A black and red sign with white text

Description automatically generated

CYB80003: Advanced Cyber Security Programming

Task 1: Develop An Assembly Language Program

Author: Gia Huy Tran

Student ID: 104224749

Class time: Monday, 6:30 p.m.– 8:30 p.m.

Last modified: 22/05/2025

Contents

[Create a project in Visual Studio 3](#_Toc198799799)

[Details of Procedures 7](#_Toc198799800)

[Flow Char 9](#_Toc198799801)

[Assembly Program 11](#_Toc198799802)

# Create a project in Visual Studio

* Download Irvine library from the provided link (<https://swinburne.instructure.com/courses/66692/files/35868494?wrap=1>), extract and save Irvine folder in C disk or anywhere you can remember.
* Create new project in Visual Studio, select “Empty Project”, make sure you choose the correct language, C++ in this case.
* Click next to create the new empty project.

\A screenshot of a computer program

AI-generated content may be incorrect.

Figure 1 - Create a new Project

* In the new project windows, give it a meaningful name such as “Assignment 1, etc”.
* Change the folder location to your preferred location, I used OneDrive folder to make sure I can work with my project anywhere.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 2 - Configure new Project

* After creating a new project, right click on project’s name, choose “Build Dependencies 🡪 Build Customisations…”. Select masm(.targets,.props) like the picture below.

\A screenshot of a computer

AI-generated content may be incorrect.

Figure 3 - Select dependencies

* Right click on project’s name, select Properties.
* In the Properties pages, select Linker 🡪 General, add additional Library Directories, e.g. C:\irvine 🡪 click Apply.
* Make sure Configuration is Active(Debug) and Platform is Win32 or Active(Win32).

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4 - Add additional library directories

* Select Input, in Additional Dependencies, add $(CoreLibraryDependencies);%(AdditionalDependencies);irvine32.lib.
* Click Apply.

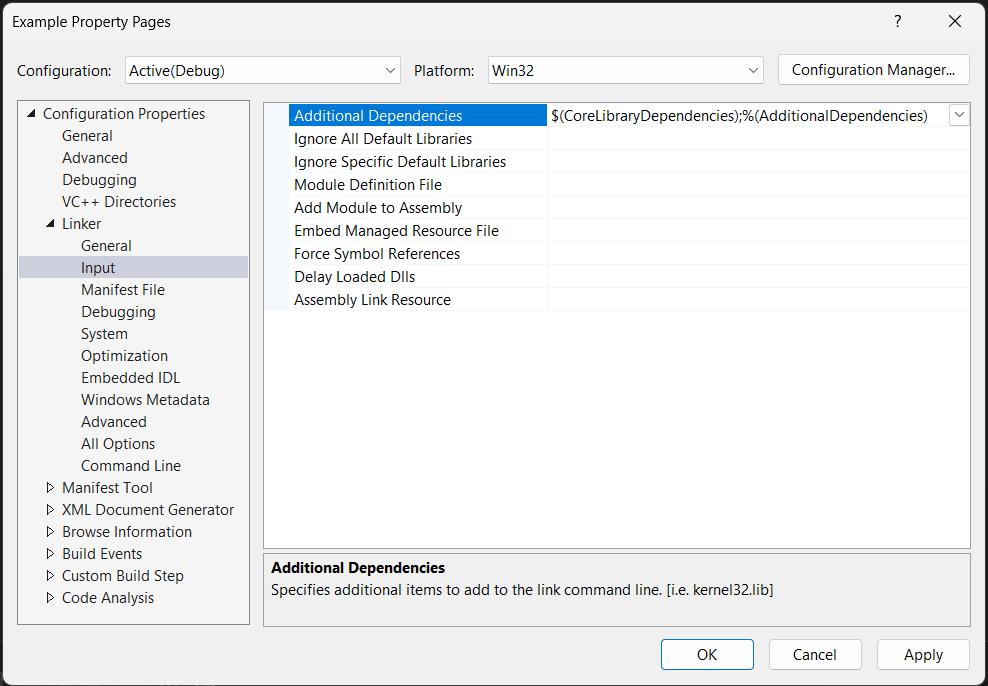


Figure 5 - Add additional dependencies

* After completing configure the dependencies and properties, right click on project’s name 🡪 add 🡪 Existing Item…
* Navigate to the folder that contain .asm file, and choose Check\_ID\_Passwd.asm.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 6 - Import the existing item

* Right click on Check\_ID\_Passwd.asm, in General choose Microsoft Macro Assembler.
* In Microsoft Macro Assembler 🡪 General, add Included Paths C:\irvine 🡪 apply.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 7 - Add path for library

* Before debugging, make sure to change solution platform to x86 otherwise you will not be able to build the project.



Figure 8 - Change solution platform

# Details of Procedures

A screenshot of a computer

AI-generated content may be incorrect.

Figure 9 - PROC chart

1. Main

Coordinates the entire login and verification process. It handles prompting for User ID and password, validating credentials, encrypting input, reading stored values from a file, comparing data, and handling success or failure results. It also controls user attempts and manages final output and file writing.

1. CheckUserIDs

Responsible for accepting and validating the User ID. It prompts the user to enter their ID, ensures the ID starts with 'S', is exactly 10 characters long, and contains 9 valid digits in a defined numeric range. It uses a loop to count characters and performs numeric validation via ASCII-to-integer conversion.

* *countLoop*: counts characters
* *checkLength*: ensures ID is 10 characters
* *validateDigits*: converts ASCII to number and checks range
* *shonInvalidID*: displays an error if validation fails

1. CheckPasswd

Handles password entry and encryption. Prompts the user to enter their password, checks that it does not exceed 30 characters, and encrypts it using a fixed XOR key. The encrypted password is then converted into a two-digit hexadecimal format per character.

* *encLoop*: XOR encrypts each character
* *BytetoHex*: converts each encrypted byte to 2 ASCII hex characters
* *NibbletoHex*: converts 4-bit values to ASCII hex digits
* File reading logic using *OpenInputFile*,*ReadFromFile*, and *CloseFile*

1. parseID

Scans the loaded file data up to the first colon (:) and stores the User ID portion of the saved line. This is used for credential comparison.

1. parsePass

Extracts the encrypted password string from the file data, starting after the first colon and ending at the next one. This value is compared with the user-inputted encrypted password.

1. compareFail

Handles the result of a failed login attempt. Displays an error message, increases the failed attempts counter, and either loops back to retry or exits if the maximum number of attempts is reached.

1. tooMany

Triggers after three failed login attempts. Displays a warning message and terminates the program to prevent further brute force attempts.

1. fileError

Displays an error and exits the program if the password file cannot be opened or created. This prevents further operations from proceeding without valid file access.

1. loginSuccess

Executed after both User ID and password are verified. Prompts the user to enter their full name and email, then formats and saves all data into the passwd file. Overwrites the file with the following format:

Sxxxxxxxxx:encrypted\_password:CYB80003:202501:Full Name:Email::

* CreteOutputFile: opens file for writing
* WriteStringToFile: writes each field and delimiter to the file

# Flow Char

A diagram of a computer program

AI-generated content may be incorrect.

Figure 10 - Flow chart

**Prompt for User ID**

The program asks the user to enter a User ID. It then checks if the format is valid:

* Must start with 'S'
* Must be exactly 10 characters long
* Must contain a numeric part between 100000000 and 109000000

**Invalid User ID Format?**

* If the User ID is invalid, the program displays an error and prompts the user to try again.
* This loop continues until a valid ID is entered.

**Prompt for Password**

Once a valid User ID is entered, the program prompts the user to enter their password.

**Password Too Long?**

If the password exceeds 30 characters, the program treats it as a possible intrusion attempt and exits immediately.

**Check Password Validity**

* The password is encrypted and compared to the stored password in the file.
* If the password is incorrect, the user is given up to **3 attempts**.
* After 3 failed attempts, the program exits with a "too many attempts" message.

**Successful Login**

* If the password is correct, the program proceeds to ask for:
  + **Full Name**
  + **Email Address**

**Save and Display**

* All user information (ID, encrypted password, course, semester, name, email) is saved in the file in the required format.
* A success message is displayed, and the program exits.

# Assembly Program

Users are required to log in using their User ID and Password, both of which are pre-defined. Only after successfully authenticating with valid credentials can users proceed.

Once login is successful, the program prompts users to enter their full name and email address, which are then saved to the passwd file in a structured format.

A computer screen shot of a black screen

AI-generated content may be incorrect.

Figure 11 - Perfect scenario

The user data is stored in the following format:

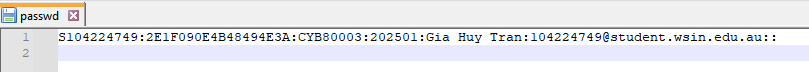


Figure 12 - Expected passwd fie

If a user attempts to log in with incorrect credentials three times, the program considers this potentially malicious and terminates the session with an appropriate warning.

A computer screen shot of a computer program

AI-generated content may be incorrect.

Figure 13 - Wrong credentials scenario

Additionally, access is denied if the User ID format is invalid. The ID must begin with an uppercase 'S' followed by a 9-digit number within the specified range. Any deviation from this format will prevent the login process from continuing.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 14 - Incorrect UserID format scenario

If the user enters a password that exceeds 30 characters, the program immediately blocks access and displays a security alert. This is treated as a potential intrusion attempt. To try again, the user must restart the application and log in with valid input.

A screen shot of a computer

AI-generated content may be incorrect.

Figure 15 - Inappropriate password scenario