Las estructuras de repetición nos permiten realizar un conjunto de acciones en nuestros programas de manera repetitiva de manera rápida, precisa y confiable. Un bucle o **ciclo** es una construcción que nos permitirá repetir una secuencia de instrucciones un número de veces. Este grupo de instrucciones que se repiten en un ciclo lo llamaremos **cuerpo** del ciclo y cada una de las repeticiones la llamaremos **vuelta** o **iteración**.

Supongamos que tenemos la necesidad de cargar las notas de todos los estudiantes de un curso para calcular el promedio general. Sabiendo que son cinco alumnos y que todos rindieron un examen. Una forma correcta pero poco práctica sería resolverlo de la siguiente manera:

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

int main(){
      int nota, suma = 0;
      float promedio;
      cout << "Ingresar las notas del curso: " << endl;
      cin >> nota;
      suma = suma + nota;
      cin >> nota;
      suma = suma + nota;
      cin >> nota;
      suma = suma + nota;
      cin >> nota;
      suma = suma + nota;
      cin >> nota;
      suma = suma + nota;

      promedio = (float) suma / 5;
      cout << "El promedio general es: " << promedio << endl;
      return 0;
}
```

Si bien podemos observar que el resultado es correcto, se puede notar como nuestro código se torna repetitivo realizando exactamente la misma tarea una y otra vez. Aquí es

donde entran en juego las estructuras repetitivas para ayudarnos a escribir código más corto, prolijo y mantenible.

Las estructuras de repetición en C y C++, como en muchos otros lenguajes, se dividen en dos: el **ciclo exacto** (ciclo for) y el **ciclo inexacto** (ciclo while y ciclo do-while).

Ciclo exacto

El ciclo exacto, es una estructura repetitiva de programación que nos permite realizar ciclos cuyo cuerpo se ejecute una *cierta* cantidad de veces. Esto quiere decir que mediante el ciclo exacto se sabe exactamente la cantidad de iteraciones que el ciclo realizará, ya sea porque lo definimos con un valor constante o mediante una variable.
Podríamos definir la estructura base del ciclo exacto de la siguiente manera:

```
for(i=inicio; i<fin; i++){
  //Cuerpo del ciclo (instrucciones a repetir)
}
```

Donde **for** es la palabra reservada que utilizaremos para definir un ciclo exacto. Para ello además necesitaremos definir, entre paréntesis, tres elementos clave que conforman un ciclo exacto:
Inicialización → i = inicio
Comparación → i < fin
Incremento → i++

Como podrán notar, el ciclo exacto es totalmente dependiente de al menos una variable. La **variable que controla el for** en el ejemplo es la variable i. A esta variable particular podemos llamarla **numerador** del ciclo, ya que se encarga de adquirir los valores en la variable que controla el for (que generalmente son valores numéricos). Esta variable puede nombrarse de la manera que uno desee (aunque se suele usar la letra i) y suele ser una variable entera aunque también puede ser float o char. Esta variable será de utilidad para controlar la cantidad exacta de iteraciones que realizará el ciclo, comenzando por la **inicialización**.
La inicialización, como su nombre da a entender, es el valor inicial que tendrá la variable. Esta instrucción se ejecuta una vez al comenzar el ciclo y no se vuelve a ejecutar. Los valores más comunes para iniciar una variable en un for son el 1 o el 0. Ya que solemos comenzar a contar desde estos valores. Pero no, al momento de programar, tenemos la libertad de elegir el valor inicial de nuestra variable según la necesidad de nuestro algoritmo.

Si para darle solución a lo que queremos resolver necesitamos un ciclo exacto que comience con el valor 50. Podremos asignarle dicho valor sin problemas.

Luego de la inicialización se ejecuta la **comparación**. La misma es una proposición lógica cuyo resultado puede ser verdadero o falso. Si es evaluada como verdadera entonces realiza una iteración o, dicho de otra manera, ejecuta todas las instrucciones que componen el cuerpo del ciclo. Si la proposición lógica es evaluada como falsa el ciclo exacto finaliza y la ejecución del programa continúa en la instrucción próxima inmediata al ciclo.
Se podría decir que con la inicialización y la comparación ya podemos asegurar la cantidad exacta de vueltas que dará el ciclo pero falta un elemento más que es clave para esto. El **incremento** que también puede ser un **decremento**, aumenta o disminuye el valor de la variable que controla el for de manera que, en algún momento, la comparación sea falsa.

Ejemplos de ciclos y cantidad de iteraciones

```
for(i=1; i<=10; i++){
  //Comienza en 1; hasta 10 inclusive; incrementa de a 1.
  //10 iteraciones
} //Finaliza cuando i contiene 11
```

```
for(i=0; i<10; i++){
  //Comienza en 0; hasta 10 no inclusive; incrementa de a 1.
  //10 iteraciones
} //Finaliza cuando i contiene 10
```

```
for(i=10; i>=0; i--){
  //Comienza en 10; hasta 0 inclusive; decrementa de a 1.
  //11 iteraciones
} //Finaliza cuando i contiene -1
```

```
for(i=1; i<=10; i=i+2){
  //Comienza en 1; hasta 10 inclusive; incrementa de a 2.
  //5 iteraciones
} //Finaliza cuando i contiene 11
```

```
for(i=2; i<10; i=i+2){
  //Comienza en 2; hasta 10 no inclusive; incrementa de a 2.
  //4 iteraciones
} //Finaliza cuando i contiene 10
```

Volviendo al ejemplo de las notas y los estudiantes. Podríamos resolverlo de la siguiente manera:

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

int main(){
  const int ALUMNOS = 5;
  int i, nota, suma = 0;
  float promedio;
  cout << "Ingresar las notas del curso: " << endl;

  for(i=1; i<=ALUMNOS; i++){
    cin >> nota;
    suma = suma + nota;
  }

  promedio = (float) suma / ALUMNOS;
  cout << "El promedio general es: " << promedio << endl;
  return 0;
}
```

En el código de arriba podemos observar la simpleza y legibilidad del código. Resuelve exactamente lo mismo que la primera de las soluciones pero con la ventaja de resolver la carga de las calificaciones y su posterior acumulación utilizando una estructura de repetición. Además, se hace uso de una constante para referir a la cantidad de alumnos (que originalmente es 5). Esto, como pueden suponer, tiene la capacidad de escalar y que el programa pueda procesar un curso de 500 alumnos simplemente cambiando el valor de la constante ALUMNOS. ¿Podrían imaginarse a ustedes mismos resolviendo el programa para 500 alumnos pero con la primera solución? Felicitaciones, esto significa que ya aprendieron a utilizar el ciclo exacto.

Desafío

Tu sobrina está aprendiendo las tablas de multiplicar y quisiera disponer de un programa

para que pueda corroborar que las cuentas realizadas en su cuaderno sean correctas. Ella sabe que estás estudiando programación en la universidad y le parece que es algo que no debería llevarte más que unos pocos minutos.
Puntualmente te pidió que puedas ingresar un número entre 1 y 15 y dibuje por pantalla las tablas de dicho número desde el 0 hasta el 10 inclusive.

Por ejemplo:

```
Ingresa el número del cual quieres ver los cálculos: 6

6 x 1 = 6
6 x 2 = 12
6 x 3 = 18
6 x 4 = 24
6 x 5 = 30
6 x 6 = 36
6 x 7 = 42
6 x 8 = 48
6 x 9 = 54
6 x 10 = 60
```

Recorrer un ciclo exacto mediante una variable numérica no es la única manera de realizar un ciclo exacto. Existe también una alternativa de ciclo exacto que utiliza un **iterador**. Este particular ciclo referencia en cada iteración un elemento distinto dentro de una colección. Comenzando desde el primer elemento de la colección hasta el último de ellos (el ciclo sabe cuál es el elemento inicial y cuál el final). Este tipo de ciclo se conoce en otros lenguajes como **for each.** En C++ su sintaxis es como la siguiente:

```
for (int elemento: coleccionElementos){
   // Hacer algo con el elemento de la iteración
}
```

En este ejemplo se utiliza *int* dentro del for porque, para el ejemplo, se asume que la colección de elementos es de números enteros. Pero podría ser string, char, float, bool, etc.

Para esto es necesario conocer los objetos que nos permiten representar colecciones de información. Como los vectores, listas, etc. Esto se verá más adelante en la carrera.