CHARITH AKALANKA

103800782

Task 1 Submission 2

CYB80003 - Advanced Cybersecurity Programming

Table of Contents

[1.0 Instructions for assembly 2](#_Toc135226661)

[1.1 Visual Studio Setup 2](#_Toc135226662)

[1.2 Assembling code 11](#_Toc135226663)

[1.3 Execution 14](#_Toc135226664)

[2.0 Program Structure 15](#_Toc135226665)

[2.1 Flow Chart 15](#_Toc135226666)

[2.2 Procedures Used 16](#_Toc135226667)

[2.2.1 WriteMsg 16](#_Toc135226668)

[2.2.2 RetrieveSid 16](#_Toc135226669)

[2.2.3 ConvertIntToString 16](#_Toc135226670)

[2.2.4 VerifyPass 16](#_Toc135226671)

[2.2.5 Crypt 16](#_Toc135226672)

[2.2.6 GetDetails 16](#_Toc135226673)

[2.2.7 ClearBuffer 16](#_Toc135226674)

[2.2.8 Str\_concat 17](#_Toc135226675)

[2.2.9 CreateDataString 17](#_Toc135226676)

[2.2.10 WriteDetails 17](#_Toc135226677)

[References 18](#_Toc135226678)

# Instructions for assembly

The following instructions are targeted for Visual Studio 2022 that is already installed with requirements for C++ programming. If that is not the case, Visual Studio Installer utility must be used to install the dependencies for C++.

## Visual Studio Setup

A screenshot of a computer

Description automatically generated

Figure 1 - Create a new project

A screenshot of a computer program

Description automatically generated with medium confidence

Figure 2 - Select C++ "Empty Project" template

This step assumes that your Visual Studio 2022 installation is already initialised for C++ programming.

A screenshot of a computer

Description automatically generated

Figure 3 - Give a name and select the location

A screenshot of a computer

Description automatically generated

Figure 4 - Create the project

A screenshot of a computer

Description automatically generated

Figure 5 - Delete unused sub directories from the project tree

Now, the project must be specified as an Assembly program.

A screenshot of a computer

Description automatically generated

Figure 6 - Right click on the Project name and open "Build Dependencies..." > "Build Customizations..."

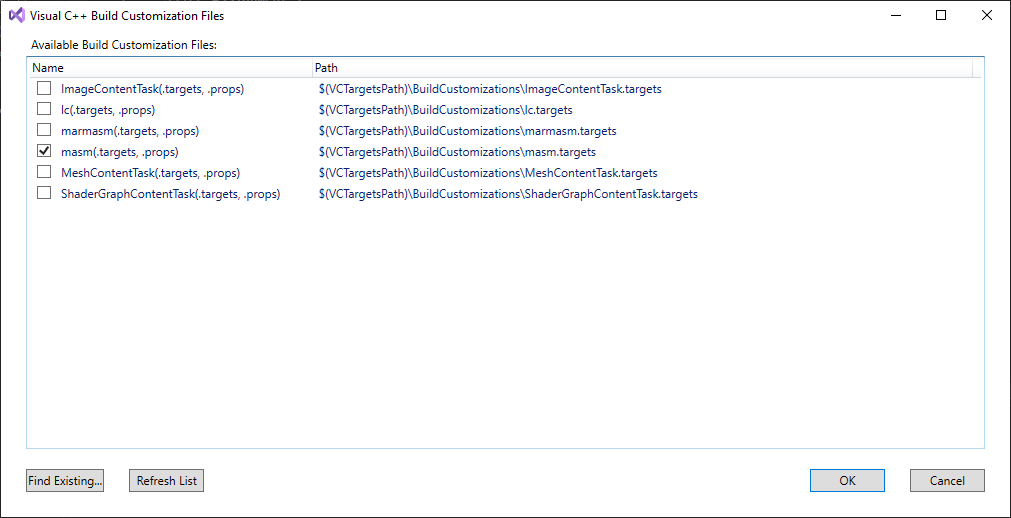


Figure 7 - Tick "masm"

Now, open the assembly source code.

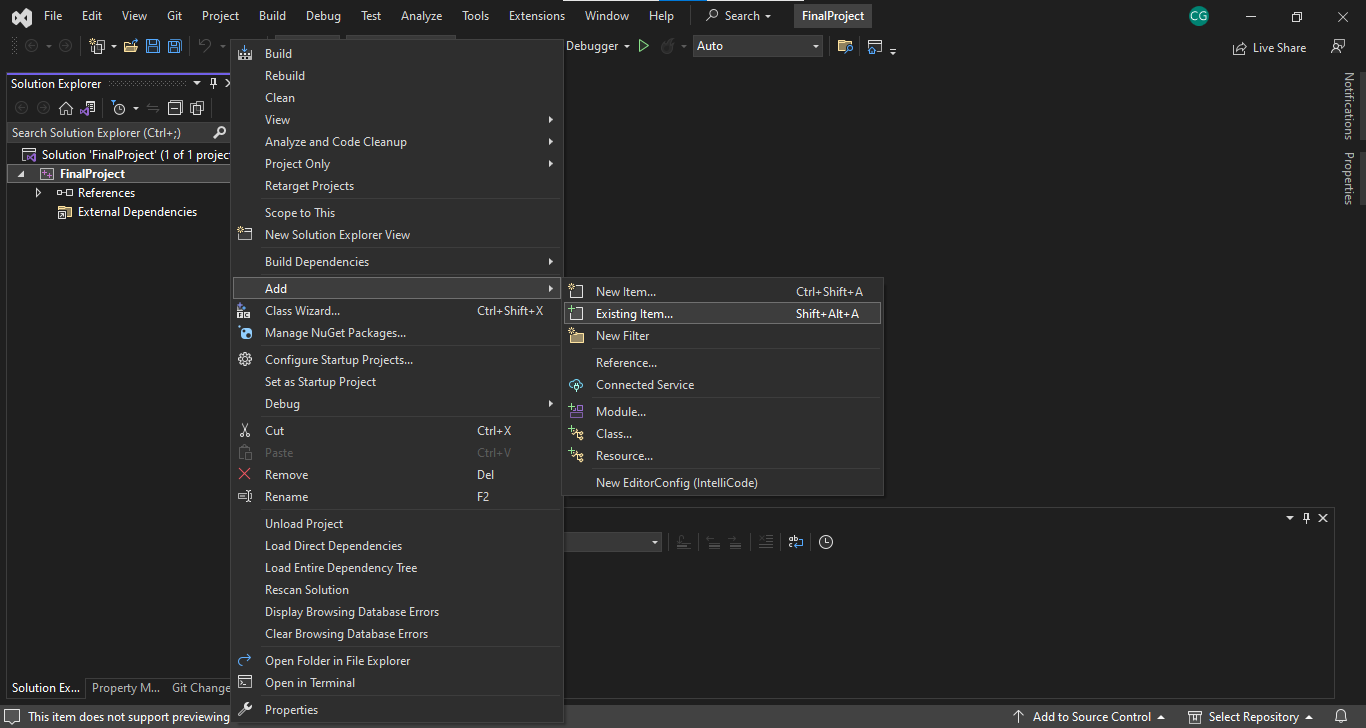


Figure 8 – Right click on Project name and go to "Add" > "Existing Item..."

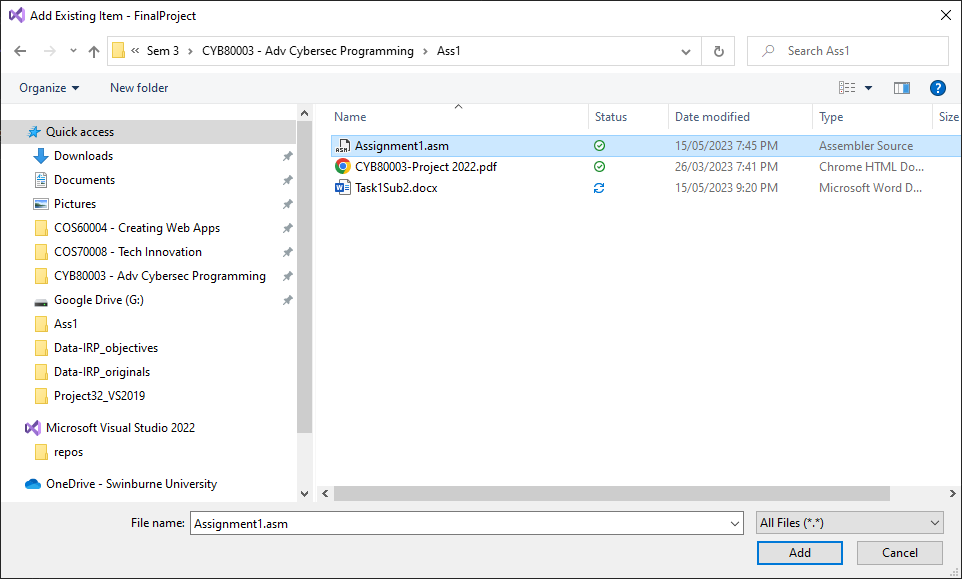


Figure 9 - Select the assembly source file

A screenshot of a computer program

Description automatically generated with medium confidence

Figure 10 - Assembly source code should appear in the project tree

A screenshot of a computer

Description automatically generated

Figure 11 - Right click on the .asm file and open Properties

A screenshot of a computer

Description automatically generated with medium confidence

Figure 12 - Item type must be " Microsoft Macro Assembler"

If all steps up to this point were successful, Visual Studio must automatically assign “Microsoft Macro Assembler” as the item type.

The project is now ready for working with Assembly code. But the source code relevant to this report uses components from the Irvine32 library. Hence the project must be configured to read the Irvine files.

A screenshot of a computer

Description automatically generated

Figure 13 - Right click on Project name and go to Properties

A screenshot of a computer

Description automatically generated

Figure 14 - Go to "Linker" > "General" Tab

Change the "Additional Library Directories" and assign the path to the Irvine folder.

Click on “Apply” to save changes before continuing.

A screenshot of a computer

Description automatically generated

Figure 15 - Open "Linker" > "Input" Tab

Change “Additional Dependencies” to also include the Irvine32 library.

This can be done by clicking on the value and selecting “Edit” option.

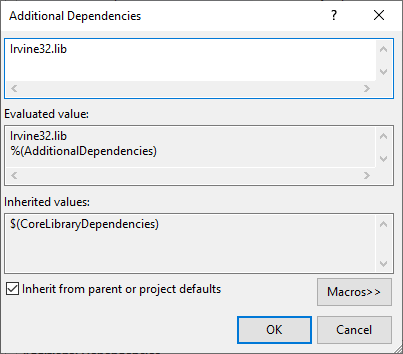


Figure 16 - Type "Irvine32.lib"

Type the library file name, “Irvine32.lib” and click OK.

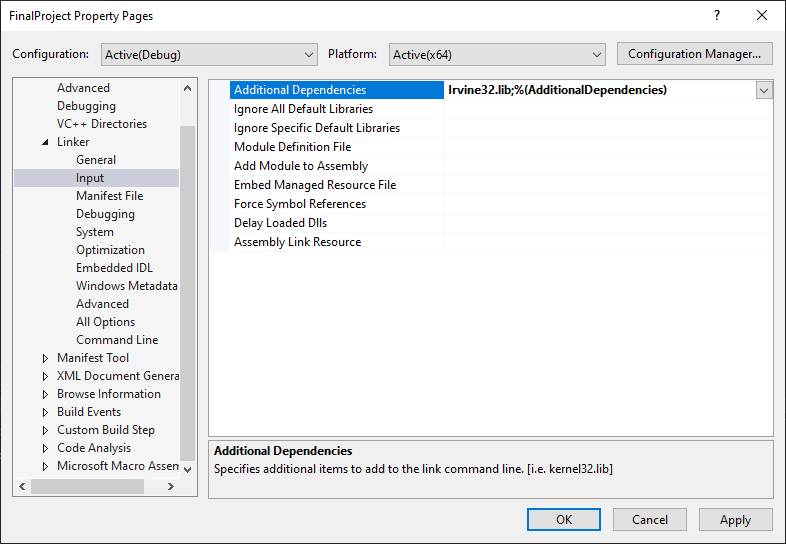


Figure 17 - Additional Dependencies added

Click on “Apply” to save changes before continuing.

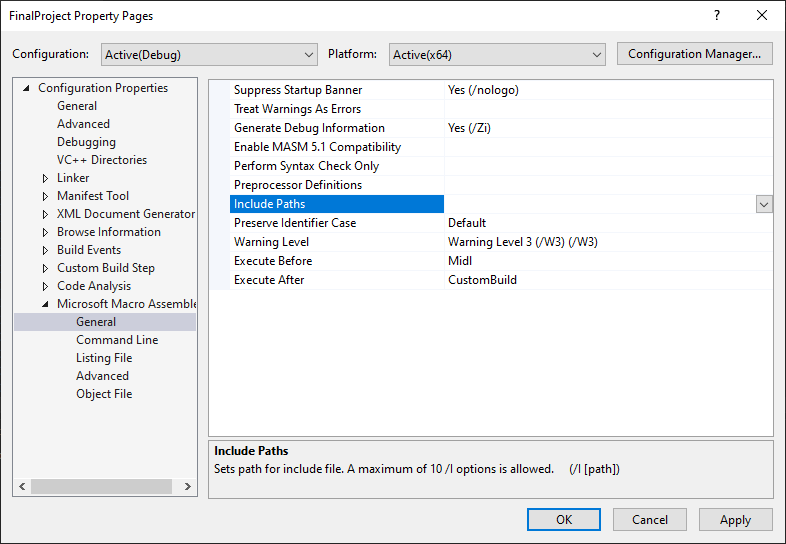


Figure 18 - Go to "Microsoft Macro Assembler" > General Tab

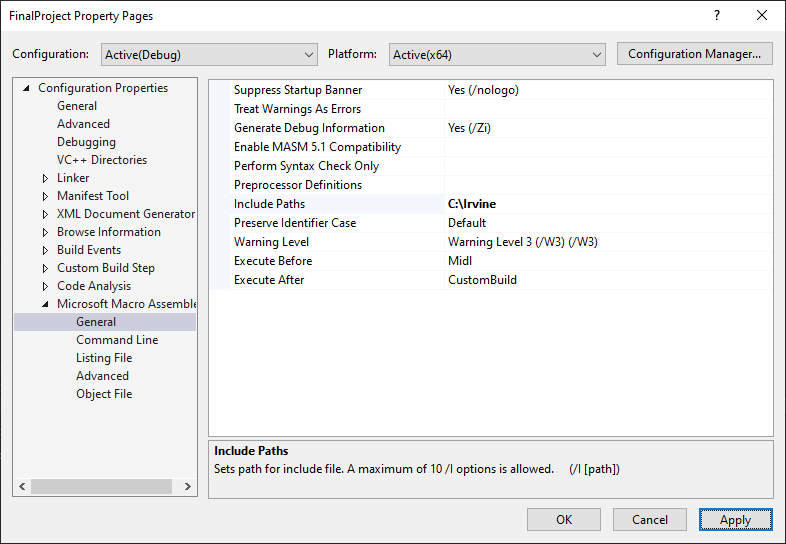


Figure 19 - Change "Include Paths" to the path to Irvine folder

Click on “Apply” to save changes before continuing.

All dependencies have been configured. The project is now ready to be assembled.

## Assembling code

A screenshot of a computer program

Description automatically generated with medium confidence

Figure 20 - Select the Solution Platform to be x86

A screenshot of a computer

Description automatically generated

Figure 21 - From the main menu go to Build > Build Project

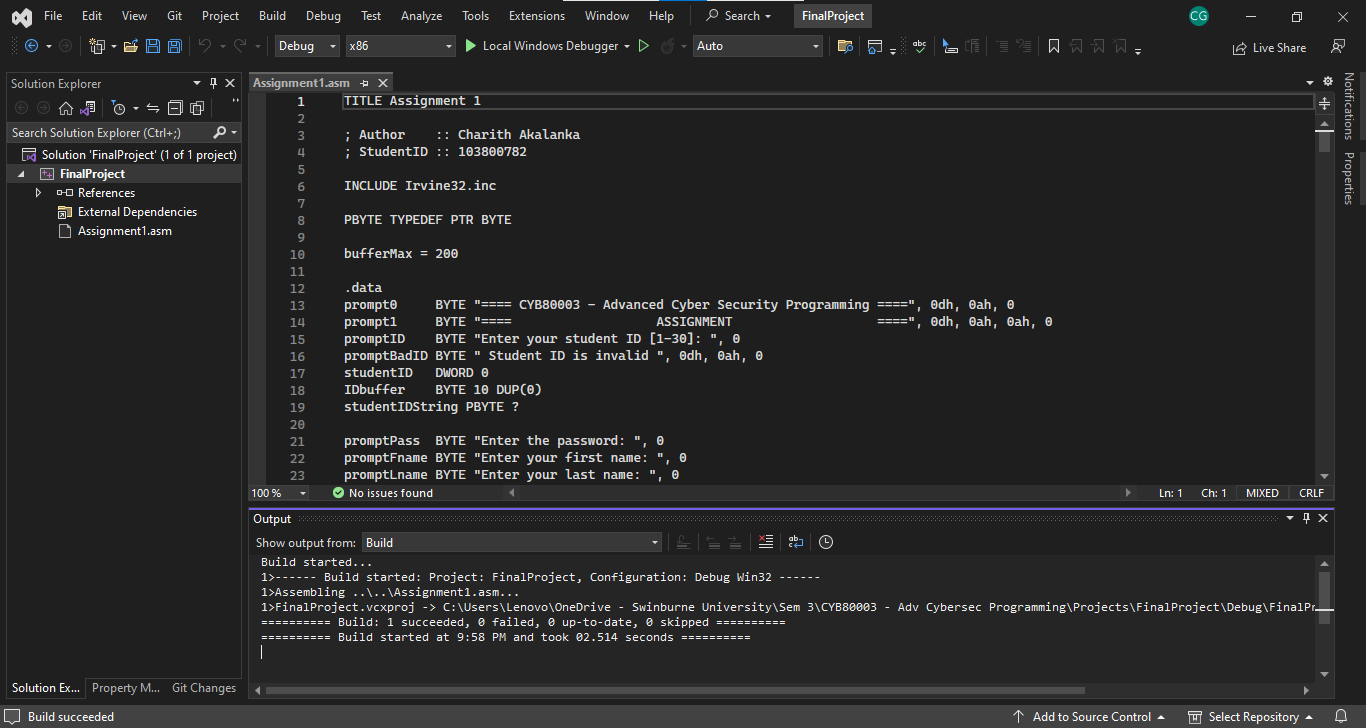


Figure 22 - Successful build

A screenshot of a computer program

Description automatically generated with medium confidence

Figure 23 - Output message for a Successful build

This message indicates that the source code was successfully assembled into an executable.

## Execution

Options for executing the assembled code can be found on the “Debug” menu.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 24 - Go to Debug menu for execution

Select “Start Without Debugging” to run the program.

A screenshot of a computer

Description automatically generated

Figure 25 - Program execution

# Program Structure

## Flow Chart

WriteMsg

RetrieveSid

ConvertIntToString

VerifyPass

GetDetails

ClearBuffer

CreateDataString

WriteDetails

## Procedures Used

### WriteMsg

This procedure prints the welcome message. Uses the WriteString procedure defined in Irvine32 library to perform this.

### RetrieveSid

This procedure encloses all the steps for retrieving the Student ID (SID). It prints the prompt using the WriteString procedure, and reads the user input using the ReadDec procedure, both defined in Irvine32 library. ReadDec is capable of validatingReadDec can validate the user inputs and accepting onlyto be valid positive decimal values. But additional checks have been put in place to verify if the value is within the range of 1 to 30. The procedure will loop until the user inputs a valid number, while displaying an error message.

### ConvertIntToString

RetrieveSid procedure stores the correct StudentID as a binary value. This procedure converts this value into a string consisting of individual characters represented as ASCII values, so that it may be printed on the data file at the end of the program.

### VerifyPass

This procedure encloses all the steps required to retrieve and verify password. WriteString procedure is used to display the prompt and error messages. ReadChar procedure from the Irvine32 library is used to read user inputs, one character at a time, without displaying it on the screen. An infinite loop is setup to read a single key stroke and echo “\*” to indicate the key stroke. Pressing the ENTER key causes the program to break out of the loop and compare the user input against the hardcoded password.

### Crypt

This procedure processes a given string by XORing every character with a predefined key value. XOR is a reversible operation [1]. Hence the same procedure can be called to encrypt and decrypt the same string.

This procedure processes a given string by XORing every character with a predefined key value. XOR is a reversible operation .

An encrypted version of the password is hardcoded onto the source code. Crypt procedure is called inside the VerifyPass procedure before the user input is compared. This causes the password to be decrypted on the memory location it currently resides in. After the comparison is done, Crypt is executed again to encrypt the password and protect it for the rest of the program’s execution.

### GetDetails

This procedure encloses all the steps for retrieving user details. WriteString is used for printing the prompts and ReadString is used for reading user inputs, except the date of birth. The procedure is designed to read the date in DD/MM/YYYY format.

For reading the date of birth, ReadChar is used to read the user input, one key stroke at a time. Only numerical keys will be accepted, and date separator “/” is added automatically by the program. Any other key press will be ignored. Once all positions are filled, the input gets submitted automatically. Else, the user can press ENTER key to abort and retry.

### ClearBuffer

A large buffer is used to read user inputs. This procedure is used to reset the entire buffer into a sequence of 0 s, thereby erasing any data residual data.

### Str\_concat

This procedure takes the addresses of a source string and a target string and copies the content in source string at the end of the target string, thereby concatenating the two strings. This procedure is used in CreateDataString.

### CreateDataString

This procedure creates the data string that will be written to the output file. Str\_copy procedure in Irvine32 library, and the Str\_concat procedure are used to sequentially concatenate various hardcoded strings and user inputs on to the buffer. The buffer can be printed onto the console to preview the output that will be written onto the data file.

### WriteDetails

This procedure encloses all the steps from opening the text file to appending data. The CreateFile function is used to open the existing data file or create a new data file. Once the file is open, data is appended to the end of the file, thereby preserving any data that may already be on the file. Finally, the file is closed.

# References

[1] C. Paar and J. Pelzl, *Understanding Cryptography: A Textbook for Students and Practitioners*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. doi: 10.1007/978-3-642-04101-3.