University of Technology, Sydney

Faculty of Engineering and Information Technology
School of Software

31251 – DATA STRUCTURE AND ALGORITHMS
32510 – OBJECT ORIENTATED PROGRAMMING IN C++
AUTUMN 2009

ASSIGNMENT 2 - DUE 11:59pm, 7/05/2010

This assignment is worth 15% of the total marks for this subject.

Requirement

You are to prepare a C++ program that accepts one command line argument. This argument will be the name of a file containing a maze. For example, if the name of your compiled program is `a.out` and the name of the maze file is `maze1` then you would run the program as follows.

```
./a.out maze1
```

Maze File

The maze file is a text file made up of spaces for pathways, `#`'s for walls plus one '`s`' and one '`f`', representing the start and finish of the maze. For the maze to be well formed it must meet the following requirements

1. The maze must be surrounded by an unbroken perimeter of `#`'s. This means there are no paths on the inside of the maze leading to the outside.
2. The start and finish are both inside the maze.
3. After the final `#` in each line of the maze the next character is a newline.
4. There are no spaces within the maze that cannot be reached by moving down one of the maze paths.
5. There are no U shaped mazes. See diagram below.

The following is an example of a well formed maze file.

```
        #######
        #     #############
######## ### #           ###
#         #f### #########   #
#s#### # #     #        ### #
###    # ####### ### ###   #
  # #### #         # ###   ###
  #    ###########     ###
  #### #             #####
     ###############
```

The following points can also be noted about the mazes

1. The maze can be any width and length and, as you can see, does not have to be rectangular.

2. When moving in the maze you may only move up and down and right and left. You may not move diagonally.

The following are examples of U shaped mazes and are not considered well formed.

```
###  ###         ####### ###
#s#  #f#         #     # #s#
# #### #   or    # ### ### #
#      #         #f# #     #
########         ### #######
```

Such mazes create problems since it's more difficult to work out what's the inside and what's the outside of the maze.

Program Requirements

Your program must do the following

1] Load the maze.
2] Calculate some metrics about the maze.
3] Print the maze along with the metrics.
4] Free up all space allocated to the maze.

Please see the Benchmark Program section below to learn exactly how the maze and metrics are to be displayed.

Maze Data Structure

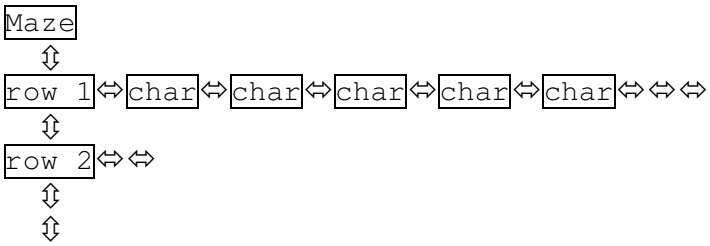
You can assume that all test cases given to your program will be well formed but you will need to check for the following problems:
1. Multiple '`s`' or '`f`' in the maze.
2. '`s`' or '`f`' declared outside of the maze.
3. '`s`' or '`f`' not declared at all.
4. Invalid characters other than spaces, `#`, `s`, `f` or `\n`.

When checking for invalid characters or `f` or `s` outside of the maze, you only have to check left, above and below the maze, not the right side. You will find this will make the assignment easier to code.

If you find a maze violation you will exit the program printing an error message stating you are unable to load the maze. Run the benchmark program below to see the exact format of the message.

PLEASE NOTE. If you have to exit your program for any reason please use `exit(0)` not `exit(1)` or any other number. This is necessary in order for the test program (see below) to work correctly.

Since the maze can be any size you ***MUST*** use STL linked lists to store it. At no stage may you use arrays within your program to store the maze. Your linked list will have the following structure. The use of the ⇔ and ⇳ characters means I'm using doubly linked lists.



As you can see, you have a linked list of rows and each row is a linked list of chars.

Maze Metrics
You will calculate two metrics about the maze:
1. The total number of branches in the maze.
2. The total number of dead-ends in the maze.

A branch point occurs when a path in the maze can go in 3 or 4 directions. If it can go in 3 directions we have 1 branch and if it can go in 4 directions we have 2 branches. The only exception is at the start point of the maze. The number of different directions we can take from the start point, minus one, is the number of branches. Therefore, we can have anywhere from 0 to 3 branches emanating from the start.

A dead-end occurs when we have a point in the maze that has only one access path. The only exception to this is if the finish or start is at a dead end. These are not included in the count.

The following diagrams illustrate.

###### #x ###### A dead-end at point x (unless point x is the start or finish). If the point x is the start of the maze then we have 0 branches	####### x ####### 0 branches at point x. If the point x is the start of the maze then we have 1 branch	# # #### # x# ###### 0 branches at point x. If the point x is the start of the maze then we have 1 branch	# # ### ### x ####### 1 branch at point x. If the point x is the start of the maze then we have 2 branches	# # ### ### x ### ### # # 2 branches at point x. If the point x is the start of the maze then we have 3 branches.

In the following example I have placed a digit to represent the number of branches at any fork in the maze and a `d` to represent a dead-end.

```
     ##########
     #d #      #############
######  # ###d#            ###
#s    2 1#f### #########   #
# #### # #     #    1 d### #
# #    # ####### ###d###   #
#  1####          #d###   ###
###    ############1    ###
  ####               #####
     ###############
```

From this, there are a total of 7 branches (including one from the start) and 5 dead ends.

Hints

To help you use the STL list object I have included a file - `exstdlist.cpp` - in the assignment directory on `/pub` and the DSA website, which you can compile and run. Study the code closely as it demonstrates how to use doubly linked lists from the STL.

Benchmark Program

To clarify how the program works, a benchmark executable version has been placed on the server. You can run it by typing the following command

```
/home/glingard/maze
```

To give you an idea of the size of this assignment, the source code for the benchmark program takes about 380 lines of code, including whitespace and comments.

The output from your compiled program must exactly math that of the benchmark.

In addition, foru files - `maze1.txt`, `maze2.txt` , `maze3.txt` and `maze4`.txt have been placed in `/pub/prprog/assign/assign2` for you to use as tests. The first three mazes are perfectly valid. The fourth maze has errors you can test for.

Assignment Objectives

The purpose of this assignment is to demonstrate competence in the following skills.

- Program design
- Using command line parameters.
- File handling.
- Link list manipulation

These tasks reflect all the subject objectives apart from objective 4.

As part of your subject workload assessment, it is estimated this assignment will take 22 hours to complete.

Queries

If you have a question, please refer it to the UTS Online discussion board where I will answer questions.

There is already an assignment FAQ in the DS & A web site. You should read this.

If serious problems are discovered the class will be informed and an update will be included in the following file

```
/pub/prprog/assign/assign2/errata.txt
```

Please keep a look out for this file.

PLEASE NOTE. If the answer to your questions can be found directly in any of the following

- subject outline
- assignment specification
- errata.txt
- UTS Online discussion board
- DS & PP web page

You will be directed to these locations rather than given a direct answer.

PLEASE NOTE. Please do not send email in HTML format or with attachments. They will not be read or opened. Only emails sent in plain text format will be read. Emails without subject lines will be automatically deleted by the junk mail filters I have in place.

Assignment Submission

You must submit your program via the `submitass2` program. I will not accept assignment submissions from any other source.

It is assumed that your C++ file is called `assign2.cpp` (although you may call it something else). You must start in a directory that is within your home directory on `rerun` and then run

```
/home/glingard/submitass2 assign2.cpp
```

This assumes that `assign2.cpp` is in the current directory.

The `submitass2` program will then perform a number of tests. These include.

Compiling
Your program must compile without fatal errors or warnings – using the `g++` compiler. Your program will be compiled with the following commands and options

```
g++ -pedantic-errors -Wall -Werror -ansi
```

You can learn what these compiler options mean by running `man g++`.

PLEASE NOTE. Your program must compile on the student UNIX server. Programs written on Windows' machines sometimes don't compile or run properly on the student server.

Style Feedback Program
Your program will be run through a style feedback program, which will check that your code meets a minimum style layout. If your program does not meet the standard a warning message will be printed. The style feedback program will display messages showing which lines of code do not conform to the standard and why they don't.

Code Feedback Program
Your program will be run through a code feedback program, which will check that your code meets minimum coding practices. If your program does not meet the standard a warning message will be printed. The code feedback program will display messages showing which lines of code are not acceptable and why they aren't.

Test Filter
Your program will be tested with 8 different sets of tests. The results of your program will be compared to the results generated using the benchmark `maze` program.

PLEASE NOTE. Make sure the output from your program is ***EXACTLY*** the same as that from the benchmark executable. ***ANY*** deviation of the output from your program to that of the benchmark executable will cause the test to fail.

Plagiarism Agreement
You will be required to agree to a statement that the assignment is your own work and that you haven't given your code to anyone else.

If all goes well, `submitass2` will reply with a message saying you have successfully submitted the assignment. It will also place in your current directory a file called `receipt.txt`.

You may submit your assignment as many times as you like. The last assignment received will be the one marked.

PLEASE NOTE. Only assignments submitted via the `submitass2` program will be accepted for marking.

receipt.txt

`receipt.txt` is your proof that you have submitted your assignment. Once you have received it, copy it to somewhere safe such as your home directory. Additionally, copy your C++ file into the same location. Do not modify them in any way. If you wish to continue developing your program then do it on a duplicate file.

The `receipt.txt` file contains three pieces of information
1. A line saying you have submitted your file and when you did it.
2. A checksum of your C++ file
3. A checksum of your receipt

If you tamper with your C++ file or the receipt then the checksums will be become invalid and therefore your receipt will become invalid. No actions will be taken if receipts have invalid checksums.

Acceptable Practice vs Academic Malpractice

- Students should be aware that there is no group work within this subject. All work must be individual. However, it is considered acceptable practice to adapt code examples found in the lecture notes, labs and the text book for the assignment. Code adapted from any other source, particularly the Internet and other student assignments, will be considered academic malpractice. The point of the assignment is to demonstrate your understanding of the subject material covered. It's not about being able to find solutions on the Internet. You should also note that assignment submissions will be checked using software that detects similarities between students programs.
- Do not let anyone submit their assignment from your account. The `submitass2` program copies your assignment into a secure directory based upon your user login name. Anyone else using your account will have their assignment placed in your directory. Students who do this will find themselves reported to the Faculty for possible academic malpractice and misuse of Faculty resources.
- Students are reminded of the principles laid down in the "Statement of Good Practice and Ethics in Informal Assessment" in the Faculty Handbook. Assignments in this subject should be your own original work. Any collaboration with another participant should be limited to those matters described in the "Acceptable Behaviour" section. Any infringement by a participant will be considered a breach of discipline and will be dealt with in accordance with the Rules and By-Laws the University. The Faculty penalty for serial misconduct of this nature is zero marks for the subject. For more information, see

http://wiki.it.uts.edu.au/start/Academic_Integrity

Assignment Security

It is important to note that you have a responsibility to maintain the security of your assignment files. You can read more details about this in the Academic Malpractice section of the DS & PP web site - *http://learn.it.uts.edu.au/dsa/*

Assessment

Marks will be awarded out of 20 based upon the following scheme.

- Between 0 and 8 marks will be awarded by the computer based on the number of tests passed. 1 mark per test passed.
  PLEASE NOTE. Some students have been known to write their code to artificially pass the tests rather than solve the assignment problem. In such cases a reduced mark will be given for the tests, including being given a 0.

The following marks will only be awarded if a score of 5 or more is awarded for the tests and both the code feedback and the style feedback programs are passed without generating warnings.

- Between 1 and 9 marks will be awarded on the quality of your code design and the algorithms used. This includes good OO design, using `const` where possible, etc.
- Between 1 and 3 marks will be awarded on the presentation style of the program. This involves meaningful variable names, intelligent use of comments and so forth.

For further details of the marking scheme please check out the assignments section of the DS & A web page. *http://learn.it.uts.edu.au/dsa/*

PLEASE NOTE. It is a fundamental requirement of this assignment that you use the linked list structure outlined in the Maze Data Structure section. Students who use arrays to store their maze will receive 0 for the assignment.

Late Assignments, Extensions and Special Consideration

Please read the subject outline regarding late assignments and extensions. If you did not get the outline a copy can be found at the DS & A web site

Assignments that are late by less than one week will incur a penalty of 1 mark for each day or part thereof late. Assignments more than a week late will not be accepted under any circumstance as the assignment solution will have been made available by then.

An extension of one week will only be granted if there is a fully documented reason which merits it. The documentation must be presented to the Subject Coordinator before the assignment due date. Extensions longer than a week will not be granted under any circumstance. If a one week extension is granted that means the assignment will be accepted up to a week after the due date without penalty. It will not be accepted later than that.

Students may apply for special consideration if they consider that illness or misadventure has adversely affected their performance in the assignment. For more information go to

http://www.sau.uts.edu.au/exams_ass/spec_cons.html

Return of Assessed Assignments

The code of your assignment will be printed out and marked. It will list marking codes for particular types and programming issues. These codes can be found in the DS & A web site in the `assessment->assignment->assessment` menu section and will give a complete break down of your marks.

The marked assignments will be returned via the Student Center on level 2 of building 10. The estimated return date is 21/5/10

Getting Marks

You can check on your marks by running the following program.

```
/home/glingard/getmarks
```

Apart from your marks this program will give the following information.

- Given out – the date the assignment is given out.
- Due date – the date the assignment is due.
- Late date – assignments will be accepted up to one week after the due date but with a penalty of –1 mark per day or part thereof late.
- Marks released – The date the marks are released.
- Close date – The assignment is closed and the solution will be released. The close date will be one week after the assignments are marked and given back.

PLEASE NOTE. It is <u>your</u> responsibility to check that I have received your assignment and given you a mark. Even if you have a receipt you should check your mark and inform me if there is any problem before the close date. The `getmarks` program allows you to do this very easily. Unless you have a valid receipt or there are exceptional circumstances (e.g. serious medical conditions) no further correspondence regarding the assignment will be entered into after the close date.