**Sabanci University**

Faculty of Engineering and Natural Sciences

CS204 Advanced Programming

Summer 2018-2019

Homework 3 – A Great Day for Freedom

Due: 6 August 2019 11.55pm (SHARP)

|  |
| --- |
| **DISCLAIMER:**  **Your program should be a robust one such that you have to consider all relevant user mistakes and extreme cases; you are expected to take actions accordingly!**    **Only checking the sample run cases might not be sufficient as your solution will be checked against a variety of samples different than the provided samples; however checking these cases are highly encouraged and recommended.**    **You must** NOT **collaborate with your friends and discuss your solutions with each other. You have to write down the code on your own. Plagiarism will not be tolerated AND cooperation is not an excuse!** |

**Introduction**

The aim of this homework is to practice on stack data structures, class design issues and operator overloading. In this homework, you will implement a program that finds the shortest exit from a given maze.

**Inputs to Your Program**

Your program should prompt for an input file name and read the file name from the standard input (cin). This file will have the structure of the maze and the entry point for it, as exemplified below.

|  |
| --- |
|  |

*maze1.txt*

If the input file cannot be opened successfully, your program should give an error message and terminate.

While reading the file, **you must store the matrix in a dynamic 2D array**. **Refusing to do so will result in a reduction in your grade**.

**Format of the Inputs**

In the first line of the data file containing the maze matrix, there will be the dimensions of the respective matrix, i.e. the number of columns and the number of rows, in this given order, separated by a single space character. In the next line, the x and y coordinates of the entry point is given, again separated by a single space character. In the remaining lines of this file, there will be the matrix of zeros and ones, where zeros represent available passages and ones represent walls. Here, the top leftmost cell corresponds to the coordinate (*0*, *0*), while the bottom rightmost cell corresponds to the coordinate (*row-1*, *col-1*).

In each line of this file, the numbers will be separated by a single space character, and the rows will be separated by a newline ('\n') character, as seen above in *maze1.txt*. Also, there will not be any empty lines in this file. You may assume that this file won't contain any characters other than digits, spaces, newlines and EOF. You may also assume that the column and row count given at the beginning of the file will always be correct, and the entry point given will always be existing and available (zero).

**Program Flow and Search**

Your program should start by asking the filename for the matrix file. If the file can be successfully opened, your program should read the file and store the numbers in a **Dynamic 2D Array**. Any other data structure (like vector, built-in array, dynamic 1D array, linked list, etc.) will **not** get full credits. At the end of the program, you **must** clear this dynamic array from the memory.

After reading the maze matrix into a 2D dynamic array, the search will start from the given entry point. This entry point is not considered as an exit. There can be multiple paths which yield to an exit from the maze. Your program should be able to discover all of these exits and find the shortest one. There are two assumptions that you can make to easen the process:

1. For any point *p* on the maze, there can be at most one path that goes from the entry point to *p,* which also ensures that, from the entry point to any specific exit point, there will always be a single path.

1. There may not be two paths sharing the shortest path crown. This means that, you will always be able to find a single shortest path.

Your program will search for all the paths that yield to an exit. Throughout this process, you **must store the active path in a Dynamic Stack. Using any other data structure for remembering the active path will cause a grade of zero (marking the active path on the maze array itself is also not acceptable).**

Throughout the execution, your program will find many viable solutions, but it will only need to remember the shortest one. During this process, **you cannot use any aggregate data structures (arrays, lists, vectors etc.) to store a collection of stacks/paths.** You can use a single stack object to store the shortest path found until that point of execution, and rebuild this object with a new path whenever a shorter one is found.

At the very end, of course, your program should display the shortest path, i.e. the visited coordinates, on the standard output (cout).

**The Stack Class**

For this homework, you **must** build a stack class; not only from obligation, but also for your own good. You can start with DynIntStack class that we had shared with you under lecture resources. However, this class works with a single integer only. For more convenience, you are expected to use a Point struct in the Stack class. Following are the Point and StackNode struct definitions that you **must** use.

struct Point{

int x;

int y;

};

struct StackNode{

Point value;

StackNode \*next;

};

In order to push you for better overall class design and practice on operator overloading, there are more restrictions on what you are expected of while implementing your Stack class:

1. **Default Constructor Obligation:** Your class should have a default constructor, which constructs an empty stack.
2. **Destructor Obligation:** Your class should have a destructor, which deallocates all the StackNodes in the Stack.
3. **Must implement operator << for Stack push:** Your class must not have a regular public push method. Rather than that, you must overload the operator << which will have our Stack object on the left-hand side and a Point object on the right-hand side (i.e. myStack << myPoint). The Point should be added to the top of the stack.
4. **Must implement operator >> for Stack pop:** Your class must not have a regular public pop method. Rather, you must overload the operator >> which will have our Stack object on the left-hand side and a Point object on the right-hand side (i.e. myStack >> myPoint). The top element of the stack should be removed and the information should be moved to the Point object on the right-hand side of the operator.
5. **Must implement operator << for ostream:** Your program must implement a << operator for *ostream << stack* type of operations. This is useful for printing the shortest path after the search is completed.
6. **Must implement assignment operator:** Your class must implement an assignment operator which clears the stack on the left-hand side and deep copies the stack on the right-hand side to the one on the left-hand side. This operator is particularly useful when you need to move information from one stack variable to another, or when you need to duplicate a stack.

If not prohibited by any of these rules, you can add other fields or methods to your class. However, we would like to remind you that any workaround which destroys the philosophy of object-oriented design will hurt your grade. If you cannot make sure whether your class design is solid or not, you may ask your TAs about it.

**Sample Runs**

Below, we provide some sample runs of the program that you will develop. The *italic* and **bold** phrases are the standard input (cin) taken from the user (i.e., like ***this***). You have to display the required information in the same order and with the same words/spaces as here; in other words, there must be an exact match!

We will be automatically grading your homework using GradeChecker, so it is very important to satisfy the exact same output given in the sample runs. You can utilize GradeChecker [(http://sky.sabanciuniv.edu:8080/GradeChecker/)](http://sky.sabanciuniv.edu:8080/GradeChecker/) to check whether your code is working in the expected way. To be able to use GradeChecker, you should upload all of your files used in the homework **without zipping them**. Just a reminder, you will see a character ¶ which refers to a newline in your expected output.

**Sample Run 1**

Welcome to the homework "A Great Day for Freedom".

Please enter a file name for the maze: ***matrix1.txt***

File name is incorrect. Program exiting...

**Sample Run 2**

Welcome to the homework "A Great Day for Freedom".

Please enter a file name for the maze: ***maze1.txt***

Printing the shortest exit path, with the length of 8

0,5

1,5

1,4

2,4

3,4

3,5

3,6

4,6

**Sample Run 3**

Welcome to the homework "A Great Day for Freedom".

Please enter a file name for the maze: ***maze2.txt***

Printing the shortest exit path, with the length of 17

11,0

11,1

10,1

9,1

8,1

8,2

8,3

8,4

7,4

6,4

6,3

6,2

5,2

4,2

3,2

3,1

3,0

**Some Important Rules**

Although some of the information is given below, please also read the homework submission and grading policies from the lecture notes of the first week. In order to get a full credit, your program must be efficient, modular (with the use of functions), well commented and indented. Besides, you also have to use understandable identifier names. Presence of any redundant computation, bad indentation, meaningless identifiers or missing/irrelevant comments may decrease your grade in case that we detect them.

When we grade your homeworks, we pay attention to these issues. Moreover, in order to observe the real performance of your code, we are going to run your programs in Release mode and **we may test your programs with very large test cases**. Hence, take into consideration the efficiency of your algorithms other than correctness.

**How to get help?**

You may ask your questions to TAs or to the instructor. Information regarding the office hours of the TAs and the instructor are available at [Course Google Drive Folder](https://drive.google.com/drive/folders/1EozjiM4Ogau08Zd2gemSbNjzQxBj-Rma?usp=sharing).

**YOU SHOULD USE GRADE CHECKER FOR THIS HOMEWORK!**

You should use Grade Checker (<http://sky.sabanciuniv.edu:8080/GradeChecker/>) to check your expected grade. Just a reminder, you will see a character ¶ which refers to a newline in your expected output.

Make sure you upload the .txt files, too.

Grade Checker and the automated grading system use a different compiler than MS Visual Studio does. Hence, you should check the "***Common Errors***" page to see some extra situations to consider while doing your homework. If you do not consider these situations, you may get a lower score (even zero) even your program works correctly with Visual Studio.

***Common Errors Page***: <http://sky.sabanciuniv.edu:8080/GradeChecker/commonerrors.jsp>

Grade Checker can be pretty busy and unresponsive during the last day of the submission. Due to this fact, leaving the homework for the last day generally is not a good idea. You may wait for hours to test your homework or make an untested submission, sorrily..

Grade Checker and Sample Runs together give a good estimate of how correct your implementation is, however we may test your programs with different test cases and **your final grade may conflict with what you have seen on Grade Checker.** We will also **manually** check your code (comments, indentations and so on), hence do not object to your grade based on the Grade Checker results; but rather, consider every detail on this documentation. **So please make sure that you have read this documentation carefully and covered all possible cases, even some other cases you may not have seen on Grade Checker or Sample Runs**. The cases that you *do not need* to consider are also given in this documentation.

Submit via SUCourse ONLY! **Grade Checker is not considered as a submission**. Paper, e-mail or any other methods are not acceptable, either.

The internal clock of SUCourse might be a couple of minutes skewed, so make sure you do not leave the submission to the last minute. In the case of failing to submit your homework on time:

"No successful submission on SUCourse on time = A grade of 0 directly."

**What and where to submit (PLEASE READ, IMPORTANT)**

You should test your program using Grade Checker. We will use the same UNIX based C++ compiler that Grade Checker uses for grading your homework.

It'd be a good idea to write your name and lastname in the program (as a comment line of course). Do not use any Turkish characters anywhere in your code (not even in comment parts). If your full name is "Duygu Karaoğlan Altop", and if you want to write it as comment; then you must type it as follows:  
 *// Duygu Karaoglan Altop*

Submission guidelines are below. Since the grading process will be automatic, you are expected to strictly follow these guidelines. If you do not follow these guidelines, your grade will be *zero*. The lack of even one space character in the output will result in your grade being zero, so please test your programs yourself and with the Grade Checker tool explained above.

* Name your cpp file that contains your program as follows:

***"SUCourseUserName\_hw3.cpp"***

Your SUCourse user name is actually your SUNet username which is used for checking sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SU e-mail address is **atam@sabanciuniv.edu**, then the file name must be: **"atam\_hw3.cpp"**

* Please make sure that this file is the latest version of your homework program.
* You should upload all the .txt files and class filesto SUCourse as well.
* Do not zip any of the documents but upload them as separate files only.
* Submit your work **through SUCourse only**! You can use the Grade Checker only to see if your program can produce the correct outputs both in the correct order and in the correct format. It will not be considered as the official submission. You must submit your work to SUCourse.

*You may visit the office hours if you have any questions regarding submissions.*

**Plagiarism**

Plagiarism is checked by automated tools and we are very capable of detecting such cases. Be careful with that...

Exchange of abstract ideas are totally okay but once you start sharing the code with each other, it is very probable to get caught by plagiarism. So, do NOT send any part of your code to your friends by any means or you might be charged as well, although you have done your homework by yourself. Homeworks are to be done personally and you have to submit your own work. **Cooperation will NOT be counted as an excuse.**

In case of plagiarism, the rules on the Syllabus apply.

Good Luck!

Tolga Atam, Duygu K. Altop