1. Sentencias repetitivas

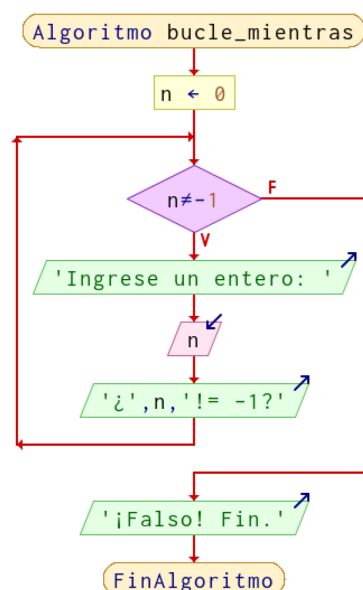
En el encuentro anterior se expusieron las sentencias de control *selectivas* en las cuales podíamos bifurcar el flujo o secuencialidad de nuestro algoritmo para ejecutar instrucciones y omitir otras. A diferencia de estas, las sentencias de control **repetitivas** nos permiten *volver a ejecutar* instrucciones dada una condición o expresión lógica.

En el ámbito de la programación es común que se quiera evitar, en la medida de lo posible, la repetición de código. Es decir, escribir lo mismo (o casi lo mismo) varias veces en el algoritmo. Es por ello que existen estas estructuras donde uno puede ejecutar las mismas instrucciones cuantas veces lo requiera. A estas estructuras suelen nombrárselas como *bucles* o *loops*.

1.1. Mientras

Así como el condicional simple es la sentencia selectiva más simple, el *"Mientras"* es el primer bucle que suele introducirse. En este caso, las instrucciones dentro de la estructura se repetirán hasta que la *expresión lógica* sea *falsa* o, dicho de otra forma, **mientras** sea *verdadera*. Se utiliza la palabra reservada `while` para este fin.

```cpp
#include <iostream>
using namespace std;
int main() {
  int n = 0;
  while(n != -1) {
    cout << "Ingrese un entero: ";
    cin >> n;
    cout <<"¿"<<n<<" != -1?"<<endl;
  }
  cout << "¡Falso! Fin." << endl;
}
```



Aquí se evaluará la expresión `n` $\neq$ `-1` y, de ser verdadera, se pedirá ingresar un entero. *"Mientras"* el entero sea distinto de -1, entonces se seguirá pidiendo ingresar valores. Ha de notarse que `n` existe antes de evaluar la condición con un valor que asegura su estado verdadero, ya que sino no se ingresaría al mismo. Una posible salida de este programa sería:

```
Ingrese un entero: 3
¿3 != -1?
Ingrese un entero: -1
¿-1 != -1?
¡Falso! Fin.
```


Matías S. Ávalos© CC BY-NC-SA 4.0

¡Importante!

Las expresiones lógicas dentro de una sentencia de repetición **deben *hacerse* falsas en algún momento**, ya que de lo contrario, estaríamos en un ***bucle infinito*** que causaría que nuestro programa "se cuelgue". Si bien en algunos contextos es deseable tener un *bucle infinito*, este debe ser programado a conciencia, nunca por accidente.

1.2. Hacer/Repetir

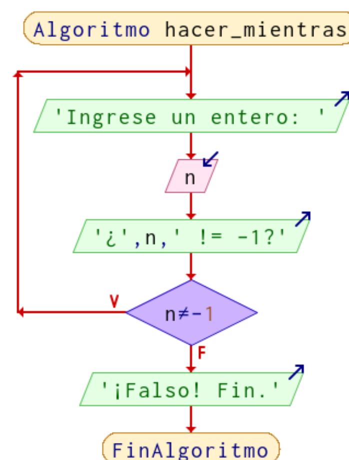
Dependiendo del lenguaje de programación podemos encontrar dos variantes que son similares o hasta podríamos decir *"iguales pero opuestas"*. Una es ***"Hacer … mientras"*** y la otra suele ser conocida como ***"Repetir … hasta que"***. Dependiendo del lenguaje de programación podemos encontrar una de las dos, ambas o ninguna. En C++, que es el lenguaje que utilizamos solo existe el primero de ellos, y es el que fomentaremos a la hora de plantear un algoritmo. Pero el otro es sencillamente reproducible debido a la sutiliza que las separa.

1.2.1. Hacer … Mientras Que

Como se adelantó, en C++ existe únicamente la sentencia `do-while`, la cual empieza un bloque con la palabra reservada `do`, dentro de las llaves se escriben las sentencias a repetir, y al final de las llaves se evalúa la condición a través de una sentencia `while`.

Programa 1: Hacer...mientras

```cpp
#include <iostream>
using namespace std;
int main() {
  int n;
  do {
    cout << "Ingrese un entero: ";
    cin >> n;
    cout<<"¿"<<n<<" != -1?"<<endl;
  } while(n != -1);
  cout << "¡Falso! Fin." << endl;
}
```



A diferencia del `while` expuesto en el apartado anterior, aquí **al menos 1 vez** se ejecutará el bloque, luego se evaluará la condición y se repetirá *mientras* sea verdadera.

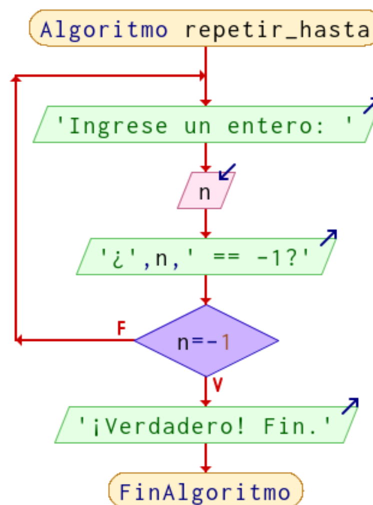
1.2.2. Repetir … Hasta Que

Como se ha indicado, **no existe** este bloque en C++, pero se puede *emular* con un `do-while` aplicando el operador lógico `not` a la condición.

Matías S. Ávalos© CC BY-NC-SA 4.0

Programa 2: Repetir...hasta que

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4    int n;
5    do {
6      cout << "Ingrese un entero: ";
7      cin >> n;
8      cout<<"¿"<<n<<" == -1?"<<endl;
9    } while(not (n == -1));
10   cout << "¡Verdadero! Fin."<< endl;
11 }
```



El flujo a simple vista parece igual, cambia el punto de vista lógico de la expresión a evaluar. *Repetir hasta que la condición sea verdadera.* Lo cual, como podemos observar equivale a utilizar una expresión contraria a la de un *mientras*.

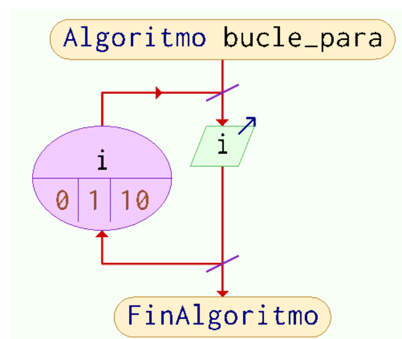
1.3. Para

La sentencia "Para" es una construcción sintética utilizada frecuentemente para ciclar una cantidad finita de veces y, habitualmente, se la utiliza con una variable que auspicia de *"contador"*. Ésta puede incrementar o decrementar según sea necesario.

En `C++` este bucle se implementa con la palabra reservada `for` y dentro de los paréntesis está compuesto por 3 espacios separados por `';'`

Programa 3: Bucle **para**

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4    for(int i = 0; i < 10; i++) {
5      cout << i << "." << endl;
6    }
7  }
```



En este caso, se muestra la inicialización de "i" en 0, incrementando de a 1 hasta 10. En otros diagramas aparece como "(0|10|1)" igual que se escribe en `C++`. La *variable i* se incrementa en cada repetición con el operador matemático `'++'` el cual equivale a `i=i+1`.

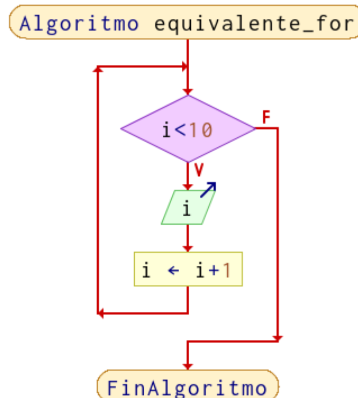
Este algoritmo muestra los números desde el 0 al 9, ya que la segunda expresión es la condición de permanencia del bucle que va desde que `i=0` (*condición inicial*) y mientras `i < 10`, es decir hasta 9.

El `for` anterior es equivalente al siguiente código utilizando un `while`:

Matías S. Ávalos© CC BY-NC-SA 4.0

Programa 4: Equivalente con `while` del `for`

```
#include <iostream>
using namespace std;
int main() {
  int i = 0;
  while(i < 10) {
    cout << i << "." << endl;
    i++;
  }
}
```



Incremento/decremento y asignaciones

- `++`: Incremento, ejemplo `i++` equivale a `i=i+1`.
- `--`: Decremento, ejemplo `i--` equivale a `i=i-1`.
- `+=`: asignación con incremento, ejemplo `i+=2` equivale a `i=i+2`.
- `-=`: asignación con decremento, ejemplo `i-=2` equivale a `i=i-2`.
- `*=`: asignación con multiplicación, ejemplo `i*=2` equivale a `i=i*2`.
- `/=`: asignación con división, ejemplo `i/=2` equivale a `i=i/2`.

2. Ejercitación

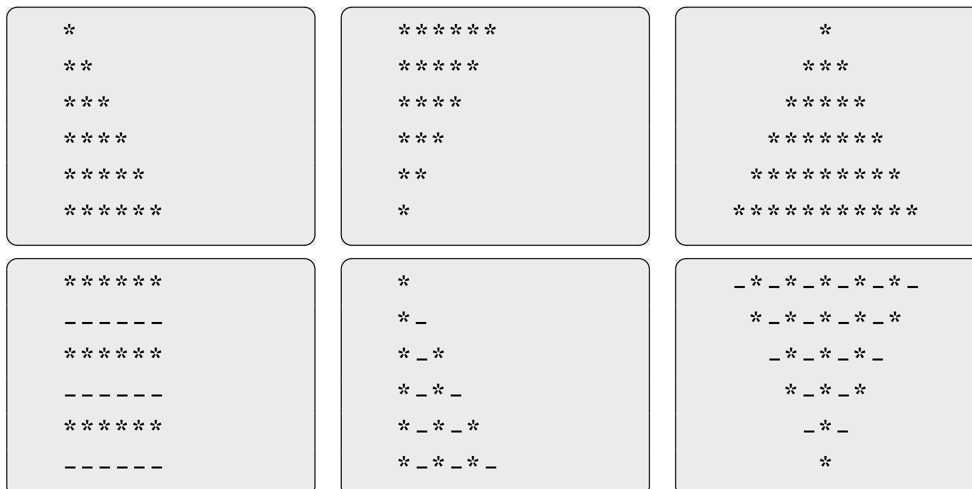
Intente resolver los siguientes problemas planteando los algoritmos en diagramas y codifíquelos en C++:

1. Pida ingresar una nota entre 1 y 10, y si es mayor o igual a 6 muestre el texto "aprobó" de lo contrario "desaprobó".
2. Pida ingresar 3 notas, verifique que sean entre 1 y 10, y luego muestre el promedio.
3. Muestre los primeros 10 número pares.
4. Muestre los primeros 10 número impares.
5. Muestre la tabla del 2.
6. Pida ingresar un número y un texto, repita el texto la cantidad de veces indicadas por el número ingresado.
7. Pida un número entre 1 y 10, luego muestre su tabla de multiplicar.
8. Pida ingresar una cantidad de notas, las notas deben ser entre 1 y 10, si se ingresa una nota inválida, debe pedirse nuevamente. Luego, calcule el promedio entre ellas.

Matías S. Ávalos© CC BY-NC-SA 4.0

9. Haga un programa que muestre la tabla ASCII estándar con el formato N° = X donde 'N°' es el valor decimal y 'X' es el carácter a ASCII.

10. Genere los siguientes patrones con bucles, teniendo en cuenta que la cantidad de filas debe ser variable. Es decir que debe pedir la cantidad de filas al usuario y luego generar los patrones. En los ejemplo se ven solamente 6 filas a modo de ejemplo:



Matías S. Ávalos© CC BY NC SA 4.0