El ciclo inexacto es una estructura repetitiva que nos permite realizar un conjunto de instrucciones una indeterminada cantidad de veces. Lo que determina si el ciclo debe realizar una iteración o finalizar es una (o más de una) proposición lógica. Si el valor de la condición es verdadero entonces el ciclo realizará una iteración, en cambio, si es falso el ciclo finalizará.
Podríamos definir la estructura base del ciclo inexacto de la siguiente manera:

```
while(condicion){
  //Cuerpo del ciclo (instrucciones a repetir)
}
```

Supongamos que tenemos una cuenta corriente en un banco con un saldo de $13000. Queremos hacer un programa que nos permita cargar el importe de compras realizadas que se debitan de esa cuenta y que deje de pedirnos información cuando comencemos a deber dinero. El programa informará la cantidad de compras que se ingresaron.

En este caso, cada iteración del ciclo nos va a permitir cargar un importe que representa una compra a debitar del saldo de la cuenta corriente. Como no se sabe de antemano los importes de las compras, no es posible conocer exactamente la cantidad de ellas que ingresaremos. Lo que sí es seguro es que la continuidad o no del ingreso de datos va a depender de que el saldo de la cuenta corriente no registre deuda. Es decir que **mientras el saldo no sea negativo** el programa deberá seguir pidiendo importes de compras y llevando la cuenta de cuántas se ingresaron. El código fuente del programa podría quedar así:

```
#include <iostream>
using namespace std;
int main()
{
   float saldo = 13000, importeCompra;
   int cantCompras = 0;

   while(saldo >= 0){
      cout << "Importe $: ";
      cin >> importeCompra;
```

```
        saldo = saldo - importeCompra;
        cantCompras++;
    }

    cout << "Cantidad de compras registradas: " << cantCompras << endl;

    return 0;
}
```

Como se puede observar, el ciclo igual continúa iterando siempre y cuando el saldo sea cero o positivo. En cada iteración, solicita el importe de una nueva compra el cual utiliza para restarlo al saldo e incrementa en uno el contador de compras. Luego, puede ocurrir que la cuenta quede con saldo negativo (y finalizar el ciclo inexacto) o bien que el mismo aún sea positivo o cero y continuar solicitando datos. Al finalizar el ciclo, se informa la cantidad de compras.
Una posible ejecución del programa sería la siguiente:

```
Importe $: 10000
Importe $: 2500
Importe $: 600

Cantidad de compras registradas: 3
```

Veamos un ejemplo un poco más complejo. En este caso supongamos que un docente tiene que ingresar las notas de examen de un curso y el programa debe calcular el promedio general e informar la cantidad de alumnos que rindieron. Al docente no le interesa cargar la cantidad de alumnos que hay en el curso porque muchos de ellos pueden ausentarse al examen. A él le gustaría cargar las notas una a una y cuando decida finalizar la carga, el programa informe los resultados. Como las calificaciones oscilan entre 1 y 10, decidió que la manera de indicar que no hay más exámenes para corregir es ingresando una calificación fuera de ese rango.

En este caso tenemos un conjunto de instrucciones que se repiten (la carga de notas y la sumatoria de notas y de exámenes para calcular el promedio). Y también sabemos que este conjunto de repeticiones se hacen una cantidad indeterminada de veces (porque va a depender de la cantidad de alumnos que rindieron examen y no es un dato que conozcamos de antemano). De hecho, es un dato que obtendremos como parte de nuestro procesamiento de datos.

¿Qué representa el ciclo inexacto?

El ciclo inexacto representa la carga de notas. Con su respectivo acumulado y contador para calcular el promedio de ellas.

¿Qué condición debe tener el ciclo inexacto para continuar pidiendo notas?

El ciclo inexacto debe evaluar que la nota ingresada por el docente se encuentre dentro del rango entre 1 y 10.

Veamos cómo quedaría el código del programa:

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

int main()
{
   int cantExamenes = 0;
   float sumaNotas = 0, nota;

   cout << "Nota: ";
   cin >> nota;

   while (nota >= 1 && nota <= 10){

      cantExamenes++;
      sumaNotas += nota;

      cout << "Nota: ";
      cin >> nota;

   }

   cout << endl << "Cantidad exámenes: " << cantExamenes << endl;
   if (cantExamenes > 0){
      cout << "Promedio: " << sumaNotas / cantExamenes << endl;
   }


   return 0;
}
```

Una posible carga de datos para este programa es la siguiente:

```
Nota: 2
Nota: 6
```

```
Nota: 10
Nota: -5

Cantidad exámenes: 3
Promedio: 6
```

En este ejemplo podemos ver una curiosidad. En el código se solicita el ingreso de la primera nota fuera del ciclo inexacto y luego se ingresa la nota nuevamente dentro del ciclo al final del mismo. Esto se debe a que la nota, además de ser un valor de entrada necesario para calcular el promedio, también es el dato que se evalúa para finalizar el ciclo. En nuestro caso, el ciclo finaliza cuando la nota ingresada sea un valor fuera del rango [1; 10]. Podríamos decir que finaliza cuando se ingresa un valor absurdo a la calificación. Y como es un valor absurdo, no queremos que sea tenido en cuenta para nuestros cálculos. Esto significa que si ingreso una nota con valor -99 para finalizar la carga de datos, no quiero que ese valor forme parte de la sumatoria para el promedio.

Ahora bien, ¿por qué ocurre esto?¿Por qué tenemos un valor o conjunto de valores "absurdos" para indicar el fin de la carga de información?

Bueno, la respuesta es por practicidad. De lo contrario, deberíamos preguntarle al usuario luego de la carga de cada registro si desea continuar cargando más datos. Haciendo que la carga de información sea aún mayor y menos ágil. Veamos la siguiente comparativa:

Ingrese nota: 8 Ingrese nota: 9 Ingrese nota: 4 Ingrese nota: 5 Ingrese nota: -6	Ingrese nota: 8 Continuar (S/N): S Ingrese nota: 9 Continuar (S/N): S Ingrese nota: 4 Continuar (S/N): S Ingrese nota: 5 Continuar (S/N): N

Esta técnica en la que se ingresan valores y se los comparan contra un valor determinado para indicar el fin de la carga de información se la conoce como **Ciclo con centinela**. Siendo el centinela el o los valores que determinan el fin de la carga de información.
El esquema de ciclo con centinela sería el siguiente:

```
Pedir dato
Mientras el dato ingresado (no coincida/coincida) con el centinela
```

```
- Realizar cálculos
- Pedir dato
```

En el programa que hicimos anteriormente la estructura era:

```
Pedir nota
Mientras la nota no se encuentre fuera del rango entre 1 y 10
- Acumular notas y contar exámenes
- Pedir nota
```

Esta estructura además, garantiza que la nota con la que finalizamos la carga no sea tenida en cuenta en los cálculos. También brinda la oportunidad de que nunca ingresemos una nota válida si la primera nota ingresada está fuera del rango válido. Algo que sería totalmente aceptable si ningún alumno se presentó al examen.

Desafío

Vamos a hacer un juego de adivinación. La computadora debe calcular un número aleatorio entre el 1 y el 100 y debe almacenarlo en una variable. El jugador debe intentar adivinar el número que la computadora calculó. Si el usuario adivina el número, el juego debe mostrar la cantidad de intentos que le llevó hacerlo. De lo contrario, debe solicitarle al usuario otra posibilidad para adivinar y contabilizar un intento fallido.

Como alternativa, se puede solicitar antes de comenzar el juego el nivel de dificultad.

Fácil: Cantidad de intentos ilimitados. Cada cierta cantidad de intentos fallidos la computadora da una pista del tipo "el número que pensé es par", "el número que pensé es primo", etc.

Normal: Diez intentos como máximo. Al quinto intento fallido la computadora da una pista del tipo "El número es mayor/menor al que ingresaste recién".

Difícil: Cinco intentos como máximo. No hay pistas.

¿Cómo hacemos que la PC pueda calcular un número aleatoriamente?

** Spoiler **

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

int main()
{
   int numero;
   srand(time(0));
   numero = rand() % 100 + 1;
   cout << numero;

   return 0;
}
```

En esas pocas líneas ocurre lo necesario para calcular un número al azar y mostrarlo por pantalla. La escasa longitud del programa se debe a que estamos haciendo uso de funciones (tema que veremos más adelante). Particularmente usamos srand para inicializar el algoritmo de cálculo de números aleatorios. Esta función necesita recibir un número que sea diferente cada vez que corramos el programa, de lo contrario los números que genere de manera aleatoria serían siempre los mismos y carecería de utilidad. Para garantizar que el número que reciba sea distinto cada vez hacemos uso de time(0) que es una función que devuelve una marca de tiempo que representa el tiempo actual del sistema (busquen en Internet si no saben lo que es una marca de tiempo). Por último, la función rand() es la que devuelve un número al azar. El problema es que devuelve un número no negativo que puede ser cualquiera. No tenemos ningún control sobre el rango del número calculado. Para limitarlo entre 1 y 100 usamos una simple división. Como pueden observar el número calculado por rand() es dividido por 100 y nos quedamos con el resto de esa división. Cualquier resto de dividir un número por 100 nos dará como resultado un valor entre 0 y 99. A eso le sumamos 1 para garantizar que esté entre 1 y 100.

Es posible que por ahora esto parezca bastante complejo. Con tiempo y práctica esos llamados a funciones tendrán sentido y no presentarán ninguna dificultad. Por el momento, pueden utilizar lo necesario de ese código para resolver el desafío del juego de adivinación.