# 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 unit’s 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, Muralitharan\_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 display a menu of tasks it can perform. The program will then prompt the user to enter a number corresponding to a set task to perform. Each task should be implemented in a separate function and once complete the menu should be displayed again so that the user can choose another task to be performed. The tasks are described as follows:

**Task 01**

Load and Parse a dictionary file. First ask the user to either enter the file name or use the ‘default’ file, that being **dictionary\_2023S1.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** (details of this struct provided after description of tasks) and add this instance to a STL vector of **Words** called **Dictionary**.

**General Functionality of Task 02 onwards**

Each task to be implemented as a function that receives the **Dictionary** STL vector as input. If no dictionary is loaded the task should just print ‘No dictionary loaded’ and return to the menu, otherwise the functionality should be as follows:

**Task 02**

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

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

Table : Print the word type using this scheme.

|  |  |
| --- | --- |
| Word type: | To be printed to the console: |
| Noun | [noun] |
| Verb | [verb] |
| Adverb | [adverb] |
| Adjective | [adjective] |
| Preposition | [preposition] |
| Miscellaneous | [miscellaneous] |
| Proper Noun | [proper noun] |
| Noun and Verb | [noun and verb] |

**Task 03**

Find (and print to the screen) all words that contain more than three ‘z’ characters in them.

**Task 04**

Add a word to the dictionary. Allow the user to enter the name, type and definition of a word. If the word is already in the dictionary, the program should print ‘error: word exists, elevated privileges required to edit existing words’. If the word is not in the dictionary, and providing a valid ‘type’ is entered – create a corresponding **Word** struct instance and add it to the **Dictionary** STL vector.

Following the successful addition of the word, save the Dictionary to a new file (file name specified by the user), in the same format as the original dictionary file (so that it can be read using Task 01).

## 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** |
| Menu, Word struct and Vector of Words correctly implemented | **4** |  |
| Task 01 | **4** | Dictionary file loaded and parsed correctly. |
| Tasks 02 - 04 | **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 should work when user enters both lowercase and capital letters for their entries. |
| Program crashes | **5** | If your program crashes during execution then you will lose marks. |
| Program unable to compile | Maximum assignment mark |  |

# 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, constructor and the *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:

**Task 05** - Palindromes - List all words in the dictionary that are palindromes. e.g .”civic”

**Task 06** - Rhyming words – prompt the user to enter a word and print all words from the dictionary that end in the same sequence of three letters as the entered word.

Intermediate tasks

**Task 07** - Guess the fourth word – Your program should choose a random word from the dictionary, one that has more than 4 words in its definition. The program should initially welcome the player and print the current high score. Then the program should print the word and the definition, with the fourth word of the definition replaced by underscores. The player then has to guess the missing word. If the player succeeds, the program should print ‘congratulations’ and continue by presenting another word and definition. If the player guesses incorrectly, the program should print a message informing the player of the correct word and the players score, the game then finishes. The player should receive 10 points for each correctly guessed word and the program should keep a record (which should be preserved if they exit the game and attempt other tasks) of the highest score, with the player informed when they have beaten it.

Advanced task

**Task 08** - Cheat at Searchdle (see: https://wtfseo.com/searchdle/). Design, implement and test functionality that will allow the user to search for all words in the Dictionary that meet a set of parameters:

* 1. Have the same number of letters as the ‘mystery’ word.
  2. Do not contain any of the letters in a user-specified set (i.e., the grey letters),
  3. Contain the entire set of letters in a second, user-specified, set, anywhere in the word) (i.e., the yellow/orange letters),
  4. 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).
  5. 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 how long they are.
  6. 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 the game before running your program (ie. give some instructions)**.

## 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 should work when user enters both lowercase and capital letters for their entries. |
| Program crashes | 5 | If your program crashes during execution then you will lose marks. |
| Program unable to compile | Maximum assignment mark |  |

# Appendix 1: Format of the dictionary file:

Notes about dictionary2023S1.txt

* Text format (ascii)
* Initial header with dictionary information and blank line
* Each word entry proceeded by “<word>” on its own line.
* Each word entry followed by “</word>” on its own line.
* 3 lines per word
  + Line 1: Word name
  + Line 2: Word definition - The definition(s) (all as one line)
  + Line 3: Word type
* Word name
  + 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).
* Outside of the <word> </word> construct, anything goes – ie. the next word may not be immediately after the previous one.

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.

|  |  |
| --- | --- |
| Type abbreviation in file | Meaning |
| v | verb (“run”, “jump”) |
| n | noun (“cat”, “dog”) |
| adv | adverb (“slowly”) |
| adj | adjective (“big”, “glowing”, “inexpensive”) |
| prep | preposition (“beneath”, “against”) |
| pn | proper noun (“Perth”, “Edith Cowan”) |
| n\_and\_v | This word is a noun and a verb (“Rain”, “Phone”) |
| misc | other words e.g. “shh”, “and”, “but” |