Development Environment

Game Programming Series

Visual Studio 2022 Edition

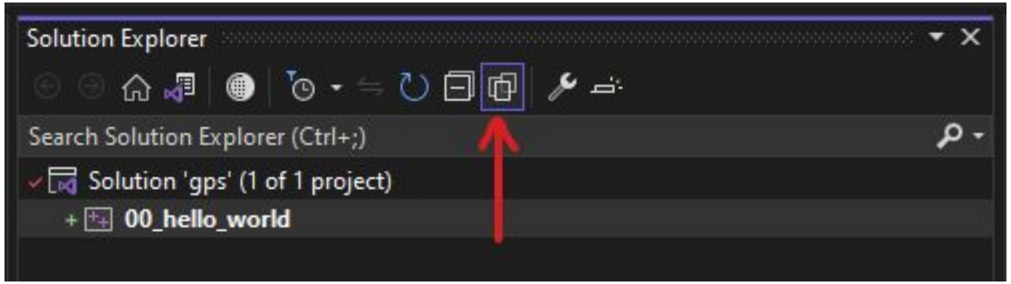
- Visual Studio is the standard development environment for C, C++ and C# projects on Windows
- Go on https://visualstudio.microsoft.com
- Install the **Community** Edition
- Open the installer
- Select the “Desktop development with C++” workload
- Finish the installation process

Create a Test Project

- Open Visual Studio
- Create New Project
- Select the C++ Empty Project
- Choose a Solution Name, a Location and a Project Name
  - Solution: collection of related projects
  - Project: collection of source files and related assets that are built together
  - Location: where the project is locally stored on the hard drive
    - I usually use C:\Dev or C:\Repos
    - Avoid white spaces in your paths. They may cause problems
- Create

Show All Files

- When using Visual Studio, we use the Solution Explorer to browse through the Solution
- This, by default, doesn't mimic the project directory structure
- This, in my opinion, is very confusing
- I always enable the "Show All Files" option

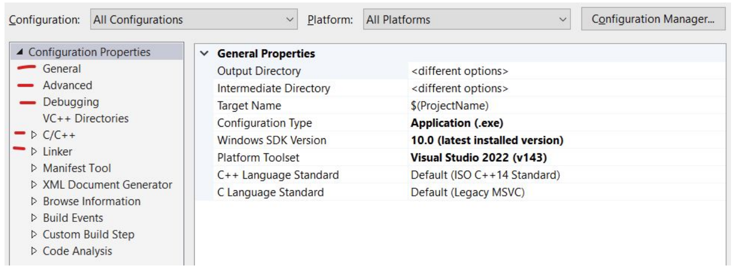

Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'gps' (1 of 1 project)
00_hello_world

Basic Folders

- To add a new folder, Right Click on your Project > Add > New Folder
- Create “src” folder
    - This is for all our .cpp files, and optionally for a **precompiled header**
- Create “include” folder
- Create “include\project-name” or “include\solution-name” folder
    - This is for all our .h files

Properties

- We can edit the Project Properties by Right Clicking on our Project in the Solution Explorer and selecting Properties
- Properties are, for us, a user friendly interface with the msvc C\C++ compiler and linker
- When editing properties, make sure to select the right Configuration and Platform. Usually All Configurations and All Platforms
- The most important sections are
    - General
    - Advanced
    - Debugging
    - C/C++
    - Linker



Configuration: All Configurations
Platform: All Platforms

General Properties

Output Directory	<different options>
Intermediate Directory	<different options>
Target Name	$(ProjectName)
Configuration Type	**Application (.exe)**
Windows SDK Version	**10.0 (latest installed version)**
Platform Toolset	**Visual Studio 2022 (v143)**
C++ Language Standard	Default (ISO C++14 Standard)
C Language Standard	Default (Legacy MSVC)

How Properties Work

- Our project has Properties
- Our .cpp files have Properties
- Our .h files have Properties
- We set Properties on a project and these properties will be inherited by all files that belong to the project
- We can override the value inherited by a file, by setting it manually ourselves
- If we have a lot of projects in our solution that share the same configuration for properties, we can create and use property sheets from which we can inherit the shared properties
- Go to View > Other Windows > Property Manager to inspect the properties inheritance hierarchy

Properties: What I Usually Do (1)

- Disable 32 bit build
  - Configuration Manager > Active Solution Platform > Edit > x86 > Remove
  - In Project Contexts > Platform > Edit > Win32 > Remove
- Change Default Output and Intermediate Directories
  - The default folder for temporary build files is inside the project folder
  - Since I use the “Show All Files” option, this is annoying
  - General > Output Directory > Edit > $(SolutionDir)\bin\$(ProjectName)-$(Platform)-$(Configuration)\
  - General > Intermediate Directory > $(SolutionDir)\tmp\$(ProjectName)-$(Platform)-$(Configuration)\
  - **Remember** to .gitignore them!

Properties: What I Usually Do (2)

- Character Set
  - Advanced > Character Set > Use Multi-Byte Character Set
- Include Directory
  - C/C++ > General > Additional Include Directories > $(ProjectDir)\include
- Multi Processor Compilation
  - C/C++ > General > Multi-Processor Compilation > Yes
- Preprocessor
  - Debug Configuration
    - C/C++ > Preprocessor > Preprocessor Definitions > _PROJECT_NAME_DEBUG
  - Release Configuration
    - C/C++ > Preprocessor > Preprocessor Definitions > _PROJECT_NAME_RELEASE

Properties: What I Usually Do (3)

- Runtime Libraries
  - Debug Configuration
    - C/C++ > Code Generation > Runtime Library > Multi-threaded Debug
  - Release Configuration
    - C/C++ > Code Generation > Runtime Library > Multi-threaded
- Floating Point Model
  - C/C++ > Code Generation > Floating Point Model > Fast
- Windows Subsystem
  - Linker > System > SubSystem > Windows

Precompiled Header (1)

- Speed up C++ compile time
- Collect the headers that you use the most and don’t change often into a precompiled header
- This gets precompiled into an intermediate form that is faster to process by the compiler
  - This precompilation phase takes time
  - If you change your precompiled header all the time, it’s not worth it
- This intermediate form is cached, and used for later builds
- Enable the use of a Precompiled Header
  - C/C++ > Precompiled Headers > Precompiled Header > Use
  - C/C++ > Precompiled Headers > Precompiled Header File > pch.h

Precompiled Header (2)

- In the src folder, create a pch.h and a pch.cpp files
- In pch.cpp, include pch.h
- In pch.h collect all the headers that you commonly use (C++ standard lib)
- In the Solution Explorer, Right Click on pch.cpp > Properties > C/C++ > Precompiled Headers > Precompiled Header > Create
- We have to include pch.h at the start of every .cpp file that we have set to use pch.h as a precompiled header
- If we don’t do so, we can’t compile the project
- If for some .cpp files we don’t want to use our pch.h file, override their property for Precompiled Header to Not Using Precompiled Headers

Windows.h

- We are directly interfacing with the win32 API
- Hence we must include Windows.h
- Define `WIN32_LEAN_AND_MEAN` before including Windows.h to have a stripped down version of the header file
- This slightly improves compilation speed
- We have less global namespace pollution

hello.cpp and hello.h

- To add a new source file: Right Click on a Folder > Add > New Item > C++ File or Header File > Name it > Add
- Create hello.cpp and hello.h
- Write a `hello` function declaration in hello.h
- Write a `hello` function implementation in hello.cpp
  - Use `MessageBox` function to display a hello world message
  - https://learn.microsoft.com/windows/win32/api/winuser/nf-winuser-messagebox

main.cpp

- Create main.cpp
- We are using the Windows Subsystem
- The program entry point is **not** `main` but `WinMain`
  - https://learn.microsoft.com/windows/win32/learnwin32/winmain--the-application-entry-point

YouTube Channel

If you like my content and you want to support my, please subscribe to my YouTube channel https://www.youtube.com/@emanuelefranchi7087