ***PI*** value does not change, we define it as ***const***. so if you accidentally try to change its value the compiler will give you an error.
- System("PAUSE") is user to hold the console on until we press a key.
- Return EXIT_SUCCESS, as the main returns an integer indicates that the status of the program exit state. By returning this value we indicate that the program exited normally. Similar to return 0;

2- Write a program that prints all the numbers from 1 to 20 whose second and third bits are 1.
For example 4 is represented as 101, its second bit is 0 and third bit is 1 ; so it is not printed.
But 6 is represented as 110 so its second and third bit are 1, so it is printed.

To deal with the bits, it is better to use the bitwise operators like (& and, | or, ~ not, ^ xor)
To check the third and second bits are 1, we "and" the number with 6 and if and only if the result is 6 the number contains 1 in its second and the third bits.

```
// this program is found in the problem_2.cpp
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
  /* this program prints all the numbers from 1 to 20 that its second and third bits are 1.
    * ECE 250
    * Hany Samuel*/

   for(int i=1;i<20;i++)
   {
      if((i&6)==6)
         cout<<i<<endl;
    }
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

- Note that the ***int i*** is defined with in the ***for*** loop body, so if you try to access it from outside the ***for*** loop body you will get an error.
- The ***if*** statement contains only on line; so we were able to get ride of the {} for if. But it is safer to use them.
- We have to include the expression i&6 inside ( ), so it is evaluated before the ==. Try to remove the ( ) and run the program you will find it gives wrong results.
  As it will evaluate 6==6 first and this return 1 (as it is true) then 1 is anded with i that gives nonzero integer which is considered true also, so you will find the output is simply all the numbers. C++ not like C# , it considers the bool type as integer so you must take care.

3- Write a program that finds all the numbers, dividable by 3 or 4,ranges from 1 to n.

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

int main(int argc, char *argv[])
{
  /* this program to print all the numbers that are dividable by 3 or 4
   *  ranges  from 1 to n
   * ECE 250
   * Hany Samuel            */
   int n;
   cout<<"please enter n \n";
   cin>>n;
   for(int i=1;i<=n;i++)
   {
    if( i%3==0 || i%4==0)
        cout<<i<<"\n";
          }

   system("PAUSE");
   return EXIT_SUCCESS;
}
```

4- Write a C++ program to Calculate n!=n(n-1)(n-2)..(2)(1).

We first implement a function for caculating the factorial of a number.

```
#include <iostream>
  int factorial(int);

using namespace std;

main(int argc, char *argv[])
{

   /* this program to calculate factorial n (n!)
    * ECE 250
   * Hany Samuel           */
int n;
 cout<<"please enter n \n";
 cin>>n;
 if(n<0)  // we can not calculate the factorial of negative numbers
{
cerr<<" can not calculate the factorial of a negative number"<<endl;
system("PAUSE");
return 0; // we end the program, we can use exit ; instead
}
  int result=factorial(n);
   cout<<result<<"\n";
   system("PAUSE");
   return 0;
}
int factorial(int x)
{
    int fact=1;
    for(int i=1;i<=x;i++)
     {
        fact=fact*i;  // can be written fact *=i;
     }
     return fact;
}
```

If you have noticed we added a line after #include that declare the factorial function. We have to do that as the factorial function is defined after the main function.

A better technique is to define the factorial function in a header file like fact.h and we add another include statement # include “fact.h” (see the attached source code).
You also should note that we said #include “fact.h” not <fact.h> and that is because it is our defined header (you usually put it in the same project directory) file not a standard header file.
Also note that ***cerr*** is similar to cout , but we use it as the standard error output stream(for example the error may be recorded in a log file not the screen).

Now we will write the factorial function in another way (see source code files problem_4_recursion.cpp)

```
int factorial(int x)
{
   if(x==0||x==1) return 1;  // as factorial 1 and 0 is equal to 1
   return x*factorial(x-1);
 }
```

Note that this function must start with the if condition, can you figure why ? yes if you change the order of the previous 2 lines the program will enter infinite loop of calling the function factorial until a stack overflow occur.

5- Write C++ program that read 10 numbers and print their sum, average, maximum and minimum.

We first read the numbers (of course through a for loop) into an array.
We take the first number and consider it the minimum. We compare the minimum with all the other numbers, if we find a number less than the minimum this number will become our new minimum.
Similarly for the maximum.

```
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
    /* this program to 10 numbers and print their sum, average, maximum and
      *minimum.
      * ECE 250
      * Hany Samuel           */
   const int n=10;
    int numbers[n];
   for(int i=0;i<n;i++)
    cin>>numbers[i];
    int min;
    int max;
    int sum;
    float average;
min=max=sum=numbers[0];
// the minimum equals the maximum equals the summation equals the first number at
//the start

for(int i=1;i<n;i++)  // we  consider starting from the next element in the array
    {
     if(numbers[i]<min)
            min=numbers[i];
      if(numbers[i]>max)
           max=numbers[i];
     sum+=numbers[i];
     }
     average=((float) sum)/n;
     cout<<"sum =" <<sum<<" , average= "<< average<<"\n";
     cout<<" maximum =" <<max<<" , minimum= "<<min;
         system("PAUSE");
   return EXIT_SUCCESS;
}
```

Note that as sum is int and n is int, when we divide them we get integer too.

For example if we sum the number from 1 to 10, the sum is 55 and the average should be 5.5 . But if you divides the integers directly you will get 5 not 5.5
To solve that we ***cast*** one of the variables to float (or double) just during the division so we get a float result.

average=((float) sum)/n;

note that we can not write **average=(float) sum/n;** because here the division will happen before the cast and we will still get the wrong result.
So be always at the safe side and use parentheses as long as you are not sure of the order of the expression execution.

Now if we want to define a dynamic array. That changes at the runtime. It is better to use pointers. The previous program will change as follows

```
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
        /* this program to calculate factorial n (n!)
         * ECE 250
        * Hany Samuel           */
  int n;
    cout<<"enter the numbers count \n";
    cin>>n;
    int* numbers=new int[n];
    cout<<"start entering the numbers"<<endl;
    for(int i=0;i<n;i++)
    cin>>*(numbers+i);
     int min;
    int max;
    int sum;
    float average;
    min=max=sum=*(numbers+0);
for(int i=1;i<n;i++)
    {
      if(*(numbers+i)<min) min=*(numbers+i);
      if(*(numbers+i)>max) max=*(numbers+i);
      sum+=*(numbers+i);
     }
     average=((float)sum)/n;
     cout<<"sum =" <<sum<<" , average= "<< average<<"\n";
     cout<<" maximum =" <<max<<" , minimum= "<<min;
     delete [ ] numbers;
    system("PAUSE");
    return 0;}
```

Note that numbers[i] is equivalent to *(numbers+i). Also note that the space allocated using the new operators must be manually de-allocated using the delete operator.

6- Write a C++ program to check if the number n is prime or not .
The prime number has two divisors itself and 1 only. So we made a program that check the number with all the numbers less than it (except 1).

```
#include <iostream>
using namespace std;
main(int argc, char *argv[])
{
int n;
 cout<<"please enter n \n";
 cin>>n;
 int absn=n;
 if(n<0)  // we deal the positive numbers
{
        absn=-n;
}
  bool prime=true; // we assume it is prime untill we proof it is not

  for(int i=2;i<absn;i++)
  {
    if(n%i==0) prime=false;
        }
  if(absn==2) prime=true;
  if(prime) cout<<n<<" is prime"<<endl;
  else      cout<<n<<" is not prime"<<endl;
   system("PAUSE");
   return 0;
}
```

Actually this program is not so intelligent. As you may noticed we checked all the numbers till n-1 !!. we could instead check it until n/2 or $\sqrt{n}$.

7- Write a C++ program that prints all the prime numbers between 1 and n.
We define a function isPrime(n) that check if n is prime or not (using problem 6)

```
#include <iostream>
  bool isPrime(int);
using namespace std;
main(int argc, char *argv[])
{
int n;
 cout<<"please enter n \n";
 cin>>n;
 for(int i=2;i<=n;i++)
 {
   if(isPrime(i)) cout<<i<<endl;
         }
   system("PAUSE");
   return 0;
}

bool isPrime(int n)
{
 int absn=n;
 if(n<0)  // we deal the positive numbers
{
         absn=-n;
}
if(absn==2) return true;
  bool prime=true; // we assume it is prime untill we proof it is not
  for(int i=2;i<=absn/2;i++)
  {
     if(n%i==0) prime=false;
          }
  if(prime) return true;
  else      return false;

     }
```

8- This program just investigates the difference of passing parameters by value, pointer, and reference

```
#include <iostream>
 void increment(int,int*, int &);
using namespace std;
main(int argc, char *argv[])
{
int x=5;
int y=6;
int z=7;
increment(x,&y,z);
cout<<"x= "<<x<<", y= "<<y<<", z= "<<z;

  system("PAUSE");
  return 0;
}
void increment(int x, int* y, int & z)
{
    x++;
    (*y)++;
    z++;

    }
```

When you run the program you will find y and z changed (incremented) but not x. Because passing by reference is equivalent to passing the variable itself (not a copy of it as in case of x), and of course passing a pointer enable us to access the main variable through its address.

9- This program demonstrates some topics in classes. We define the complex class

```
/* this is comp.h*/
class Complex
{
     private:
           double real;
           double img;
     public:
     Complex(double r):real(r),img(0)
           { }
    Complex(double r, double i):real(r),img(i)
           { }
     double getReal() const
     {   return real;  }

     double getimg() const
     {   return img;  }
     void setReal(double r)
     { real=r;}
     void setImg(double i)
     { img=i;}
     Complex add(Complex x)
     {
        double r = real + x.real;
        double i = img + x.img;
        return Complex(r,i);
      }
      bool operator ==(Complex x)
      {
          return (x.real==real && x.real==real);
          }
   friend Complex external_add(Complex,Complex);
   friend void show(Complex );
   friend Complex operator +(Complex,Complex);
};
Complex operator + (Complex x, Complex y)
{
   double r=x.real+y.real;
   double i=x.img+y.img;
   return Complex(r,i);
      }

Complex external_add(Complex x, Complex y=Complex(0,0))
// here we use default argument value for y
```

```
{
 /* is similar to the add method . but this time it is external
 *that it is not part of an object , so we pass two objects to add this time
 */
 double r=x.real+y.real;
 double i=x.img+y.img;
 return Complex(r,i);
      }


/ * this is problem_9.cpp*/
#include <iostream>
#include "Comp.h"


using namespace std;

void show (Complex x)
{
   cout<<x.real<<" + j "<< x.getimg()<<endl;
   /*note that show as a friend function can access
   *the private data  real directly and also the public as getimg
   */
    }


main(int argc, char *argv[])
{
     Complex x(3);
     Complex y(4,5);
     Complex z=y.add(x); // here we use the member function for addition
     Complex w=external_add(x,y);// here we use the external function
     Complex m=external_add(x); // here y will have its default value of 0,0
     show(z);
     show(w);
     show(m);
     if(z==w) cout<<"equal"<<endl; // here we use the overloaded == operator
     Complex a=z+w;              // here we use the overloaded + operator
     show(a);

   system("PAUSE");
  return 0;
}
```

10- Assume we want a generic add function that adds whatever passed to it even if it is the complex that we used in the previous example.
Obviously we must use templates

```
#include <iostream>
#include "Comp.h"
using namespace std;
void show (Complex x)
{
   cout<<x.real<<" + j "<< x.getimg()<<endl;
   /*note that show as a friend function can access
   *the private data  real directly and also the public as getimg
   */
    }
template<typename object> object addanything( object x, object y)
{
 return x+y;
}

main(int argc, char *argv[])
{
     Complex x(3,1);
     Complex y(4,5);
     Complex z=addanything<Complex>(x,y);// in this case the function use the
overloaded operator
     show(z);
     int d=addanything<int>(3,5);
     cout<<"now we added integers 3+5= " << d<<endl;
   system("PAUSE");
  return 0;
}
```