# CO2411: Software Engineering Practices (2018/19)

# Assignment: Using Libraries and Debugging

**Date Issued:** 12/11/2018 **Hand in Date:** 15/12/2018

**IMPORTANT**

* As work is submitted on-line, **the deadline is midnight on the hand in date**. Do not leave it to the last minute. However, if problems occur, email the zip file to me before the deadline.
* **Read the marking scheme carefully**. Some of the later sections do not depend on your completing all the earlier sections, so don’t stop just because you find something difficult.
* **This is an individual project** and no group work is permitted.

## Assignment Overview

In this assignment, you will

* Use Visual Studio to create and use static and dynamically-linked libraries.
* Demonstrate that you can debug using the Visual Studio debugger and other approaches.

**Problem Description**

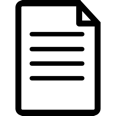
Data (“signals”) from a device such as a microphone can be processed (“filtered”) in various ways, for example it can be smoothed to reduce the effect of random noise or limited to prevent the signal outside a specified range.

This assignment is about a prototype system that can process signals using different algorithms (“filters”) each stored in a different DLL. Figure 1 gives an overview of the signal processing system.

**Figure 1: An overview of spectrographic data processing application**

**microphone**

**Input file with measurements monitor**



**ASSIGNMENT SCOPE**

Main Application

**STATIC LIBRARY**

**Functions to**

1. Read input file & store data in array
2. Use a DLL to filter data
3. Output data to a file

**DLL - A**

Filter using Algorithm A

**DLL - B**

Filter using Algorithm B



Signals are stored in a text file (one integer per line). Your prototype application will use a static link library (SLL) containing functions

* to read the data from the input file and store the measurements in an array,
* to load a Dynamically-Linked Library (DLL) by name and to use the function it contains to process the measurements in the array.
* to output the data to a file.

Example of format of input file

7 // number of data items. Note: the number of data items can vary in different files

-100 // the first data value

-50 // the second data value

-24

-80

0

25

70 // the final (seventh) data value

**Assignment Details**

You will develop a statically linked library (SLL) that contains functions to load data from a file and save the data to a file. It contains a function, named UseFilter to load a DLL by name and to use the function the DLL contains to process the data in an array. The signature of the functions in the library are defined below.

This library will allow programs to be written to use different DLLs, each containing a specific filter algorithm, to process the data without having to know how to load a DLL dynamically. For this to work, the filter function in every DLL that can be used must have exactly the same signature. The signature of that must be followed by all filter functions is defined below.

Implement all Visual C++ libraries as Console Applications using pre-compiled headers. The DLL you create must be usable by software written in other programming languages, so the functions must be properly exported from the DLL. Represent strings in function parameters as null-terminated arrays of characters, that is, C-style strings. The C++ strings from the include file <string> do not work well with DLLs. Don’t use them.

**Part 1: The Static Library**

Create a static library named SLL that provides implementations of the functions listed below, along with a header file, filter.h, which

* 1. defines constants and types for use in any program handling signal data.
  2. declares the functions in the static library.
  3. declares the function in the dll **so that the same header file** can be used in the dll source to **export** the functions as well in programs that **import** functions by linking to the dll at load time.
  4. Declares a function pointer type to be used in programs that link to the dll at run time.

**Use standard preprocessor directives to ensure the header file is safe to use.** Do not use compiler-specific directives (e.g. #pragma).

There must only be 1 version of this file for use in the DLL and in programs that use the DLL. The header file must be suitable for use in programs that link the DLL at load time as well as programs that link the DLL at run time. **There is an initial, incomplete version of filter.h in Appendix 1.**

#### Definition of LoadMeasurements

int LoadMeasurements(WCHAR\* inputFile, int data[], int maxSize);

#### Purpose

To read data from input file (named inputFile) into array (data).

#### Parameters

WCHAR\* inputFile // the name of the file that contains the signal data

// this is a null-terminated character array

int data[] // an array of signal measurements

// Note: this is a pointer to the first element in the array

int maxSize // the maximum amount of data that the array can store

#### Result

The number of data items read if successful OR one of the error codes **InvalidParameters** or **UnknownError** defined in filter.h.

Notes

1. This function should report any detected errors only by returning an error code.

**Definition of UseFilter**

int **UseFilter** (WCHAR\* dllName, int data[], int count, WCHAR\* parameterString);

#### Purpose

To load a dynamic link library **at run time** from file (dllName) and process the data stored in array (data) using the filter function from the DLL, passing the parameter string each time the filter function is called.

#### Parameters

WCHAR\* dllName // the name of the DLL containing the filter function to process the data

int data[] // an array of measurements

int count // the number of measurements in the array

WCHAR\* parameterString // a string of additional parameters to be passed to the filter function

#### Result

The number of data items processed if successful, OR an error code: **DllNotFound**, **DllInvalidFormat**, **InvalidParameters** or **UnknownError**, which are defined as constants in filter.h (Appendix 1). The error codes are all negative.

#### Definition of SaveMeasurements

int LoadMeasurements(WCHAR\* outputFile, int data[], int maxSize);

#### Purpose

To save data from the array (data) to the output file (named outputFile).

#### Parameters

WCHAR\* outputFile // the name of the file to contain the measurements

int data[] // an array of measurements

// Note: this is a pointer to the first element in the array

int size // the amount of data in the array

#### Result

The number of data items written if successful OR one of the error codes **InvalidParameters** or **UnknownError** defined in filter.h.

Notes

1. This function should report any detected errors only by returning an error code.

**Part 2: DLL Implementation**

Each DLL developed to process the data will use the header file, filter.h, from the SSL and will export a function called Filter(), which must satisfy the signature below. The implementation of Filter will be different in different DLLs to provide different ways of processing (“filtering”) the data.

Write a DLL containing a function called Filter with the following signature. Your implementation of the function, Filter, will accept a parameter string containing two integers separated by a space. It will process the array replacing any value that is less than the first integer by the first integer and any value that is greater than the second integer by the second integer.

For example Filter([2, 3, 1, 5, 10], 5, “3 6”) will set the array to [3, 3, 3, 5, 6]

#### Definition of the signature for all filter functions

#### int Filter(int data[], int count, WCHAR\* parameterString);

#### Purpose

To apply an algorithm to the given data.

#### Parameters

int data[] // an array of measurements

int count // the number of measurements in the array

WCHAR\* parameterString // a string of additional parameters specific to the filter algorithm

#### Result

Return **Success** and a correctly modified array, data, OR one of **InvalidParameters** or **UnknownError** (defined as constants in filter.h in Appendix 1).

Notes

1. Different filter algorithms can expect different parameters to be passed in parameterString

For example,

* A filter to multiply each element by a number might expect a single integer to be passed as a string.

Filter(theData, theSize, “4”) if the DLL contains a function to multiply each element

* A filter to replace very small or very large values might expect two integers to be passed as a string.

Filter(myData, numItems, “-10 +10”) if the DLL contains a function to ensure that no item is less than -10 or greater than +10

### Part 3: Using the Debugger

Demonstrate that you can use the debugger by providing a f screenshots and brief comments to show that you can use the following features competently:

* Breakpoints (use 2 break points)
  + When breakpoints are placed, there should be a reason. No marks will be given for breakpoints that do not serve a purpose.
  + Each breakpoint presented should be accompanied by 1-2 sentences explanation.
* Single stepping (provide 2 examples)
  + You should demonstrate two locations where Step Into and Step Over should be used
  + Each command presented should be accompanied by 1-2 sentences explanation.
* Viewing variables
  + Demonstrate the use of a Watch Window by showing how, inside a DLL, a single line is processed and transformed into an array element.

**Part 4: Using the Preprocessor to Assist Debugging**

1. Demonstrate that you can use output statements and the preprocessor to manage when those output statements are displayed by providing screenshots and brief comments to explain why you have placed the output statements and how the preprocessor helps control these statements.
2. Demonstrate that you can use a relevant assert macro to help debugging by providing screenshots and brief comments to explain why you have placed the assert macro in your code, why it is useful when debugging and how the preprocessor processes the macro.

## Grading Criteria

Marks will be awarded based on the following criteria. Within each part, aim to complete the work for each section before moving on to the next as may will not get full credit for later sections if there are major defects in an earlier section. **However, do not simply stop if you are stuck on one part, but can do later parts.**

In assessing the work within a section, factors such as simplicity, quality and appropriateness of comments, lack of bugs and quality and completeness of the program will be considered.

|  |  |  |  |
| --- | --- | --- | --- |
| **Part** | **Range** | **Criteria** | **Comment** |
| 1a  (30%) | 0-7 | Static library compiles and links with the main application.  Main program runs, loads the data using the library function **LoadMeasurements**, uses **UseFilter** to process the data. **LoadMeasurements** and **SaveMeasurements** handle errors appropriately.  If appropriate, main program saves the data using the function, **SaveMeasurements**, from the library. |  |
| 0-15 | Static library function, **UseFilter**, declares appropriate variables to allow a run-time linked DLL to be loaded and used.  **UseFilter** loads a DLL with the given name and uses it appropriately. Returns appropriate error codes, if necessary |  |
| 0-8 | **Filter()** function obtained from the DLL and called with appropriate parameters |  |
| 2  (25%) | 0-4 | A **single** header file, filter.h making appropriate declarations for use in multi-file programs using **standard** C++ Compiler facilities to avoid duplicate inclusion of the header file |  |
| 0-13 | The header file, filter.h, suitable for inclusion in the cpp file defining **Filter()** in the DLL and in programs that link to the DLL **at load time or at run time**. Appropriate use of C++ Compiler facilities to ensure the same header file can be used in the DLL implementation to export the function defined in the DLL and in programs that import the function from the DLL at load time. |  |
| 0-8 | DLL1: Contains a correct **Filter()** function that returns an appropriate code to show that it has processed the data according to its specification or that an error has occurred. |  |
| 3 (20%) | 0-5 | Evidence of use of breakpoints |  |
| 0-10 | Evidence of use of single-stepping |  |
| 0-7 | Evidence of use of viewing variables, including arrays |  |
| 4 (25%) | 0-4 | Example C++ code using preprocessor to support debugging without a dynamic debugger |  |
| 0-9 | Explanation of this debugging approach and how to use it  Explanation of use of preprocessor in this approach |  |
| 0-4 | Example C++ code using assert macro to support debugging |  |
| 0-8 | Explanation of debugging approach using assert  Explanation of role of preprocessor in use of assert |  |

## Deliverables

Submit the following deliverables for Parts 1-4 in a single file in ZIP format via the Blackboard site.

1. The marking scheme in word format, using the space for comments to giving a brief overview of what you've done, **and** your expected mark for these parts with justification.
2. **All the source files (.cpp, .h) in a single folder**
3. A document in word format containing screenshots and explanation of use of debugger and preprocessor facilities
4. Executable file of your solution with the static library (.lib) and the DLL. This program must run without requiring any additional libraries.

**DO NOT SUBMIT THE VISUAL STUDIO SOLUTION/PROJECT, etc.** I just want the source and the executable (.exe), the static library (.lib) file and the DLL

**Submission of assignment work**

* Anonymous marking is being used. Avoid doing anything that would allow you to be identified from your work.
* *Keep a complete copy of the work you hand in.*
* Avoid submitting work at the last minute, but if there is a technical problem uploading to Blackboard, email the zip file to me before the deadline and upload the work when Blackboard is available.

**Extenuating circumstances, extensions and late work**

Except where an extension of the hand-in deadline date has been approved (see <https://www.uclan.ac.uk/students/support/extensions.php> ), work that is handed in up to 5 days late will be capped to 40%. After this, it will receive a mark of 0%:

**Cheating**

The consequences of cheating in assessments are serious. Cheating is using or attempting to use unfair means to enhance performance. This includes plagiarism (presenting someone else's work as if it was your own), collusion (working with others on an individual assignment), taking prohibited material into examinations and allowing other students to access your work. Make sure that you do not give someone the opportunity to steal your work (e.g. *by asking them to print it out for you*). We tell students about cheating both during induction and in your student handbook, but if you have any doubt about what cheating is or how to reference material properly, please ask a tutor. We recommend that you use the Harvard system for referencing.

The University operates an electronic plagiarism detection service where your work may be uploaded, stored and cross-referenced against other material. The software searches the World Wide Web and extensive databases of reference material to identify duplication.

For more information about plagiarism, please see the University Academic Regulations and the Assessment Handbook (<http://www.uclan.ac.uk/aqasu/academic_regulations.php>). See the Student Union website: <http://www.uclansu.co.uk/academicmatters/unfairmeans>

# Appendix 1

**filter.h**

const int Success = 0;

const int Failure = -1;

const int DllNotFound = -2; // can't find dll to import

const int DllInvalidFormat = -3; // Unable to find import function in the dll

const int UnknownError = -4;

const int InvalidParameters = -5;

const int MAX\_DATA = 1000; // the maximum size of the array for loading data

const int DATE\_SIZE = 19; // the maximum size of a date as a null-terminated string

int LoadMeasurements(WCHAR\* inputFile, int data[], int maxSize);

int UseFilter (WCHAR\* dllName, int data[], int count, WCHAR\* parameterString);