



# PROPUESTA PARA EL DESARROLLO DEL PROYECTO DEL TITULADO: "LIBC++" C++ STANDARD LIBRARY"

PRESENTAN:

CABRERA RAMÍREZ GERARDO

MORALES CARRILLO GERARDO

SANTANA GONZÁLEZ JESÚS SALVADOR

NO. DE CONTROL:

171080187

171080120

171080127

# INTRODUCCIÓN:

Dado que debemos de realizar un proyecto del tema que el profesor nos ha asignado, en nuestro caso fue: “libc++” C++ Standard Library”, deberemos investigar acerca de la plataforma donde se programara y diferentes características que pueden ser implementadas (Depende de lo que el proyecto será a futuro).

- 

En este proyecto se hablará sobre las características y funciones de esta librería ya que tiene muchas cosas de las cuales se pueden aprovechar, como el poder implementarla en otros lenguajes de programación con los cuales sean compatibles. En este caso vamos a corroborar la implementación de “libc++” C++ Standard Library con el lenguaje de programación de python, al crear un proyecto en python y tratar de afirmar que en verdad se pueda compilar el código con la librería de c ++, todo esto siendo documentado paso a paso.

# DESCRIPCIÓN GENERAL:

- **Descripción general**

- libc++ es una nueva implementación de la biblioteca estándar C++ para C++ 11 y superior.
- Funciones y objetivos
- Corrección de acuerdo con la definición del estándar C++ 11.
- Ejecución rápida.
- Uso mínimo de memoria.
- Tiempo de compilación rápido.
- Pruebas unitarias extensivas.
- Diseño e implementación.
- Pruebas unitarias extensas
- El modelo de enlazador interno se puede volcar / leer en formato de texto Se pueden conectar funciones de enlace adicionales "a través de" CPU de factor específico y código específico del sistema operativo
- Según la [página clang libc++](#), (s/e) dice:
- A partir de años de experiencia (incluida la implementación de la biblioteca estándar antes), y cambios fundamentales en cómo se implementan. Por ejemplo, generalmente se acepta que construir std::string usando la "optimización de cadena corta" en lugar de usar Copy On Write (COW) es un enfoque superior para máquinas multinúcleo (particularmente en C++ 11, que tiene referencias rvalue). Se determinó que romper la compatibilidad de ABI con versiones antiguas de la biblioteca era fundamental para lograr los objetivos de rendimiento de libc++. Recuperado de. <https://libcxx.llvm.org/>

La línea principal libstdc++ cambio a GPL3, que es una licencia que los desarrolladores de libc++ no pueden usar. libstdc++ 4.2 (la última versión de GPL2) se puede extender de forma independiente para admitir C++ 11, pero esta será una rama del código base (para proyectos, generalmente se considera peor que comenzar un nuevo código de forma independiente). Otro problema con la librería es que está integrado con el desarrollo de G++ y, a menudo, está relacionado con la versión correspondiente de

# JUSTIFICACIÓN:

- La realización de este proyecto tiene como finalidad lo siguiente: “Utilizando el intérprete de Python en Visual Studio comprobaremos que se pueda implementar en C++ y que en realidad se pueden hacer implementaciones de python y c++”
- Hemos escogido la **metodología tradicional** porque es la que más se acopla ya que nuestro proyecto es de investigación y al final se implementarán las técnicas encontradas para pasar a la etapa de prueba y error

## PROBLEMA ESCOGIDO:

- Creación de una extensión de C++ para Python

# DESARROLLO:

## Desarrollo:

- La página de microsoft nos dice que los requerimientos para realizar la comprobación son:
- Visual Studio 2017 o versiones posteriores con las cargas de trabajo Desarrollo para el escritorio con C++ y Desarrollo de Python instaladas con opciones predeterminadas.
- En la carga de trabajo Desarrollo de Python, seleccione también el cuadro de la derecha de Herramientas de desarrollo nativo de Python. Esta opción establece la mayor parte de la configuración descrita en este artículo. (Esta opción también incluye la carga de trabajo de C++ automáticamente).

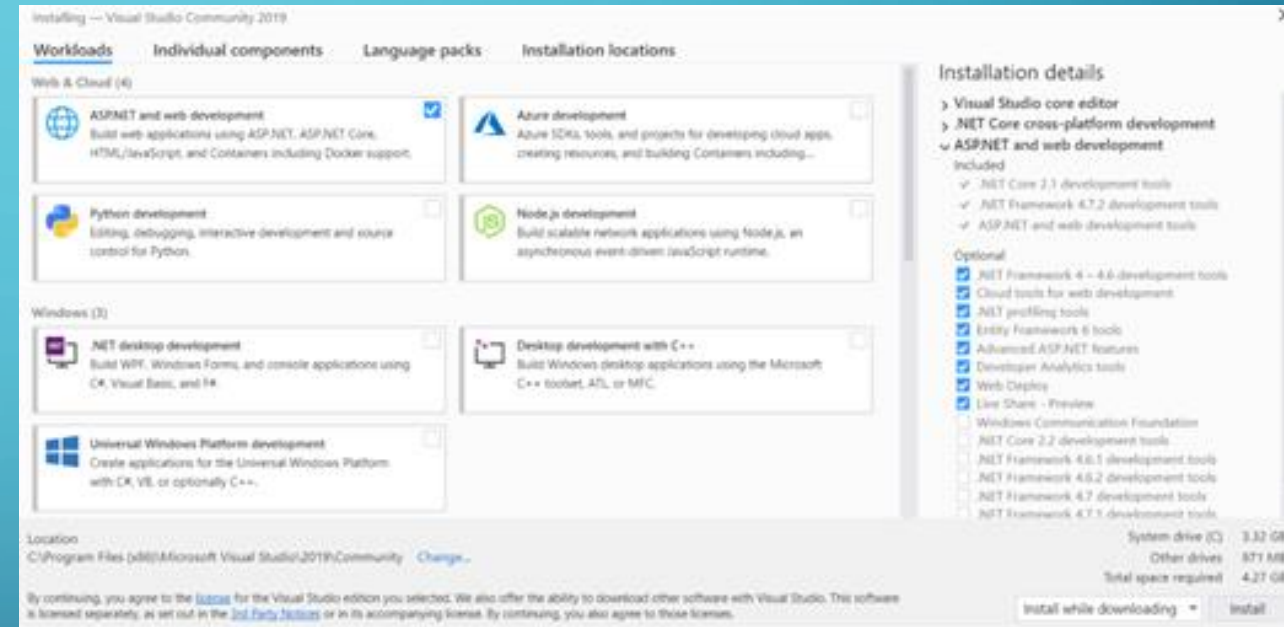
1.- Haremos la creación e implementación de c++ para phyton, revisaremos los requisitos que se necesiten:

- \*Visual studio 2017 (o posteriores) con las cargas de trabajo (Desarrollo para el escritorio c++ y desarrollo de phyton)
- \*Primero instalaremos visual studio de primera instancia, el cual será obtenido de la página de Microsoft



# PASO 1:

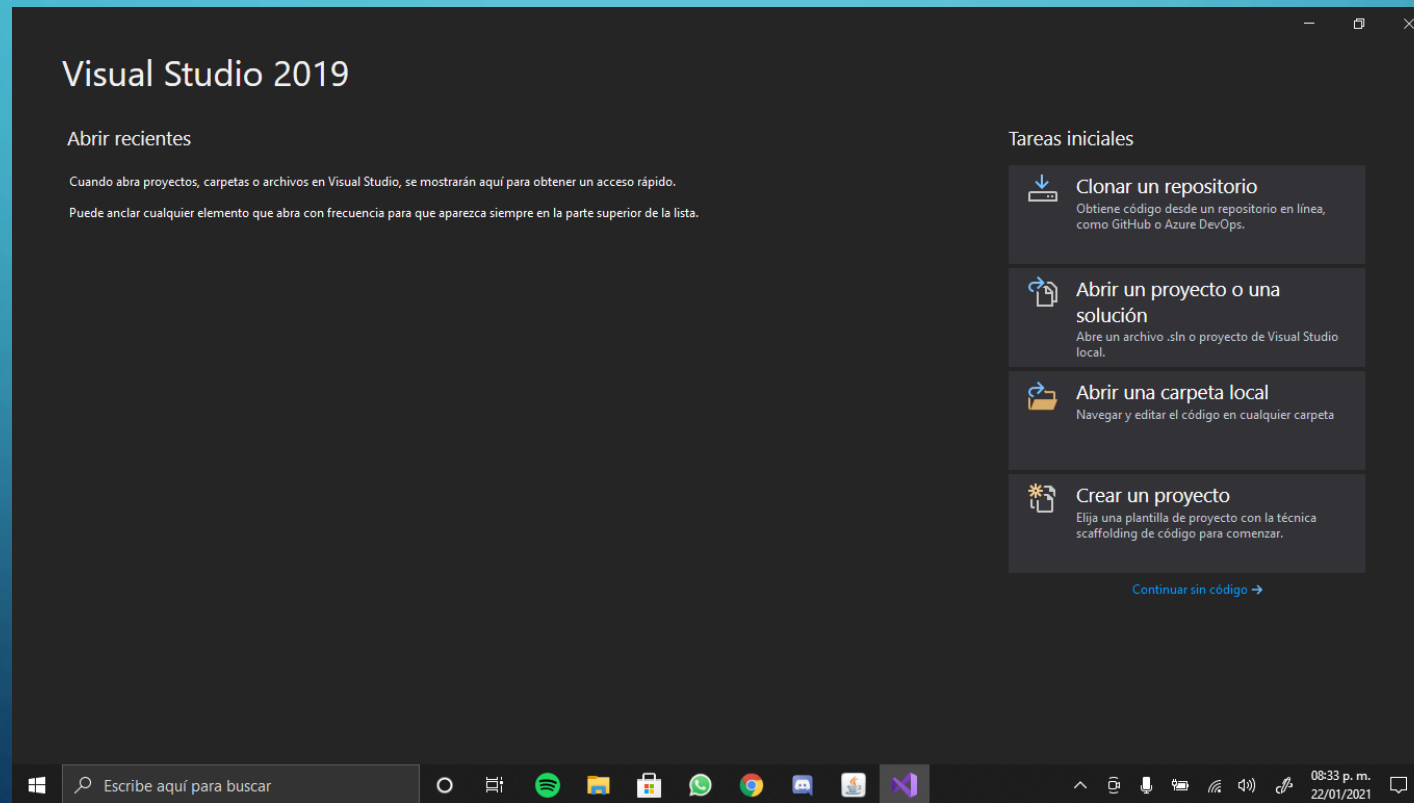
- Paso 1:
- - Comprobar los requisitos del sistema para comprobar compatibilidad con el programa.
- - Ya descargado visual studio, haremos doble click en el archivo .exe
- - A continuación, veremos una pantalla en la cual daremos en la opción “sí” para ejecutar el programa, términos y condiciones: aceptar, aparecerá una nueva ventana en donde elegiremos las cargas de trabajo que necesitamos, en este caso, serán las siguientes:
- Desarrollo para el escritorio c++ y desarrollo de python



## PASO 2:

- Esto es para obtener la compatibilidad con c y c++, cuando la instalación de visual studio se complete, daremos en el botón “Iniciar”.

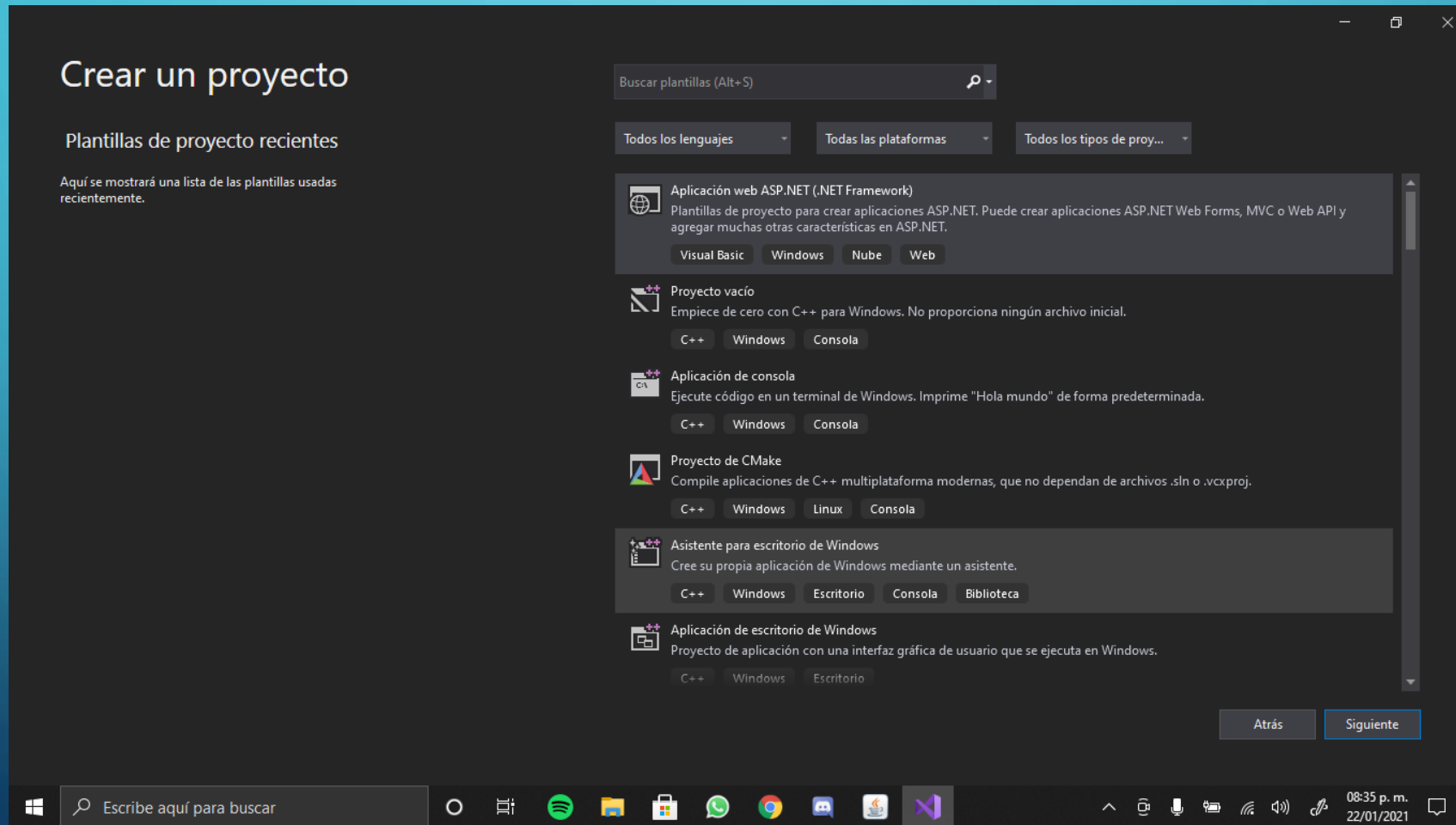
Estaremos en la ventana de inicio, elegiremos “Crear un proyecto nuevo”





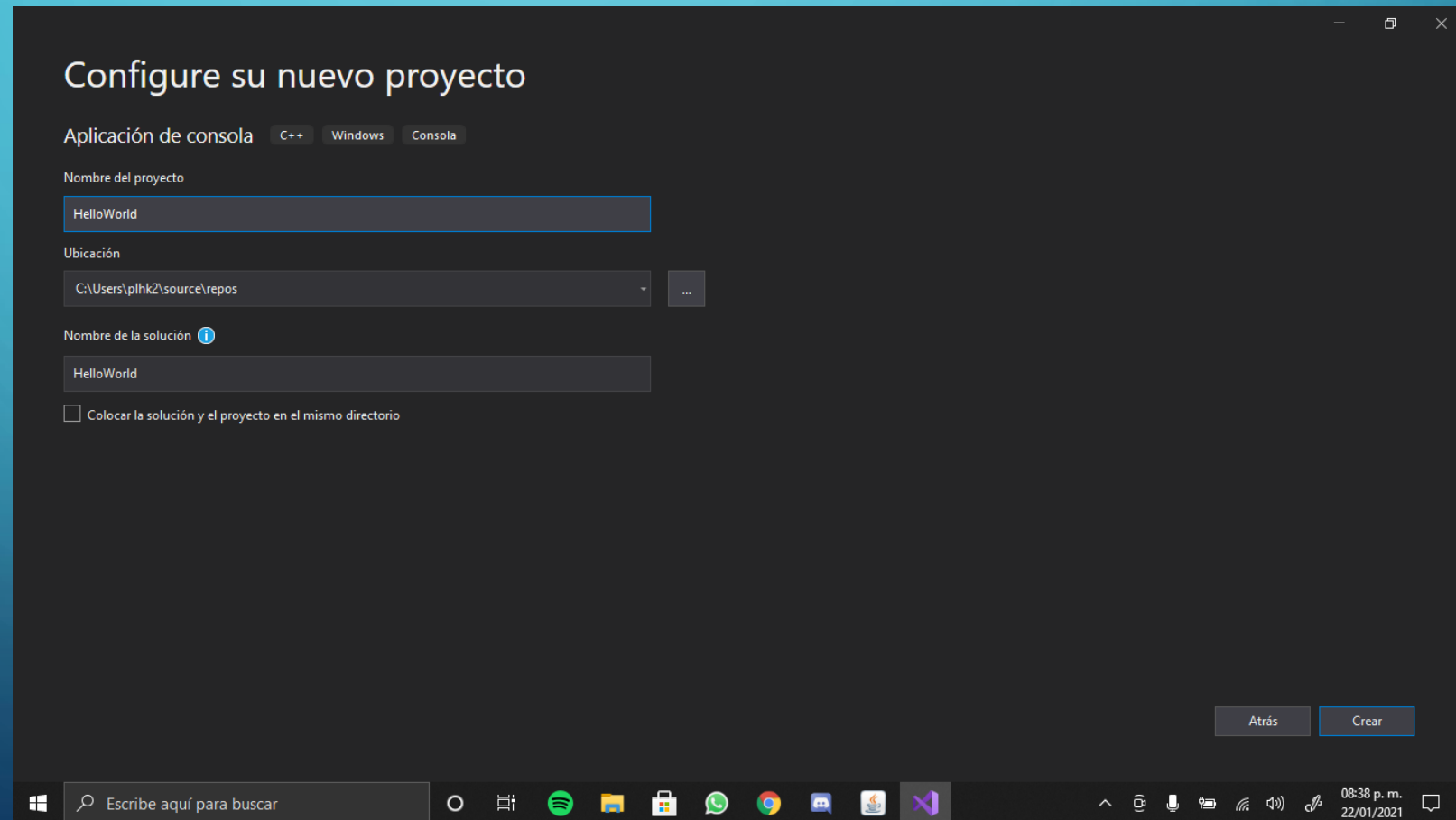
# PASO 3:

- Accederemos al cuadro de búsqueda, donde elegiremos “Proyecto vacío”



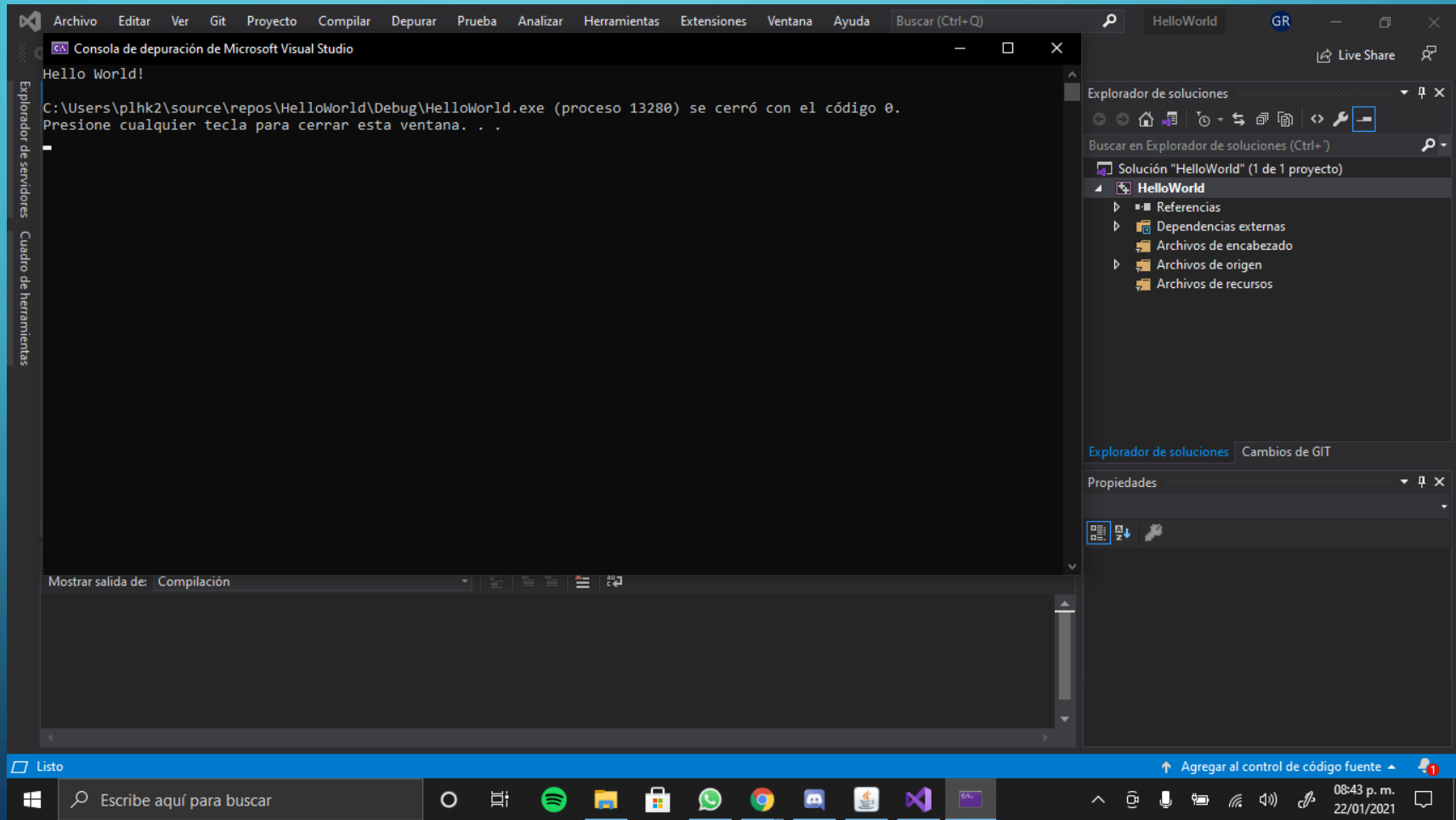
# PASO 4:

- Daremos en siguiente, ahora, seleccionaremos “console app”, se abrirá un cuadro de diálogo , “Configurar el nuevo proyecto”, haremos el conocido “Hello world”
- Daremos
- en “Crear”:



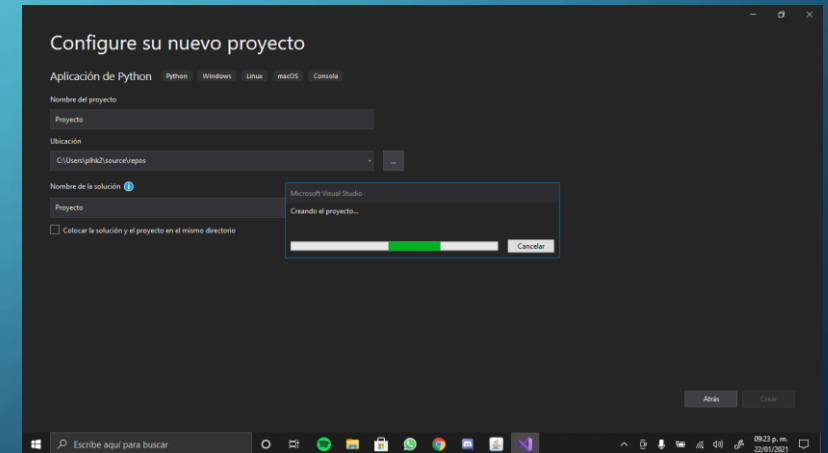
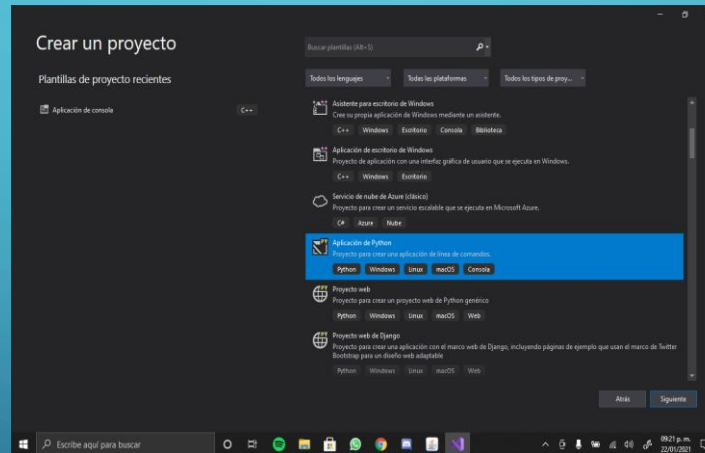
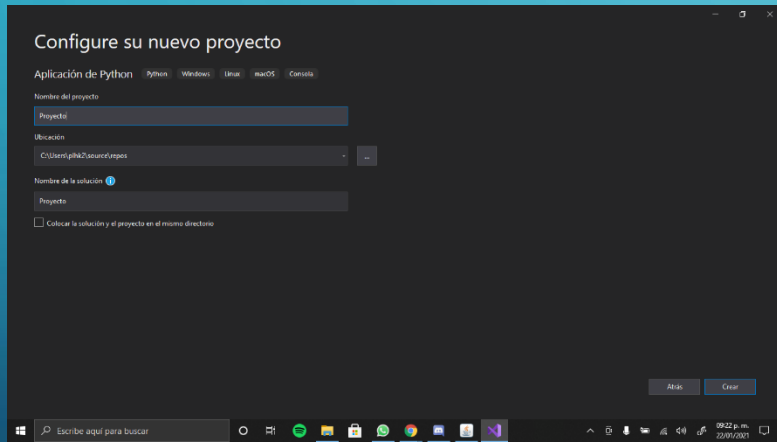
# PASO 5:

- Haremos click en el botón “Compilar”, “Compilar solución”, y para ejecutar el código, le daremos a “Depurar”, “Iniciar sin depurar”, aparecerá una nueva ventana:



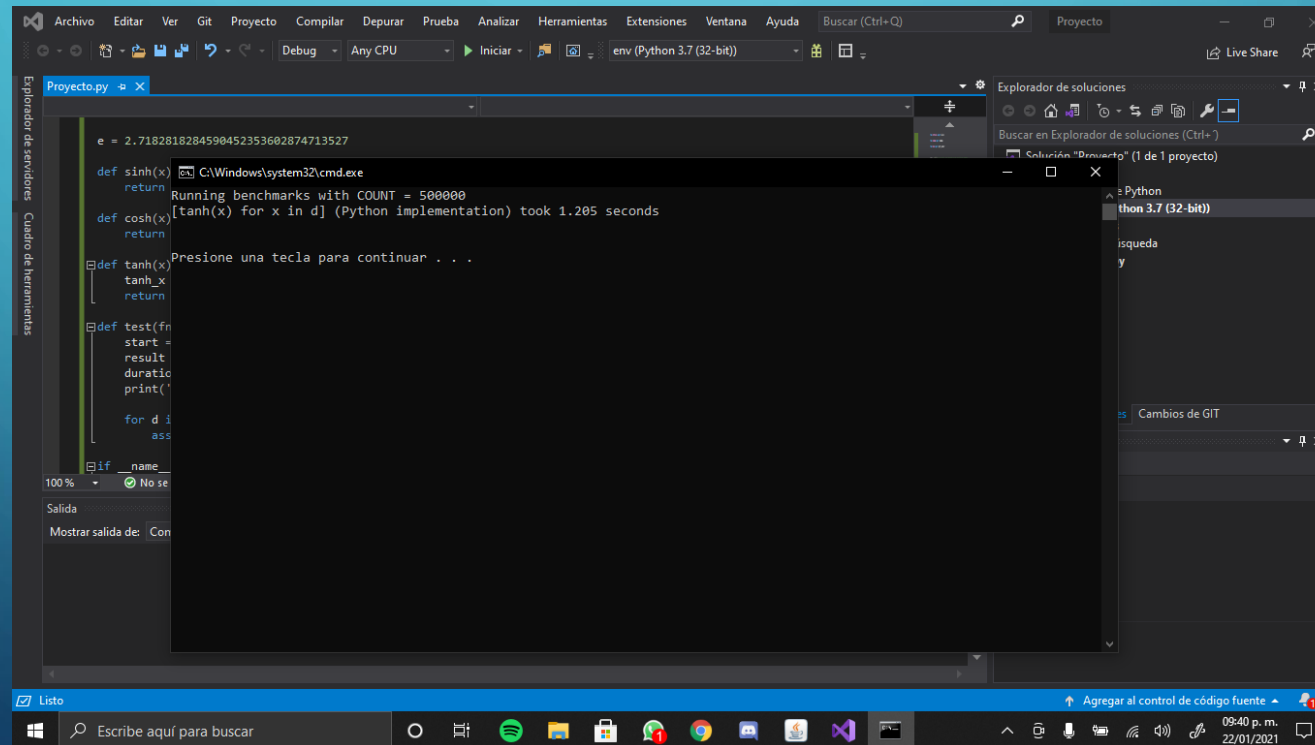
# PASO 6:

- Después de instalar Visual Studio y haber creado un proyecto de C ++, compilarlo y depurarlo, empezaremos con la creación de la extensión de C ++ para Python.
- Crearemos un proyecto de Python en Visual Studio, seleccionamos la plantilla de aplicación de Python y le daremos en siguiente, le daremos nombre y en crear



## PASO 7:

- En la página nos recomiendan usar el intérprete de Python de 32 bits. En el archivo .py pegaremos el código que nos proporciona la página el cual es una prueba comparativa del cálculo de una tangente hiperbólica, después de eso ejecutaremos el programa mediante depurar y luego iniciar sin depurar.



```
Projecto.py x
e = 2.7182818284590452353602874713527

def sinh(x):
    return ...
def cosh(x):
    return ...
def tanh(x):
    tanh_x = ...
    return tanh_x

def test(f, start, end):
    result = ...
    duration = ...
    print(f'{f.__name__:10s} took {duration:10f} seconds')
    for d in range(start, end):
        ...

if __name__ == '__main__':
    ...

100% No se
Salida
Mostrar salida de: Con
```

Explorador de soluciones

Buscar en Explorador de soluciones (Ctrl+)

Solución: "Proyecto" (1 de 1 proyecto)

Python (32-bit)

isqueda

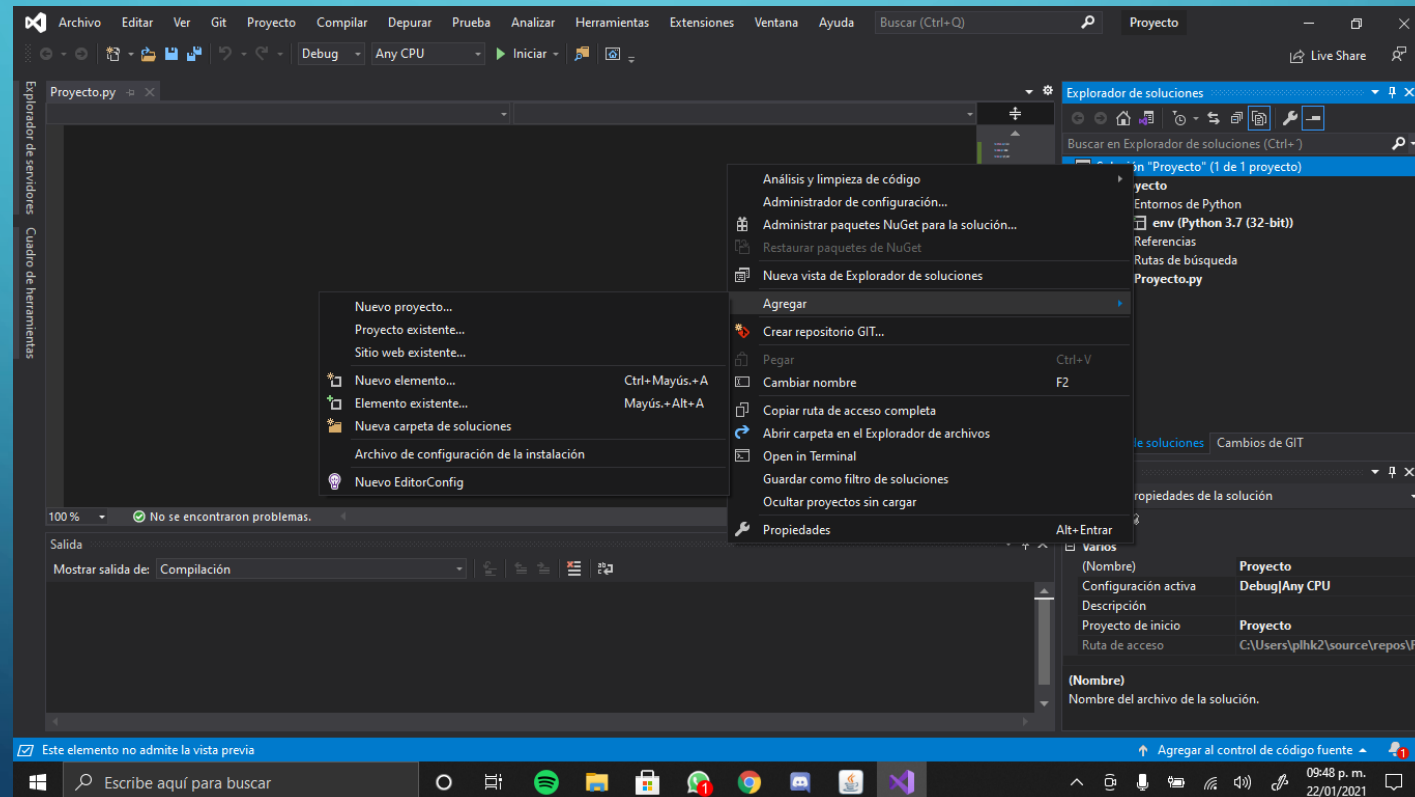
y

Cambios de GIT

09:40 p. m. 22/01/2021

## PASO 8:

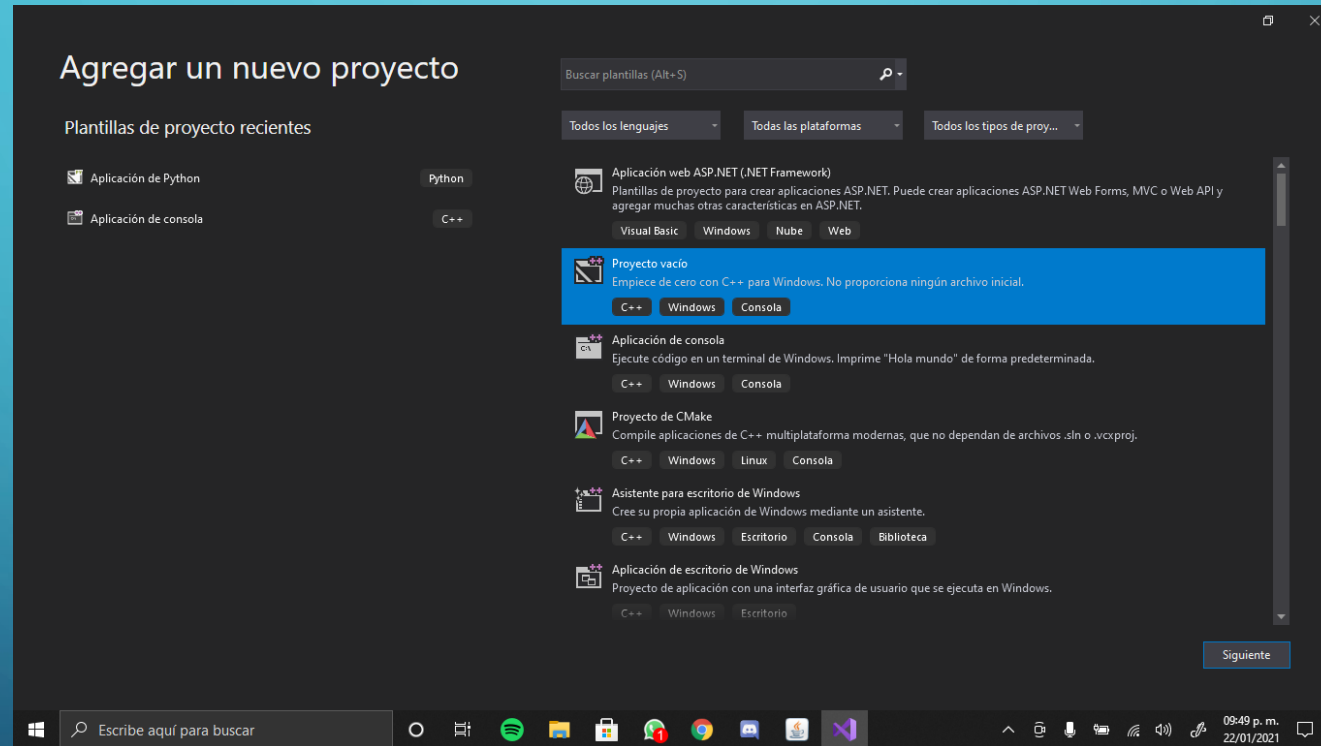
- Seguiremos las instrucciones que se nos indican para crear dos proyectos de C++ idénticos denominados “superfastcode” y “superfastcode2”. Después de eso vamos a dar clic derecho en el explorador de soluciones y seleccionemos **agregar**





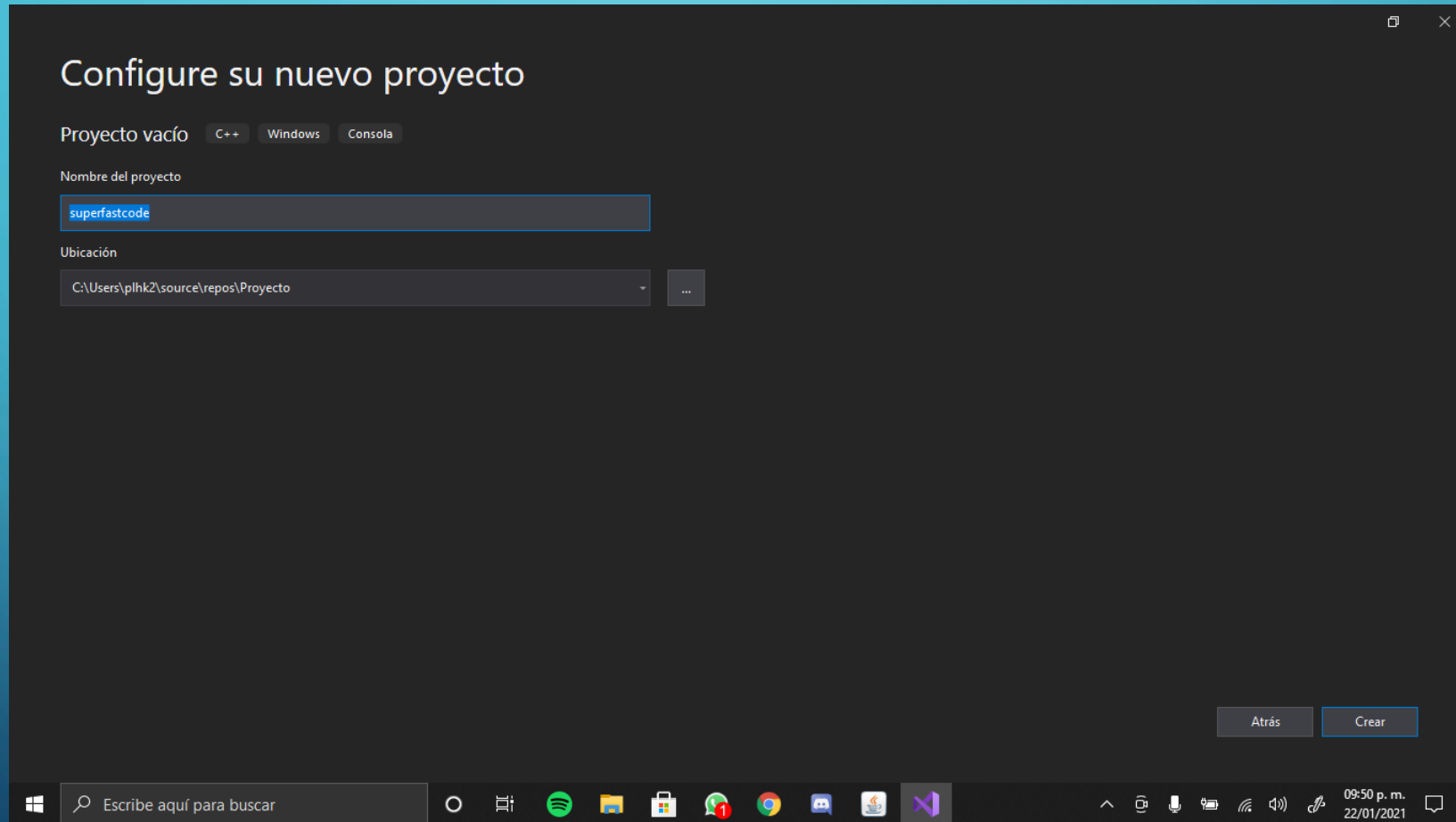
# PASO 9:

- Y luego en nuevo proyecto, buscaremos C++ y seleccionaremos proyecto vacío



# PASO 10:

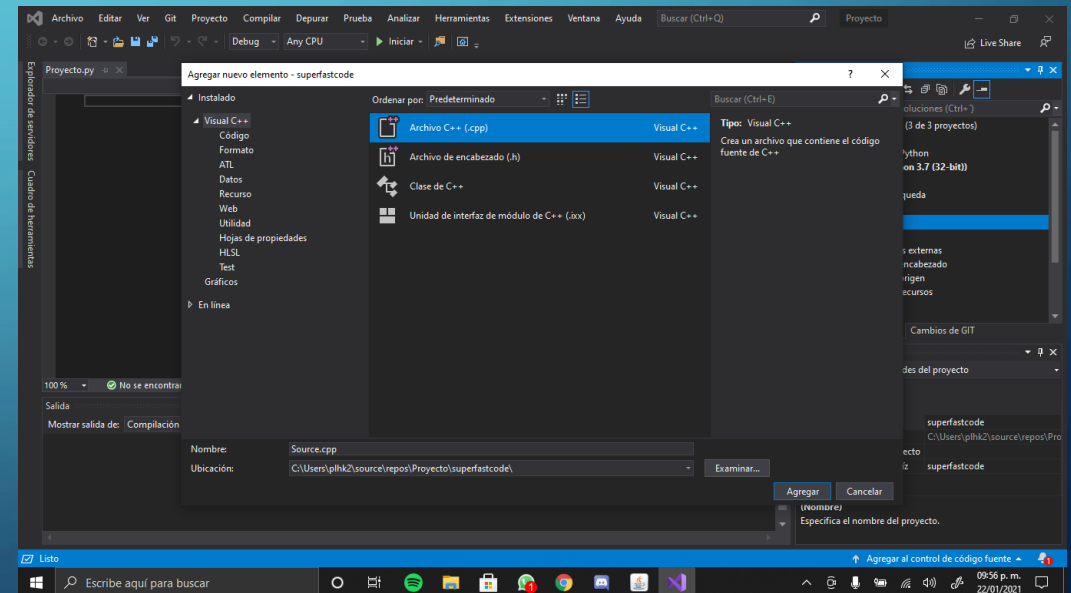
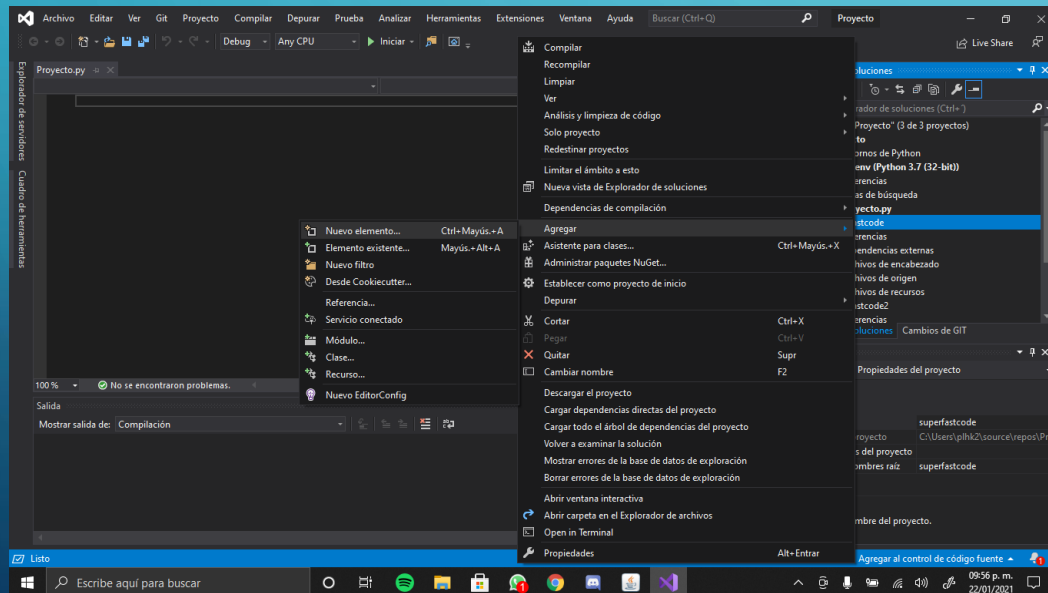
- Y le daremos el nombre de superfastcode



# PASO 11:

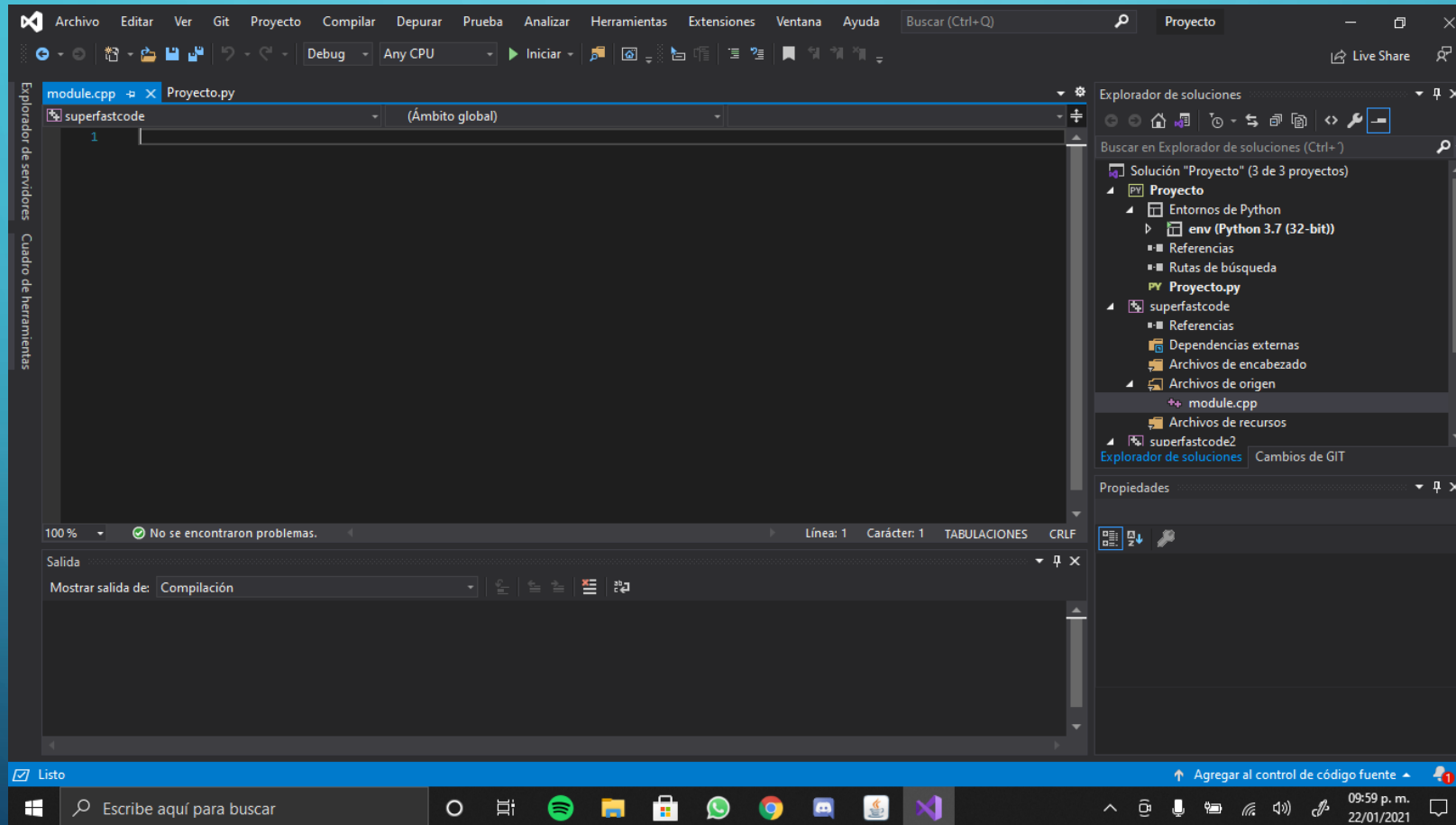
Y haremos otro proyecto con el nombre superfastcode2.

Crearemos un archivo de C++, para ello hacemos clic derecho en el archivo del código fuente y en agregar nuevo elemento y agregamos un archivo de C++ en .cpp



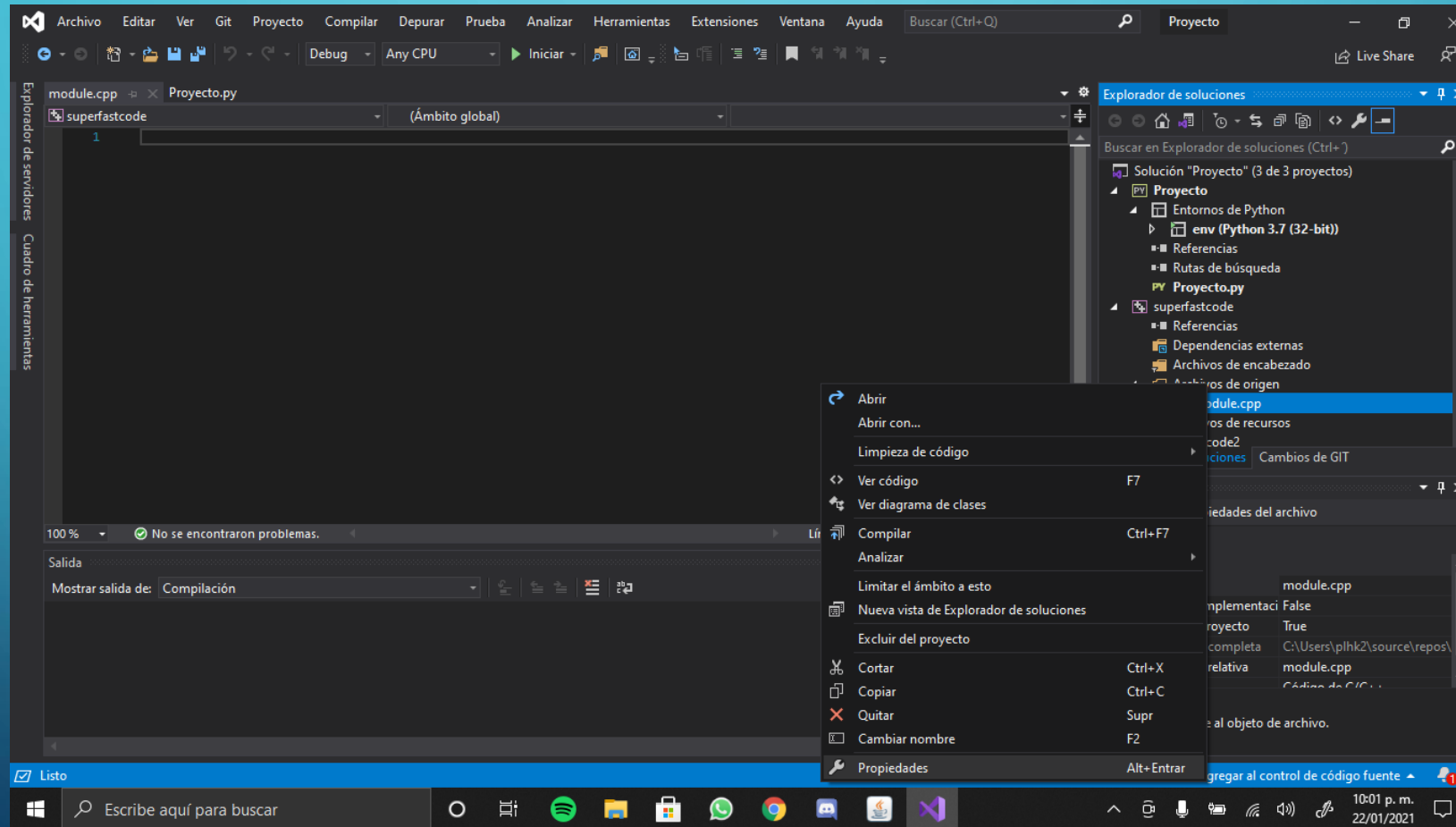
# PASO 12:

- Cambiaremos el nombre por “module.cpp”



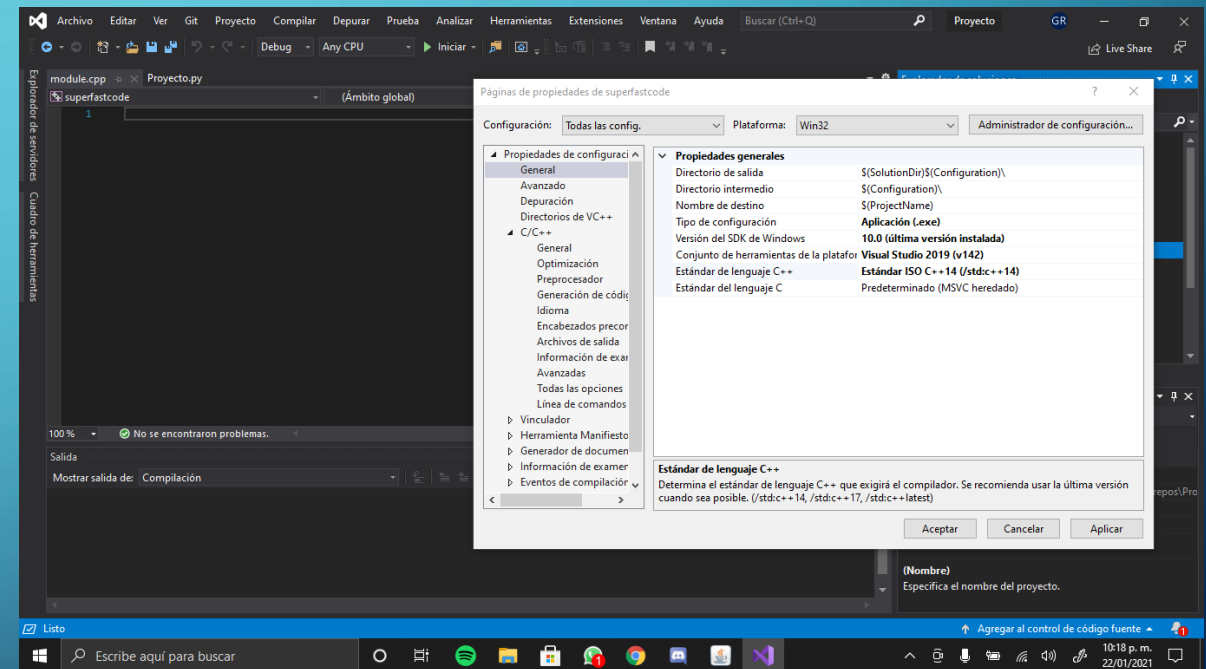
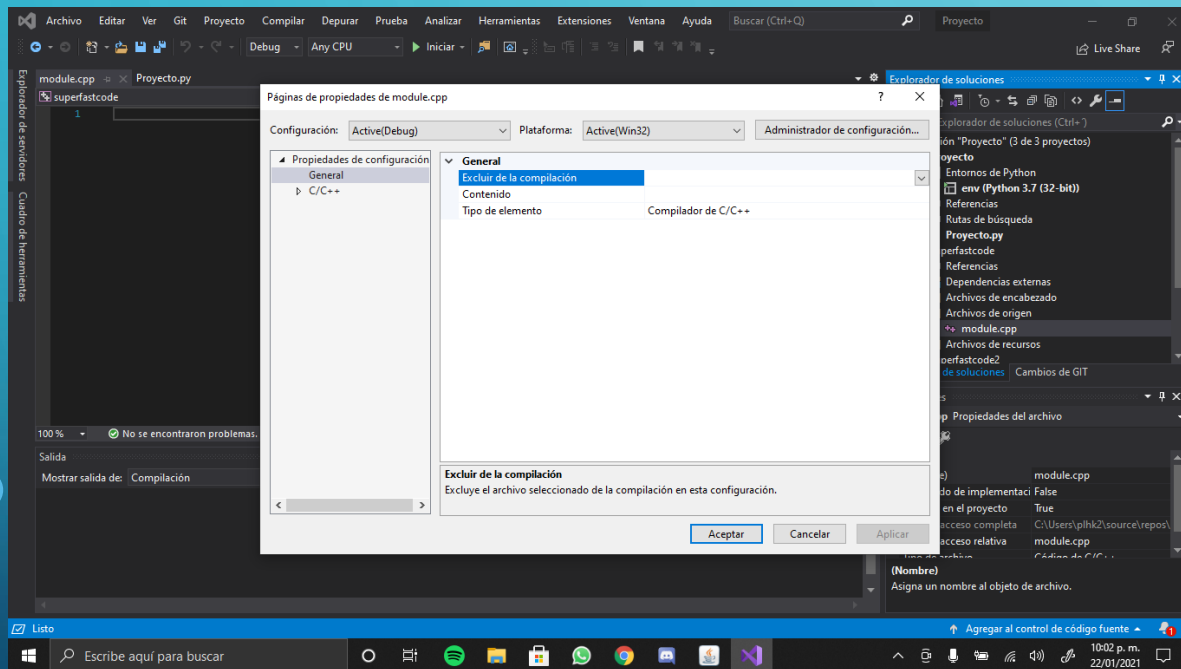
# PASO 13:

- hacemos clic derecho en el proyecto de C++ y le daremos en propiedades



# PASO 14:

- y estableceremos la plataforma como win 32



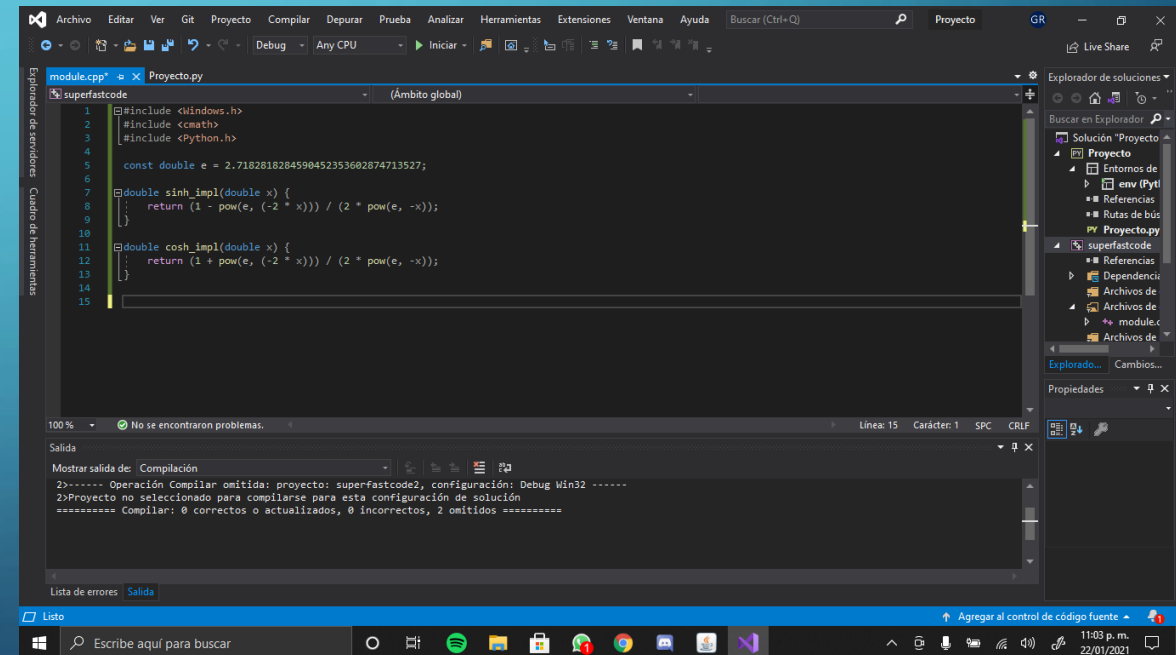


# PASO 15:

- **Convertiremos los proyectos de C ++ en extensiones para python**
- Para convertir el dll en una extensión para python debemos de modificar los métodos exportados de modo que interactúen con tipos de python, para ello debemos agregar una función que exporte el module.

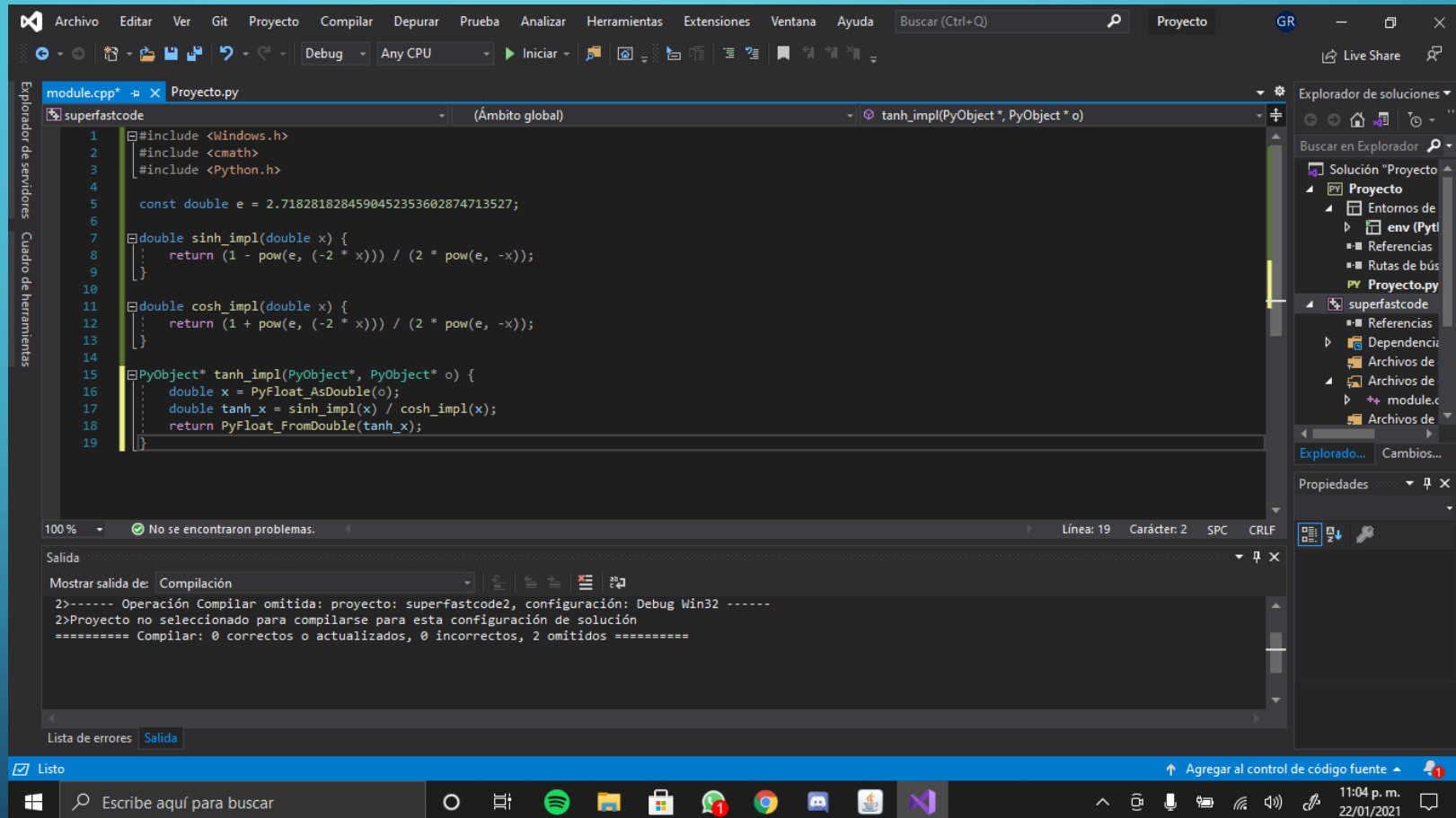
## Extensiones de C python

- 1- En la parte superior del module.cpp incluiremos python.h



# PASO 16:

- 2- Modifica el método `tanh_impl` para aceptar y devolver tipos de Python (`PyObject*`).



The screenshot shows the Visual Studio Code editor with the file `module.cpp` open. The code defines a C++ module for Python, including headers for Windows, cmath, and Python. It defines constants for `e` and `2`, and implements functions for `sinh`, `cosh`, and `tanh`. The `tanh_impl` function is modified to accept a `PyObject*` argument and return a `PyObject*`.

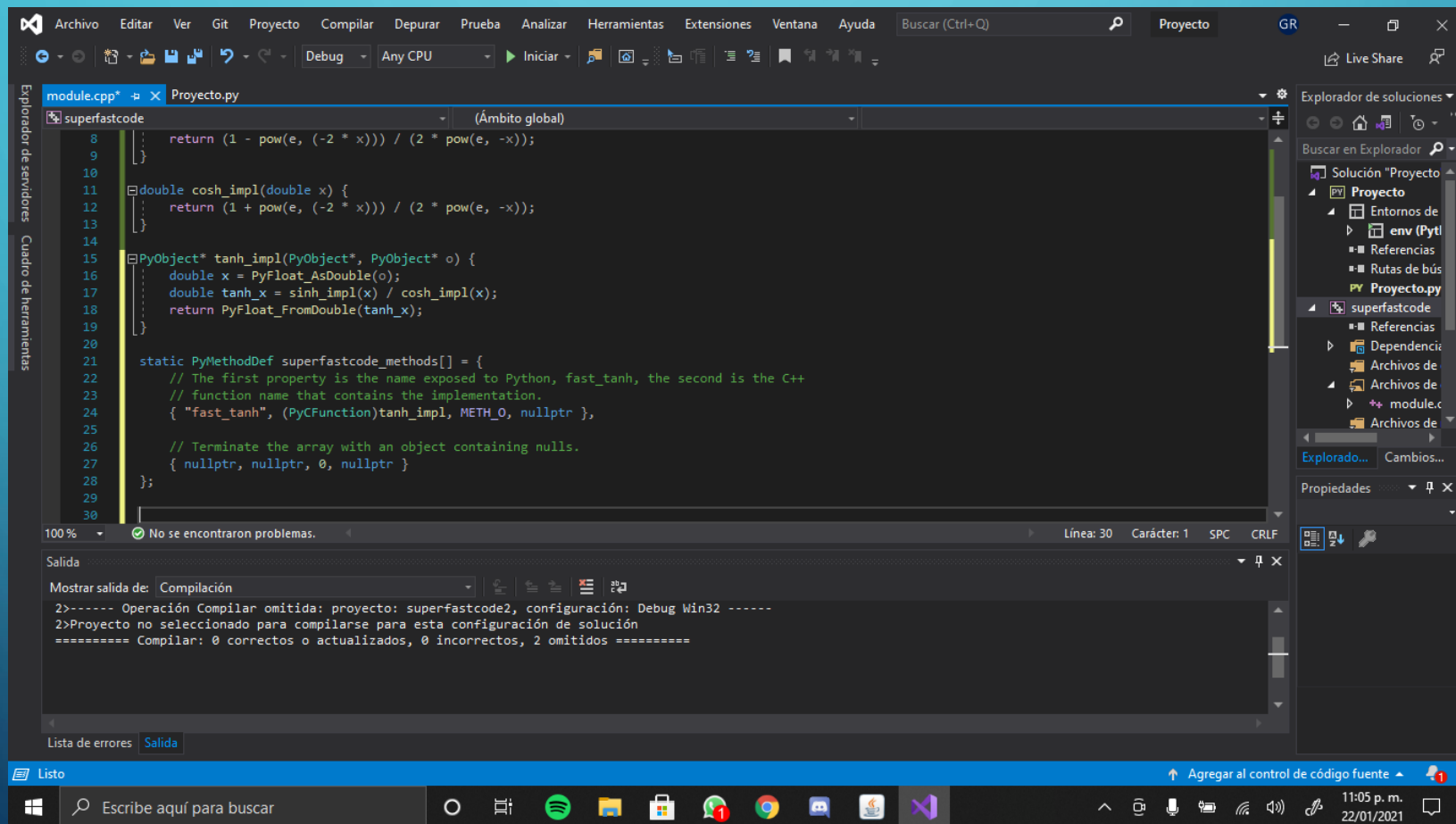
```
1 #include <Windows.h>
2 #include <cmath>
3 #include <Python.h>
4
5 const double e = 2.7182818284590452353602874713527;
6
7 double sinh_impl(double x) {
8     return (1 - pow(e, (-2 * x))) / (2 * pow(e, -x));
9 }
10
11 double cosh_impl(double x) {
12     return (1 + pow(e, (-2 * x))) / (2 * pow(e, -x));
13 }
14
15 PyObject* tanh_impl(PyObject*, PyObject* o) {
16     double x = PyFloat_AsDouble(o);
17     double tanh_x = sinh_impl(x) / cosh_impl(x);
18     return PyFloat_FromDouble(tanh_x);
19 }
```

The output window shows the compilation results:

```
2>----- Operación Compilar omitida: proyecto: superfastcode2, configuración: Debug Win32 -----
2>Proyecto no seleccionado para compilarse para esta configuración de solución
===== Compilar: 0 correctos o actualizados, 0 incorrectos, 2 omitidos =====
```

# PASO 17:

- 3- Agregue una estructura que defina la manera en que la función `tanh_impl` de C++ se muestra en Python:



```
module.cpp* x Proyecto.py
superfastcode (Ámbito global)
8      return (1 - pow(e, (-2 * x))) / (2 * pow(e, -x));
9
10
11     double cosh_impl(double x) {
12         return (1 + pow(e, (-2 * x))) / (2 * pow(e, -x));
13     }
14
15     PyObject* tanh_impl(PyObject*, PyObject* o) {
16         double x = PyFloat_AsDouble(o);
17         double tanh_x = sinh_impl(x) / cosh_impl(x);
18         return PyFloat_FromDouble(tanh_x);
19     }
20
21     static PyMethodDef superfastcode_methods[] = {
22         // The first property is the name exposed to Python, fast_tanh, the second is the C++
23         // function name that contains the implementation.
24         { "fast_tanh", (PyCFunction)tanh_impl, METH_O, nullptr },
25
26         // Terminate the array with an object containing nulls.
27         { nullptr, nullptr, 0, nullptr }
28     };
29
30
```

Salida

Mostrar salida de: Compilación

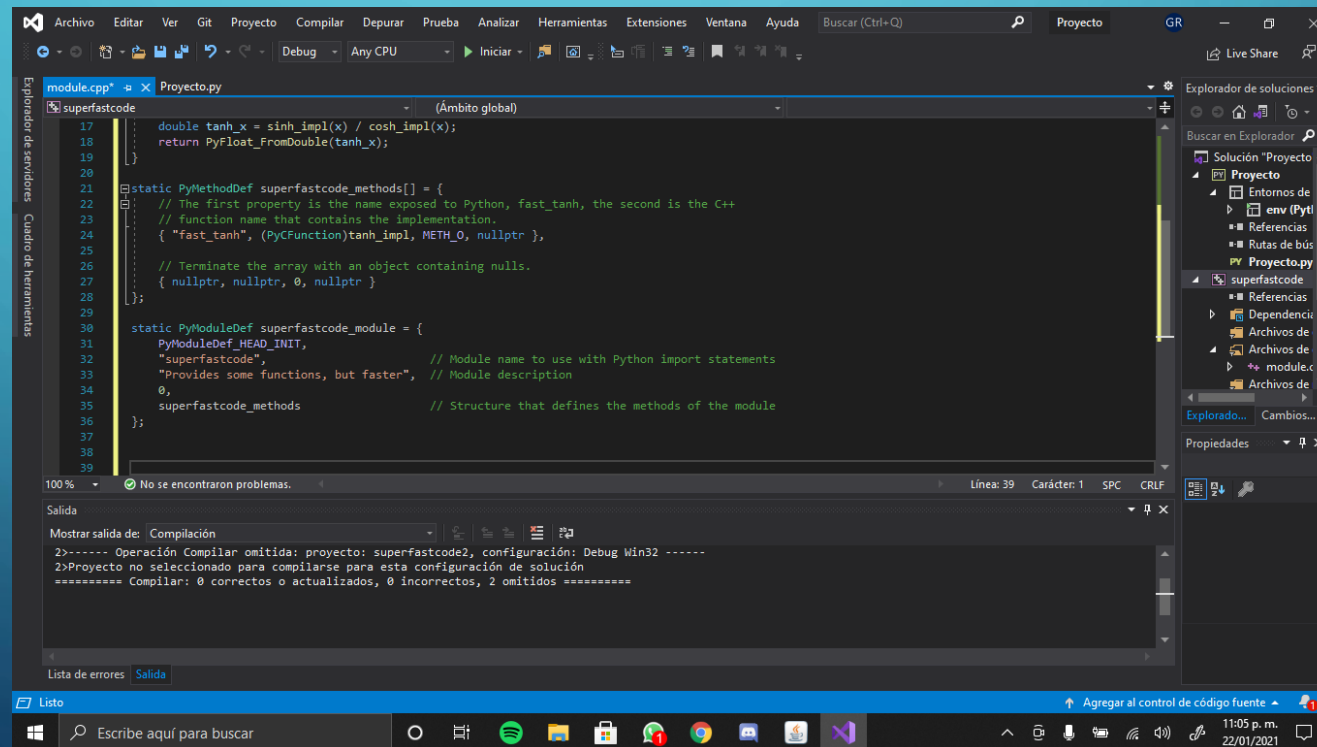
```
2>----- Operación Compilar omitida: proyecto: superfastcode2, configuración: Debug Win32 -----
2>Proyecto no seleccionado para compilarse para esta configuración de solución
===== Compilar: 0 correctos o actualizados, 0 incorrectos, 2 omitidos =====
```

Lista de errores Salida

11:05 p. m. 22/01/2021

# PASO 18:

- 4- Agregue una estructura que defina el módulo como quiere que se haga referencia a él en el código de Python, concretamente cuando se usa la instrucción `from...import`. (Hágalo coincidir con el valor de las propiedades del proyecto en Propiedades de configuración > General > Nombre de destino). En el ejemplo siguiente, el nombre de módulo "superfastcode" significa que puede usar `from superfastcode import fast_tanh` en Python, porque `fast_tanh` se define en `superfastcode_methods`. (Los nombres de archivo internos del proyecto de C++, como `module.cpp`, no tienen importancia).



# PASO 19:

- 5- Agregue un método al que Python llame cuando cargue el módulo, con el nombre `PyInit_<module-name>`, donde `<module_name>` debe coincidir exactamente con la propiedad General > Nombre de destino del proyecto de C++ (es decir, coincide con el nombre del archivo de `.pyd` compilado por el proyecto).

The screenshot shows the Visual Studio IDE with the following components:

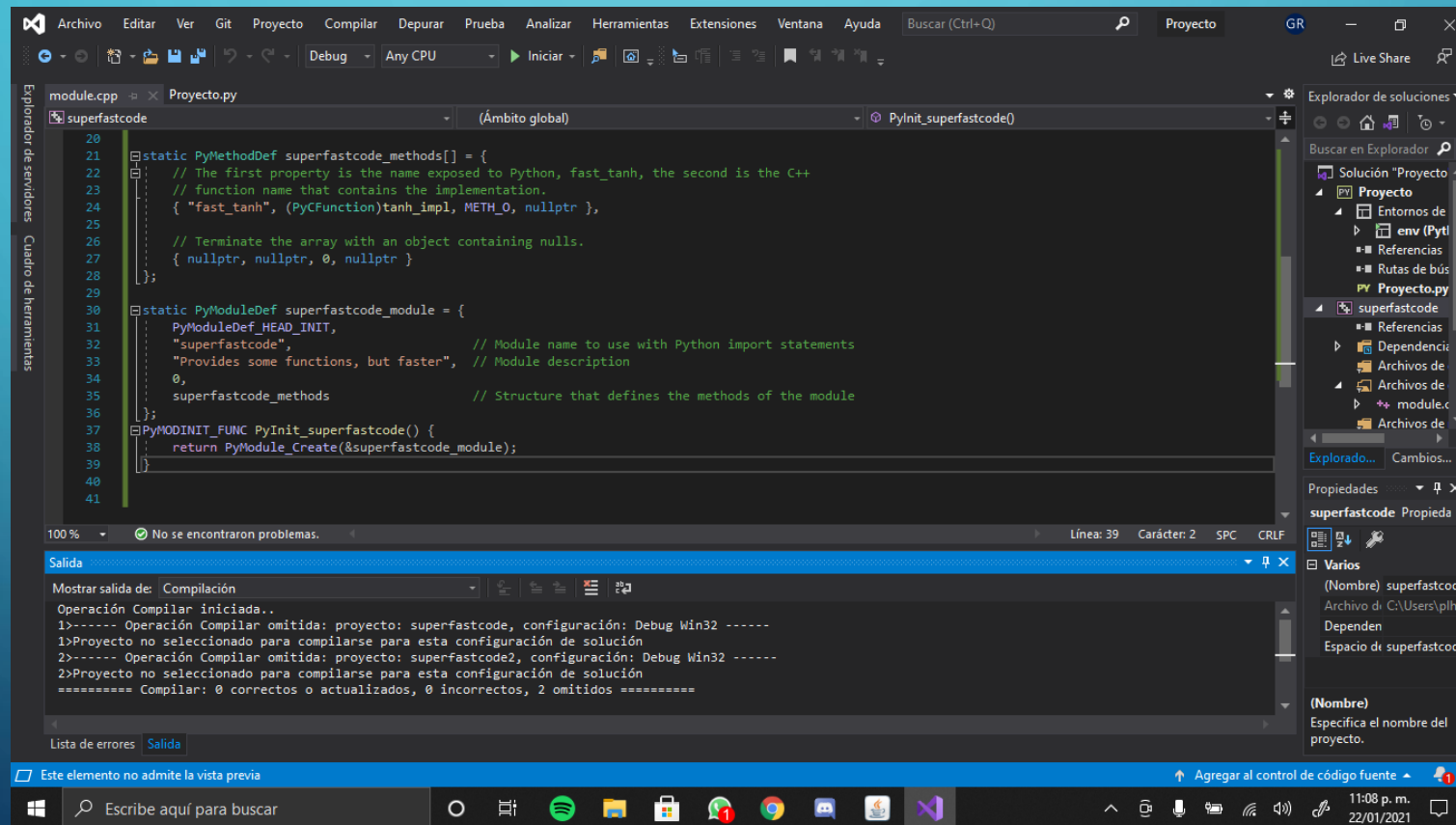
- Editor:** Displays `module.cpp` with the following code:

```
20
21
22 static PyMethodDef superfastcode_methods[] = {
23     // The first property is the name exposed to Python, fast_tanh, the second is the C++
24     // function name that contains the implementation.
25     { "fast_tanh", (PyCFunction)tanh_impl, METH_O, nullptr },
26
27     // Terminate the array with an object containing nulls.
28     { nullptr, nullptr, 0, nullptr }
29 };
30
31 static PyModuleDef superfastcode_module = {
32     PyModuleDef_HEAD_INIT,
33     "superfastcode",           // Module name to use with Python import statements
34     "Provides some functions, but faster", // Module description
35     0,
36     superfastcode_methods      // Structure that defines the methods of the module
37 };
38
39 PyMODINIT_FUNC PyInit_superfastcode() {
40     return PyModule_Create(&superfastcode_module);
41 }
```
- Output Window:** Shows the compilation output:

```
2>----- Operación Compilar omitida: proyecto: superfastcode2, configuración: Debug Win32 -----
2>Proyecto no seleccionado para compilarse para esta configuración de solución
===== Compilar: 0 correctos o actualizados, 0 incorrectos, 2 omitidos =====
```
- Explorer:** Shows the project structure with folders for `env (Pyt)`, `Referencias`, `Rutas de búsqueda`, and `Proyecto.py`.
- Taskbar:** Shows the Windows taskbar with the search bar and various application icons.

# PASO 20:

- 6-Establezca la configuración de destino en Lanzamiento y compile el proyecto de C++ de nuevo para comprobar el código.

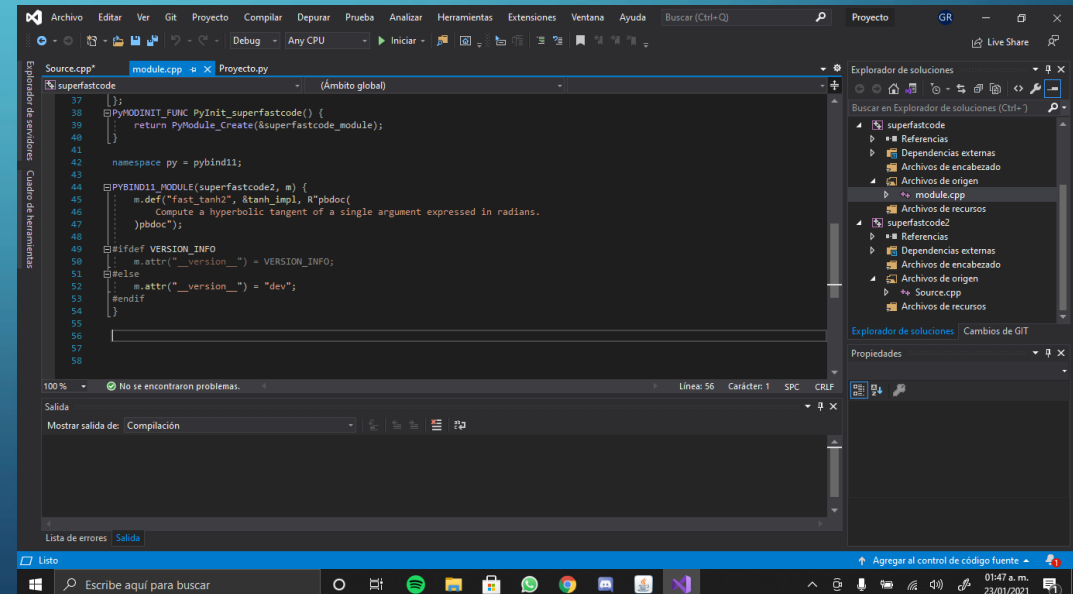
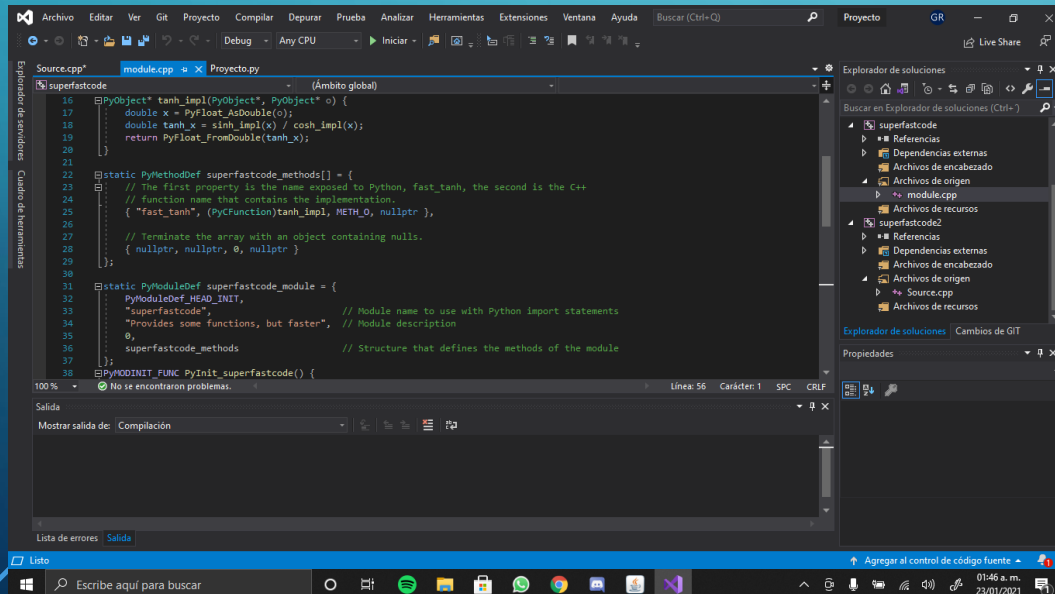




# PASO 21:

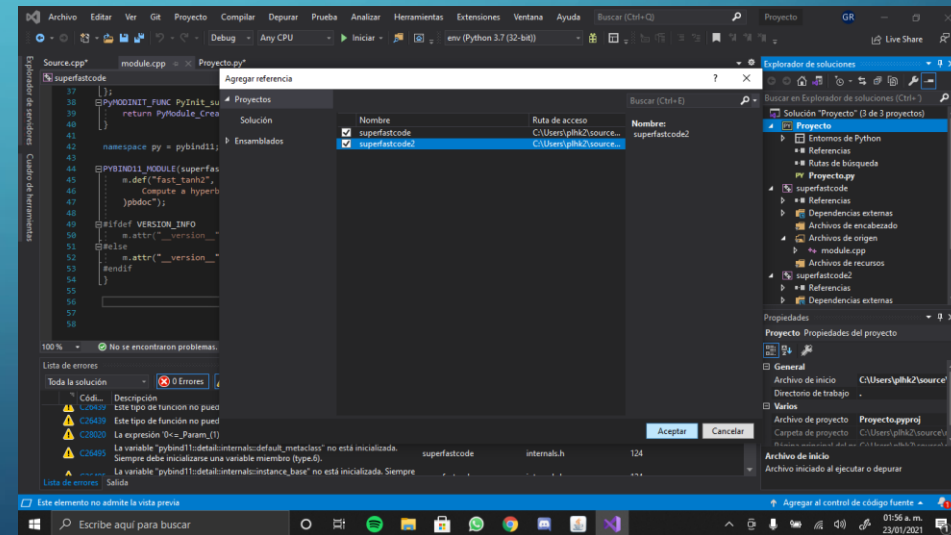
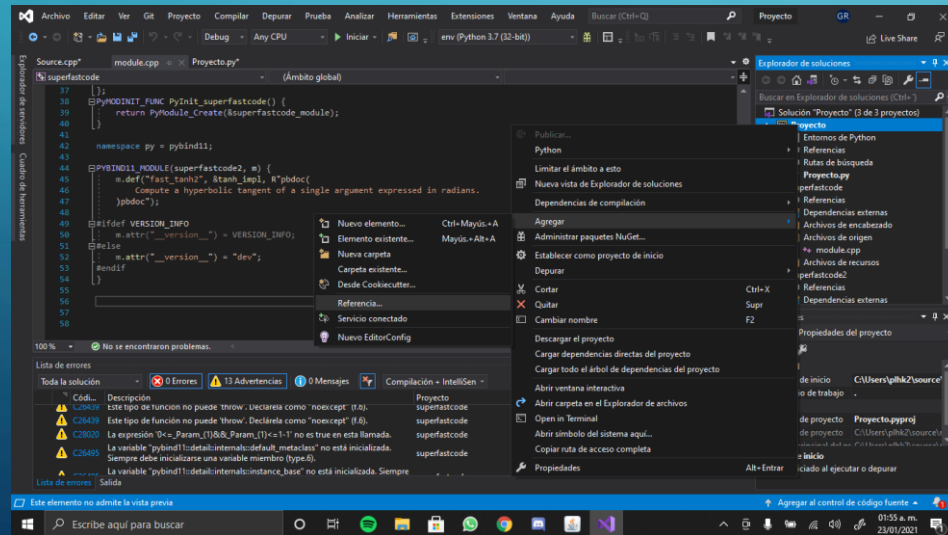
- **Pybind11**

- 1-Instale PyBind11 mediante pip: `pip install pybind11` o `py -m pip install pybind11`
- 2-En la parte superior de module.cpp, incluya pybind11.h:
- 3-En la parte inferior de module.cpp, use la macro `PYBIND11_MODULE` para definir el punto de entrada a la función de C++:
- 4-Establezca la configuración de destino en Lanzamiento y compile el proyecto de C++ para comprobar el código. Si se producen errores, vea la sección Solución de problemas a continuación.



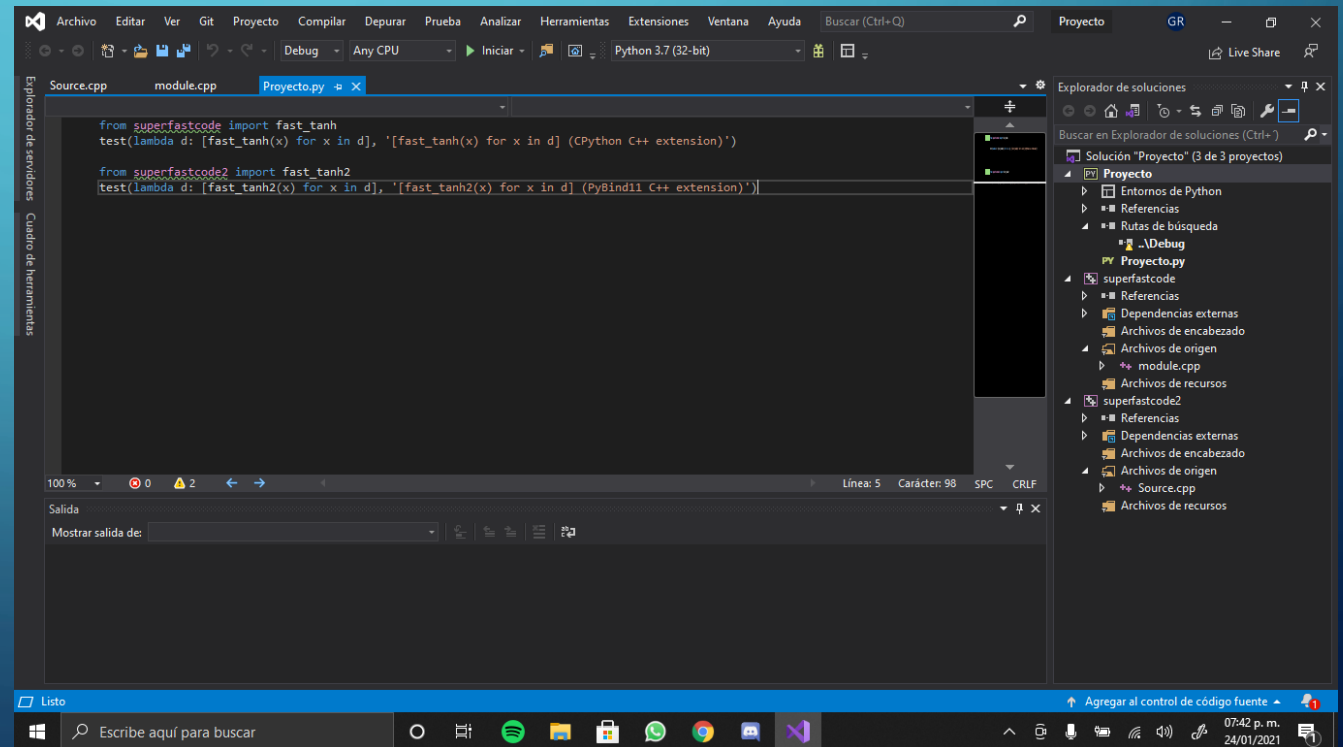
# PASO 22:

- 1- Probar el código y comparar los resultados Ahora que tiene los archivos DLL estructurados como extensiones de Python, puede hacerles referencia desde el proyecto de Python, importar los módulos y usar sus métodos.
- 2- Poner el archivo DLL a disposición de Python Hay dos maneras de hacer que el archivo DLL esté disponible para Python. El primer método funciona si el proyecto de Python y el de C++ se encuentran en la misma solución.
- Vaya al Explorador de soluciones, haga clic con el botón derecho en el nodo Referencias del proyecto de Python y, después, seleccione Agregar referencia. En el cuadro de diálogo que aparecerá, seleccione la pestaña Proyectos, los proyectos superfastcode y superfastcode2 y Aceptar.



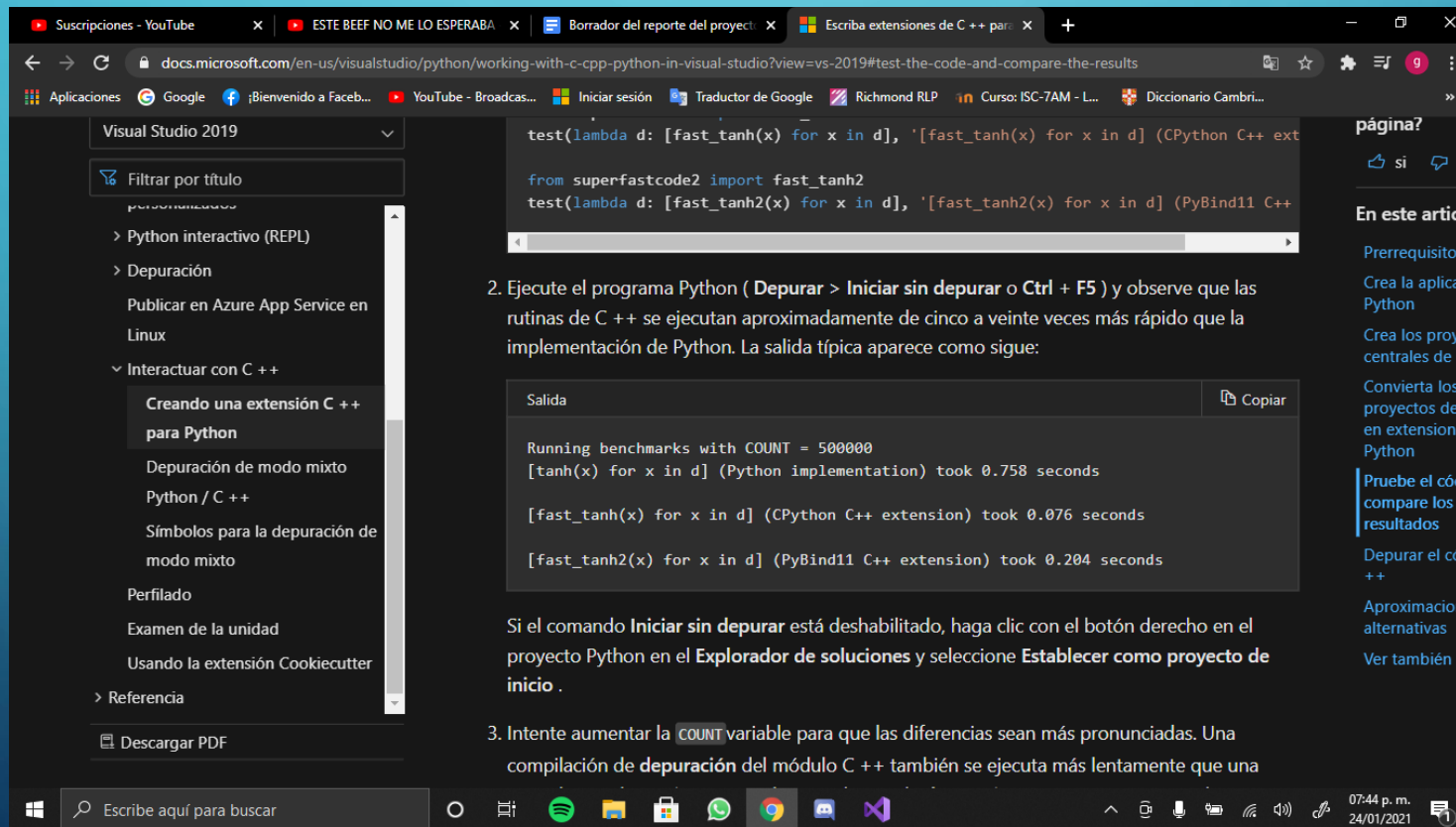
# PASO 23:

- Llamar al archivo DLL desde Python Una vez que el archivo DLL está disponible para Python como se ha descrito en la sección anterior, ahora puede llamar a las funciones `superfastcode.fast_tanh` y `superfastcode2.fast_tanh2` del código de Python y comparar su rendimiento con la implementación de Python:
- 1- Agregue las líneas siguientes en el archivo `.py` para llamar a los métodos exportados desde los archivos DLL y mostrar los resultados.



# PASO 24:

- Ejecute el programa Python ( Depurar > Iniciar sin depurar o Ctrl + F5 ) y observe que las rutinas de C ++ se ejecutan aproximadamente de cinco a veinte veces más rápido que la implementación de Python. La salida típica aparece como sigue:



The screenshot shows the Visual Studio 2019 interface. On the left, the 'Explorador de soluciones' (Solution Explorer) shows a project named 'Creando una extensión C++ para Python'. The main editor displays a C++ file with the following code:

```
test(lambda d: [fast_tanh(x) for x in d], '[fast_tanh(x) for x in d] (CPython C++ ext
from superfastcode2 import fast_tanh2
test(lambda d: [fast_tanh2(x) for x in d], '[fast_tanh2(x) for x in d] (PyBind11 C++
```

Below the code, the 'Salida' (Output) window shows the results of the benchmark:

```
Running benchmarks with COUNT = 500000
[tanh(x) for x in d] (Python implementation) took 0.758 seconds

[fast_tanh(x) for x in d] (CPython C++ extension) took 0.076 seconds

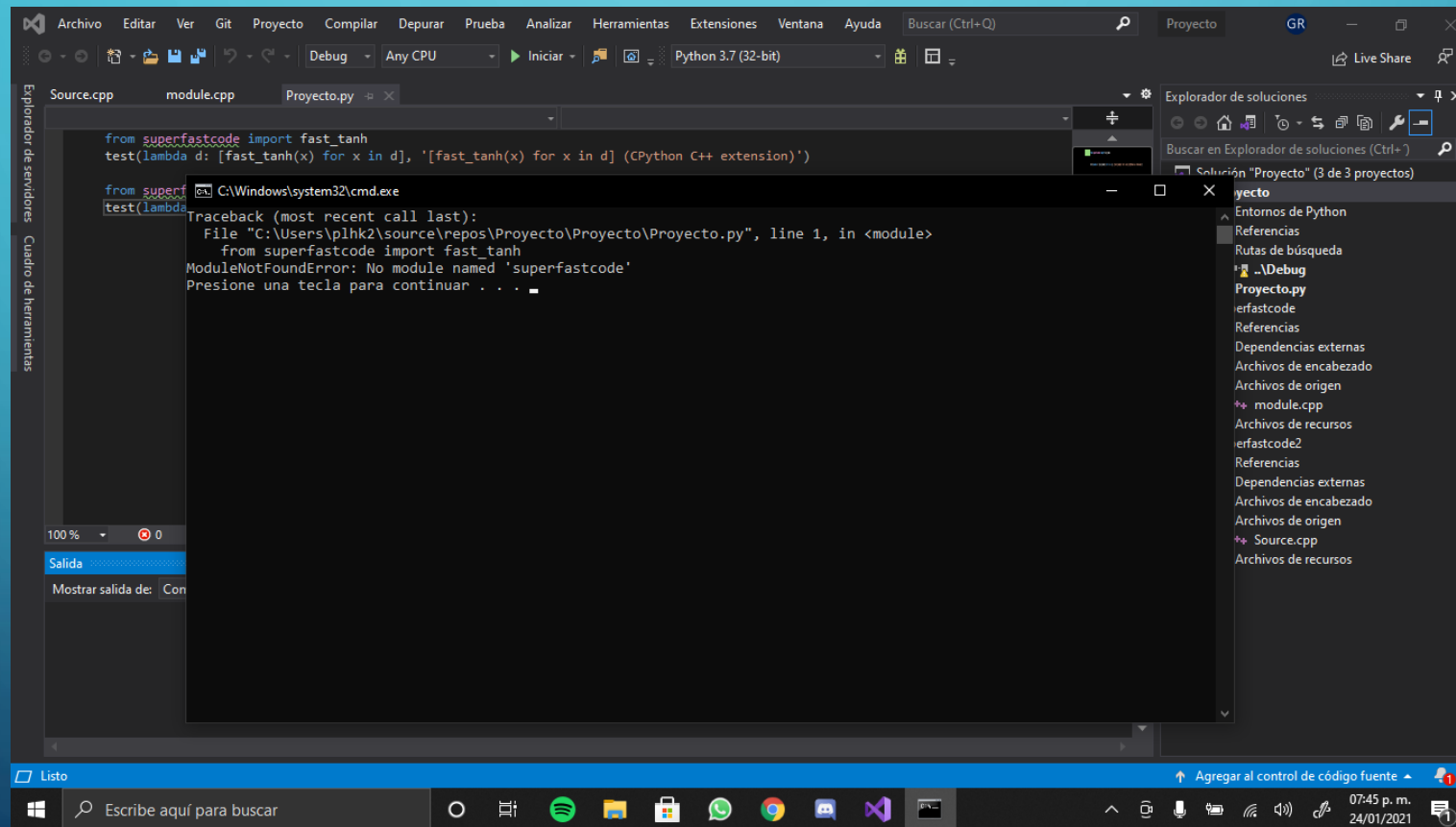
[fast_tanh2(x) for x in d] (PyBind11 C++ extension) took 0.204 seconds
```

Text below the output window states: 'Si el comando **Iniciar sin depurar** está deshabilitado, haga clic con el botón derecho en el proyecto Python en el **Explorador de soluciones** y seleccione **Establecer como proyecto de inicio** .

Below this, the next step is listed: '3. Intente aumentar la **COUNT** variable para que las diferencias sean más pronunciadas. Una compilación de **depuración** del módulo C++ también se ejecuta más lentamente que una

# PASO 25:

- En nuestro caso nos salió un resultado diferente ya que nos muestra la siguiente ventana





# ANOTACIONES:

- Al revisar las soluciones encontramos que en la página la solución mencionada requiere de Visual Studio de versión anterior puesto que la versión que tenemos es la más reciente(2019) y el post del que nos basamos para comprobar dicha conexión no está actualizado para esta versión no nos fue posible demostrar la conexión entre dos lenguajes de programación.
- Lo que notamos en el desarrollo de este proyecto fue que una de las ventajas que tiene esta nueva librería que nos asignó el profesor es la compatibilidad con otro lenguaje de programación ya que cuenta con nuevas librerías dentro de ella que se lo permiten y son más rápidas.