COMP2045, C++ Programming, 2019/20, Coursework part 1, 2

Overview

The coursework of this module contains two parts. Part 1 and 2 are worth 15% and 25% of the final mark respectively. For each part, you need to submit an individual C++ program that solves the given task. The deadline for the coursework is 23:55pm on Tuesday May 26th 2020.

General requirements:

You MUST meet the following general requirements:

1. All of the code that you submit must be either your own work or copied from the given code. You may not use work from anyone else. You may NOT work with other people to develop your coursework. You may NOT have someone else write any or all of the code in your submission.

2. Create a program which runs without crashing or errors and meets the functional requirements listed below and on the following pages.

3. The aim of this coursework is to investigate your understanding of C++ concepts and techniques and applying them in programming.

- Your program MUST use the supplied class in the way in which it is intended. i.e. to create subclass based on the given class and create objects for comparing cpp files.

- You must not alter the supplied class. It is a good exercise to use existing classes and potentially sub-class them to extend or change behaviour rather than changing the original class. E.g. with many commercial classes you may not have the ability to change their code.

- You should not use any external libraries other than standard C++ class libraries. There any many useful libraries which you could use to improve your programs, but the point of this exercise is to show that you can understand and use fundamental C++ concepts and techniques.

4. If you have any questions about this exercise, please ask in the Q&A forum on Moodle, after a lecture, in a lab, or during the advertised office hours. Do not post your program or parts of your program to Moodle as you are not allowed to share your coursework programs with other students. If any questions require this exercise to be clarified then this document will be updated and everyone will be notified via Moodle.

General marking criteria:

You will lose marks if any of the following apply. In each case we will apply the lower of a percentage penalty or a mark penalty.

- Your program crashes on exit or has a memory leak. (-2 marks or -10% mark)
- Your program crashes at least once during its operation. (-4 marks or -20% of your mark)
- Your program crashes multiple times. (-6 marks or -30% of your mark)
- Your program crashes frequently. (-8 marks or -40% of your mark of your mark)
- Your program has some odd/unexpected behaviour/errors. (-2 mark or -10% mark)
- Your program has a lot of unexpected behaviour/errors. (-4 marks or -20% of your mark)

Late submissions will lose 2 percentage points per hour, rounded up to the next whole hour. No late submissions will be accepted more than 48 hours after the exercise deadline. If you have extenuating circumstances you must file them before the deadline.

Plagiarism

You should complete this coursework on your own. Anyone suspected of plagiarism will be investigated and punished in accordance with the university policy on plagiarism (see your student handbook and the University Quality Manual). This may include a mark of zero for this coursework.

You should write the source code required for this assignment yourself. If you use code from other sources (books, web pages, etc), you should use comments to acknowledge this. You must not copy or share source code with other students. You must not work together on your solution. You can informally talk about higher-level ideas but not to a level of detail that would allow you all to create the same source code.

Remember, it is quite easy for experienced lecturers to spot plagiarism in source code. We also have automated tools that can help us identify shared code, even with modifications designed to hide copying. If you are having problems you should ask questions rather than plagiarize. If you are not able to complete the exercise then you should still submit your incomplete program as that will still get you some of the marks for the parts you have done (but make sure your incomplete solution compiles and partially runs!).

Coursework part 1

In this part of the coursework, you are required to write a program to detect the similarities between multiple cpp files. A cpp file is a text file that contains C++ code. If two files are exactly the same, the similarity is 100%. If two files are different, a value within the range of 0 and 1 should be given to describe the similarity. Your program must use the below class cppfile.

```
class cppfile
{
 public:
                virtual void read_a_cppfile(string) = 0;

 protected:
                int num_of_class;
                int num_of_cin;
                int num_of_cout;
                int num_of_public;
                int num_of_protected;
                int num_of_private;
                int num_of_const;
                int num_of_if;

                int length_of_file;
                int num_of_lines;
                int num_of_blocks;
                int num_of_asterisks;
                int num_of_ampersands;
                int num_of_commas;

                bool use_new;
                bool use_friend;
                bool use_static;
                bool use_argc;
}
```

When you program runs, it should ask the user to enter the number of cpp files to compare, and then ask the user to enter the names of all cpp files, and then output the similarities between those files in the form of a matrix.

You must submit a single C++ source code file containing all your code for this exercise. The file must be named as **compare.cpp** and it should compile and run without needing any external library.

The first line of your program should be a comment which contains your student ID number, username, and full name, of the form:

// 6512345 zy12345 Joe Blogs

The program must compile on **X2Go** without warnings or errors when I use the command

g++ -std=c++11 compare.cpp -o compare

This command will be run on our Linux server cslinux. If it does not compile, for any reason, then you will lose all the marks for testing. The completed source code file should be uploaded to the Coursework 1 Submission link on the Moodle page.

Example input/output

```
zlizpd3 $ g++ -std=c++11 compare.cpp -o compare
zlizpd3 $ ./compare
Enter the number of cpp files to compare (no more than 15): 3
Enter the name of file 1: a.cpp
Enter the name of file 2: b.cpp
Enter the name of file 3: c.cpp
Similarity matrix:
1.000000   0.843464   0.264332
0.843464   1.000000   0.358754
0.264332   0.358754   1.000000
zlizpd3 $
```

Functional requirements

A number of tests will be run against your program with different input data designed to test if your program works well. If your program compiles and runs correctly, up to 9 marks may be received for testing. Up to 6 more marks are set for optional requirements. Total 15%.

The optional requirements include:

- Does the program handle incorrect input? (1 mark)
- Does the program handle incorrect file names? (1 mark)
- Does the program handle large text files? (1 mark)
- Does the program output reasonable similarity values? For example, if the similarity between files A and B is high and the similarity between files A and C is high, the

similarity between files B and C should be high. On the other hand, If the similarity between files A and B is high and the similarity between files A and C is low, the similarity between files B and C should be low. (3 marks)

Hint:

- There may be comments contained in cpp files, which should NOT be considered in the comparison of cpp files.
- You do not need to consider whether the code in a cpp file is correct or not.
- You can use more criteria than what are given in the class cppfile.

Coursework part 2

In this part of the coursework, you are required to design the algorithm and data structure for a problem called travelling salesman problem (TSP). Your program should be able to solve the problem and more importantly, allow other people to extend your classes to support future development.

The travelling salesman problem is a well-known combinatorial optimisation problem. In this problem, a salesman is initially located at a starting position and he has a list of cities that must be visited. Each city must be visited exactly once. After visiting all these cities, he then need to return to his starting position. You don't need to worry about the actual path from one city to another, because they are irrelevant in this problem. To simplify the problem, each city is given an identifier, an x coordinate and a y coordinate. The travelling distance from one city to another is the Euclidean distance calculated based on their coordinates. What you need to work out in this problem is to find a tour that minimises the total distance travelled.

There is no way to solve a large TSP problem to optimal. Thus, don't bother finding the best solution.

Your main program should be called **TSP_Solver.cpp**. For a problem instance file, it should compute a solution to the instance.

Each problem instance is essentially a text file with a list of cities. The first city is the starting point of the salesman. The format of the text file is like the one below:

```
City  x     y
0     0     0
1     5     7
2     -12   12
```

When reading the text file, you need to skip the first line as it only contains column information. Columns will be separated by tabs ('\t'). Once you have read everything into your program, convert the text into your inner data structure.

A feasible solution of a TSP is a tour that visits every city exactly once . Thus, the correct output of your program should be a sequence of cities. The sequence matches to a problem instance. An output of your solution should be in this format:

"distance\tcity …\tcity\tcity"

Make sure that each item in one solution is separated by a tab ('\t'). For example, a solution to the above instance may be

43.29 0 1 2 0

The value 43.29 is the total distance travelled. The tour starts from city 0, then city 1, city 2, and finally city 0.

You must submit a single C++ source code file containing all your code for this exercise. The file must be named as **TSP_Solver.cpp** and it should compile and run without needing any external library.

The first line of your program should be a comment which contains your student ID number, username, and full name, of the form:

// 6512345 zy12345 Joe Blogs

The program must compile on **X2Go** without warnings or errors when I use the command

g++ -std=c++11 TSP_Solver.cpp -o tsp

This command will be run on our Linux server cslinux. If it does not compile, for any reason, then you will lose all the marks for testing. The completed source code file should be uploaded to the Coursework 2 Submission link on the Moodle page.

Example input/output

```
zlizpd3 $ g++ -std=c++11 TSP_Solver.cpp -o tsp
zlizpd3 $ ./tsp instance1.txt
43.29  0  1  2  0
zlizpd3 $ ./tsp instance2.txt
154.54  0  8  5  2  4  3  9  7  6  1  0
zlizpd3 $ ./tsp inst9.txt
File not found.
zlizpd3 $
```

Functional requirements

A number of tests will be run against your program with different instance files designed to test if your program works well. If your program provides solutions for all valid test files, up to 15 marks may be received. Up to 10 more marks are set for optional requirements. Total 25%.

I will run your program on cslinux. Your program will have 10 minutes running time for each problem instance with less than 1000 cities and 20 minutes for instances with 1000+ cities.

The optional requirements include:

- Does the program handle incorrect instance files? (1 mark)
- Does the program always provide correct solutions? (1 mark)

- Is there any memory leak? (1 marks)
- Does the program handle large instance files (more than 5,000 cities)? (2 marks)
- Does the program use OOP design wisely? (2 marks)
- Does the program provide better solutions than most students in many instances? (3 marks)

Hint:

- Avoid using global functions and global variables.
- There are plenty of algorithms for TSP in the literature, some of them are simple while others are really complicated to implement. A simple algorithm generally cannot provide good solutions.
- You can use multiple algorithms in your program. Some aims to generate a feasible solution quickly and some aims to generate better solutions.
- You should manage your time and choose appropriate algorithms if you want to earn the optional 3 marks.