AST21105 Object-Oriented Programming & Design

Lab 6 – Separate Compilation

A. Submission Details

In this lab, you are required to submit **ONE** C++ program with **ELEVEN** files to solve the given problem shown in the section "Mark System". To make the program implementation easier, you are suggested to write your program using Visual Studio .NET instead of doing it directly using paper & pencil. After you have completed the implementation, submit your program files (i.e. DataEntry.h, DataEntry.cpp, Engine.h, Engine.cpp, Histogram.h, Histogram.cpp, Search.h, Search.cpp, Sort.h, Sort.cpp and main.cpp for this lab, zip the files and name the zip file with your student ID, e.g. 54321234.zip.) using the electronic "drop-box" in Canvas, **One week** after your lab section is conducted. For details, please refer to the following:

Tuesday sections	*by*	*19:00 on 1 March 2016 (Tuesday)*
Wednesday sections	*by*	*19:00 on 2 March 2016 (Wednesday)*

You are reminded to double-check your solution to verify everything is correct before submission. You are also required to put your name and student ID at the beginning of your source file.

Important: You are only able to submit your work once.

B. Objective

The objective of this lab is to help you get familiar with separate compilation that you have learned in the last lecture. The first part of this lab is meant to be a simple review of separate compilation. While the second part of this lab is a practical task that allows you to use all the techniques introduced.

C. Review

1. Why implementing a program in separate files?

   i) Easily re-use the functions in another program (i.e. application).
   ii) Can speed up compilation (only need to re-compile for those files with code modification).
   iii) Can facilitate team programming.
   iv) Can keep the details of function implementation secrets.

2. How to break down program code into separate files?

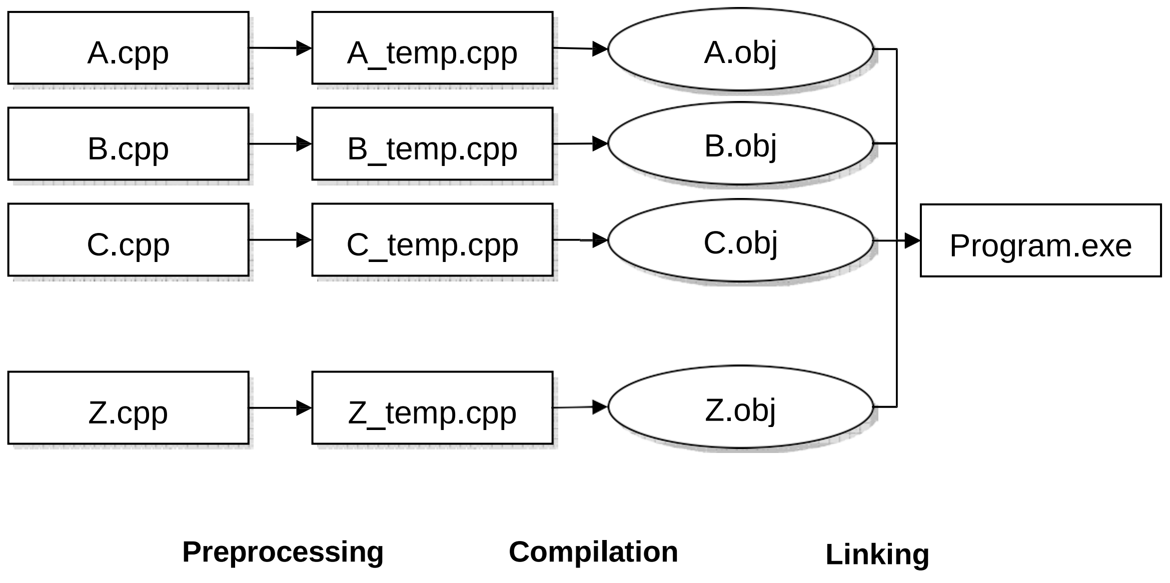
   There is no unique way to do this. Since this is actually very much depending on the programming design. But a common way of doing this would be grouping functions with similar functionalities together in a separate file.

   Moreover, we usually separate the **function declaration** and **function definition** in two files.
   i) Function declarations are written in a header file (.h file)
   ii) Function definitions are written in a source file (.cpp file)

3. Why separate function declaration and function definition into .h and .cpp file?

   i) We normally only give header file to other programmers so as to allow them to know what functions are available to call.
   ii) Other programmers can read the .h file to know what functions are provided, instead of reading the .cpp to know how functions are implemented. This is not what most programmers are interested in.

4. What is the complete procedure for generating an executable file from multiple files?



5. Preprocessing directives

There are a lot of preprocessor directives in C++. In this course, we will only cover the following preprocessor directives:

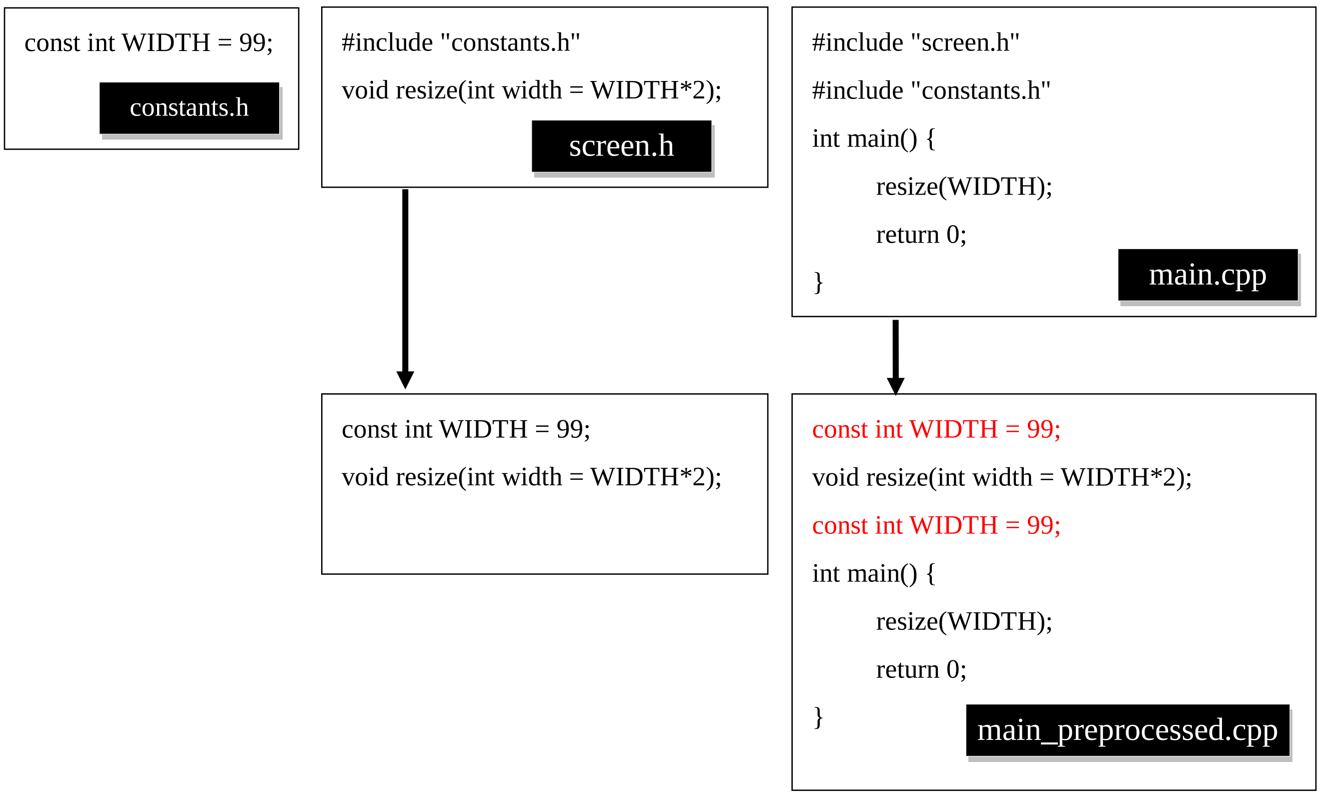
(1) #include

(2) #ifndef

(3) #define

(4) #endif

Recall, #include is used to copy the content of the header file to replace the line of #include. Note: This may introduce compilation error when a header file includes another header file.



constants.h
```
const int WIDTH = 99;
```

screen.h
```
#include "constants.h"
void resize(int width = WIDTH*2);
```

main.cpp
```
#include "screen.h"
#include "constants.h"
int main() {
        resize(WIDTH);
        return 0;
}
```

```
const int WIDTH = 99;
void resize(int width = WIDTH*2);
```

main_preprocessed.cpp
```
const int WIDTH = 99;
void resize(int width = WIDTH*2);
const int WIDTH = 99;
int main() {
        resize(WIDTH);
        return 0;
}
```

From this example, we can see that the statement: "const int WIDTH = 99;" is included **TWICE**, which causes compilation error.

1. To resolve the problem we can use #ifndef (if not defined), #define, and #endif in header files.
2. #define can be used to define some symbols in preprocessor.
3. The code between #ifndef and #endif will be discarded if the symbol after #ifndef is defined.
4. We can use #ifndef, #define, and #endif to enclose the content of header files.

```
#ifndef CONSTANT_H
#define CONSTANT_H
const int WIDTH = 99;
#endif
```
constants.h

```
#ifndef CONSTANT_H
#define CONSTANT_H
#include "constants.h"
void resize(int width = WIDTH*2);
#endif
```
screen.h

```
#include "screen.h"
#include "constants.h"
int main() {
        resize(WIDTH);
        return 0;
}
```
main.cpp

D. Mark System

In this part, you are required to:

i) Create a New Project and give your project the name Lab6.

ii) In this lab, you are required to break up the given program "MarkSystem.cpp" into a number of header and source files. Suggested break up would be the following header and source files:

• DataEntry.h	[Consists of function prototype of performDataEntry()]
• DataEntry.cpp	[Consists of function definition of performDataEntry()]
• Engine.h	[Consists of function prototypes: performSortMarks(), performSearchMark(), and performComputation()]
• Engine.cpp	[Consists of function definitions: performSortMarks(), performSearchMark(), and performComputation()]
• Histogram.h	[Consists of function prototype of performHistogram()]
• Histogram.cpp	[Consists of function definition of performHistogram()]
• Search.h	[Consists of function prototype of performBinarySearch()]
• Search.cpp	[Consists of function definition of performBinarySearch ()]
• Sort.h	[Consists of function prototype of performQuickSort()]
• Sort.cpp	[Consists of function definition of performQuickSort ()]
• main.cpp	[Consists of functions: executeMenu(), selectMenu(), main()]

iii) More details about the given program:

The given program is a small marking system, which should be able to perform the following tasks:

a) Read n marks from keyboard, where n is a user-specified value.
b) Sort the input mark in ascending order.
c) Search the location of a particular mark in the array
d) Compute the statistic of marks (e.g. mean and unbiased variance)
e) Plot the histogram with a user-defined number of bins.

At startup, the system should clear the screen and show a text-based menu:

```
Select function:
--------------------
1.) Mark Entry
2.) Sort Marks
3.) Search Mark
4.) Compute Statistics
5.) Plot Histogram
6.) Quit

Please enter your choice:
```

After reading the selection, the system should perform the corresponding action, then pause and prompt the user to press any key to continue by showing:

```
Press any key to continue
```

Example output should look like this:

```
Select function:
--------------------
1.) Mark Entry
2.) Sort Marks
3.) Search Mark
4.) Compute Statistics
5.) Plot Histogram
6.) Quit

Please enter your choice: 1
```

```
--------------------
Input
--------------------
How many marks to store? 10
Input marks: 87 65 20 37 92 80 90 100 12 45
```

```
Select function:
--------------------
1.) Mark Entry
2.) Sort Marks
3.) Search Mark
4.) Compute Statistics
5.) Plot Histogram
6.) Quit

Please enter your choice: 2
```

```
-------------
   Sorting
-------------
Marks sorted in ascending order: 12 20 37 45 65 80 87 90 92 100
Press any key to continue ...
```

```
--------------
  Searching
--------------
Mark to find (-1 to quit)? 65
The mark 65 is found at index 4
Mark to find (-1 to quit)? 99
The mark is not found
Mark to find (-1 to quit)? -1
```

```
Select function:
--------------------
1.) Mark Entry
2.) Sort Marks
3.) Search Mark
4.) Compute Statistics
5.) Plot Histogram
6.) Quit

Please enter your choice: 4
```

```
10 20 37 45 65 80 87 90 92 100
Mean : 62.8
Unbiased variance : 1028.62
```

```
Select function:
--------------------
1.) Mark Entry
2.) Sort Marks
3.) Search Mark
4.) Compute Statistics
5.) Plot Histogram
6.) Quit

Please enter your choice: 5
```

```
How many bins you want to use to group the data? 10
min = 12
max = 100
[12, 20.8] : **
[20.8, 29.6) :
[29.6, 38.4) : *
[38.4, 47.2) : *
[47.2, 56) :
[56, 64.8) :
[64.8, 73.6) : *
[73.6, 82.4) : *
[82.4, 91.2) : **
[91.2, 100) : **
Press any key to continue ...
```

```
Select function:
--------------------
1.) Mark Entry
2.) Sort Marks
3.) Search Mark
4.) Compute Statistics
5.) Plot Histogram
6.) Quit

Please enter your choice: 6
```

Marking Scheme:

Graded items	Weighting
1. Correctness of program (i.e. whether your code is implemented in a way according to the requirements as specified.)	60%
2. Indentation	30%
3. Documentation (with reasonable amount of comments embedded in the code to enhance the readability.)	10%
	100%

Program Submission Checklist

Before submitting your work, please check the following items to see you have done a decent job.

Items to be checked	☑ / ☒
1. Did I put my name and student ID at the beginning of all the source files?	☐
2. Did I put reasonable amount of comments to describe my program?	☐
3. Are they all in .cpp extension and named according to the specification?	☐
4. Have I checked that all the submitted code are compliable and run without any errors?	☐
5. Did I zip my source files using Winzip / zip provided by Microsoft Windows? Also, did I check the zip file and see if it could be opened? ***(Only applicable if the work has to be submitted in zip format.)***	☐
6. Did I submit my lab assignment to Canvas?	☐

-End-