DEPARTMENT OF COMPUTER SCIENCE
ASSESSMENT DESCRIPTION 2014/15 (EXAM TESTS AND COURSEWORK)

MODULE DETAILS:

Module Number:	08227	Semester:	1 and 2
Module Title:	Advanced Programming		
Lecturer:	Darren McKie / John Rayner / Warren Viant		

COURSEWORK DETAILS:

Assessment Number:	2	of	3		
Title of Assessment:	Data Structure Analysis				
Format:	Program	Report			
Method of Working:	Individual				
Workload Guidance:	Typically, you should expect to spend between	55	and	65	hours on this assessment
Length of Submission:	This assessment should be **no** more than: *(over length submissions **will be** penalised as per University policy)*	2500 **words** *(excluding diagrams, appendices, references, code)*			

PUBLICATION:

Date of issue:	w/c 23 February 2015

SUBMISSION:

ONE copy of this assessment should be handed in via:	E-Bridge	If Other (state method)		
Time and date for submission:	**Time**	16:00	**Date**	8 May 2015
If **multiple hand–ins** please provide details*:*				
Will submission be scanned via TurnitinUK?	Yes	If submission is via TurnitinUK within E-Bridge students MUST only submit Word, RTF or PDF files. Students MUST NOT submit ZIP or other archive formats.		

The assessment must be submitted **no later** than the time and date shown above, unless an extension has been authorised on a *Request for an Extension for an Assessment* form which is available from the Departmental Office (RB-308) or http://intra.net.dcs.hull.ac.uk/student/exam/Advice%20regarding%20resits%20in%20modules%20passed%20by%20compe/Forms/AllItems.aspx.

A student who has submitted the wrong file to E-Bridge will still incur a late penalty if their resubmission is made after the coursework deadline.

MARKING:

Marking will be by:	Student Name

COURSEWORK COVERSHEET:

BEFORE submission, you **must** ensure you complete the **correct** departmental ACW cover sheet (if required) and attach it to your work. The coversheets are available from: http://intra.net.dcs.hull.ac.uk/student/ACW%20Cover%20Sheets/Forms/AllItems.aspx	NO coversheet required as E-Bridge submission

ASSESSMENT:

The assessment is marked out of:	100	and is worth	40	% of the module marks
N.B If multiple hand-ins please indicate the marks and % apportioned to each stage above (i.e. Stage 1 – 50, Stage 2 – 50). It is these marks that will be presented to the exam board.				

ASSESSMENT STRATEGY AND LEARNING OUTCOMES:
The overall assessment strategy is designed to evaluate the student's achievement of the module learning outcomes, and is subdivided as follows:

LO	Learning Outcome	Method of Assessment *{e.g. report, demo}*
1	*Identify and apply advanced programming concepts.*	Program, Report
2	*Develop a robust efficient and real-time application. Evaluate both these approaches and solutions to the problem.*	Program, Report
3	*Deploy tools to create, debug and optimise an application. Evaluate both these approaches and solutions to this problem in order to enhance the outcome.*	Program, Report

Assessment Criteria	Contributes to Learning Outcome	Mark
Data Structures Implementation	1, 2, 3	70
Report	1, 2, 3	30

FEEDBACK

Feedback will be given via:	Mark Sheet	Feedback will be given via:	N/A
Exemption (staff to explain why)			
Feedback will be provided no later than 4 'teaching weeks' after the submission date.			

This assessment is set in the context of the learning outcomes for the module and does not by itself constitute a definitive specification of the assessment. If you are in any doubt as to the relationship between what you have been asked to do and the module content you should take this matter up with the member of staff who set the assessment as soon as possible.

You are advised to read the **NOTES** regarding late penalties, over-length assignments, unfair means and quality assurance in your student handbook, also available on the department's student intranet at:

- http://intra.net.dcs.hull.ac.uk/student/ug/Handbooks/Forms/AllItems.aspx (for undergraduate students)
- http://intra.net.dcs.hull.ac.uk/student/pgt/Student%20Handbook/Forms/AllItems.aspx (for postgraduate taught students).

In particular, please be aware that:

- Your work will be awarded zero if submitted more than 7 days after the published deadline.
- The overlength penalty applies to your written report (which includes bullet points, and lists of text you have disguised as a table. It does not include contents page, graphs, data tables and appendices). Your mark will be awarded zero if you exceed the word count by more than 10%.

Please be reminded that you are responsible for reading the University Code of Practice on the use of Unfair means (http://student.hull.ac.uk/handbook/academic/unfair.html) and must understand that unfair means is defined as any conduct by a candidate which may gain an illegitimate advantage or benefit for him/herself or another which may create a disadvantage or loss for another. You must therefore be certain that the work you are submitting contains no section copied in whole or in part from any other source unless where explicitly acknowledged by means of proper citation. In addition, **please note** that if one student gives their solution to another student who submits it as their own work, **BOTH** students are breaking the unfair means regulations, and will be investigated.

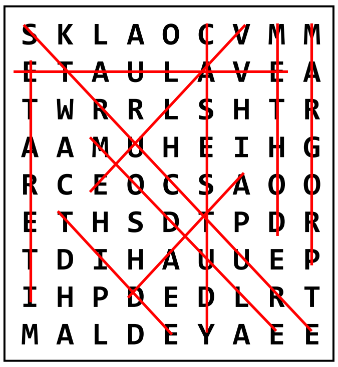
In case of any subsequent dispute, query, or appeal regarding your coursework, you are reminded that it is your responsibility, not the Department's, to produce the assignment in question.

Investigation of Alternative Data Structures for WordSearch Puzzles

Overview

The objective of this ACW is to develop two alternative data structures to represent WordSearch puzzles, and to develop two alternative data structures to represent the WordSearch dictionaries, and to investigate their efficiency.

A WordSearch puzzle comprises a rectangular (normally square) grid of letters in which a number of words are hidden, and the goal of the puzzle is to find all the words present. The number of words hidden may or may not be specified, and the search may be aided by a dictionary of words to find, or it may be that the topic context is known (e.g. computer science) and the puzzler has to use their knowledge to decide what relevant words they can find.



The words may be hidden within rows and/or columns of the grid, and possibly diagonally, and possibly each in either direction (see illustration). Words may also cross over, so one letter cell in the grid may contribute to several target words.

If a dictionary is provided to support the searching, this may contain either the target words only, or the target words together with many other words to make the puzzle harder, or the enjoyment greater, depending on the puzzler's point of view!

In this ACW, the puzzle grid can be assumed to be square, and that it is always 9x9 in size. There will also be dictionaries of words to support the search in each puzzle grid. Therefore, the objective of the work is to develop and implement relevant data structures to represent the grid and the dictionary for each puzzle, and to evaluate the performance of these data structures in operation as your program solves each puzzle.

What will be the best data structures to represent the grid, and to hold the dictionary? Two alternative structures for each aspect are proposed for investigation. *For each scenario*, you are required to develop a C++ program to implement two structures (one for the grid, one for the dictionary), and add diagnostic code to report on the behaviour and efficiency of the data structures while grid letter sequences are being compared to dictionary contents in order to discover hidden words.

Input and Output

Each grid puzzle will be in a text file called **wordsearch_grid.txt** and you are to implement code that can populate the grid of your data structure by reading the letters contained in the file. The data file consists of n * n letters, each in the range A-Z, for example see the grid on the right, corresponding to the example grid given earlier. You can assume that all letters and words provided are in uppercase.

```
S K L A O C V M M
E T A U L A V E A
T W R R L S H T R
A A M U H E I H G
R C E O C S A O O
E T H S D T P D R
T D I H A U U E P
I H P D E D L R T
M A L D E Y A E E
```

After you have solved the puzzle, you are to output a number of findings (described later) to a file called either:

results_simple_puzzle_simple_dictionary.txt or
results_simple_puzzle_advanced_dictionary.txt or
results_advanced_puzzle_simple_dictionary.txt or
results_advanced_puzzle_advanced_dictionary.txt.

These files will be in the format specified in **Appendix A**.

There will be dictionary files provided, called **dictionary.txt**.

Unseen puzzles and dictionaries will be used in the marking of this ACW and some diagnostics will be performed on your results file. Therefore, failure to use the correct file names, and failure to output findings in the correct format, will result in a loss of marks.

Structures for the Puzzle Grid

(a) The simplest structure for the puzzle grid is a 2D array of characters. This can be populated by reading from a given text data file (one line of characters per row of the grid) and traversed simply across rows (back and forth); by columns (up and down) and by diagonals (main and secondary, in either direction) by suitable use of for loops or similar. From each grid position, your program should step along each direction in turn, comparing each letter sequence against the dictionary content to see if each sequence forms a word contained in the dictionary data.

(b) A more advanced data structure recognises that any letter cell in the grid can form part of eight sequences (horizontal, vertical, and two diagonals, each of which may be read in either direction). Therefore, a data structure can be created based on individual 'letter cell' objects that are linked into sequences that can be checked uniformly by one standard comparison method. This would be called for each direction from each cell of the puzzle grid in turn, to compare the letter sequences from that point against the dictionary content.

Structures for the Dictionary

(a) The simplest structure for the dictionary is an array of strings (or std::vector of strings). This can be populated by reading from a text file (one line per word or string value) and searched systematically to match puzzle grid content as the letter sequences are traversed.

(b) A more advanced data structure recognises that many longer words may begin with the same letter sequence as (or the entirety of) some shorter word(s). For example, the dictionary may contain the words HAND and HANDS, therefore HANDS only requires one more letter matching than the word HAND. If the word matching process has to start each dictionary word string from the beginning, then such common sequences will be matched several times fruitlessly until the match is found or fails completely. A dictionary structured as a tree of letters should be more efficient during processing, although more complex to initialise.

Code Development

Code is required to define each of the above data structures, and to operate the WordSearch solving process based on each of four combinations (simple and advanced grid structures, each combined in turn with a simple and an advanced dictionary implementation). For each combination, the program must be run with each of the trial puzzles and dictionaries to be provided, and in each case the operational timings recorded by use of suitably positioned timing statements. The number of words matched, the actual words matched, the number of grid cells visited, the number of dictionary entries (or tree nodes) visited, along with the raw code timings, must be all recorded (see **Appendix A** for the format of the output).

These observations should then be evaluated and discussed, to derive conclusions as to the relative operational behaviour of the different combinations of data structures and code processes. The deliverables for the assessment will be:

- program code;
- timing and activity results for each scenario combination of data structures;
- discussion of results;
- conclusion

Code Implementation

A test harness has been provided to you. You are to use this to test your implementation. If you do not implement the functionality of a particular method listed below, then simply output to the Console (**cout**) a message stating that the particular method has not been implemented and return false (see test harness). You should assume that the given WordSearch class is not good C++, and so you will need to use Parasoft to make sure that your final code does not violate any Parasoft rules. Parasoft will be used to mark the quality of your C++ code. You are to implement the following class and class methods:

Class:

`WordSearch`

Methods:

Default constructor

Deconstructor

Any appropriate assignment operators and copy constructors

`bool ReadSimplePuzzle()` – This method will then read the puzzle and store the letters in the simple grid data structure. This method will return true if the file is read successfully, and will return false if it is not read successfully.

`bool ReadSimpleDictionary()` – This method will then read the dictionary and store the words in the simple dictionary data structure. This method will return true if the file is read successfully, and will return false if it is not read successfully.

`bool ReadAdvancedPuzzle()` – This method will then read the puzzle and store the letters in the advanced grid data structure. This method will return true if the file is read successfully, and will return false if it is not read successfully.

`bool ReadAdvancedDictionary()` – This method will then read the dictionary and store the words in the advanced dictionary data structure. This method will return true if the file is read successfully, and will return false if it is not read successfully.

`bool SolveSimplePuzzleWithSimpleDictionary()` – This method will solve the puzzle using the simple grid data structure with the simple dictionary data structure. This will return true if it has been correctly implemented, and will return false if it has not been implemented.

`bool SolveSimplePuzzleWithAdvancedDictionary()` – This method will solve the puzzle using the simple grid data structure with the advanced dictionary data structure. This will return true if it has been correctly implemented, and will return false if it has not been implemented.

`bool SolveAdvancedPuzzleWithSimpleDictionary()` – This method will solve the puzzle using the advanced grid data structure with the simple dictionary data structure. This will return true if it has been correctly implemented, and will return false if it has not been implemented.

`bool SolveAdvancedPuzzleWithAdvancedDictionary()` – This method will solve the puzzle using the advanced grid data structure with the advanced dictionary data structure. This will return true if it has been correctly implemented, and will return false if it has not been implemented.

`void WriteResults()` – This method will take a string parameter that will represent the filename of the output file. This method will output the results from the previous Solve() method to the given output file, which will be either results_simple_puzzle_simple_dictionary.txt or results_simple_puzzle_advanced_dictionary.txt or results_advanced_puzzle_simple_dictionary.txt or results_advanced_puzzle_advanced_dictionary.txt.

Constants:

`PUZZLE_NAME` – This constant holds the correct filename for the puzzle (i.e. wordsearch_grid.txt)

`DICTIONARY_NAME` – This constant holds the correct filename for the dictionary (i.e. dictionary.txt)

Deliverables

You are to submit all of your code, once it has been cleaned, to the module eBridge site.

You are also to submit a separate report, to the module eBridge site, that will contain the following sections:

- Timing and activity results (suitably tabulated) for each scenario combination of data structures;
- Discussion of results;
- Conclusion

Your discussion of results and conclusion should address the following issues:

- A brief review of the data structures you have implemented: how are they organised and how are they operated?
- For each combination of dictionary and puzzle data structure you have investigated, how does the performance compare with the other scenarios (this discussion should make reference to your timing and activity results);
- Discuss whether it is more efficient to (a) select words from the dictionary and then search for them in the puzzle grid, or (b) visit each letter in the puzzle grid and attempt to match sequences from that position against the dictionary content. To what extent are these alternative approaches influenced by the alternative data structure strategies?

Marking Scheme

A detailed marking scheme will be published a short time after this ACW description has been published. This marking scheme will contain a breakdown of all of the marks and will give you the ability to mark yourself as you develop your software and write your report.

Appendix A – Format of each results file

The following is the format of the results, and the heading names that you **MUST** have **EXACTLY** in your **output** file.

```
NUMBER_OF_WORDS_MATCHED n

WORDS_MATCHED_IN_GRID
n1 n2 WORD1
n1 n2 WORD2
etc.

NUMBER_OF_GRID_CELLS_VISITED n

NUMBER_OF_DICTIONARY_ENTRIES_VISITED n

TIME_TO_POPULATE_GRID_STRUCTURE f

TIME_TO_SOLVE_PUZZLE f
```

The headings (in capitals) MUST appear in your results file as shown above.

The **n** items denote that this represents a single integer number

The **n1 n2 WORD1**, **n1 n2 WORD2** and **etc.** items denote that these represent all of the words matched in the grid – you will list all of the words here, one word per line. The **n1** and **n2** before each matched word denotes the column/row position (starting at index 0) of the first letter of the word in the puzzle grid. E.g. **`0 2 HAND`** – denotes that the word HAND was found starting at column=0, row=2 (or x=0, y=2 of the grid).

The **f** items denote that this represents a single floating-point number in seconds