Mock Take Home Assessment

(Based on previous Assessed Practical 5)

Tuesday 21st April 2020

Before you begin, please read the instructions carefully.

If you have Visual Studio create a new Empty Project named TakeHome_Practical for the assessment. Inside the project, add a new source file takehome_main.cpp.

If not open create a new folder TakeHome_Practical. Open text editor of your choice, e.g. Notepad++ . Create a new text file called takehome_main.cpp in the new folder.

In both instances, the file will be used to hold part of your solutions for the assessment. This will also be the place to create your main function to test all your solutions. Do your best to write accurate C++, although minor syntax errors will be overlooked.

Part 1 Two-Dimensional Arrays and File I/O

1. Two-D arrays

In `takehome_main.cpp` write a function

```
int median(int N, int M, int minV, int maxV)
```

in which the following operations are performed:

(1) Creating a dynamic 2-d integer array of size `N`×`M`, where `N` and `M` are positive integers;
(2) Populating the array with random numbers within the range from `minV` to `maxV` inclusive, where `minV` and `maxV` are any integer numbers with `maxV > minV`;
(3) Finding the median number `medV` (array[n][m]) in the array that has minimum distance to `minV` and `maxV`, where the distance is defined as follows:
"Distance of medV to minV and maxV" = $(medV - minV)^2 + (medV - maxV)^2$;
(4) Deleting the array;
(5) Returning the medV value found.

Demonstrate that your function works correctly by calling it with appropriate `N`, `M`, `minV` and `maxV` values.

2. File I/O

In `takehome_main.cpp` write a function

```
void longest_pure_word(char* fname)
```

The function is called with a char string for the file name of a text file, and will find the longest word in this file. Use the supplied text file `corpus.txt` as an example to write your code. In `corpus.txt` each line contains a text sentence, beginning with a sentence code (e.g., 011a0101). You should ignore this code as it is not a word. This file also contains many word items which include some non-alphabetic character(s), e.g., \', \-, \., \,, \", \&, \%, \; …. You should ignore all these word items and only count the length of the **pure** words, in which each character is an alphabetic character. If there are several longest words of the same length, just find the first instance.

Your code should include: (1) opening the file for input, (2) reading each word in the file until the end of the file, (3) finding the longest pure word, (4) printing the longest pure word found and its length, and finally (5) after finishing testing the function adding a comment at the end of the

function describing your finding: "The longest pure word found in corpus.txt is X, with length = Y." where X and Y are produced by running your program on `corpus.txt`.

If you are using Visual Studio then in the Debug mode, you may place `corpus.txt` in the same folder with `takehome_main.cpp` such that your code can load the file with the name "`corpus.txt`". It is noted that with Visual Studio 2017 you may have to use `(char*)"corpus.txt"` to call the function, which casts the `const` file name to a non-`const` char string.

Part 2 Graphs

The supplied class files "Graph.h" and "Graph.cpp" provide facility to construct Graphs based on the adjacency matrix representation. Save them in your project folder, add them to your project, and #include "Graph.h" in `takehome_main.cpp OR` save them in CSC2040Mock and examine them.

(1) Create the definition file: `graph.txt`, for the weighted graph shown in Fig.1. Save the file in a convenient location so it can be easily loaded for the following test.

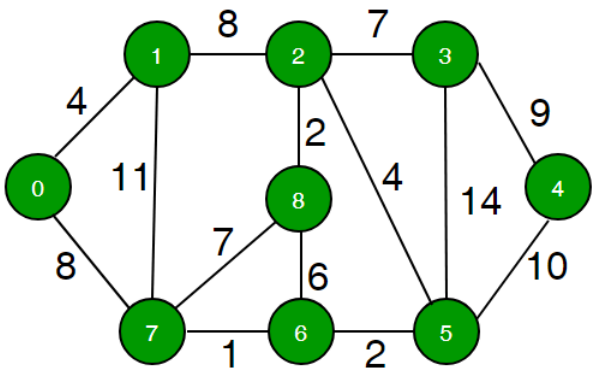

Fig.1. A weighted graph

(2) In the given class files `Graph.h` and `Graph.cpp`, add a new public member function `void get_max_weight_dest(int source)`. For each given source vertex `source`, this function will find, from all the destination vertices that may have a direct edge from `source`, the destination vertex that has maximum weight. It will print out the index of the destination vertex found and the value of the corresponding maximum weight, in the following format:

```
X is the max-weight destination from source Y.
The weight is Z.
```

where X and Y are suitable destination and source indices and Z is their edge weight. (In this representation we assume that all weights >= 0.0; two vertices without an edge have an edge with weight = 0.0).

(3) In `takehome_main.cpp`, write a `testGraph` function to test if your program is correct. The operations in the `testGraph` function should include: (1) to create a `Graph` object, to represent the graph defined in `graph.txt`, and (2) to test the new member function of the class `get_max_weight_dest`, by calling it using variable source vertices taken from the keyboard.

Part 3 Hash tables

The following is a function which extends the `find` function in the `HashTable` class in Lecture 14 and Practical 14. The function takes a `string`-type `key` and calculates its hash index (i.e., location) in the hash table `Table` of size `TableSize`. It returns the `index` at which the matching key is found, or the index of the first empty slot for its storage if it is not found in the `Table`, by using the linear probing method.

```
int find(string* Table, int TableSize, string key)
{
   // calculate the hash index of the matching key in the Table
   unsigned int index = 0;
   for (int i = 0; i < key.size(); i++)
      index = index * 37 + key[i];

   // use linear probing to revise the index if there is a collision, to locate
   // the matching key or the first empty slot for its storage
   while (!Table[index].empty() && Table[index] != key) {
      index = index + 1;
   }

   // return the index found
   return index;
}

// test hash table
string* Table = new string[17];
cout << "hash index = " << find(Table, 17, "belfast") << endl;
delete[] Table;
```

Type the above `find` function into `takehome_main.cpp`, and then type the code after that into your main function. The testing code calls `find` with a specific example for the `Table` and `key`. Run the program and you will see it crashing. This is because there are two mistakes in `find`. Find these and correct them. You should only submit the corrected `find` function.

Submission Notes

1) Save your project.
2) Make sure you have the following FOUR source files for submission:
   Takehome_main.cpp, Graph.h, Graph.cpp in your project/folder TakeHome_Practical
   graph.txt (in a location of your choice)
3) ZIP these four files.
4) Check that your .zip file contains the above 4 source files (double click on the .zip file. You do not need to extract the files again).
5) In case something goes wrong with your submission, first make a backup of your zip file on to a USB drive, and/or email it to yourself.
6) On the Canvas Page for CSC2040, click Assignments and then Mock Take Home Assignment. Choose Submit File, followed by File Upload. Click Choose File and then locate the zip.