COSC1252/1254 - Assignment 1 - Data Structures

Due Date: Friday 25 August, 23:59.

Each week an exercise will be published which will help build towards this assignment called a "Lab Exercise". If you complete these exercises each week it will greatly reduce the amount of work required from you when the assignment deadline arrives.

The C++ programming language provides a range of data structures via the Standard Template Library, some are provided via the boost library and sometimes you need to write your own.

Your task in this assignment is to implement the same task using a variety of data structures then time and profile the performance of these data structures and write a report on their performance.

Throughout the first six weeks of this course we are going to explore a range of data structures in more depth than you may have in previous courses. You will need to implement a class that manages a singly linked list (this should be revision for all of you) and a class that manages a binary search tree - these are custom data structures and must be your own work. The data type that each of these will manage is a std::string.

Along with these custom types you will write, you will also use the std::vector, std::list, std::map and std::set classes.

Your program will open up two textfiles, a "dictionary" file with one word per line and a text document (such as a novel) to gather data from.

For each of:

- std::list
- std::vector
- std::multiset
- Your custom list
- Your custom binary search tree

Your program will load each line from both the dictionary file and the text file that you are testing into the selected data structure. Once this has been done you will need to tokenize each line in the text file (I recommend you use boost::tokenizer for this) and then search through the dictionary list for the word. If it is there, you should save the word to a text file as specified from the command line. Then you compute the edit distance between this word and each of word in dictionary list, and find the closest fussy matches. You should save the closest fuzzy matches of this word to the text file specified as well. The details on edit distance and its calculation can be found from the following link:

https://nlp.stanford.edu/IR-book/html/htmledition/edit-distance-1.html

Please note that the edit distance algorithm takes a long time to run (think about its runtime complexity) and so you are advised to test this on smaller files first.

Example output to the output file when a word is found:
apple: 12
or
apple, 12

Where the number is the count of matches for that word in the file.

Example output to the output file when a word is not found:

appld was not found in the dictionary. The closest matches were: apple.

Note: there may be more than one closest match so output all of them.

You will also be required to write a several page report in which you compare the performance of these data structures in detail with an aim to select the best data structure for the job and identifying the bottleneck in the code with a view to recommend changes for a performance increase.

Note: your are not required to use inheritance in this program as we have not covered it but it might simplify your implementation if you do.

All use of C function calls is forbidden in this assignment unless there is no available alternative in modern C++. If you assert there is not but I find an alternative that you would be reasonably expected to know about (such as via a web search or via the presented course materials), you will lose marks.

You are not allowed to use gotos in your code and you should avoid global variables unless what you are trying to do can't be done otherwise (this is very unlikely).

You are required to use solely modern standards and methods for implementation of your program (we will be discussing this early in the course) and you should aim for a well-designed, modern object oriented solution to the problem. In particular what this means is that use of the new keyword will be penalised (use the unique_ptr class) as will particularly poorly performing implementations as these are issues we will be discussing. For example, storing pointers where object literals will do will make your program run slower. Keep your use of pointers to a minimum as they are poisonous to modern cpu caching.

You have no right to assume that operations that you request have actually occurred. All I/O operations in particular need to be validated. You need to be thinking about what a user may do wrong and validate for that.

Having said that, don't over-validate. For example, if you validate for the text file and dictionary existing in the current directory and I load it from a different location and your program rejects this, you will be penalised. Do not make any assumptions about the operating system that the final user may be using either.

Your program must compile with the compiler flags -Wall -pedantic -std=c++14. If you wish to play around with c++17 features I will allow this via the -std=c++1z flag however you must test your program and ensure it runs in a stable way on the server.

You program should be portable and run without errors or warnings or memory problems on any modern platform. As such you are wise if you validate for memory issues using tools like valgrind and address sanitization. (If you don't know what these are, please ask). If your program shows unspecified behaviour or other bugs, this will be grounds for a reduction in marks.

We want professional, well-designed well-commented solutions that you would be happy to show a prospective employer.

Compiling on the Linux Servers

You are required to compile your code specifying the c++14 standard. When using g++ on titan, jupiter and saturn, this means using as a minimum the compiler flags –Wall –pedantic –std=c++14 (or –std=c++1z if playing with c++17 features). Please note that there are some incompatibilities that arise when moving your code between systems and you are advised to test run your program on the servers. How the program runs on your machine is not our concern. You may develop your code wherever you wish however it MUST compile and execute correctly in the target environment (the linux servers provided).

In order to compile in the linux environment you will need to run a devtoolset script – run the following after logging in to the linux servers:

source /opt/rh/devtoolset-6/enable

Which will give you access to g++ version 6 which supports the –std=c++14 flag.

Marking Criteria

Command line arguments (3 Marks)

In this requirement, you will need to parse command line arguments. The main program will be run as follows:

./test_datastructures -s datastructure -d dictionary -t textfile -o outputfile

These arguments should have the following values:

- Data structure could be one of:
    - list : use the std::list class to hold the dictionary and the document you are processing
    - vector : use the std::vector class to hold the dictionary and the document you are processing
    - set : use the std::multiset class to hold the dictionary and the document you are processing
    - custom_list : use your custom list class to hold the dictionary and the document you are processing
    - custom_tree : use your custom binary search tree class to hold the dictionary and the document you are processing
- Dictionary is the dictionary file that you will load in first. I will provide you an example file.
- Textfile is the text file that you are processing.
- Outputfile: the file name to save the list of words without matches and their fussy matches to.

Arguments may be specified in any order but all arguments must be specified. We recommend you use the boost::program_options library as this is a good cross-platform way to achieve this task.

Edit Distance implementation (5 Marks)

You are to implement a function that operates over two std::string strings. This must be done in an object-oriented manner. Marks will be deducted for undefined behaviour (bugs) or for invalid memory accesses. Marks will also be deducted for incorrect implementation.

Linked List implementation (8 Marks)

You are to implement a class that manages a singly linked list of std::string. This must be done in an object-oriented manner. Marks will be deducted for undefined behaviour (bugs) or for invalid memory accesses. Marks will also be deducted for incorrect implementation.

BST Implementation (8 Marks)

You are to implement a class that manages a binary search tree of std::string. This must be done in an object-oriented manner. Marks will be deducted for undefined behaviour (bugs) or for invalid memory accesses. Marks will also be deducted for incorrect implementation. Your implementation of a binary search tree must allow for the storing of duplicates.

Read files Into Memory (4 marks)

You are to read both the dictionary file and the text file to be processed into memory and store each line in the currently selected data structure. Marks will be deducted for invalid memory accesses, for lack of validation of the input files, etc.

Text files read into memory should be tokenized using the following characters as delimiters:
" 1234567890!@#$%^&*()_+=[{}]\\|;:'\"<>,./?"

On google drive I have provided examples of text files and dictionary files.

Implementation of the word counting algorithm (6 Marks)

You are to implement the algorithm to iterate over the lines read in from the text file.

For each line, tokenize into words, check if the word is in the dictionary list and if it is, add to the map of word counts. We recommend you use boost::tokenize for this component.

Characters that delimit words include all whitespace, punctuation and numbers.

Please ensure you validate all actions taken are validated. You cannot assume that just because you make a request that it succeeded.

Use of std::list (5 Marks)

Implement the component of the system that manages both the dictionary list and textfile line list via std::list objects. You could do this using inheritance however that is not required as will not have covered that by the assignment due date.

Use of std::multiset (5 Marks)

Implement the component of the system that manages both the dictionary list and textfile line list via std::set objects. You could do this using inheritance however that is not required as will not have covered that by the assignment due date.

Use of std::vector (5 Marks)

Implement the component of the system that manages both the dictionary list and textfile line list via std::vector objects. You could do this using inheritance however that is not required as will not have covered that by the assignment due date.

Use of custom list (5 Marks)

Implement the component of the system that manages both the dictionary list and textfile line list via your custom list class. You could do this using inheritance however that is not required as will not have covered that by the assignment due date.

Use of custom tree (5 Marks)

Implement the component of the system that manages both the dictionary list and textfile line list via your custom tree class. You could do this using inheritance however that is not required as will not have covered that by the assignment due date. You should allow for duplicates in your tree implementation.

Save word counting results to a text file (4 marks)

Once you have iterated over all the lines of the text file and stored the count of words in a std::map, you will need to save this data to a text file. This file should be a csv file where the first element is the word, followed by a comma followed by the number of times that word was encountered. You will need to validate all output requests and ensure that there are no invalid memory accesses or other bugs.

Good object oriented design (6 Marks)

You must show sensible design decisions in how you have designed the classes for your program.

Modern C++ Solution (6 Marks)

Your solution must be object oriented which means

a) your choices of classes are sensible
b) No parts of your solution are procedural
c) You must not use any C functions in your solution unless you can show there is no equivalent in modern C++.

Makefile / Cmake (5 marks)

You must write a Makefile or a cmake file to be used with compiling your program. You must compile each object file separately and then link them together. You must pass the appropriate flags into the appropriate phase of compilation.

Coding Conventions and Practices (5 Marks)

- Avoiding global variables unless they are necessary.
- Avoiding goto statements.
- Consistent use of spaces or tabs for indentation. We recommend 4 spaces for every level of indentation. Be careful to not mix tabs and spaces. Each “block” of code should be indented one level.
- Keeping line lengths of code to a reasonable maximum such that they fit in the default terminal screen width (80 characters wide).
- Commenting (including function header comments).
- Complete author identification on all files.
- Appropriate identifier names.
- Avoiding magic numbers.
- Avoiding platform specific code.

Demonstration (5 Marks)

- You must demonstrate your program in labs before the assignment due date.

Report (20 Marks)

You are to write a report of 5 pages or less that compares the performance of the data structures used in this assignment. You should time how long each one takes to run and report those timings and compare. For full marks you should use software such as Kcachegrind to profile the amount of time your program spends in each function. As part of your report you should specify the fastest data structure, why it is the fastest and how you might speed up the execution of your programs if you were to make further modifications. This component of the assignment must be submitted via a turnitin assignment so we may check that this is your original work. An acceptable assignment should show a green square for the level of originality.

This report must be submitted separately via the provided turnitin submission link on blackboard.

Penalties

Marks will be deducted for the following:

- Incorrect compiler flags - 5 mark penalty
- Compile errors and warnings - a 5 marks deduction for compiler warnings, a 10 mark penalty for small compiler errors that your marker can fix quickly. If we cannot get your program to compile, your program will get a maximum of 40% of the marks available for the assignment.
- Fatal runtime errors such as segmentation faults, bus errors, infinite loops, etc. Sections affected will get a maximum of 40%. If there are components we cannot run due to fatal runtime errors then those sections will also get a maximum of 40% of the marks available for that section even if the code is correct. It should be runnable and testable.
- Missing files (affected components get zero marks).
- not filling-in the readme file or not providing a readme file - 2 marks penalty.
- Not identifying yourself in all files submitted - this is your work so take ownership! 2 mark deduction for this.
- Late submission: your mark will be reduced by 10 marks for each day late up to a maximum of 5 days. Submissions later than 5 days will not be accepted.

Extensions of Time

You may apply for an extension of time by emailing the instructor at least one week before the due date but please note that I will only give extensions for extenuating circumstances. Common requests for extensions such as “My hard disk crashed and I lost all my work.” won’t be accepted and we advise you to keep an up to date backup of all relevant source files. Likewise, applications for extensions based on working overtime or having a heavy study load will not be granted.

Applications for extensions must be sent to the lecturer via email at seven days before the due date. All extensions of time will comply with university policy as outlined here: https://www.rmit.edu.au/students/student-essentials/assessment-and-exams/assessment/adjustments-to-assessment

Assignment Submission

Your assignment will be submitted in two parts - your source code which will be compressed into a zip file, and a pdf of your report. The report will be submitted via a turnitin assignment submission and it is your responsibility to ensure that the report shows a low similarity level (you should see a green square).

When/how do I get my marks?

Assignment marks will be made available within 10 working days of the final submission deadline. An announcement will be made on Blackboard when marks are available. An announcement will also be made with regards to what you need to do if there are any mistakes in your marks.

Help!

Please utilise the following with regards to getting help for your assignments:

- For general assignment questions, please use the Blackboard discussion board. That way all students can see all questions once.
- Please do not post large pieces of code to the Blackboard discussion board for plagiarism reasons. If you need help with your code, send me an email.
- You are generally welcome to bring assignment-related questions to class.
- If you are having problems that may prevent you from completing your assignment on time, please talk to someone as early as possible. If you find any problems with the assignment specifications (such as mistakes or bugs), please post your queries to the Blackboard discussion board.