Simulación RTL en C++

Ing. Leonardo Luis Ortiz

Instituto Balseiro
Centro Atómico Bariloche

14 de agosto de 2025

Instituto Balseiro

Agenda

1. C++ Review
2. Python Review
3. Referencias

Instituto Balseiro

C++ Review

Instituto Balseiro

Desarrollado por Bjarne Stroustrup en 1989.

Un standard formal desarrollado a principios de 1997.

C++ es un lenguaje de propósito general de tipado estático que se basa en clases y funciones virtuales para tener soporte de programación orientada a objetos, templates (plantillas) para dar soporte de programación genérica y proporciona funciones de bajo nivel para permitir la programación detallada de sistemas.

C++ es un lenguaje multiparadigma. En otras palabras, C++ fue diseñado para admitir una amplia gama de estilos. Ningún lenguaje puede admitir todos los estilos. Sin embargo, se pueden admitir diversos estilos dentro del marco de un solo lenguaje. Cuando esto es posible, se obtienen importantes ventajas al compartir un sistema de tipos común, un conjunto de herramientas común, etc. Estas ventajas técnicas se traducen en importantes beneficios prácticos, como permitir que grupos con necesidades moderadamente diferentes compartan un lenguaje en lugar de tener que aplicar varios lenguajes especializados.

Instituto Balseiro

Introducción

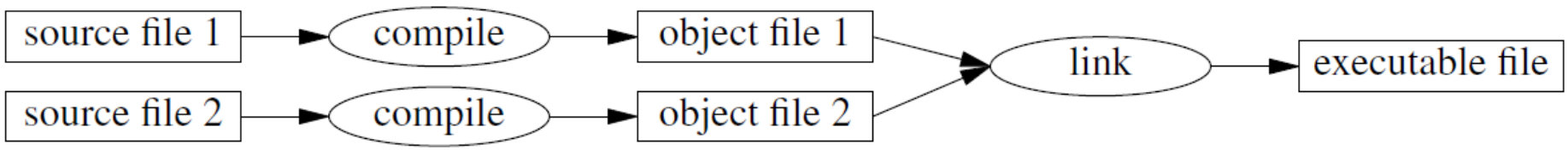
Este capítulo presenta de manera informal la notación de C++, el modelo de memoria y computación de C++, y los mecanismos básicos para organizar el código en un programa. Estas son las facilidades del lenguaje que soportan los estilos más comunes en C y que a veces se denominan programación procedimental.

- Programas
- Hola Mundo!
- Funciones
- Tipos, variables y aritmética
- Alcance (scope) y Tiempo de vida (lifetime)
- Constantes
- Punteros, Arrays y Referencias
- Testeos
- Mapeo a Hardware

Instituto Balseiro

Programas

C++ es un lenguaje compilado. Para que un programa se ejecute, su código fuente debe ser procesado por un compilador, que genera archivos objeto, los cuales se combinan mediante un enlazador para producir un programa ejecutable. Un programa C++ suele constar de muchos archivos de código fuente (normalmente denominados simplemente archivos fuente).



El estándar ISO C++ define dos tipos de entidades:

- *Características básicas del lenguaje*, como tipos integrados (por ejemplo, **char** e **int**) y bucles (por ejemplo, sentencias **for** y **while**)
- *Componentes de la biblioteca estándar*, como contenedores (por ejemplo, **vector** y **map**) y operaciones de E/S (por ejemplo, « y **getline()**)

Instituto Balseiro

Programas

C++ es un lenguaje de tipado estático. Es decir, el compilador debe conocer el tipo de cada entidad (por ejemplo, objeto, valor, nombre y expresión) en el momento de su uso. El tipo de un objeto determina el conjunto de operaciones que se pueden aplicar a él y su disposición en la memoria.

Instituto Balseiro

Hola Mundo!

```
import std;    // import the declarations for the standard library

int main()
{
  std::cout << "Hello, World!\n";
}
```

```
#include <iostream>

using namespace std; // make names from std visible without std::

int main()
{
  cout << "Hello, World!\n";
}
```

Instituto Balseiro

Funciones

```
double getAreaFromShape(Shape& ob); // Declaration

int main(){
...
}

double getAreaFromShape(Shape& ob)   // Definition
{
   ...
}
```

```
Elem* next_elem(); // no argument; return a pointer to Elem (an
    Elem*)
void exit(int); // int argument; return nothing
double sqrt(double); // double argument; return a double
double sqrt(double d); // return the square root of d
double square(double); // return the square of the argument
double get(const vector<double>& vec, int index); // type: double
    (const vector<double>&,int
char& String::operator[](int index); // type: char& String::(int)
```

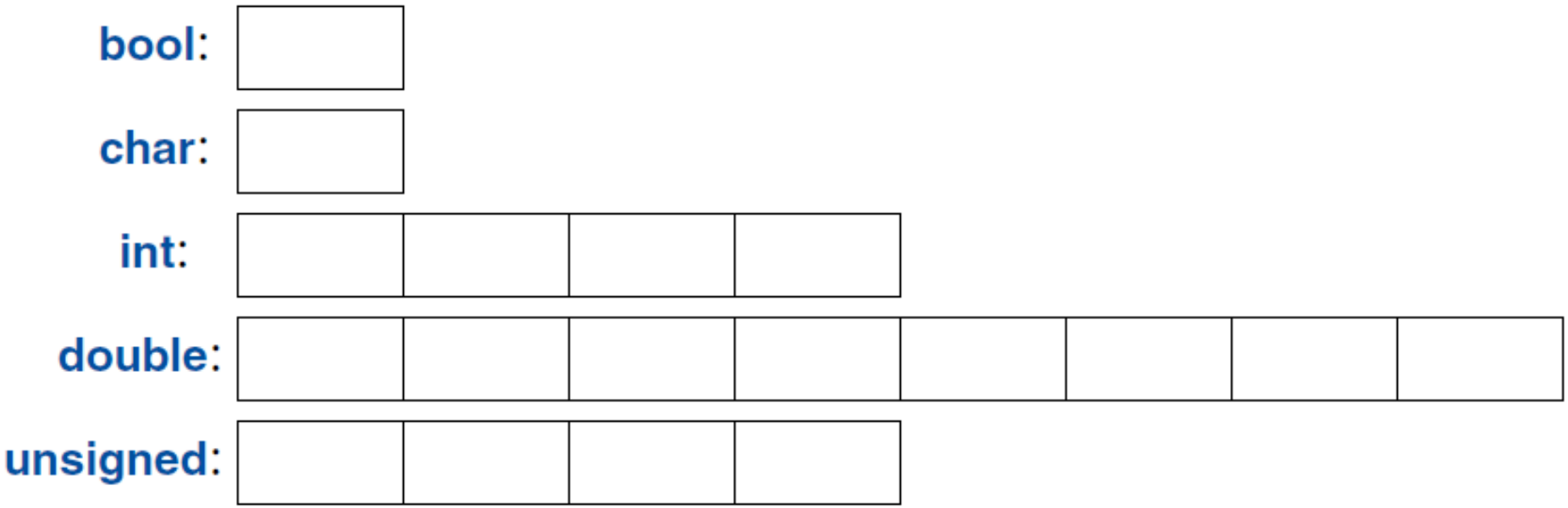
Tipos, variables y aritmética

Cada nombre y cada expresión tiene un tipo que determina las operaciones que pueden realizarse con él.
Una declaración es una sentencia que introduce una entidad en el programa y especifica su tipo:

- Un *tipo* define un conjunto de valores posibles y un conjunto de operaciones (para un objeto).
- Un *objeto* es una memoria que contiene un valor de algún tipo.
- Un *valor* es un conjunto de bits interpretados según un tipo.
- Una *variable* es un objeto con nombre.

Instituto Balseiro

Tipos, variables y aritmética




```
pi = 3.14159'26535'89793'23846'26433'83279'50288;
```

Instituto Balseiro

Operadores aritméticos

```
x+y // plus                                    1
+x // unary plus                               2
x-y // minus                                   3
-x // unary minus                              4
x*y // multiply                                5
x/y // divide                                  6
x%y // remainder (modulus) for                 7
    integers
```

```
x+=y // x = x+y
++x // increment: x = x+1
x-=y // x = x-y
--x // decrement: x = x-1
x*=y // scaling: x = x*y
x/=y // scaling: x = x/y
x%=y // x = x%y
```

Operadores de comparación

```
x==y // equal                                  1
x!=y // not equal                              2
x<y // less than                               3
x>y // greater than                            4
x<=y // less than or equal                     5
x>=y // greater than or equal                  6
                                               7
```

Operadores lógicos

```
x&y // bitwise and
x|y // bitwise or
x^y // bitwise exclusive or
~x // bitwise complement
x&&y // logical and
x||y // logical or
!x // logical not (negation)
```

Instituto Balseiro

Tipos, variables y aritmética

El orden de evaluación es de izquierda a derecha para **x.y**, **x->y**, **x(y)**, **x[y]**, **x«y**, **x»y**, **x&&y**, y **x||y**.
Para asignaciones es de derecha a izquierda **x+=y**.

Instituto Balseiro

Inicialización

```
double d1 = 2.3; // initialize d1 to 2.3
double d2 {2.3}; // initialize d2 to 2.3
double d3 = {2.3}; // initialize d3 to 2.3 (the = is optional with
    {... })
complex<double> z = 1; // a complex number with double precision
    floating point scalars
complex<double> z2 {d1,d2};
complex<double> z3 = {d1,d2}; // the = is optional with {...}
vector<int> v {1, 2, 3, 4, 5, 6}; // a vector of ints

int i1 = 7.8; // i1 becomes 7 (surprise?)
int i2 {7.8}; // error: floating-point to integer conversion

auto b = true; // a bool
auto ch = 'x'; // a char
auto i = 123; // an int
auto d = 1.2; // a double
auto z = sqrt(y); // z has the type of whatever sqrt(y) returns
auto bb {true}; // bb is a bool
```

Instituto Balseiro

Alcance y Tiempo de vida

Una declaración introduce su nombre en un ámbito:

- *Local scope*: un nombre declarado en una función o lambda se denomina nombre local. Su alcance se extiende desde el punto de declaración hasta el final del bloque en el que se produce la declaración. Un bloque está delimitado por un par de . Los nombres de los argumentos de las funciones se consideran nombres locales.
- *Class scope*: un nombre se denomina nombre miembro (o nombre miembro de un clase) si se define en una clase, fuera de cualquier función, lambda o clase enumerada. Su ámbito se extiende desde la apertura de su declaración envolvente hasta la correspondiente .
- *Namescape scope*: un nombre se denomina nombre miembro del espacio de nombres si se define en un espacio de nombres fuera de cualquier función, lambda, clase o clase enumerada. Su ámbito se extiende desde el punto de declaración hasta el final de su namespace.

Instituto Balseiro

Alcance y Tiempo de vida

Un nombre que no se declara dentro de ninguna otra construcción se denomina nombre global y se dice que está en el namespace global. Además, podemos tener objetos sin nombre, como los temporales y los creados mediante **new**. Por ejemplo:

```
vector<int> vec; // vec is global (a global vector of integers)

void fct(int arg) // fct is global (names a global function)
// arg is local (names an integer argument)
{
  string motto {"Who dares wins"}; // motto is local
  auto p = new Record{"Hume"}; // p points to an unnamed Record (
      created by new)
  // ...
}

struct Record {
  string name; // name is a member of Record (a string member)
  // ...
};
```

Constantes

C++ admite dos conceptos de inmutabilidad (un objeto con un estado inalterable):

- **const**: significa aproximadamente «prometo no cambiar este valor». Se utiliza principalmente para especificar interfaces, de modo que los datos se puedan pasar a funciones utilizando punteros y referencias sin temor a que se modifiquen. El compilador hace cumplir la promesa hecha por **const**. El valor de un **const** se puede calcular en tiempo de ejecución.
- **constexpr**: significa aproximadamente «evaluarse en tiempo de compilación». Se utiliza principalmente para especificar constantes, para permitir la colocación de datos en memoria de solo lectura (donde es poco probable que se corrompan) y para mejorar el rendimiento. El valor de un **constexpr** debe ser calculado por el compilador.

Instituto Balseiro

Mapeo a Hardware

C++ ofrece una asignación directa al hardware. Al utilizar una de las operaciones fundamentales, la implementación es la que ofrece el hardware, normalmente una sola operación de máquina. Por ejemplo, al sumar dos enteros, x+y ejecuta una instrucción de máquina de suma de enteros. Una implementación de C++ ve la memoria de una máquina como una secuencia de ubicaciones de memoria en las que puede colocar objetos (tipificados) y direccionarlos mediante punteros.

con punteros

```
int x = 2;                                    1
int y = 3;                                    2
int* p = &x;                                  3
int* q = &y; // p!=q and *p!=*q               4
p = q; // p becomes &y; now p==q              5
     so (obviously)*p==*q
```

con referencias

```
int x = 2;
int y = 3;
int& r = x; // r refers to x
int& r2 = y; // r2 refers to y
r = r2; // read through r2, write
     through r: x becomes 3
```

Instituto Balseiro

Tipos definidos por el usuario

- Estructuras
- Clases
- Enumeraciones
- Uniones

Instituto Balseiro

Estructuras

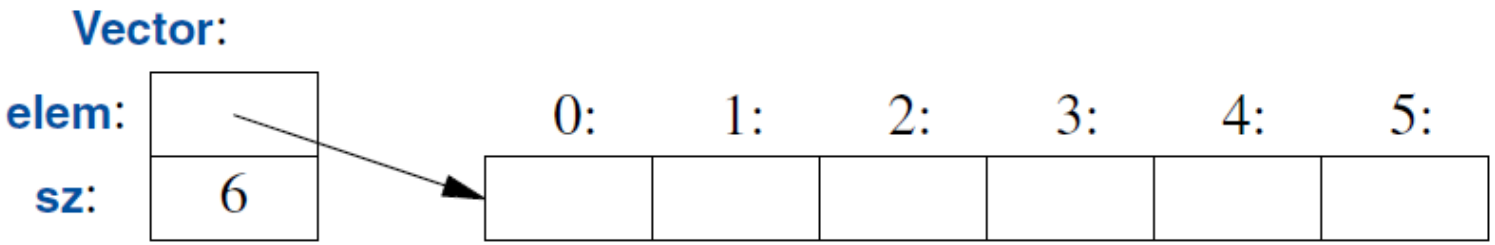
```
struct Vector {
  double* elem; // pointer to elements
  int sz; // number of elements
};
Vector v;

void vector_init(Vector& v, int s) // initialize a Vector
{
  v.elem = new double[s]; // allocate an array of s doubles
  v.sz = s;
}
```

```
double read_and_sum(int s){
  Vector v;
  vector_init(v,s); // allocate s elements for v
  for (int i=0; i!=s; ++i)
    cin>>v.elem[i]; // read into elements
  double sum = 0;
  for (int i=0; i!=s; ++i)
    sum+=v.elem[i]; // compute the sum of the elements
  return sum;
}
```

Instituto Balseiro

Clases

```
class Vector {
  public:
    Vector(int s) :elem{new double[s]}, sz{s} { } // construct a
        Vector
    double& operator[](int i) { return elem[i]; } // element access
        : subscripting
    int size() { return sz; }
  private:
    double* elem; // pointer to the elements
    int sz; // the number of elements
};

Vector v(6); // a Vector with 6 elements
```



Instituto Balseiro

Enumeraciones

```
enum class Color { red, blue, green };
enum class Traffic_light { green, yellow, red };
Color col = Color::red;
Traffic_light light = Traffic_light::red;
```

El **class** después de **enum** especifica que la enumeración es fuertemente tipeada y que los enumeradores tienen ámbito (scoped).

```
Traffic_light& operator++(Traffic_light& t) // prefix increment: ++
{
  switch (t) {
    case Traffic_light::green: return t=Traffic_light::yellow;
    case Traffic_light::yellow: return t=Traffic_light::red;
    case Traffic_light::red: return t=Traffic_light::green;
  }
}
auto signal = Traffic_light::red;
Traffic_light next = ++signal; // next becomes Traffic_light::green
```

Instituto Balseiro

Enumeraciones

Si la repetición del nombre de la enumeración, `Traffic_light`, se vuelve demasiado tediosa, podemos abreviarlo en un ámbito.

```
Traffic_light& operator++(Traffic_light& t) // prefix increment: ++
{
  using enum Traffic_light; // here, we are using Traffic_light

  switch (t) {
    case green: return t=yellow;
    case yellow: return t=red;
    case red: return t=green;
  }
}
```

Uniones

Una **union** es una **struct** en la que todos los miembros se asignan a la misma dirección, de modo que ocupa solo el espacio de su miembro más grande. Naturalmente, una **union** solo puede contener el valor de un miembro a la vez.

```
enum class Type { ptr, num }; // a Type can hold values ptr and num
struct Entry {
  string name; // string is a standard-library type
  Type t;
  Node* p; // use p if t==Type::ptr
  int i; // use i if t==Type::num
};
```

Los miembros **p** e **i** nunca se usan simultáneamente, por lo que se desperdicia espacio. Esto se puede recuperar fácilmente especificando que ambos deben ser miembros de una **union**, como se muestra a continuación:

Instituto Balseiro

Uniones

```
union Value {
  Node* p;
  int i;
};
```

Ahora **Value::p** y **Value::i** se colocan en la misma dirección de memoria de cada objeto **Value**. El lenguaje no realiza un seguimiento de qué tipo de valor contiene una **union**, por lo que el programador debe hacerlo:

```
struct Entry {
  string name;
  Type t;
  Value v; // use v.p if t==Type::ptr; use v.i if t==Type::num
};
```

Instituto Balseiro

Módulos

C++ admite la compilación independiente, donde el código de usuario solo ve las declaraciones de los tipos y funciones utilizados. Esto se puede hacer de dos maneras:

- *Header files*: Coloca las declaraciones en archivos separados, llamados archivos de cabecera, y textualmente #**include** un archivo de encabezado donde se necesiten sus declaraciones.
- *Modules*: Define los archivos **module**, los compila por separado e **import**a donde sea necesario. Solo las declaraciones explícitamente **export**adas son vistas por el código **import**ando el **module**.

Instituto Balseiro

Header Files

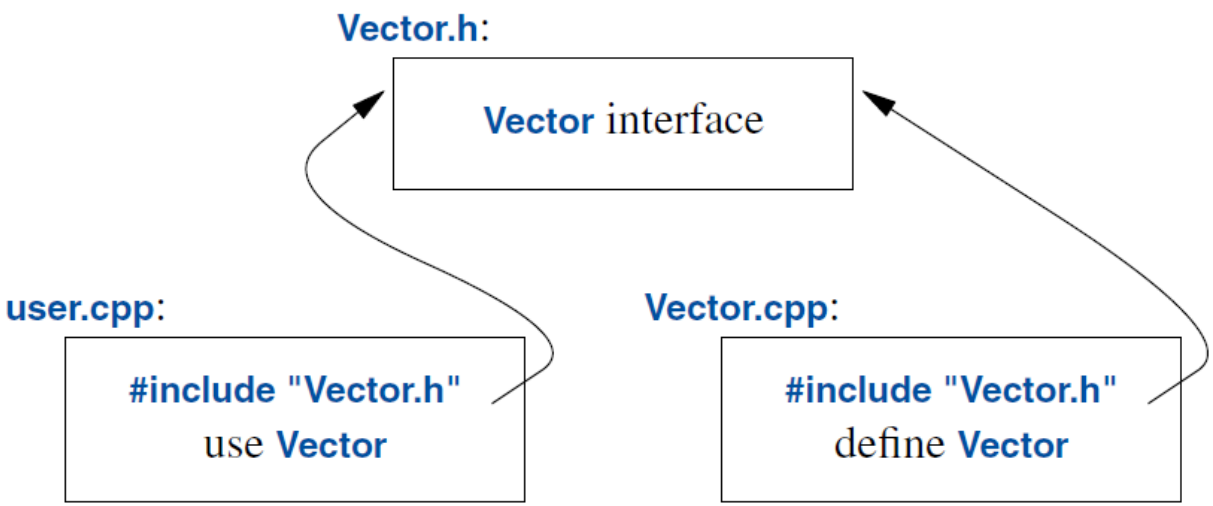
```
// Vector.h:
class Vector {
  public:
    Vector(int s);
    double& operator[](int i);
    int size();
  private:
    double* elem; // elem points to an array of sz doubles
    int sz;
};
```

```
// Vector.cpp:
#include "Vector.h" // get Vector's interface

Vector::Vector(int s) :elem{new double[s]}, sz{s} // initialize
    members
{ ... }
double& Vector::operator[](int i) {
  return elem[i];
}
int Vector::size(){
  return sz;
}
```

Instituto Balseiro

Header Files

```cpp
// user.cpp:
#include "Vector.h" // get Vector's interface
#include <cmath> // get the standard-library math function
    interface including sqrt()

double sqrt_sum(const Vector& v)
{
  double sum = 0;
  for (int i=0; i != v.size(); ++i)
  sum +=std::sqrt(v[i]); // sum of square roots
  return sum;
}
```



Instituto Balseiro

Modules

```
export module Vector; // defining the module called "Vector"

export class Vector {
  public:
  Vector(int s);
  double& operator[](int i);
  int size();
  private:
  double* elem; // elem points to an array of sz doubles
  int sz;
};

Vector::Vector(int s)
  :elem{new double[s]}, sz{s} // initialize members
{
}

double& Vector::operator[](int i)
{
  return elem[i];
}
```

Modules

```
int Vector::size()
{
  return sz;
}

export bool operator==(const Vector& v1, const Vector& v2)
{
  if (v1.size()!=v2.size())
    return false;
  for (int i = 0; i<v1.size(); ++i)
    if (v1[i]!=v2[i])
      return false;
  return true;
}
```

Esto define un **module** llamado **Vector**, que exporta la clase **Vector**, todas sus funciones miembro y el operador de definición de función no miembro ==.

Instituto Balseiro

Modules

Este módulo se utiliza importándolo donde lo necesitamos. Por ejemplo:

```
// file user.cpp:

import Vector; // get Vector's interface
#include <cmath> // get the standard-library math function
    interface including sqrt()

double sqrt_sum(Vector& v)
{
  double sum = 0;
  for (int i=0; i!=v.size(); ++i)
    sum+=std::sqrt(v[i]); // sum of square roots
  return sum;
}
```

Se podría haber **import**ado también las funciones matemáticas de la biblioteca estándar, pero se usó el clásico #**include** solo para demostrar que podemos combinar funciones antiguas con nuevas. Esta combinación es esencial para actualizar gradualmente el código antiguo, pasando de usar #**include** a usar **import**.

Instituto Balseiro

Modules

Las diferencias entre headers y módulos no son solo sintácticas.

- Un módulo se compila una sola vez (en lugar de en cada unidad en la que se utiliza).
- Se pueden importar dos módulos en cualquier orden sin cambiar su significado.
- Si se **import** o #**include** algo en un módulo, los usuarios del módulo no acceden implícitamente a (y no les molesta): **import** no es transitivo.

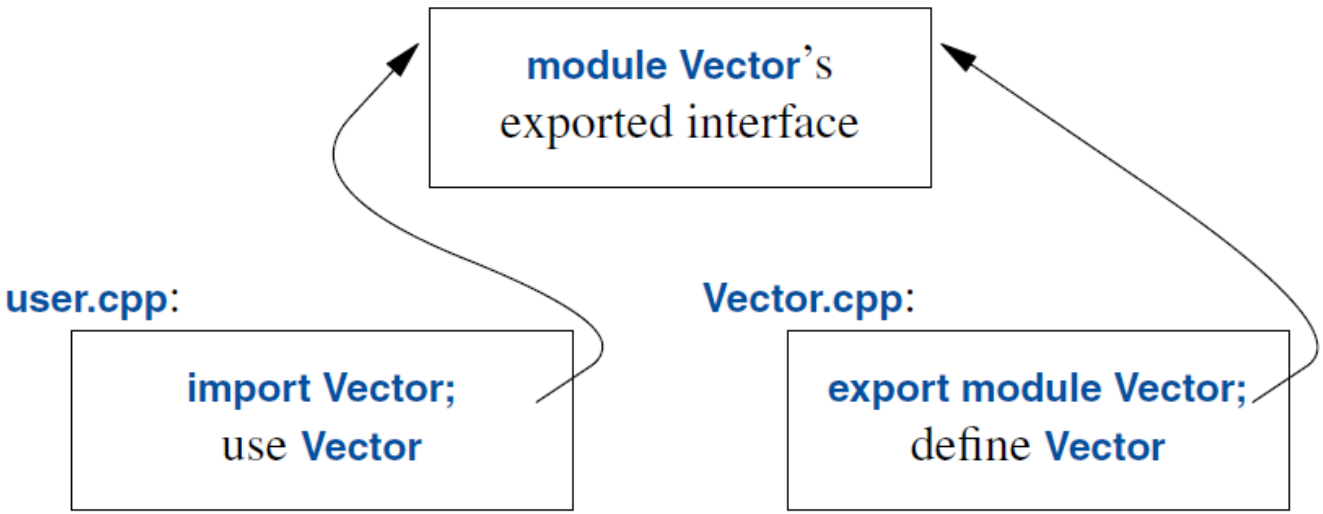
NOTA: Algunos compiladores todavía no admiten el uso de **import std**, por lo que en esos casos se debe seguir usando #**include**<**iostream**>

Instituto Balseiro

Modules

Al definir un módulo, no es necesario separar las declaraciones y definiciones en archivos separados; podemos hacerlo si esto mejora la organización del código fuente, pero no es obligatorio. Podríamos definir el módulo **Vector** simple así:

```
export module Vector; // defining the module called "Vector"
export class Vector {
  // ...
};
export bool operator==(const Vector& v1, const Vector& v2){
  // ...
}
```



Instituto Balseiro

Namespaces

Además de las funciones, las clases y las enumeraciones, C++ ofrece *namespaces* como mecanismo para expresar que algunas declaraciones pertenecen juntas y que sus nombres no deben entrar en conflicto con otros nombres.

```
void my_code(vector<int>& x, vector<int>& y)
{
  using std::swap;  // make the standard-library swap available
      locally
  // ...
  swap(x,y);        // std::swap()
  other::swap(x,y);   // some other swap()
  // ...
}
```

Instituto Balseiro

Tipos de clases

- Concretas: La idea básica es que se comportan ïgual que los tipos integrados". Por ejemplo, un tipo de número complejo y un entero de precisión infinita son muy similares a un int integrado, excepto, por supuesto, que tienen su propia semántica y conjuntos de operaciones. La característica definitoria de un tipo concreto es que su representación es parte de su definición.
- Abstractas: Es un tipo que aísla completamente al usuario de los detalles de implementación. Para ello, desacoplamos la interfaz de la representación y renunciamos a las variables locales genuinas.
- Jerarquías de clases: Es un conjunto de clases ordenadas en una red creada por derivación (p. ej., : public). Usamos jerarquías de clases para representar conceptos que tienen relaciones jerárquicas, como «Un camión de bomberos es un tipo de camión que a su vez es un tipo de vehículo».

La palabra **virtual** significa “puede ser redefinido posteriormente en una clase derivada de ésta”.

Instituto Balseiro

Clases

```
class MyClass
{
  public:
  // everything in here
  // has public access level
  protected:
  // everything in here
  // has protected access level
  private:
  // everything in here
  // has private access level
};
```

Instituto Balseiro

Clases

Un tipo de aritmética clásica definida por el usuario que trabaje con números complejos:

```
class complex {
  double re, im; // representation: two doubles
public:
  complex(double r, double i) :re{r}, im{i} {} // construct complex
       from two scalars
  complex(double r) :re{r}, im{0} {} // construct from one scalar
  complex() :re{0}, im{0} {} // default complex: {0,0}
  complex(complex z) :re{z.re}, im{z.im} {} // copy constructor
  ~complex() { std::cout << "Destructor invoked."; }

  double real() const { return re; }
  void real(double d) { re=d; }
  double imag() const { return im; }
  void imag(double d) { im=d; }

  complex& operator+=(complex z)
  {
    re+=z.re; // add to re and im
    im+=z.im;
    return *this; // return the result
  }
```

Clases - continuación

```
  complex& operator-=(complex z)
  {
    re-=z.re;
    im-=z.im;
    return *this;
  }
  complex& operator*=(complex); // defined out-of-class somewhere
  complex& operator/=(complex); // defined out-of-class somewhere
};

complex& complex::operator*=(complex other) {
  // ...
  return *this;
}

complex& complex::operator/=(complex other) {
  // ...
  return *this;
}
```

Instituto Balseiro

Clases - continuación

```
int main()
{
  complex cpx_value{ 1.5, -2.0 }; // invoke a user-defined
      constructor
  std::cout << cpx_value.real() << ' ' << cpx_value.imag() << 'j';
}
```

Instituto Balseiro

Herencia

```
class MyBaseClass
{
  public:
  char c;
  int x;
};

class MyDerivedClass : public MyBaseClass
{
  // c and x also accessible here
};

int main()
{
  MyDerivedClass o;
  o.c = 'a';
  o.x = 123;
}
```

En este ejemplo, **MyDerivedClass** hereda de **MyBaseClass**.

Instituto Balseiro

Herencia

Si definimos la **MyBaseClass** con las variables miembro como **protected**

```
class MyBaseClass
{
  protected:
    char c;
    int x;
};

class MyDerivedClass : public MyBaseClass
{
  // c and x also accessible here
};

int main()
{
  MyDerivedClass o;
  o.c = 'a'; // Error, not accessible to object
  o.x = 123; // error, not accessible to object
}
```

En este ejemplo, **MyDerivedClass** hereda de **MyBaseClass**.

Instituto Balseiro

Herencia

Si definimos la **MyBaseClass** con las variables miembro como **private**

```
class MyBaseClass
{
  private:
    char c;
    int x;
};
class MyDerivedClass : public MyBaseClass
{
  // c and x NOT accessible here
};
int main()
{
  MyDerivedClass o;
  o.c = 'a'; // Error, not accessible to object
  o.x = 123; // error, not accessible to object
}
```

En este ejemplo, **MyDerivedClass** hereda de **MyBaseClass**.

Instituto Balseiro

Polimorfismo

Se dice que la clase derivada es una clase base. Su tipo es compatible con el tipo de la clase base. Además, un puntero a una clase derivada es compatible con un puntero a la clase base. Esto es importante, así que repitámoslo: un puntero a una clase derivada es compatible con un puntero a una clase base. Junto con la herencia, esto se utiliza para lograr la funcionalidad conocida como **polimorfismo**. El polimorfismo significa que el objeto puede transformarse en diferentes tipos.

El polimorfismo en C++ se logra mediante una interfaz conocida como funciones virtuales. Una función virtual es una función cuyo comportamiento puede sobrescribirse en clases derivadas posteriores. Y nuestro puntero/objeto se transforma en diferentes tipos para invocar la función apropiada.

Instituto Balseiro

Polimorfismo

```cpp
#include <iostream>

class MyBaseClass
{
  public:
  virtual void dowork()
  {
    std::cout << "Hello from a base class." << '\n';
  }
};

class MyDerivedClass : public MyBaseClass
{
  public:
  void dowork()
  {
    std::cout << "Hello from a derived class." << '\n';
  }
};
```

Instituto Balseiro

Polimorfismo

```
int main()
{
  MyBaseClass* o = new MyDerivedClass;
  o->dowork();
  delete o;
}
```

Instituto Balseiro

Los tres pilares de la programación orientada a objetos son:

- Encapsulación
- Herencia
- Polimorfismo

La *encapsulación* consiste en agrupar los campos en diferentes zonas de visibilidad, ocultar la implementación al usuario y exponer la interfaz, por ejemplo.
La *herencia* es un mecanismo mediante el cual podemos crear clases heredando de una clase base. La herencia crea una jerarquía de clases y una relación entre ellas.
El *polimorfismo* es la capacidad de un objeto de transformarse en diferentes tipos durante la ejecución, garantizando que se invoque la función correcta. Esto se logra mediante la herencia, las funciones virtuales y sobrescritas, y los punteros de clase base y derivada.

Instituto Balseiro

Templates

Los templates (plantillas) son mecanismos que facilitan la programación genérica. En términos generales, "genérico"significa que podemos definir una función o clase sin importar qué tipos acepte. Definimos estas funciones y clases usando un tipo genérico. Y al instanciarlos, usamos un tipo concreto. Por lo tanto, podemos usar templates cuando queremos definir una clase o función que acepte prácticamente cualquier tipo.

```
#include <iostream>

template <typename T>
void myfunction(T param){
  std::cout << "The value of a parameter is: " << param;
}

int main()
{
  myfunction<int>(123);
  myfunction<double>(123.456);
  myfunction<char>('A');
}
```

Templates

```
#include <iostream>

template <typename T>
class MyClass {
  private:
    T x;
  public:
    MyClass(T xx);
};
```

Instituto Balseiro

Templates

```cpp
template <typename T>
MyClass<T>::MyClass(T xx)
  : x{xx}
{
  std::cout << "Constructor invoked. The value of x is: " << x << '
      \n';
}

int main()
{
  MyClass<int> o{ 123 };
  MyClass<double> o2{ 456.789 };
}
```

Instituto Balseiro

PYTHON Review

Instituto Balseiro

El lenguaje de programación Python fue creado al principio de los 90' por Guido van Rossum.

Es un lenguaje de programación potente y fácil de aprender. Tiene estructuras de datos de alto nivel eficientes y un simple pero efectivo sistema de programación orientado a objetos. La elegante sintaxis de Python y su tipado dinámico, junto a su naturaleza interpretada lo convierten en un lenguaje ideal para scripting y desarrollo rápido de aplicaciones en muchas áreas, para la mayoría de plataformas.

Instituto Balseiro

Tipos de datos: básicos

Los tipos de datos son la base de cualquier programa, ya que determinan cómo se almacenan y manipulan los valores. Existen tipos primitivos como enteros, flotantes, cadenas de texto y booleanos, y estructuras de datos más avanzadas como listas, tuplas, conjuntos y diccionarios.

Nombre	**Palabra reservada**	**Sintaxis**
Entero	int	`0`
Flotante	float	`3.14`
Cadena de texto	str	`"Hola"`
Booleano	bool	`True / False`
Complejo	complex	`1 + 2j`
Byte	bytes	`b"Hola"`
Nulo	NoneType	`None`

Instituto Balseiro

Tipos de datos: Estructuras

Nombre	**Palabra reservada**	**Sintaxis**
Lista/Pila	list	`[1, 2, 3]`
Tupla	tuple	`(1, 2, 3)`
Conjunto	set	`{1, 2, 3}`
Conjunto inmutable	frozenset	`frozenset({1, 2, 3})`
Diccionario	dict	`{“clave": "valor"}`
Bytearray	bytearray	`bytearray(5)`
Rango	range	`range(5)`
Memory view	memoryview	`memoryview(bytes(5))`

Instituto Balseiro

Sintaxis básica

Nombre	**P. reservada**	**Sintaxis**
Condicional if	if	`if` *condición* `:`
Condicional elif	elif	`elif` *otra_condición* `:`
Condicional else	else	`else:`
Bucle for	for	`for` *elemento* `in` *secuencia* `:`
Bucle while	while	`while` *condición* `:`
break / continue	break / continue	`break`, `continue`
Función def	def	`def` *nombre*`(`*parámetros*`):`
Clase class	class	`class` *NombreClase* `:`
Variable global	global	`global nombre_variable`
Excepciones	try, except	`try: ... except` *Excepción* `:`
Retorno	return	`return` *valor*
Comprobar tipo	is	`if` *variable* `is` *tipo* `:`
Existe	in	`if` *elemento* `in` *colección* `:`
Módulo	import, from	`import` *módulo* `from` *módulo* `import` *nombre*

Instituto Balseiro

Operadores: Aritméticos y de Comparación

Nombre	**Representación**	**Sintaxis**
Suma	+	a + b
Resta	-	a - b
Multiplicación	*	a * b
División	/	a / b
División entera	//	a // b
Módulo (residuo)	%	a % b
Exponenciación	**	a**b

Igualdad	==	a == b
Distinto de	!=	a != b
Mayor que	>	a > b
Menor que	<	a < b
Mayor o igual que	>=/	a >= b
Menor o igual que	<=	a <= b

Instituto Balseiro

Operadores: Lógicos y de Asignación

Nombre	**Representación**	**Sintaxis**
AND	and	a and b
OR	or	a or b
NOT	not	not a

Asignación	=	x = 10
A. con suma	+=	x += 10
A. con resta	-=	x -= 10
A. con mult.	*=	x *= 10
A. con div.	/=	x /= 10
A. div. entera	//=	x //= 10
A. con módulo	%=	x%= 10
A. con expo.	**=	x **= 3
A. con bitwise AND	&=	x &= 3
A. con bitwise OR	|=	x |= 3
A. con bitwise XOR	≙	x ≙ 3

Instituto Balseiro

Operadores

Nombre	Representación	Sintaxis
Identidad		
Es	is	a is
No es	is not	a is not b
Pertenecia		
Pertenece	in	x in *lista*
No Pertenece	not in	x not in *lista*
Bitwise		
AND bit a bit	&	a & b
OR bit a bit	|	a | b
XOR bit a bit	^	a ^b
NOT bit a bit	~	a ~b
shift left	«	a « n
shift right	»	a » n

Instituto Balseiro

Funciones

Nombre	Operación
print()	Muestra texto o variables en la consola
input()	Permite la entrada de datos desde la consola
len()	Devuelve la cantidad de elementos
type()	Devuelve el tipo de dato
range()	Genera una secuencia de números en un rango
int(), float(), str()	Convierte a entero, punto flotante o cadena de caracteres
list(), tuple(), set(), dict()	Crea listas, tuplas, conjuntos o diccionarios
sum()	Suma los elementos iterables
min(), max()	Encuentra el mínimo o máximo en iterables
sorted()	Ordena elementos en listas y otros iterables
abs()	Retorna el valor absoluto de un número
round()	Redondea a un número con decimales concretos
map()	Aplica una función a cada elemento iterado
filter()	Filtra elementos iterables según una condición
reduce()	Aplica una función a los elementos iterables para reducirlos a un único valor
enumerate()	Agrega un índice a cada elemento iterable
zip()	Une iterables en pares de elementos
open()	Abre un archivo para leer, escribir o modificar

Instituto Balseiro

Funciones

Nombre	**Operación**
next()	Retorna el siguiente valor de un iterador
iter()	Transforma un iterable en un iterador
chr(), ord()	Trabaja con valores Unicode de caracteres
bin(), oct(), hex()	Convierte números enteros a diferentes bases
any(), all()	Evalúa condiciones en iterables para al menos uno o todos los elementos
pow()	Calcula una potencia
divmod()	Retorna el cociente y el resto de una división
eval()	Ejecuta una expresión en forma de string
format()	Formatea cadenas con valores personalizados
reversed()	Invierte el orden de un iterable
slice()	Crea una porción de un iterable sin modificarlo
complex()	Crea números complejos
bool()	Transforma un valor en booleano
bin()	Crea la representación binaria de un número
globals()	Retorna las variables globales
locals()	Retorna las variables locales

Instituto Balseiro

Estructuras de datos: Listas

Nombre	Operación	Sintaxis
append()	Agrega un elemento al final	lista.append(valor)
extend()	Agrega múltiples elementos	lista.extend(iterable)
insert()	Inserta un elemento en un índice	lista.insert(i, valor)
remove()	Elimina la primera ocurrencia de un valor	lista.remove(valor)
pop()	Elimina y devuelve un elemento (por índice o último)	lista.pop(i) lista.pop()
index()	Devuelve el índice de un valor	lista.index(valor)
count()	Cuenta las ocurrencias de un valor	lista.count(valor)
sort()	Ordena la lista	lista.sort()
reverse()	Invierte el orden de la lista	lista.reverse()
copy()	Crea una copia de la lista	lista.copy()
clear()	Elimina todos los elementos	lista.clear()

Instituto Balseiro

Estructuras de datos: Tuplas (tuple)

Nombre	**Operación**	**Sintaxis**
count()	Cuenta las ocurrencias de un valor	tupla.count(valor)
index()	Devuelve el índice de un valor	tupla.index(valor)

Instituto Balseiro

Estructuras de datos: Conjuntos (set)

Nombre	**Operación**	**Sintaxis**
add()	Agrega un elemento al conjunto	conjunto.add(valor)
remove()	Elimina un elemento (error si no existe)	conjunto.remove(valor)
discard()	Elimina un elemento (sin error si no existe)	conjunto.discard(valor)
pop()	Elimina y devuelve un elemento aleatorio	conjunto.pop()
clear()	Elimina todos los elementos	conjunto.clear()
union()	Une dos conjuntos	conjunto.union(otro)
intersection()	Elementos comunes en ambos conjuntos	conjunto.intersection(otro)
difference()	Elementos en A pero no en B	conjunto.difference(otro)

Instituto Balseiro

Estructuras de datos: Diccionarios (dict)

Nombre	**Operación**	**Sintaxis**
keys()	Devuelve las claves del diccionario	diccionario.keys()
values()	Devuelve los valores del diccionario	diccionario.values()
items()	Devuelve pares clave-valor	diccionario.items()
get()	Obtiene el valor de una clave (sin error)	diccionario. get(clave, defecto)
pop()	Elimina y devuelve un valor	diccionario.pop(clave)
popitem()	Elimina y devuelve el último par clave-valor	diccionario.popitem()
update()	Agrega o actualiza claves con valores	diccionario. update(otro_dict)
setdefault()	Obtiene el valor o lo asigna si no existe	diccionario. setdefault(clave, valor)
copy()	Crea una copia del diccionario	diccionario.copy()
clear()	Elimina todos los elementos	diccionario.clear()

Instituto Balseiro

Archivos

Nombre	Representación	Sintaxis
open()	Abre un archivo en diferentes modos de acceso	archivo = open(“archivo.txt”, “r”)
read()	Lee el contenido completo del archivo	contenido = archivo.read()
readline()	Lee una única línea del archivo	línea = archivo.readline()
readlines()	Lee todas las líneas y las devuelve en una lista	líneas = archivo.readlines()
write()	Escribe datos en un archivo (sobrescribe)	archivo.write(“Texto”)
writelines()	Escribe múltiples líneas en un archivo	archivo. writelines(lista_de_texto)
close()	Cierra el archivo para liberar recursos	archivo.close()
flush()	Fuerza la escritura de datos del buffer en el archivo	archivo.flush()
with open()	Abre un archivo con manejo automático de cierre	with open(“archivo.txt”) as f:
seek()	Mueve el cursor a una posición específica	archivo.seek(0)
tell()	Devuelve la posición actual del cursor	pos = archivo.tell()

Instituto Balseiro

Archivos

Nombre	Representación	Sintaxis
truncate()	Corta el archivo a un tamaño específico	archivo.truncate(50)
exists()	Comprueba si un archivo existe	os.path.exists(“archivo.txt”)
remove()	Elimina un archivo	os.remove(.ªrchivo.txt")
rename()	Renombra un archivo	os.rename("viejo.txt", “nuevo.txt”)
mkdir()	Crea un directorio	os.mkdir(“nueva_carpeta”)
rmdir()	Elimina un directorio vacío	os.rmdir(“nueva_carpeta”)
listdir()	Lista los archivos en un directorio	os.listdir(“ruta”)

Modos de apertura en **open()**:

- “r” → Lectura (por defecto)
- “w” → Escritura (borra contenido previo)
- “a” → Agregar contenido al final
- “rb” / “wb” → Lectura y escritura en modo binario

Es recomendable usar **with open()** en lugar de **open() + close()** para evitar problemas de memoria.

Instituto Balseiro

Módulos estándar

Nombre	Descripción
os	Permite interactuar con el sistema operativo (archivos, directorios, procesos)
sys	Proporciona acceso a variables y funciones del intérprete de Python
math	Ofrece funciones matemáticas avanzadas (raíces, logaritmos, trigonometría)
random	Genera números aleatorios y selecciona elementos al azar
datetime	Maneja fechas y horas
time	Funciones para manejar el tiempo y pausas en la ejecución
json	Permite trabajar con datos en formato JSON (serialización y deserialización)
csv	Facilita la lectura y escritura de archivos CSV
re	Proporciona herramientas para trabajar con expresiones regulares
collections	Contiene estructuras de datos avanzadas (deque, counter, defaultdict)
itertools	Ofrece herramientas para trabajar con iteradores y combinaciones de datos

Instituto Balseiro

Módulos estándar

Nombre	Descripción
functools	Permite funciones de orden superior y optimización con caché
operator	Proporciona funciones rápidas para operaciones matemáticas y lógicas
statistics	Contiene funciones estadísticas como media, mediana y desviación estándar
logging	Proporciona herramientas para registrar eventos y errores en aplicaciones
argparse	Facilita el manejo de argumentos en la línea de comandos
shutil	Permite la manipulación de archivos y directorios (copiar, mover, eliminar)
socket	Soporta la comunicación en red mediante sockets
threading	Permite ejecutar múltiples hilos de forma concurrente
multiprocessing	Soporta la ejecución de múltiples procesos en paralelo
subprocess	Permite ejecutar comandos del sistema y capturar su salida
tkinter	Biblioteca estándar de Python para interfaces gráficas (GUIs)

Instituto Balseiro

Módulos externos

Nombre	**Descripción**
numpy	Librería para cálculo numérico y manipulación de arrays multidimensionales
pandas	Facilita el análisis y manipulación de datos con estructuras como DataFrames
matplotlib	Biblioteca para crear gráficos y visualizaciones
seaborn	Extensión de matplotlib con estilos más avanzados para visualización de datos
scipy	Contiene herramientas avanzadas de cálculo científico y optimización
scikit-learn	Librería para Machine Learning con algoritmos de clasificación, regresión y clustering
tensorflow	Framework para Machine Learning y Deep Learning desarrollado por Google
torch	Biblioteca de Deep Learning desarrollada por Meta (PyTorch)
opencv-python	Procesamiento de imágenes y visión artificial
pillow	Biblioteca para manipulación y procesamiento de imágenes

Todos estos módulos deben instalarse con **pip install** *nombre_del_modulo* antes de ser utilizados.

Instituto Balseiro

Referencias

- A Tour of C++ Third Edition - Bjarne Stroustrup
- Modern C++ for Absolute Beginners - Slobodan Dmitrović
- `https://en.cppreference.com/`
- `https://hackingcpp.com/`
- `https://docs.python.org/`
- `https://mouredev.pro/recursos`

Instituto Balseiro

Q&A

Instituto Balseiro