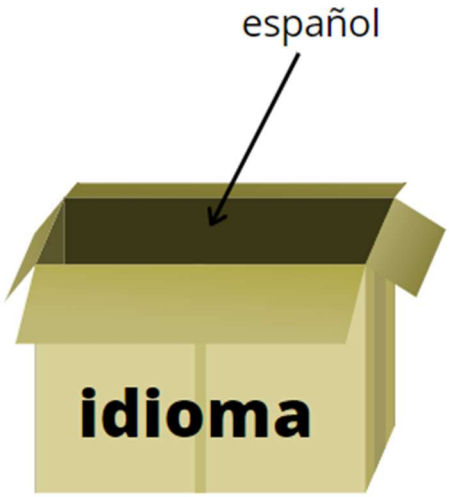
Entrenamiento 1: Introducción a la Programación en C++

Variables

Sin que nosotros nos demos cuenta, nuestro ordenador está constantemente recabando y guardando pequeñas piezas de información: cuánta batería queda, tu código PIN, quién te acaba de seguir en Instagram…

Estos datos se guardan en variables. Una variable es como una caja que guarda un tipo de dato. Por ejemplo, la variable *idioma* guarda el idioma que elige el usuario, que podría considerarse el contenido de la caja.



Como su propio nombre indica, las variables no tienen un valor fijo. Si el usuario cambia el idioma por otro, esto se verá reflejado en el valor que tiene la variable.

Además, cada variable guarda un tipo concreto de dato. La variable *fecha* puede guardar cualquier conjunto de día, mes y año. Pero si tratas de guardar en ella la palabra “pepinillo” te dará error, ya que no está pensada para guardar palabras.

Cuando escribimos nuestros propios programas, creamos y usamos variables todo el rato. Vamos a ver un ejemplo:

```
int edad;
```

Esta instrucción en C++ está creando una variable del tipo **int** y llamándola *edad*. Que una variable sea del tipo **int** significa que puede guardar números enteros entre -2,147,483,648 y 2,147,483,647 (pronto veremos por qué son estos los límites). *edad* es el nombre que le damos a la variable. Como programadora, puedes elegir el nombre que quieras (mientras no tenga espacios y empiece por una letra o _ y solo contenga letras, números y _), pero es bueno que sea descriptivo para que te acuerdes de qué contiene cada variable en tu programa, ya que es normal crear varias.

Al final de cada comando, ponemos punto y coma, que es la forma que tiene C++ de ver que hemos terminado una instrucción. Crear variables, como hemos hecho ahora, se llama *declarar* variables.

Pero claro, lo interesante es poder asignarles valores. Esto se llama *inicializar* una variable y se hace de la siguiente forma:

```
edad = 17;
```

Puedes cambiar el valor de la variable de la misma forma:

```
edad = 18;
```

Además, puedes declarar varias variables en la misma línea, mientras sean del mismo tipo. Por ejemplo:

```
int edadMarta, edadAna, edadAlba;
```

Si, al declarar una variable, quieres darle un valor, como por ejemplo:

```
int numeroDeJugadores;

numeroDeJugadores = 10;
```

puedes hacerlo de una forma más sencilla:

```
int numeroDeJugadores = 10;
```

e incluso se puede declarar e inicializar varias variables en la misma instrucción:

```
int edadMarta = 16, edadAna = 17, edadAlba = 15;
```

En C++ hay varios tipos de variable, en función del tipo de dato que quieras guardar. Los principales son:

Tipo	Valor
int	Enteros entre -2,147,483,648 y 2,147,483,647
long long int	Enteros entre -9,223,372,036,854,775,808 y 9,223,372,036,854,775,807
float	Números decimales con una precisión de 7 dígitos
double	Números decimales con una precisión de 15 dígitos
char	Un carácter
string	Secuencias de caracteres (texto)
boolean	Verdadero (true) o falso (false)

Entrada y salida de datos

Es fundamental poder comunicarte con el usuario cuando haces un programa.

La entrada de datos (input) en C++ se realiza mediante el comando **cin**. Por ejemplo:

```
cin >> nombre;
```

`cin` le dice al ordenador que debe leer la entrada por teclado para obtener el valor de la variable *nombre* y guardarlo ahí. **`cin`** lee hasta el siguiente espacio en el input y toma eso como el valor de la variable. Esto es útil de cara a obtener números o caracteres. Sin embargo, muchas veces nos interesan **strings** de más de una palabra.

Para esto, tenemos otra función:

```
getline(cin, nombre);
```

`getline`, como su propio nombre indica, recoge la siguiente línea del input entera, guardándola en la variable nombre.

Además de recabar datos del usuario, también le podemos dar información mediante el comando **cout**.

```
cout << "Voy a ganar la OIFem.\n";
```

`cout` va seguido de la información a imprimir. El carácter `'\n'` significa *nueva línea*. Además, puedes imprimir variables:

```
cout << nombre;
```

e incluso juntar variables y texto con `<<`:

```
cout << "Me llamo " << nombre << ".\n";
```

En C++ también se puede usar `endl` en vez de `'\n'` (significan lo mismo), pero `'\n'` es más rápido, por lo que es la opción que se suele usar cuando programamos de forma competitiva.

Aritmética

C++ te permite hacer numerosas operaciones a través de operadores. Los operadores aritméticos de C++ son:

Operador	**Función**
+	Suma dos números (enteros o decimales) y concatena dos strings (“Hola,” + “ Alba” = “Hola, Alba”)
-	Resta un número de otro (enteros o decimales)
*	Multiplica dos números (enteros o decimales)
/	Calcula la división de dos números. Si estos son decimales, devuelve un resultado decimal, pero si son enteros, devuelve la parte entera (y no la fracción, si hay) de este resultado. Por lo tanto: 5.0/2.0 = 2.5 5/2 = 2
%	Calcula el resto de la división de dos enteros
++	Incrementa enteros
--	Decrementa enteros

Selección

Hay veces que queremos que nuestro programa tenga una bifurcación y pueda tomar caminos distintos en función del estado del programa. Esto se hace a través de bloques de **if-else**.

Las instrucciones dentro de un bloque *if* se ejecutan si la condición del *if* es verdadera.

```
if (edad >= 18) {
    cout << "Eres mayor de edad\n";
}
```

Si no lo es, el programa se salta ese bloque y continúa su ejecución como si no existiera.

Además, podemos incluir un bloque *else*, que se ejecutará si la condición es falsa.

```
if (edad >= 18) {
     cout << "Eres mayor de edad\n";
} else {
     cout << "Aún eres menor :(\n";
}
```

Si queremos tener más de dos opciones, usamos un *else if*:

```
if (edad > 18) {
     cout << "Eres mayor de edad\n";
} else if (edad == 18) {
     cout << "A sacarte el carnet!\n";
} else {
     cout << "Eres menor :(\n";
}
```

Pueden usarse infinitos *else if* en un bloque, pero solo puede haber un *if* y un *else*.

Iteración

Para ahorrarnos repetir el mismo código con cambios mínimos, existen los bucles. Son estructuras que repiten el contenido de un bloque delimitado por llaves. En C++ las hay de tres tipos: **for**, **while** y **do-while**.

Un bucle *for* tiene tres partes: inicialización, condición y cambio. Primero, le da un valor a una variable, antes de la primera ejecución. Entonces, comprueba la condición. Si es verdadera, se ejecuta el bloque de código y se efectúa el cambio. Así, hasta que la condición es falsa y se sale del bucle.

```
for (int i = 1; i <= 10; i++) {
     cout << i << "\n";
}
```

Este código imprime los números del 1 al 10, línea por línea.

Los bucles *while* revisan una condición con cada iteración. Si esta es verdad, ejecutan el bloque. Si no, se sale del bucle y el programa debajo de este se ejecuta. Por lo tanto, se corre el riesgo de que no se ejecute ninguna vez, al igual que el *for*. El mismo programa pero con un *while*:

```
int i = 1;
while (i <= 10) {
    cout << i << "\n";
    i++;
}
```

Por último, hay un bucle llamado do-while. Este se ejecuta al menos una vez, ya que comprueba la condición al final de cada ejecución. Funciona como un *while* salvo por esta diferencia. De nuevo, el programa que cuenta hasta 10:

```
int i = 1;
do {
    cout << i << "\n";
    i++;
} while (i <= 10)
```

Comentarios

Para ayudarte a leer el código, puedes poner comentarios en el archivo. Estas son líneas que el compilador se salta y no lee, pero que pueden ser útiles para que vayas haciendo anotaciones.

Un comentario de una sola línea se hace empezando la línea con //:

```
// esto es un comentario
```

Uno de varias líneas se hace con los delimitadores /* y */:

```
/* esto
es
un
```

```
comentario */
```

Programa completo

```
int main {
     // este programa calcula la media de una clase
     double suma = 0, nota;
     int alumnos;
     cin >> alumnos;
     for (int i = 0; i < alumnos; i++) {
          cin >> nota;
          // sumamos la nota al total
          suma += nota;
     }
     // calculamos e imprimimos la media
     cout << "La media es " << suma/alumnos << "\n";
     return 0;
}
```