Apuntes OIFem II Nivel 1

Nuestro primer programa

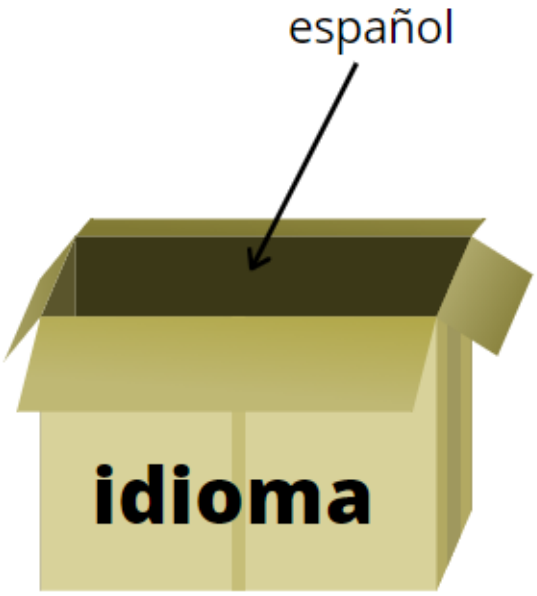
Dónde programaremos

En la clase vimos dos páginas que usaremos a lo largo del resto del curso: C++ Tutor para visualizar código y Online C++ Compiler para compilar el código, es decir, convertir nuestras instrucciones en C++ a un archivo ejecutable.

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:

```
1 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:

```
1 edad = 17;
```

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

```
1 edad = 18;
```

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

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

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

```
1 int numeroDeJugadores;
2 numeroDeJugadores = 10;
```

puedes hacerlo de una forma más sencilla:

```
1 int numeroDeJugadores = 10;
```

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

```
1 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:

- `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:

```
1 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 la entrada 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:

```
1 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`.

```
1 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:

```
1 cout << nombre;
```

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

```
1 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.

Enviando problemas a jueces online

Ya vimos en clase cómo subir nuestra solución a un problema a un juez online. Cada vez que hacemos un envío, el juez online prueba nuestro código con varios valores de entrada que tiene preparados. Sacaremos AC (Accepted) si nuestro código imprime la salida esperada dentro de los límites de tiempo y memoria para todos los tests. Si no, obtendremos otro veredicto, como WA (Wrong Answer), que nos dirá en qué hemos fallado. Los más frecuentes son:

- AC/Accepted- ¡enhorabuena! Tu código ha superado todos los tests
- WA/Wrong Answer- hay algún test con el que tu código falla e imprime una salida que no es la esperada
- TLE/Time Limit Exceeded- hay algún test en el que tu código se pasa del límite de tiempo
- MLE/Memory Limit Exceeded- hay algún test en el que tu código se pasa del límite de memoria
- PE/Presentation Error- faltan/sobran carácteres de separación, es decir, que faltan/sobran espacios y/o nuevas líneas

Operadores aritméticos

C++ tiene una serie de operadores que nos permiten realizar operaciones aritméticas: `+`, `-`, `*`, `/` y `%`.

Los tres primeros funcionan exactamente como nos esperaríamos pero es importante tener en cuenta que la división de dos enteros en C++ retornará otro entero, con independencia de que pueda haber un resto en la división. Por lo tanto, $5/2 = 2$ en tu programa. Si esto no es lo que buscas, deberás usar variables con decimales.

Por último, el operador `%` nos dice el resto de una división, es decir, que $5\%2 = 1$.

Además, el operador `+` también nos sirve para concatenar `string`s.

Nota: a partir de ahora será todo addendum, es decir, temario no cubierto en clase pero para quien tenga interés en avanzar más y en sacar los problemas más difíciles.

`max`, `min` y `abs`

C++ tiene también tres funciones muy útiles: `max`, `min` y `abs`. `max` y `min` comparan variables numéricas del mismo tipo, retornando respectivamente el mayor/menor valor de los dos. `abs` retorna el valor absoluto de la variable introducida.

Cambiando el tipo de variable: enteros y decimales

Si queremos pasar un entero a a una variable de tipo decimal b, por ejemplo para realizar una división, basta con convertirlo de la siguiente manera:

```
double b = (double) a;
```

¿Sabrías predecir lo que va a imprimir el siguiente programa? Pruébalo para comprobar tu respuesta.

```
#include <iostream>
using namespace std;

int main() {
  int a = 5;
  cout << a/2 << '\n';
  double b = (double) a;
  cout << b/2 << '\n';
  return 0;
}
```

Además, puedes pasar de variables decimales a enteros, perdiendo el valor tras el punto decimal, de la siguiente manera:

```
int a = (int) b;
```

Cambiando el tipo de variable: enteros y strings

A veces necesitamos poder pasar de una `string` cuyos caracteres son todos dígitos a un entero. Por ejemplo, convertir `"123"` en `123`. Podemos hacerlo con la función `stoi`:

```
#include <iostream>
using namespace std;

int main() {
  string a = "123";
  int b = stoi(a);
  cout << b+1 << '\n';
  return 0;
}
```

También podemos hacer lo contrario con la función `to_string`:

```cpp
#include <iostream>
using namespace std;

int main() {
  int a = 123;
  string b = to_string(a);
  cout << b + "1" << '\n';
  return 0;
}
```

Cambiando el tipo de variable: la tabla ASCII

Por último, hablaremos sobre la tabla ASCII. Cuando C++ guarda un carácter, ya sea en una variable de tipo `char` o en una `string`, guarda en realidad un entero. Cada carácter tiene un entero correspondiente y lo podemos encontrar en la tabla ASCII. Esto nos permite hacer cosas como comparar caracteres y podemos pasar de un tipo a otro de la siguiente manera:

```cpp
#include <iostream>
using namespace std;

int main() {
  cout << (int) 'a' << '\n';
  cout << (char) 97 << '\n';
  return 0;
}
```

Usando booleanos como enteros

Vimos en clase que si tratábamos de imprimir el valor de una variable booleana, en vez de imprimirse true/false se imprimía 1/0. Podemos usar variables de tipo `bool` como si fueran `int`s. Por ejemplo, si y es igual a `true`, `2*y = 2`. Esto nos permitirá resolver muchos problemas.

Operadores para comparar valores

Además de poder asignar true/false a una variable booleana, también podemos asignarle el resultado de una comparación de dos valores. Por ejemplo:

```cpp
int x = 5;
int y = 7;
bool x_mayor_que_y = x > y;
```

Aquí la variable booleana será `true` si $x > y$ y `false` en caso contrario.

El resto de operadores de comparación son los siguientes:

- `x == y`- esta expresión será `true` si $x = y$ y `false` en caso contrario
- `x < y`- esta expresión será `true` si $x < y$ y `false` en caso contrario
- `x <= y`- esta expresión será `true` si $x \leq y$ y `false` en caso contrario
- `x > y`- esta expresión será `true` si $x > y$ y `false` en caso contrario
- `x >= y`- esta expresión será `true` si $x \geq y$ y `false` en caso contrario.