C++ II

C/C++

- Variables/Constantes
  - Tipos datos
  - Punteros
- Operadores
- **Comentarios**
- **Namespace**
- **Condicionales**
- **Bucles**
- **Funciones**
- Orientación a objetos

- Compilación

Comentarios

De una linea

```
int x = 5; // This is a comment
```

De múltiples líneas

```
/* This is a
   multi-line comment
   that spans multiple lines */
int y = 10;
```

Namespace

- Permite agrupar funciones, clases, variables etc.. bajo un prefijo
- Los espacios de nombres se utilizan para evitar conflictos de nombres entre diferentes partes de un programa que podrían usar el mismo nombre para diferentes propósitos.
- Una definición de espacio de nombres comienza con la palabra clave ‘namespace’, seguida del nombre del espacio de nombres y un conjunto de llaves que encierran los miembros del espacio de nombres.
- Para llamar un elemento que pertenece a un espacio de nombre se usa el operador ::
- Para llamar un namespace de forma global se usa la palabra “using namespace” seguido del nombre del namespace .

https://github.com/jmejiagu/BigData-UdeARootBasicCurse-20231/blob/main/codes/Unidad1/namespace.cxx

Condicionales

Los condicionales sirven para controlar el flujo de la aplicación:

If / else; switch break, default ; y el operador ternario “?”

```
if (condition) {
    // Code to be executed if the condition is true
}
```

```
if (condition) {
    // Code to be executed if the condition is true
} else {
    // Code to be executed if the condition is false
}
```

```
switch (expression) {
    case value1:
        // Code to be executed if expression == value1
        break;
    case value2:
        // Code to be executed if expression == value2
        break;
    //...
    default:
        // Code to be executed if none of the cases match
}
```

```
condition ? expression1 : expression2;
```

Condicionales Ejemplos

https://github.com/jmejiagu/BigData-UdeARootBasicCurse-20231/tree/main/codes/Unidad1

Bucles

Permiten ejecutar un bloque de código de forma repetida

for: permite iterar un número específico de veces

```
for (initialization; condition; increment) {
    // Code to be executed in each iteration
}
```

```
for (int i = 1; i <= 5; i++) {
    cout << i << endl;
}
```

Bucles

while: permite iterar mientras una condición sea cierta, no necesariamente se sabe cuántas iteraciones se deben ejecutar

do-while: el código es ejecutado antes de comprobar la condición

```
while (condition) {
    // Code to be executed in each iteration
}
```

```
int i = 1;
while (i <= 5) {
    cout << i << endl;
    i++;
}
```

```
do {
    // Code to be executed in each iteration
} while (condition);
```

```
int i = 1;
do {
    cout << i << endl;
    i++;
} while (i <= 5);
```

Bucles

Las sentencias de control de bucles con break y continue.

break: rompe la ejecución del bucle de forma prematura sin terminar de ejecutar las demás órdenes

continue: permite saltar la ejecución de un bloque de código y continúa con la siguiente iteración.

```
for (int i = 1; i <= 10; i++) {
    if (i == 5) {
        break;
    }
    cout << i << endl;
}
```

```
for (int i = 1; i <= 10; i++) {
    if (i == 5) {
        continue;
    }
    cout << i << endl;
}
```

Bucles ejemplos

https://github.com/jmejiagu/BigData-UdeARootBasicCurse-20231/tree/main/codes/Unidad1

Funciones

Una función en C++ es un bloque de código que realiza una tarea específica y puede ser llamado desde cualquier parte del programa. Las funciones se utilizan para dividir un programa en piezas más pequeñas y manejables, lo que hace que el código sea más legible y más fácil de mantener.

Es importante tener en cuenta que las funciones deben ser declaradas antes de ser utilizadas en el código principal, o bien definidas antes de la función que las llama. Esto se hace para que el compilador sepa de antemano cómo se definirá la función cuando se llame a ella.

Las funciones son una parte esencial de la programación en C++, ya que permiten una mayor modularidad y reutilización del código.

Funciones

```
tipo_de_retorno nombre_de_función(parámetros){
   // Cuerpo de la función
   return valor_de_retorno;
}
```

Donde:

- **tipo_de_retorno**: es el tipo de valor que devuelve la función. Puede ser un tipo de dato básico, como **int, float, double, char, etc**., o un tipo definido por el usuario, como una clase o estructura.
- **nombre_de_función**: es el nombre que se le da a la función, que se utiliza para llamarla desde el código.
- **parámetros**: son los valores que se pasan a la función para que los utilice en su ejecución. Pueden ser de cualquier tipo de dato, separados por comas, y pueden ser opcionales.
- **cuerpo_de_la_función**: es el conjunto de instrucciones que se ejecutan cuando se llama a la función. Aquí es donde se define la tarea que realiza la función.
- **valor_de_retorno**: es el valor que devuelve la función al final de su ejecución. Este valor es opcional y puede ser de cualquier tipo de dato que coincida con el tipo de retorno de la función

Funciones

Hay funciones en C++ que no necesitan devolver un valor, ya que simplemente realizan una tarea y no requieren que se devuelva un resultado. Estas funciones se llaman funciones void

```
int sumar(int x, int y) {
   int resultado = x + y;
   return resultado;
}
```

```
void nombre_de_funcion(parámetros){
   // Cuerpo de la función
}
```

```
int a = 5;
int b = 7;
int c = sumar(a, b); // c tendrá el valor 12
```

```
// Función para imprimir un mensaje por pantalla
void imprimirMensaje(string mensaje) {
   cout << mensaje << endl;
}
```