ENGG1340 / COMP2113, Assignment 2

Due Date: Nov 9, 2020 23:59

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

Total marks: 100 marks

- 5 marks for proper code comments, indentation and use of functions
- 95 marks for program correctness

General Instructions

Read the instructions in this document carefully.

In this assignment you will solve 5 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

Note carefully whether your C/C++ programs should read from the **standard input**, **command line input** or **file input**. Also, unless specified otherwise, your answer should be printed through the **standard output**. If you failed to follow the instructions, 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

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 2 test case 3 as an example. The sample input and the expected output are given in the files `input2_3.txt` and `output2_3.txt`, respectively. Suppose that your program is named "2", do the followings at the command prompt of the terminal to check if there is any difference between your output and the expected output.

```
./2 < input2_3.txt > myoutput.txt
diff myoutput.txt output2_3.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 given by the sample test cases.

Coding environment

You must make sure that your program can compile, execute and generate the required outputs on our standard environment, namely, the gcc C/C++11 environment we have on the CS Linux servers (academy*). 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.

For Problems 1,2,3 and 5 on C++ programming, make sure the following compilation command is used to compile your programs:

```
g++ -pedantic-errors -std=c++11 [yourprogram].cpp
```

For Problem 4 on C programming, make sure the following compilation command is used to compile your program:

```
gcc -pedantic-errors -std=c11 4.c
```

Submission

Name your C/C++ programs as the following table shows and put them together into one directory. Make sure that the folder contains only these source files (`*.cpp` / `*.c` / `*.h` / `Makefile`) and no other files. **Compress this directory as a `[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. Resubmission after the deadline is not allowed.

A maximum of 5 marks will be deducted if you fail to follow the submission instructions strictly.

Note that code templates for Problems 1 to 5 are provided, and you should base your work on them.

Problem	Code files provided	Files to Submit
1	`1.cpp`	`1.cpp`
2	`2.cpp`	`2.cpp`
3	`3.cpp`	`3.cpp`
4	`4.c`	`4.c`
5	`d2boh.h` , `d2boh.cpp`	`d2boh.cpp` , `main.cpp` , `Makefile`

Late submission

If submit within 3 days after the deadline, 50% deduction. After that, no mark.

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.

Discussion forum

You are not alone! If you find yourself stuck on something, post your question to the course forum. 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.

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: (C++) Inner Matrix Rotation

The completed program will read the dimension `dim` and the elements of a `dim` x `dim` matrix, and then display the elements of its inner `dim-2` x `dim-2` matrix rotated clockwise through 90 degrees.

You should start working from the `1.cpp` provided. The `main()` function has been completed for you and your task is to write definitions of functions `read_matrix()` and `display_inner_cw_rotated()` to satisfy the following specifications:

- `read_matrix()`: reads `dim`, an integer supplied by the user, into the variable passed as the second argument, where `dim` is guaranteed to be in {3, 4, ...,10}. It then reads values of elements of a `dim` x `dim` square matrix of integers, each separated by whitespace. The values are supplied by the user row by row and are read into the 2D array passed as the first argument. The size of the array is `MAXDIM` x `MAXDIM`, where `MAXDIM` is a global constant and thus is available for use in your function definitions. `MAXDIM` is initialized with a value of 10 and, therefore, the array is large enough to hold the maximum number of values that the user will supply.
- `display_inner_cw_rotated()`: takes as a first argument a 2D array containing a square matrix and, as a second, the dimension, `dim`, of that matrix. It displays the values of a `dim-2` x `dim-2` matrix formed by deleting the outermost rows and columns of the original matrix (that is, those that form its outer perimeter) and then rotating that reduced matrix clockwise through ninety degrees. The elements of the resulting matrix are displayed row by row **with a trailing space after each element** (check carefully your outputs against the given sample outputs using `diff`).

Sample Test Cases

User inputs are shown in blue.

1_1:

```
3
11 12 13
21 22 23
31 32 33
22
```

1_2:

```
4
11 12 13 14
21 22 23 24
31 32 33 34
41 42 43 44
32 22
33 23
```

1_3:

```
5
11 12 13 14 15
21 22 23 24 25
31 32 33 34 35
41 42 43 44 45
51 52 53 54 55
42 32 22
43 33 23
44 34 24
```

Problem 2: (C++) Isolated Bounding Boxes

This problem builds on top of Problem 5 of Assignment 1 (A1P5). Your C++ program will accept the same input as in A1P5, but this time your program should compute the axis-aligned boungind boxes (AABBs) for each of the input 2D geometries, and also identify the isolated AABBs which do not overlap with any other AABBs. (Note: Two AABBs are said to overlap if they share a common point in the 2D plane).

You are provided with a template program `2.cpp`.

Input: Each line of the user input begins with a character indicating the type of geometry, followed by some parameters of the geometric object. The input line can be one of the followings:

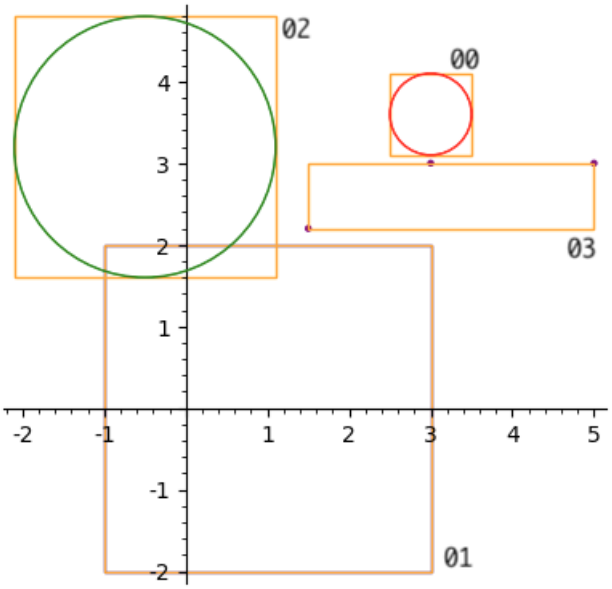
- `R` $x\ y\ width\ height$
  where `R` represents an input rectangle, x, y are floating-point numbers for the x-, y-coordinates of the rectangle center, $width$ and $height$ are floating-point numbers for the rectangle size along the x- and y-axes, respectively.
- `C` $x\ y\ radius$
  where `C` represents an input circle, x, y are floating-point numbers for the x-, y-coordinates of the circle center, and $radius$ is a floating-point number for the radius of the center
- `P` $n\ x_1\ y_1\ x_2\ y_2\ \ldots\ x_n\ y_n$
  where `P` represents an input point set, n is an integer indicating the number of points in the set, and x_i, y_i, $i = 1, \ldots, n$ are floating point numbers for the x-, y-coordinates of the n points
- `#`
  indicates end of input

Output:

- A line
  `AABB **: xmin xmax ymin ymax`
  for EACH of the input geometries (in the same order of the input), where `**` is a 2-digit ID (left padded with zeros) indicating which geometry this AABB corresponds to. The ID is given by the input order of the geometry, it should be `00` for the first input geometry, `01` for the second, and so on.
- then a line `Isolated AABBs:` followed by the ID of the isolated AABBs on each subsequent line, in increasing order of the IDs.

Example:

- Take sample test case `2_4` as an example. There are four input geometries, a circle (r = 0.5), a rectangle, a circle (r = 1.6), a point set, and they will have IDs `00`, `01`, `02`, `03`, respectively. The first four lines of output give the AABBs (x, y, $width$, $height$) of the four geometries, and the last two lines of the output tell that each of the AABBs `00` and `03` do not overlap with the AABBs of any other geometries.



Requirement:
You should start working from the `2.cpp` provided. A `struct AABB` has been defined for you. Check the code comment for its structure. You will need to complete the program with the following tasks:

- **Task 1**: Complete the member functions `ReadRectangle()` and `ReadCircle()` of `struct AABB`. These function are called by `main()` to read the input geometries and compute the AABBs. You may take the completed member function `ReadPointSet()` as reference.
- **Task 2**: The `main()` function has already read in the geometries and stored the AABBs of the geometries in an array `boxes`. Write your code to display the information of the AABBs for all input geometries by running through `boxes`.
- **Task 3**: Complete the function `IsOverlap()` which determines whether two AABBs overlap with each other.
- **Task 4**: In `main()`, write your code to identify the isolated AABBs and display their IDs in the output formats specified above.

Note:

- You may assume that the maximum number of input geometries is 20.
- Use the `setw()` and `setfill()` functions provided by the header `<iomanip>` to achieve zero padding. (http://www.cplusplus.com/reference/iomanip/setfill/)
- You may add your own functions wherever appropriate for better program modularity.

Sample Test Cases

User inputs are shown in blue.

2_1

```
R 0 0 3 2
#
AABB 00: -1.5 1.5 -1 1
Isolated AABBs:
00
```

2_2

```
#
Isolated AABBs:
```

2_3

```
P 2 3 -2 -1 4
C -0.5 3.2 1.6
P 3 -1.5 3 3 3 5 3
R 0 5.75 3 2
#
AABB 00: -1 3 -2 4
AABB 01: -2.1 1.1 1.6 4.8
AABB 02: -1.5 5 3 3
AABB 03: -1.5 1.5 4.75 6.75
Isolated AABBs:
```

2_4

```
C 3 3.6 0.5
R 1 0 4 4
C -0.5 3.2 1.6
P 3 1.5 2.2 3 3 5 3
#
AABB 00: 2.5 3.5 3.1 4.1
AABB 01: -1 3 -2 2
AABB 02: -2.1 1.1 1.6 4.8
AABB 03: 1.5 5 2.2 3
Isolated AABBs:
00
03
```

Problem 3: (C++) Word Search

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 template program `3.cpp`.

Input:

- The 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 your program is named `wordsearch`, then the following command at the command prompt
  `./wordsearch abc t1.txt t2.txt t3.txt`
  will ask your program `./wordsearch` 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 paramters 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 `main()` body 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 from the `3.cpp` provided.
- 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 `main()` function has been completed for you, and you do not need to make any change to it. Read the code in `main()` and understand what it does before you start working.
- You may add your own functions wherever appropriate for better program modularity.

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, assuming that your program name is `wordsearch` .

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 3_1) to check for correctness:

```
./wordsearch abc t1.txt t2.txt t3.txt > myoutput.txt
diff myoutput.txt output3_1.txt
```

3_1

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

3_2

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

3_3

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

Problem 4: (C) Character Counter

Write a C program to count the number of occurrences of each character in an input string.

Input: A string (that does not contain any space characters).

- You may assume that the string is of at most 50 characters.

Output: The number of occurrences for each character.

- Uppercase and lowercase of a character would be considered as the same.
- The output order of the characters should be in the same order of their occurrences in the original string. (See sample cases.)
- Output characters should be in lowercase.

Requirements.

- Use the command `gcc -pedantic-errors -std=c11 4.c -o 4` to compile your program. (Or specify `-pedantic-errors -std=c11` as the compiler flags in the ATOM editor.)

Sample Test Cases

User inputs are shown in blue.

4_1

```
ENGG1340-COMP2113
e 1
n 1
g 2
1 3
3 2
4 1
0 1
- 1
c 1
o 1
m 1
p 1
2 1
```

4_2

```
Micropachycephalosaurus
m 1
i 1
c 3
r 2
o 2
p 2
a 3
h 2
y 1
e 1
l 1
s 2
u 2
```

4_3

```
(^_^;)(-_-;)(~_~;)^^;^_^;(#^.^#)(^^;)
( 5
^ 10
_ 4
; 6
) 5
- 2
~ 2
# 2
. 1
```

Problem 5: (Makefile, C++) Number System Conversion

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 “main.cpp” which behaves as follow:

```
> ./main
Enter a positive integer in the range [1, 100]: 13
Binary of the given number = 1101
Octal of the given number = 15
Hexadecimal of the given number = D

> ./main
Enter a positive integer in the range [1, 100]: 36
Binary of the given number = 100100
Octal of the given number = 44
Hexadecimal of the given number = 24

> ./main
Enter a positive integer in the range [1, 100]: 45
Binary of the given number = 101101
Octal of the given number = 55
Hexadecimal of the given number = 2D

> ./main
Enter a positive integer in the range [1, 100]: 95
Binary of the given number = 1011111
Octal of the given number = 137
Hexadecimal of the given number = 5F

> ./main
Enter a positive integer in the range [1, 100]: 100
Binary of the given number = 1100100
Octal of the given number = 144
Hexadecimal of the given number = 64
```

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 main.cpp d2boh.cpp -o main`. Then run the main program using the command `./main` 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`
`main.o`: which depends on `main.cpp` and `d2boh.h`
`main`: which depends on `d2boh.o` and `main.o`
`clean`: for cleaning `main`, `main.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 main` and cleaning all object files by issuing the command `make clean`.