

Instituto Tecnológico de iztapalapa

Computer Systems Engineering

Proposal for the development of the project entitled:
"libc ++" C ++ Standard Library "

Presents:

Cabrera Ramirez Gerardo
Morales Carrillo Gerardo
Santana Gonzalez Jesus Salvador

Control Number:

171080187

171080120

171080127

Internal Advisor:

Abiel Tomas Parra Hernandez



General project summary:

The same concept of a standard C library, but specific to C ++. The C ++ standard library is a set of C ++ class templates that provides common programming data structures and functions such as lists, stacks, arrays, algorithms, iterators, and any other C ++ component you can think of. The C ++ standard library It incorporates the C standard library as well and is specified in the C ++ standard.

Since we must carry out a project on the subject that the teacher has assigned us, in our case it was: "libc ++" C ++ Standard Library ", we will have to investigate about the platform where it will be programmed and different characteristics that can be implemented (It depends on what the project will be in the future.)

In this project we will talk about the characteristics and functions of this library since it has many things that can be used, such as being able to implement it in other programming languages with which they are compatible. In this case We are going to corroborate the implementation of "libc ++" C ++ Standard Library with the python programming language, by creating a project in python and trying to affirm that the code can really be compiled with the c ++ library, all this being documented Step by Step.



PHOT
0

COMPANY: Los musqueteros TELEPHONE: 5567893456 FAX:

ADDRESS:

STUDENT: Cabrera Ramirez Gerardo, Morales Carrillo Gerardo, Santana Gonzalez Jesus

CAREER: Computer systems engineering

PROJECT NAME: "libc ++" C ++ Standard Library

INTERNAL ADVISOR: Hernandez Parra Abiel

EXTERNAL ADVISOR: Instituto tecnologico de iztapalapa

START DATE: 11/26/2020 END DATE: 01/24/2021

PROJECT OBJECTIVE: Creation and research of a project related to C ++ libraries

Activity		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Collection of information	P																
Selection of information that will be put in the document	P																
Search for the program to be carried out	P																
Select the program to be carried out	P																
Implementation of the program	P																
Check of the program	P																
Documentation of the	P																



implementation																		
Delivery	P																	

Risks	Solution
That a member is absent during the project (due to illnesses or events beyond our control)	Restrict the team's way of working to cover the pending issues that the absent member may have.
That we give access without realizing anyone.	Access restrictions only for users of the team or who are working on the project.
Not having a backup in case of an emergency (Deletion of data for example).	Always make backups
Not deliver the project on time and correctly (With the requirements that are requested).	Make an organization chart to follow to organize the activities and have a structure in the times so as not to have any mishap.
Claim from a user for copyright for the use of their code.	Contact the user to request permission to use their code.

Overview

libc++ is a new implementation of the C++ standard library for C++ 11 and higher.

Functions and objectives

Correction according to the definition of the C++ 11 standard.

Fast execution.

Minimal memory usage.

Fast compilation time.

Extensive unit testing.

Design and implementation.

Extensive unit testing

Internal linker model can be dumped / read in text format Additional link functions can be connected "through" specific factor CPU and operating system specific code

Per [clang libc++ page](#). (s / e) says:

From years of experience (including implementing the standard library before), and fundamental changes to how they are implemented. For example, it is generally accepted that building std::string using "short string optimization" instead of using Copy On Write (COW) is a superior approach for multi-core machines (particularly in C++ 11, which has rvalue references) Breaking ABI's compatibility with older versions of the library was determined to be critical to achieving the performance goals of libc++. Recovered from. <https://libcxx.llvm.org/>

The main line libstdc++ changed to GPL3, which is a license that libc++ developers cannot use. libstdc++ 4.2 (the latest version of GPL2) can be independently extended to support C++ 11, but this will be a branch of the codebase (for projects,

it is generally considered worse than starting new code independently). Another problem with the library is that it is integrated with G ++ development and is often related to the corresponding version of

Justification:

The purpose of this project is to do the following: "Using the Python interpreter in Visual Studio We will verify that it can be implemented in C ++ and that Python and C ++ implementations can actually be made "

We have chosen the **traditional methodology** because it is the one that best suits since our project is research and in the end the techniques found will be implemented to move on to the trial and error stage

Chosen problem:

Creating a C ++ extension for Python

Theoretical framework:

In addition to the C and C ++ compilers, certain files are also included, generally called libraries. The library contains the object code for many programs that allow you to perform common operations such as reading the keyboard, writing on the screen, processing numbers, performing mathematical functions, and so on.

Here we have to develop the theoretical framework of our project, citing certain writings that we found interesting during our research:

According to the page <https://bcain-llvm.readthedocs.io/> (Unspecified):

- "The main line libstdc ++ has been changed to GPL3, which is a license that libc ++ developers cannot use. libstdc ++ 4.2 (the latest version of GPL2) can be independently extended to support C ++ 11, but this will be a branch of the codebase (for projects this is generally considered worse than starting new code independently) . Another problem with libstdc ++ is that it is tightly integrated with G ++ development, and is often closely related to the corresponding version of G ++. "
- Also citing that: "The STLport and Apache libstdcxx libraries are two other popular candidates, but both lack C ++ 11 support. Our experience (and the experience of the libstdc ++ developers) is that adding C ++ support ++ 11 (especially rvalue references and move-only types) almost requires changes in every class and function, which is actually rewritten. Faced with rewriting,

we decided to start from scratch and evaluate each design decision based on experience based on first principles. “

Retrieved from

https://docs.google.com/document/d/1crku6_1vpmVRZIKbo00sk5JMsmKQuROOj7EMpCCgie4/edit, 03/12/2020

The GCC 11 (G ++) C ++ front-end will probably support using the LLVM standard libc ++ library. Recently, a question was asked on the GCC mailing list about the ability to do `-stdlib = libc ++` to use the LLVM C ++ standard library in conjunction with the GCC C ++ compiler.

Those wondering about LLVM's current libc ++ capabilities can find it via libcxx.llvm.org.

Development:

The microsoft page tells us that the requirements to perform the check are:

- Visual Studio 2017 or later versions with the Desktop Development with C ++ and Python Development workloads installed with default options.
- In the Python Development workload, also select the box to the right of Python Native Development Tools. This option sets most of the settings described in this article. (This option also includes the C ++ workload automatically.)

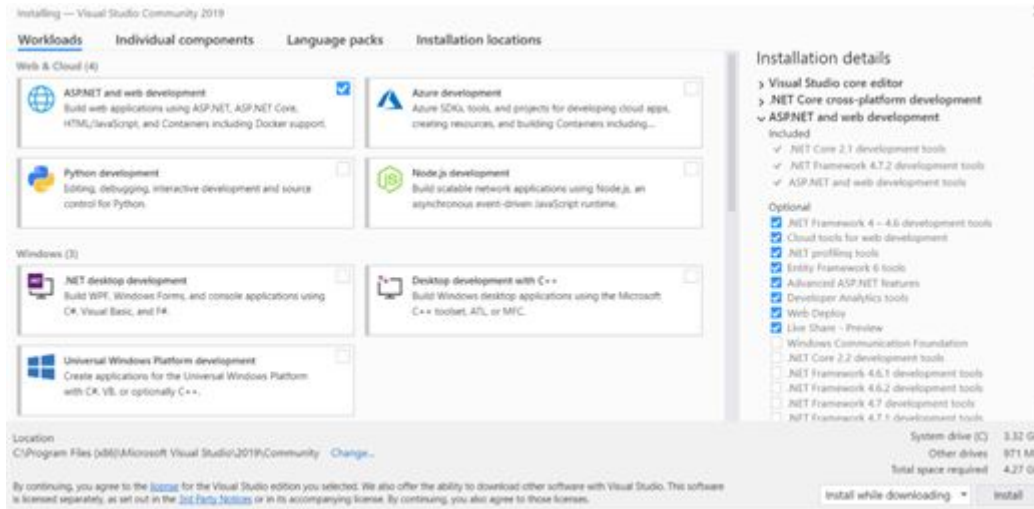
1.- We will create and implement c ++ for phyton, we will review the requirements that are needed:

* Visual studio 2017 (or later) with the workloads (Development for the c ++ desktop and development of phyton)

* First we will install visual studio from first instance, which will be obtained from the Microsoft page

Step 1:

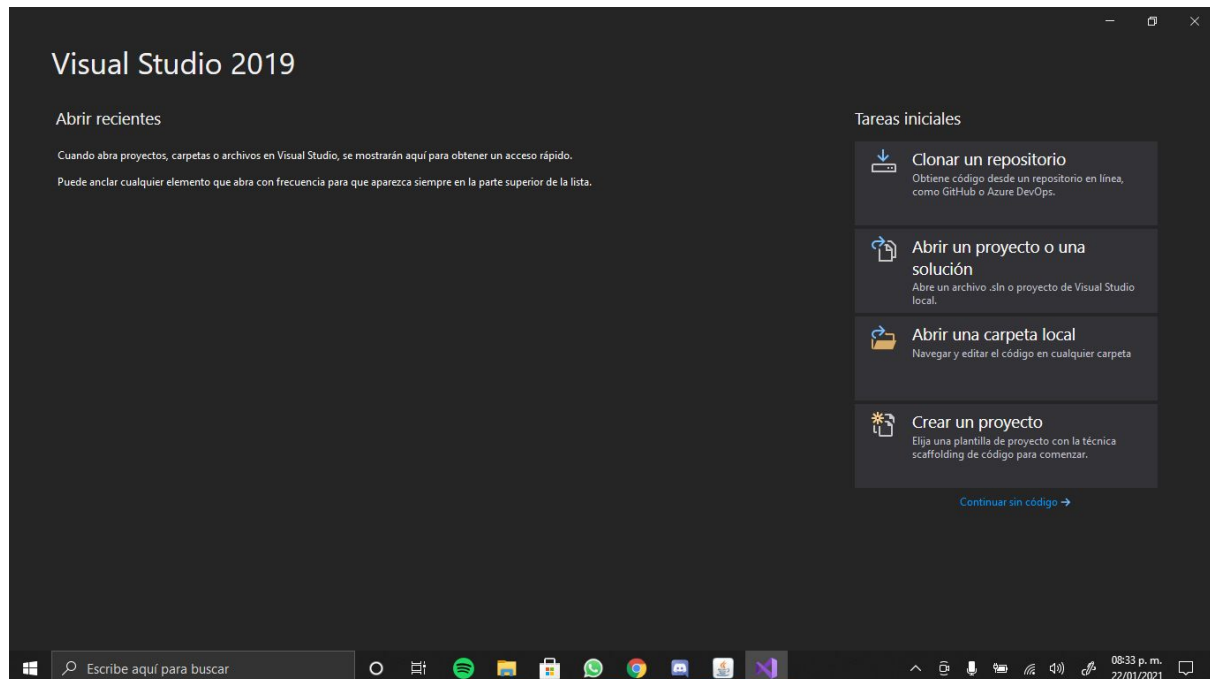
- Check the system requirements for compatibility with the program.
- Once visual studio has been downloaded, we will double click on the .exe file
- Next, we will see a screen in which we will give the option “yes” to execute the program, terms and conditions: accept, a new window will appear where we will choose the workloads we need, in this case, they will be the following:
 - Development for the desktop c ++ and development of phyton



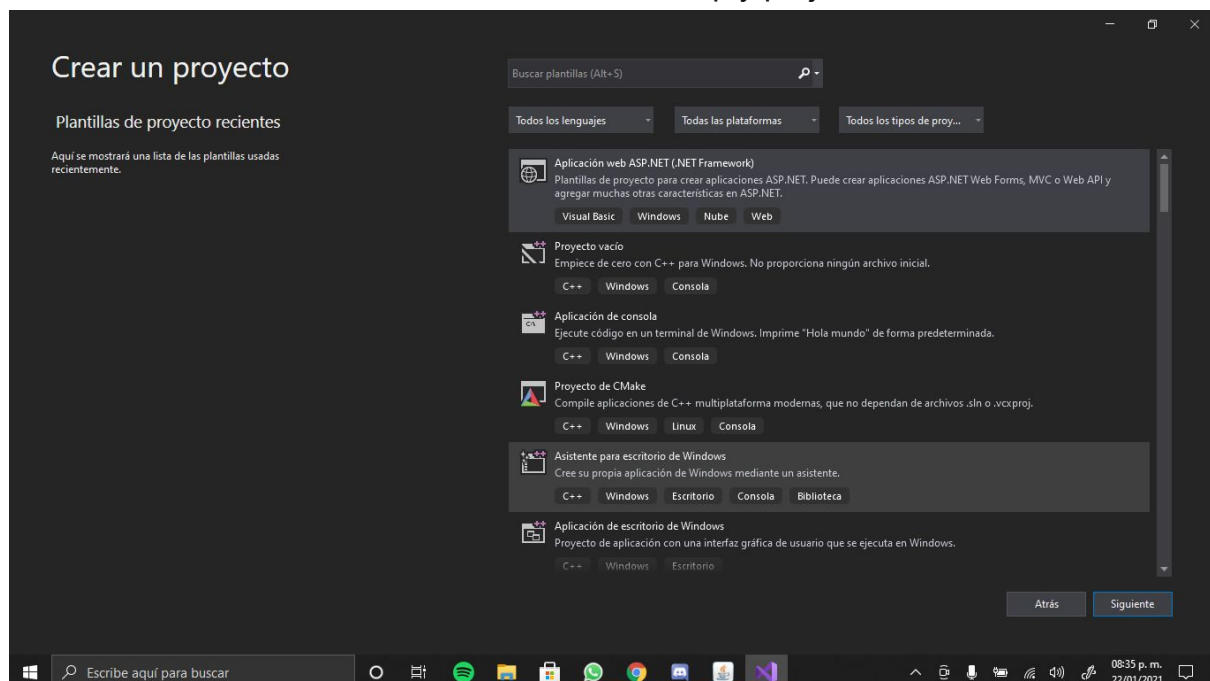
This is to obtain the compatibility with c and c ++, when the installation of visual studio is complete, we will click on the "Start" button.

We will be in the start window, we will choose "Create a new project" We will

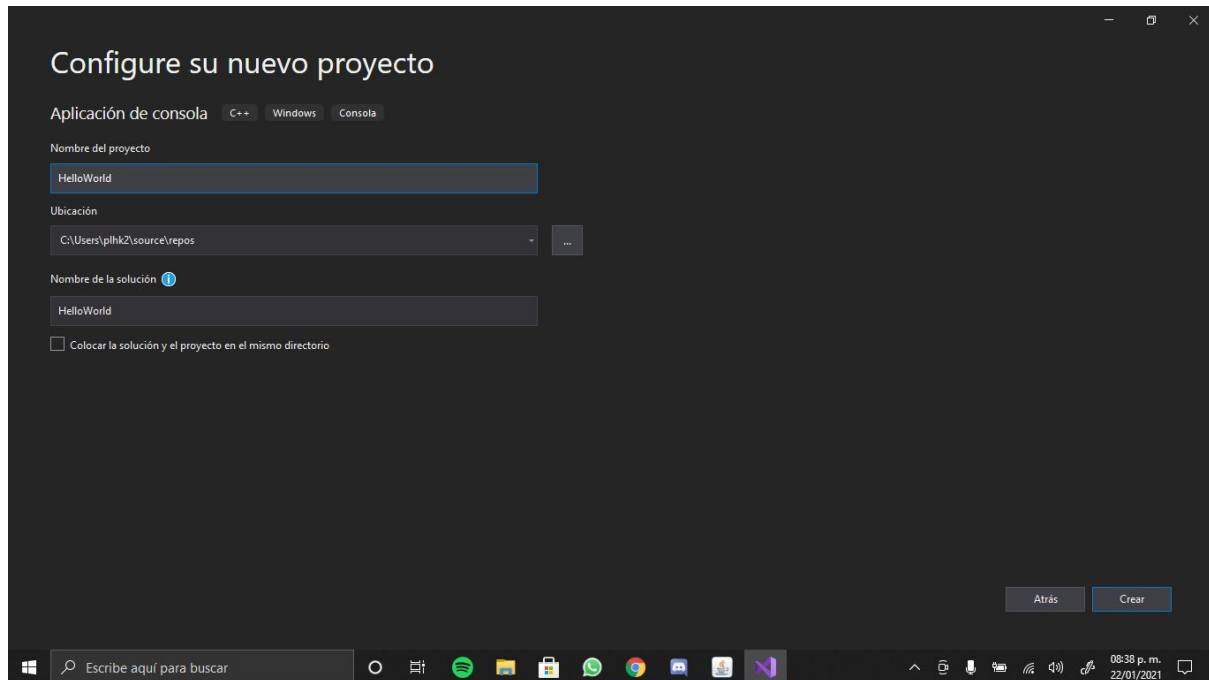




access the search box, where we will choose "Empty project"

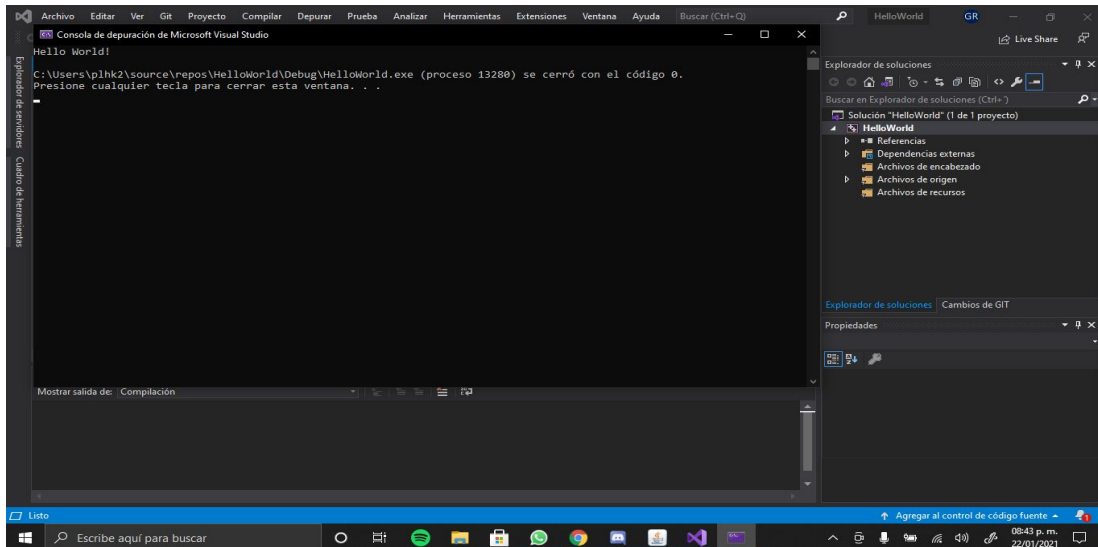


We will next, now, we will select "console app", a dialog box will open, "Configure the new project ", We will make the well-known " Hello world "

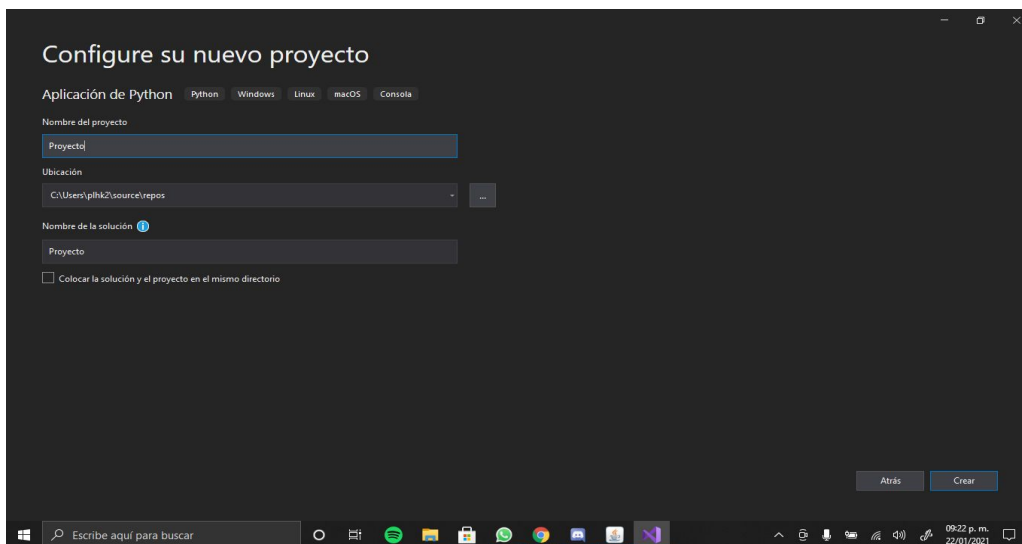


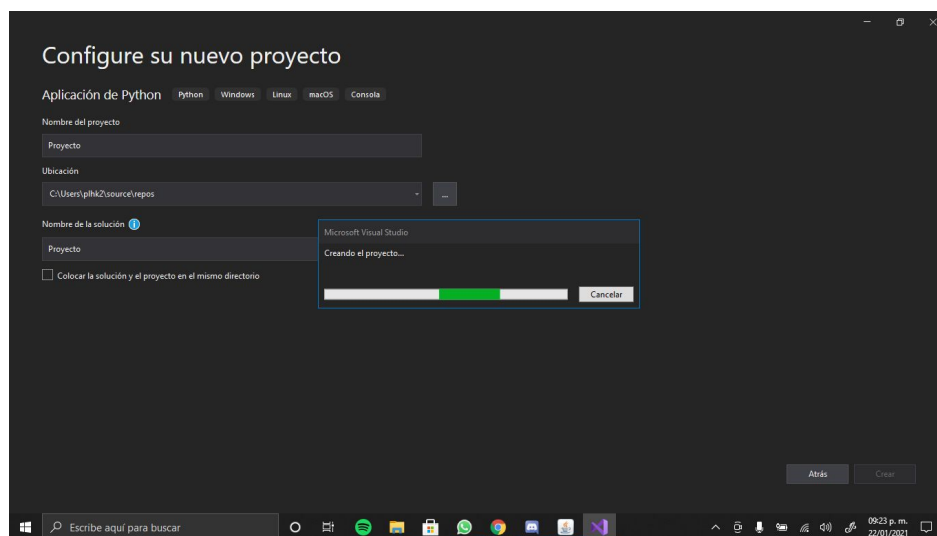
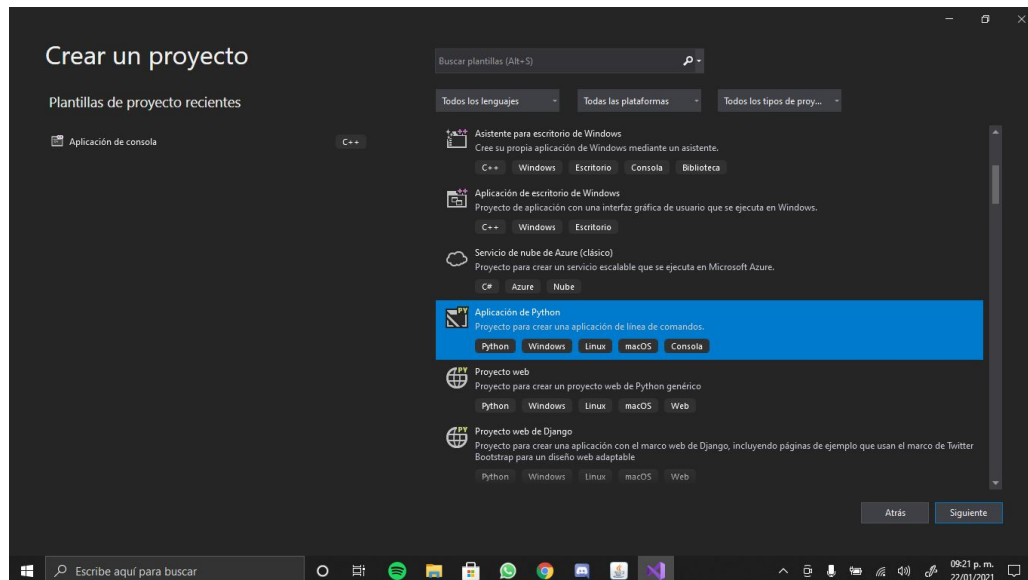
Weon" Create ":

will clickWe will click on the" Compile "button," Compile solution ", and to execute the code, we will click" Debug "," Start without debugging ", a new window:

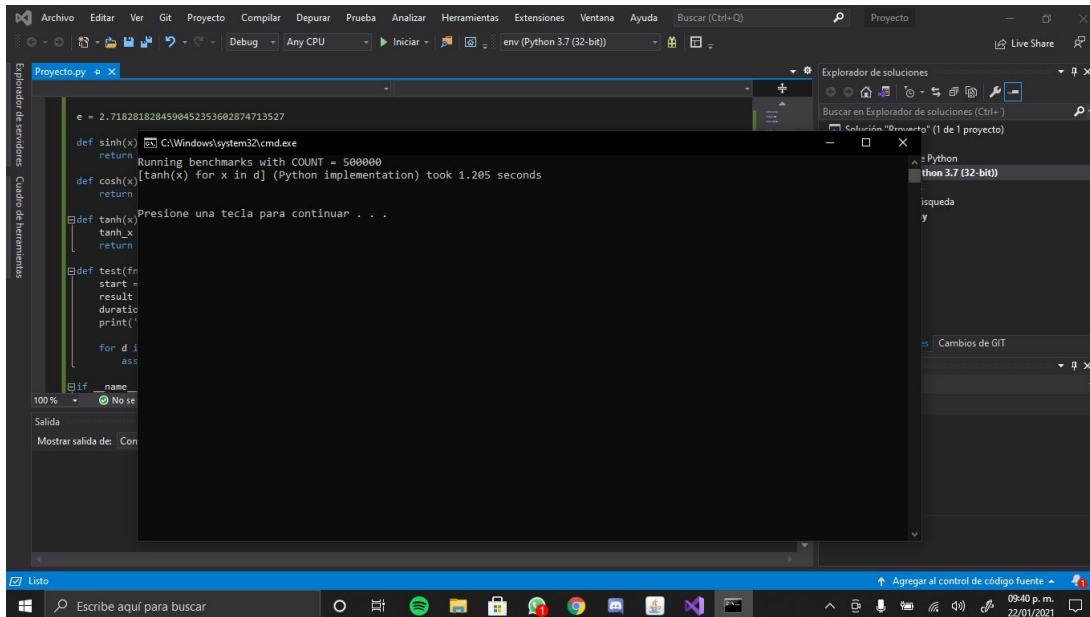


After installing Visual Studio and creating a C ++ project, compiling it and debugging it, we will begin with creating the C ++ extension for Python. We will create a Python project in Visual Studio, we select the Python application template and we will give it next, we will give it a name and create

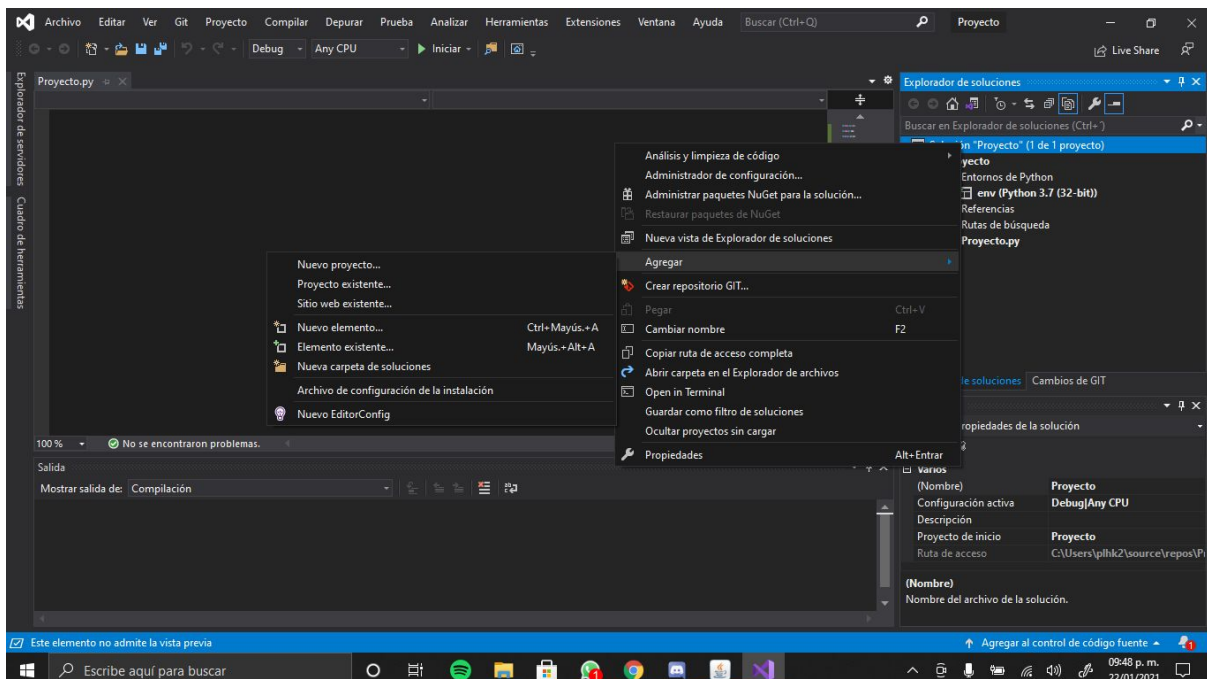




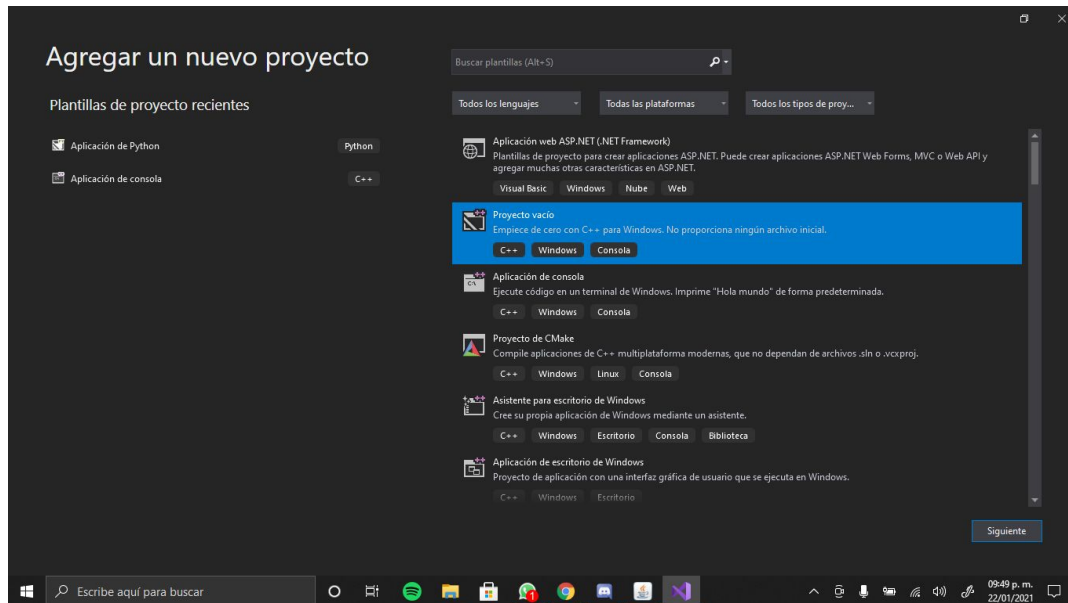
On the page they recommend using the 32-bit Python interpreter. In the .py file we will paste the code provided by the page which is a comparative test of the calculation of a hyperbolic tangent, after that we will execute the program by debugging and then starting without debugging.



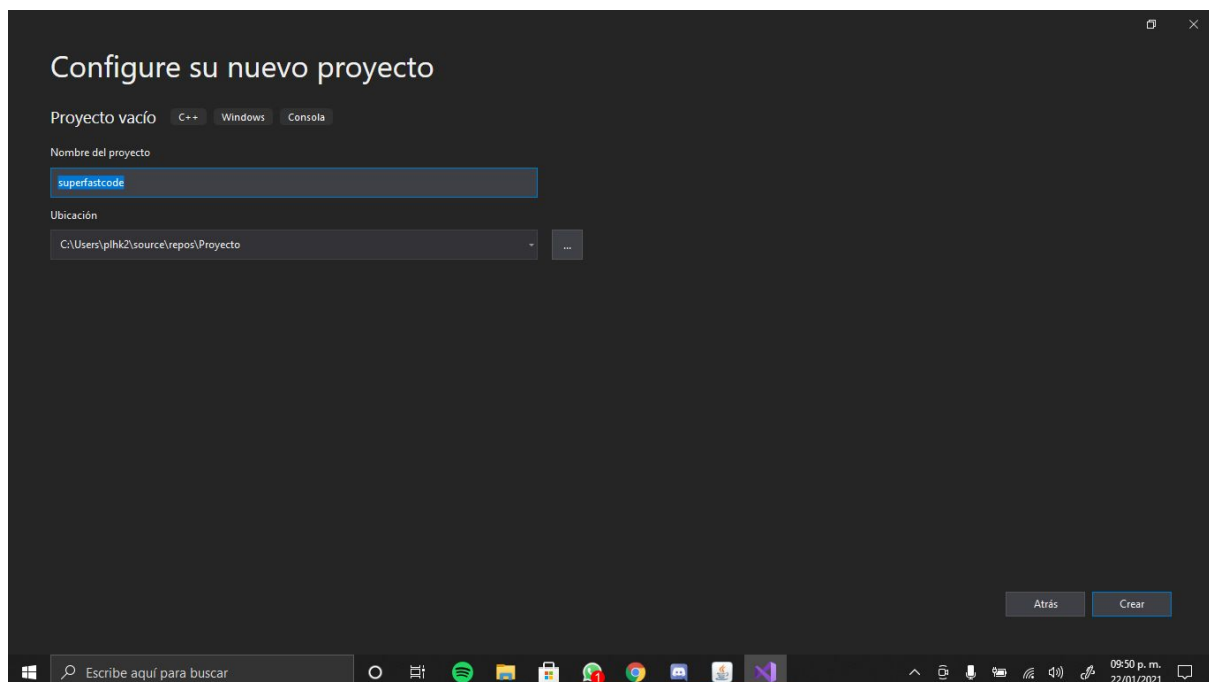
We will follow the instructions given to us to create two identical C ++ projects called "superfastcode" and "superfastcode2". After that we are going to right click in the solution explorer and select add



And then in new project, we will search for C ++ and select empty project

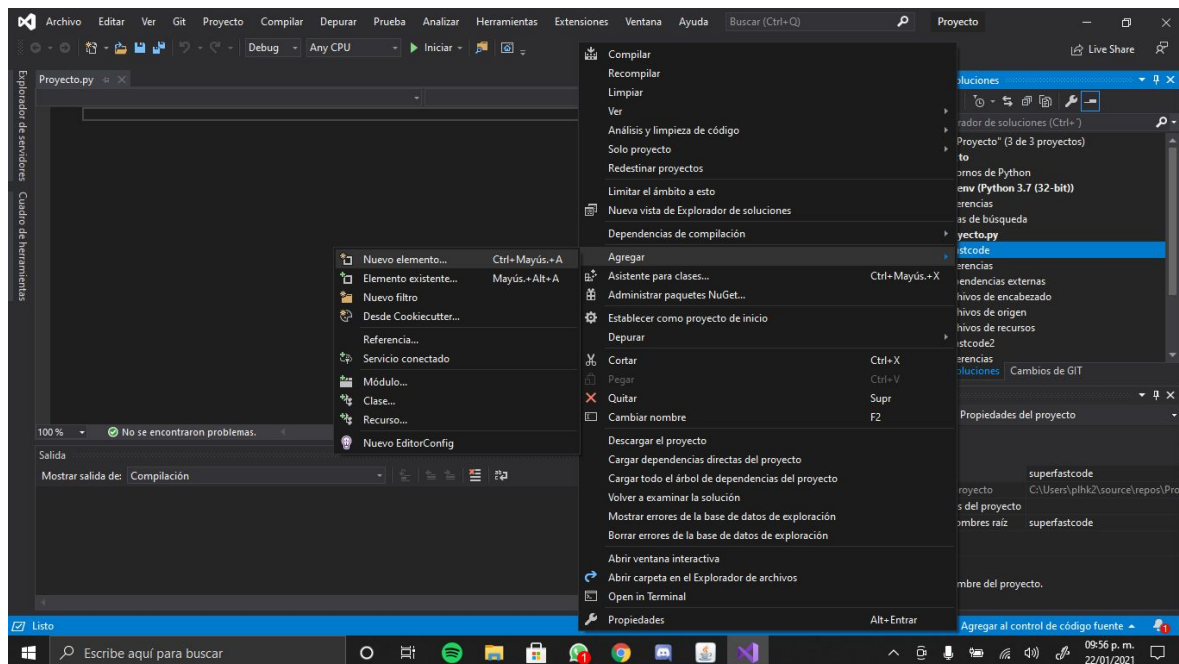


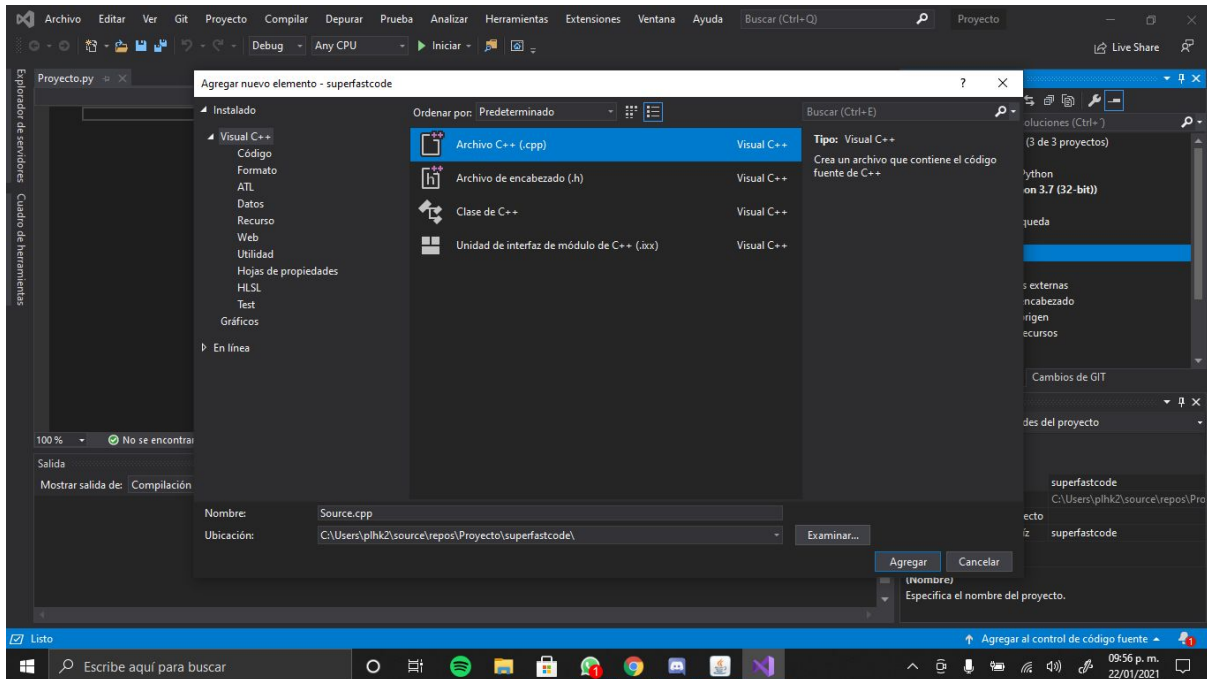
And we will give it the name of superfastcode



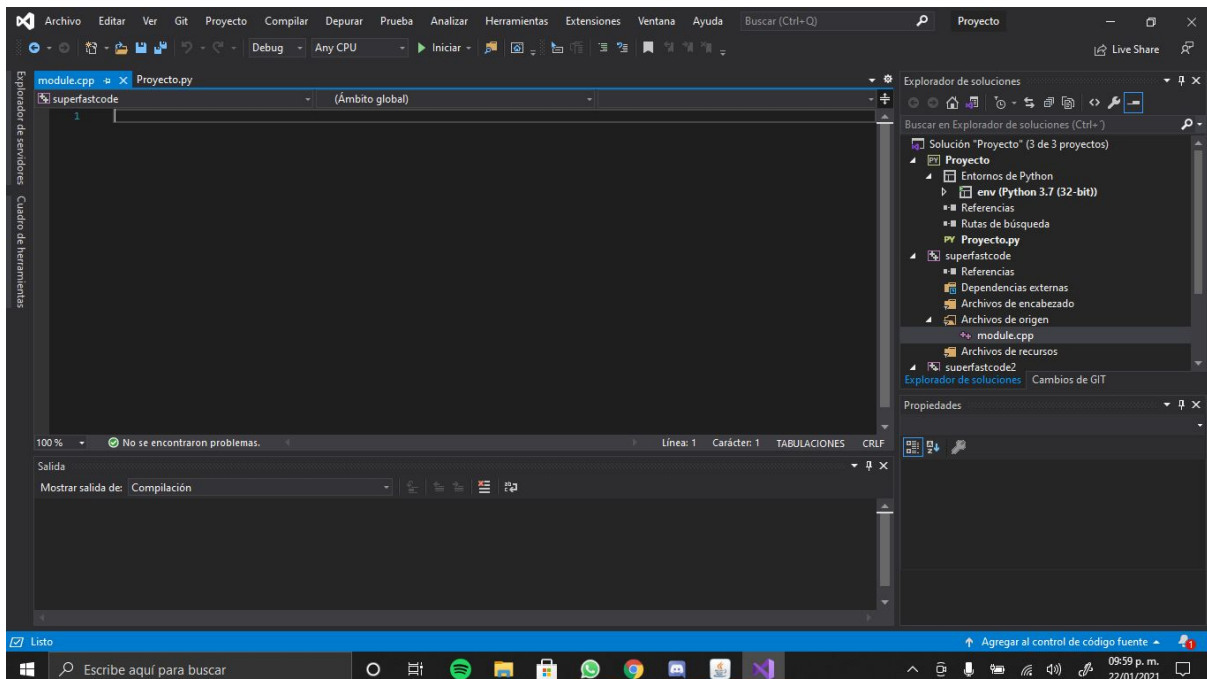
And we will make another project with the name superfastcode2.

We will create a C ++ file, for this we right click on the source code file and add new element and add a C ++ file in .cpp We will

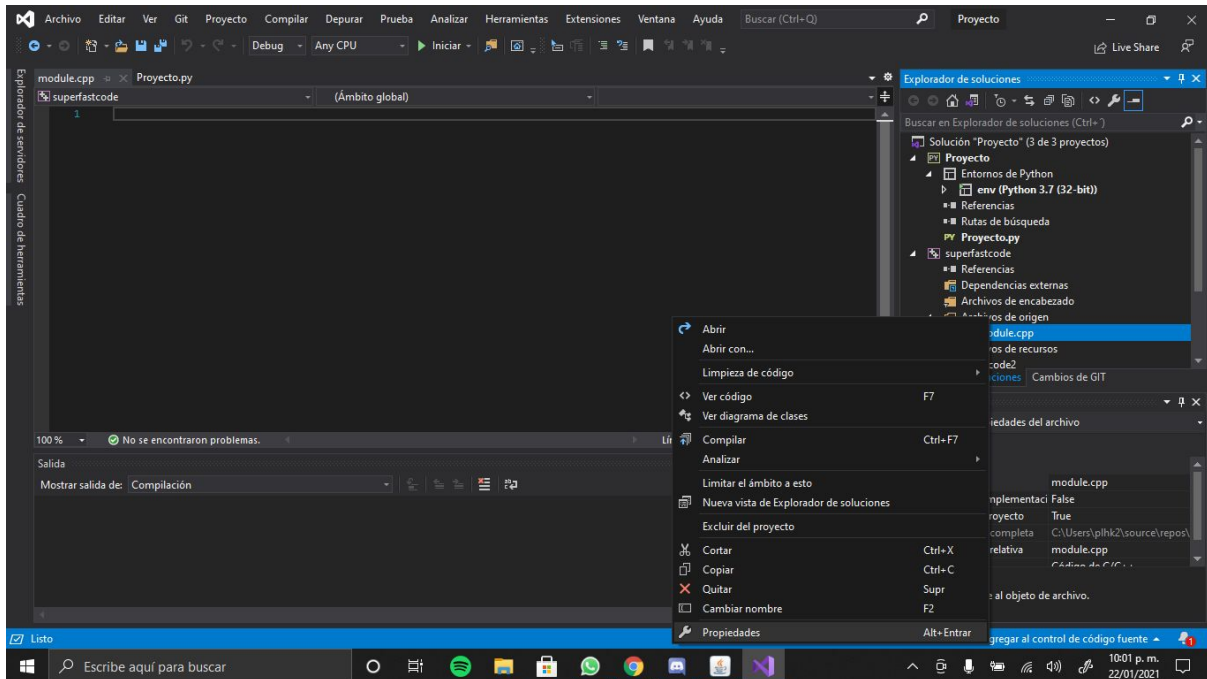




change the name to “module.cpp”

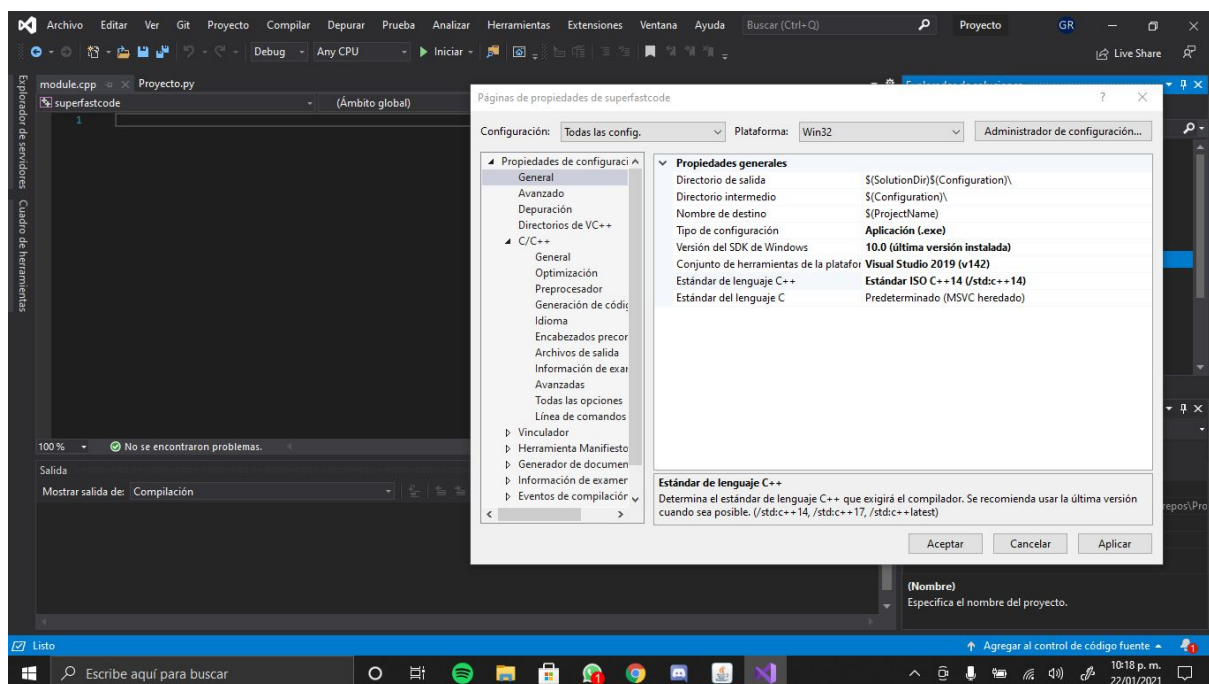
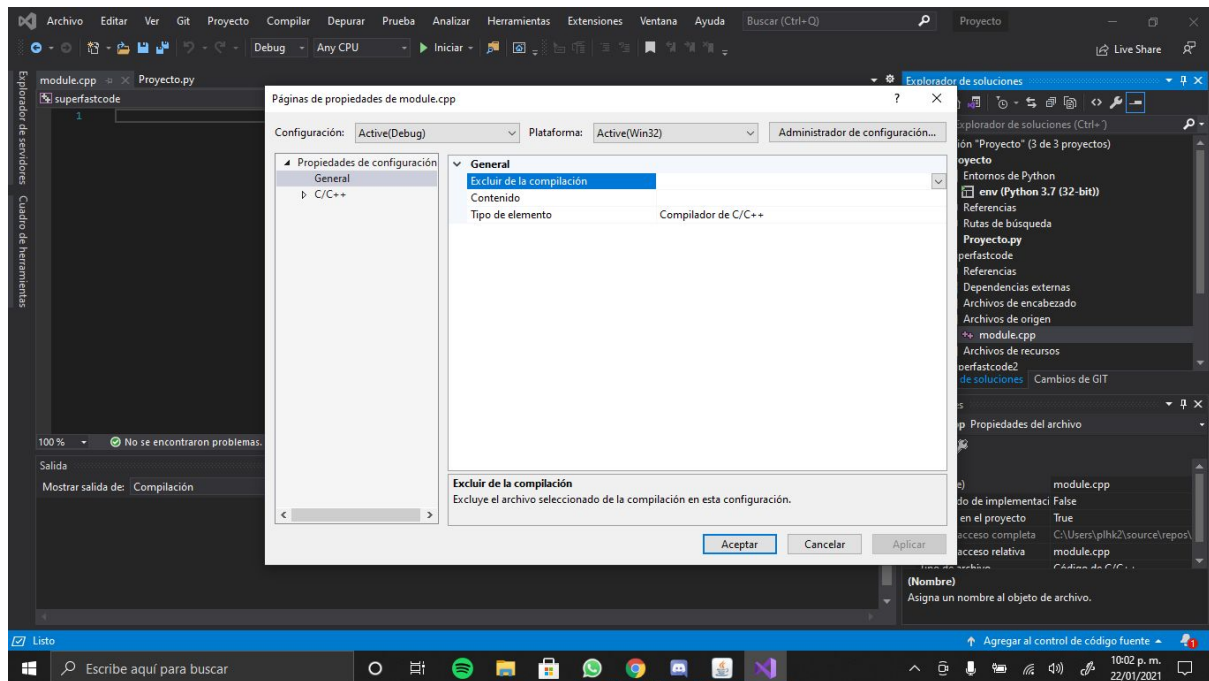


we right click on the C ++ project and we will give it in properties



and we will establish the platform as win 32 We will



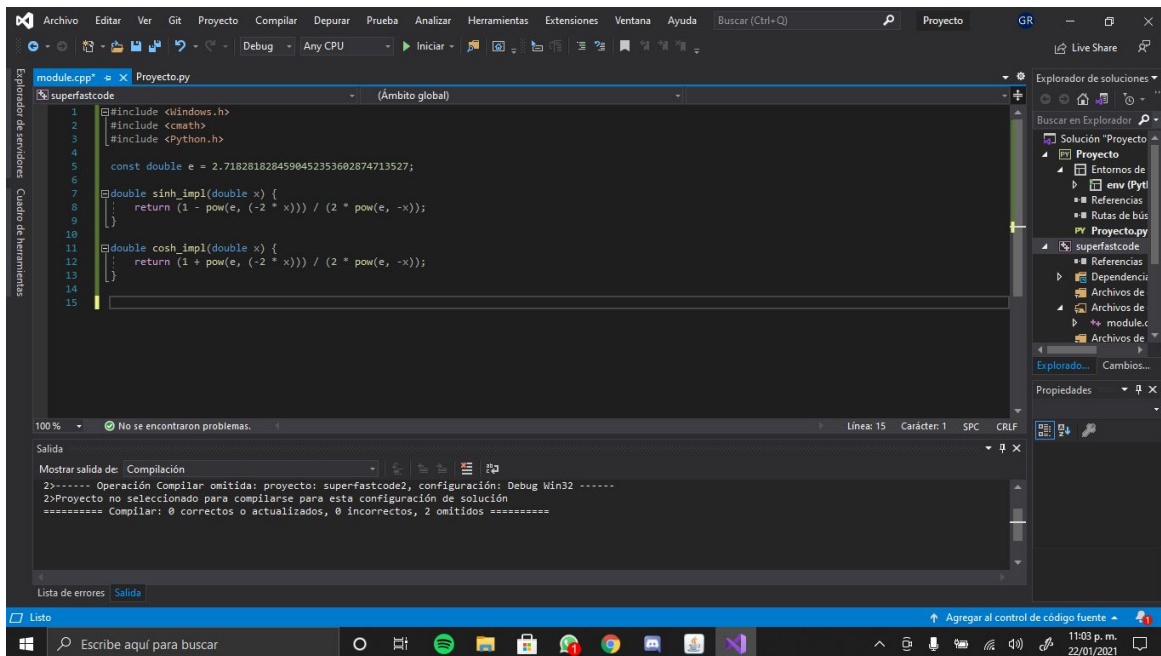


convert the C ++ projects into extensions for python

To convert the dll into an extension for python we must modify the exported methods so that they interact with python types, for this we must add a function that exports the module.

C extensions python

1- At the top of the module.cpp we will include python.h



```
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
```

100% No se encontraron problemas. Línea: 15 Carácter: 1 SPC CRLF

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

2- Modify the tanh_impl method to accept and return Python types (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 }
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     // Terminate the array with an object containing nulls.
26     { nullptr, nullptr, 0, nullptr }
27 };
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 =====

```

3- Add a structure that defines how the C ++ tanh_impl function is displayed in Python:

```

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     // Terminate the array with an object containing nulls.
26     { nullptr, nullptr, 0, nullptr }
27 };
28
29
30

```

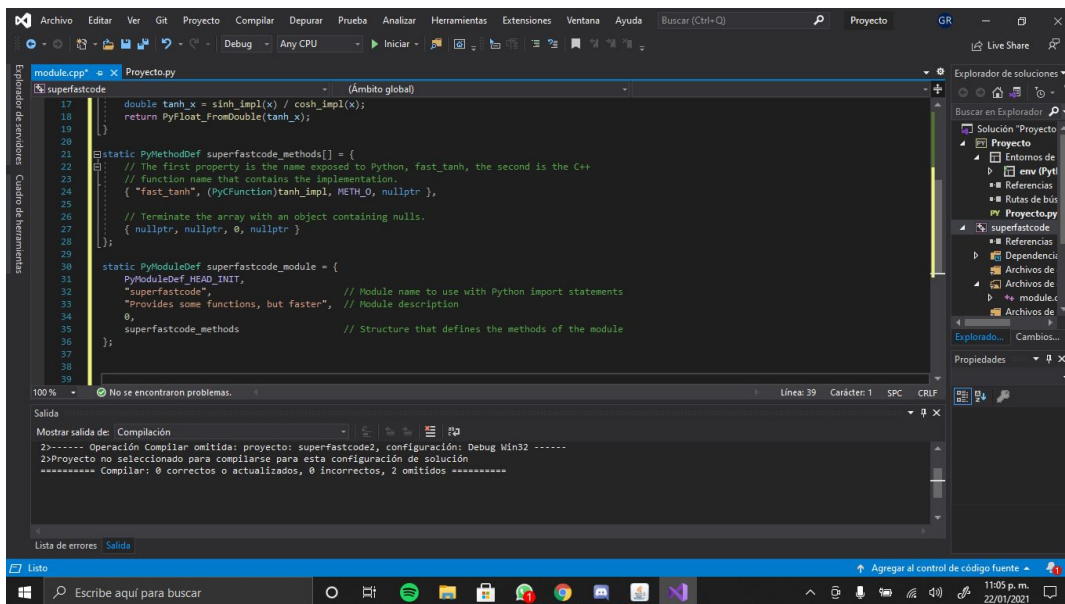
Salida

```

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 =====

```

4- Add a structure that defines the module how you want it to be referenced in Python code, specifically when using the statement from ... import. (Match it to the value of the project properties in Configuration Properties> General> Target Name). In the following example, the module name "superfastcode" means that you can use from superfastcode import fast_tanh in Python, because fast_tanh is defined in superfastcode_methods. (The internal file names of the C ++ project, such as module.cpp, do not matter.)



```

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", (PyFunction)tanh_impl, METH_O, nullptr },
25
26     // Terminate the array with an object containing nulls.
27     { nullptr, nullptr, 0, nullptr }
28 };
29
30 static PyModuleDef superfastcode_module = {
31     PyModuleDef_HEAD_INIT,
32     "superfastcode", // Module name to use with Python import statements
33     "Provides some functions, but faster", // Module description
34     0,
35     superfastcode_methods // Structure that defines the methods of the module
36 };
37
38
39

```

Salida

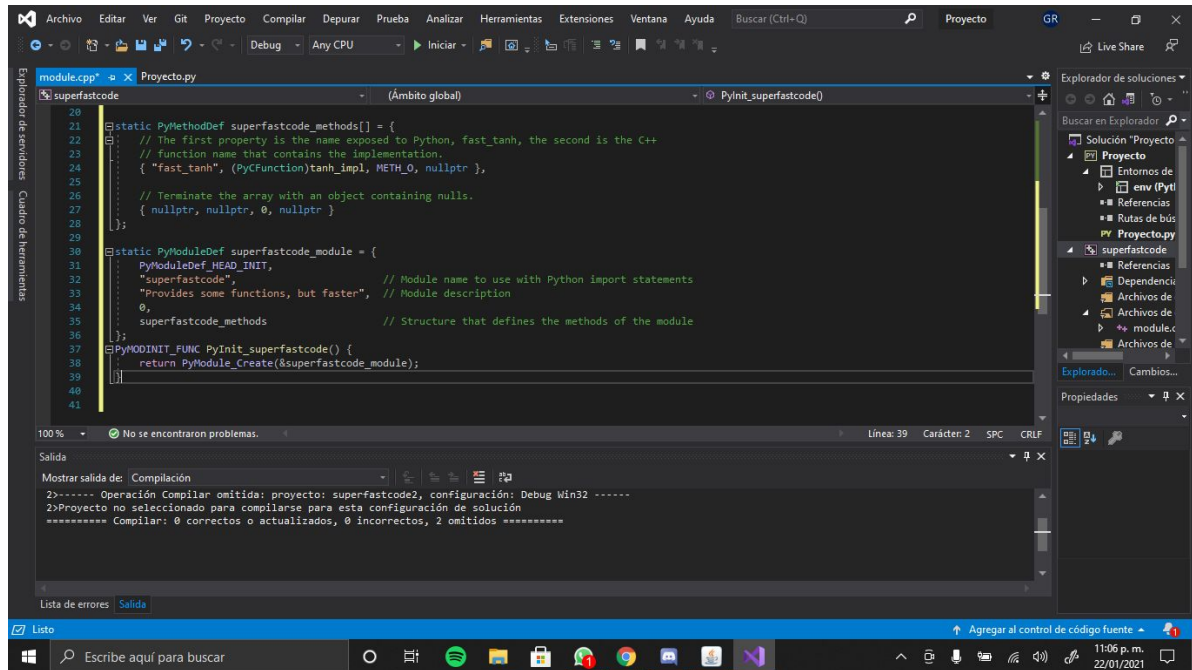
Mostrar salida de: Compilación

```

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 -----

```

5- Add a method that Python calls when the module loads, with the name PyInit_<module-name>, where <module_name> must exactly match the General> Target name property of the C ++ project (that is, it matches the name of the .pyd file compiled by the project).



```

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 static PyModuleDef superfastcode_module = {
31     PyModuleDef_HEAD_INIT,
32     "superfastcode", // Module name to use with Python import statements
33     "Provides some functions, but faster", // Module description
34     0,
35     superfastcode_methods // Structure that defines the methods of the module
36 };
37
38 PyMODINIT_FUNC PyInit_superfastcode() {
39     return PyModule_Create(&superfastcode_module);
40 }
41

```

100 % No se encontraron problemas. Línea: 39 Carácter: 2 SPC CRLF

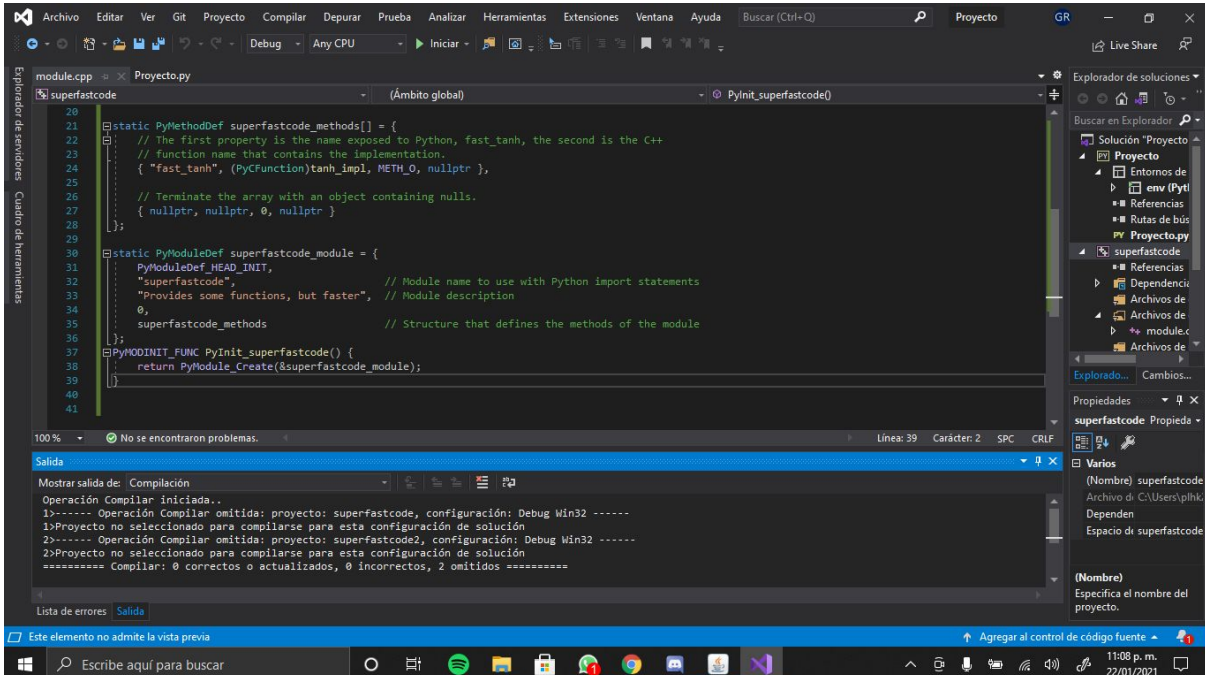
Salida

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

6-Set the target configuration to Launch and compile the C ++ project again to check the code.



```

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 static PyModuleDef superfastcode_module = {
31     PyModuleDef_HEAD_INIT,
32     "superfastcode", // Module name to use with Python import statements
33     "Provides some functions, but faster", // Module description
34     0,
35     superfastcode_methods // Structure that defines the methods of the module
36 };
37
38 PyMODINIT_FUNC PyInit_superfastcode() {
39     return PyModule_Create(&superfastcode_module);
40 }
41

```

100% No se encontraron problemas. Línea: 39 Carácter: 2 SPC CRLF

Salida

Mostrar salida de: Compilación

Operación Compilar iniciada..

1>----- Operación Compilar omitida: proyecto: superfastcode, configuración: Debug Win32 -----

2>Proyecto no seleccionado para compilarse para esta configuración de solución

2>----- Operación Compilar omitida: proyecto: superfastcode2, configuración: Debug Win32 -----

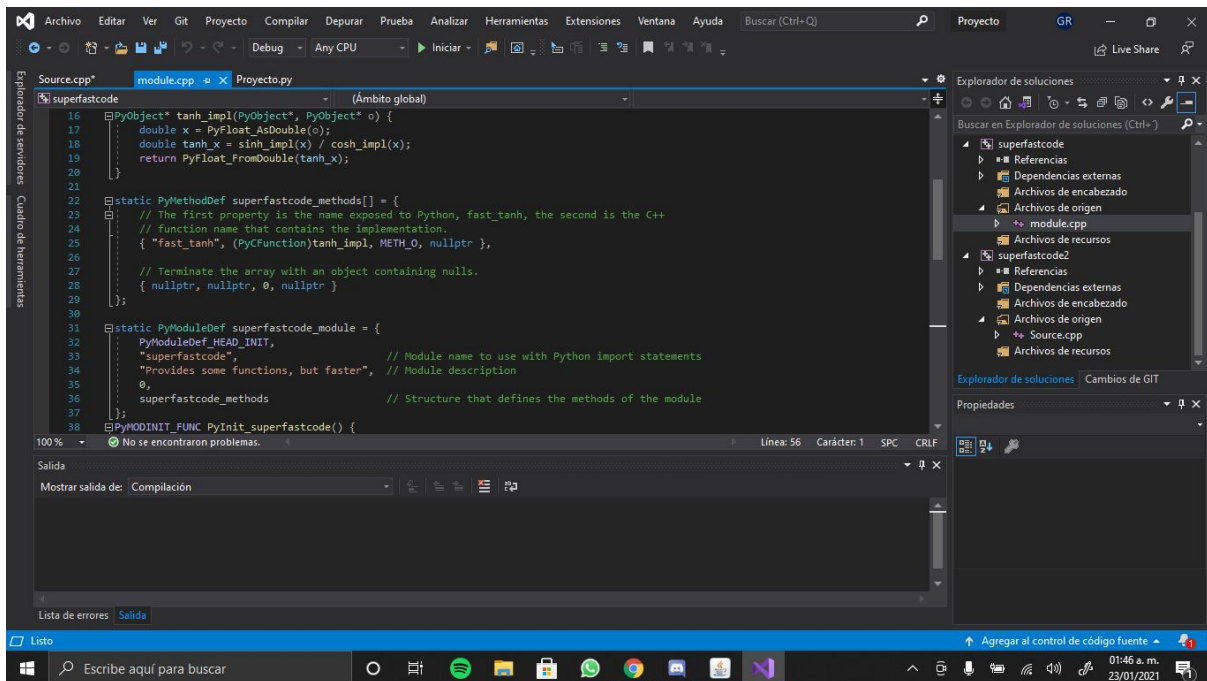
2>Proyecto no seleccionado para compilarse para esta configuración de solución

***** Compilar: 0 correctos o actualizados, 0 incorrectos, 2 omitidos *****

Lista de errores: Salida

Pybin11

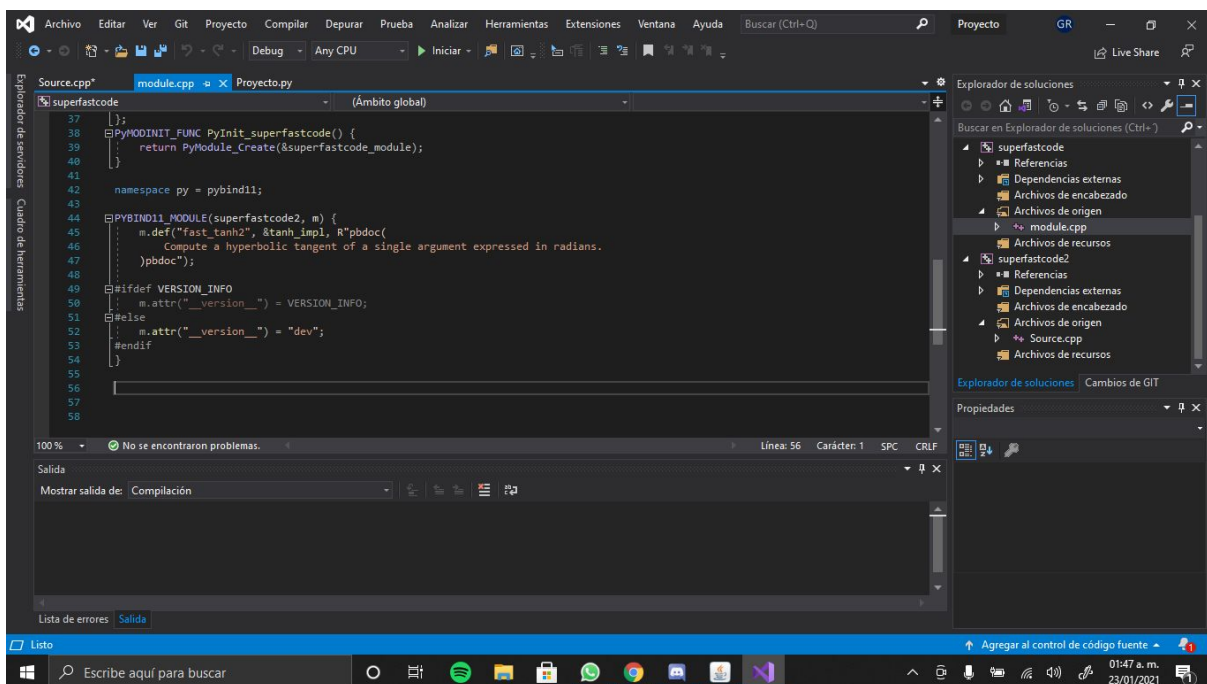
- 1-Install PyBind11 using pip: `pip install pybind11` or `py -m pip install pybind11`
- 2-At the top of module.cpp, include `pybind11.h`:
- 3-At the bottom of module.cpp, use the macro `PYBIND11_MODULE` to Define the entry point to the C ++ function:
- 4-Set the target configuration to Launch and compile the C ++ project to check the code. If errors occur, see the Troubleshooting section below.



```

16 PyObject* tanh_impl(PyObject*, PyObject* o) {
17     double x = PyFloat_AsDouble(o);
18     double tanh_x = sinh_impl(x) / cosh_impl(x);
19     return PyFloat_FromDouble(tanh_x);
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",
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     // No se encontraron problemas.
41 }

```



```

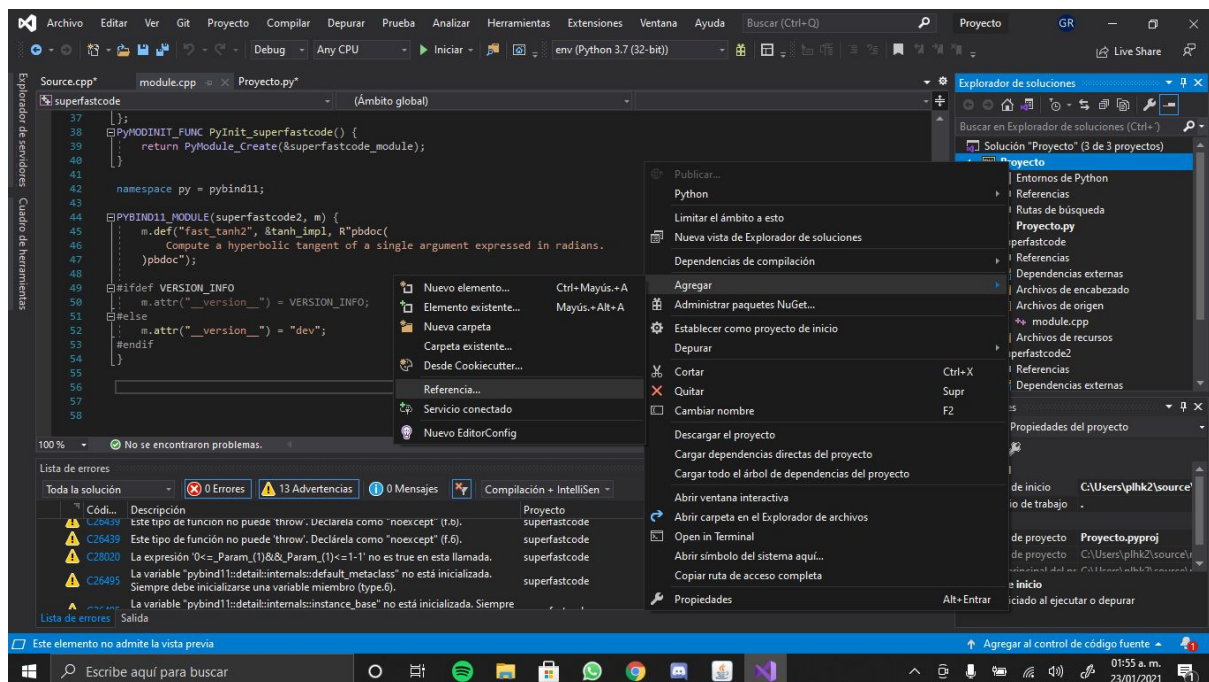
37 };
38 #PyMODINIT_FUNC PyInit_superfastcode() {
39     return PyModule_Create(&superfastcode_module);
40 }
41
42 namespace py = pybind11;
43
44 #PYBIND11_MODULE(superfastcode2, m) {
45     m.def("fast_tanh2", &tanh_impl, R"pbdoc(
46         Compute a hyperbolic tangent of a single argument expressed in radians.
47         )pbdoc");
48
49 #ifdef VERSION_INFO
50     m.attr("__version__") = VERSION_INFO;
51 #else
52     m.attr("__version__") = "dev";
53 #endif
54 }
55
56
57
58

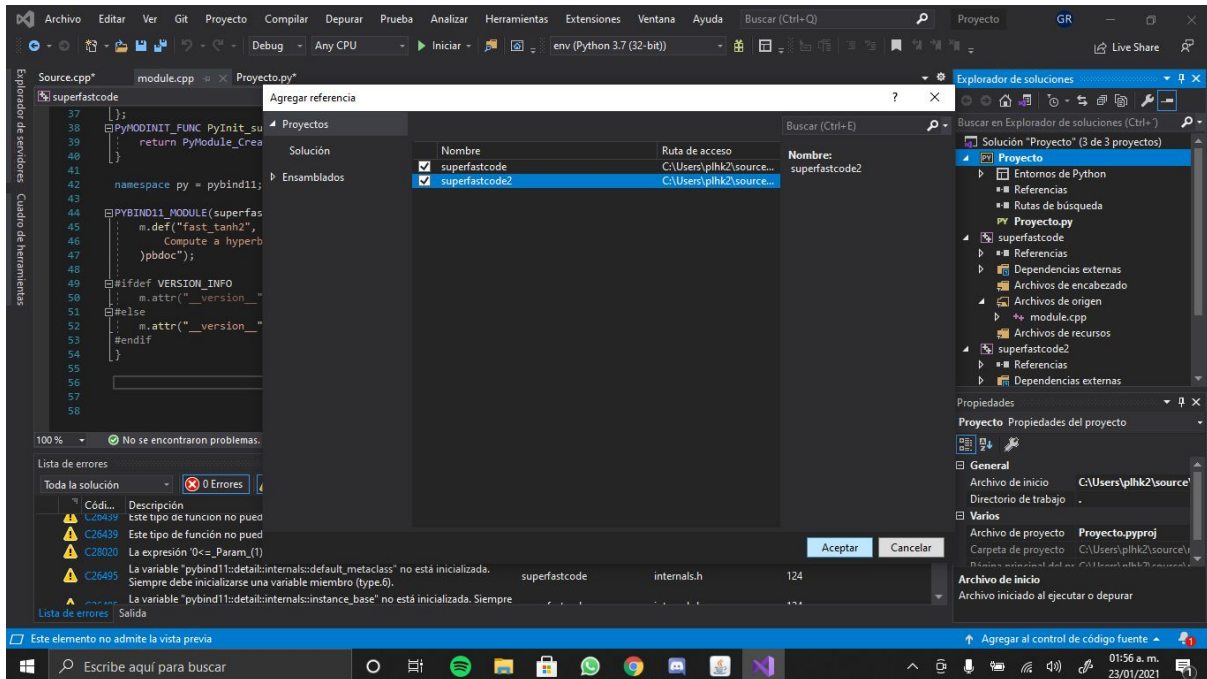
```

1- Test the code and compare the results Now that you have the DLL files structured as Python extensions, you can refer to them from the Python project, import the modules and use their methods.

2- Make the DLL file available to Python There are two ways to make the DLL file available to Python. The first method works if the Python project and the C ++ project are in the same solution.

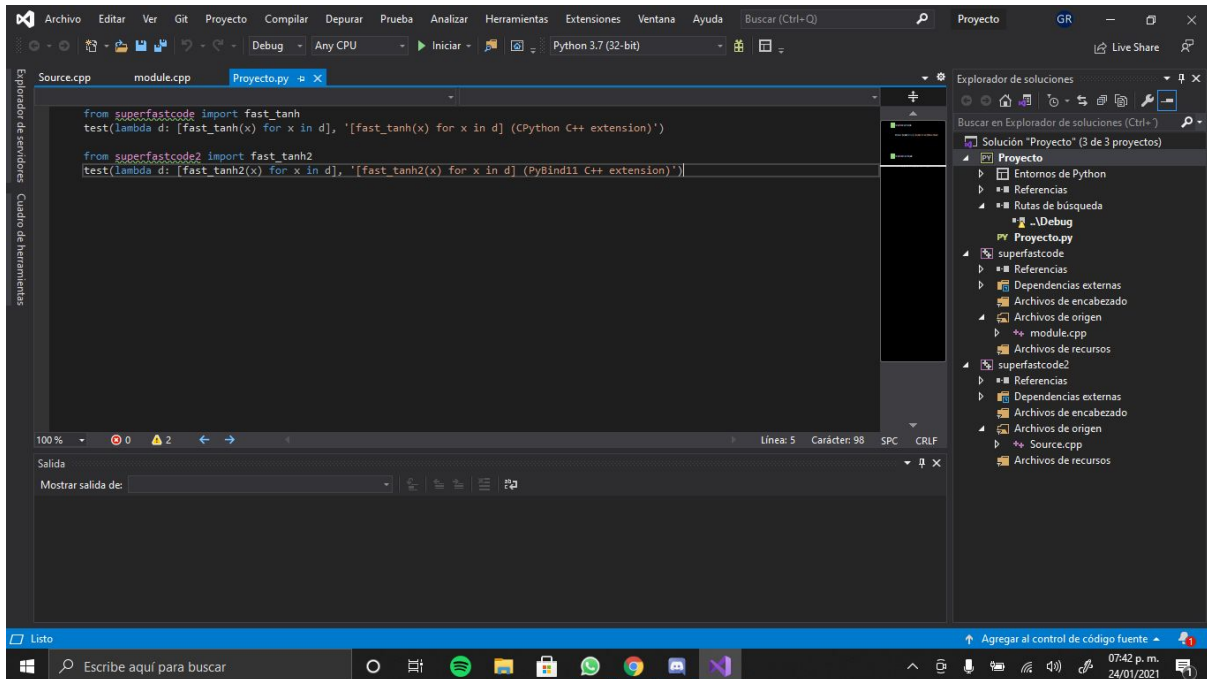
- Go to Solution Explorer, right-click the References node in your Python project, and then select Add Reference. In the dialog that appears, select the Projects tab, the superfastcode and superfastcode2 projects, and OK.



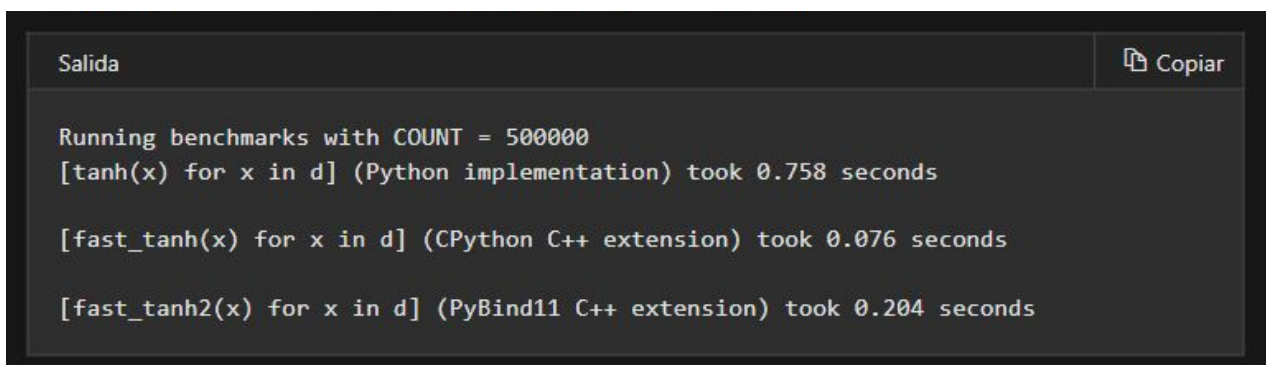


Calling the DLL from Python Once the DLL is available to Python as described in the previous section, you can now call the `superfastcode.fast_tanh` and `superfastcode2.fast_tanh2` functions from the Python code and compare its performance against the implementation of Python:

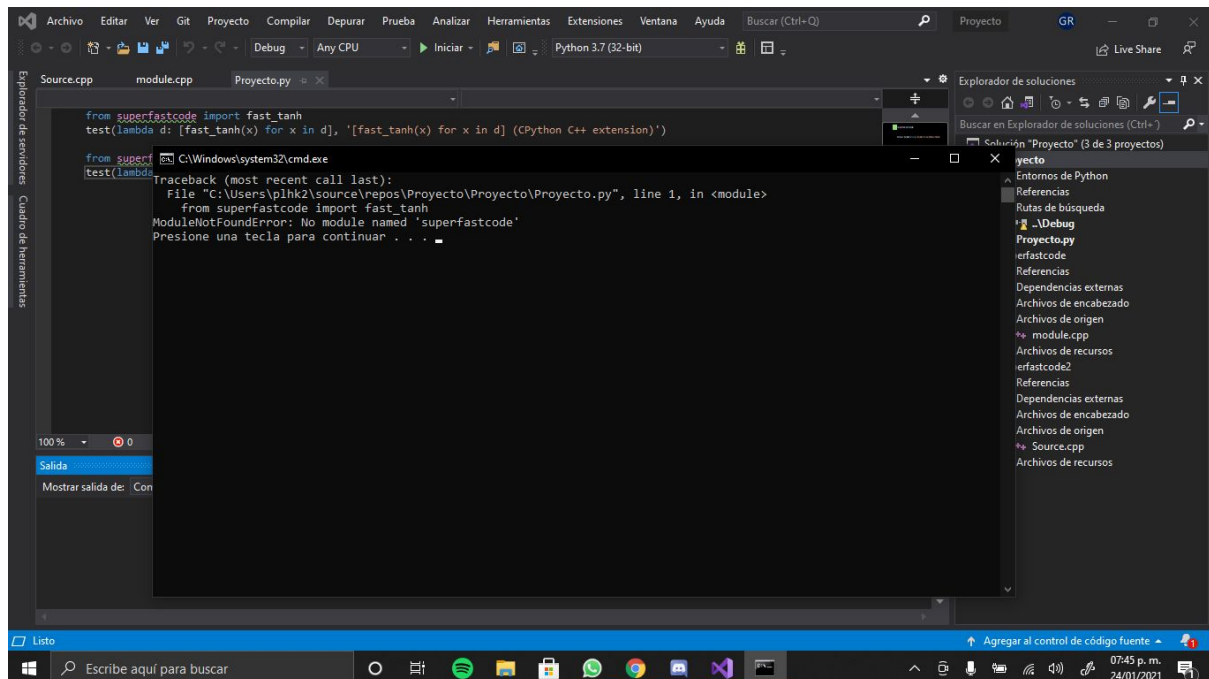
1- Add the following lines in the .py file to call the exported methods from the DLLs and display the results.



Run the Python program (Debug> Start Without Debugging or Ctrl + F5) and notice that C++ routines run approximately five to twenty times faster than the Python implementation. The typical output appears as follows:



In our case we got a different result since it shows us the following window.



When reviewing the solutions we found that on the page the mentioned solution requires Visual Studio of the previous version since the version we have is the most Recent (2019) and the post on which we rely to verify this connection is not updated for this version, it was not possible for us to demonstrate the connection between two programming languages.

What we noticed in the development of this project was that one of the advantages of this new library that the teacher assigned us is the compatibility with another programming language since it has new libraries within it that allow it and are faster .

Functional and non-functional requirements:

Functional:

- Better support in Visual Studio IDE
- edit, compile and debug C ++ projects Visual Studio
- Improves performance in c ++ development environment Visual Studio

does not functional:

- Quick execution.
- Fix as defined by the C ++ 11 standard
- . Fast compile times.
- Minimal memory usage.

SOURCES:

Anonymous. (s / f). C ++ standard library "libc ++". December 2020, from LLVM
Website: <https://libcxx.llvm.org/>

Anonymous. (s / f). libc ++ 8.0 documentation. December 03, 2020, from bcain
Website: <https://bcain-llvm.readthedocs.io/projects/libcxx/en/latest/>

Meza Juan D. (2020). Libraries or libraries in C ++. Declaration and use of libraries.
Include in C ++. Retrieved on 12/10/2020, from the website
<https://www.programarya.com/Ccursos/C++/Bibliotecas-o-Librerias>

Anónimo (2018). Creating a C ++ extension for Python. Retrieved 12/12/2020, from
the site
<https://docs.microsoft.com/es-es/visualstudio/python/working-with-c-cpp-python-in-visual-studio?view=vs-2019>

Anonymous (2020). Clang / LLVM support in Visual Studio projects. Retrieved
12/12/2020. from the site
<https://docs.microsoft.com/en-us/cpp/build/clang-support-msbuild?view=msvc-160>

Emanuel (2020). Clang ++ ABI is the same as g ++ ?. Retrieved on 01/08/2021 from
the site <https://www.javaer101.com/es/article/2948533.html>

Anonymous. (04/20/2020). Build and run a C ++ console application project.
01/22/2021, from Microsoft Website:
<https://docs.microsoft.com/es-es/cpp/build/vscpp-step-2-build?view=msvc-160>

Anonymous. (04/20/2020). Create a C ++ console application project. 01/22/2021,
from Microsoft Website:
<https://docs.microsoft.com/es-es/cpp/build/vscpp-step-1-create?view=msvc-160>

Anonymous. (05/11/2020). C and C ++ compatibility installation in Visual Studio. 01/22/2021, from Microsoft Website:

<https://docs.microsoft.com/es-es/cpp/build/vscpp-step-0-installation?view=msvc-160>

Anonymous. (04/20/2020). Create a C ++ console application project. 01/22/2021, from Microsoft Website:

<https://docs.microsoft.com/es-es/cpp/build/vscpp-step-1-create?view=msvc-160>