



# El lenguaje C++



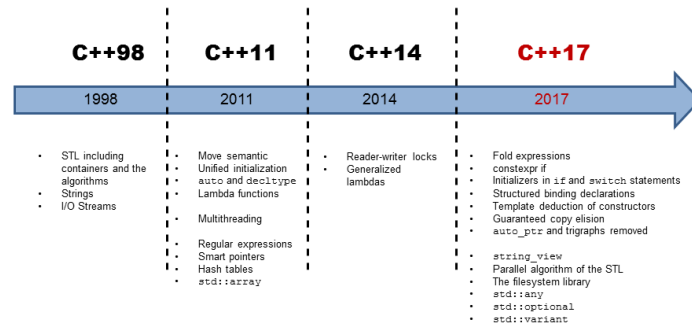
**Bach. Rodolfo Mercado Gonzales**  
**Programación Competitiva UPC**

# Conceptos para competencias

The C++ logo, featuring a red 'C' with two red '+' signs, all set against a yellow square background with rounded corners.

# Versiones estándar de C++

- ❑ **C++ 98:** tiene la mayor parte de C++ que usaremos.
- ❑ **C++ 11:** nos brinda facilidades como “auto” y “range-based for”.
- ❑ **C++ 14**
- ❑ **C++ 17**



# Programa básico

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6      cout << "Hello world";
7      return 0;
8  }
```

directivas para el preprocesador

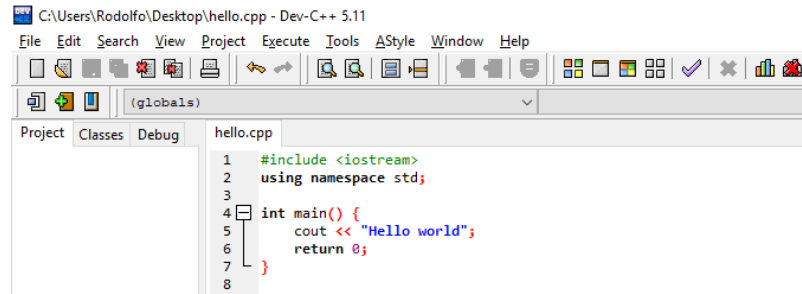
namespace

función principal

instrucciones en C++

# Ambiente de trabajo

## Dev C++



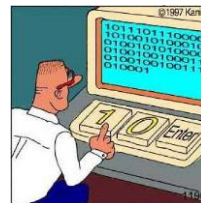
The screenshot shows the Dev-C++ IDE interface. The title bar indicates the file path: C:\Users\Rodolfo\Desktop\hello.cpp - Dev-C++ 5.11. The menu bar includes File, Edit, Search, View, Project, Execute, Tools, AStyle, Window, and Help. The toolbar contains various icons for file operations, editing, and execution. The left sidebar has tabs for Project, Classes, and Debug. The main editor window displays the code for hello.cpp, which includes a preprocessor directive, namespace declaration, and a main function that prints "Hello world".

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     cout << "Hello world";
6     return 0;
7 }
8
```

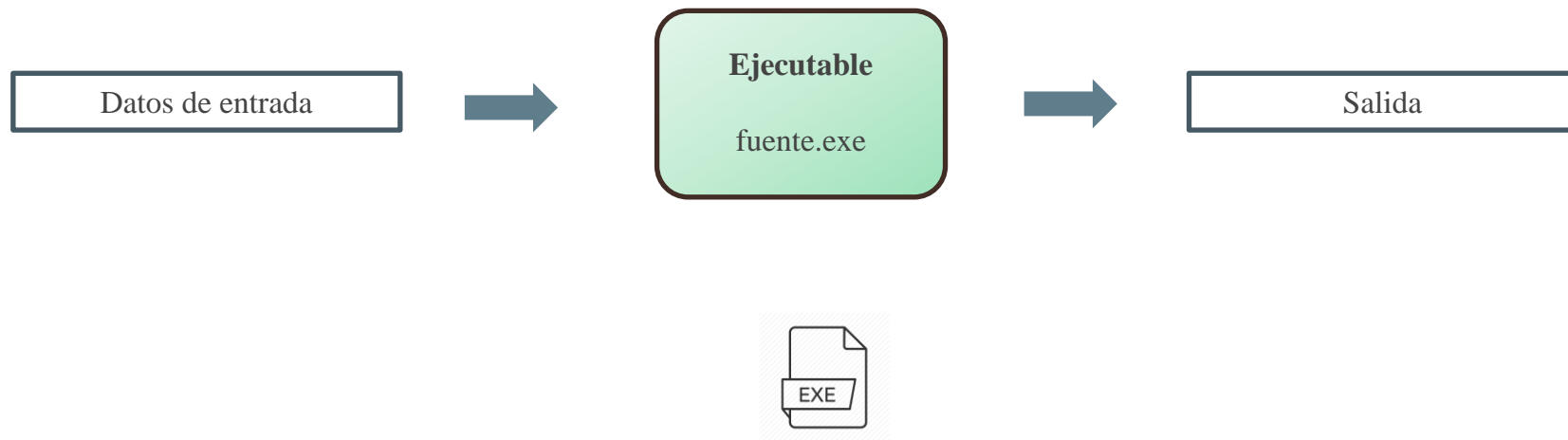
# Compilación



```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6      cout << "Hello world";
7      return 0;
8  }
```



# Ejecución



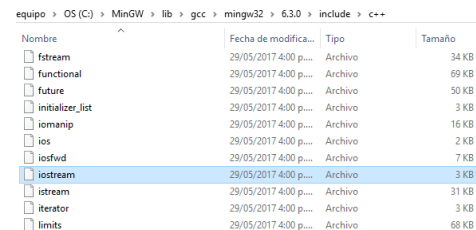
# Preprocesador

- ❑ Procesa directivas (comandos) antes de la compilación real de un programa.
- ❑ Las directivas inician con el símbolo `#` y terminan con el final de línea.
- ❑ Utiliza una sintaxis diferente a C++.
- ❑ Las directivas más comunes son `#include` y `#define`.



# Directiva #include

- ❑ Directiva que le indica al preprocesador incluir al programa actual el contenido de otro archivo.
- ❑ Los archivos que se incluyen en otros programas se denominan **headers** y son necesarios para usar ciertas funcionalidades.
- ❑ Por ejemplo **#include <iostream>** le dice al preprocesador que copie el contenido del archivo **iostream**, el cual nos permite manejar el ingreso y salida de datos.



equipo > OS (C:) > MinGW > lib > gcc > mingw32 > 6.3.0 > include > c++

Nombre	Fecha de modifica...	Tipo	Tamaño
fstream	29/05/2017 4:00 p...	Archivo	34 KB
functional	29/05/2017 4:00 p...	Archivo	69 KB
future	29/05/2017 4:00 p...	Archivo	50 KB
initializer_list	29/05/2017 4:00 p...	Archivo	3 KB
iomanip	29/05/2017 4:00 p...	Archivo	16 KB
ios	29/05/2017 4:00 p...	Archivo	2 KB
iosfwd	29/05/2017 4:00 p...	Archivo	7 KB
<b>iostream</b>	29/05/2017 4:00 p...	Archivo	<b>3 KB</b>
istream	29/05/2017 4:00 p...	Archivo	31 KB
iterator	29/05/2017 4:00 p...	Archivo	3 KB
limits	29/05/2017 4:00 p...	Archivo	68 KB

# ⟨bits/stdc++.h⟩

- ❑ Header especial que incluye a todos los headers del estándar C++.
- ❑ Solo esta presente en el compilador G++.
- ❑ En las competencias de programación se usa el compilador G++.

# Directiva #define

- ❑ Define las denominadas macros, las cuales nos permiten manejar sustitución de términos en el código.
- ❑ La macros tienen el siguiente formato: `#define macro reemplazo`
- ❑ El preprocesador busca todas la ocurrencias de `macro` en el programa y las sustituye por `reemplazo`, antes de iniciar la compilación real.

```
#define MAX_N 10
```

```
#define FOR(i, a, b) for(int i = a; i < b; ++i)
```

# Namespace

- ❑ Para evitar homonimia en los nombres de las variables o funciones, en C++ se define el espacio de nombres (namespace).
- ❑ Todas las variables y funciones predeterminadas en C++ se encuentra en el namespace denominado `std`.

```
namespace std{  
    istream cin;  
    ostream cout;  
    ...  
}
```

```
#include <bits/stdc++.h>  
using namespace std;  
  
namespace spanish {  
    void print() {  
        cout << "hola";  
    }  
}  
  
namespace english {  
    void print() {  
        cout << "hello";  
    }  
}  
  
int main() {  
    english :: print();  
    return 0;  
}
```

# Función main

- ❑ Es la primera función a ser llamada, el resto de funciones son llamadas directa o indirectamente por ésta.
- ❑ El retornar 0 indica que el programa terminó con éxito.

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

# Variables y constantes

```
char letra;  
string nombre;  
bool flag;  
int edad;  
long long nro_conexiones;  
float costo;  
double area;
```

```
const int MESES = 12;  
const int DIAS_SEMANA = 7;  
const double PI = acos(-1.0);
```

# Lectura y escritura

- ❑ La entrada estándar viene desde teclado.
- ❑ La salida estándar se muestra en consola.
- ❑ Ambos pueden ser redireccionados mediante línea de comandos.

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a, b, c;
    cin >> a >> b >> c; //lectura
    c = a + b;
    cout << c; //escritura
    return 0;
}
```

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a, b;
    //lectura hasta fin de archivo
    while (cin >> a >> b) {
        cout << a + b << "\n";
    }
    return 0;
}
```

# Lectura y escritura rápida

- ❑ Desactivamos la sincronización de `cin` y `cout` con `stdio`
- ❑ Evitamos un `flush` del buffer cada vez que se hace un `cin`.
- ❑ “`endl`” realiza un `flush` del buffer, en su lugar usaremos “`\n`”.

```
#include <bits/stdc++.h>
#define fast_io ios::sync_with_stdio(false); cin.tie(NULL)
using namespace std;

int main() {
    fast_io;
    int x;
    for (int i = 0; i < 3; ++i) {
        cin >> x;
        cout << x * x << "\n";
    }
}
```



# Escritura de números reales

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    double a, b;
    cin >> a >> b;
    cout << fixed;
    cout << setprecision(2) << a << "\n";
    cout << setprecision(2) << b << "\n";
    return 0;
}
```

# Estándares de codificación / condicionales

```
if (condition) { // no spaces inside parentheses
    ...
} else if (...) { // The else goes on the same line as the closing brace.
    ...
} else {
    ...
}
```

Short conditional statements may be written on one line if this enhances readability. You may use this only when the line is brief

```
if (x == kFoo) return new Foo();
if (x == kBar) return new Bar();
```

This is not allowed when the if statement has an else:

```
// Not allowed IF statement on one line when there is an ELSE clause
if (x) DoThis();
else DoThat();
```

# Condicionales / if-else

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int mi_edad, tu_edad;
    cin >> mi_edad >> tu_edad;
    if (mi_edad < tu_edad) {
        cout << "soy menor";
    } else {
        cout << "soy mayor o tenemos la misma edad";
    }
    return 0;
}
```

# Estándares de codificación / repetitivas

Braces are optional for single-statement loops.

```
for (int i = 0; i < kSomeNumber; ++i)
    printf("I love you\n");

for (int i = 0; i < kSomeNumber; ++i) {
    printf("I take it back\n");
}
```

```
while (condition) {
    // Repeat test until it returns false.
}
```

# Repetitivas / for y range-based for

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    for (int i = 0; i <= 10; ++i) {
        cout << i << endl;
    }
    return 0;
}
```

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int arr[] = {3, 6, 2, -1, 8};
    for (int elem : arr) {
        cout << elem << endl;
    }
    return 0;
}
```

# Repetitivas / while

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int i = 0;
    while (i <= 10) {
        cout << i << endl;
        i = i + 1;
    }
    return 0;
}
```

# Auto

Deduce el tipo de dato de una variable, a partir de la expresión usada para su inicialización.

```
//auto
auto cadena = "hola";
auto real = 5;
cout << "type: " << typeid(real).name() << endl;
```

```
int arr[] = {5, 10, 6};
//auto en for basado en rango
for (auto elem : arr) cout << elem << endl;

//auto&
for (auto& elem: arr) elem++;
for (int elem : arr) cout << elem << endl;
```

# Pair

- ❑ Estructura que agrupa un par de elementos, los cuales pueden ser de distintos tipos de datos.
- ❑ Los operadores de comparación ya se encuentran definidos en los pares.

```
pair<string, int> p1, p2, p3; //declaración

p1.first = "Lima", p1.second = 100; //asignación
p2 = make_pair("Surco", 90); //asignación
p3 = {"Jesus Maria", 81}; //asignación

pair<int, string> p4(45343454, "Kun"); //declaración con asignación

cout << p3.first << " " << p4.second; //impresión
```



# Tuple

- ❑ Estructura que agrupa un conjunto de elementos, los cuales pueden ser de distintos tipos de datos.
- ❑ Los operadores de comparación ya se encuentran definidos en las tuplas.

```
tuple<string, int, double> p1, p2, p3; //declaración

get<0>(p1) = "Kun", get<1>(p1) = 24, get<2>(p1) = 65.7; //asignación
p2 = make_tuple("Dev", 22, 73.2); //asignación
p3 = {"Nek", 21, 72.3}; //asignación

tuple<string, int, double> p4("Jaz", 20, 63.1); //declaración con asignación

cout << get<0>(p4) << " " << get<1>(p2); //impresión
```

# Struct

- ❑ Estructura que agrupa un conjunto de variables (las cuales pueden ser de distintos tipos de datos) y funciones.
- ❑ Si es necesario hacer comparaciones, se requiere definir el operador `<`.

```
struct Tupla {  
    string name;  
    int age;  
    double w;  
    Tupla(){}  
    Tupla(string _name, int _age, int _w) {  
        name = _name, age = _age, w = _w;  
    }  
    bool operator < (Tupla aux) {  
        if (name != aux.name) return name < aux.name;  
        if (age != aux.age) return age < aux.age;  
        return w < aux.w;  
    }  
};
```

```
Tupla p1("Kun", 23, 70.2), p2;  
p2 = Tupla("Kun", 22, 62.5);  
if (p1 < p2) {  
    cout << "si";  
} else {  
    cout << "no";  
}
```

# Sort

- ❑ Ordena de manera ascendente un conjunto de elementos de un arreglo o contenedor.
- ❑ El tipo de dato de los elementos debe tener definido el operador  $<$  , o en su defecto se debe definir una función de comparación para el `sort`.

```
bool comp(int x, int y) {  
    return abs(x) < abs(y);  
}  
  
int main() {  
    int A[] = {3, -2, 4, 1, -5, 10};  
    int n = 6;  
    //ordenamiento por valor absoluto  
    sort(A, A + n, comp);  
    for (auto elem : A) cout << " " << elem;  
    return 0;  
}
```

# Problemas ad hoc



# Problemas Ad Hoc

- ❑ Problemas que al intentar ser clasificados no encajan en ninguna “categoría estándar”.
- ❑ No requerimos de grandes algoritmos para resolverlos.
- ❑ Basta conocer lo básico de un lenguaje de programación.

# ¿Cómo afrontarlos?

- ❑ Leer cuidadosamente la descripción del problema.
- ❑ Muchas veces solo tenemos que hacer una “simulación” o “implementación”.
- ❑ Tener paciencia si la implementación es extensa.
- ❑ El tiempo límite generalmente no es inconveniente.

# Ejercicios

- [Codeforces – Watermelon](#)
- [AtCoder – Five Antennas](#)
- [UVA – Automated Checking Machine](#) (ICPC 2014)
- [AtCoder – Great Ocean View](#)
- [Codeforces – Domino Piling](#)



# Desafíos

- [AtCoder – Buttons](#)
- [Codechef – Chef and Price Control](#)
- [AtCoder – Ordinary Number](#)
- [UVA – Identifying tea \(ICPC 2015\)](#)
- [HackerRank – Breaking the Records](#)
- [HackerRank – Diagonal Difference](#)
- [HackerRank – Staircase](#)
- [Codeforces – Theatre Square](#)
- [UVA – Bingo! \(ICPC 2010\)](#)

**CHALLENGE ACCEPTED**







# Lectura de strings

```
string s; //declaracion
s = "hola mundo"; //asignacion
cin >> s; //lee un token (hasta un ' ' o '\n')
getline(cin, s); //lee un línea
```

¡cuidado al combinar cin y getline!

```
int tc;
string s;
cin >> tc; // deja un salto de línea
cin.ignore(); //ignora el salto de línea
getline(cin, s);
```

# Lectura de strings

leer líneas hasta fin de archivo

```
string s;
while (getline(cin, s)) {
    cout << s << "\n";
}
```

leer caracteres hasta fin de archivo  
(no incluye espacio ni salto de línea)

```
char c;
while (cin >> c) {
    cout << c;
}
```

leer todos los caracteres hasta fin de archivo

```
char c;
while (cin.get(c)) {
    cout << c;
}
```

# Operaciones con strings

```
/*concatenación*/  
string s1 = "hola", s2 = "mundo";  
string texto = "";  
texto += s1; //concat. un string  
texto += ' '; //concat. un char  
texto = texto + s2;
```

```
/*substrings*/  
string s = "programa";  
string s1 = s.substr(3); //grama  
string s2 = s.substr(2, 4); //ogra
```

```
/*conversiones*/  
int x = 213;  
string s = to_string(x); // int a string  
x = stoi(s); // string a int  
  
/*otros*/  
reverse(s.begin(), s.end()); // invertir  
sort(s.begin(), s.end()); //ordena caracteres
```

# Código ASCII

Es la representación numérica de cada uno de los 256 caracteres disponibles en el tipo de dato char.

```
/*código ascii*/
int ascii = 'a';
char c = 48;

/*conversiones*/
int d = '5' - '0'; // char a dígito
char c = 5 + '0'; //dígito a char
```

CHAR	ASCII
0	48
@	64
A	65
a	97

```
//minúscula y mayúscula
char c1 = toupper('a'); //a mayúscula
char c2 = 'a' - 32; //a mayúscula
char c3 = tolower('A'); //a minúscula
char c4 = 'A' + 32; //a minúscula
bool flag1 = islower('a'); //valida minúscula
bool flag2 = isupper('A'); //valida mayúscula
```

# Stringstream

Buffer interno (string) que simula comportarse como un archivo.

```
string texto = "grupo de programacion competitiva";
int anio = 2020;
ostringstream oss;
oss << texto << " " << anio;
cout << oss.str();
```

```
string texto = "grupo de programacion competitiva";
istringstream iss(texto);
string token;
while (iss >> token) {
    cout << token << "\n";
}
```

# Ejercicios

- [AtCoder – Fifty-Fifty](#)
- [Codeforces – Boy or Girl](#)
- [UVA – Guessing Game](#)
- [UVA – Flowers Flourish from France](#) (ICPC 2010)



# Desafíos

- [AtCoder – Palindrome-phia](#)
- [AtCoder – ROT N](#)
- [Codeforces – Word](#)
- [Codechef – Chef and String](#)
- [Codechef – Chef and Card Game](#)
- [UVA – Reverse and Add](#)
- [UVA – TEX Quotes](#)
- [UVA – Mother bear](#)
- [Codechef – Fancy Quotes](#)

**CHALLENGE ACCEPTED**





# Referencias

- ❑ Steve Oualline - Practical C++ programming
- ❑ HackerEarth - [The Build Process-C/C++](#)
- ❑ Google - [Google C++ Style Guide](#)
- ❑ Codeforces - [C++ Tricks](#)
- ❑ Steven Halim & Felix Halim - Competitive Programming 3



“Opportunities don’t happen, you create them.”

- Chris Grosser