

An abstract graphic on the left side of the slide. It features a light gray, semi-circular background. Inside this shape, there is a complex network of thin, dark gray lines connecting various colored dots. The dots are in shades of blue, orange, yellow, purple, and black. Some dots are larger than others. The overall effect is that of a complex, interconnected system, possibly representing a network or a molecular structure.

CURSO PROGRAMACIÓN

PROGRAMACIÓN GENERAL EN C++

¿QUÉ ES PROGRAMAR?

- Programar es dar instrucciones a una máquina.
- Esta las seguirá al pie de la letra y las ejecutará.
- Permite realizar y optimizar muchas tareas cotidianas.
- Existen muchos lenguajes para escribirle a la máquina, cada uno con sus beneficios y potencias.
- C++ es un lenguaje padre que proviene de C y se relaciona bastante con la máquina.
- C++ es didáctico y permite comprender la programación completamente, lo que hace aprender los demás lenguajes de una forma muy sencilla.

¿QUÉ SE NECESITA SABER PARA PROGRAMAR?

- Solo las ganas de aprender.
- Cualquiera puede programar.
- Lógica y simplicidad.
- Orden y práctica, resolución de ejercicios.
- Repetición y repetición.

PROGRAMA PRINCIPAL

- Nuestro programa principal se compone de una librería para hacer funcionar lo que escribamos (1), un código para evitar escribir la palabra "std" (2) y una función principal (3).
- Dentro de la función principal "main" se ejecutará todo nuestro código que se muestre por pantalla, si escribimos código fuera no se ejecutará (esto se verá más adelante).
- Podemos escribir comentarios con // o /* */ (una línea, varias líneas respectivamente), es decir, códigos que no se ejecutarán, se utilizan para guiarse o entender el código.
- Las funciones reciben algo y entregan otra cosa, retornaremos algo para finalizar el programa (no es importante comprender ahora).

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

Figura 1. Las líneas se observan como números verticalmente (línea 1,2,3,4...)

VARIABLES

- Son "cajas" que permiten guardar un cierto tipo de dato.
- Cada caja se identifica con lo que puedes guardar.
- Los tipos de datos que tenemos (más comunes) son: int, char, bool, string, float.
- La sintaxis es [tipo de dato] [nombre de tu variable] ;
- Siempre utilizar nombres de la variable que sean lógicos, para que otro entienda.

C OUT Y C IN

- Con cout podemos imprimir por consola/terminal lo que queramos, tanto variables como texto.
- Al imprimir una variable, se mostrará el contenido dentro de esta. Es importante que tenga algo dentro.
- Con cin podemos ingresar algún dato por terminal y guardarlo en su respectiva caja (cuidado con tipo de dato).

OPERACIONES MATEMÁTICAS

- Tenemos: $*$ $+$ $-$ $/$
- Es recomendable agrupar en paréntesis.
- Tenemos $<=$ $>=$ $==$ $!=$ y para negar algo $!$
- Existe una librería extra para operaciones matemáticas.
- Tenemos $\%$ para obtener el módulo, sirve para verificar par e impar.

IF / ELSE

- Tipo condición.
- "If" si algo que se cumple, se hace esto => `if(condición){ }`
- `Else{ }` en cualquier otro caso, "lo demás, lo que sobra de diferentes opciones".
- Por ejemplo, si eres mayor de 18 años, eres mayor de edad. En cualquier otro caso (else) no eres mayor de edad.
- Puedes combinar "else if" para verificar si se cumple otra condición, cuando ya la anterior no se cumplió.
- Puedes colocar varios "if" juntos, sin else, para ver si se cumplen varias condiciones a la vez.

CICLOS: WHILE

- Una variable que itera significa que va avanzando en un ciclo, iteración 1,2,3,4... puede avanzar de uno en uno, de dos en dos, o retroceder, etc.
- Mientras algo sea verdadero, se vuelve a ejecutar esas líneas de código.
- Es importante romper el ciclo en algún momento para que no se ejecute indefinidamente.
- Si no hacemos una pausa, como un cin, se imprimirá de forma infinita.
- Sintaxis `while(condicion){ }`
- Dentro de los ciclos, al igual que de los if/else, podemos escribir cualquier línea de código.

CICLOS: DO WHILE

- Primero se ejecutará algo sí o sí, no existe condición.
- Luego de ejecutarse ese "do", se verifica una condición para volver a ejecutarla.
- Sintaxis [do{ }] [while(condicion);]

CICLOS: FOR

- El ciclo for se utiliza cuando conocemos la cantidad de repeticiones.
- Se utiliza siempre cuando trabajemos con arreglos.
- Debemos indicar tres cosas: inicio del ciclo, condición para seguir repitiéndose y en cuánto aumentará una variable para avanzar en las iteraciones.
- Dentro del for podemos definir una variable, la cual sólo existirá en este ciclo, no fuera.
- No podemos llamar en otro lado del código la variable creada por el ciclo for, ya que sólo es local.

ARREGLOS

- Primera estructura de datos que se verá en el curso. Lo combinaremos con el ciclo for.
- Hasta ahora las variables guardan un solo dato, con arreglos, podemos guardar varios de ellos.
- El tamaño (cuántos datos guardaremos en el arreglo) depende del número que indiquemos en el arreglo.
- Si le damos tamaño 20, puede guardar 20 datos.
- Nuevamente, necesita de un tipo de dato, por lo que solo podemos guardar datos de ese tipo, no combinar.
- tipo de dato + nombre arreglo + agregar corchetes así: [tamaño] ;
- El arreglo se crea vacío, o podemos crearlo directamente con datos: `int arreglo[] = {-1,0,1,2,3,4,5};`
- En este caso, el tamaño del arreglo sería 7 números enteros, lo cual sabemos ya que lo creamos nosotros.

FUNCIONES

- Las funciones las podemos definir como trozos de código independiente del resto, que realizan alguna tarea.
- En nuestra función principal "main", podemos pedir la ayuda de estas funciones para realizar una tarea.
- Esto hace un código más ordenado y ahorra el reescribir código para repetir tareas.
- Existen dos tipos de funciones, las vacías y las que retornan, no hay más. Toda función puede o no recibir parámetros, en un orden.
- Los parámetros son los datos que recibe la función para cumplir su tarea, son ilimitados, usamos el nombre que queramos para usarlos.
- Si la función retorna algo (es decir, devuelve un valor), la función entrega un solo dato y termina su ejecución automáticamente. Se indica que retorna por un return. A la hora de llamarla, su resultado se debe imprimir o guardar en una variable.
- Si la función no retorna, es void, es decir, vacía. A la hora de llamarla, se llama solo con su nombre, nada más.
- La función, si retorna un valor, debe indicarse su tipo de dato a entregar. Retorna el mismo tipo de dato indicado.

PASO POR REFERENCIA

- En las funciones, los datos que recibe como parámetros se pasan por valor. Esto significa que, si la función recibe un número 5, creará una copia de ese 5 y trabajará con ella, sin afectar la variable original guardada en memoria.
- Esto puede ser un problema si deseamos crear una función que modifique el valor de una variable y se vea reflejado en la memoria (o en la función "main").
- A la hora de pasar el parámetro, para indicar que se pasará por referencia (el dato guardado original en memoria) se incluye antes del nombre del parámetro un &.
- El concepto de punteros tiene que ver con la memoria, para trabajar con direcciones de memoria directamente. Se utilizará en los siguientes cursos.
- Los arreglos se pasan como referencia siempre, no es necesario indicar un &.