Algorithms

Tutorial for C++, Compiler and Makefile Example

0. Objectives

This small example shows you what a C++ project look like. We will sort array by insertion sort and STL. Although you do not have to submit Tutorial C++, you are strongly recommended to run this example on EDA union lab machines before you start working on your PA.

1. Insertion Sort

Please try our first program, the insertion sort. Please go to the insertion sort directory by typing

```
cd TutorialC++/insertionsort
```

Use your texteditor to open the file *insertionsort.cpp*

```
/*
Program: Insertion Sort
Author Name:
Usage: insertionsort
Revision:  V.2010.2.1
*/

#include <iostream>
using namespace std;
#define ELEMENTS 6

void insertion_sort(int x[],int length)//define function
{
  int key,i;
  for(int j=1;j<length;j++)
  {
     key=x[j];
     i=j-1;
     while(x[i]>key && i>=0)
     {
        x[i+1]=x[i];
        i--;
     }
     x[i+1]=key;
  }
}

int main()
{
  int A[ELEMENTS]={5,2,4,6,1,3};//initial array
  int x;

  cout<<"NON SORTED LIST:"<<endl;
  for(x=0;x<ELEMENTS;x++)
  {
      cout<<A[x]<<endl; //display the initial array
  }
  insertion_sort(A,ELEMENTS);//call insertion_sort function
  cout<<endl<<"SORTED LIST"<<endl;
  for(x=0;x<ELEMENTS;x++)
  {
      cout<<A[x]<<endl; //display the sorted array
  }
  return 0;
}
```

to compile it, type

```
g++ insertionsort.cpp -o insertionsort
```

or

```
make insertionsort
```

to execute, type

```
./insertionsort
```

, where './ ' means this directory. Please note that in Linux, you need to specify the correct path of files. If you simply type:

```
insertionsort
```

Linux will complain that it cannot find the file. This is a mistake often made by beginners.

> NOTE:
> If you are currently in the ~/my/TutorialC++/ directory, then
> ~/ means my home directory
> ./ means this current path ~/my/TutorialC++/
> ../ means the upper directory ~/my/
> ./test/ means the lower directory ~/my/TutorialC++/test

2. Using Sort Function in STL

C++ STL is a *standard template library*, which contains useful containers, adaptors, iterators, function objects and algorithms. Now we show a simple example using the STL function '*sort*'.

```
cd ../STL
```

open the *stlsort.cpp* file and check the difference. Note that two lines are changed from the last example.

```
#include <iostream>
#include <algorithm>  // This line is different
using namespace std;
#define ELEMENTS 6

int main()
{
  int A[ELEMENTS]={5,2,4,6,1,3};//initial array
  int x;
  cout<<"NON SORTED LIST:"<<endl;
  for(x=0;x<ELEMENTS;x++)
  {
       cout<<A[x]<<endl;//display the initial array
  }
  sort (A, A+ELEMENTS);  // This line is different
  cout<<endl<<"SORTED LIST"<<endl;
  for(x=0;x<ELEMENTS;x++)
  {
       cout<<A[x]<<endl;//display the sorted array
  }
  return 0;
}
```

then compile and execute.

```
g++ stlsort.cpp –o stlsort
```

or

```
make stlsort
```

to execute, type

```
./stlsort
```

3. Makefile

In a large project, source codes are separated into many files. For example, we may decide to have different sorters, each of which is an independent file. Please change the directory,

```
cd ../makefile_demo
```

Now *main.cpp* is like this. Please note that the *replaceable_sorter*() is no longer defined here in *main.cpp*. To make sure the compilation is successful, we must include a new header file *replaceable_sorter.h*. To compile this *main.cpp*

```
g++ -c main.cpp -o main.o
```

```
#include <iostream>
using namespace std;
#include "replaceable_sorter.h"  // add this line
#define ELEMENTS 6

int main()
{
  int A[ELEMENTS]={5,2,4,6,1,3};//initial array
  int x;

 ...
  replaceable_sorter(A,ELEMENTS);  // change this line
...

  return 0;
}
```

You will see a *main.o* object file generated. Please note that the –c option tells the compiler to generate an object file only — no linking is done for now.

The *replaceable_sorter.h* file just provides the forward declaration of the function.

```
void replaceable_sorter(int x[],int length);
```

Suppose we have two implementations of the replaceable_sorter: one is insertionsort and the other is the stlsort. You can check the *stlsorter.cpp* file to see the details. To compile the stlsort, type

```
g++ -c stlsort.cpp -o stlsort.o
```

```
#include <algorithm>
using namespace std;
void replaceable_sorter(int x[], int length)
{
  sort (x, x+length);
}
```

Now, we can link the object files to produce an executable file *demo_stl*.

```
g++ main.o stlsort.o -o demo_stl
```

The other implementation can also be compiled in the same way.

```
g++ -c insertionsort.cpp -o insertionsort.o
g++ main.o insertionsort.o -o demo_is
```

As you can see, this process is very long and tedious. So we can write all this compilation instructions into a *makefile* like the following.

```
# CC and CFLAGS are variables
CC=g++
CFLAGS =  -c
# -c option ask g++ to compile the source files, but do not link.


all      : demo_stl  demo_is

demo_stl : main.o stlsort.o
             $(CC)  main.o stlsort.o -o demo_stl

demo_is      : main.o insertionsort.o
             $(CC)  main.o insertionsort.o -o demo_is

main.o          : main.cpp replaceable_sorter.h
             $(CC) $(CFLAGS) main.cpp

stlsort.o      : stlsort.cpp
             $(CC) $(CFLAGS) stlsort.cpp

insertionsort.o : insertionsort.cpp
             $(CC) $(CFLAGS) insertionsort.cpp

# clean all the .o and executable files
clean:
         rm -rf *.o demo_is demo_stl
```

CC and CFLAGS are variables that will be used in the following text. The structure of makefile is simply '**target: source /n command**'. For example, the two lines in bold means to compile the *stlsort.o*, we need *stlsort.cpp*. And the command is g++ -c *stlsort.cpp*. The makefile can also be written in a more concise way by neglecting some common commands and variables that will be automatically generated with system default behaviors. An example is makefile_2.

To compile *stlsort.o*, please type

```
make stlsort.o
```

To compile and link *demo_stl*, please type

```
make demo_stl
```

If you want to remove all the .o and executable file, simply type

```
make clean
```

Actually, you can compile both *demo_stl* and *demo_is* in just one step.

```
make
```

4. File IO

Now we learn how to read/write files in c++.

```
cd ../fileIO/
```

Use your texteditor to open *fileIO.cpp*. This file contains two functions.

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

int Max (int a, int b)
{
       if (a> b) return a;
       else return b;
}
```

```
int main ()
{
	ifstream inFile("test.in");
	ofstream outFile("test.out");
	int a,b;
	inFile >> a;
	inFile >> b;
	outFile << Max(a, b) << endl;
	outFile.close();
	inFile.close();
	return 0;
}
```

Also open the file *test.in* and you will see two numbers.

To compile it, type this command,

```
    g++ fileIO.cpp -o fileIO
```

or

```
    make fileIO
```

To execute, type this command

```
    ./fileIO
```

Check the results in *test.out*.

```
    cat test.out
```

5. To learn more….

1. Makefile
   https://mropengate.blogspot.com/2018/01/makefile.html

2. STL:Vector
   http://www.cplusplus.com/reference/vector/vector/
   http://www.runoob.com/w3cnote/cpp-vector-container-analysis.html

3. Class
   https://openhome.cc/Gossip/CppGossip/ClassABC.html
   https://mropengate.blogspot.com/2018/01/makefile.html

4. Call by value, address, reference
   https://ppt.cc/fkwgdx
   https://ppt.cc/fFg9bx

5. PA0!!