# Assignment

## CSP2104 Object-oriented Programming with C++

# Overview

**Due date**:

Part 1: **Monday, Week 7: 8th April 2019 (9AM)**

Part 2: **Monday, week 12: 27th May 2019**

# This assignment is to create an application which loads an English dictionary and then performs certain tasks using that dictionary. The assignment is worth 40 marks (or 40% of the total mark for the unit). The assignment contains tasks of varying complexity and students are expected to consult sources outside of the unit materials in order to accomplish them.

# Related Learning Outcomes from Unit Outline

# On completion of this assignment students should be able to:

# Formulate computer algorithms using the operations, control structures and classes provided in the C++ programming languages;

# Write, test and debug computer programs written in C++;

# Design and implement a class library as an abstraction, using the facilities of the C++ language and environment;

# Implement efficient exception handling mechanisms in user-defined classes;

# General

Whilst you are allowed to discuss aspects of the assignment with your colleagues, 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.

Your assignments will be processed with plagiarism detection software.

# Submission Instructions

The assignment submission should be made via the Unit’s Blackboard assignments site. Penalties will be applied for late submission.

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, smith\_1001234.zip.

Your submission must be uploaded to BlackBoard 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.

# Part 1

Write a program that when executed will parse the dictionary file, **dictionary2019.txt**, provided with this assignment. It will load each record from the file into a new instance of a **Word** class and add this instance to a list of Word objects contained in an instance of a **Dictionary** class. Your software will then display a menu of tasks it can perform. It will then prompt the user to enter a number corresponding to one of the following menu items:

Basic Tasks

1. Prompt the user to enter a word. If that exact word is in the dictionary, print the word, followed by its definition (on the next line). If the word is not in the dictionary print ‘word not found.’
2. Find the word(s) with more than three ‘z’’s
3. List the words that have a 'q' without a following 'u' eg 'Iraqi'

# Classes to Create (for Part 1) DONE

This is to help you get started with your dictionary. Words in italics indicate what you MUST call your classes, methods or fields if you wish to receive marks for your effort.

Create a *'Word'* class

* Fields:
  + *word* (string) The word in the dictionary
  + *definition* (string)
* Methods:
  + Getters for *word, definition and type*
  + *printDefinition – print the word’s definition to the console in accordance with the requirement for basic task 1.*

Create a ‘*Dictionary’* class which

* maintains an appropriate STL container of Word objects (or references)
* loads the dictionary file into its array of Word objects
* Performs the tasks needed for this assignment.
* provides the methods:
  + *loadDictionary*() - (loads the dictionary file into its array of Word objects)
  + *Other appropriate methods so as to implement the tasks.*

**Appendix 1 at the end of this document outlines the structure of the dictionary file.**

# Part 2

An extension of Part 1. When executed your program will parse a dictionary file. It will load each record from the file into a new instance **of an appropriate class**.

Your software will then display a menu of tasks it can perform. It will then prompt the user to enter a number corresponding to one of the following menu items:

Basic Tasks, in addition to those from Part 1:

1. List all words that are a noun and a verb e.g. “Phone”
   1. Must use inherited and correctly overridden isVerb() and isNoun() methods to differentiate verbs and nouns. Within my load function I classify based on its type but Im not sure how to then get this to print
2. List all words that are palindromes. e.g .'civic' something to do with sort function and reverse

Intermediate tasks

1. Prompt the user for a word, and report all words that are anagrams of the word (e.g. “admirer” and “married”) something to do with sort
2. 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. After the first incorrect guess, reveal the first letter of the word, after the second incorrect guess reveal the second letter. Unsure how to do this

Advanced tasks

1. Fun with Tri-Grams
   1. In a separate file, create a class that encapsulates the following functionality: From any given text document (you can use the dictionary), calculate the probability of any character occurring after any combination of two characters (including where the first character is a space, i.e. the start of a word) and store these probabilities. Based on an input of two characters, return the three most likely characters to occur after the two characters. Use the class to generate 10 random words which sound like English words, but do not exist in the dictionary Unsure how to do this;

# Classes to Create

This is to help you get started with your dictionary, additional classes will need to be created for some of the tasks. Words in italics indicate what you MUST call your classes, methods or fields if you wish to receive marks for your effort.

Create a *'Word'* class

* Fields:
  + *word* (string) The word in the dictionary done
  + *definition* (string) done
* Methods:
  + Getters for *word* & *definition* done but can you check them
  + *isPalindrome()* which return true if the word is a palindrome
  + *isVerb();* returns false from objects of the Word class. done but can you check them
  + *isNoun();* returns false from objects of the Word class. done but can you check them

Create the following classed which inherit from *Word – override the appropriate methods (eg. isVerb, is Noun)*

* *Noun* done but can you check them
* *Verb* done but can you check them
* *Adverb* done but can you check them
* *Adjective* done but can you check them
* *MiscWord* done but can you check them

Create a P*reposition* class which inherits from *MiscWord* done but can you check them

Create a *ProperNoun* class which inherits from *Noun* done but can you check them

* The getter for the *word* field when called for this class should return a word with the first letter capitalised Solve with isUpper or something?

Create the *NounAndVerb* class which inherits from *Noun* and *Verb* using multiple inheritances – this can be tricky, see lecture notes on multiple inheritance; I’ve had a go at this but can’t get it working It currently crashes my program but if you comment it out the program runs fine

* isVerb() and isNoun() methods will return true

Create a *Dictionary* class which

* maintains an array (or appropriate STL container) of Word objects (or pointers). done
* loads the dictionary file into its array of Word objects done
* performs the assignment tasks
* provides the methods:
  + *getTotalNumberOfWords*() done
  + *loadDictionary*() - (loads the dictionary file into its array of Word objects) done

**Dictionary Class**

The Dictionary class should maintain an array (or appropriate STL container) of Word objects (or pointers to them). Because the *Noun, Verb, Adverb, Adjective, Preposition MiscWords* and *ProperNouns* classes all share *Word* as a base class, they can all be stored into this array. This is a fundamental advantage of using Object Orientation.

You may like to define a MAX\_WORDS constant to use as the maximum size of this array, and store a separate field which keeps track of how many array positions have actually been used.

A findWord(string) method should exist which returns a *Word* object for the string passed in; or null if the word does not exist

**The Definition Accessor (Getter)**

The getDefinition() should be overloaded in every class so that a type string appears in brackets before the definition. eg:

*myDctionary.findword(“diagraph”).getDefinition()*

would return an “(n.)” prepended to the definition as follows:

(n.) A drawing instrument combining a protractor and scale.

Type strings by class:

Noun (n.), Verb (v.), Adverb (adv.), Adjective(adj.), Preposition(prep) MiscWords(misc.),ProperNouns (pn.), Word(?),NounAndVerb (n. v.)

# Marking Guide

Earn marks

|  |  |  |
| --- | --- | --- |
| **Part 1** | **Marks** | **Note** |
| Word class and Dictionary class correctly implemented. | 2 |  |
| Dictionary file loaded and parsed correctly. | 2 |  |
| 3 Basic Tasks | 6 |  |
|  |  |  |
| **Part 2** | **Marks** | **Note** |
| Word Classes and inherited child classes | 4 | Correct use of Object Orientation |
| Correct implementation of isVerb and isNoun | 4 | Virtual method correct in all classes! |
| 2 basic tasks | 2 |  |
| 2 intermediate tasks | 10 |  |
| 1 advanced task | 10 |  |

Lose marks

|  |  |  |
| --- | --- | --- |
| **Problem** | **Marks (up to)** | **Note** |
| Poor programming practice. | 5 | Lack of functions  Lack of commenting  Poor indentation  Magic numbers instead of constants  BaDcAmElCaSe  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

* 106,184 word definitions
* Text format (ascii)
* 4 lines per definition
  + The word
  + The definition (on one line)
  + Type
* Word
  + Only uses characters A-Z a-z and the hyphen ‘-‘
  + No punctuation or similar
  + Abbreviations are given without the ‘.’ For example, “e.g.” would be “eg”
  + No words are presented with spaces, the words are joined OR a hyphen is used. e.g. “bumble bee” is “bumblebee”
  + The language conversions are made eg: é=e æ=ae ö=o
  + No word has more than 45 characters
  + ALL words are in lower case, even proper nouns.
* Definition
  + Multiple definitions are separated by a semicolon;
  + No Definition has more than 6014 characters
  + The language conversions are made eg: é=e æ=ae ö=o
* Type, a single word in one line of text;



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>.