ENGG1340 Computer Programming II / COMP2113 Programming Technologies
Assignment 2
Deadline: 14 April (Friday) 23:59

If you have any questions, please post to the Moodle discussion forum on Assignment 2.

Total marks: 100 marks

General Instructions

Read the instructions in this document carefully.

In this assignment you will solve 3 tasks and a tester would automatically test your submitted program. So if your submitted files and program outputs do not conform to our instructions given here, your programs cannot be evaluated and you will risk losing marks totally.

Sample test cases are provided with each task in this document. Note that the test cases may or may not cover all boundary cases for the problem. It is also part of the assessment whether you are able to design proper test cases to verify the correctness of your program. We will also use additional test cases when marking your assignment submission.

Input and output format

Your C++ programs should read from the **standard input**. Also, your answer should be printed through the **standard output**. If you failed to follow this guide, the tester may not able to give a score for your program. Additionally, you should strictly follow the sample output format (including space, line breaker, etc.), otherwise, your answer might be considered as wrong.

How to use the sample test cases for problems 1 and 3?

Sample test cases in text file formats are made available for you to check against your work. Here's how you may use the sample test cases. Take problem 1 test case 2 as an example. The sample input and the expected output are given in the files input1_2.txt and output1_2.txt, respectively. Suppose that your program is named "1", do the followings at the command prompt of the terminal to check if there is any difference between your output and the expected output.

```
./1 < input1_2.txt > myoutput.txt
diff myoutput.txt output1_2.txt
```

Testing against the sample test cases is important to avoid making formatting mistakes. The additional test cases for grading your work will be of the same formats as the sample test cases.

Coding environment

You must make sure that your C++ program can compile, execute and generate the required outputs on our standard environment, namely, the gcc C++11 environment we have on the CS Linux servers (academy*). Make sure the following compilation command is used to compile your programs:

```
g++ -pedantic-errors -std=c++11 [program_name].cpp -o [object_code_name]
```

The name after the "-o" option is the name of the object code. In the above example, you can run the program using the following command:

```
./[object_code_name]
```

As a programmer/developer, you should always ensure that your code can work perfectly as expected on a target (e.g., your client's) environment, not only on yours.

While you may develop your work on your own environment, you should always try your program (compile & execute & check results) on our standard environment before submission.

Submission

Name your C++ program files (and Makefile for Problem 3) as the following table shows and put them together into one directory. Make sure that the folder contains only these source files (*.cpp / *h / Makefile) and no other files. **Compress this directory as [uid].zip file where [uid] is your university number** and check carefully that the correct file have been submitted. We suggest you to download your submitted file from Moodle, extract them, and check for correctness. **You will risk receiving 0 marks for this assignment if you submit incorrect files.** Resubmission after the deadline is not allowed.

Filename	**Description**
1.cpp	Problem 1
searchword.cpp	Problem 2
d2boh.cpp, main3.cpp, Makefile	Problem 3

Late submission

If you submit within 3 days after the deadline, there will be 50% mark deduction. After that, no mark will be given.

Evaluation

Your code will be auto-graded for technical correctness. In principle, we use test cases to benchmark your solution, and you may get zero marks for not being able to pass any of the test cases. Normally partial credits will not be given for incomplete solution, as in many cases the logic of the programs are not complete and an objective assessment could be difficult. However, your work may still be considered on a case-by-case basis during the rebuttal stage.

Academic dishonesty

We will be checking your code against other submissions in the class and from the Internet for logical redundancy. Please be reminded that no matter whether it is providing your work to others, assisting others to copy, or copying others will all be considered as committing plagiarism and we will follow the departmental policy to handle such cases. Please refer to the course information notes for details.

Getting help

You are not alone! If you find yourself stuck on something, post your questions to the course forum, send us emails or come to our support sessions. We want this assignment to be rewarding and instructional, not frustrating and demoralizing. But we don't know when or how to help unless you ask.

Discussion forum

Please be careful not to post spoilers. Please don't post any code that is directly related to the assignments. However, you are welcome and encouraged to discuss general ideas on the discussion forums. If you have any questions about this assignment, you should post them in the discussion forums.

Problem 1 [30%]:

A playing card consists of a suit (“Heart”, “Diamond”, “Club” or “Spade”) and a number (1 to 13). For simplicity, let’s use characters ‘H’, ‘D’, ‘C’ and ‘S’ to represent suits “Heart”, “Diamond”, “Club” and “Spade” respectively. So, a playing card is represented by one character combined with a number such as “H2”, “C6” and “D12”. Two cards are said to be a pair when they have the same number. Please write a C++ program to read an integer `N` and then `N` playing cards afterwards. The program then outputs the number of pairs among the `N` playing cards. For example, the sets {“H3”, “D3”} and {“H3”, “D3”, “C3”} contain 1 pair each. The sets {“H3”, “D3”, “C3”, “S3”} and {“H3”, “D3”, “C6”, “S6”} contain 2 pairs each. You can assume that the input playing cards are always valid.

Sample Test Cases

User inputs are shown in blue.

1_1:

```
8 H5 D6 C5 S6 C13 H10 D5 C6
2
```

1_2:

```
9 H2 H1 D2 D1 C2 C1 S2 S5 D5
4
```

1_3:

```
10 D6 D9 H9 C9 S12 S6 H6 C6 H12 S9
5
```

Problem 2 [35%]:

Write a C++ program that searches a word from multiple files, and output the number of lines that the word appears and the total number of occurrences of the word in each file.
You are provided with a header file `searchword.h` and a main program `main2.cpp`. Please complete the implementation file `searchword.cpp`.

Input:

- The main program **accepts command line argument as input**. The command line of the program is given by: `<program_name> <word_to_search> <file1> <file2> <file3>`
  E.g., if you compile your program using `g++ -pedantic-errors -std=c++11 searchword.cpp main2.cpp -o main2`, then the following command at the command prompt
  `./main2 abc t1.txt t2.txt t3.txt`
  will ask your object program `main2` to search for the string `abc` in the files `t1.txt`, `t2.txt`, `t3.txt`.
- Command line arguments in C/C++ are implemented by having input parameters for the `main()` function: `int main(int argc, char* argv[])`
- Read the comments in the provided template to see the usage of the paramaters `argc` (an integer) and `argv` (an array of C-strings).
- The `main2.cpp` program has been completed for you, which means that command line argument parsing has been done for you. But you should read the code and understand how it works, as you may need to deal with command line arguments in your later courses too.

Output:

- A line for each file specified in the command line arguments: first the filename, then the number of lines the word appears in the file (`n1`), followed by the number of total occurrences of the word in the file (`n2`):
  `<filename>: <n1> <n2>`
  or if there is any error accessing the file:
  `<filename>: error opening file`

Requirements:

- You should start working on `searchword.cpp` and complete the function
  `int SearchWord(string word, string fileName, int &nLines, int &total)`
  which search for a `word` in a file named `fileName`, and stores the number of lines that `word` appears in the file in `nLines` and the number of total occurrences in `total`. The function returns `0` if file operation is successful and `1` if otherwise.
- The matching of word is **case-insensitive**.

Note:

- The header file `searchword.h` and the main program `main2.cpp` have been completed for you, and you do not need to make any change to them. Read the code in them and understand what they do before you start working.
- We may use another `main2.cpp` to test your program.

Sample Test Cases

Sample text files `t1.txt`, `t2.txt`, `t3.txt`, `t4.txt` are provided.
We show only the contents of `t1.txt`, `t2.txt`, `t3.txt` here:

t1.txt	t2.txt	t3.txt

`Cab AbB CAB Ab CaB ABc aBb AB`
`cAb caB aBcAB CAB ABcAb AbcAb`
`aBCAB AB ABcAb aBCaB ABCAB Abb`
`aBC abC AbC ABCAB abCab ABC`
`aBCAB ABcAB AB cAb`
	`ab abcab Ab abc aBCAB AbCAb abC`
	`AbC ab abcab aBc aBc ab abcAB`
	`Abc aBC ab abC ab aB AbCAB AbcaB`
	`Ab ab ABc aBcab abC ABc AbCab`
	`ABcAb`
		`app THe Elf DeEd keep SHe DEED`
		`dEEd aPP Ab sheeP Cde THe sHe`
		`ElF CDE sHe KeEP aB eLF An CaT`
		`sHE hE KEeP he SHE dEED ab she`

Commands entered by the users at the command line prompt ">" to run your program are shown in blue.

Note that since there is no user input from the standard input, here's how you should test your output against the sample output (e.g., for test case 2_1) to check for correctness:

```
./main2 abc t1.txt t2.txt t3.txt > myoutput.txt
diff myoutput.txt output2_1.txt
```

2_1

```
./main2 abc t1.txt t2.txt t3.txt
t1.txt: 2 5
t2.txt: 4 11
t3.txt: 0 0
```

2_2

```
./main2 ab t1.txt t2.txt t3.txt
t1.txt: 3 4
t2.txt: 4 9
t3.txt: 3 3
```

2_3

```
./main2 he t3.txt t4.txt t5.txt
t3.txt: 1 2
t4.txt: 3 4
t5.txt: error opening file
```

Problem 3 [35%]:

In this question, let us implement a simple converter to convert a given integer in the range [1, 100] to its binary, octal and hexadecimal representation.

To allow other programs to reuse our program code for decimal-to-binary, decimal-to-octal and decimal-to-hexadecimal conversions, let us put the code in a separate file.

Let us first define the header file “d2boh.h” as follows:

```
// d2boh.h
#ifndef D2BOH_H
#define D2BOH_H
int decimal_to_binary(int input, int output[10]);
int decimal_to_octal(int input, int output[10]);
int decimal_to_hexadecimal(int input, char output[10]);
#endif
```

The interfaces of the functions are explained below:

`decimal_to_binary`:
Parameter 1: An input integer (1 – 100) in decimal form
Parameter 2: An integer array to hold the binary representation of the input integer. For example, if the input integer is 10, the array will contain {1, 0, 1, 0}.
Return value: An integer to represent the number of digits in parameter 2. Using the above example, the return value will be 4.

`decimal_to_octal`:
Parameter 1: An input integer (1 – 100) in decimal form
Parameter 2: An integer array to hold the octal representation of the input integer. For example, if the input integer is 10, the array will contain {1, 2}.
Return value: An integer to represent the number of digits in parameter 2. Using the above example, the return value will be 2.

`decimal_to_hexadecimal`:
Parameter 1: An input integer (1 – 100) in decimal form
Parameter 2: An integer array to hold the hexadecimal representation of the input integer. For example, if the input integer is 10, the array will contain {A} (one single element).
Return value: An integer to represent the number of digits in parameter 2. Using the above example, the return value will be 1.

The skeleton of the implementation file “d2boh.cpp” is given below:

```
// d2boh.cpp
#include <iostream>
#include "d2boh.h"
using namespace std;
int decimal_to_binary(int input, int output[10]) {
        // To be implemented
}
int decimal_to_octal(int input, int output[10]) {
        // To be implemented
}
int decimal_to_hexadecimal(int input, char output[10]) {
        // To be implemented
}
```

Requirements

Task 1: Please complete the implementation of the 3 functions.

Task 2: Please implement a main program “main3.cpp” which accepts 2 positive integers.

- The first integer indicates the mode (2 for binary conversion, 8 for octal conversion and 16 for hexadecimal conversion).
- The second integer is the input decimal number which is in the range [0, 1000].

The program then prints the binary, octal or hexadecimal representation of the input decimal number on screen based on the mode. If the input decimal number is 0, the output should be 0.

You must call the functions
`decimal_to_binary()`, `decimal_to_octal()`, `decimal_to_hexadecimal()`
defined in “d2boh.cpp” in your main program.

Please compile your programs using the command `g++ -pedantic-errors -std=c++11 main3.cpp d2boh.cpp -o main3`. Then run the main program using the command `./main3` to see whether it behaves as expected.

Task 3: Please create a `Makefile` to generate the following targets:
`d2boh.o`: which depends on `d2boh.cpp` and `d2boh.h`
`main3.o`: which depends on `main3.cpp` and `d2boh.h`
`main3`: which depends on `d2boh.o` and `main3.o`
`clean`: for cleaning `main3`, `main3.o` and `d2boh.o`

You should assume that there can be a file named “clean” in the folder and you need to make sure that running `make clean` will not cause any conflict.

Upon `Makefile` is created appropriately, you should be able to compile all programs by issuing the command `make main3` and cleaning all object files by issuing the command `make clean`.

Sample Test Cases

User inputs are shown in blue.

3_1:

```
2
13
1101
```

3_2:

```
8
36
44
```

3_3:

```
16
45
2D
```

3_4:

```
2
95
```

```
1011111
```

3_5:

```
16
100
64
```

If you face any problems, **please feel free to contact us or join our STA support session! We are very happy to help you!**
We wish you enjoy this assignment! ☺