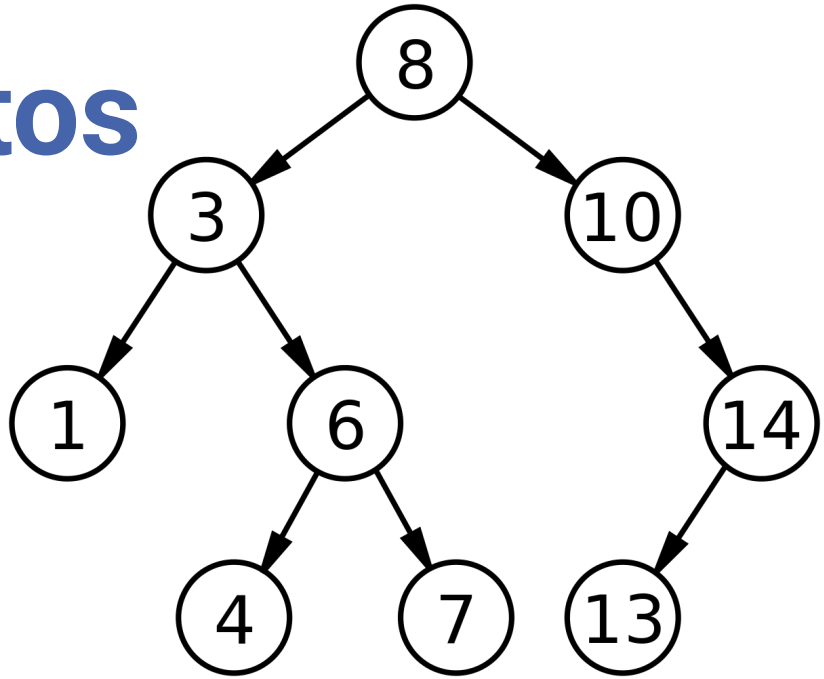
Estructuras de datos



Introducción a C++

Tipos de datos, operadores, estructuras de control, arreglos, structs, punteros, memoria dinamica, strings y archivos

Franco Cerda

Créditos: Sebastián Torrealba

Lenguaje de programación C++

Es un lenguaje de programación diseñado en 1979 por *Bjarne Stroustrup*. La intención de su creación fue extender al lenguaje de programación *C* y añadir mecanismos que permiten la manipulación de objetos.

¿Cómo programar en C++?

A diferencia de python usaremos un compilador llamado **GNU C++**, tambien conocido como **g++**

Se usa el siguiente comando:

```
g++ programa.cpp -o compilado
```

¡Importante!

Tienen que instalar alguna distribución **Linux** que les permita compilar en la terminal usando **g++**, ya que el no usar **g++** no asegura que su tarea pueda compilar a los ayudantes

Primer programa en C++

Normalmente el primer programa que uno hace es el siguiente:

```cpp
#include<iostream>

using namespace std;

int main() {
    cout << "Hola mundo" << endl;
    return 0;
}
```

Variables y tipos de datos

Una variable en programación es un espacio de almacenamiento que tiene un nombre simbólico (identificador) y está asociado con un valor y una ubicación en la memoria.

Un tipo de dato, por otro lado, es un atributo de una variable que le dice al compilador o al intérprete cómo el programador tiene la intención de usar la variable.

Tipos de datos primitivos

Son los tipos de datos más fundamentales que tenemos en **C++**

- **Enteros:** int, short, long, long long.
- **Flotantes:** float, double
- **Caracter:** char
- **Booleano:** bool

Operadores

Los operadores que ocuparemos de **C++** son los siguientes:

- **Aritméticos:** `+ - * / %`
- **Comparación:** `== != < <= > >=`
- **Lógicos:** `&& || !`
- **Asignación:** `= += -= *= /= %=`
- **Incremento/decremento prefijo y postfijo:** `++ --`

Estructuras de control

Permiten modificar el flujo de ejecución de un programa. En **C++** se define un bloque de código **todo** lo que esté encerrado entre llaves.

Ejemplo:

```
if (condición) {
  // Acá estaría un bloque de código
}
```

Estructuras de control

Comparando con **Python**, así se haría un *if* en **C++**

```cpp
if (condicion) {
  cout << "Hola" << endl;
}
```

```python
# Python!
if condicion:
  print("Hola!")
```

Estructuras de control

Comparando con **Python**, así se haría un *if-else* en **C++**

```cpp
if (condicion) {
  cout << "Hola" << endl;
} else {
  cout << "Hola O.o" << endl;
}
```

```python
# Python!
if condicion:
  print("Hola!")
else:
  print("Hola O.o!")
```

Estructuras de control

En **C++** no existe tal cosa de un *elif* como en **Python**, simplemente lo que se hace es hacer un **else** de una linea junto a un **if**:

```cpp
if (condicion) {
  cout << "Hola" << endl;
} else if (condicion2) {
  cout << "Holiwis" << endl;
}
```

```python
# Python!
if condicion:
  print("Hola!")
elif condicion2:
  print("Holiwis")
```

Estructuras de control

Comparando con **Python**, así se haría un *while* en **C++**

```cpp
while(condicion) {
  cout << "Holaaa" << endl;
}
```

```python
# Python!
while condicion:
  print("Holaaa")
```

Estructuras de control

C++ tiene una estructura de control que no posee **Python**, el *do-while* nos permite ejecutar el código dentro de un bloque de código y despues revisar la condición para volvera a iterar

```
do {
  cout << "Hola -_-" << endl;
} while (condicion);
```

Entrada estandar

La entrada estandar es una forma de obtener información del usuario usando la terminal, en **Python** se ocupa la función `input()`, en **C++** se ocupa la `cin`.

Ejemplo:

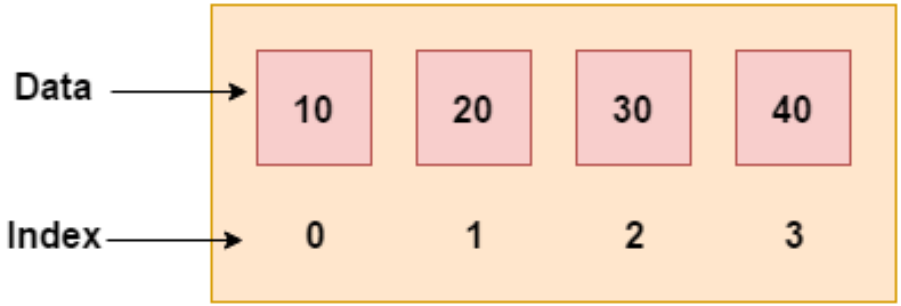
```
int numerito;
cin >> numerito
```

Esto permitira pedir un numero al usuario y guardarlo en la variable *“numerito”*.

Arreglos

Permite almacenar una colección de elementos del mismo tipo bajo un solo nombre. Cada elemento en el arreglo está identificado por un índice o posición específica.

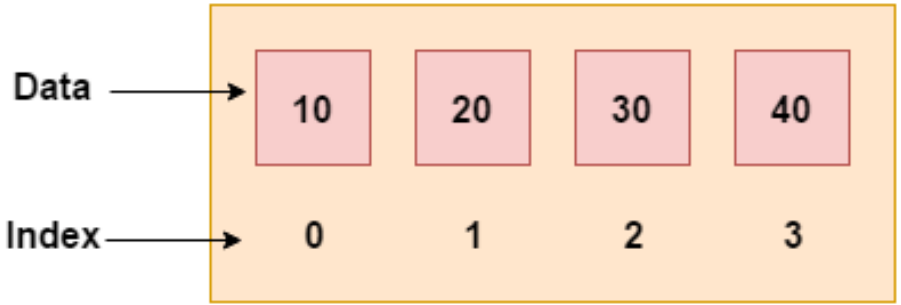
Similares a las listas de **Python**, pero tienen **grandes** diferencias.



Arreglos

Para declarar un arreglo se tiene que definir el tamaño estaticamente, es decir, el tamaño del arreglo no puede cambiar en tiempo de ejecución.

```
int data[4];
data[0] = 10;
data[1] = 20;
data[2] = 30;
data[3] = 40;
```



Arreglos

Información importante sobre los arreglos

- Se almacenan en celdas contiguas de memoria
- La dirección de memoria de un arreglo es la primera posición del arreglo
- No se pueden tener diversos tipos de datos en un mismo arreglo

Structs

Un *struct* es una forma de definir un tipo de datos que agrupa distintas variables bajo un mismo nombre.

Ejemplo:

```
// Así se define un struct
struct Nombre {
  int variable_1, variable2;
  char caracter1;
};
```

```
// Así se usa un struct
Nombre ejemplo;
ejemplo.variable_1 = 2;
ejemplo.variable_2 = 7;
ejemplo.caracer1 = 'x';
```

Punteros

Los punteros son un tipo de variable que nos permite guardar la dirección de memoria de otra variable.

Los punteros *apuntan* a una variable.

Ejemplo:

```
int var = 8;
int *ptr = &var;
(*ptr)++; // Ahora "var" tiene como valor 9
```

Punteros

Cuando estemos usando structs y queramos evitar hacer el uso de `(*ptr).propiedad` para acceder a esa variable, podemos hacer uso del operador **“arrow”**, que es un **azúcar sintactico**.

Por lo que `(*ptr).propiedad` es **exactamente lo mismo** que `ptr->propiedad`.

Azúcar sintactico

Es un término utilizado en programación para describir una característica de lenguaje que proporciona una sintaxis más concisa o amigable, pero que no añade funcionalidad fundamental.

Funciones

Para crear una función en *C++* tiene que devolver un tipo de datos pre-definido (en caso opuesto, usar void) y definir los parametros con sus respectivos tipos de dato.

Por ejemplo, si queremos crear una función que devuelva la suma de dos valores:

```
int suma(int a, int b) {
  return a + b;
}
```

Funciones

Al pasar una variable por parametro, existen distintas formas de hacerlo

- **Copia:** No modifica la variable al momento de usarla dentro de la función.
- **Puntero:** Al tener acceso a la dirección de la memoria, cambias la variable fuera de la función.
- **Referencia:** Se modifica la variable fuera de la función

Memoria dinamica

Para poder manejar manualmente la memoria podemos hacer uso de los operadores **new** y **delete**

- **new[ ]**: Crea una sección contigua de memoria y devuelve la primera posición en memoria de la sección.
- **delete [ ]**: Permite liberar memoria creada con el operador **new**, recibe la posición a liberar

¡Importante!

Si no se hace **delete** por cada **new** que se hace, tendremos memoria asignada que no podrá ser usada por el programa aunque no esté siendo utilizada, se le llama **memory leak**.

Memoria dinamica

Para crear un arreglo con tamaño dinamico, podemos usar el **new**, por ejemplo, así se crearia un arreglo de numeros enteros:

```
int n;
cin >> n; // Nos podemos dar cuenta que el valor de "n" no es constante
int* arreglo_dinamico = new int[n];
```

Para liberarlo, simplemente tendriamos que hacer:

```
delete[] arreglo_dinamico;
```

Strings

Al momento de querer manejar cadenas de caracteres, **C++** por defecto no tiene un tipo de dato que permite manejar las cadenas. A cambio, existe la libreria *<string>* que permite manejar este tipo de dato de forma eficiente.

Ejemplo:

```
#include<string> // Así se importa
string S = "Hola"; // Así se crea un string
```

Strings

Dentro de los *strings* tenemos ciertos **métodos** y operaciones que nos permite acceder a información importante, alguna de ellas son:

- **S.length():** Retorna el largo del string
- **S.empty():** Verdadero si el largo es igual a 0
- **S[i]:** Accede al caracter en la posición i
- **S + T:** Concatena dos strings
- **S == T:** Compara si dos strings son iguales
- **S < T:** Compara si **S** es menor lexicográficamente a **T**

Archivos

Al igual que en **Python** tenemos acceso a los archivos del computador en caso de que queramos leer o guardar de forma persistente.

Para hacer uso de los archivos de **C++** hay que hacer uso de la libreria *fstream*, esto nos abre la posibilidad de leer y escribir archivos.

Ejemplo:

```
#include<fstream> // Así se importa
fstream archivo("nombre.txt", ios::in); // Ejemplo de lectura
```

Archivos

Al momento de usar un archivo tenemos distintas opciones dependiendo de que operación queremos aplicar, estas son:

- **ios::in** → Lectura de datos
- **ios::out** → Escritura de datos
- **ios::app** → Escribir al final del archivo sin borrar su información previa
- **ios::binary** → Abre el archivo en modo binario

Se pueden mezclar distintas opciones ocupando el operador **or** de bits.

```
fstream archivo("archivo.txt", ios::in | ios::binary);
```

Archivos

Para leer un archivo tenemos que usar el operador **>>**.

Para escribir un archivo tenemos que usar el operador **<<**

Ejemplo:

```
archivo >> numero; // Se lee el archivo y se guarda en "numero"
archivo << "Escribiendo!"; // Se escribe el string en el archivo
```

¡Fin!