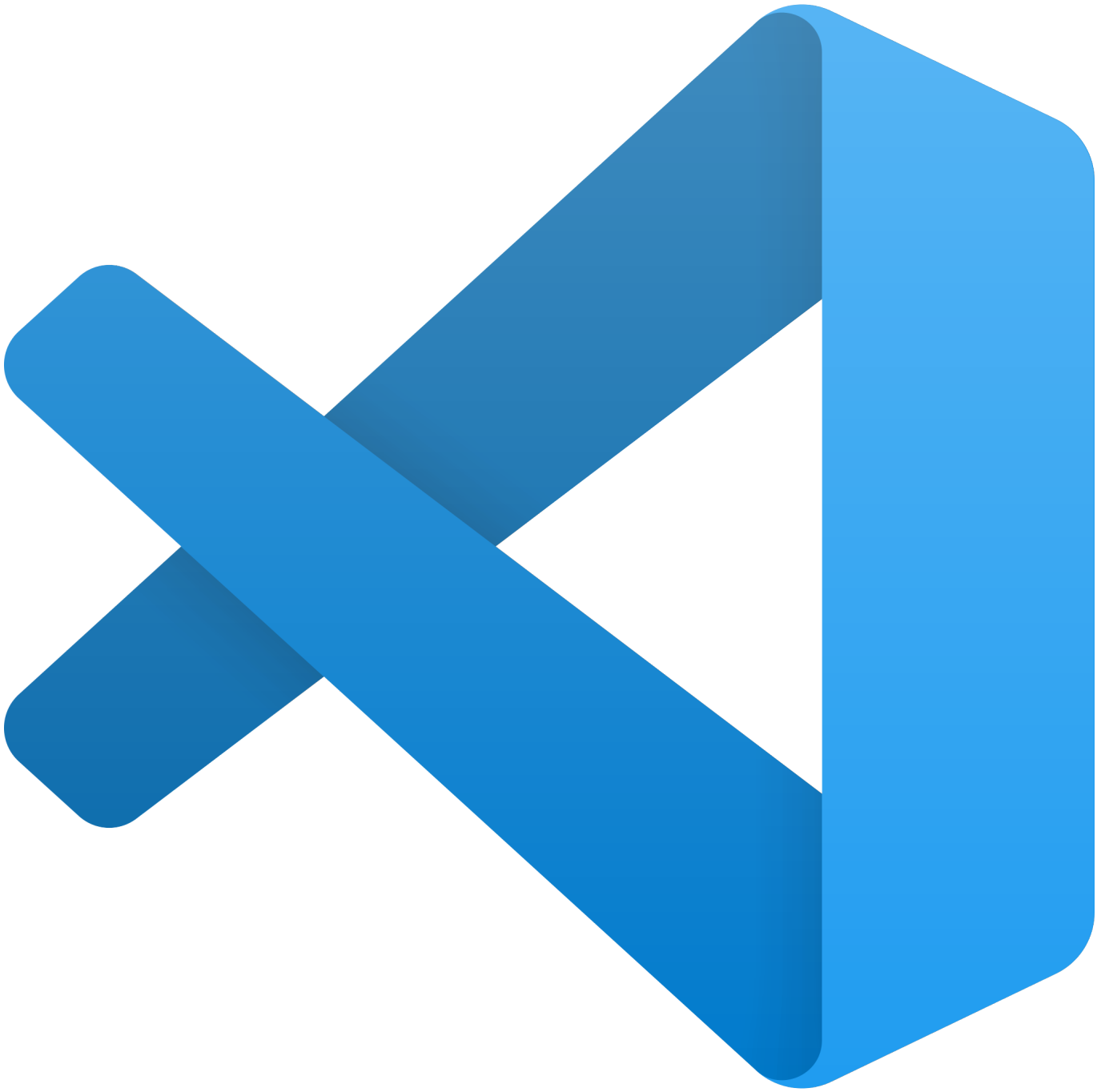
Notas de Clase - Programación

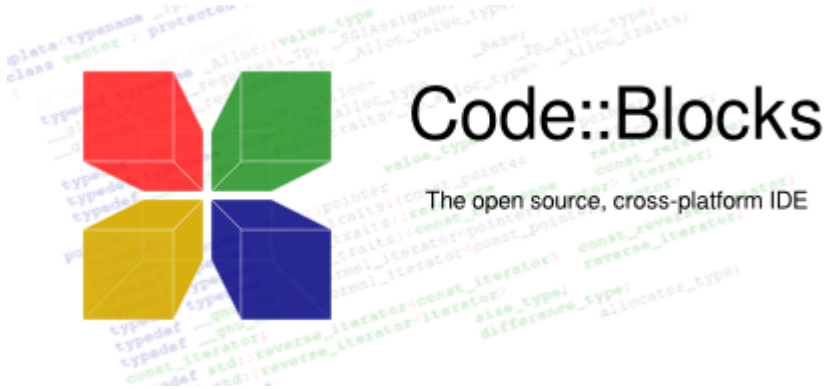
Primer Bimestre

- Preparación

  Antes de empezar a programar necesitamos un entorno de trabajo o Workspace.
  En nuestro caso utilizaríamos VS Code por sus extensiones (pero usaremos CodeBlocks para los ejemplos, ya que es más simple), un compilador de C/C++, y Git (para trabajar en la nube).

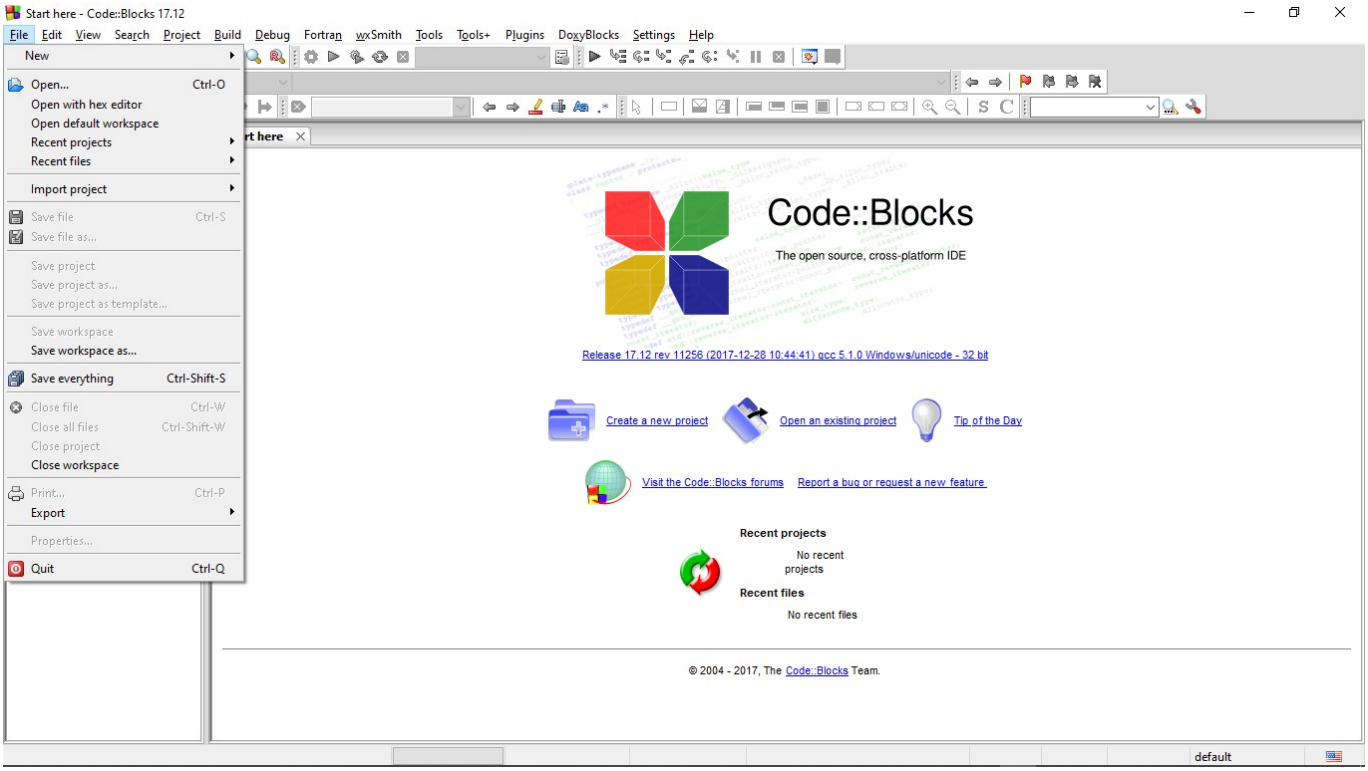



GCC

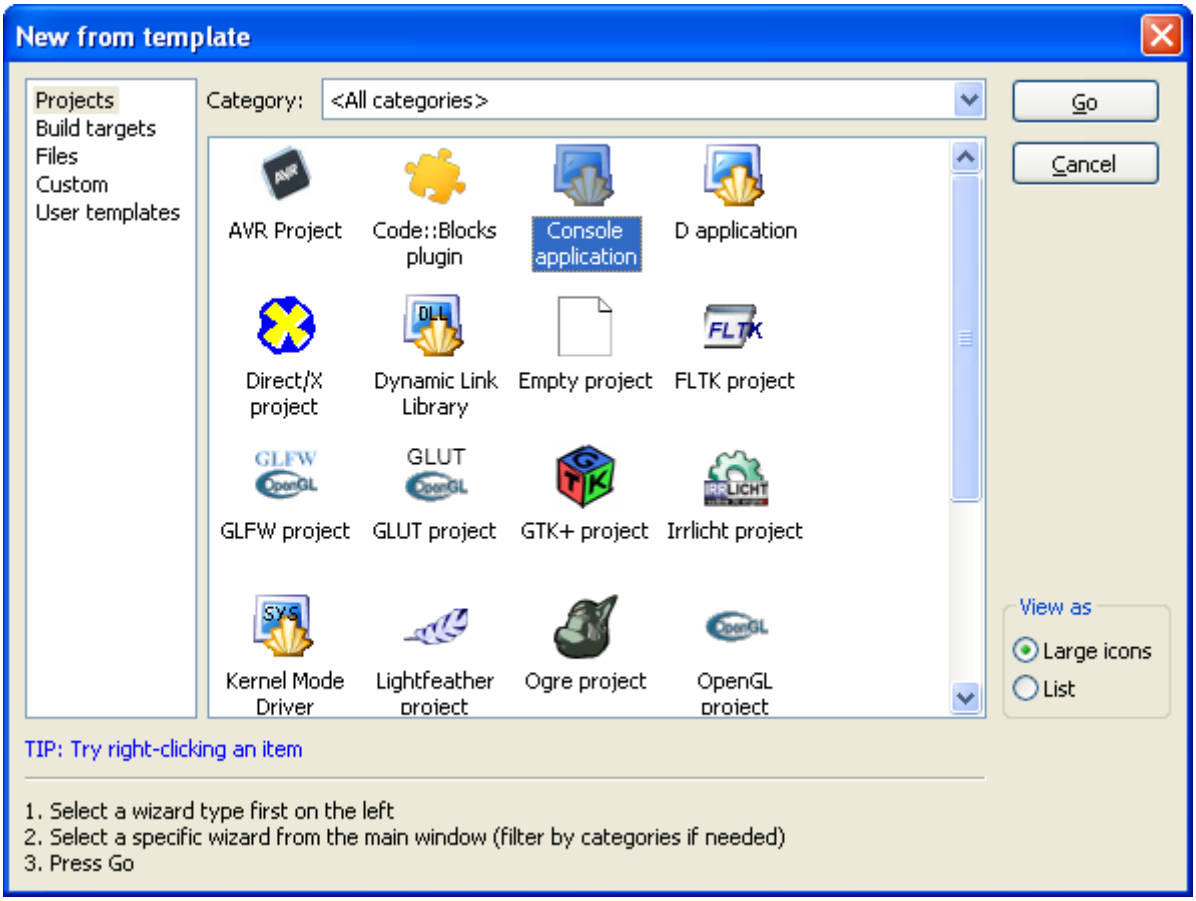


Configuración de un proyecto

Una vez intalado el IDE, lo abrimos y pulsamos en nuevo proyecto

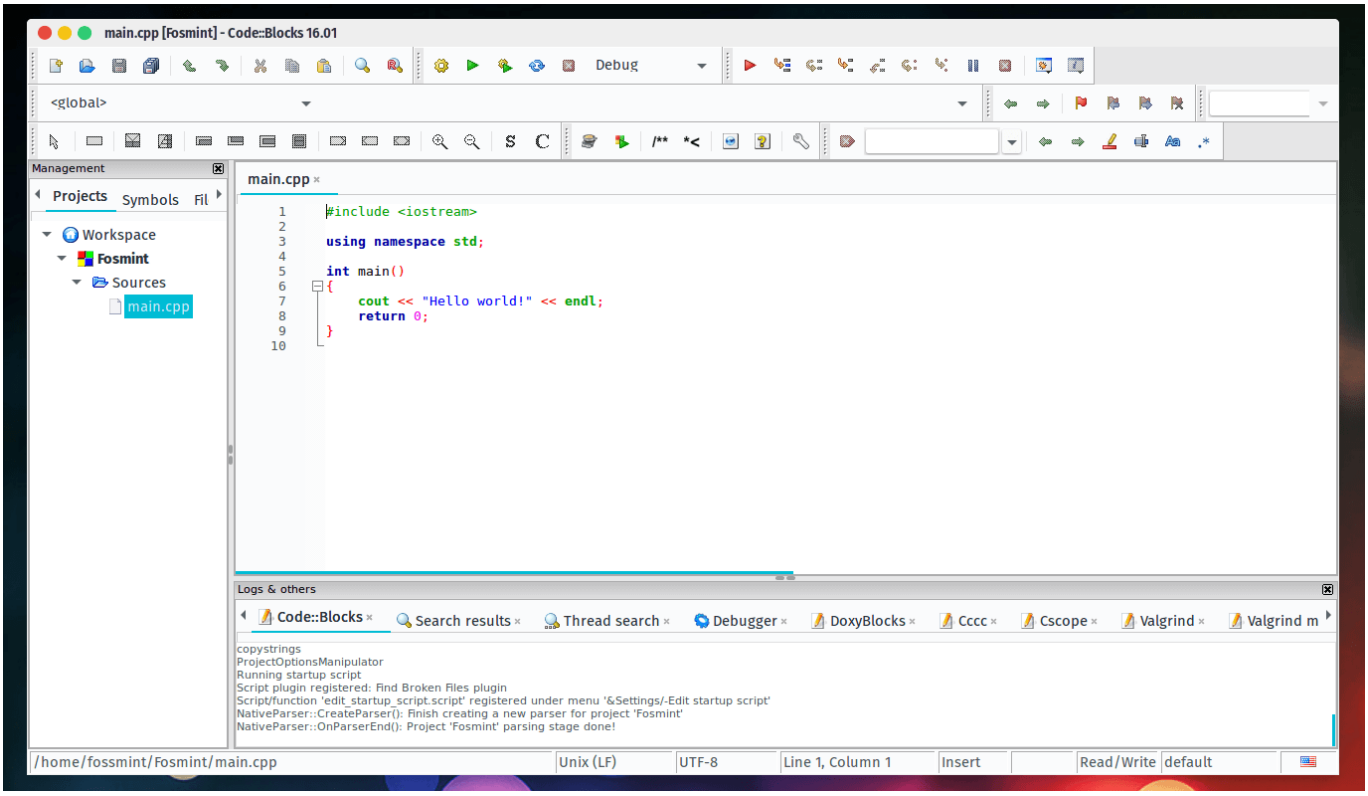


Seleccionamos "Console Aplicattion"



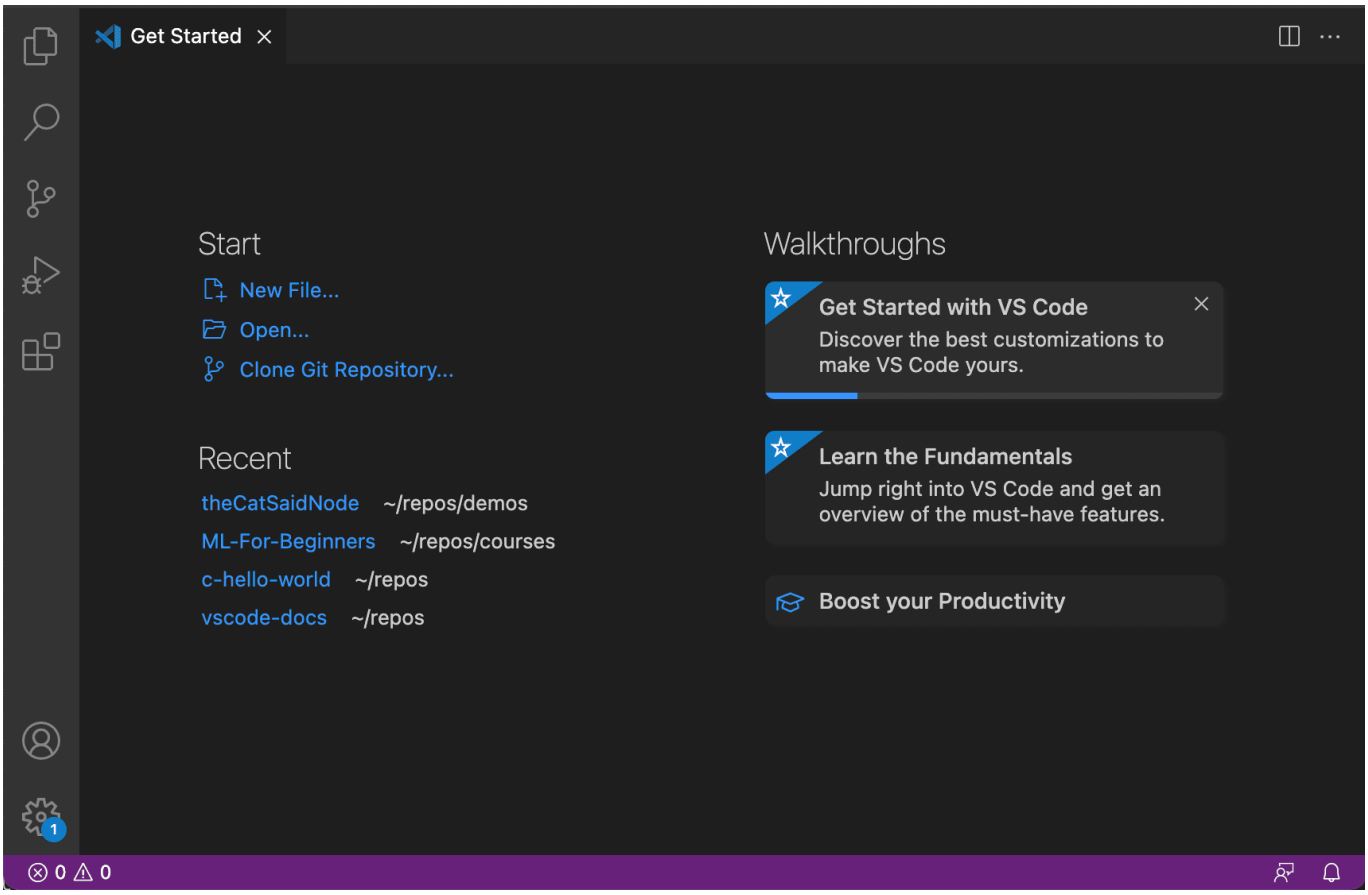
Seleccinamos nuestro lenguaje C o C++, elegimos la ubicación y le damos siguiente (varias veces).

Esto nos creará una carpeta junto con un archivo ".cpp" . En ella y el trabajaremos.



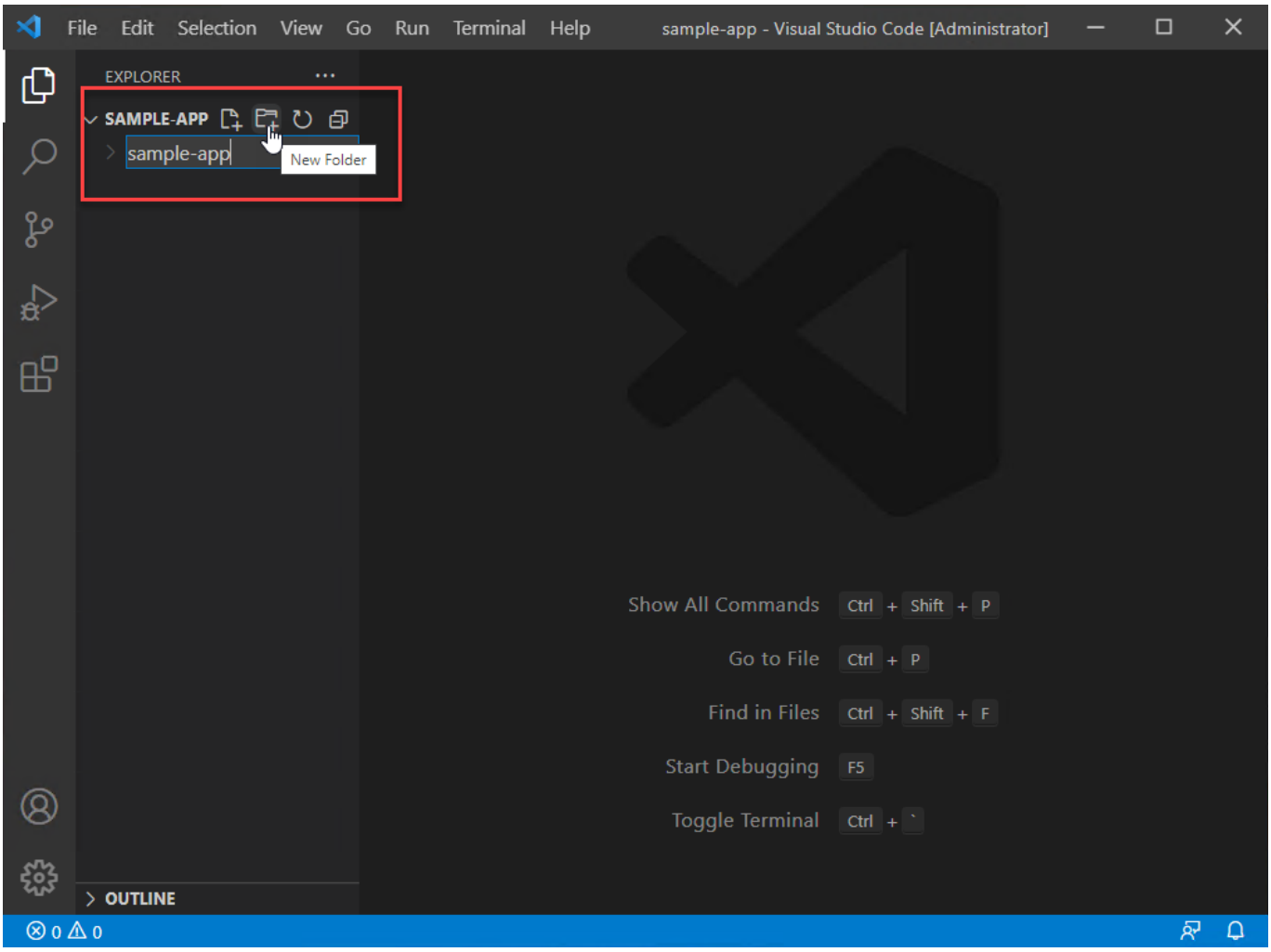
> Para imprimir sólo vamos al botón verde con la ruedita de arriba

En Visual Studio en cambio,

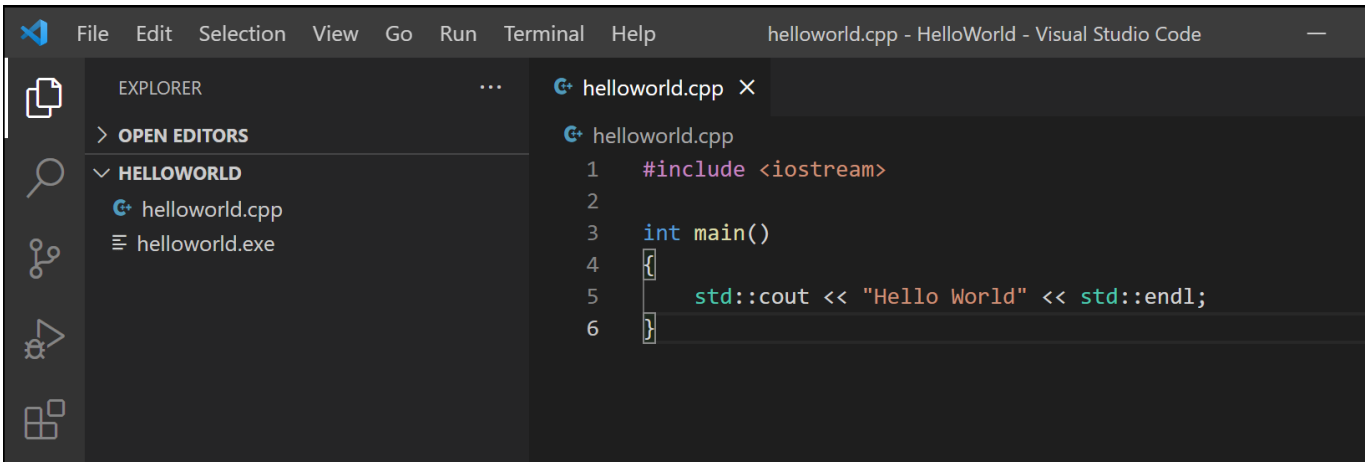


Le damos a Open y elegimos la ubicación.

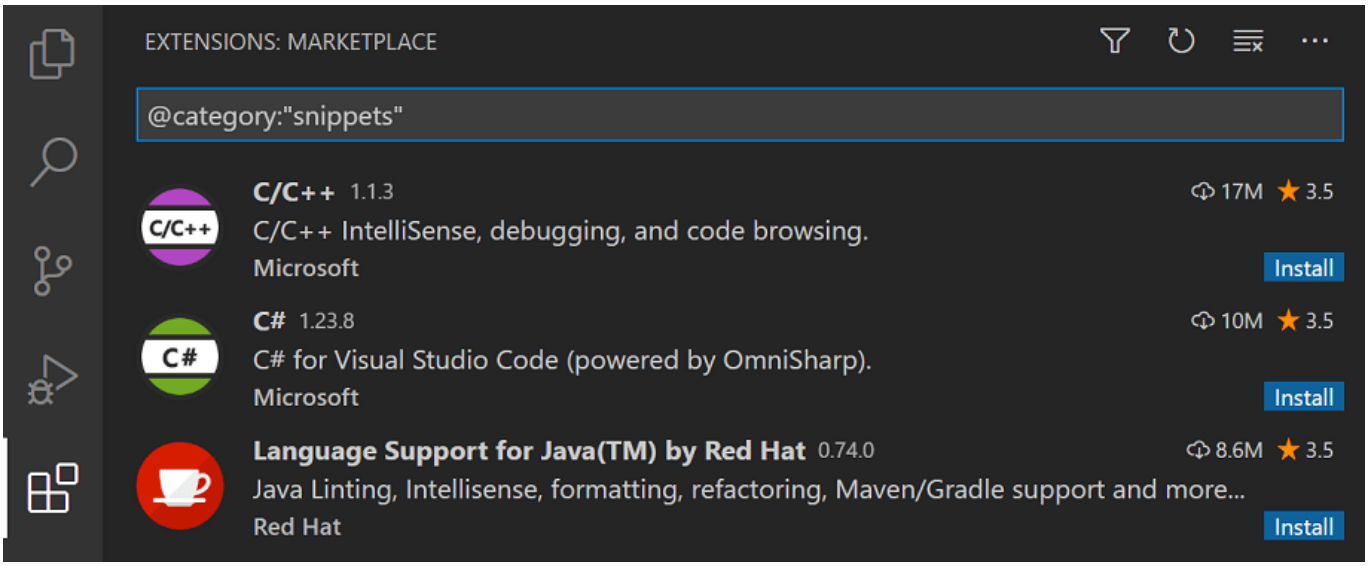
En nuestra carpeta creamos un archivo con el ícono que aparece a lado de la bara de la carpeta



Lo nombramos y terminamos con un .c o .cpp



Al crear el archivo el propiop programa nos ofrecerá intalar extensiones para C. Aceptamos y Listo!



Una vez configurado el entorno, intalado el compilador y modificado el repositorio; Es hora de escribir código 😃

- C/C++ lenguaje

  El lenguaje C y C++ difieren un poco en lo que respecta a su funcionamiento, mas no en su sintaxis por lo que eligiremos a C++ como lenguaje principal. Creado el archivo obtenemos lo siguiente

```
#include <iostream> //Esto serían las librerías a usar
using namespace std; //Formato para no escribir "std::" todo el tiempo

int main()  //Nuestra función principal
{
  cout << "Hello world!" << endl;
  return 0; //Es el valor que nos devolverá la función al terminar su tarea
}
```

```
#include <stdio.h> //esta es la librería estándar en C

int main(){
printf("Hello World!\n");
return 0;
}
```

> "Cout" significa imprimir y es lo que aparecerá en nuestra pantalla "printf" si elegiste C.

> De la misma manera "endl" o "\n" hará que el programa imprima lo que le digas una línea abajo, ya que el lenguaje por defecto imprime las cosas sin espacios.

Este código imprimirá lo siguiente:

```
Hello World!
```

Sobre las librerías y funciones hablaremos más adelante.

- Dibujando una forma

Cuando corramos un programa la funcion main() iumprimirá cualquier línea de código escrito entre {}.

Cada oración funciona como una instrucción por lo que debemos terminar con un ; cada una de ellas (existen palabras y comandos que no lo necesitan después hablaremos de ello).
Por ejemplo si modificamos el número de comandos cout e improimos un símbolo:

```
#include <iostream>
using namespace std;

int main()
{
  cout << "|----/" << endl;
  cout << "|   /" << endl;
  cout << "|  /" << endl;
  cout << "| /" << endl;
  cout << "|/
}
```

Tenemos:

```
|----/
|   /
|  /
| /
|/
```

Lo que sería un triángulo de cabeza mal dibujado.

- Variables

Una variable almacena valores que el programa leerá cuando funcione

```
#include <iostream>

using namespace std;

int main()
{
  cout << "There once was a man named George" << endl;
  cout << "He was 70 years old" << endl;
  cout << "He like the name George" << endl;
  cout << "But did not like being 70" << endl;
```

```
  return 0;
}
```

Imprime

```
There once was a man named George
He was 70 years old
He like the name George
But didn't like being 70
```

Tal vez queramos cambiar el nombre de nuestro personaje o su edad

```
#include <iostream>

using namespace std;

int main()
{
  cout << "There once was a man named John" << endl;
  cout << "He was 35 years old" << endl;
  cout << "He like the name John" << endl;
  cout << "But did not like being 35" << endl;
  return 0;
}
```

Para cambiar el nombre tendríamos que buscarlo y modificarlo de uno en uno.
Lo que en grandes cantidades de datos nos sería imposible. Así que creamos una variable que almacene el nombre y la edad.

```
#include <iostream>

using namespace std;

int main()
{
  string characterName = "John";
  int characterAge; //podemos asignarle un valor a la variable en el momento
o expresarlo más adelante
  characterAge = 35;
  cout << "There once was a man named John" << endl;
  cout << "He was 35 years old" << endl;
  cout << "He like the name John" << endl;
  cout << "But did not like being 35" << endl;
  return 0;
}
```

> Aquí string indica el tipo de dato lo que sería una cadena de caracteres, e int como integer (entero) almacena valores numéricos sin decimales.

Cuando trabajamos con datos en C/C++ las comillas indican oraciones o caracteres que se imprimirán como tal, en cambio los números no funcionan así por lo que no necesitarán "" alrededor suyo

Para imprimir la variable y su valor debemos modificar nuestro cout command.

```
#include <iostream>

using namespace std;

int main()
{
  string characterName = "John";
  int characterAge;
  characterAge = 35;
  cout << "There once was a man named" << characterName << endl;
  cout << "He was 35 years old" << endl;
  cout << "He like the name John" << endl;
  cout << "But did not like being 35" << endl;
  return 0;
}
```

```
#include <iostream>

using namespace std;

int main()
{
  string characterName = "John";
  int characterAge;
  characterAge = 35;
  cout << "There once was a man named " << characterName << endl;
  cout << "He was " << characterAge << " years old" << endl;
  cout << "He like the name " << characterName << endl;
  cout << "But did not like being " << characterAge<< endl;
  return 0;
}
```

> Las comillas en la onceava línea indican que el valor de la variable se imprimirá en ese lugar, no al final como las otras oraciones.

Y tenemos:

```
There once was a man named John
He was 35 years old
```

```
He like the name John
But did not like being 35
```

Ahora podemos modificar como queramos:

```
There once was a man named Tom
He was 11 years old
He like the name Tom
But did not like being 11
```

También podemos actualizar el valor de la variable para imprimir otros valores dentro de la misma función:

```
#include <iostream>

using namespace std;

int main()
{
  string characterName = "Tom";
  int characterAge;
  characterAge = 11;
  cout << "There once was a man named " << characterName << endl;
  cout << "He was " << characterAge << " years old" << endl;
  characterName = "Mike";
  cout << "He like the name " << characterName << endl;
  cout << "But did not like being " << characterAge<< endl;
  return 0;
}
```

Output:

```
There once was a man named Tom
He was 11 years old
He like the name Mike
But did not like being 11
```

Divertido ¿no?

- Tipos de datos

Existen múltiples tipos de datos en C/C++ aquí están los más comunes:

```
#include <iostream>

using namespace std;

int main()
{
  char grade = 'A';
  string phrase = "Learning to code";
  int age = 50;
  float e = 2.71828182846;
  double pi = 3.14159265359;
  bool isMale = true;

  cout << grade << endl;
  cout << phrase << endl;
  cout << age << endl;
  cout << e << endl;
  cout << pi << endl;
  cout << isMale << endl;

  return 0;
}
```

char: almacena un solo caracter, en este caso "A".

string: guarda mútiples caracteres.

int: como se especificó arriba, guarda solo valores enteros.

float: al igual que double almacena valores con cifras decimales, pero a diferencia de el, este guarda menor cantidad.

double: mayor cantidad de cifras decimales

boolean: escrito "bool" almacena valores de verdadero o falso; true 1 | false 0.

Output:

```
A
Learning to code
50
2.71828
3.14159
1
```

- Trabajando con strings

Existen funciones de strings, estas nos ayudarán a obtener información de la string o modificarla.

- **.length()** permite saber el numero de caracteres introducidos en la variable

```
#include <iostream>

 using namespace std;

int main()
 {
 string collegeName = "EPN";
 cout << collegeName.length();
 return 0;
 }
```

Output:

```
3
```

- **Cambiar valores en la cadena** Podemos acceder a diferentes valores dentro de la cadena, llamando al index de aquel valor

```
#include <iostream>

using namespace std;

int main()
{
                      012
string collegeName = "EPN";
//El index de un valor de string estaría dado desde el 0.
cout << collegeName[0];
return 0;
}
```

Output:

```
E
```

Podemos cambiar los caracteres usando el index

```
#include <iostream>

using namespace std;

int main()
```

```
{
string collegeName = "EPN";
collegeName[0] = 'A';
cout << collegeName;
return 0;
}
```

Output:

```
APN
```

- **.find()** encontrar posiciones dentro de una cadena.

```
#include <iostream>

using namespace std;

int main()
{
string collegeName = "Escuela Politécnica Nacional ";
cout << collegeName.find("Escuela", 0);
return 0;
}
```

> Aquí queremos encontrar en qué posición está la primera letra de "Escuela" dentro de la otra string, así que se lo damos de parámetros.

Output:

```
0
```

- **.substr** Coger un número de elementos dentro de cierta posición.

```
#include <iostream>

using namespace std;

int main()
{
string collegeName = "Escuela Politécnica Nacional ";
cout << collegeName.substr(10, 3);
return 0;
}
```

> Al igual que la anterior le pasamos ciertos parámetros para indicarle que quermos hacer.

Output:

```
lit
```

Podemos guardar aquel valor dentro de otra variable

```
#include <iostream>

using namespace std;

int main()
{
string collegeName = "Escuela Politécnica Nacional ";
string cutCollegename = collegeName.substr(10, 3);
cout << cutCollegename << endl;
return 0;
}
```

Output:

```
lit
```

- Trabajando con números

En C++ tenemos la librería , esta nos ayudará a usar operaciones más complejas Por ejemplo:

```
#include <iostream>
#include <cmath>

using namespace std;
//Obtener el mayor de dos números
int main()
{
  cout << max(5, 7);
  return 0;
}
```

Output:

```
7
```

O:

```
#include <iostream>
#include <cmath>

using namespace std;
//Obtener el menor de dos números
int main()
{
  cout << min(4,8);
  return 0;
}
```

Output:

```
4
```

> Puedes usar sqrt(para raiz), pow(para elevar) o redondear(round), siempre usando dos numeros como parámetros.

- Obtener información del usuario

Usamos la palabra "cin >>" para recibir información del usuario para trabajar.

```
#include <iostream>

using namespace std;
int main()
{
  int age;
  cout << "enter your age: ";
  cin >> age;
  cout << "Your are " << age << " years old";

  return 0;
}
```

Output:

```
enter your age:
```

el programa esperará hasta que sea introducida la información o algún carácter incorrecto
Si quieres obterner más información por ejemplo un nombre o una línea de texto, puedes usar

getline(cin, nombre_de_la_variable) donde el espacio donde se guardará el valor de la variable será un string.

```
include <iostream>

using namespace std;
int main()
{
  string name;
  cout << "Enter your name: ";
  getline(cin, name);
  cout << "Your are " << name;

  return 0;
}
```

Output:

```
Enter your name:
```

- Arrays

Un array o arreglo es una variable que almacena múltiples datos se lo define con [] (dentro de el puedes colocar el espacio que usarás, cuando no estás segurop de la informacion a guardar), y para acceder a sus datos tienes que escribir el nombre del array junto a su index como se vio al inicio.

```
#include <iostream>

using namespace std;
int main()
{
 int luckyNums[] = {4, 8, 15, 16, 23, 42};
 cout << luckyNums[5];

 return 0;
}
```

Output:

```
42
```

> Puedes colocar los elementos por ti mismo a dejarlos en blanco.

- Funciones

Son bloques de código que realizan acciones

```
#include <iostream>

using namespace std;
void sayHi(){
  cout << "Hello You";
}

int main()
{
  sayHi();
  return 0;
}
```

En este caso cuando la función es llamado realiza la acción de lo contrario permanecerá inmutable.

Output:

```
Hello You
```

También podemos pasarle parámetros para realizar la operación con un valor en específico

```
#include <iostream>
using namespace std;
void sayHi(string name){  //aquí creamos la variable en la que entrará el
valor
  cout << "Hello "<< name;
}

int main()
{
  sayHi("Eduardo"); //Introducimos el parámetro
  return 0;
}
```

Output:

```
Hello Eduardo
```

Podemos reusar funciones pero pasarles diferentes valores

```
#include <iostream>
```

```
using namespace std;
void sayHi(string name,int age){
  cout << "Hello "<< name << " You are " << age;
}

int main()
{
  sayHi("Eduardo", 19);
  return 0;
}
```

Output:

```
Hello Eduardo You are 19
```

> Recuerda escribir o definir la función antes de main

```
#include <iostream>

using namespace std;

void sayHi(string name,int age);
int main()
{
  sayHi("Eduardo", 19);
  sayHi("Emily", 34);
  sayHi("Remi", 24);
  sayHi("Tom", 18);

  return 0;
}
void sayHi(string name,int age){
  cout << "Hello "<< name << " You are " << age << endl;
}
```

Output:

```
Hello Eduardo You are 19
Hello Emily You are 34
Hello Remi You are 24
Hello Tom You are 18
```

- "return" keyword

  Usar return dentro de una función hará que esta devuelva valores con los que puedes trabajar en otra función

```
#include <iostream>

using namespace std;

double cube(double num){
return num*num*num;

}

int main()
{
  cout << cube(5.0);

  return 0;
}
```

Output:

```
125
```

cualquier valor dato que introduzcas despues de la palabra return no será leído ya que el trabajo de la función terminó

- If statements

Permite que un bloque de código se ejecute si cumple una condición

```
#include <iostream>

using namespace std;

int main()
{
  bool isMale = true;
  if(isMale){
      cout << "You are a male";
  }
  else {
      cout << "You are not"
  }
  cout << cube(5.0);

  return 0;
}
```

Output:

```
You are a male
```

De lo contrario:

```
#include <iostream>

using namespace std;

int main()
{
  bool isMale = false;
  if(isMale){
      cout << "You are a male";
  }
  else {
      cout << "You are not"
  }
  cout << cube(5.0);

  return 0;
}
```

Output:

```
You are not
```

También puedes usar los operadores lógicos con esto
&& y
|| o
! negación

- Switch

Utilice la sentencia switch para seleccionar uno de los muchos bloques de código que se van a ejecutar. El funcionamiento es el siguiente:

- La expresión switch se evalúa una vez
- El valor de la expresión se compara con los valores de cada caso
- Si hay una coincidencia, se ejecuta el bloque de código asociado
- Las palabras clave break y default son opcionales

El ejemplo siguiente utiliza el número del día de la semana para calcular el nombre del día de la semana:

```
#include <iostream>

using namespace std;
int main(){
  int day = 4;
  switch (day) {
    case 1:
      cout << "Monday";
      break;
    case 2:
      cout << "Tuesday";
      break;
    case 3:
      cout << "Wednesday";
      break;
    case 4:
      cout << "Thursday";
      break;
    case 5:
      cout << "Friday";
      break;
    case 6:
      cout << "Saturday";
      break;
    case 7:
      cout << "Sunday";
      break;
  }
}
```

Output:

```
Thursday
```

La palabra clave break Cuando C++ alcanza una palabra clave break, rompe el bloque switch. Esto detendrá la ejecución de más código y pruebas de caso dentro del bloque.
Cuando se encuentra una coincidencia, y el trabajo está hecho, es hora de un break. No hay necesidad de más pruebas.

- Loops

  Los bucles pueden ejecutar un bloque de código mientras se cumpla una condición especificada.
  Los bucles son útiles porque ahorran tiempo, reducen los errores y hacen que el código sea más legible.

  - **while loop** El bucle while repite un bloque de código mientras se cumpla una condición especifíca:

    ```
    ```

```
#include <iostream>

using namespace std;

int main()
{
int i = 0;
while(i < 5){
cout << i << "\n";
i++;
}
return 0;
}
```

Output:

```
0
1
2
3
4
```

- **do while** el bucle do/while es una variante del bucle while.
  Este bucle ejecutará el bloque de código una vez, antes de comprobar si la condición es verdadera, luego repetirá el bucle mientras la condición sea verdadera.

```
#include <iostream>

using namespace std;

int main()
{
int i = 0;
do {
cout << i << "\n";
i++;
}
while (i < 5);
return 0;
}
```

Output:

```
0
1
2
```

```
3
4
```

- **For** Si sabes exactamente cuántas veces deseas repetir un bloque de código, utiliza el bucle for en lugar del bucle while:

```
#include <iostream>

using namespace std;

int main()
{
for (int i = 0; i < 5; i++) {
cout << i << "\n";
}
return 0;
}
```

Output:

```
0
1
2
3
4
```

La primera parte establece una variable antes de que comience el bucle (int i = 0).

Después se define la condición para que se ejecute el bucle (i debe ser menor que 5). Si la condición es verdadera, el bucle comenzará de nuevo, si es falsa, el bucle terminará.

Al fnal incrementa un valor (i++) cada vez que se ejecuta el bloque de código del bucle.

Loop con array

```
#include <iostream>

using namespace std;

int main()
{
int myNumbers[5] = {10, 20, 30, 40, 50};
for (int i : myNumbers) {
cout << i << "\n";
}
return 0;
}
```

Output:

```
10
20
30
40
50
```

- 2D Arrays & Loops anidados

Generalmente conocidos como matrices

```
#include <iostream>

using namespace std;
int main()
{
int numberGrid[3][2] = {
                            {1, 2},
                            {3, 4},
                            {5, 6}
};
for (int i = 0; i < 3; i++){
    for(int j= 0; j < 2; j++){
        cout << numberGrid [i][j];
}
cout << endl;
}
return 0;
}
```

Output:

```
12
34
56
```

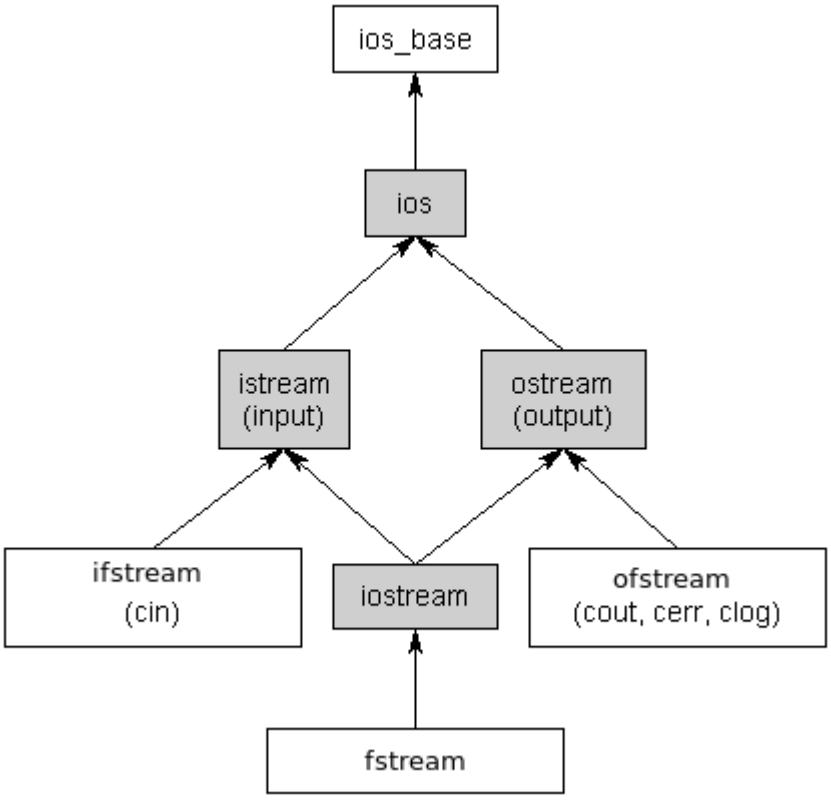
Se encarga de indexar dos arrays.

- Archivos

Se agrega la librería "fstream", se utiliza ifstream para leer datos y ofstream para sobreescribirlos.

```
#include <iostream>
#include <cstdlib>  //libreria para crear variables globales
#include <fstream>

using namespace std;

string linea;
string texto;

int main()
{
  ifstream archivo("test.txt");

  while(getline(archivo, linea)){  //leer todas las líneas al guardarse en
una variable
    texto = texto+linea+"\n";
  }
  archivo.close();  //eliminar memoria residual de abrir
  cout << texto << endl;

  ofstream archivo2("test.txt");  //crea documento y sobreescribe
  archivo2 << texto <<"Nueva linea de Texto";

  system("pause"); //para seguir trabajando
}
```

Explicación de las funciones cin, cout, cerr.



Página para saber más

- Pointers

Se llaman así a los tipos de datos que contienen la direccion de la variable en la memoria
&nombre_de_la_variable

```
#include <iostream>

using namespace std;
int main()
{
  int age = 19;
  double gpa = 2.7;
  string name = "Mike";

  cout << &age << endl;
  cout << &gpa << endl;
  cout << &name << endl;
  return 0;
}
```

Output:

```
0xe2773ff628
0xe2773ff620
0xe2773ff600
```

> Estas son las direcciones en la memoria de mis variables.

También se puede usar la dirección como un dato e introducirlo en una variable

```
#include <iostream>

using namespace std;
int main()
{
  int age = 19;
  int *pAge = &age;
  double gpa = 2.7;
  double *pGpa = &gpa;
  string name = "Mike";
  string *pName = &name;

  cout << pAge << endl;
  cout << pGpa << endl;
  cout << pName << endl;
  return 0;
}
```

Output:

```
0xe2773ff628
0xe2773ff620
0xe2773ff600
```

Y juguemos con los punteros usando el valor al que apuntan y obteniendo el valor que contienen las variables.

```
#include <iostream>

using namespace std;
int main()
{
  int age = 19;
  int *pAge = &age;
  double gpa = 2.7;
  double *pGpa = &gpa;
  string name = "Mike";
  string *pName = &name;

  cout << *pAge << endl;
  cout << *pGpa << endl;
  cout << *pName << endl;
  return 0;
}
```

Output:

```
19
2.7
Mike
```

- Clases y Objetos

POO son las siglas de Programación Orientada a Objetos.

La programación procedimental consiste en escribir procedimientos o funciones que realicen operaciones sobre los datos, mientras que la programación orientada a objetos consiste en crear objetos que contengan tanto datos como funciones.

La programación orientada a objetos tiene varias ventajas sobre la programación procedimental:

La programación orientada a objetos es más rápida y fácil de ejecutar.
La programación orientada a objetos proporciona una estructura clara a los programas.
La programación orientada a objetos ayuda a mantener el código C++ DRY "Don't Repeat Yourself" (no

te repitas), y hace que el código sea más fácil de mantener, modificar y depurar.
La programación orientada a objetos permite crear aplicaciones totalmente reutilizables con menos código y menos tiempo de desarrollo.

¿Y las clases..?

Una clase es una plantilla para objetos, y un objeto es una instancia de una clase. Cuando se crean los objetos individuales, se heredan todas las variables y funciones de la clase.

```
#include <iostream>

using namespace std;
class Book {
  public:
      string title;
      string author;
      int pages;
};
int main()
{
  Book book1;
  book1.title = "Harry Potter" << endl;
  book1.author = "JK Rowling" << endl;
  book1.pages = 500 << endl;

  cout << book1.author;
  cout << book1.pages;
  cout << book1.title;

  Book book2;
  book2.title = "Emma";
  book2.author = "Jane Austen";
  book2.pages = 400;

  return 0;
}
```

Output:

```
JK Rowling
500
Harry Potter
```

> Se podría decir que la clase es la plantilla para los datos, mientras que el objeto contiene valores que entrarán en esa plantilla

- Constructors Functions

Work like that:

```
#include <iostream>

using namespace std;
class Book {
  public:
      string title;
      string author;
      int pages;

      Book(string aTitle, string aAuthor, int aPages){
          title = aTitle;
          author = aAuthor;
          pages = aPages;

      }


};
int main()
{
  Book book1("Harry Potter", "JK Rowling", 500);
  Book book2("Emma", "Jane Austen", 400);

  cout << book1.title;

   return 0;
}
```

Output:

```
Harry Potter
```

- Objects Functions

```
#include <iostream>

using namespace std;
class Book {
  public:
      string title;
      string author;
      int pages;

      Book(string aTitle, string aAuthor, int aPages){
          title = aTitle;
          author = aAuthor;
          pages = aPages;
```

```
        }
        Book(){
            title = "no title";
            author = "no author";
            pages = 0;
        }
        bool easytoread(){
            if(pages < 600){
                return true;
            }
            return false;
        }
};

int main()
{
  Book book1("Harry Potter", "JK Rowling", 500);
  Book book2("Emma", "Jane Austen", 400);
  Book book3;
  cout << book1.easytoread() << endl;
  cout << book2.easytoread();

   return 0;
}
```

Output:

```
1
1
```

Porque la condición es cierta.

- Getter and Setters

Permitir que tu código sea modificado o no usando private en la Clase y controlar el acceso a elementos individuales

```
#include <iostream>
using namespace std;

class Movie {
  private:
      string rating;
  public:
      string title;
      string director;

      Movie(string aTitle, string aDirector, string aRating){
          title = aTitle;
```

```
          director = aDirector;
          setRating(aRating);
      }

      void setRating(string aRating){
          if(aRating == "G" || aRating == "PG" || aRating == "PG-13" ||
aRating == "R")
             rating = aRating;
          else {
              rating = "NR";
          }
      }

      string getRating(){
          return rating;
      }
};

int main()
{
  Movie avengers("The Avengers", "Joss Whedon", "PG-13");
  avengers.setRating("Dog");
  cout << avengers.getRating();
}
```

Output:

```
NR
```

- Inheritance -Herencia

Obtener datos públicos de otra clase, aumentar el numero de funcionalidades o sobreescribir funciones.

SuperClass Aquella de la que se hereda

SubClass La que recibe los datos

```
#include <iostream>
using namespace std;

class Chef{
public:
  void makeChicken(){
      cout << "The chef makes yummy chicken\n";
  }
   void makeSalad(){
      cout << "The chef makes salad\n";
  }
```

```
    void makeSpecialDish(){
        cout << "The chef makes bbq ribs\n";
    }
};

class italianChef : public Chef{
    public:
    void makePasta(){
        cout << "The chef makes pasta\n";
    }
    void makeSpecialDish(){
        cout << "The chef makes chicken parm\n";
    }
};

int main()
{
    Chef chef;
    chef.makeSpecialDish();

    italianChef italianchef;
    italianchef.makeSpecialDish();

    return 0;
}
```

Output

```
The chef makes bbq ribs
The chef makes chicken parm
```

- Recursos

  Giraffe Academy