University of Technology, Sydney

# Faculty of Engineering and Information Technology
# School of Software

**31251 – DATA STRUCTURE AND ALGORITHMS**
**AUTUMN 2014**

**ASSIGNMENT 2 - DUE 11:59pm, 30th May 2014**

**This assignment is worth 30% of the total marks for this subject.**

**DRAFT – Awaiting final approval**

## Requirement

In assignment 1 you created a linked list of linked lists to store a maze. From this you printed the maze plus the following maze metrics – number of maze branches and number of dead-ends.

In this assignment you are going to change the linked lists to binary trees. Additionally, instead of printing the maze metrics you will print the maze so it shows the maze solution marked out in dots. For example

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

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 has exactly the same structure as in assignment one. Apart from the basic checks required you do not need to do any deeper checking of the maze. You can assume that all mazes given to you will be well formed – as detailed in assignment 1.

There is one further limitation to the mazes. Every path in the maze leads to a dead-end or the finish. There are no paths in the maze that will loop back on themselves. This will simplify some of the coding issues with finding a route through the maze.

**Program Requirements**

Your program must do the following

1] Load the maze.
2] Print the maze with its solution marked in dots.

Please see the Benchmark Program section below to learn exactly how the maze is 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.

Your maze ***MUST*** have the following structure

Maze object:
As a minimum, this will have a binary tree of row objects. Make sure the tree is balanced. It may contain any other data you need to make the program function properly.

Row object:
As a minimum, this will have an int y value telling which row it is. The binary tree in the maze object will use this value as the key value to store the row object. The row object will also contain a binary tree of maze point objects.

Maze point object:
As a minimum, this will contain an int x value saying where in the row the point fits. The binary tree in the row object will use this value as the key value to store the maze point. The maze point will also contain a char variable storing the type of maze point. That is, a ' ', '#', 's' or 'f'.

Since the maze can be any size you ***MUST*** use binary trees to store it. At no stage may you use arrays or linked lists within your program to store the maze. As there is no binary tree in the STL you must use the template binary tree supplied with this assignment. A copy of the

`bintree.h` and `binnode.h` files can be found in the `/pub/prprog/assign/assign2` directory and the DSA web site.

*PLEASE NOTE 1*: there has been a modification to the way the `find` function operates from the lecture notes. Study the bintree and binnode code to see how it works.

*PLEASE NOTE 2*: You may not modify any of the code in `bintree.h` or `binnode.h`

### Hints

The simplest way to search a maze is to use recursion. There are plenty of examples on the net as to how to do it. The main problem you will have is that they are done using 2D arrays rather than the tree structure required for this assignment. If, however, you have structured your objects well it is relatively straight forward to adapt the algorithms to do the job.

### 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/maze2
```

To give you an idea of the size of this assignment, the source code for the benchmark program takes about 480 lines of code, including whitespace and comments. It should take you far less lines though as you will be adapting your code from assignment 1 to do the job. The better you have structured your code in assignment two the easier it will be to adapt your code.

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

In addition, four 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.
- Binary tree manipulation
- Recursion

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 30 hours to complete.

### Assignment Errata

It is quite possible that errors or ambiguities maybe found in the assignment specification or the benchmark. If so, updates will be made and announcements made regarding the amendment. It is your responsibility to keep up to date on such amendments and ensure you are using the latest version of the Assignment Specification and Benchmark.

## 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.

*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.

*PLEASE NOTE*. Your file must include all the code you have written for the assignment.

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.

- **Testing**: 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.

- **Design**: 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.

- **Style**: 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.

*PLEASE NOTE*. It is a fundamental requirement of this assignment that you use the binary tree structure outlined in the Maze Data Structure section. Students who use other data structures 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 breakdown of your marks.

The estimated return date is 14/6/10. An announcement will be made as to where you can pick up the marked assignment.

## 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 your 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.