CSCI251 Advanced Programming

Spring 2023

Lab 2 for Week 4 & 5

General Note

In these series of laboratory exercises you will be learning and practicing various programming concepts enabled by C++ programming language. Mostly, these concepts will be introduced through simple and sometimes complex examples. You are encouraged to think about how and why things work the way they do. The exercises are not set up to trip you, but you need to think carefully.

In order to focus on the task at hand, you can use an integrated development environment (IDE) to write, compile and test your code. In one of the notes provided on Moodle, you can find the steps required to install Code::Blocks on several OS (e.g. Windows, Linux, MacOS). You are encouraged to follow the instructions and install it for your operating system. The operating system on which you develop your code should not matter. All that needs to happen is that your code is C++17-compliant. The system on which your code will be tested has g++ 7.5 installed and your code must work on that system (i.e. **capa.its.uow.edu.au**).

In order to test that your code is compliant and will work on **capa.its.uow.edu.au**, you will need to do a secure login to capa and test your code. The appropriate option required to ensure that you are compiling against C++17 is the -std=c++17 added to your command line instruction. For example:

g++ file1.cpp file2.cpp file3.cpp -std=c++17 -o output_exec

Get familiar with your IDE and ensure that the build option is set to use c++17.

Use sensible names for your variables and insert comments to let the reader know what your code is doing. If your code contains no comments you will lose marks.

Code Submission

For each exercise, name your code as instructed. Each Lab will be due for submission at midnight of Weeks 3, 5, 7, 9 and 11. You will also be reminded accordingly as the session progresses. Submission points will be set up on Moodle. For each task you might save the code file(s) like:

t1.cpp: only one file.

t1-1.cpp: if multiple subtasks

t1.zip: if there are multiple files

At the end, please submit a (big) zip _le that includes all task for the lab, and name it like: “familyname_studentId_lab#.zip”. The symbol # represents the lab number. For example, in Lab 1 (week 2&3), the submission can be named as “Jordan_12345_lab1.zip”.

Task 1 (20 points)

1. (6 points) Debug: Debug-A.cpp. This file is used to print integers from highest to lowest (inclusively).
2. (6 points) Debug: Debug-B.cpp. The software will accept two integers and computer the log of log10(10), log(e) (and e = 2.718), and the lower input (of those two integers).
3. (8 points) Debug: Debug_C.cpp. This file is for recording the book information and later saving them into a file of library.txt;

Task 2 (20 points)

Write a program *Read-words.cpp* to read words from the file Words, and output them together with the line number of each word to the file *output.txt*. Include appropriate checks on the opening of the file.

Task 3 (20 points)

Create a piece of code *Cat.cpp*. This code should contain a struct *Cat* with the following properties:
1. Data fields (at least) for name, breed and age.
2. A constant static field for the insurance, which is AUD $99.9.
3. For a cat, its owning cost, for simplicity, is calculated as age*1.5 + insurance.
4. You need to use member functions to set and display the data.
5. The main() function has been shown below (you cannot change this main function):

```
Cat myCat;
myCat.setCat("Tiger", "British Shorthair", 1);
myCat.calculatePrice();
myCat.showCat();
```

The related output should look like:

```
Cat: Tiger is a British Shorthair
Its age is 1
Insurance fee: $99.9
Owing cost is: $101.4
```

Task 4 (20 points)

Write a program which takes as its input two float matrices M1 and M2 and their corresponding dimensions and computes the matrix multiplication. The function definition should be like this:

```
void multiplyMatrices(float M1[][10], float M2[][10], int M1_rows, int M1_cols,
int M2_rows,
int M2_cols)
```

The dimensions (M1_rows, M1_cols, M2_rows, M2_cols) and two matrices are passed to the function in the main method, like

```
/* the example input */
float M1[10][10] = {{0.1067, 0.7749},{0.9619, 0.8173},{0.0046, 0.8687}};
float M2[10][10] = {{0.0844, 0.2599},{0.3998, 0.8001}};
multiplyMatrices(M1,M2,3,2,2,2);
/* end of example input */
```

Function *multiplyMatrices* performs the following steps:
1. check the input is valid (the dimensions of the matrices) or not, if not, return an error message stating "Invalid Matrix Dimensions"
2. if the input is valid then computes M1 * M2 and store the results in another matrix, say M3
3. print this M3 with elements of 2 decimal places, separated by blank spaces, and each row is printed on a new line

Task 5 (20 points)

You have been provided the following customised exception class, that could be used to throw a warning message. Write the main() function that handles the division with the provided exception class. That is, with the case of division by zero, the program should try to catch it and throw a message saying "Error with the division by zero" .

```
class MyException : public exception{
    public:
    // this function will be called to output a string when an exception araised.
    // more details can be found:
    // https://www.cplusplus.com/reference/exception/exception/what/
        const char * what() const throw()
        {
            return "Attempted to divide by zero!\n";
        }
};
```

The related output should look like:

```
Enter the numbers of x and y:
5
2
x/y = 2.5

Enter the numbers of x and y:
10
0
Attempted to divide by zero!
```