BUCLES

CADA APARTADO PRINCIPAL CORRESPONDE CON EL NOMBRE DEL PROYECTO CON EJEMPLOS DE LO APUNTADO AQUÍ

1. INTRODUCCIÓN A LOS BUCLES

- Los bucles son estructuras que nos permiten hacer tareas que se repiten de forma relativamente fácil.
- Si lo que quieres hacer tienes que repetirlo 10 veces puedes hacerlo y no va a pasar nada, pero si la tarea tienes que repetirla 100000 veces es cuando los bucles empiezan a ser útiles, ya que simplifican muchísimo el trabajo.
- Los principales bucles son:
  - for
  - for in range
  - while
  - do while

2. BUCLE FOR

- Los bucles for se usan para repetir tareas, cuando sabes el número exacto de veces que tienes que repetir la tarea.
- Los pilares de cualquier bucle son:
  - Iterador: Variable empleada para navegar a través del bucle.
  - Punto de inicio: Valor con el que se inicializa el iterador.
  - Controlar cuando el bucle va a parar.
  - El incremento (o decremento) del iterador por cada vuelta que da el bucle.
  - El cuerpo del bucle, es decir la tarea que queremos que se repita.

- Ejemplo de bucle for:
  - ```
    for(int i = 0 ; i < 10 ; i++)
    {
      std::cout << “Esta línea se va a repetir 10 veces”
      << std::endl;
    }
    ```

- En el ejemplo de arriba:
  - La `i` es el iterador.
  - El `0` es el punto de inicio.
  - El bucle se parará cuando `i < 10`
  - La `i` aumentará en 1 por cada vuelta que el bucle da.
  - El cuerpo del bucle es:
    ```
    std::cout << “Esta línea se va a repetir 10 veces”
    << std::endl;
    ```
- A la hora de definir el iterador se puede usar `size_t` como tipo de dato, que en verdad no es un tipo de dato, sino más bien un alias para representar números enteros no negativos (`unsigned int`). Se suele usar para que el código sea más legible.
- Se pueden realizar operaciones con el iterador dentro del cuerpo del bucle.
- Si el cuerpo del bucle solo tiene una sentencia, no es necesario que se añadan los `{}`
- La variable definida como iterador del bucle solo va a funcionar dentro del bucle. Si intentas acceder a ella fuera del bucle, no podrás ya que no existe a nivel local en la función. Son los `{}` los que marcan el inicio y el final del bucle.
  En caso de querer usar la variable fuera del bucle tendrás que definirla fuera y luego usarla como iterador del bucle.
- *Hardcodear* el valor en el que el bucle se tiene que parar es mala idea, ya que cuando tienes un código muy grande puede ser muy molesto ir bucle por bucle modificando el valor. Es mejor definir una variable constante fuera del bucle y usarla en él, para que al modificar la variable se modifiquen todos los bucles a la vez.
    ```

3. BUCLE WHILE

- El `while` se usa cuando quieres repetir varias veces una misma tarea si una condición es cierta. Si la condición es falsa, el bucle no se ejecuta ni una vez.
  Otra característica del `while` que lo diferencia del `for`, es que se puede usar cuando no sabes el número concreto de veces que tienes que repetir la tarea, es decir, puedes crear bucles infinitos usando `true` como condición y un `break` dentro del bucle para salir de él.
- Ejemplo de bucle `while`:
  - ```
    const unsigned int CONTADOR = 10;
    unsigned int i = 0;

    while (i < CONTADOR)
    {
      std::cout << "Esta línea se va a repetir 10 veces"
      << std::endl;
      i++;
    }
    ```

- En el ejemplo de arriba:
  - La `i` es el iterador.
  - El `0` es el punto de inicio.
  - El bucle se parará cuando `i < 10`, que en este caso está definido en la constante `CONTADOR`.
  - La `i` aumentará en 1 por cada vuelta que el bucle da.
  - El cuerpo del bucle es:
    ```
    std::cout << "Esta línea se va a repetir 10 veces"
    << std::endl;
    ```
- Al igual que en el `for`, también se puede usar `size_t`.
- Al igual que en el `for`, se pueden realizar operaciones con el iterador dentro del cuerpo del bucle.
  Al igual que en el `for`, la variable definida como iterador del bucle solo va a funcionar dentro del bucle. Si quieres usarla fuera del bucle tendrás que definirla fuera y luego usarla como iterador.
- Al igual que en el `for`, *hardcodear* el valor en el que el bucle se tiene que parar es mala idea. Es mejor definir una variable constante fuera del bucle y usarla en él.
    ```

4. BUCLE DO WHILE

- A diferencia del `while`, este tipo de bucle se asegura de que al menos la tarea se realiza una vez, en el `do while` primero se realiza la tarea y luego se evalúa la condición, por eso se realiza como mínimo una vez si o si.
- Ejemplo de `do while`:
  - ```
    const unsigned int CONTADOR = 10;
    unsigned int i = 0;

    do
    {
      std::cout << "Esta línea se va a repetir 10 veces"
      << std::endl;
      i++;
    } while (i < CONTADOR)
    ```

- En el ejemplo de arriba:
  - La `i` es el iterador.
  - El `0` es el punto de inicio.
  - El cuerpo del bucle es:
    ```
    std::cout << "Esta línea se va a repetir 10 veces"
    << std::endl;
    ```
  - La `i` aumentará en 1 por cada vuelta que el bucle da.
  - El bucle se parará cuando `i < 10`, que en este caso está definido en la constante `CONTADOR`.
- Al igual que en el `while`, también se puede usar `size_t`.
- Al igual que en el `while`, se pueden realizar operaciones con el iterador dentro del cuerpo del bucle.
  Al igual que en el `while`, la variable definida como iterador del bucle solo va a funcionar dentro del bucle. Si quieres usarla fuera del bucle tendrás que definirla fuera y luego usarla como iterador.
- Al igual que en el `while`, *hardcodear* el valor en el que el bucle se tiene que parar es mala idea. Es mejor definir una variable constante fuera del bucle y usarla en él.
    ```