UNIVERSIDAD NACIONAL DE LOMAS DE ZAMORA

FACULTAD DE INGENIERÍA

INGENIERÍA MECATRÓNICA

FUNDAMENTOS DE LOS COMPUTADORES DIGITALES

MÓDULO 3

INTRODUCCIÓN AL LENGUAJE C

CURSO 2024

AUTORES

Diego Servetto - Lic. en Informática - Esp. en Gestión Tecnológica (UNLZ)
Ezequiel Blanca - Ingeniero en Electrónica (UTN)
Bianca Rodriguez - Estudiante avanzado de Ing. Mecatrónica (UNLZ)
Ramiro Gazillón - Estudiante avanzado de Ing. Mecatrónica (UNLZ)
Kenneth Gonzalez - Estudiante avanzado de Ing. Mecatrónica (UNLZ)
Néstor Maceda - Licenciado en Informática Educativa (UNLa)

ÍNDICE

INTRODUCCIÓN A LA LÓGICA

Introducción

La programación informática como lenguaje formal de expresión de algoritmos, tiene íntima relación tanto con el lenguaje informal natural (en su sintaxis y semántica) como al lenguaje formal de la matemática.

Por ello, para aprender a programar correctamente es importante poseer conceptos elementales de matemática que nos permitan comprender los aspectos fundamentales de los lenguajes de programación de computadoras.

Por ejemplo, en la unidad anterior, realizamos la programación de las **operaciones matemáticas aritméticas fundamentales** de sumar, restar, multiplicar y dividir. Y que no hubiéramos podido programar las mismas en una computadora, si no supiéramos realizarlas en nuestra vida cotidiana. Contenidos elementales que aprendimos en la educación general básica.

En este apartado, se realizará una breve introducción a la **lógica matemática** y posteriormente a **lógica computacional** necesarias para la comprensión posterior de contenidos referidos a conceptos de:

- Operaciones lógicas.
- Estructura de datos
- Funciones (unidades posteriores)

Definición de lógica

Parte de la filosofía que estudia las formas y principios generales que rigen el **conocimiento** y el **pensamiento** humano, considerado puramente en sí mismo, sin referencia a los objetos. El término de lógica deriva del griego antiguo λογική logikḗ, que significa “dotada de razón, intelectual, dialéctica, argumentativa” y que a su vez viene de λόγος (lógos), “palabra, pensamiento, idea, argumento, razón o principio”.

Tipos de lógica

La capacidad de pensamiento del hombre no tiene límites. Tantas culturas nos indican que hay muchas formas de pensar el mundo y esto ha generado en forma natural muchos tipos de razonamientos o bien, tantos tipos de lógicas como disciplinas tecnológicas y científicas existen. Por ejemplo, sólo en filosofía, en su estudio de la lógica, encontramos un amplio rango de teorías: lógica tradicional o clásica, la lógica no clásica, la lógica modal, la lógica libre, etc.
Por ello, en nuestro caso particular, reduciremos nuestro estudio de lógica, solo al tema de la lógica computacional de programaciòn que se sustenta en la **lógica matemática.**

Lógica proposicional

Definición

La lògica proposicional, tambièn conocida como lógica de enunciados o lógica de orden cero, es una lògica matemàtica que se basa en la lògica aristotèlica con operaciones sobre entidades atòmicas denominadas proposiciones.La **lógica proposicional,** como su nombre lo indica, es aquella parte de la matemática que estudia y argumenta un razonamiento a través de **operaciones lógicas** cuyos operandos atómicos son denominados **proposiciones** y sus operadores denominados **conectivos lógicos.**

Proposición

Concepto

Una **proposición** es una **oración declarativa** (afirmativa o negativa) de la cual podemos decir que es verdadera o falsa (pero no ambas). Es una entidad atómica de la lógica proposicional, portadora de **valor de verdad**. Una proposición es una oración que afirma o cuenta algo, sobre lo que podemos decir que es cierto o no lo es. Las proposiciones, usan la función informativa del lenguaje (también llamada aseverativa).
A continuación se ejemplifican los distintos tipos de oraciones con el objeto de comprender aún mejor la diferencia con las oraciones del tipo declarativas de las proposiciones

- ❏ Oraciones interrogativas: ¿vamos al cine? - (Pregunta)
- ❏ Oraciones imperativas: Ven ahora - (Orden)
- ❏ Oraciones exclamativas: ¡Qué susto me diste! (Emoción)
- ❏ Oraciones declarativas: 1978 es un año bisiesto (Enunciado que puede ser V ó F)

Notación

La notación de una proposición se realiza normalmente con la letra p y las de su entorno sucesivo: q, r, s, t.
Las proposiciones se suelen anotar con la letra p en minúscula. En caso de requerir más letras, estas pueden ser: p, q, r, s, ... Es importante observar que dichas letras no son variables sino constantes.
Ejemplo de proposiciones son:

$p_1 : Doce\ es\ mùltipo\ de\ tres$

$p_2 : La\ tierra\ es\ plana$

$q : 3 + 5 = 8$

$r : 3 - 5 = 1$

s : $El\ nùmero\ 5\ es\ mayor\ al\ 3$

Valor de Verdad

Como se mencionò anteriormente una proposiciòn es una oraciòn declarativa de la cual podemos decir que es verdadera o falsa. Para indicar el valor de verdad de una proposición se indica con una V.

$$V(p_1) = V(Doce\ es\ mùltipo\ de\ tres) = True$$

$$V(p_2) = V(La\ tierra\ es\ plana) = Falso$$

$$V(q) = V(3 + 5 = 8) = Verdadero$$

$$V(r) = V(3 - 5 = 1) = Falso$$

$$V(s) = V(El\ nùmero\ 5\ es\ mayor\ al\ 3) = Verdadero$$

Operaciones lógicas

Conjunción

La operación lógica de conjunción (simbolizado con el operador o conectivo $\wedge$) es aquella que solo es **verdadera** cuando todos sus operandos (proposiciones) son todas verdaderas. Caso contrario siempre da falso. En el lenguaje natural es representado con la letra (y).

p	q	(p^q)
F	F	F
F	V	F
V	F	F
V	V	V

Esta tabla es conocida como **tabla de verdad** de la **conjunción**. En esta tabla se puede observar todas las combinaciones de todos los valores de verdad posibles para **p** y **q**. Observamos también que normalmente se pone **p** y **q** pero lo correcto sería poner **V(p)** y **V(q)** y **V(p∧q)** ya que lo que se coloca en la tabla son sus valores de verdad y no el enunciado. Pero es normal en matemática ahorrar notación y sobreentender sintaxis. Tal es el caso de las operaciones aritméticas cuando no colocamos un signo más delante de un número positivo.

Disyunción

La operación lógica de disyunción (conectivo lógico u operador: $\vee$) es aquella que solo es falsa cuando todas las proposiciones son falsas. En el lenguaje natural es representado con la letra (o) para formar oraciones.

p	q	(pvq)
F	F	F
F	V	V
V	F	V
V	V	V

Esta tabla es conocida como **tabla de verdad** de la **disyunción**.En esta tabla podemos observar que solo es Falsa cuando todas sus

proposiciones son falsas. Es el caso de la primera fila (F F F). Para todos los demás casos es Verdadera.

p	(~p)
F	V
V	F

Negación

La operación lógica de negación, invierte el valor de verdad de la proposición. Si la proposición es falsa al agregarle el conectivo unario de negación, transforma el valor en verdadero. De la misma forma si el valor de la proposición era verdadero al negarse se obtiene un falso. En este caso para la negación se empleó el símbolo (~) pero también es común el empleo del símbolo (¬).

Proposiciones simples y compuestas

Las proposiciones simples son aquellas que no tienen otras oraciones dentro de sí mismas. Las proposiciones compuestas son aquellas que contienen dentro de sí más de una proposición simple u oración.
Por ejemplo, las p, q y r son proposiciones simples.

p: *La tierra es plana*
q : *Las vacas vuelan*
r : *Estoy loco*

En cambio la presente oración es una proposición compuesta.

s: *La tierra es plana o las vacas vuelan, pero no estoy loco*

En este caso, “s” es una oración compuesta porque está formada por “p”, “q” y “r”.
Si reemplazamos los enunciados por las variables nos queda:

s: p *o* q *pero no* r

Ahora reemplacemos los conectivos lógicos

s: $(p \vee q) \wedge (\sim r)$

Se reemplazó el conectivo lingüístico o por el conectivo lógica ∨, el conectivo lingüístico pero por ∧ el no por ∼. Puede observarse que el conectivo lógico ∧ era equivalente al conectivo lingüístico y, sin embargo, ahora fue el sustituto de la palabra pero. Como el lenguaje natural es informal, existen muchas formas de expresar los distintos conectores lógicos.
En este ejemplo observamos dos notaciones de la proposición “s”, una en lenguaje natural y otra en lenguaje matemático formal.

¿Cómo saber el valor de verdad de una proposición compuesta?

$$s\colon (p \vee q) \wedge (\sim r)$$

Pues hay que analizarla por partes. Vamos reemplazando cada proposición por su valor de verdad. Sabemos que obviamente, V(p) = Falso, V(q) = Falso y V(r) = Verdadero.
Por lo tanto nos queda:

$$s\colon (p \vee q) \wedge (\sim r)$$
$$V(s)\colon (F \vee F) \wedge (\sim V)$$

Luego vamos resolviendo la operación lógica siguiendo un orden de jerarquía, primero los conectivos unarios, luego los paréntesis y finalmente los conectivos binarios y teniendo en cuenta las tablas de verdad mencionadas.

Por lo tanto, nos queda reemplazando:

$$V(s)\colon (F \vee F) \wedge (\sim V)$$
$$V(s)\colon \quad (F) \quad \wedge \quad (F)$$
$$V(s)\colon \qquad F$$

El valor de verdad de la proposición compuesta “s” es entonces Falso. Estos cálculos se pueden hacer mentalmente o bien mediante una calculadora lógica online[1]

Lógica de programación

En la lógica computacional, empleamos los mismo conceptos pero con otros símbolos, dependiendo del lenguaje de programación utilizado para representar un algoritmo.
De acuerdo a los contenidos de nuestra cátedra expresaremos los mismos conforme a la notación del lenguaje C

Lenguaje Natural	Lógica proposicional	Lenguaje programación C

[1] https://calculadorasonline.com/generador-de-tablas-de-verdad-logica-proposicional-algebra-booleana/

o	v	||
y	∧	&&
no	~	!

Estructuras de Control - Condicionales

Las estructuras de control permiten controlar la forma en que se ejecutarán las sentencias del programa.

En los ejemplos vistos hasta ahora, siempre se ejecutaban todas las líneas de código, una a continuación de la otra. En esta sección veremos que es posible utilizar estructuras de control las cuales nos permitirán evaluar condiciones para decidir si ejecutar o no una sección del programa.

Las mismas modifican la ejecución secuencial. Esto se realiza mediante la evaluación de una condición, si la misma se cumple, se ejecutará un bloque de código.
A continuación veremos diferentes tipos de condicionales.

Estructura de control simple "if"

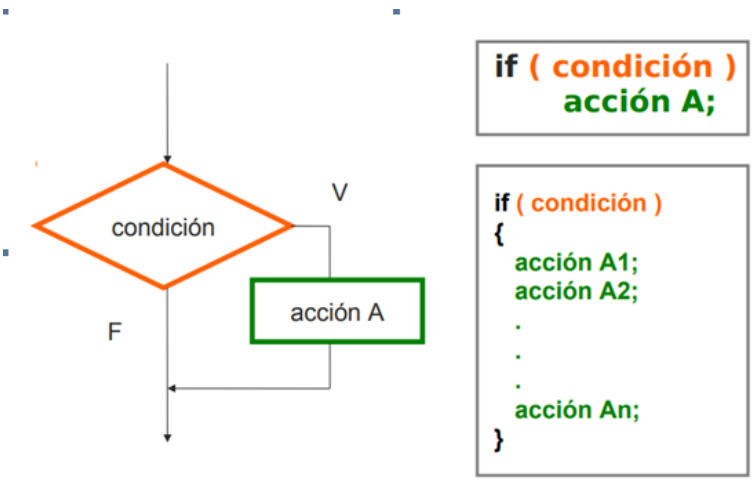
A este tipo de estructura se la denomina "if". En el "if" se evalúa una condición, si la misma es verdadera, se ejecuta el bloque de código que tiene a continuación. Si la condición a evaluar no es verdadera, se saltea el bloque que tiene debajo del "if".

Ejemplo:

```
if (condicion)
{
    // Bloque de código que se ejecuta si la condición se cumple

}
// Siguiente línea a ejecutar
```

A continuación se muestra el diagrama de flujo de este tipo de estructuras:



Ejemplo:
Supongamos que obtenemos un entero mediante la función scanf y queremos verificar si el mismo es mayor a cero:

```
if (numero>0)
{
        printf("El numero ingresado es mayor a cero\n");
}
```

En este ejemplo, cuando se ingresa un número menor o igual a cero, no se imprime nada.

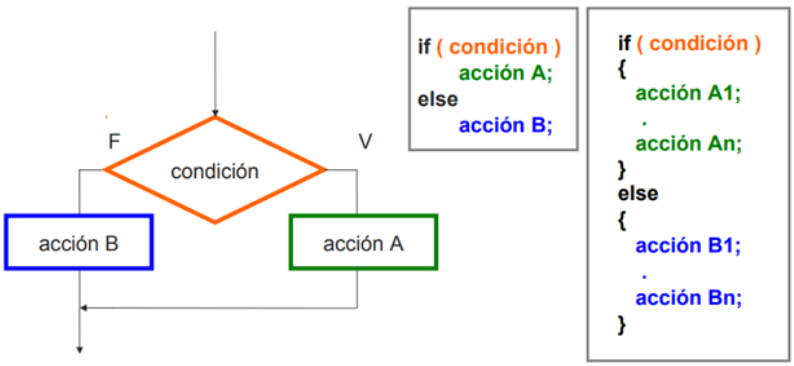
Estructura de control doble "if – else"

"if else" es una estructura de control en la cual existe una condición que es evaluada en la línea de código del "if", si la misma se cumple, se ejecuta el bloque de código que está a continuación, caso contrario, se ejecuta el bloque de código correspondiente al "else":

```
if (condicion)
{
        // Bloque de código que se ejecuta si la condición se cumple

}
else
{
        // Bloque de código que se ejecuta si no se cumple la condición
}
// Siguiente línea a ejecutar
```

El uso de las llaves es obligatorio solo si a continuación del "if" o del "else" es necesario ejecutar más de una sentencia. Por más que se use una sola sentencia, se recomienda usar las llaves ya que a futuro se podrían agregar nuevas sentencias y un error común es olvidarse de agregar las llaves en dichas situaciones.
A continuación se muestra el diagrama de flujo de este tipo de estructuras:



Ejemplo:
Supongamos que obtenemos un número entero mediante la función scanf y queremos verificar si el mismo es mayor a cero:

```
if (numero>0)
{
        printf("El numero ingresado es mayor a cero\n");
}
else
{
        printf("El valor ingresado es menor o igual a cero\n");
}
```

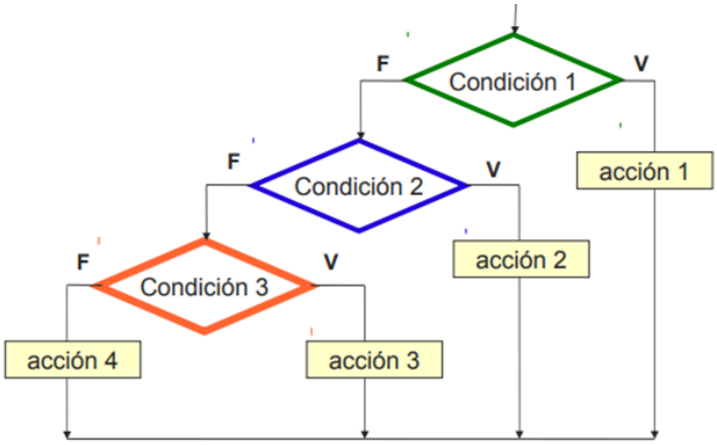
Estructura de control múltiple “if else if”

Este tipo de estructura consta de tener más de una estructura “if” relacionadas. Por ejemplo una estructura “if else” y a continuación un “if” relacionado al “else”.
El segundo “if” se ejecuta cuando no se cumple la condición del primero.
Ejemplo:

```
if (condiciónA)
    { Bloque de código que se ejecuta si se cumple la condición A}
else if (condiciónB)
    { Bloque de código que se ejecuta si se cumple la condición B }
else if (condiciónC)
    { Bloque de código que se ejecuta si se cumple la condición C }
else
    { Bloque de código que se ejecuta si no se cumple las condiciones anteriores }
```

(En este ejemplo, solo se ejecutará un bloque de sentencias)

A continuación se muestra el diagrama de flujo de este tipo de estructuras:



En los casos en los cuales solo se debe ejecutar una sentencia, no es necesario utilizar llaves:

```
main()
{
  int edad;
  printf("Introduce tu edad: ");
  scanf("%d",&edad);

 if (edad<0)
        printf("Lo siento, te has equivocado.");
  else if (edad<3) printf("Eres un bebé");
  else if (edad<13) printf("Eres un niño");
  else if (edad<18) printf("Eres un adolescente");
  else printf("Eres adulto");
}
```

Uso de más de un condicional:

Como hemos visto anteriormente, los condicionales evalúan una condición, si la misma es válida se ejecuta un bloque de código. Existen casos en los cuales es necesario evaluar más de una condición, por ejemplo evaluar si la condición A es verdadera y si también es verdadera la condición B.

Existe la posibilidad de evaluar ambas condiciones dentro del mismo "if", para esto se utiliza la lógica "and" y "or".

La lógica "and" requiere que ambas condiciones sean verdaderas para ejecutar el bloque de código del "if". Para aplicar esta lógica se utiliza el operador "**&&**".

La lógica "or" requiere que al menos una de las condiciones sea verdadera para ejecutar el bloque de código del "if". Para aplicar ésta lógica se utiliza el operador **"||**".

Nota importante:

Los operadores "and" y "or"
pueden utilizarse en las estructuras "if", "if else" y en "if else if".

En el siguiente ejemplo se muestra el uso de los operadores “and” y “or”:

```
#include <stdio.h>
int main()
{
   int numero;
   printf ("Ingrese un numero: ");
   scanf("%d", &numero);

   if (numero>0 && numero<100)
   {
      printf("El valor esta entre cero y 100");
   }
   else if (numero<=0 || numero>=100)
   {
      printf("El valor es menor o igual a cero o mayor o igual a 100");
   }
   return 0;
}
```

Operador Ternario

El operador ternario es un operador condicional que nos permite evaluar una condición y ejecutar el código correspondiente cuando se cumple la misma o bien ejecutar otro código cuando no se cumple.
Tiene la ventaja de poder escribir todo el código en una sola sentencia.
Es similar al “if else” pero nos permite resolver todo con una sola línea de código.

Forma de utilizarlo:

```
condición ? expresion1 : expresion2
```

En la línea anterior se evalúa la condición, si la misma es verdadera se ejecuta la expresión 1, si la condición es falsa se ejecutará la expresión 2.

Ejemplo:
En el siguiente ejemplo se evalúa la condición “nota>=4”, si se cumple dicha condición, se asigna el valor “1” a la variable aprobado, caso contrario se le asigna el valor cero:

CONDICIÓN T F

```
aprobado = nota>=4 ? 1 : 0;
```

La línea anterior hace exactamente lo mismo que el siguiente código:

```
if (nota >= 4)
   aprobado = 1;
else
   aprobado = 0;
```

Son 2 formas diferentes de resolver lo mismo, ambas son válidas. Cada uno elige la que más le conviene o bien aquella con la cual se siente más cómodo.

Ejemplo:
En el siguiente ejemplo se evalúa la condición "nota>0", si se cumple dicha condición, indica que es positivo, caso contrario que imprima que es negativo:

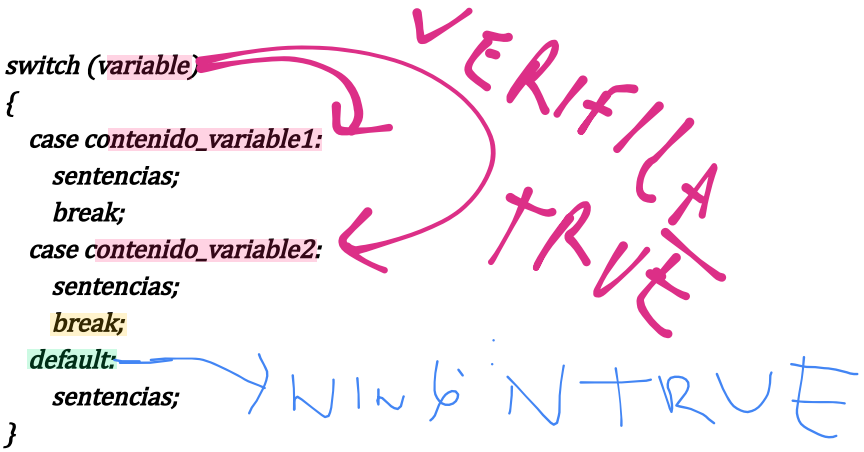
```
nota>0?  printf("es positivo") : printf("es negativo");
```

Estructuras Switch

En este tipo de estructuras se puede evaluar una variable y según el valor de la misma ejecutar las sentencias correspondientes.

Ejemplo:

```
switch (variable)
{
   case contenido_variable1:
      sentencias;
      break;
   case contenido_variable2:
      sentencias;
      break;
   default:
      sentencias;
}
```



Cada "case" permite evaluar si la variable es igual al valor correspondiente, en este ejemplo, contenido_variable1 o bien contenido_variable1.

Los "case" pueden incluir más de una sentencia sin la necesidad de utilizar llaves debido a que se ejecutarán todas las líneas de código hasta llegar a la sentencia "break".

La variable a evaluar solo puede ser del tipo entero o bien del tipo char.

En el caso de que el valor de la variable no coincida con ninguno de los declarados en los "case", se ejecutarán las sentencias de la sección "default".

Es importante recordar utilizar "break" al final de cada "case".
Ejemplo:

```
#include <stdio.h>

int main()
{
  /* Escribe el día de la semana */

  int dia;
  printf("Introduce el numero de dia de la semana: ");
  scanf("%d",&dia);
  switch(dia){
   case 1:
     printf("Lunes");
     break;
   case 2:
     printf("Martes");
     break;
   case 3:
     printf("Miercoles");
     break;
   case 4:
     printf("Jueves");
     break;
   case 5:
     printf("Viernes");
     break;
   case 6:
     printf("Sabado");
     break;
   case 7:
     printf("Domingo");
     break;
   default:
    printf("Error, numero invalido");
  }
  return 0;
}
```

Se imprimirá el día de la semana según el número de día ingresado.

Enumeradores en Switch:

En el ejemplo anterior, podemos notar que el código no es muy legible ya que para saber de qué día de la semana se trata, debemos relacionar el mismo con el número correspondiente. Sería más claro si en vez de usar “case 1” se usara “case Lunes”. Para esto en lenguaje C se tiene disponible la posibilidad de utilizar los **enumeradores**.

Los enumeradores son nombres simbólicos que representan números enteros constantes.

Para la declaración del enumerador es necesario utilizar la palabra reservada “**enum**”.

En la declaración se asigna un nombre al enumerador y a continuación entre las llaves se debe poner los nombres simbólicos que representan los números correspondientes.

Por defecto, el primer valor del **enumerador** representa al número **cero**, el segundo al 1 y así sucesivamente. Se puede modificar los valores por defecto asignando el valor deseado con el símbolo “**=**”. Si modificamos el valor de una, las siguientes constantes tendrán un valor sucesivo por defecto, es decir, que si cambiamos el valor de la primera constante en 1, la siguiente constante tendrá un valor de 2, la siguiente 3, etc.

Es posible declarar variables del tipo del **enum**.

Ejemplo:

```
enum colores {verde, rojo, amarillo};
```

En este caso, verde representa al número cero, rojo al 1 y amarillo al 2 ya que como se dijo anteriormente, por defecto se comienza desde cero.

Es posible asignar un valor numérico diferente al que se otorga por defecto:

```
enum colores {verde=3, rojo, amarillo};

//En este ejemplo, rojo tiene un valor de 4, y amarillo de 5.
```

Ejemplo:

```
#include <stdio.h>

int main()
{
      enum Dias {Lunes=1, Martes=2, Miercoles=3, Jueves=4, Viernes=5, Sabado=6,
Domingo=7};

// En vez de declarar dia como una variable del tipo int, lo declaré del tipo enum.

  enum Dias dia;

  printf("Introduce el numero de dia de la semana: ");
  scanf("%d",&dia);

  switch(dia){
   case Lunes:
     printf("Lunes");
     break;
   case Martes:
     printf("Martes");
     break;
   case Miercoles:
     printf("Miercoles");
     break;
   case Jueves:
     printf("Jueves");
     break;
   case Viernes:
     printf("Viernes");
     break;
   case Sabado:
     printf("Sabado");
     break;
   case Domingo:
     printf("Domingo");
     break;
   default:
    printf("Error, numero invalido");
  }
  return 0;
}
```