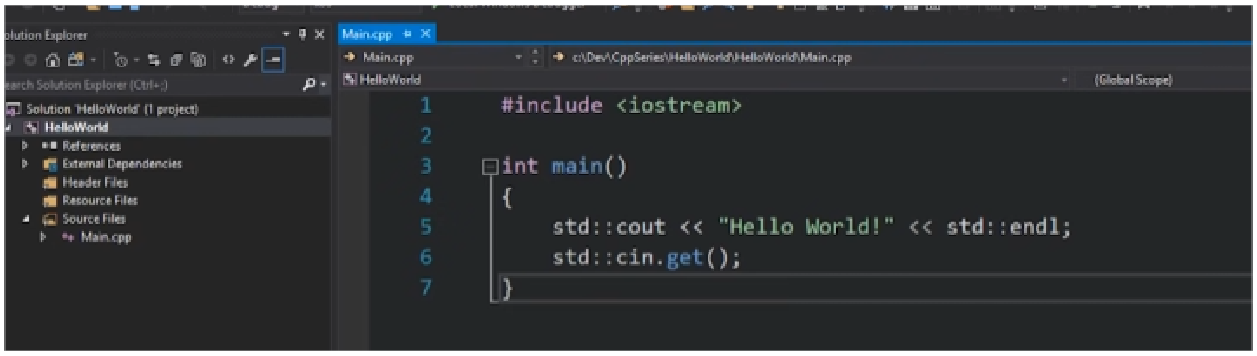
Before we start explaining how to debug, we will first explain and justify how the pipeline works for:

1. Create one or several files in C++
2. Compile them
3. And finally run a binary file.

The basic workflow of writing a simple program begins by creating and writing text files with C++ syntax, these files will be passed through a compiler, which will compile them, and if this compilation is correct, it will create a binary file.

In this example we want as a result a binary file (exe), but keep in mind that it can be either a static library (.lib) or a dynamic library (.dll).

Imagine we have a little code in the project:



```
#include <iostream>

int main()
{
    std::cout << "Hello World!" << std::endl;
    std::cin.get();
}
```

A much more extensive explanation of the phases involved in the process of compiling a C++ file can be found here:https://en.cppreference.com/w/cpp/language/translation_phases#Phase_4

We don't need to go through all of them, but the most significant ones:

- A very important phase that the compilation process goes through is that of preprocessing,

One of the first things a compiler does when it needs to compile a file is to preprocess all the preprocessor statements that exist in that file. In addition, any line thatstart for a hash (#),and a preprocessor statement. Some examples can be:

#include
#define
#pragma

For more information on preprocessor declaration types go here:

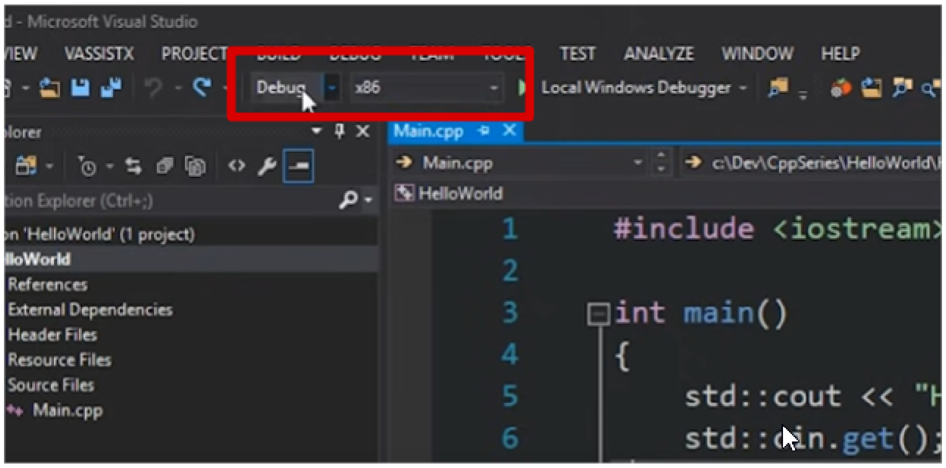
https://en.cppreference.com/w/cpp/preprocessor

In our example we find #include,what is a preprocessor statement, here the compiler will look for this file (the iostream file) and add/copy all its contents to this file.

Then, once all the preprocessor statements have been evaluated, the compiler will convert all the C++ code in the file or files into machine code.

Use:
Returning to our code, we see that we find the main, which in C++ is called the entry point, every program in C++ must have an entry point where our program begins to execute, once the program begins execution at the entry point, the following lines will be executed in order, unless there is acommand of change of execution order

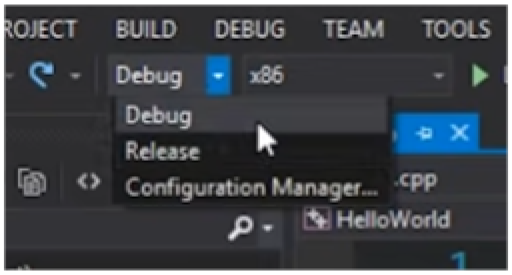
The compiler can create different types of executables, more or less optimized, with debugging information included or not, etc... how does the compiler determine the type of compilation to do? Well, Visual Studio is based on these twomenus selectable:

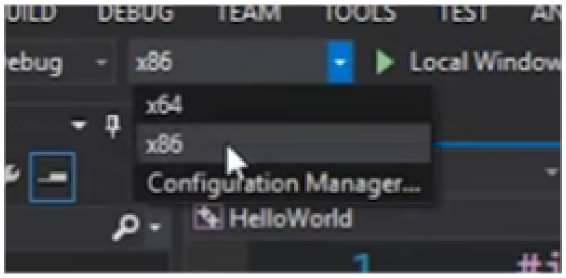


```
#include <iostream
int main()
{
    std::cout << "H
    std::cin.get();
```

The first drop-down list is related to the solution configuration, and indicates only a group of rules that apply when a project is compiled, and the second drop-down list is related to which resulting platform we are building our executable against.

By default VS defaults to Debug and x86 ( Win32 ) options. VS has by default these two build options (Debug and Release) and for the platform we can find x86 and x64 by default.





So for example, if we have Debug and x86 settings, that means our executable will contain informationdebuggable and will be generating a 32-bit application for Windows.

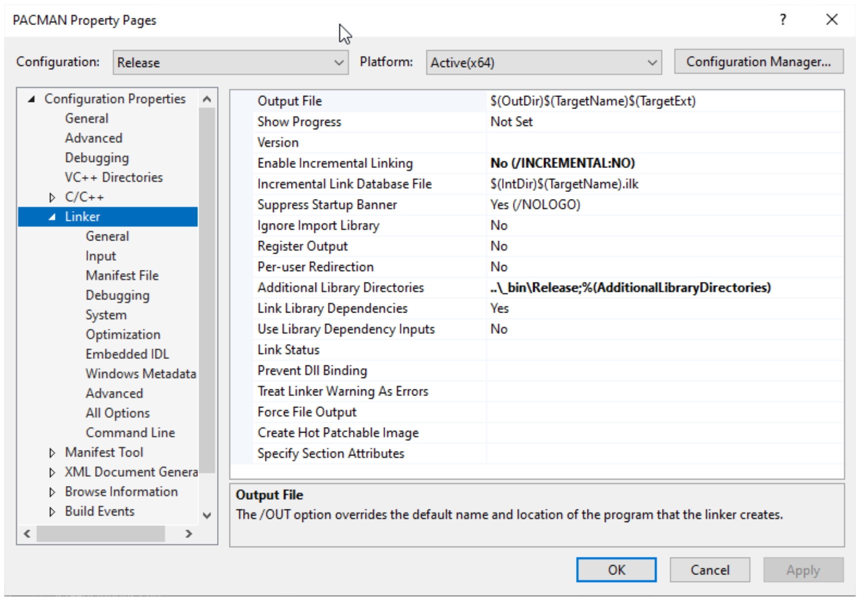
Returning to the compilation phase, it is important to mention that only the cpp files will be compiled, the .h files are never compiled, since the contents of these are added in the cpp files that include them with the #include, there it is where it will be compiled.

So in this way we will have many cpp files compiled individually, generating each of them an .obj file. Once the compiler has created all these .obj files from all the cpp files in our solution, we need to combine them all to create a single binary file (.exe), this will be the function of the Linker.

The linker basically takes all these .obj files, and matches them in such a way that we get a single file.

In VisualStudio we can configure the Linker in the "Linker" tab. Highlight:

- In the "General" section: we can define external directories to access static or dynamic libraries, in case our solution needs them.



PACMAN Property Pages

Configuration: Release   Platform: Active(x64)   Configuration Manager...

| | |
|---|---|
| Output File | $(OutDir)$(TargetName)$(TargetExt) |
| Show Progress | Not Set |
| Version | |
| Enable Incremental Linking | **No (/INCREMENTAL:NO)** |
| 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 | **.._bin\Release;%(AdditionalLibraryDirectories)** |
| 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 | |

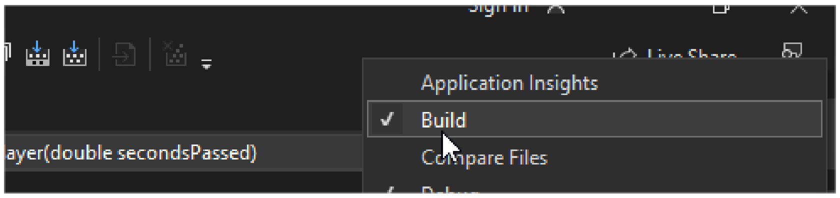
**Output File**
The /OUT option overrides the default name and location of the program that the linker creates.

# COMPILE

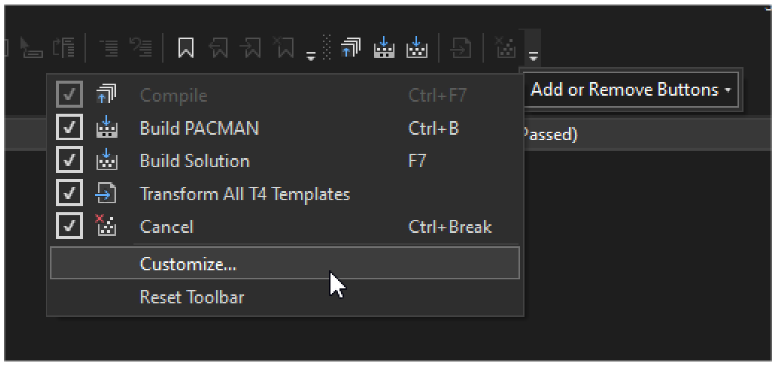
Now that we understand the phases of building our project, let's see how we can build in VS.

1. To compile a single cpp file we can do thisclicking ctrl+F7, or from the VS toolbar, this option is not found by default, to add it we will have to:

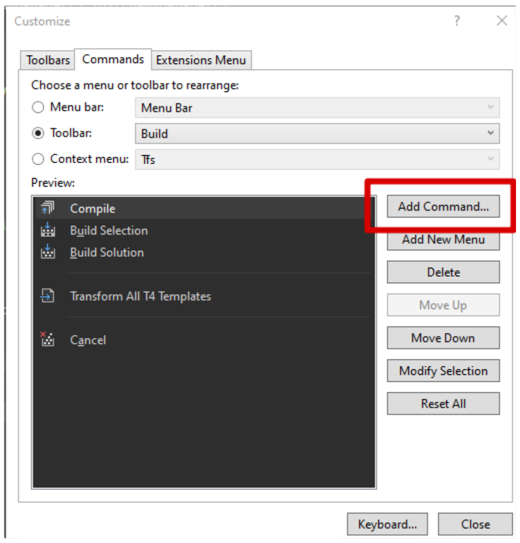
   i. By clicking the right mouse button on the toolbar, we add the "Build" section



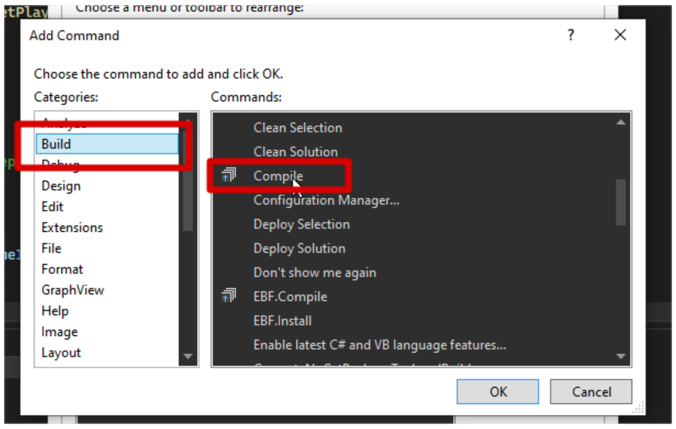
   ii. Click the drop-down list of thissection i anar a “Add or Remove Buttons’-> Customize ”:



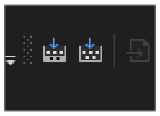
In the new window, click on "Add command":



Select the “Build” section and scroll down to select the compile button



2. Click the button to compile project or solution in itmenu superior

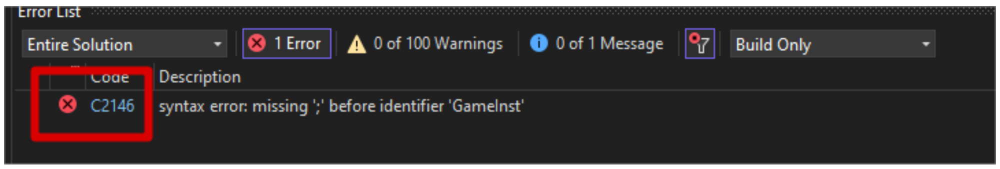


# DEBUG

It is important that you distinguish between a compilation error and a linking error, because their resolution is different.

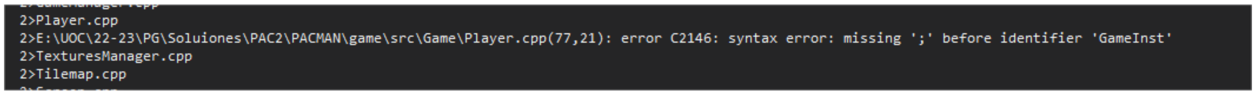
Compilation errors are basically due to an error in the syntax of the code (missing a semicolon, undeclared variable, etc...)

In VS, compilation errors appear as follows in the Error list window:



C2146 syntax error: missing ';' before identifier 'GameInst'

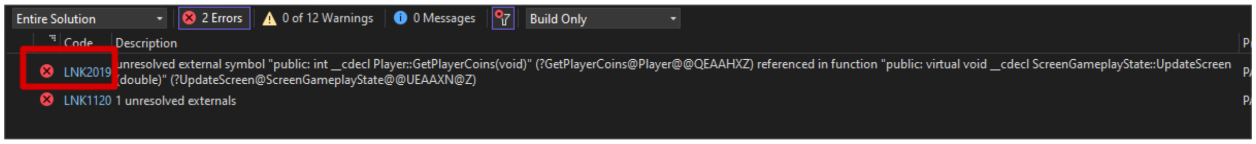
And in the output window:



```
2>Player.cpp
2>E:\UOC\22-23\PG\Soluiones\PAC2\PACMAN\game\src\Game\Player.cpp(77,21): error C2146: syntax error: missing ';' before identifier 'GameInst'
2>TexturesManager.cpp
2>Tilemap.cpp
```

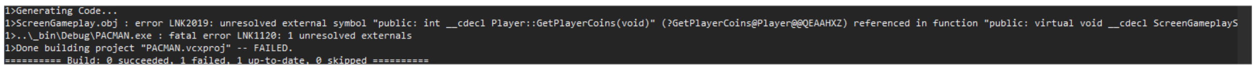
We recommend using the ouput window to consult/solve errors as it provides more information than the Error List, which can be used for an overview but not as a viable source of information.

Linking errors appear like this:



| Code | Description |
|---|---|
| LNK2019 | unresolved external symbol "public: int __cdecl Player::GetPlayerCoins(void)" (?GetPlayerCoins@Player@@QEAAHXZ) referenced in function "public: virtual void __cdecl ScreenGameplayState::UpdateScreen (double)" (?UpdateScreen@ScreenGameplayState@@UEAAXN@Z) |
| LNK1120 | 1 unresolved externals |

I to the finestra d'output:



```
1>Generating Code...
1>ScreenGameplay.obj : error LNK2019: unresolved external symbol "public: int __cdecl Player::GetPlayerCoins(void)" (?GetPlayerCoins@Player@@QEAAHXZ) referenced in function "public: virtual void __cdecl ScreenGameplayS
1>..\_bin\Debug\PACMAN.exe : fatal error LNK1120: 1 unresolved externals
1>Done building project "PACMAN.vcxproj" -- FAILED.
========== Build: 0 succeeded, 1 failed, 1 up-to-date, 0 skipped ==========
```

Mostly Linking errors occur because a cpp file cannot find the definition of a function.