Program Documentation

Name: Elvan Alandi

Table of Contents

Pre-Requisites

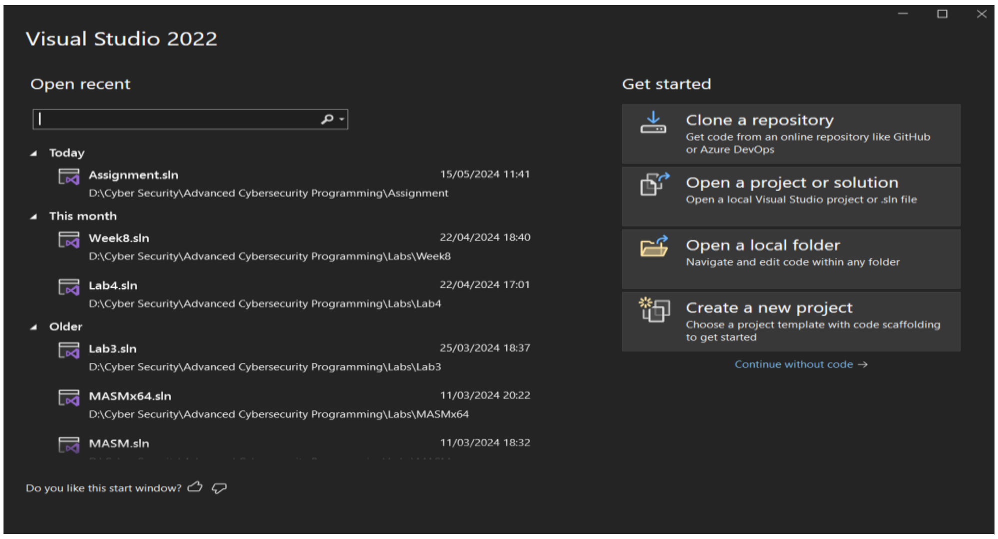
To run the program using the assembly file, some pre-requisites need to be configured:

- Microsoft Visual Studio application needs to be installed. For this documentation, I am using Microsoft Visual Studio 2022.
- Visual Studio project.
- Irvine32 library.

Microsoft Visual Studio application can be downloaded from the official website.

Below are the steps to configure Visual Studio to run the assembly file:

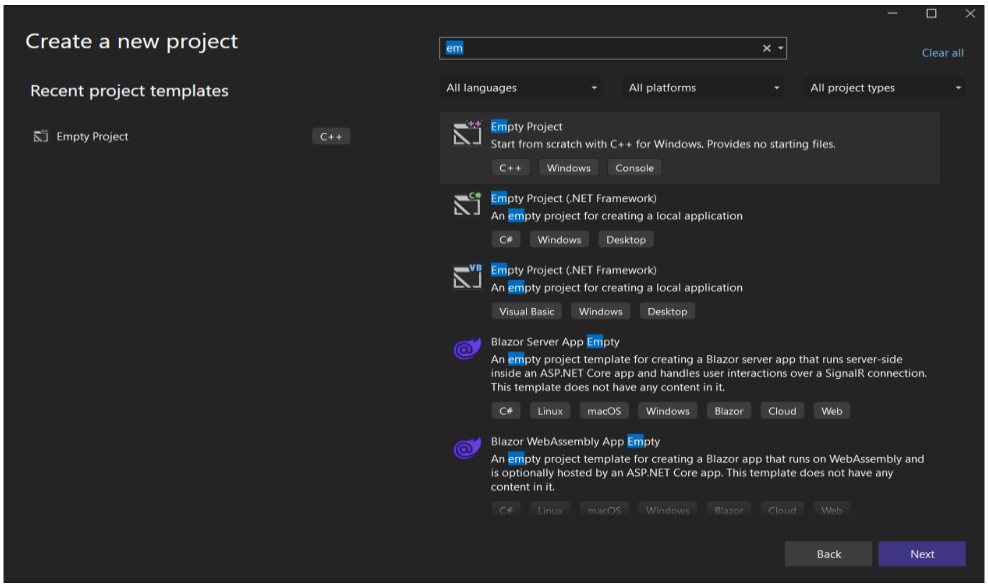
1. Open the Visual Studio application → Choose **Create a new project**.



Visual Studio 2022
Open recent
Today
Assignment.sln 15/05/2024 11:41
D:\Cyber Security\Advanced Cybersecurity Programming\Assignment
This month
Week8.sln 22/04/2024 18:40
D:\Cyber Security\Advanced Cybersecurity Programming\Labs\Week8
Lab4.sln 22/04/2024 17:01
D:\Cyber Security\Advanced Cybersecurity Programming\Labs\Lab4
Older
Lab3.sln 25/03/2024 18:37
D:\Cyber Security\Advanced Cybersecurity Programming\Labs\Lab3
MASMx64.sln 11/03/2024 20:22
D:\Cyber Security\Advanced Cybersecurity Programming\Labs\MASMx64
MASM.sln 11/03/2024 18:32
Get started
Clone a repository
Get code from an online repository like GitHub or Azure DevOps
Open a project or solution
Open a local Visual Studio project or .sln file
Open a local folder
Navigate and edit code within any folder
Create a new project
Choose a project template with code scaffolding to get started
Continue without code →
Do you like this start window?

Figure 1. Creating New Project

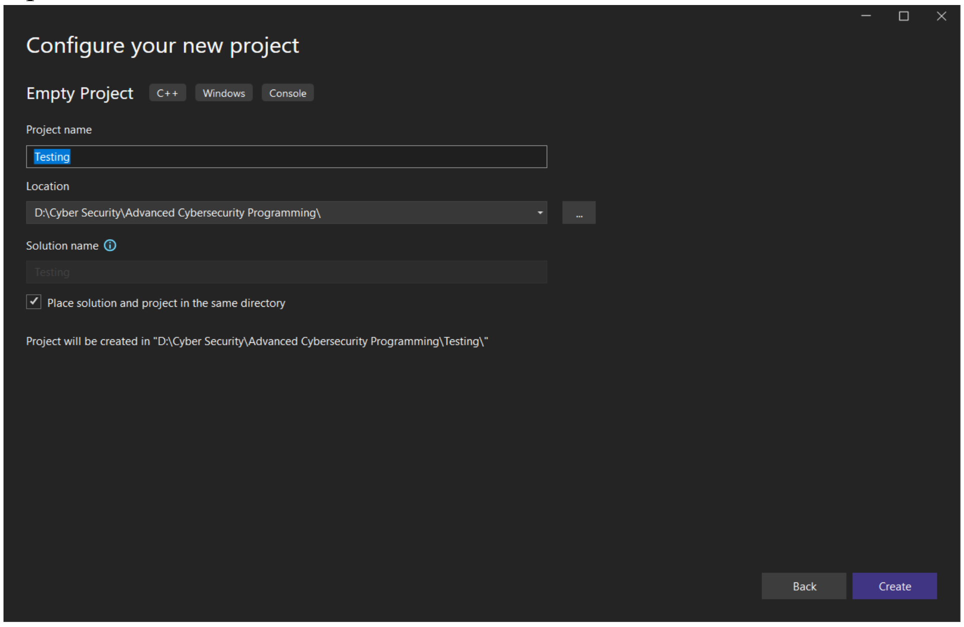
2. Search for **Empty Project** in the search bar. Choose the **C++ Empty Project,** then click **Next**.



Create a new project
Recent project templates
Empty Project C++
em
Clear all
All languages
All platforms
All project types
Empty Project
Start from scratch with C++ for Windows. Provides no starting files.
C++ Windows Console
Empty Project (.NET Framework)
An empty project for creating a local application
C# Windows Desktop
Empty Project (.NET Framework)
An empty project for creating a local application
Visual Basic Windows Desktop
Blazor Server App Empty
An empty project template for creating a Blazor server app that runs server-side inside an ASP.NET Core app and handles user interactions over a SignalR connection. This template does not have any content in it.
C# Linux macOS Windows Blazor Cloud Web
Blazor WebAssembly App Empty
An empty project template for creating a Blazor app that runs on WebAssembly and is optionally hosted by an ASP.NET Core app. This template does not have any content in it.
C# Linux macOS Windows Blazor Cloud Web
Back
Next

Figure 2. Empty Project

3. Insert the **Project name** and configure the project **Location** based on your preference. Then, click the **Create** button.


Configure your new project

Empty Project C++ Windows Console

Project name: Testing

Location: D:\Cyber Security\Advanced Cybersecurity Programming\

Solution name: Testing

☑ Place solution and project in the same directory

Project will be created in "D:\Cyber Security\Advanced Cybersecurity Programming\Testing\"

Figure 3. Configure New Project

4. After creating the project, **right-click** on the **project name** in the **Solution Explorer**, located on the right side of the project. Since, my project name is "Testing", I right-click on "Testing". In the popup menu, select **Build dependencies**, and choose **Build customizations**.

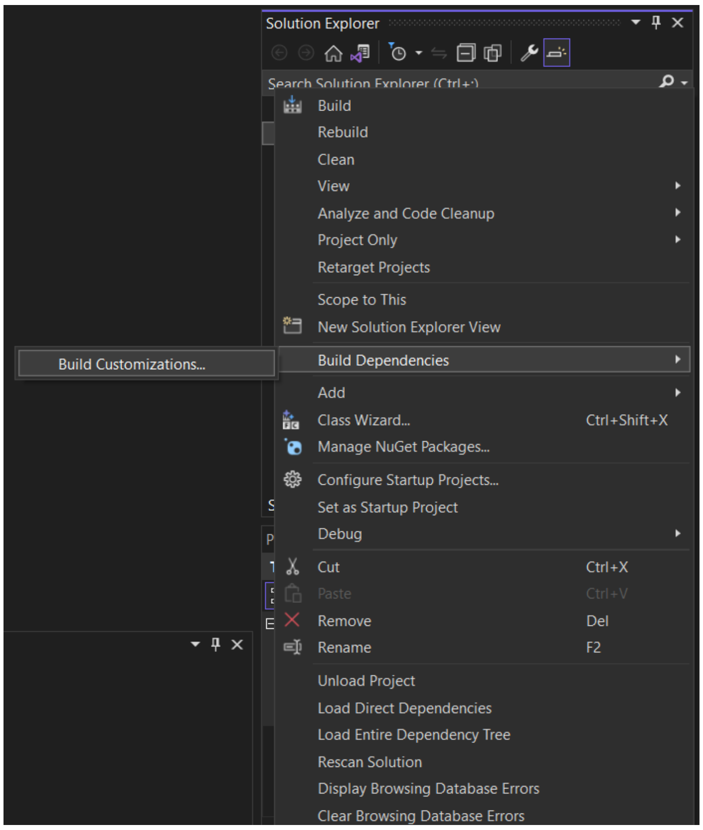
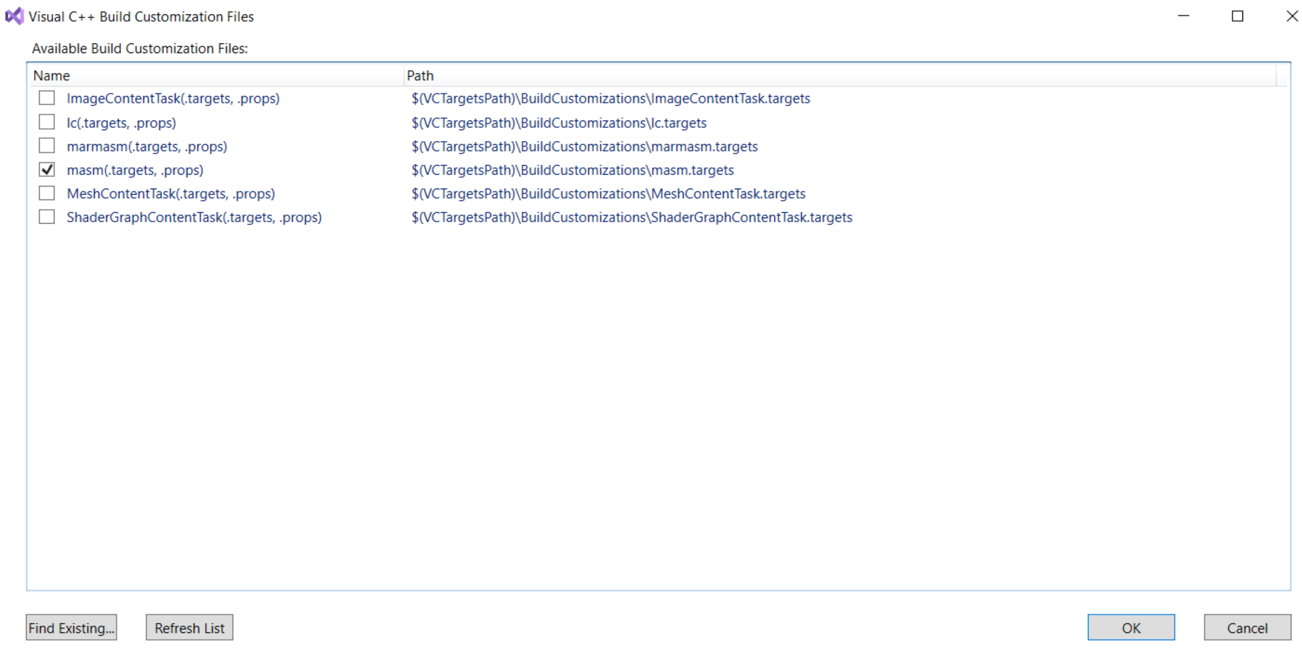


Figure 4. Build Dependencies

5. Check **masm** on the list → click **OK**.


Visual C++ Build Customization Files

Available Build Customization Files:

Name	Path
☐ ImageContentTask(.targets, .props)	$(VCTargetsPath)\BuildCustomizations\ImageContentTask.targets
☐ lc(.targets, .props)	$(VCTargetsPath)\BuildCustomizations\lc.targets
☐ marmasm(.targets, .props)	$(VCTargetsPath)\BuildCustomizations\marmasm.targets
☑ masm(.targets, .props)	$(VCTargetsPath)\BuildCustomizations\masm.targets
☐ MeshContentTask(.targets, .props)	$(VCTargetsPath)\BuildCustomizations\MeshContentTask.targets
☐ ShaderGraphContentTask(.targets, .props)	$(VCTargetsPath)\BuildCustomizations\ShaderGraphContentTask.targets

Figure 5. MASM Build Customization

6. Go to the **Build** menu and select **Configuration Manager**.

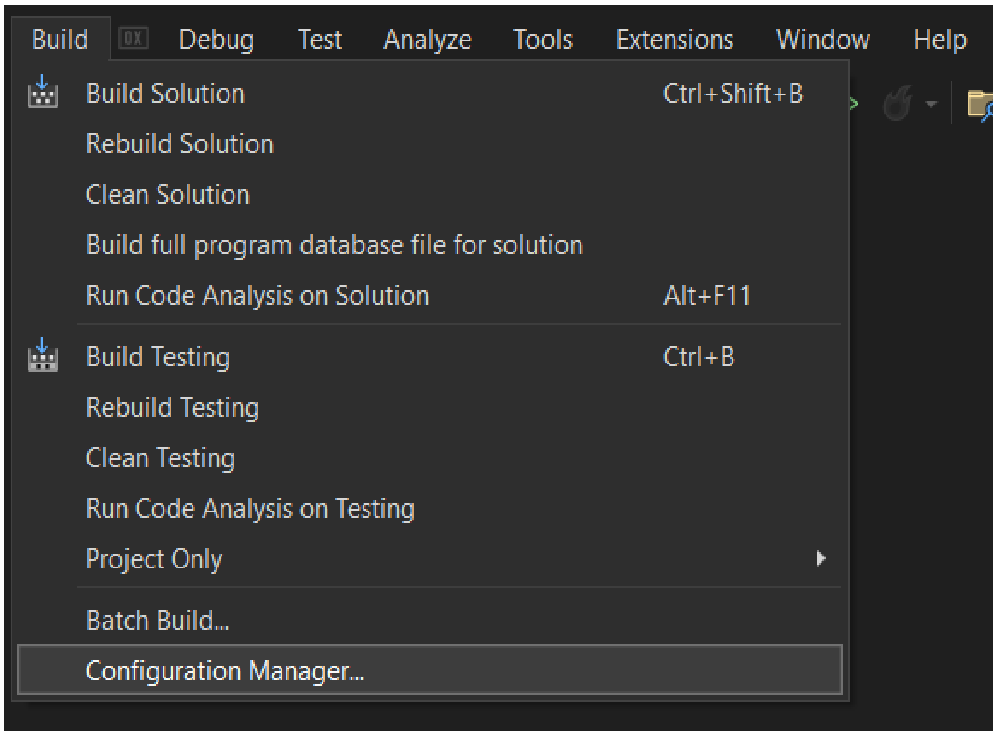
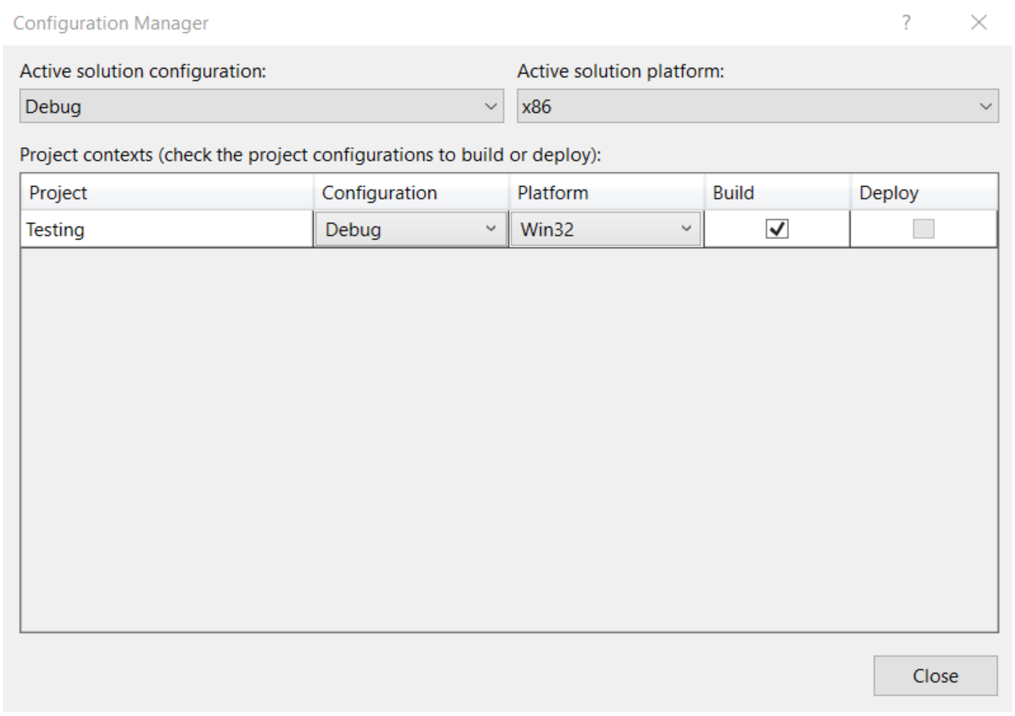


Figure 6. Configuration Manager

7. Change the **Active solution platform** to x86 and make sure the platform is **Win32**.


Configuration Manager

Active solution configuration: Debug

Active solution platform: x86

Project contexts (check the project configurations to build or deploy):

Project	Configuration	Platform	Build	Deploy
Testing	Debug	Win32	✓	

Figure 7. Platform Configuration

8. **Right-click** on the project name in the **Solution Explorer** again. This time, choose **Properties**.

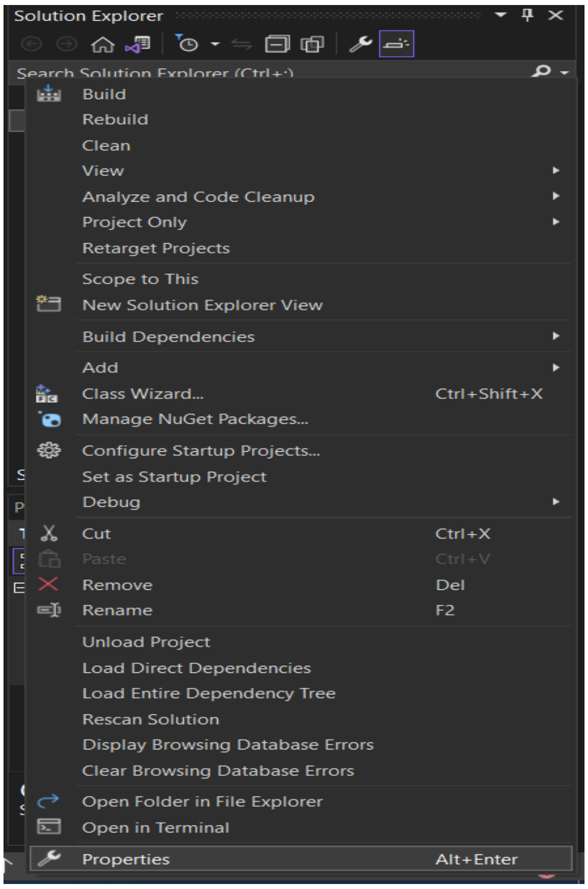
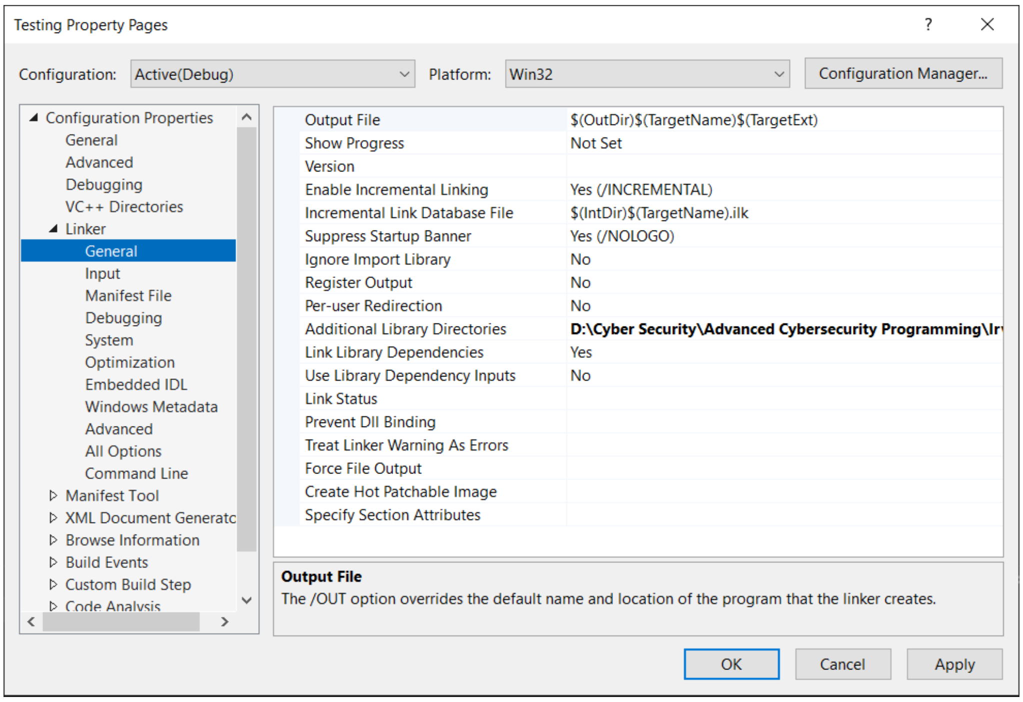


Figure 8. Project Properties

9. Select **Linker → General,** and set **Additional Library Directories** to the path of your **Irvine** library directory. In my case, I am using **"D:\Cyber Security\Advanced Cybersecurity Programming\Irvine"** as the path.



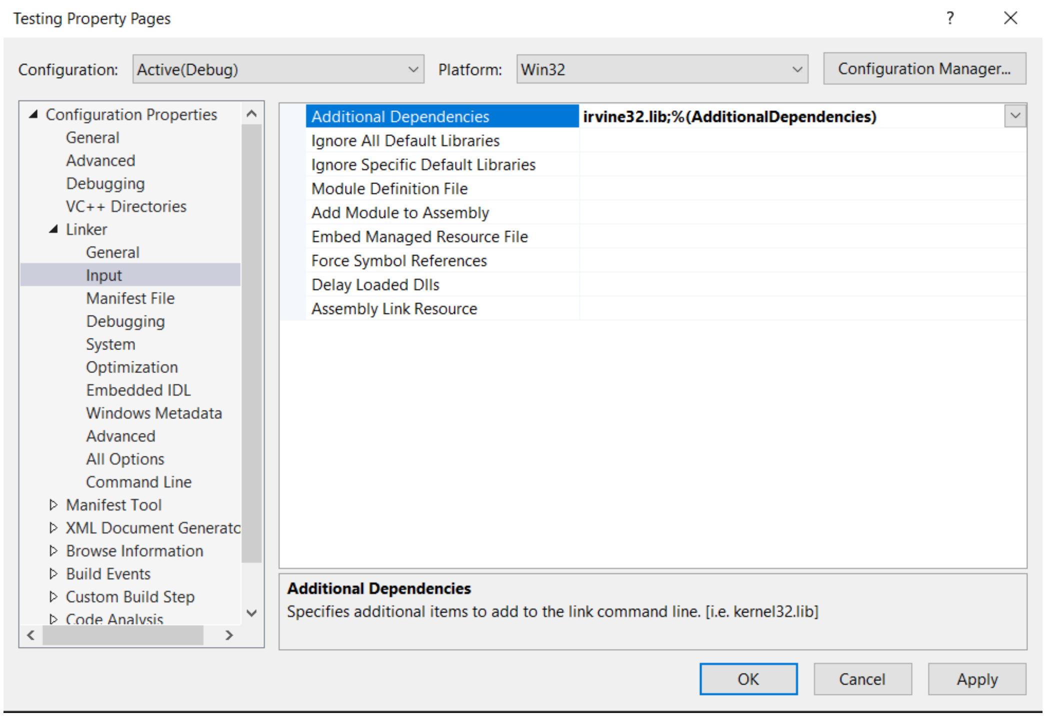
Configuration: Active(Debug)   Platform: Win32

Output File	$(OutDir)$(TargetName)$(TargetExt)
Show Progress	Not Set
Version	
Enable Incremental Linking	Yes (/INCREMENTAL)
Incremental Link Database File	$(IntDir)$(TargetName).ilk
Suppress Startup Banner	Yes (/NOLOGO)
Ignore Import Library	No
Register Output	No
Per-user Redirection	No
Additional Library Directories	**D:\Cyber Security\Advanced Cybersecurity Programming\Ir**
Link Library Dependencies	Yes
Use Library Dependency Inputs	No
Link Status	
Prevent Dll Binding	
Treat Linker Warning As Errors	
Force File Output	
Create Hot Patchable Image	
Specify Section Attributes	

Output File
The /OUT option overrides the default name and location of the program that the linker creates.

Figure 9. Linker's Library Path

10. Navigate to the **Input** menu, and in the **Additional Dependencies** section, add **irvine32.lib**. You can insert the library by **double-clicking** on the text field. Remember to add a **semicolon** after library name to separate the dependencies.



Configuration: Active(Debug)   Platform: Win32

Additional Dependencies	**irvine32.lib;%(AdditionalDependencies)**
Ignore All Default Libraries	
Ignore Specific Default Libraries	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded Dlls	
Assembly Link Resource	

Additional Dependencies
Specifies additional items to add to the link command line. [i.e. kernel32.lib]

Figure 10. Linker's Additional Dependencies

11. Now, we can add the assembly file to the project. Simply right-click on **Source Files → Add → Existing Item**.

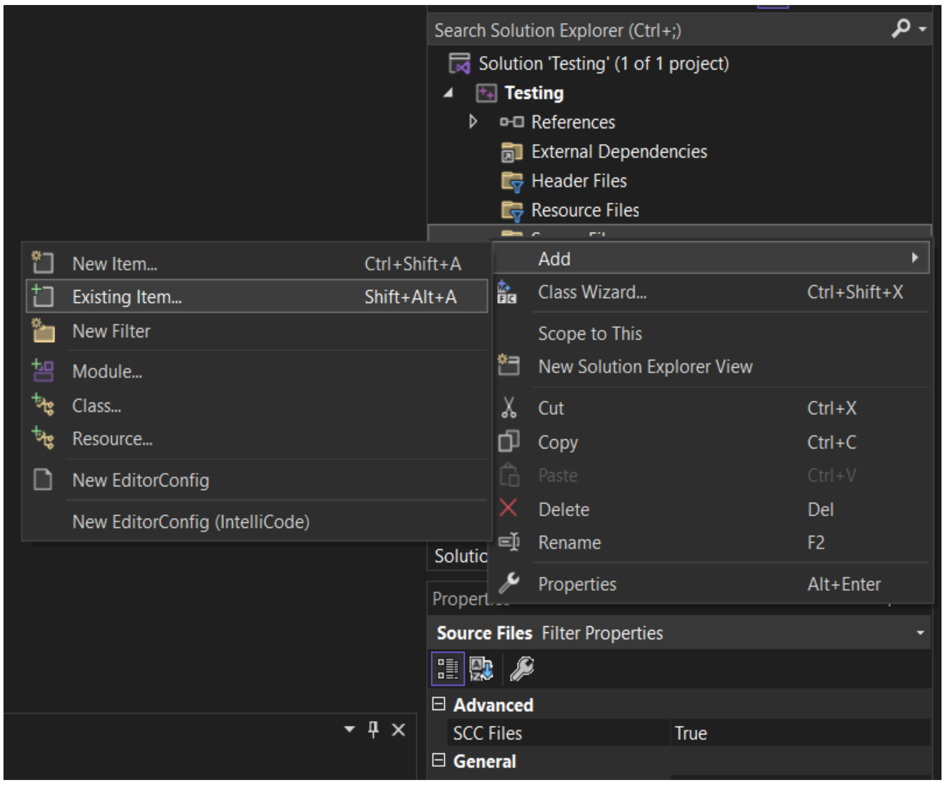
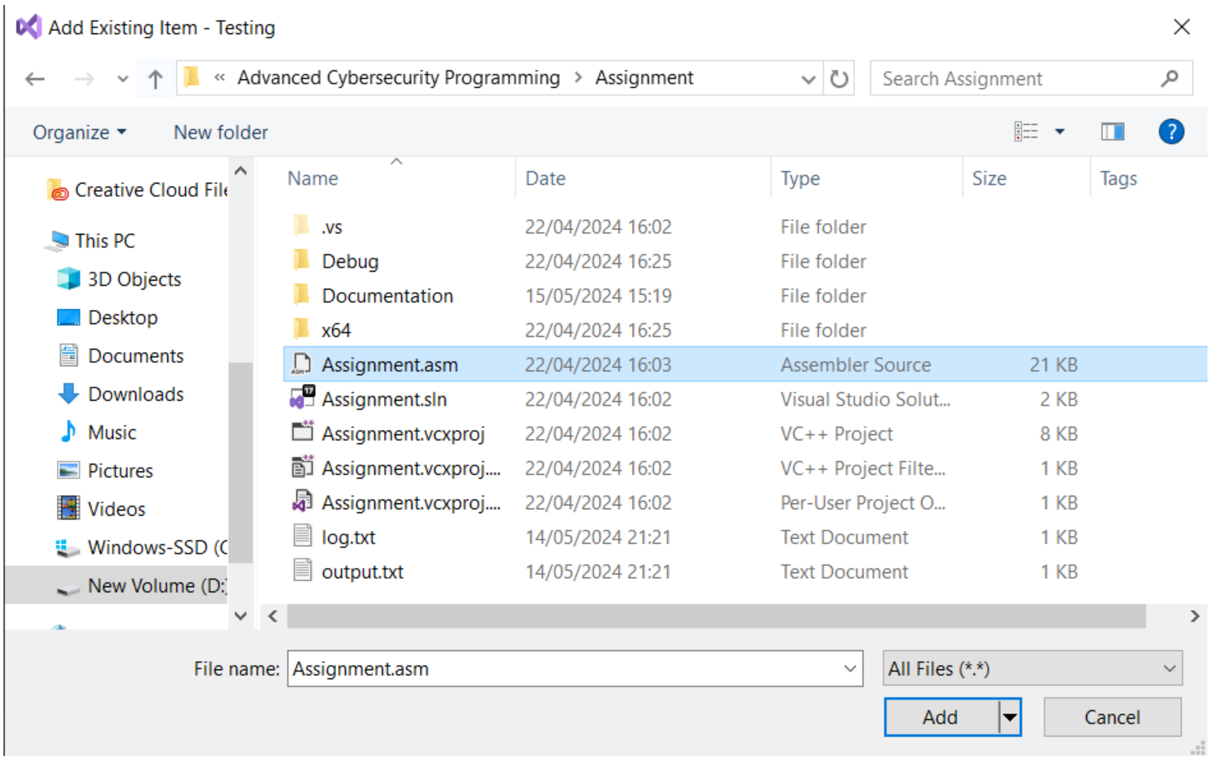


Figure 11. Add Item

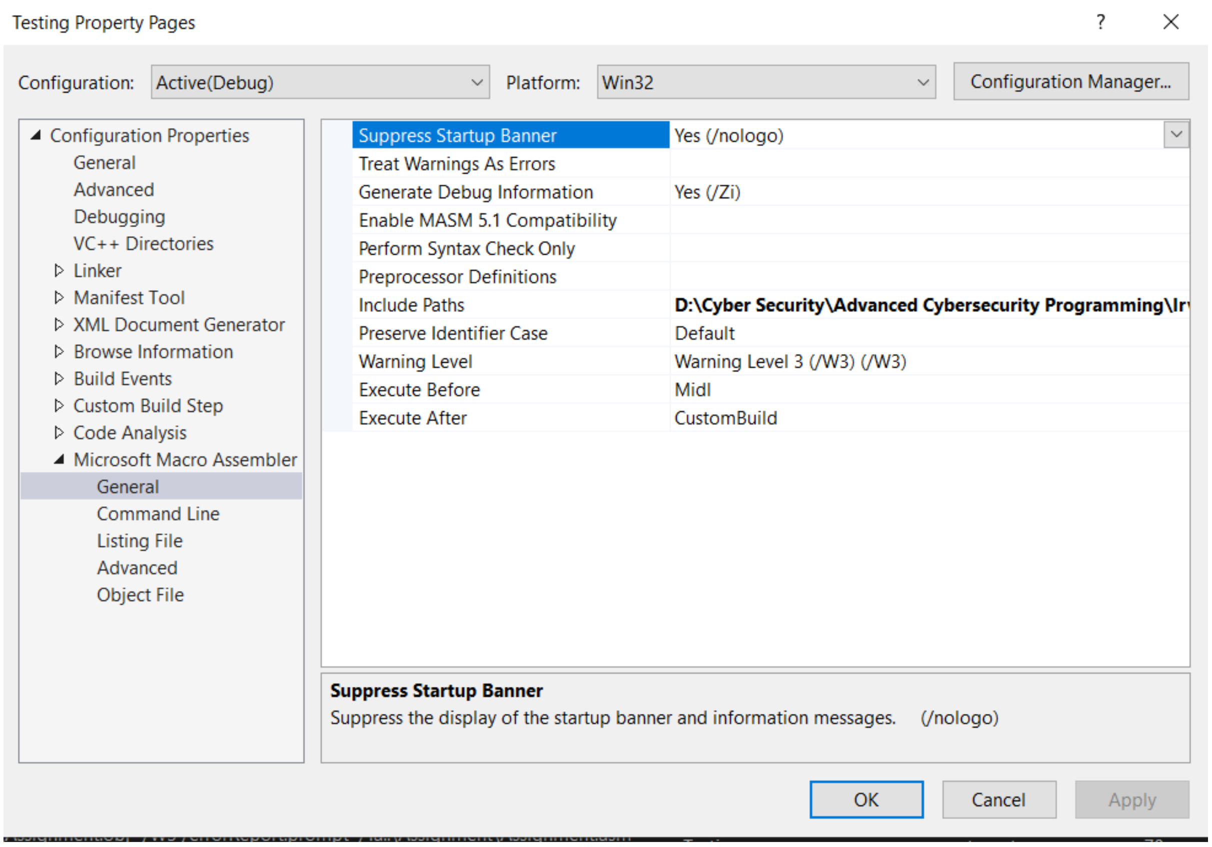
12. Locate the **assembly file**, then click **Add**.



Name	Date	Type	Size	Tags
.vs	22/04/2024 16:02	File folder		
Debug	22/04/2024 16:25	File folder		
Documentation	15/05/2024 15:19	File folder		
x64	22/04/2024 16:25	File folder		
Assignment.asm	22/04/2024 16:03	Assembler Source	21 KB	
Assignment.sln	22/04/2024 16:02	Visual Studio Solut...	2 KB	
Assignment.vcxproj	22/04/2024 16:02	VC++ Project	8 KB	
Assignment.vcxproj....	22/04/2024 16:02	VC++ Project Filte...	1 KB	
Assignment.vcxproj....	22/04/2024 16:02	Per-User Project O...	1 KB	
log.txt	14/05/2024 21:21	Text Document	1 KB	
output.txt	14/05/2024 21:21	Text Document	1 KB	

Figure 12. Add Assembly File

13. After successfully adding the assembly file, go back to the **Properties** by **right-clicking** on the project name → **Properties**. Then, navigate to **Microsoft Macro Assembler**. In the **General** section, input your **Irvine** library path into the **Include Paths**.



Suppress Startup Banner	Yes (/nologo)
Treat Warnings As Errors	
Generate Debug Information	Yes (/Zi)
Enable MASM 5.1 Compatibility	
Perform Syntax Check Only	
Preprocessor Definitions	
Include Paths	**D:\Cyber Security\Advanced Cybersecurity Programming\Ir**
Preserve Identifier Case	Default
Warning Level	Warning Level 3 (/W3) (/W3)
Execute Before	Midl
Execute After	CustomBuild

Suppress Startup Banner
Suppress the display of the startup banner and information messages. (/nologo)

Figure 13. Microsoft Macro Assembler Path Configuration

Structure Chart

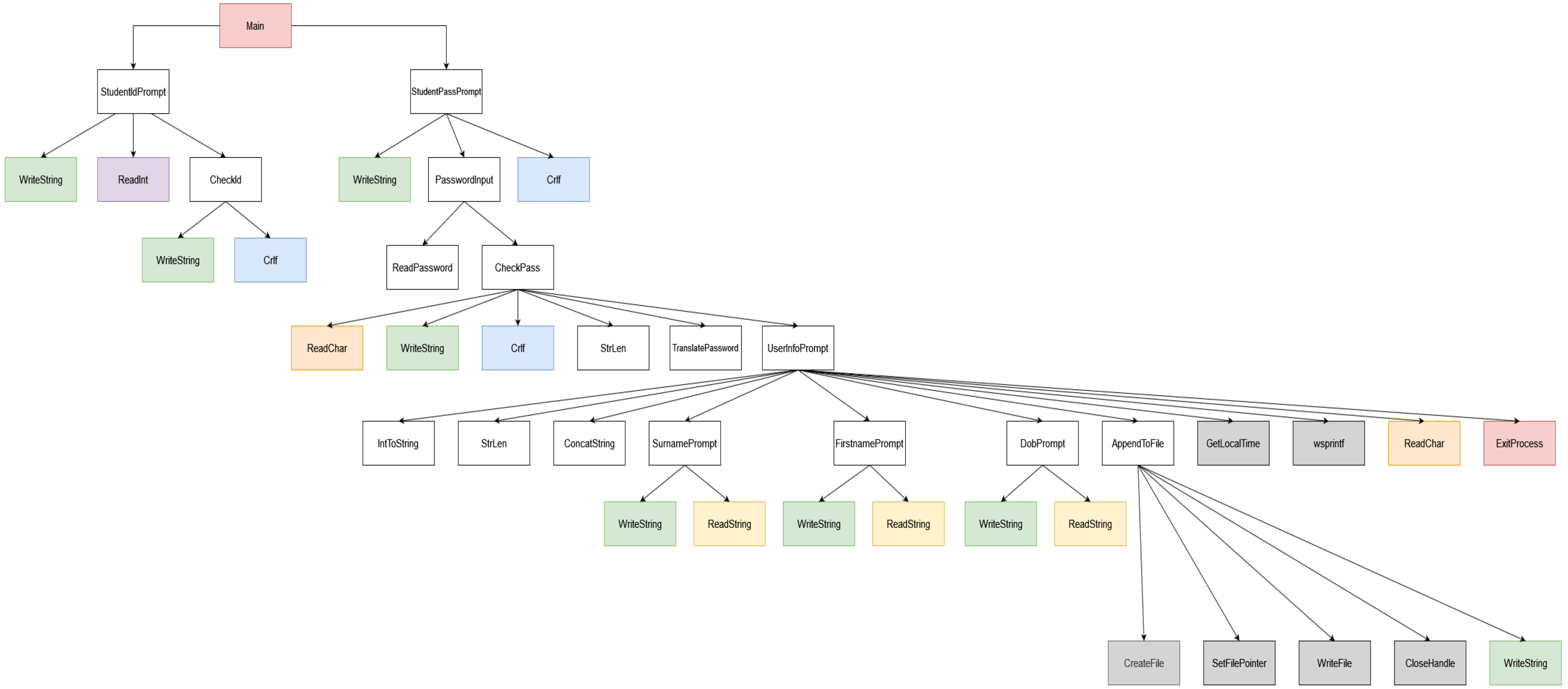


Figure 14. Structure Chart

There are 16 main procedures in the program, including main. The red blocks represent the start and end of the program, grey blocks represent the Windows library, while other coloured blocks are procedures from the Irvine library. The list of main procedures is as follows:

- **Main**: The main procedure serves as the starting point of the program. It calls two procedures: StudentIdPrompt and StudentPassPrompt.
- **StudentIdPrompt**: This procedure is used to prompt a user for a student ID in the range of 1-30. StudentIdPrompt will call CheckId to verify the student ID's validity.
- **StudentPassPrompt**: This procedure is used to prompt a user for a password.
- **CheckId**: This procedure checks if the student ID is in the range of 1-30.
- **PasswordInput**: This procedure is a continuation of StudentPassPrompt and is used to input the password into the buffer and check if it matches. It also calls ReadPassword and CheckPass
- **ReadPassword**: This procedure is used to input a password without displaying it on the screen.
- **CheckPass**: This procedure checks if the password is correct or not. CheckPass calls TranslatePassword and UserInfoPrompt if the password is correct.
- **StrLen**: This procedure retrieves the length of a string and returns ECX as the result. StrLen is called in CheckPass and UserInfoPrompt procedures.
- **TranslatePassword**: This procedure encrypts and decrypts passwords using XOR encryption.
- **UserInfoPrompt**: This procedure prompts a user for their surname, firstname, and date of birth. It writes all the information, including the ID and password to a file and creates a log as well. UserInfoPrompt also calls six main procedures: IntToString, ConcatString, SurnamePrompt, FirstnamePrompt, DobPrompt, and AppendToFile.
- **IntToString**: This procedure converts an integer to a string.
- **ConcatString**: This procedure concatenates or appends two strings.
- **SurnamePrompt**: This procedure prompts a user for their surname.
- **FirstnamePrompt**: This procedure prompts a user for their firstname.
- **DobPrompt**: This procedure prompts a user for their date of birth.
- **AppendToFile**: This procedure appends data to a file.

The description for Irvine library procedures is as follows:

- **WriteString**: This procedure writes a string from the EDX register to the console. It is represented by the green block in the chart.
- **ReadInt**: This procedure reads a 32-bit signed integer from input and returns the value in EAX. It is represented by the purple block.
- **Crlf**: This procedure writes a carriage return/line feed to the console, so a new line will be created on the console. It is represented by the blue block.
- **ReadChar**: This procedure reads a character from the input and returns the character to the AL register. It is represented by the orange block.
- **ReadString**: This procedure reads a string from the input and returns the string to the buffer pointed to by EDX. It is shown by yellow block.

The Windows library procedures used by the program are as follows, including the procedure to end the program (ExitProcess):

- **GetLocalTime**: This procedure is used to retrieve the current local date and time.
- **wsprintf**: This procedure is used for formatting the date and time and printing it to a file.
- **CreateFile**: This procedure creates a file if it does not exist and can also append data to the file.
- **SetFilePointer**: This procedure sets the pointer to the end of the file.
- **WriteFile**: This procedure writes data to a file.
- **CloseHandle**: This procedure closes the handle that has been used by CreateFile, SetFilePointer, and WriteFile.
- **ExitProcess**: This procedure terminates the program. It is represented by the red block in the middle of the chart.