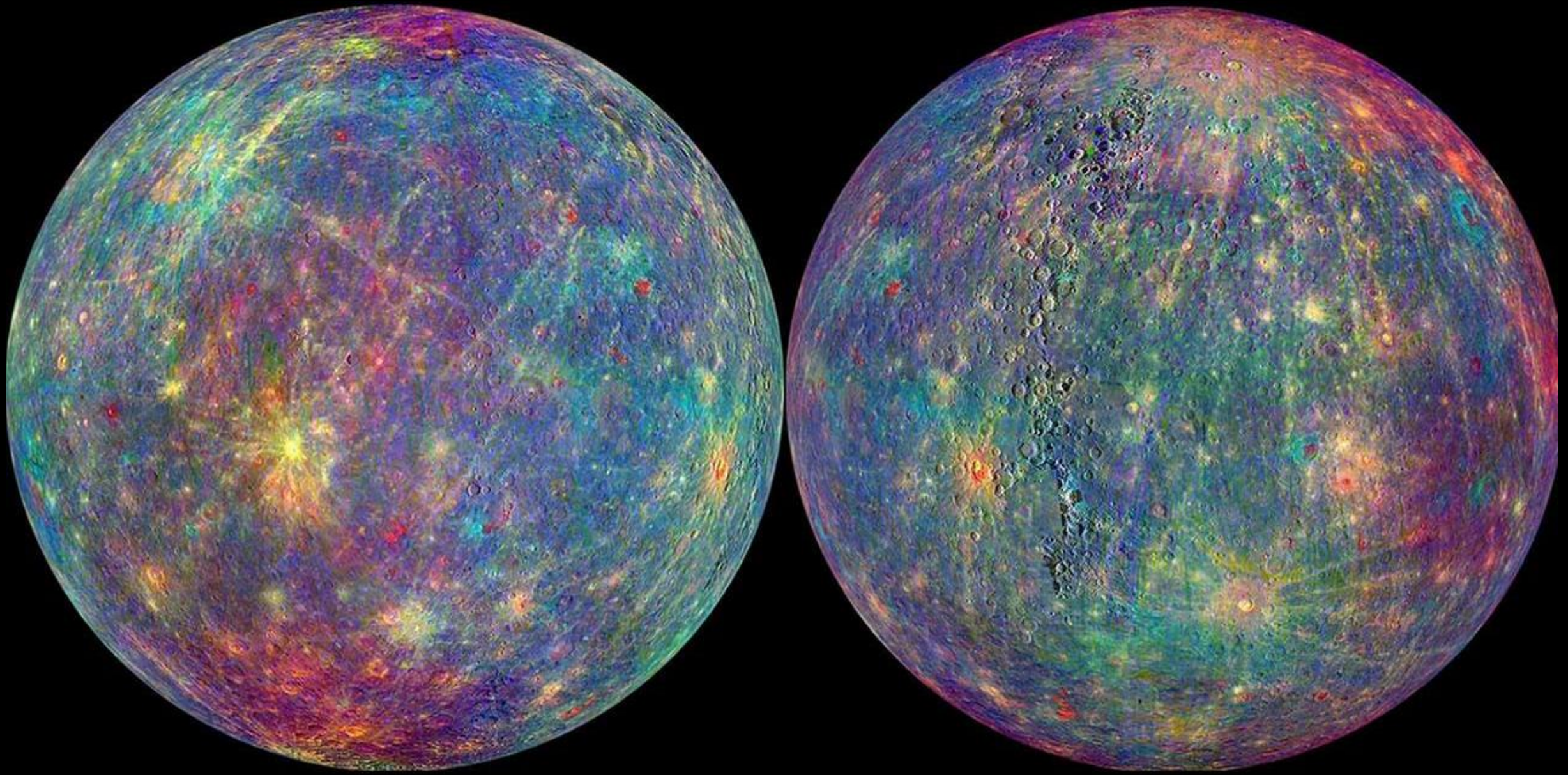


Introducción a C++

Mauricio Avilés

Contenido

- C++
- Char
- Int
- Enum
- Float
- Punteros
- Arreglos
- Strings

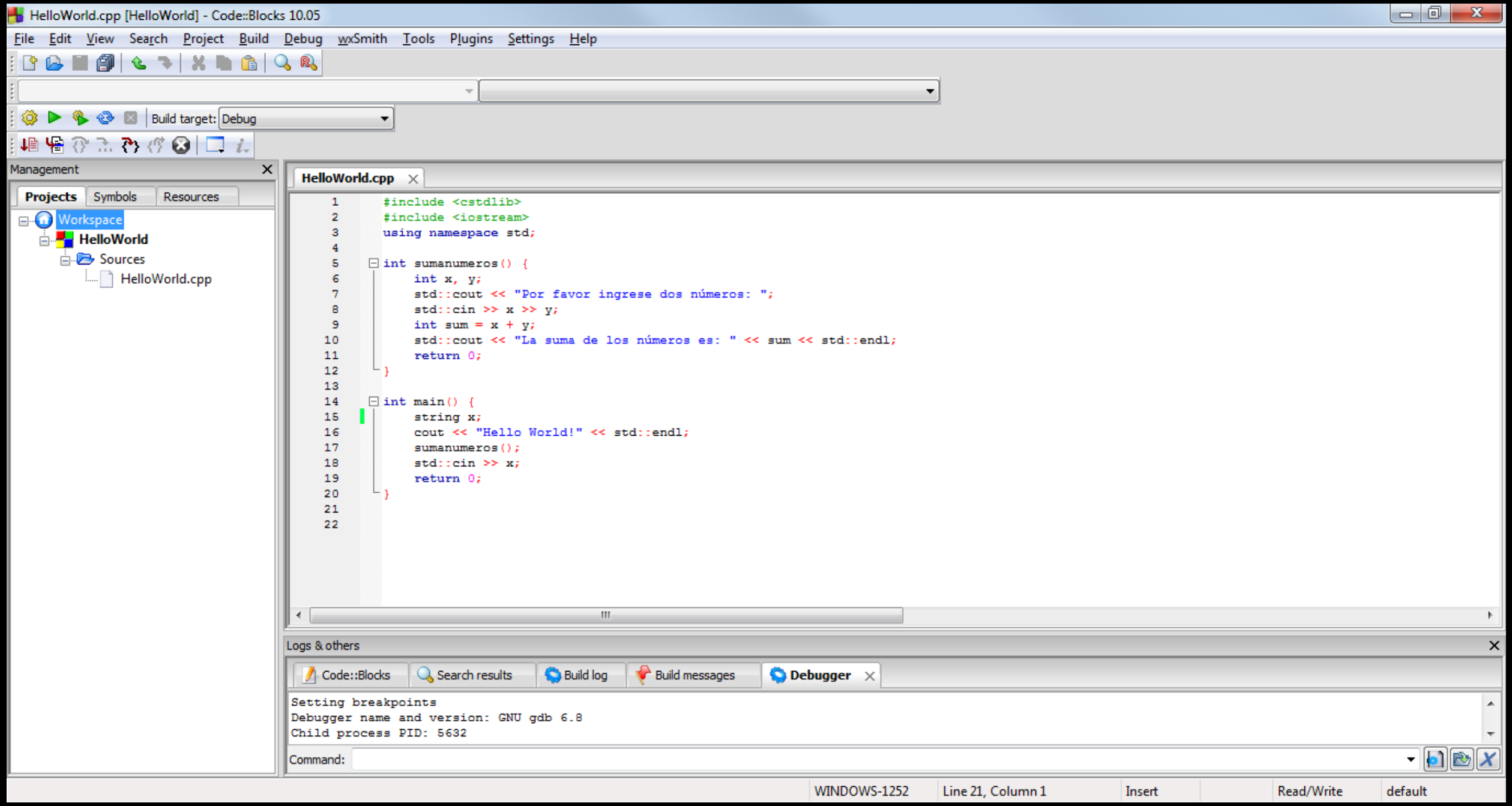
- Struct
- Const
- Typedef
- Namespace y using
- Expresiones
- Estructuras de control
- Funciones
- Clases

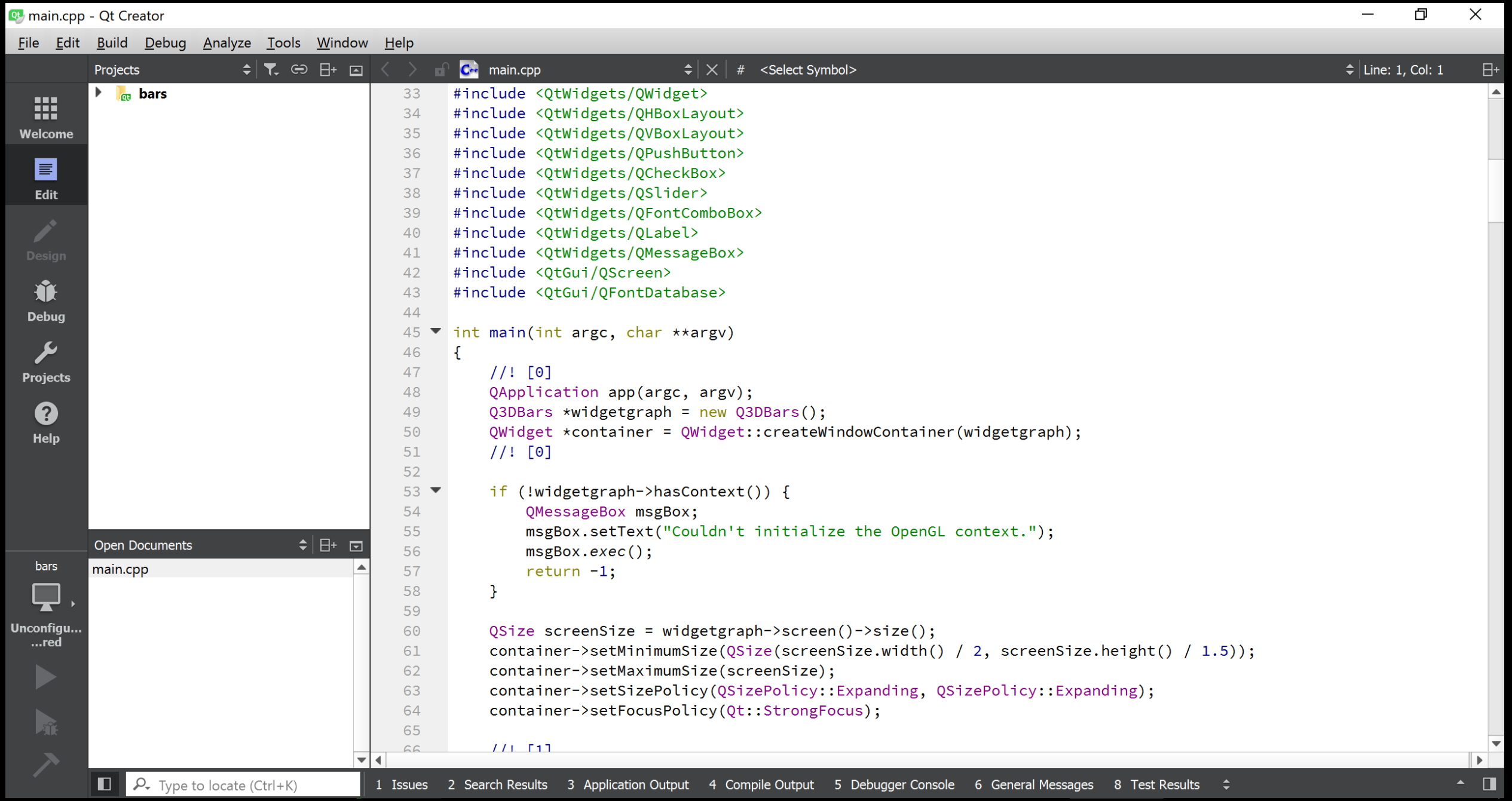
Lectura

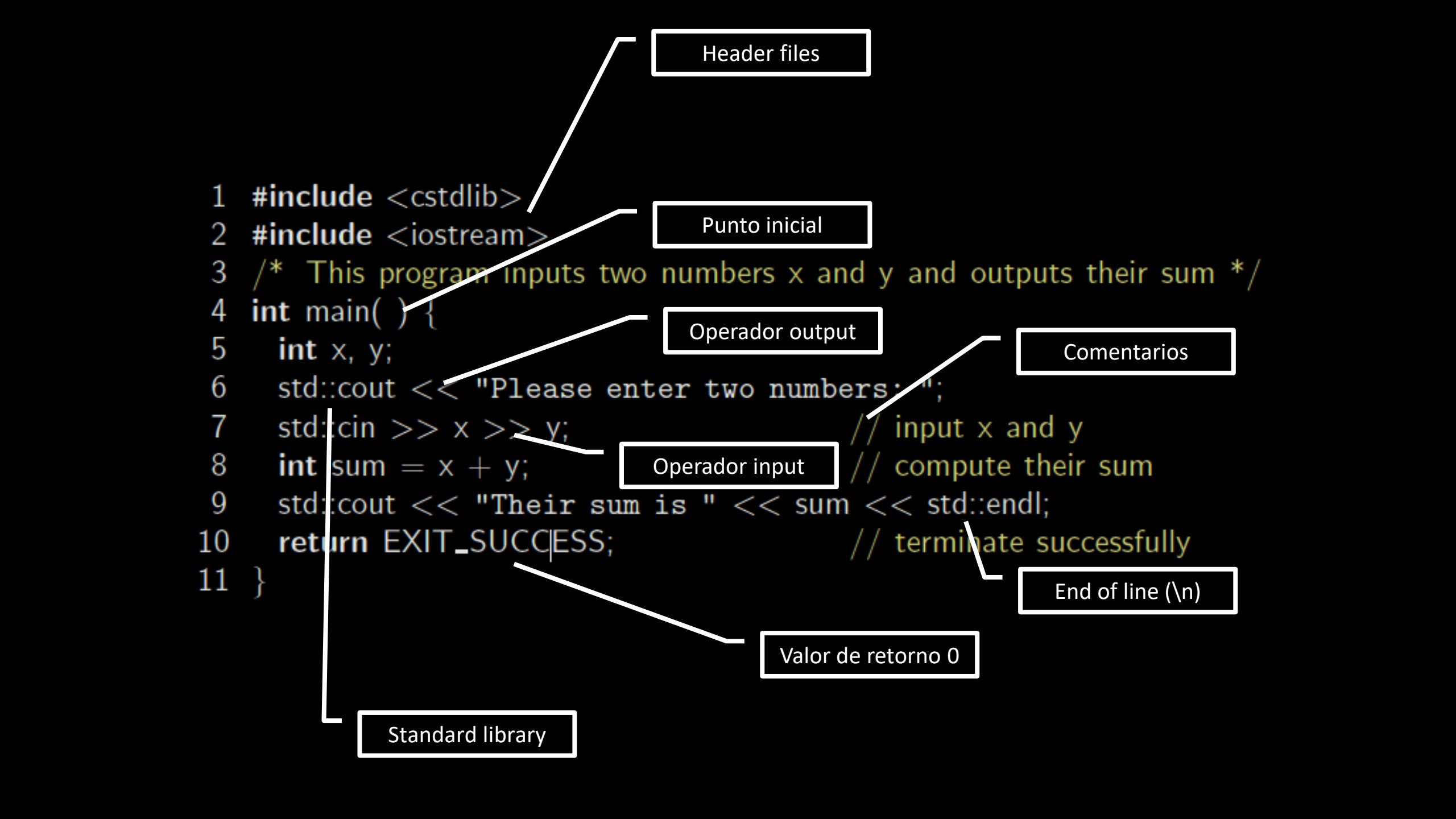
- Capítulos 2, 3, 4 y 5
  - Deitel, H. M., Deitel, P. J., & Elizondo, A. V. (2009). *C++: cómo programar* (6a ed.). México: Pearson Educación.
- Capítulo 1
  - Goodrich, M. T., Tamassia, R., & Mount, D. M. "Data structures and algorithms in C++" (2nd ed., Wiley). Hoboken, NJ: Wiley. 2011.
- Capítulos 2, 3
  - Joyanes, Aguilar, & Martínez. (2007). Estructura de datos en C ++. Madrid: McGraw-Hill Interamericana.

C++

- Está basado en C, es un superconjunto de ese lenguaje
  - Todo lo que compila en C compila en C++, pero no viceversa
- Maneja eficientemente el hardware
- Paradigma de orientación a objetos
  - Maneja la noción de clases, objetos, herencia
- Lenguaje tipado
- Manejo directo de la memoria dinámica
- No tiene garbage collector


HelloWorld.cpp [HelloWorld] - Code::Blocks 10.05
File Edit View Search Project Build Debug wxSmith Tools Plugins Settings Help
Build target: Debug
Management
Projects Symbols Resources
Workspace
HelloWorld
Sources
HelloWorld.cpp
HelloWorld.cpp
#include <cstdlib>
#include <iostream>
using namespace std;
int sumanumeros() {
int x, y;
std::cout << "Por favor ingrese dos números: ";
std::cin >> x >> y;
int sum = x + y;
std::cout << "La suma de los números es: " << sum << std::endl;
return 0;
}
int main() {
string x;
cout << "Hello World!" << std::endl;
sumanumeros();
std::cin >> x;
return 0;
}
Logs & others
Code::Blocks
Search results
Build log
Build messages
Debugger
Setting breakpoints
Debugger name and version: GNU gdb 6.8
Child process PID: 5632
Command:
WINDOWS-1252
Line 21, Column 1
Insert
Read/Write
default


main.cpp - Qt Creator
File Edit Build Debug Analyze Tools Window Help
Projects
bars
Welcome
Edit
Design
Debug
Projects
Help
Open Documents
main.cpp
bars
Unconfigu...
...red
main.cpp
<Select Symbol>
Line: 1, Col: 1
33 #include <QtWidgets/QWidget>
34 #include <QtWidgets/QHBoxLayout>
35 #include <QtWidgets/QVBoxLayout>
36 #include <QtWidgets/QPushButton>
37 #include <QtWidgets/QCheckBox>
38 #include <QtWidgets/QSlider>
39 #include <QtWidgets/QFontComboBox>
40 #include <QtWidgets/QLabel>
41 #include <QtWidgets/QMessageBox>
42 #include <QtGui/QScreen>
43 #include <QtGui/QFontDatabase>
44
45 int main(int argc, char **argv)
46 {
47 //! [0]
48 QApplication app(argc, argv);
49 Q3DBars *widgetgraph = new Q3DBars();
50 QWidget *container = QWidget::createWindowContainer(widgetgraph);
51 //! [0]
52
53 if (!widgetgraph->hasContext()) {
54 QMessageBox msgBox;
55 msgBox.setText("Couldn't initialize the OpenGL context.");
56 msgBox.exec();
57 return -1;
58 }
59
60 QSize screenSize = widgetgraph->screen()->size();
61 container->setMinimumSize(QSize(screenSize.width() / 2, screenSize.height() / 1.5));
62 container->setMaximumSize(screenSize);
63 container->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);
64 container->setFocusPolicy(Qt::StrongFocus);
65
Type to locate (Ctrl+K)
1 Issues 2 Search Results 3 Application Output 4 Compile Output 5 Debugger Console 6 General Messages 8 Test Results




```
1  #include <cstdlib>
2  #include <iostream>
3  /* This program inputs two numbers x and y and outputs their sum */
4  int main( ){
5    int x, y;
6    std::cout << "Please enter two numbers: ";
7    std::cin >> x >> y;                          // input x and y
8    int sum = x + y;                             // compute their sum
9    std::cout << "Their sum is " << sum << std::endl;
10   return EXIT_SUCCESS;                         // terminate successfully
11 }
```

bool	Valor booleano, verdadero o falso	1
char	un carácter	1
short	entero corto	2
int	entero	4
long	entero largo	4
float	número de punto flotante	4
double	número de punto flotante con doble precisión	8
void	ausencia de información	1

char

- Un solo carácter
- Generalmente 8 bits
- Caracteres especiales:
  - \n nueva línea
  - \b backspace
  - \' '
  - \\ \
  - \t tab
  - \0 null
  - \" "
- El caracter null es utilizado para indicar el final de un string
- int(c) retorna el valor asociado a un caracter

int

- Tres tamaños:
  - short
  - int
  - long
- En una literal se usa el sufijo L para indicar que es long
  - 123456789L
- Literal en octal 0400 es 256 en decimal
- Literal en hexa 0x100 es 256 en decimal

- Las variables se declaran con su tipo
- Se puede asignar un valor inicial

```
short n;                                    // n's value is undefined
int   octalNumber = 0400;                   // 400 (base 8) = 256 (base 10)
char  newline_character = '\n';
long  BIGnumber = 314159265L;
short _aSTRANGE__1234_variABlE_NaMe;
```

enum

- Enumeración: tipo de dato definido por el usuario
- Valores discretos
- Se comportan como un tipo entero
- Dar nombres significativos a diferentes valores
- Por defecto empiezan en cero, pero se puede especificar

```
enum Day { SUN, MON, TUE, WED, THU, FRI, SAT };
enum Mood { HAPPY = 3, SAD = 1, ANXIOUS = 4, SLEEPY = 2 };

Day today = THU;          // today may be any of MON ... SAT
Mood myMood = SLEEPY;     // myMood may be HAPPY, ..., SLEEPY
```

float

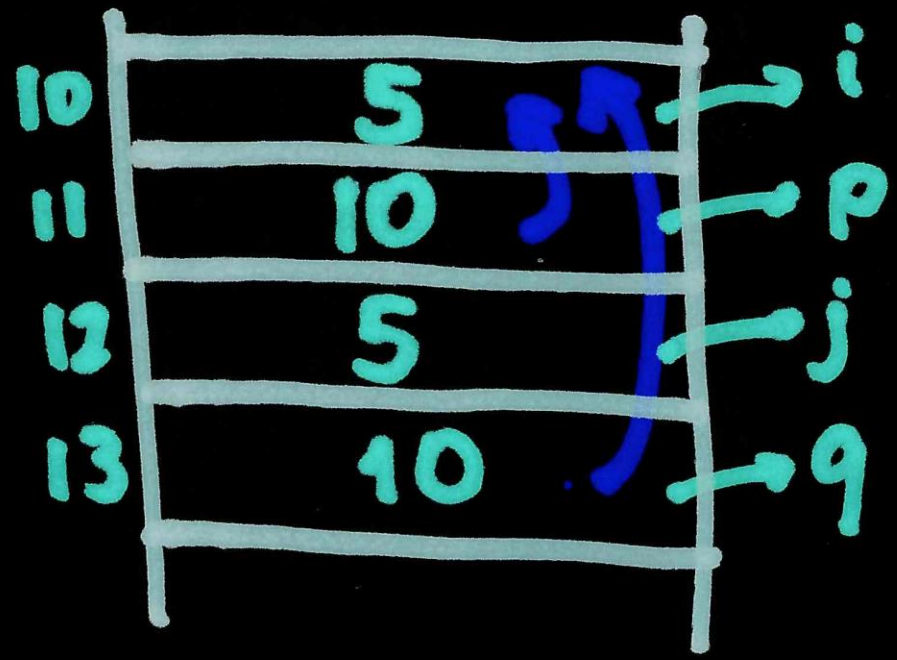
- Número de punto flotante
- Las literales como 3.14159 y -1234.567 son double
- Notación exponencial: 3.14E5 → 3.14×10^5
- Literal tipo float: 2.0F ó 1.234E-3F

Punteros

- Todas las variables se almacenan en alguna dirección de memoria
- Un puntero es una variable que guarda esa dirección
- Dado un tipo T, el tipo T* es un puntero a una variable de tipo T
- int* denota un puntero a un entero

- Operadores esenciales:
  - &: retorna la dirección de un objeto en memoria (dirección-de)
  - *: acceder el contenido de un objeto a través de su dirección

```
int i = 5;
int* p = &i;
int j = *p;
int* q = p;
```



```
char ch = 'Q';
char* p = &ch;                  // p holds the address of ch
cout << *p;                     // outputs the character 'Q'
ch = 'Z';                       // ch now holds 'Z'
cout << *p;                     // outputs the character 'Z'
*p = 'X';                       // ch now holds 'X'
cout << ch;                     // outputs the character 'X'
```

- Los punteros son importantes porque son utilizados para construir estructuras de datos donde un elemento puede apuntar hacia otro elemento
- No sólo son para apuntar a tipos básicos, si no también a tipos de usuario e incluso funciones

- Un puntero que no apunta a nada es un puntero nulo
- El valor nulo se representa con un 0
- Constante NULL

```
#include <cstdlib>
…
int *p = NULL;
```

Memoria dinámica

- El propósito principal de los punteros es poder utilizar memoria dinámica
- Los programas tienen a su disposición un espacio de memoria manejado por el sistema operativo llamado *heap*
- Esta memoria se asigna dinámicamente a los programas mientras se ejecutan, a diferencia de la memoria asignada de forma estática al cargar el programa para ejecutarlo

- Para solicitar memoria dinámica se utiliza la palabra reservada *new*
- Este operador retorna la dirección de memoria donde se encuentra el objeto solicitado
- Esta dirección se asigna a un puntero del mismo tipo para poder referenciar al objeto
- Para retornar memoria dinámica se utiliza la palabra reservada *delete*
- Se utiliza junto con el nombre del puntero
- Regresa la memoria al *heap*
- A partir de ese momento esa memoria se encuentra libre y puede ser asignada para otro propósito

```
int *p = new int;
*p = 25;
cout << *p << endl;
int *q = p;
*q = *q * 10;
cout << *q << endl;
delete p;
p = NULL;
q = NULL;
```

El operador new reserva memoria para un entero y retorna la dirección del espacio. Esta dirección se asigna al puntero p.

```
int *p = new int;
*p = 25;
cout << *p << endl;
int *q = p;
*q = *q * 10;
cout << *q << endl;
delete p;
p = NULL;
q = NULL;
```

Al espacio de memoria solicitado se le asigna el valor 25.

```
int *p = new int;
*p = 25;
cout << *p << endl;
int *q = p;
*q = *q * 10;
cout << *q << endl;
delete p;
p = NULL;
q = NULL;
```

Se imprime en pantalla el contenido de la memoria apuntada por p. Aparece el número 25 en consola.

```
int *p = new int;
*p = 25;
cout << *p << endl;
int *q = p;
*q = *q * 10;
cout << *q << endl;
delete p;
p = NULL;
q = NULL;
```

Se crea un nuevo puntero q que apunta a la misma ubicación que apunta p.

```
int *p = new int;
*p = 25;
cout << *p << endl;
int *q = p;
*q = *q * 10;
cout << *q << endl;
delete p;
p = NULL;
q = NULL;
```

El entero apuntado por q es multiplicado por 10 y es almacenado en la misma ubicación de memoria.

```
int *p = new int;
*p = 25;
cout << *p << endl;
int *q = p;
*q = *q * 10;
cout << *q << endl;
delete p;
p = NULL;
q = NULL;
```

Se imprime en pantalla el contenido de la memoria apuntada por q.
Dado que la memoria apuntada por p es la misma que la de q, el contenido apuntado por ambos punteros es el mismo.

```
int *p = new int;
*p = 25;
cout << *p << endl;
int *q = p;
*q = *q * 10;
cout << *q << endl;
delete p;
p = NULL;
q = NULL;
```

Cuando la memoria solicitada ya no va a ser necesitada por el programa, debe retornarse al heap.

```
int *p = new int;
*p = 25;
cout << *p << endl;
int *q = p;
*q = *q * 10;
cout << *q << endl;
delete p;
p = NULL;
q = NULL;
```

Una vez que la memoria ha sido retornada, la dirección apuntada ya no tiene sentido, por lo que es una buena práctica de programación sustituirla por el valor NULL.

Arreglos

- Colección de elementos del mismo tipo
- Una variable de tipo T[N] es un arreglo de N elementos del tipo T
- Los elementos se acceden por índices
- De 0 hasta N-1
- Una vez creados no pueden cambiar su tamaño
- C++ **no** lanza errores para índices fuera de los límites

```
double f[5];              // array of 5 doubles: f[0], ..., f[4]
int m[10];                // array of 10 ints: m[0], ..., m[9]
f[4] = 2.5;
m[2] = 4;
cout << f[m[2]];          // outputs f[4], which is 2.5
```

- Arreglos multidimensionales

  int a[10][30]

- Los arreglos se puede inicializar

  ```
  int a[] = { 2, 3, 5, 7, 11, 13 };  // a[6]
  bool b[] = { false, true, false, true}; // b[4]
  char c[] = {‘O’, ‘M’, ‘G’} ;// c[3]
  ```

- Arreglo de punteros

  ```
  int *r[10];
  …
  int a = *r[2];
  ```

Relación entre punteros y arreglos

- El nombre del arreglo es equivalente a un puntero al primer elemento
- Un puntero al primer elemento es equivalente al nombre del arreglo

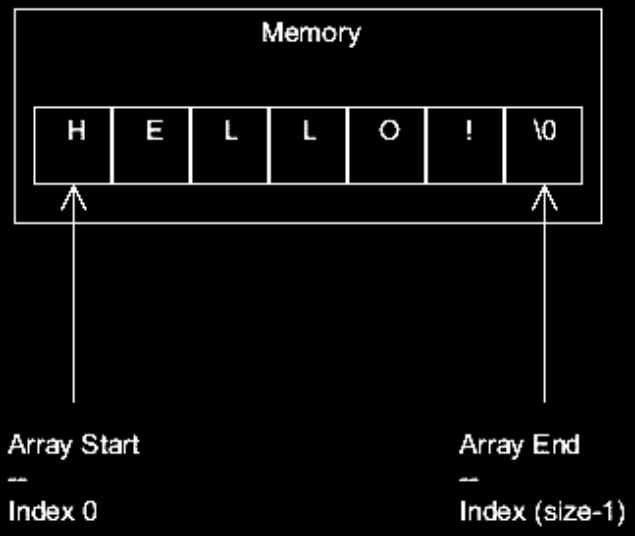
```
char c[] = {'c', 'a', 't'};
char* p = c;                    // p points to c[0]
char* q = &c[0];                // q also points to c[0]
cout << c[2] << p[2] << q[2];   // outputs "ttt"
```

Arreglo en memoria dinámica

- Capítulo 6 de ejercicios creativos

string

- En C++ los strings son estilo C
- Arreglos de largo fijo que terminan con el carácter null
- No permite muchas operaciones con strings



STL string

- Tipo de string que permite hacer operaciones con strings

```
#include <string>
using std::string;
// ...
string s = "to be";
string t = "not " + s;                  // t = “not to be”
string u = s + " or " + t;              // u = “to be or not to be”
if (s > t)                              // true: “to be” > “not to be”
  cout << u;                            // outputs “to be or not to be”
```

struct

- Estructuras de C
- Unificación de varios elementos bajo un mismo nombre
- Los elementos pueden ser de diferentes tipos
- Cada miembro o campo tiene un nombre
- Los campos del struct se acceden por medio del punto

```
enum MealType { NO_PREF, REGULAR, LOW_FAT, VEGETARIAN };

struct Passenger {
    string      name;           // passenger name
    MealType    mealPref;       // meal preference
    bool        isFreqFlyer;    // in the frequent flyer program?
    string      freqFlyerNo;    // the passenger's freq. flyer number
};

Passenger pass = { "John Smith", VEGETARIAN, true, "293145" };

pass.name = "Pocahontas";       // change name
pass.mealPref = REGULAR;        // change meal preference
```

Memoria dinámica

- Crear estructuras según se necesite
- C++ tiene un bloque de memoria (*free store*) para este propósito
- El operador new reserva el espacio necesario y retorna un puntero

```
(*puntero).miembro
```

es lo mismo que

```
puntero->miembro
```

```
Passenger *p;
// ...
p = new Passenger;                    // p points to the new Passenger
p->name = "Pocahontas";               // set the structure members
p->mealPref = REGULAR;
p->isFreqFlyer = false;
p->freqFlyerNo = "NONE";
```

- El registro existe hasta que se borre explícitamente

```
delete p;
```

- Libera la memoria que había sido reservada
- C++ no tiene *garbage collection* por lo que la responsabilidad del manejo de memoria queda en manos del programador

- Si no se libera la memoria adecuadamente, se produce una fuga de memoria (*memory leak*)
- El programa se puede quedar sin memoria
- Liberando memoria de un puntero a un arreglo

```
char* buffer = new char[500];        // allocate a buffer of 500 chars
buffer[3] = 'a';                     // elements are still accessed using []
delete [] buffer;                    // delete the buffer
```

const

- Un valor constante que no se puede cambiar
- Se puede usar en casi cualquier literal

```
const double PI          = 3.14159265;
const int    CUT_OFF[]   = {90, 80, 70, 60};
const int    N_DAYS      = 7;
const int    N_HOURS     = 24*N_DAYS; // using a constant expression
int counter[N_HOURS];                 // an array of 168 ints
```

typedef

- Asocia un identificador con un tipo
- Darle nombres significativos a los diferentes tipos
- Si se cambia la definición del tipo, se hace en un solo lugar del código

```
typedef char* BufferPtr;          // type BufferPtr is a pointer to char
typedef double Coordinate;        // type Coordinate is a double

BufferPtr p;                      // p is a pointer to char
Coordinate x, y;                  // x and y are of type double
```

Alcance local y global

- Las sentencias C++ que se escriben entre llaves ({…}) definen un bloque
- Las variables que se definen en ese bloque sólo son visibles en el mismo, son locales
- Si se declaran fuera de cualquier bloque son variables globales
- Las partes del programa donde un nombre es accesible se llama alcance (*scope*)

```
const int Cat = 1;                    // global Cat

int main() {
  const int Cat = 2;                  // this Cat is local to main
  cout << Cat;                        // outputs 2 (local Cat)
  return EXIT_SUCCESS;
}

int dog = Cat;                        // dog = 1 (from the global Cat)
```

namespace y using

- Mecanismo que permite definir un grupo de nombres en un mismo lugar
- Organizar objetos en grupos
- Minimizar problemas de usar globales
- Se pueden acceder mediante: grupo::nombre
- La palabra using hace que todos los nombres dentro de un namespace estén disponibles

Expresiones

- Combinan variables y literales con operadores para crear nuevos valores

Acceso a miembros en indización

class_name.member

pointer->member

array[exp]

Operadores aritméticos

exp + exp	suma
exp – exp	resta
exp * exp	multiplicación
exp / exp	división
exp % exp	módulo
-exp	negativo
+exp	positivo

Incremento y decremento

var++	incremento posterior
var--	decremento posterior
++var	incremento previo
--var	decremento previo

```
int a[] = {0, 1, 2, 3};
int i = 2;
int j = i++;
int k = --i;
cout << a[k++];
```

Operadores relacionales y lógicos

exp < exp	menor que
exp > exp	mayor que
exp <= exp	menor o igual que
exp >= exp	mayor o igual que
exp == exp	igual
exp != exp	diferente de
!exp	negación
exp && exp	and lógico
exp || exp	or lógico

números, caracteres, STL strings, punteros

evalúan de izq a der

Operaciones de bits

~exp	complemento
exp & exp	and
exp ^ exp	xor
exp | exp	or
exp1 << exp2	corrimiento a la izquierda
exp1 >> exp2	corrimiento a la derecha

rellena con ceros

rellena con signo

Operadores de asignación

var = exp asignación

+=, -=, *=, /=, %=

&=, |=, ^=, <<=, >>=

Expresión condicional

bool_exp? true_exp : false_exp

menor = x<y? x:y;

Es como un pequeño if funcional

Tipo	Operadores
Resolución de alcance	namespace_name::member
Acceso Indización Llamada a función Operadores posfijos	class_name.member pointer->member array[exp] function(args) var++ var--
Operadores prefijos Operadores de punteros	++var –var +exp –exp ~exp !exp *pointer &var
Multiplicación, división	* / %
Suma, resta	+ -
Corrimiento	<< >>
Comparación	< <= > >=
Igualdad	== !=
And de bits	&
Xor de bits	^
Or de bits	|
And lógico	&&
Or lógico	||
Condicional	bool_exp? true_exp : false_exp
Asignación	= += -= *= /= …

Casting

- Cambiar el tipo de una variable o expresión
- Tradicional
  - Estilo C (T)exp

```
double d = 3.5;
int i = (int)d;
```

  - Estilo funcional T(exp)

```
double d = 3.5;
int i = int(d);
```

- Estático static_cast<T>(exp)

```
double d = 3.5;
int i = static_cast<int>(d);
```

- Implícito

```
Int i = 3;
double d = i;
```

Estructuras de control

- Estructuras similares a las de otros lenguajes de alto nivel

Agrupar sentencias

```
{
        statement1;
        statement2;
        …
}
```

if

```
if ( condition )
        true_statement
else if ( condition )
        else_if_statement
else
        else_statement
```

if

```
if (a==0) {
        cout << “Es cero”;
}
else if (a < 5) {
        cout << “Es menor que cinco”;
} else {
        cout << “Es mayor que cinco”
}
```

switch

```
switch ( exp ) {
        case constant-exp 1:
                statements
        case constant-exp2:
                statements
        default:
                statements;
}
```

switch

```
switch (dia) {
        case “Lunes”:
                cout << “Energía 100%”;
                break;
        case “Martes”:
                cout << “Energía 75%”;
                break;
        case “Miércoles”:
                cout << “Energía 40%”;
                break;
        case “Jueves”:
        case “Viernes”:
                cout << “Ahorro de energía”;
                break;
        default:
                cout << “Recargando”;
                break;
}
```

while

```
while ( condition )
        loop_body_statement
```

while

```
int a[100];
//…
int i = 0;
int sum = 0;
while (i < 100 && a[i] >= 0) {
        sum += a[i++];
}
```

do

```
do {
        statements
} while (condition)
```

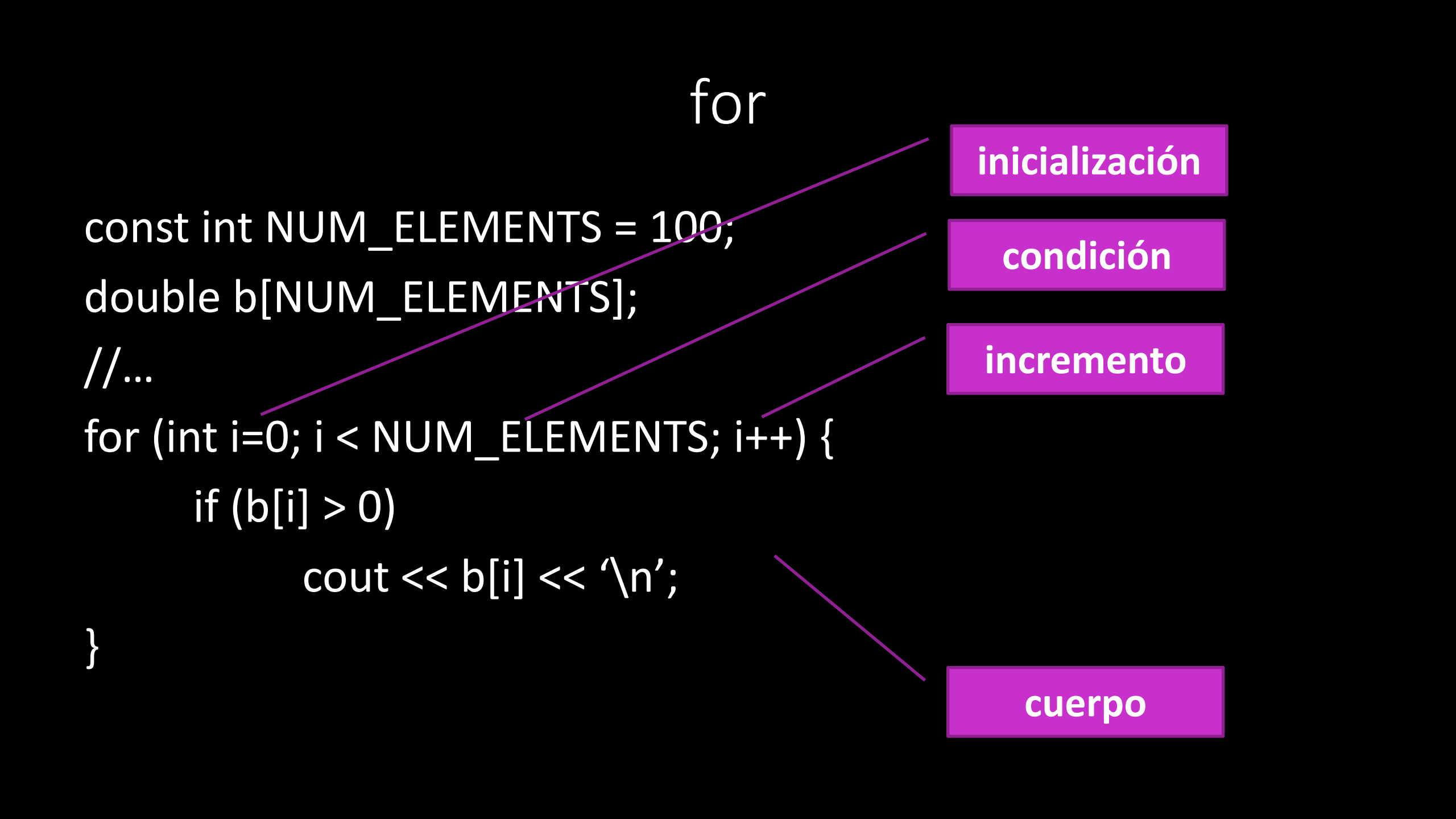
for

- Tres partes:
  - Inicialización
  - Condición
  - Incremento

```
for (initialization; condition; increment)
        loop_body_statement
```

for

```
const int NUM_ELEMENTS = 100;
double b[NUM_ELEMENTS];
//…
for (int i=0; i < NUM_ELEMENTS; i++) {
        if (b[i] > 0)
                cout << b[i] << '\n';
}
```



Break y continue

- Break: termina la ejecución del ciclo más interno
- Continue: termina la ejecución de la iteración actual y sigue con la siguiente

Funciones

```
return_type function_name ( args )
{
        statements
}
```

Funciones

tipo retorno

nombre

argumentos

cuerpo

```
bool evenSum(int a[], int n) {
        int sum = 0;
        for (int i = 0; i < n; i++)
                sum += a[i];
        return (sum % 2) == 0;
}
```

Paso de argumentos

- Por valor o por referencia
- Valor
  - Lo que la función recibe es una copia del valor de la variable que se envió como argumento
  - Si se cambia, la variable original permanece igual
- Referencia
  - La función recibe la variable como tal
  - Si se cambia, la variable original también

```
void f(int value, int& ref) {          // one value and one reference
  value++;                             // no effect on the actual argument
  ref++;                               // modifies the actual argument
  cout << value << endl;               // outputs 2
  cout << ref << endl;                 // outputs 6
}

int main() {
  int cat = 1;
  int dog = 5;
  f(cat, dog);                         // pass cat by value, dog by ref
  cout << cat << endl;                 // outputs 1
  cout << dog << endl;                 // outputs 6
  return EXIT_SUCCESS;
}
```

Paso de argumentos

- Es más eficiente si se pasan parámetros por referencia porque no hay que hacer una copia de los mismos en memoria
- Para evitar que un argumento por referencia sea modificado

  void nombreFuncion(const Passenger& pass)
- Los arreglos NO se pueden enviar por valor, sólo por referencia

Clases

```
class Counter {                          // a simple counter
public:
  Counter();                             // initialization
  int getCount();                        // get the current count
  void increaseBy(int x);                // add x to the count
private:
  int count;                             // the counter's value
};

Counter::Counter()                       // constructor
  { count = 0; }
int Counter::getCount()                  // get current count
  { return count; }
void Counter::increaseBy(int x)          // add x to the count
  { count += x; }
```

```
Passenger pass;                          // pass is a Passenger
// ...
if ( !pass.isFrequentFlyer() ) {         // not already a frequent flyer?
  pass.makeFrequentFlyer("392953");      // set pass's freq flyer number
}
pass.name = "Joe Blow";                  // ILLEGAL! name is private
```

```
Counter ctr;                              // an instance of Counter
cout << ctr.getCount() << endl;           // prints the initial value (0)
ctr.increaseBy(3);                        // increase by 3
cout << ctr.getCount() << endl;           // prints 3
ctr.increaseBy(5);                        // increase by 5
cout << ctr.getCount() << endl;           // prints 8
```

Definición de funciones dentro de la clase

```
class Passenger {
public:
    // ...
    bool isFrequentFlyer() const { return isFreqFlyer; }
    // ...
};
```

Constructores

- Inicializa los miembros de una clase
- Constructor por defecto → sin argumentos
- Se pueden definir diferentes constructores que se diferencian por la cantidad de argumentos

```
class Passenger {
private:
  // ...
public:
  Passenger();                                    // default constructor
  Passenger(const string& nm, MealType mp, const string& ffn = "NONE");
  Passenger(const Passenger& pass);  // copy constructor
  // ...
};
```

```
Passenger::Passenger() {                    // default constructor
  name = "--NO NAME--";
  mealPref = NO_PREF;
  isFreqFlyer = false;
  freqFlyerNo = "NONE";
}
                                            // constructor given member values
Passenger::Passenger(const string& nm, MealType mp, const string& ffn) {
  name = nm;
  mealPref = mp;
  isFreqFlyer = (ffn != "NONE");            // true only if ffn given
  freqFlyerNo = ffn;
}
                                            // copy constructor
Passenger::Passenger(const Passenger& pass) {
  name = pass.name;
  mealPref = pass.mealPref;
  isFreqFlyer = pass.isFreqFlyer;
  freqFlyerNo = pass.freqFlyerNo;
}
```

```
Passenger p1;                                          // default constructor
Passenger p2("John Smith", VEGETARIAN, 293145); // 2nd constructor
Passenger p3("Pocahontas", REGULAR);     // not a frequent flyer
Passenger p4(p3);                                      // copied from p3
Passenger p5 = p2;                                     // copied from p2
Passenger* pp1 = new Passenger;                // default constructor
Passenger* pp2 = new Passenger("Joe Blow", NO_PREF); // 2nd constr.
Passenger pa[20];                                      // uses the default constructor
```

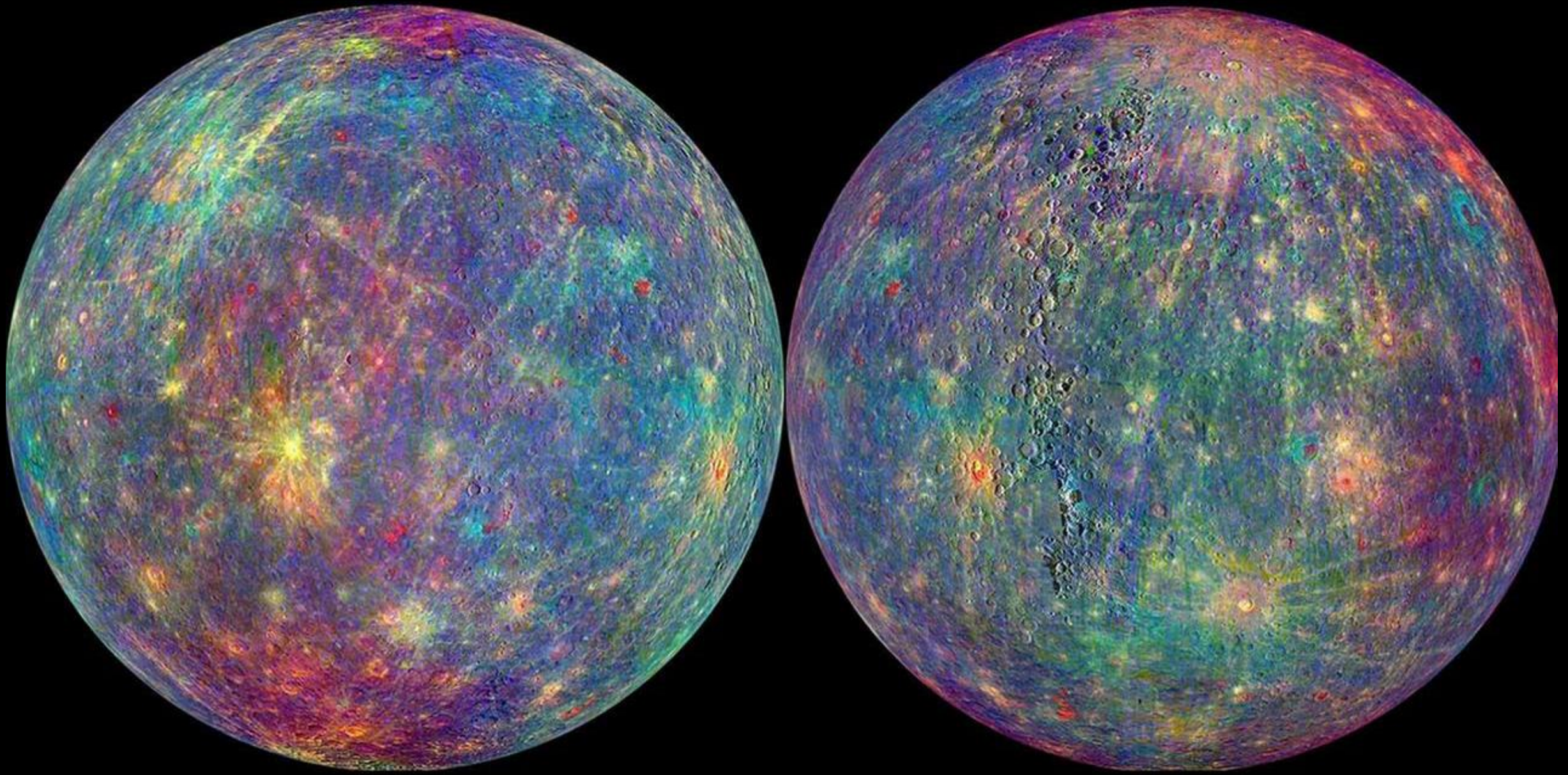
Destructores

- Método de la clase que se llama automáticamente cuando el objeto deja de existir
- new → constructor
- delete → destructor
- Son necesarios en clases que reservan memoria
- Es responsabilidad del destructor liberar esos recursos

```
class Vect {                                    // a vector class
public:
  Vect(int n);                                  // constructor, given size
  ~Vect();                                      // destructor
  // ... other public members omitted
private:
  int*          data;                           // an array holding the vector
  int           size;                           // number of array entries
};

Vect::Vect(int n) {                             // constructor
  size = n;
  data = new int[n];                            // allocate array
}

Vect::~Vect() {                                 // destructor
  delete [] data;                               // free the allocated array
}
```



Introducción a C++

Mauricio Avilés