PROGRAMACIÓN I - UNIDAD 3: ESTRUCTURAS DE CONTROL

Autor: Ing. Florencia Marina Anderson

Control de versiones:

Versión	Descripción cambios	Fecha
1.0	Inicial	
1.1	Se cambió el nombre de la estructura repetir hasta por repetir mientras que es lo que usa C. Se cambió ejemplo. Se corrige nombre documento Se corrige un diagrama de flujo	21/4/2021

Contenido

Estructuras de control

Las estructuras de control, son instrucciones que permiten romper la secuencialidad de la ejecución de un programa; esto significa que una estructura de control permite que se realicen unas instrucciones y omitir otras, de acuerdo a la evaluación de una condición.

Al escribir una estructura de control, se deberá tener presente que una correcta tabulación, permitirá que el programa sea más legible y fácil de comprender. Los lenguajes de programación, realizan esta tabulación de forma automática, siempre y cuando se escriban correctamente el inicio y final de cada estructura.

Existen 2 tipos de estructuras de control:

- Selectivas
- Repetitivas

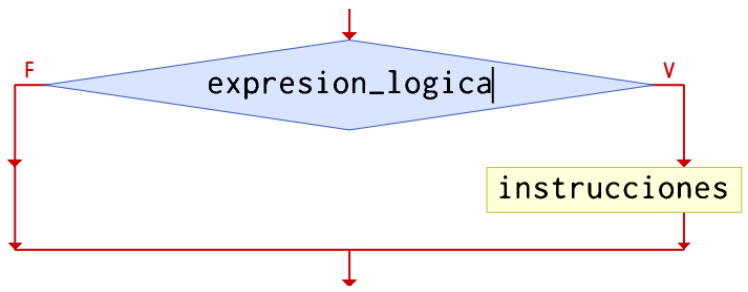
Estructuras de control selectivas

Estas estructuras permiten seleccionar un camino a ejecutarse entre dos o más opciones por única vez.

Instrucción "Si"

En este caso una instrucción se ejecuta si se cumple una expresión lógica.

Pseudocódigo	Diagrama de flujo

Proceso Principal
  Si expresion_logica
  Entonces
  instrucciones..
  FinSi
FinProceso
	

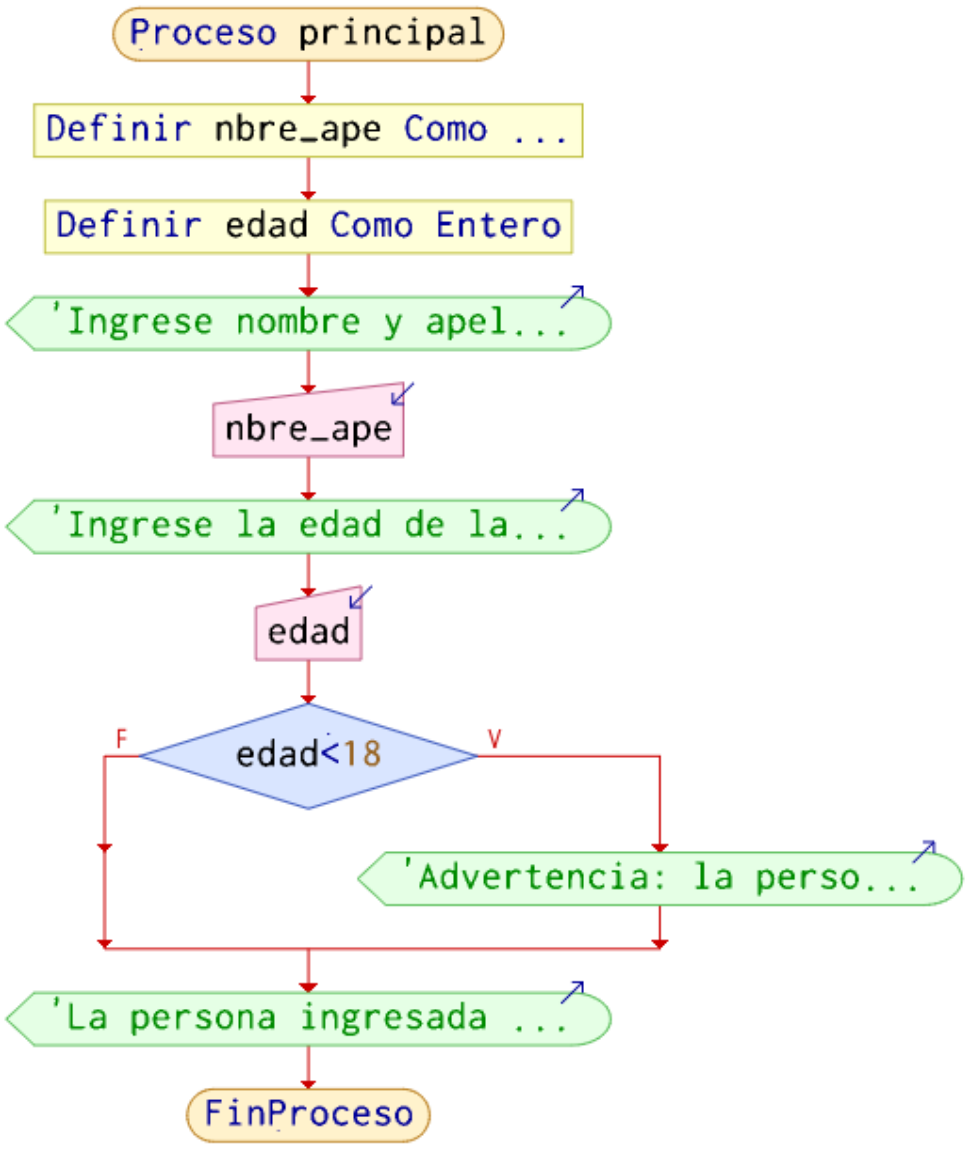
Ejemplo 1

Presentamos un ejemplo de un programa que pide al usuario que ingrese el nombre y apellido de una persona y la edad. Luego, el programa muestra los datos ingresados al usuario, pero antes, se muestra una advertencia al usuario si la persona ingresada es menor de edad.

Ejemplo en pseudocódigo

```
Proceso principal

    Definir nbre_ape Como Cadena
    Definir edad Como Entero

    Escribir "Ingrese nombre y apellido de la persona"
    Leer nbre_ape

    Escribir "Ingrese la edad de la persona"
    Leer edad

    Si edad <18 Entonces
        Escribir "Advertencia: la persona es menor de edad"
    FinSi

    Escribir "La persona ingresada es: " , nbre_ape , " (",edad," años)"

FinProceso
```

Ejemplo en diagrama de flujo

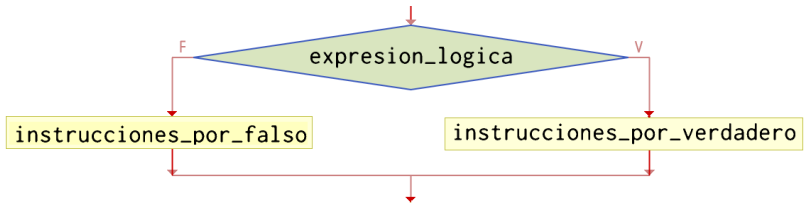


Ejemplo en C

```c
#include<stdio.h>
#define MAX_STRLEN 256

int main() {
    int edad;
    char nbre_ape[MAX_STRLEN];
    printf("Ingrese nombre y apellido de la persona\n");
    scanf("%s",nbre_ape);
    printf("Ingrese la edad de la persona\n");
    scanf("%i",&edad);
    if (edad<18) {
        printf("Advertencia: la persona es menor de edad\n");
    }
    printf("La persona ingresada es: %s (%i años)\n",nbre_ape,edad);
    return 0;
}
```

Instrucción “Si completo”

En el caso del “Si simple” solo se ejecutan instrucciones si la condición es verdadera, en cambio, con la instrucción de “Si completo” ejecutamos una serie de instrucciones para cuando la condición es verdadera y otra serie de instrucciones para cuando la condición es falsa.

Pseudocódigo	**Diagrama de Flujo**
Si expresion_logica **Entonces** instrucciones_por_verdadero **SiNo** instrucciones_por_falso **FinSi**	

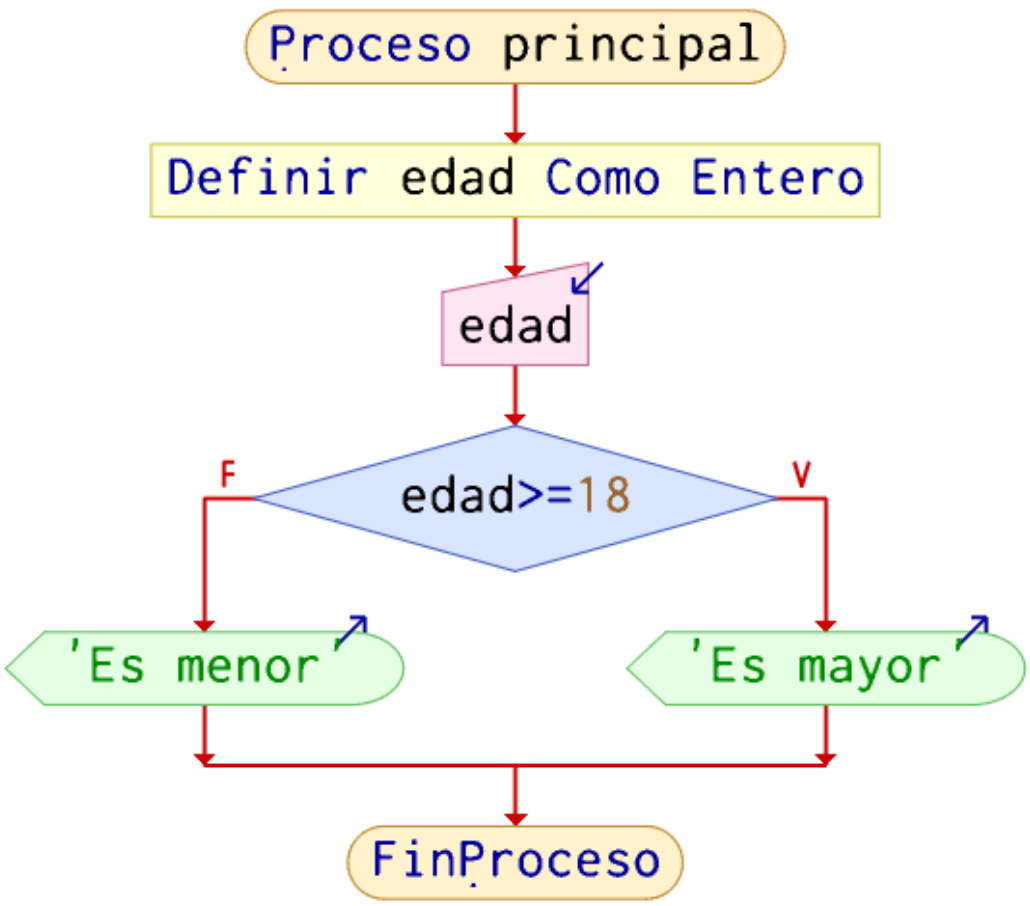
Ejemplo 2

Presentamos un ejemplo de un programa que pide al usuario que le ingrese la edad de una persona, si la edad es >=18, entonces se muestra un mensaje al usuario indicando que la persona es mayor de edad, caso contrario, se muestra un mensaje al usuario indicando que es menor de edad.

Ejemplo en pseudocódigo

```
Proceso principal

    Definir edad Como Entero

    Leer edad
    Si edad≥18 Entonces
        Escribir "Es mayor"
    SiNo
        Escribir "Es menor"
    FinSi

FinProceso
```

Ejemplo en diagrama de flujo:

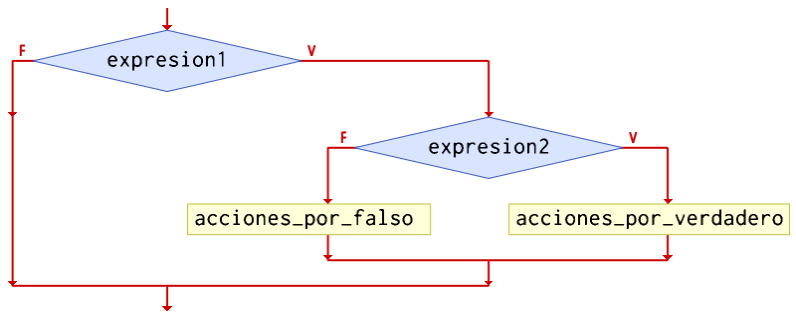


Ejemplo en C

```c
#include<stdio.h>
int main() {
    int edad;
    scanf("%i",&edad);
    if (edad>=18) {
        printf("Es mayor\n");
    } else {
        printf("Es menor\n");
    }
    return 0;
```

Instrucción “Si anidado”

Es una estructura “Si” dentro de otra.

Pseudocódigo	Diagrama de Flujo
Si expresion1 **Entonces** **Si** expresion2 **Entonces** acciones_por_verdadero **SiNo** acciones_por_falso **FinSi** **FinSi**	

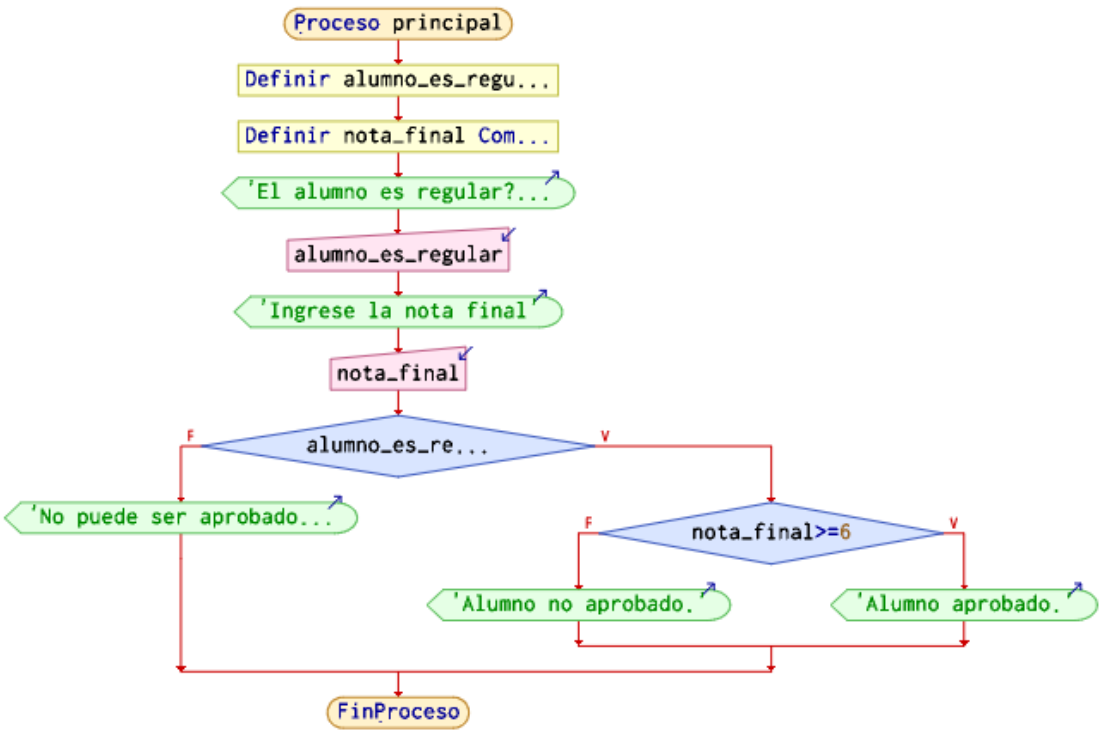
Ejemplo 3

Tomemos como ejemplo un programa en el que el usuario indica la condición de regularidad de un alumno y la nota final. Entonces el sistema debe informar si el alumno está aprobado o no. Para que un alumno esté aprobado debe estar regular y la nota debe ser mayor a 6. En caso de que no esté regular el sistema debe informar la situación y en caso de que lo esté debe informar si está aprobado o no según la nota

Ejemplo en pseudocódigo

```
Proceso principal

    Definir alumno_es_regular Como Logica
    Definir nota_final Como Numerico

    Escribir "El alumno es regular? Ingrese verdadero o falso"
    Leer alumno_es_regular

    Escribir "Ingrese la nota final"
    Leer nota_final

    Si alumno_es_regular
        Si nota_final≥6
            Escribir "Alumno aprobado."
        Sino
            Escribir "Alumno no aprobado."
        FinSi

    SiNo
        Escribir "No puede ser aprobado. No posee condición de alumno regular."
    FinSi


FinProceso
```

Ejemplo en diagrama de flujo:



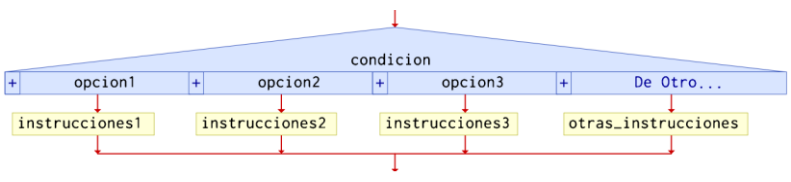
Ejemplo en C:

```c
#include <stdio.h>
#include <stdbool.h>

int main() {
    bool alumno_es_regular;
    float nota_final;
    printf("El alumno es regular? Ingrese verdadero o falso\n");
    scanf("%i",&alumno_es_regular);
    printf("Ingrese la nota final\n");
    scanf("%f",&nota_final);
    if (alumno_es_regular) {
        if (nota_final>=6) {
            printf("Alumno aprobado.\n");
        } else {
            printf("Alumno no aprobado.\n");
        }
    } else {
        printf("No puede ser aprobado. No posee condición de alumno regular.\n");
    }

    return 0;
}
```

Instrucción “Según” (o Switch)

Ésta estructura de control es una alternativa a la instrucción “Si” y se caracteriza por ofrecer la posibilidad de elegir entre más de dos opciones.

En esta instrucción no existe una condición explícita en su sintaxis, sin embargo, de acuerdo al lenguaje es posible evaluar un caracter o un número, siendo ésta una limitación que impide la escritura directa de condiciones.

Pseudocódigo	**Diagrama de Flujo**
Segun condicion **Hacer** opcion1: instrucciones1 opcion2: instrucciones2 opcion3: instrucciones3 **De Otro Modo**: otras_instrucciones **Fin Segun**	

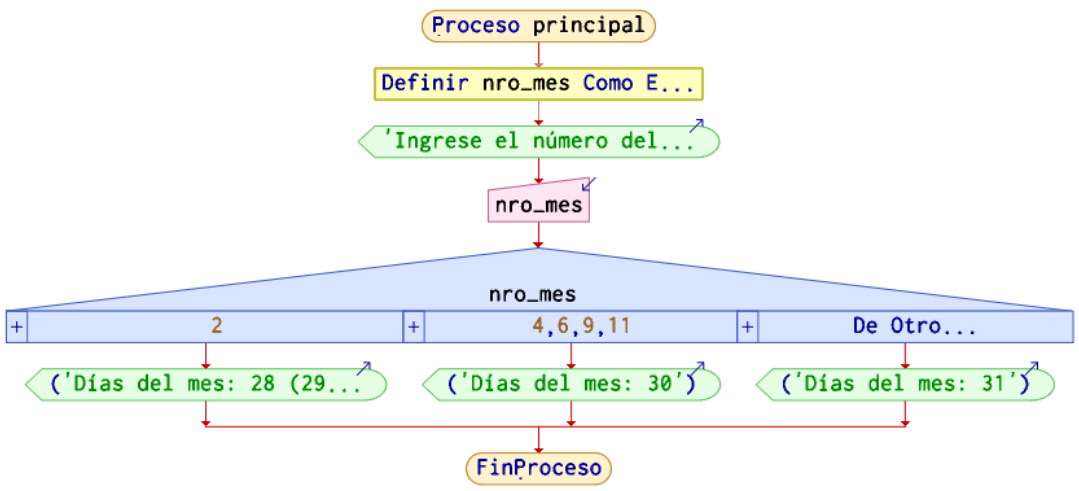
Ejemplo 4

Vamos a crear un programa que a partir del número del mes nos informe la cantidad de días que posee el mismo.

Ejemplo en pseudocódigo:

```
Proceso principal

    Definir nro_mes Como Entera

    Escribir "Ingrese el número del mes:"
    Leer nro_mes

    Segun nro_mes Hacer
        2:
            escribir("Días del mes: 28 (29 si el año es bisiesto)")
        4,6,9,11:
            escribir("Días del mes: 30")

        De Otro Modo:
            escribir("Días del mes: 31")
    Fin Segun

FinProceso
```

Ejemplo en diagrama de flujo:



Ejemplo en C:

```c
#include<stdio.h>

int main() {
    int nro_mes;
    printf("Ingrese el número del mes:\n");
    scanf("%i",&nro_mes);
    switch (nro_mes) {
    case 2:
        printf("%s\n",("Días del mes: 28 (29 si el a\xfffdo es bisiesto)"));
        break;
    case 4: case 6: case 9: case 11:
        printf("%s\n",("Días del mes: 30"));
        break;
    default:
        printf("%s\n",("Días del mes: 31"));
    }
    return 0;
}
```

Estructuras de control repetitivas

Las estructuras de control repetitivas, son aquellas que permiten ejecutar un conjunto de instrucciones varias veces, de acuerdo al valor que genere la expresión relacional y/o lógica.

A estas estructuras se les conoce también como ciclos o bucles, por su funcionamiento.

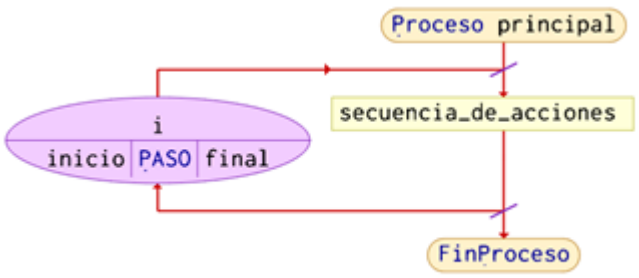
Existen 3 tipos de estructuras repetitivas:

- Para
- Mientras
- Repetir - Mientras

Instrucción Para (For)

Esta instrucción permite que se ejecute una serie de instrucciones una cantidad N de veces.

Pseudocódigo	**Diagrama de Flujo**

`Proceso principal`
`  Para i<-inicio Hasta final Con Paso paso Hacer`
`  secuencia_de_acciones`
`  Fin Para`
`FinProceso`
	

Se lee de la siguiente forma: Para la variable numérica **i** que empieza en valor inicial **inicio**, hasta que llegue al valor final **final**, incrementándose o decrementándose en el valor de **paso** haga la secuencia de acciones.

El paso por lo general es **1** (en cada iteración la variable numérica se incrementa en 1), o **-1** (en cada iteración la variable numérica se decrementa en 1), aunque se puede incrementar o decrementar en otra cantidad que se desee como por ejemplo con un paso 2 o -2

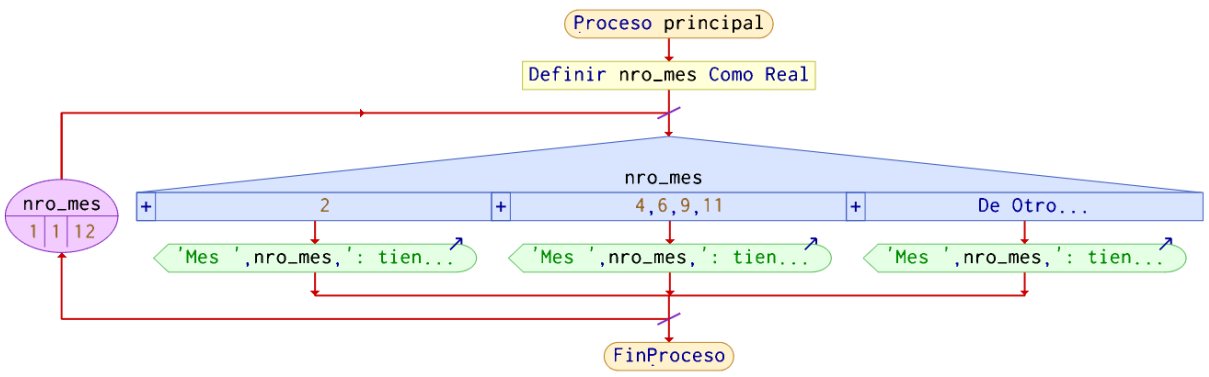
Ejemplo 5

Hacer un programa que muestre los meses del año y la cantidad de días que posee cada uno.

Ejemplo en pseudocódigo:

```
Proceso principal

    Definir nro_mes Como Numerica

    Para nro_mes←1 Hasta 12 Con Paso 1 Hacer

        Segun nro_mes  Hacer
            2:
                Escribir 'Mes ',nro_mes,': tiene 28 días (29 si el año es bisiesto)'
            4,6,9,11:
                Escribir 'Mes ',nro_mes,': tiene 30 días'
            De Otro Modo:
                Escribir 'Mes ',nro_mes,': tiene 31 días'
        FinSegun

    Fin Para

FinProceso
```

Ejemplo en diagrama de flujo:



Ejemplo en C:

```c
#include<stdio.h>

int main() {
    int nro_mes;
    for (nro_mes=1;nro_mes<=12;nro_mes+=1) {
        switch (nro_mes) {
        case 2:
            printf("Mes %i: tiene 28 días (29 si el año es bisiesto)\n",nro_mes);
            break;
        case 4: case 6: case 9: case 11:
            printf("Mes %i: tiene 30 días\n",nro_mes);
            break;
        default:
            printf("Mes %i: tiene 31 días\n",nro_mes);
        }
    }
    return 0;
}
```

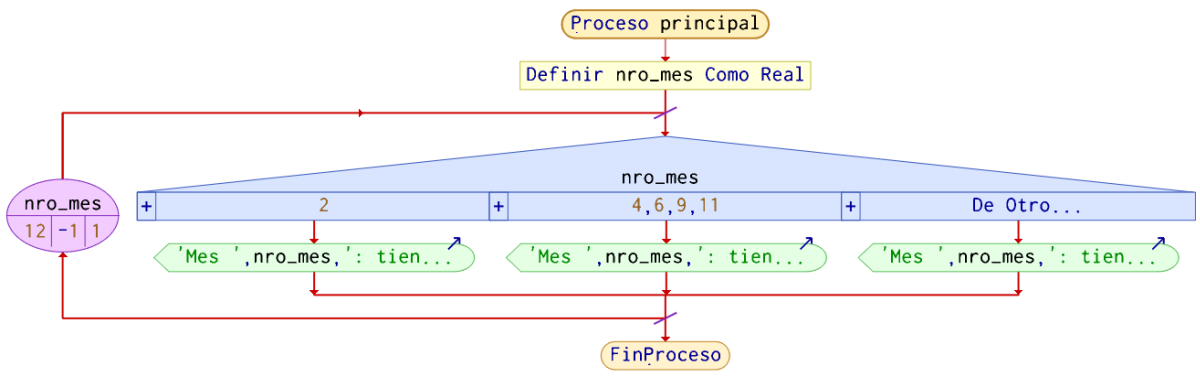
Ejemplo 6

Ahora hagamos el mismo ejercicio anterior, pero en este caso, mostrar los meses en orden inverso, es decir, desde diciembre a enero. Nótese que en este caso, el paso debe decrementar en uno, ya que el valor inicial de la variable es 12 y el valor final 1

Ejemplo en pseudocódigo:

```
Proceso principal

    Definir nro_mes Como Numerica

    Para nro_mes←12 Hasta 1 Con Paso -1 Hacer

        Segun nro_mes  Hacer
            2:
                Escribir 'Mes ',nro_mes,': tiene 28 días (29 si el año es bisiesto)'
            4,6,9,11:
                Escribir 'Mes ',nro_mes,': tiene 30 días'
            De Otro Modo:
                Escribir 'Mes ',nro_mes,': tiene 31 días'
        FinSegun

    Fin Para

FinProceso
```

Ejemplo en diagrama de flujo:



Ejemplo en C:

```c
#include<stdio.h>

int main() {
    int nro_mes;
    for (nro_mes=12;nro_mes>=1;nro_mes-=1) {
        switch (nro_mes) {
        case 2:
            printf("Mes %i: tiene 28 días (29 si el año es bisiesto)\n",nro_mes);
            break;
        case 4: case 6: case 9: case 11:
            printf("Mes %i: tiene 30 días\n",nro_mes);
            break;
        default:
            printf("Mes %i: tiene 31 días\n",nro_mes);
        }
    }
    return 0;
}
```

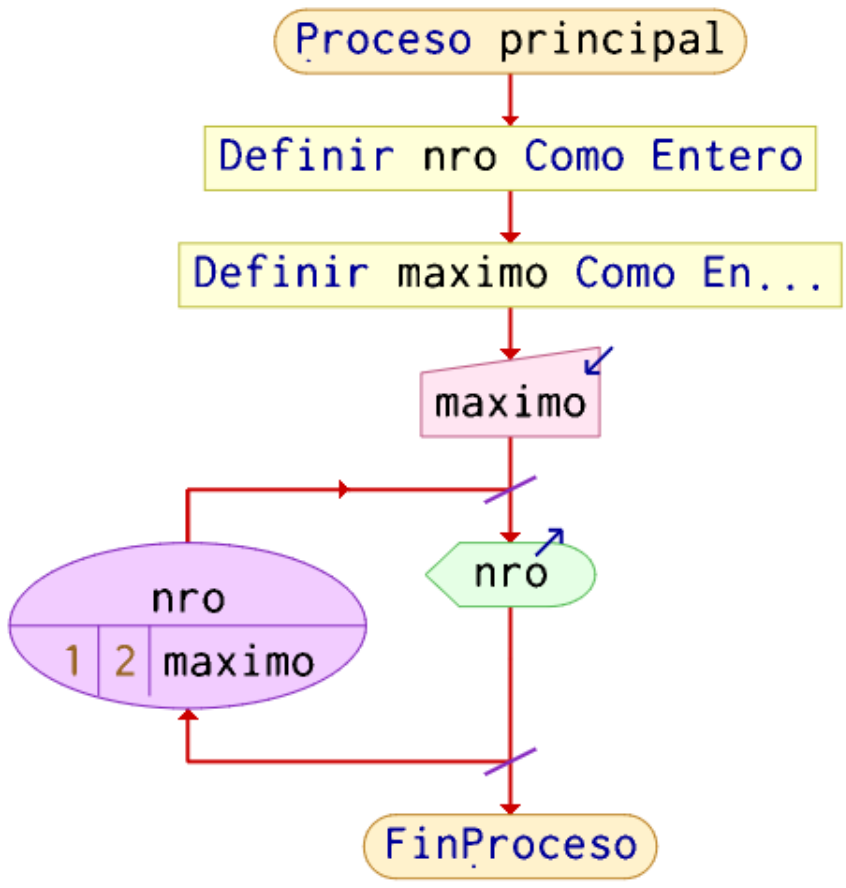
Ejemplo 7

Ahora realicemos un programa que muestre todos los números impares del conjunto de números enteros que van desde el 1 hasta N, siendo N ingresado por el usuario

Ejemplo en pseudocódigo:

```
Proceso principal
    Definir nro Como Entera
    Definir maximo como Entera
    Leer maximo
    Para nro←1 Hasta maximo Con Paso 2 Hacer
        Escribir nro
    FinPara
FinProceso
```

Ejemplo en diagrama de flujo:

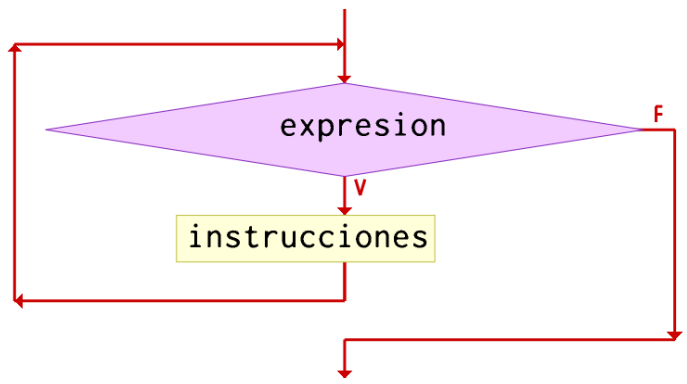


Ejemplo en C:

```c
#include<stdio.h>

int main() {
    int maximo;
    int nro;
    scanf("%i",&maximo);
    for (nro=1;nro<=maximo;nro+=2) {
        printf("%i\n",nro);
    }
    return 0;
}
```

Instrucción Mientras (While)

Ésta estructura nos permite repetir una serie de instrucciones una cantidad indefinida de veces siempre que la condición sea verdadera, cuando la condición pase a ser falsa no se volverá a iterar.

Pseudocódigo	Diagrama de Flujo
Mientras expresion_logica **Hacer** instrucciones **Fin Mientras**	

La estructura **Mientras** necesita que antes del inicio de la misma se inicialicen las variables que serán evaluadas en la condición y además el incremento del paso se da dentro del conjunto de sentencias a iterar. Con la estructura **Para** no debíamos hacer esto ya que al inicio de la estructura se definen el paso y el valor inicial de la variable que va cambiando su valor en cada iteración.

Entonces, ¿por qué no usamos siempre la estructura **Para**?

Bueno, la estructura Mientras, se utiliza cuando no se conoce de ante mano la cantidad de iteraciones.

Además, la condición que evalúa seguir o no seguir con otra vuelta puede ser más compleja utilizando **Mientras** y utilizando la estructura **Para** estamos limitados a una condición del tipo expresión relacional.

Ejemplo 8

Ahora realicemos un programa que muestre todos los números impares del conjunto de números enteros que van de desde 1 a N, siendo N un valor ingresado por el usuario, pero utilizando la estructura **Mientras**.

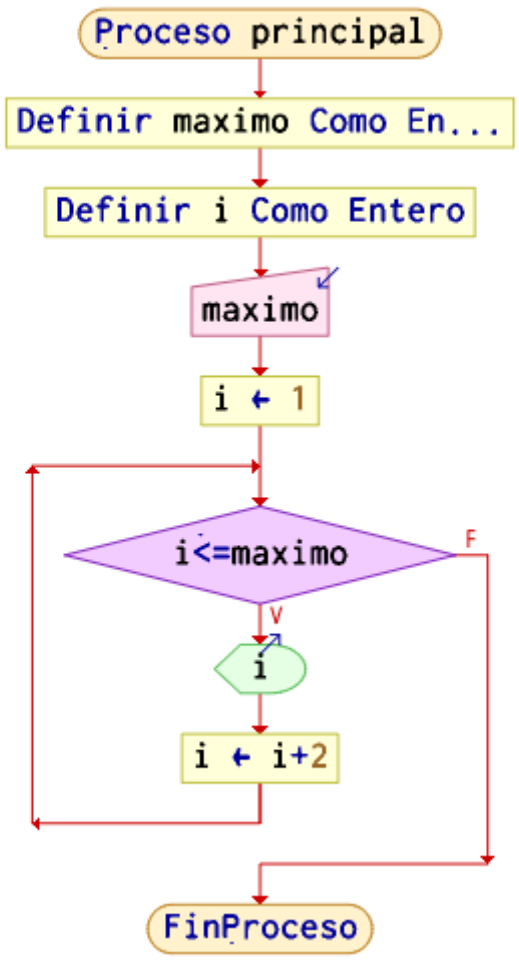
Nótese que para que éste caso funcione correctamente necesitamos inicializar la variable **i** antes de llamar a la estructura **Mientras** y luego dentro de la estructura debemos incrementar el valor de **i**.

Éste ejemplo se presenta a modo de poder apreciar la diferencia entre el bucle **Para** y el bucle **Mientras**, pero se entiende que a la hora de elegir la estructura adecuada se elegirá por la más simple, en este caso la estructura **Para.**

Ejemplo en pseudocódigo:

```
Proceso principal

    Definir maximo como Entero
    Definir i como Entero
    Leer maximo
    i ← 1
    Mientras i ≤ maximo Hacer
        Escribir i
        i=i+2
    Fin Mientras

FinProceso
```

Ejemplo en diagrama de flujo:

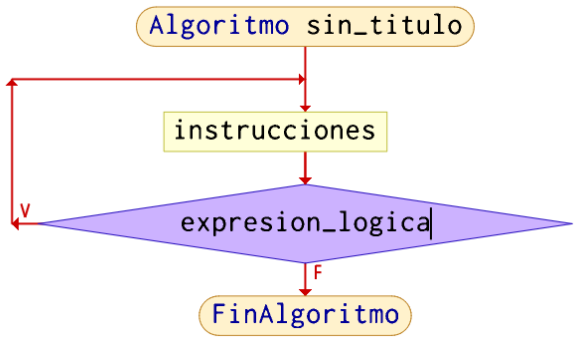


Ejemplo en C:

```c
#include<stdio.h>

int main() {
    int i;
    int maximo;
    scanf("%i",&maximo);
    i = 1;
    while (i<= maximo) {

        printf("%i\n",i);
        i = i+2;

    }

    return 0;
}
```

Instrucción Repetir - Mientras (Do-While)

Ésta estructura es similar a la de **Mientras** con la diferencia que el primer conjunto de instrucciones se ejecuta siempre, y la condición para saber si es necesario repetir se ejecuta al final de la estructura, cuando en el **Mientras** se ejecuta al inicio de la estructura

Pseudocódigo	Diagrama de Flujo
Repetir instrucciones **Mientras Que** expresion_logica	

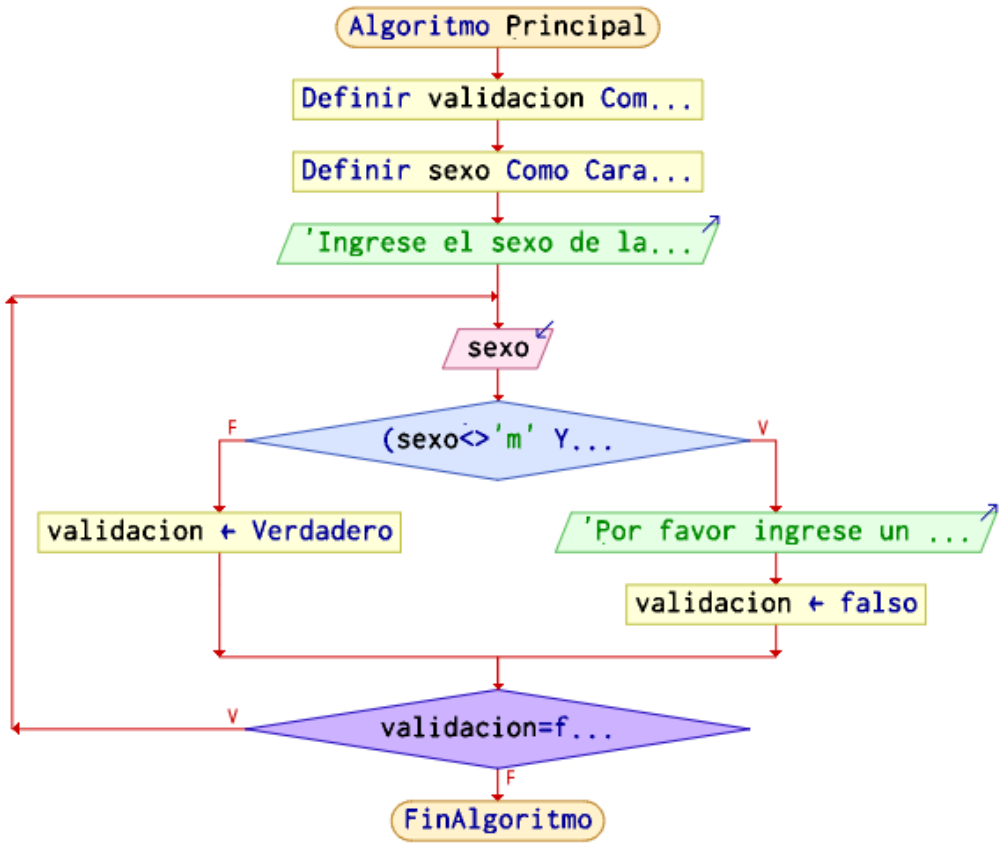
Ejemplo 9

Hagamos un programa que solicite al usuario el sexo de una persona (‘f’ o ‘m’), luego que valide si el valor ingresado es correcto, y en caso de no serlo que vuelva a solicitar el ingreso.

Ejemplo en pseudocódigo:

```
Algoritmo Principal
    Definir validacion Como Logico
    Definir sexo Como Caracter
    Escribir "Ingrese el sexo de la persona: f: femenino - m: masculino."

    Repetir
        Leer sexo
        Si(sexo <> 'm' y sexo<>'f')
            Escribir "Por favor ingrese un valor válido."
            validacion←Falso
        SiNo
            validacion←Verdadero
        FinSi
    Mientras Que validacion=falso

FinAlgoritmo
```

Ejemplo en diagrama de flujo:



Ejemplo en C:

```c
#include<stdio.h>
#include<stdbool.h>
#include<string.h>

int main() {
    char sexo[1];
    bool validacion;
    printf("Ingrese el sexo de la persona: f: femenino - m: masculino.\n");
    do {
        scanf("%s",sexo);
        if ((strcmp(sexo,"m")!=0&strcmp(sexo,"f")!=0)) {
            printf("Por favor ingrese un valor válido.\n");
            validacion = false;
        } else {
            validacion = true;
        }
    } while (validacion==false);
    return 0;
}
```

Programación estructurada

Todo programa computarizado puede ser escrito con un alto grado de estructuración, permitiendo hacer más sencillas tareas como prueba, mantenimiento y modificación.

Mediante la programación estructurada es posible leer la codificación de cualquier programa de inicio a fin en forma continua, siguiendo la lógica definida por el programador.

Éste paradigma, es un modelo de alta precisión permitiendo trabajar en equipo, acoplando módulos a la idea original, logrando programas fáciles de ser leídos, mantenidos y modificados por otros programadores.

Dicha programación se desarrolla utilizando tres estructuras lógicas de control:

1. **Secuencia**: sucesión de instrucciones.
2. **Selección o Decisión**: bifurcación condicional.
3. **Repetición**: ejecuciones repetidas con diferentes datos.

Tales estructuras de control, combinadas entre ellas, controlan los datos de forma tal de generar información. Todo programa estructurado está compuesto de módulos o procedimientos, logrados por refinamientos sucesivos, permitiendo ser leído en secuencia, desde el comienzo hasta el final sin perder el concepto del programa ni del módulo.

Un programa estructurado, además de tener una excelente presentación, es mucho más fácil de actualizar ante nuevos requerimientos.

Un programa estructurado, además de tener una excelente presentación, es mucho más fácil de actualizar ante nuevos requerimientos.