

```
<!-- PhyloGenomicsUD -->
```

Introducción a C++ {

```
<Por="Santiago Avila Gómez"/>
```

}



Contenidos

- 01 ¿Por qué C++?
- 02 Estructura básica
- 03 Sintaxis básica
- 04 Tipos de datos
- 05 Operadores y expresiones
- 06 Estructuras de control
- 07 Funciones

Prerequisites {

1. Actualizar nuestro sistema Ubuntu/
Debian



```
sudo apt update && sudo apt upgrade -y
```

2. Instalar compilador de C++ (G++)



```
sudo apt install build-essential -y
```

3. Instalar un editor de texto, en
este caso **nano**



```
sudo apt install nano -y
```

4. Verificar instalación del
compilador

```
santy-avila@DESKTOP-H08SV6A ~$ g++ --version
g++ (Debian 14.2.0-12) 14.2.0
Copyright (C) 2024 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

}

¿Por qué C++? {

- Es un lenguaje compilado y de bajo nivel controlado por el programador.
- Excelente control sobre memoria, ejecución y paralelismo.
- Altamente optimizable con herramientas modernas.
- Es base para OpenMP, MPI, CUDA y otras tecnologías paralelas.
- Muy usado en supercomputación, ciencia de datos a gran escala, simulaciones físicas, modelado, etc.



}

Compilado o Interpretado {

Lenguajes Interpretados

Como Python, Matlab y Julia, en los cuales el código (En tiempo de Ejecución) en **bytecódigo** (una representación intermedia) que luego es ejecutado por un **intérprete** en la máquina.

Lenguajes Compilados

Como C, C++ y Rust, en cambio utilizan un **compilador** para traducir directamente el código fuente a un **binario** que puede ser ejecutado por la máquina.

Un código compilado se ejecuta más rápido al ser traducido directamente a lenguaje máquina, permite optimizaciones para mejorar rendimiento y reduce errores mediante análisis estático antes de su ejecución.

}

Sintaxis básica {

El formato para escribir un programa en C++ se llama su **estructura**.

Consiste en las siguientes partes:

- **Directiva del preprocesador**
- **Función main()**
- **Cuerpo del programa**

!!! PONGAMOSLO A PRÁCTICA CON
"HOLA MUNDO" !!!

Directiva del preprocesador

Instrucción que se le da al compilador antes de la ejecución del programa real. También se la conoce como directiva del compilador. Las directivas del preprocesador comienzan con el símbolo de numeral #.

Funcion main()

La función main() es el punto de inicio de un programa en C++.

Si un programa no contiene la función main(), puede compilarse, pero no puede ejecutarse.

Cuerpo del programa

El cuerpo del programa en C++ es la parte dentro de main() donde se escriben las instrucciones que el programa ejecuta.

1 byte = 8 bits

Tipos de datos primitivos {

Los datos primitivos permiten crear información, como **enteros, caracteres, números decimal, cadenas de textos, etc.**

Nota:

El rango de números que acepta un tipo de dato con signo puede ser encontrado con la formula:

$$Rango = [-2^{n-1}, 2^{n-1} - 1]$$

Para enteros **sin signo**(unsigned), el rango es:

$$Rango = [0, 2^n - 1]$$

Donde:

- *n* es la cantidad de bits del tipo de dato

Tipos primitivos		
Tipo	Significado	Tamaño
bool	booleano	indefinido
char	character	8 bits
wchar_t	wide char	16 bits
char16_t	unicode char	16 bits
char32_t	unicode char	32 bits
short	entero corto	16 bits
int	entero	16 bits o 32 bits (Por sistema)
long	entero largo	32 bits
long long	entero muy largo	64 bits
float	Precisión simple (coma flotante)	32 bits
double	Precisión doble	64 bits
long double	Precisión extendida (doble/doble extendida)	80 bits a 128 bits (Por arquitectura)

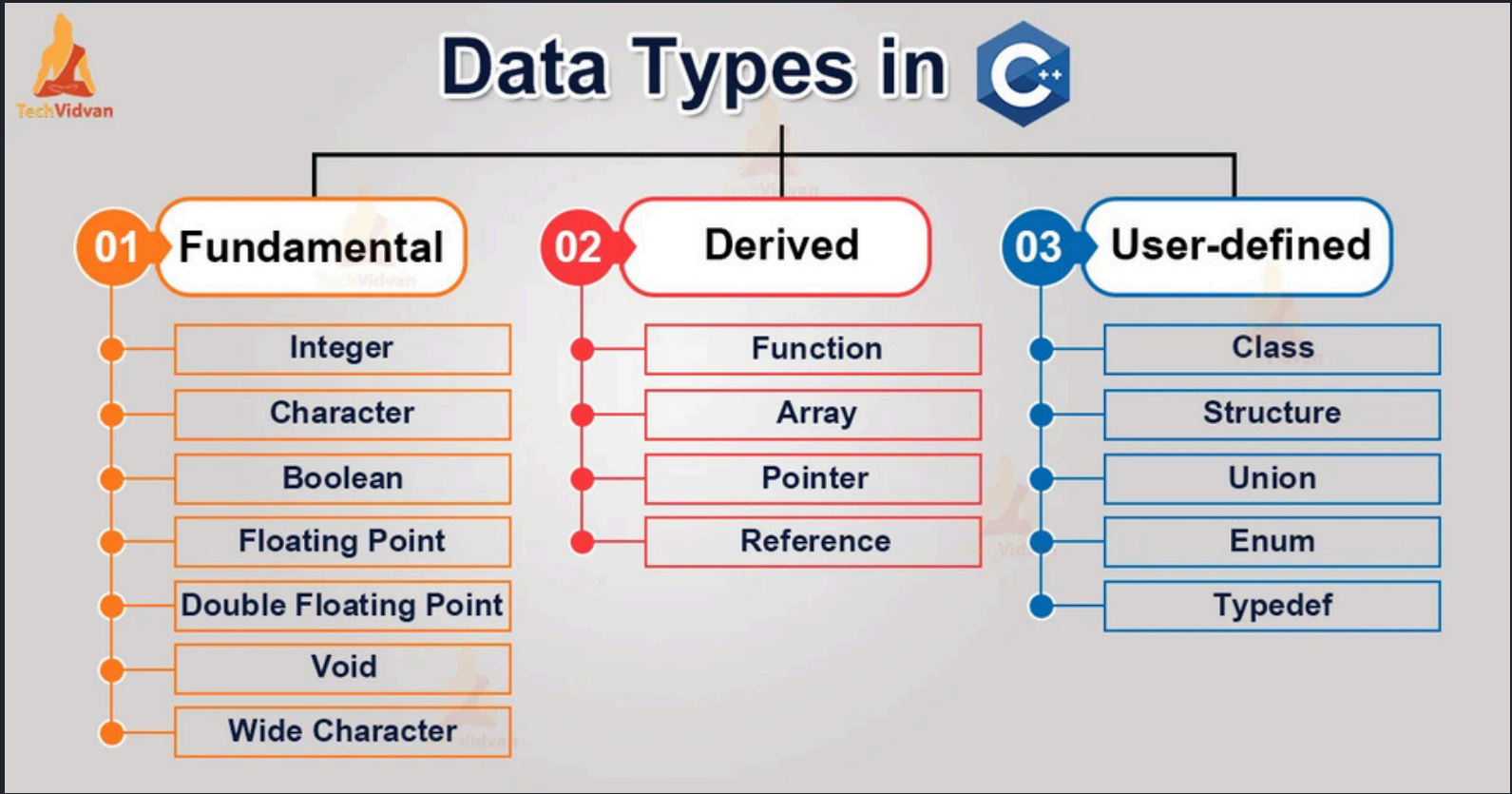
}

Tipos de datos derivados {

Los tipos derivados se construyen a partir de tipos primitivos para formar estructuras más complejas, como arreglos, punteros, funciones etc.

Tipos derivados		
Tipo	Significado	Descripcion
Array	Arreglo	Colección de elementos del mismo tipo, almacenados en posiciones contiguas.
Pointer	Puntero	Almacena la dirección de memoria de otra variable.
Reference	Referencia	Alias de una variable existente; no ocupa nueva memoria.
Function	Función	Conjunto de instrucciones que puede recibir y retornar valores.
String	Arreglo de caracteres	Secuencia de caracteres, como palabras o frases.

Los tipos de datos definidos por usuario:
Además de los tipos primitivos y derivados, C++ permite definir tipos de datos propios, conocidos como tipos definidos por el usuario, como struct, class y union.



}

Ejercicio {

El ADN está compuesto por cuatro bases nitrogenadas: **adenina (A)**, **timina (T)**, **citosa (C)** y **guanina (G)**, que pueden representarse como caracteres (***char***) ocupando 1 byte cada uno. Supongamos que estamos desarrollando un programa en C++ para almacenar una cadena de ADN de 1 millón de bases (1,000,000 de letras A, T, C, G).

1. ¿Cuántos bytes ocupará esta cadena si se almacena en un arreglo de char?

2. Si decidimos optimizar el uso de memoria usando solo los bits necesarios, ¿cuántos bits necesitas por base?

3. ¿Qué tipo de dato sería más adecuado para almacenar 4 bases en una sola variable?

}

Variables y Declaración{

En C++, las variables son espacios en memoria que almacenan datos. Antes de usar una variable, debemos declararla, indicando su tipo y nombre. Esto permite al compilador reservar el espacio adecuado y manejar el valor correctamente.

Declaración:



```
int edad;      // Variable de nombre edad y tipo entero
float altura;  // Variable de nombre altura y tipo coma flotante
char letra;    // Variable de nombre letra y tipo caracter
```

Inicialización:



```
int edad = 25;
```

Reglas para nombres de variables:

- Deben comenzar con letra o guion bajo
- No usar palabras reservadas
- Sensibles a mayúsculas y minúsculas (*edad* != *Edad*)

}

Operadores y Expresiones {

Las **expresiones** combinan variables, constantes y operadores para producir un nuevo valor. Los **operadores** realizan operaciones sobre datos, como suma, comparación o asignación.

Tipos de operadores comunes:

- **Aritméticos:** +, -, *, /, %
- **Asignación:** =, +=, -=, *=, /=
- **Relacionales:** ==, !=, <, >, <=, >=
- **Lógicos:** && (y), || (o), ! (no)
- **Incremento/Decremento:** ++, --





```
int a = 5, b = 3;  
int suma = a + b;           // suma = 8  
bool esMayor = a > b;       // true  
a += 2;                      // a = 7
```

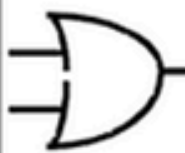
}

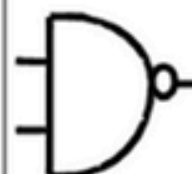
Lógica Booleana y Manejo de bits{

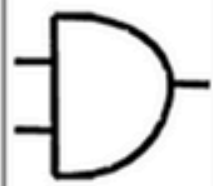
La lógica booleana es la base para la toma de decisiones en programación. Trabaja con valores **true** (verdadero) y **false** (falso), que se representan internamente como **1** y **0** respectivamente. Además, el manejo de bits permite operar a nivel muy bajo con los datos, lo que es esencial para optimizaciones.

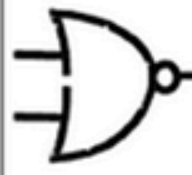
NOT	A	Salida
	0	1
	1	0

XOR	A	B	Salida
	0	0	0
	0	1	1
	1	0	1
	1	1	0

AND	A	B	Salida
	0	0	0
	0	1	0
	1	0	0
	1	1	1

NAND	A	B	Salida
	0	0	1
	0	1	1
	1	0	1
	1	1	0

OR	A	B	Salida
	0	0	0
	0	1	1
	1	0	1
	1	1	1

NOR	A	B	Salida
	0	0	1
	0	1	0
	1	0	0
	1	1	0

Compuertas logicas de lógica booleana

Ejemplo:

```
bool x = true, y = false;
bool resultado = x && y; // false

int a = 5;           // 0101 en binario
int b = 3;           // 0011 en binario
int c = a & b;        // 0001 (1 en decimal)
```



Estructuras de Control {

Las estructuras de control permiten modificar el flujo de ejecución de un programa, en función de condiciones o repeticiones.

Tipos principales:

1. **Condicionales (tomar decisiones)**
2. **Bucles (repetir instrucciones)**
3. **Control de flujo (alterar el orden de ejecución)**

Condicional if:

```
if (x > 0) {  
    cout << "Positivo";  
} else if (x < 0) {  
    cout << "Negativo";  
} else {  
    cout << "Cero";  
}
```

Condicional switch:

```
int opcion = 2;  
switch (opcion) {  
    case 1: cout << "Uno"; break;  
    case 2: cout << "Dos"; break;  
    default: cout << "Otra opción";  
}
```

Bucle for: Compara **Antes**

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

Bucle while: Compara **Antes**

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

Bucle do-while: Compara **Despues**

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

}

Funciones {

Una función es un bloque de código reutilizable que realiza una tarea específica. Permite organizar el programa en partes más manejables y evita repetir código.

Estructura de una función:

```
tipo_de_retorno nombre_funcion(parámetros) {  
    // cuerpo de la función  
    return valor; // si aplica  
}
```

Donde:

- **tipo_de_retorno** es el tipo de dato que vamos a retornar en la función
- **nombre_funcion** es el nombre de nuestra función
- **parametros** es la entrada que va a recibir nuestra función (Cada entrada debe ser de un tipo de dato)

Ejemplo: Función que recibe un aminoácido y devuelve su nombre completo



```
string obtenerAminoacido(char codigo) {  
    if (codigo == 'A') return "Alanina";  
    else if (codigo == 'C') return "Cisteína";  
    else if (codigo == 'G') return "Glicina";  
    else if (codigo == 'T') return "Treonina";  
    else return "Código desconocido";  
}
```

Para llamar a nuestra función hacemos:



```
int main() {  
    char codigo = 'A';  
    string aminoacido = obtenerAminoacido(codigo);  
    std::cout<<aminoacido<<endl;  
}
```


Gracias {

<Por="Santiago Avila Gómez"/>

}