# Brief for Assignment 1 and 2

## CSP2104 Object-oriented Programming with C++

# Overview

**Due date**:

**See Canvas for due dates for Assignment 1 and 2**

# Your task for Assignment 1 and 2 is create an application that loads an English dictionary and then performs certain tasks using that dictionary. The assignment weightings are as follows:

* Assignment 1: 20% of the unit mark
* Assignment 2: 30% of the unit mark.

# The assignments contain tasks of varying complexity and students are expected to consult sources outside of the unit materials in order to accomplish them.

# General

Whilst you can discuss aspects of the assignment with your colleagues, you cannot share your solutions, even for reference/guidance purposes. All material submitted for marking must be your own work. Plagiarism and other forms of academic misconduct will be dealt with as per the relevant ECU policy.

# Submission Instructions

The assignment submission should be made via the units assignments section. Penalties will be applied for late submission.

For each assignment, your submission must be a single zipped file containing your entire visual studio project. The file name should be in the format of <yoursurname\_studentnumber.zip>. For example, McGowan\_1001234.zip.

Your submission must be uploaded before or on the due date unless you have an approved extension. Extensions must be applied for in writing before the due date as per University policy.

Also ensure that you keep a backup copy of all documentation and program files.

Submit your work as a visual studio project. Make sure it compiles.

# Assignment 1

Write a program that when executed will parse the dictionary file, **dictionary2022\_S1.txt**, provided with this assignment (See Appendix at the end of this document for file layout). Your program will load the information for each word (name, type and definition) from the file into a new instance of a **Word** **struct** and add this instance to a STL vector of **Words** called **Dictionary**. Your software will then display a menu of tasks it can perform. The program will then show a menu of tasks and prompt the user to enter a number corresponding to a set task to perform. Each task should be implemented in a separate function, which receives the Dictionary and performs the operation specified by the task.

The tasks for Assignment 1 are as follows:

Basic Tasks

1. Prompt the user to enter a word. If that exact word is in the dictionary, print the word’s name, followed directly on the same line by the word type using the scheme in Table 1, and the word’s definition.

If the word is not in the dictionary print ‘word not found.’

Table 1: Print the word type using this scheme.

|  |  |
| --- | --- |
| Word type: | To be printed: |
| Noun | (n.) |
| Verb | (v.) |
| Adverb | (adv.) |
| Adjective | (adj.) |
| Preposition | (prep.) |
| Miscellaneous | (misc.) |
| Proper Noun | (pn.) |
| Noun and Verb | (n. v.) |

1. Find the word(s) that contain a given character a given number of times or more. The user should be prompted for the character they wish to find, then prompted for the minimum number of occurrences of that character. This should be follows by the program printing out all the words that include that character the given number of times or greater (just print the name of the word, not the type or definition).
2. List all the words in the dictionary that have a 'q' without a following 'u' eg 'Iraqi'

Once a task has been executed, the program should return to the main menu, allowing the user to specify another task.

## Program architecture for Assignment 1

* Create a *'Word'* struct to hold each word, with fields:
  + *name* (string) The word in the dictionary
  + *type* (string)
  + *definition* (string)
* The *main* function should declare an STL Vector of Word instances.
* Each task should be performed in a function outside of the *main* function.

## Documentation Requirements for Assignment 1

Each module (struct and function) should have a ‘prologue’ – a comment block summarising the function of the module, its input and output, who it was written by, and a date of creation. Within a module, comments should be included to explain what is happening in areas where it’s not obvious by looking at the code itself.

## Assignment 1 Marking Guide

Earn marks

|  |  |  |
| --- | --- | --- |
|  | **Marks** | **Note** |
| Word struct and Vector of Words correctly implemented | **4** |  |
| Dictionary file loaded and parsed correctly. | **4** |  |
| 3 Basic Tasks | **12** (4 marks each) | Each task in a separate function, with *appropriate* passing of Dictionary vector to each function. |

Lose marks

|  |  |  |
| --- | --- | --- |
| **Problem** | **Marks (up to)** | **Note** |
| Poor programming practice. | 5 | Lack of commenting  Magic numbers instead of constants  Poor variable and function names |
| Input validation fails | 5 | I will try to break your program with dodgy input. |
| Program crashes | 5 | If your program crashes during execution then you will lose marks. |

# Assignment 2

An extension of Assignment 1. First, you will re-factor your program to be more object-oriented, then you will implement some additional functionality.

1. Convert you **Word** **struct** from Assignment 1 into a **Word class**. Encapsulate the appropriate functionality of a Word into the **Word** class, including getters, setters and constructor and the a *printDefinition* function that prints that words name, type and definition according to the output requirements that were given in Assignment 1.
   1. The Word class should be defined and implemented in its own file (eg. Word.h) – for this assignment there is no requirement for a separate Word.cpp file, and you may include your implementation of Word in the header (.h) file.
2. Create a class called **Dictionary**. Use the Dictionary class to encapsulate the data and functionality related to the dictionary.
   1. The Dictionary class should be defined in a file called Dictionary.h and implemented in a file called Dictionary.cpp
   2. The Dictionary class should hold and maintain the STL Vector of Word objects.
   3. The Dictionary class should be responsible for loading the dictionary file using an appropriately named method.
   4. The functions from Assignment 1 should be encapsulated as member functions (aka methods) into the Dictionary class.
3. Once the Assignment 1 functionality is working in the new Assignment 2 program architecture, create a new class **ExtendedDictionary**, using inheritance from the **Dictionary** class.
   1. **ExtendedDictionary** should use the functions from its parent class to load and manage the Vector of **Words**, and perform Assignment 1 functionality (ie. do not re-implement these in **ExtendedDictionary**)
4. Add new functionality to the ExtendedDictionary class, allowing the following tasks to be selected from the program main menu:

Basic Tasks:

1. List all words in the dictionary that are palindromes. e.g .”civic”
2. Prompt the user for a word, and report all words that are anagrams of the word (e.g. “admirer” and “married”)

Intermediate tasks

1. Guessing game – present the definition of a random noun and the length of that noun and ask the user to guess that noun, giving three tries. If the user guesses right, print a congratulatory message. After the first incorrect guess, reveal the first letter of the word, after the second incorrect guess reveal the second letter, after the third guess, print a commiseratory message and print the whole word.

Advanced task

1. Cheat at Wordle (see: <https://www.nytimes.com/games/wordle/>). Design, implement and test functionality that will allow the user to search for all five-letter words in the Dictionary that meet a set of parameters:
   1. Do not contain any of the letters in a user-specified set (i.e., the grey letters in Wordle),
   2. Contain the entire set of letters in a second, user-specified, set, anywhere in the word) (i.e., the yellow/orange letters in Wordle),
   3. Contain the entire set of letters in a third user-specified set at positions specified for each of the letters (i.e., the green letters in Wordle).
   4. The task should be implemented as a new method in the ExtendedDictionary class, however Word objects should be responsible for determining what letters they have, don’t have, and whether they are five letters long.
   5. For this task it is up to you to design a means of interaction with the program and how the results are displayed, aim for an interface that is easy to use by someone proficient with a console interface or menu style interface. **Assume the user has never heard of Wordle before running your program**.

## Documentation Requirements for Assignment 2

Each module (class, function) should have a ‘prologue’ – a comment block summarising the function of the module, its input and output, who it was written by, and a date of creation. Within a module, comments should be included to explain what is happening in areas where it’s not obvious by looking at the code itself.

## Assignment 2 Marking Guide

Earn marks

|  |  |  |
| --- | --- | --- |
| **Assignment 2** | **Marks** | **Note** |
| Word and Dictionary class implemented correctly. | 4 | Each class to encapsulate the required functionality and be implemented in the required files. |
| ExtendedDictionary correctly implemented through inheritance. | 2 |  |
| 2 basic tasks | 8 |  |
|  |  |  |
| 1 Intermediate task | 6 |  |
| 1 Advanced task | 10 | Methods appropriate to the task. Instructions/interface appropriate for the target user. |

Lose marks

|  |  |  |
| --- | --- | --- |
| **Problem** | **Marks (up to)** | **Note** |
| Poor programming practice. | 5 | Lack of commenting  Magic numbers instead of constants  Poor variable and function names |
| Input validation fails | 5 | I will try to break your program with dodgy input. |
| Program crashes | 5 | If your program crashes during execution then you will lose marks. |

# Appendix 1: Format of the dictionary file:

Notes about dictionary.txt

* Text format (ascii)
* 3 lines per definition
  + Line 1: Word type
  + Line 2: Word name
  + Line 3: Word definition - The definition(s) (all as one line)
  + Separator line, containing “\*\*”, not part of the actual word.
* Word
  + Only uses characters a-z and the hyphen ‘-‘
  + No words are presented with spaces, the words are joined OR a hyphen is used. e.g. “bumble bee” is “bumblebee”
  + ALL words are in lower case, even proper nouns.
* Type, a single word (see table 1)
* Definition
  + Multiple definitions of the same word separated by semicolon followed by two spaces (semicolon followed by one space is just a semicolon).

The definitions were not written by your lecturer or ECU. We take no responsibility for any inaccuracies or the content.

* + The definitions are from the GCIDE project, made available under the terms of the GNU general Public License, GCIDE\_XML is necessarily also published under those terms. See the file gpl.txt or <http://www.gnu.org/copyleft/gpl.txt>.

Table Types present in the dictionary and the key Words used to denote them.

