Programación I

Clase 1
Conceptos Básicos

Universidad Tecnológica Nacional - Cuch Sede Chivilcoy

Profesores

Luciana Denicio

Matias Garro

Concepto de informática

La Informática es la ciencia que estudia el análisis y resolución de problemas utilizando computadoras.

Ciencia *se relaciona con una metodología fundamentada y racional para el estudio y resolución de los problemas.*

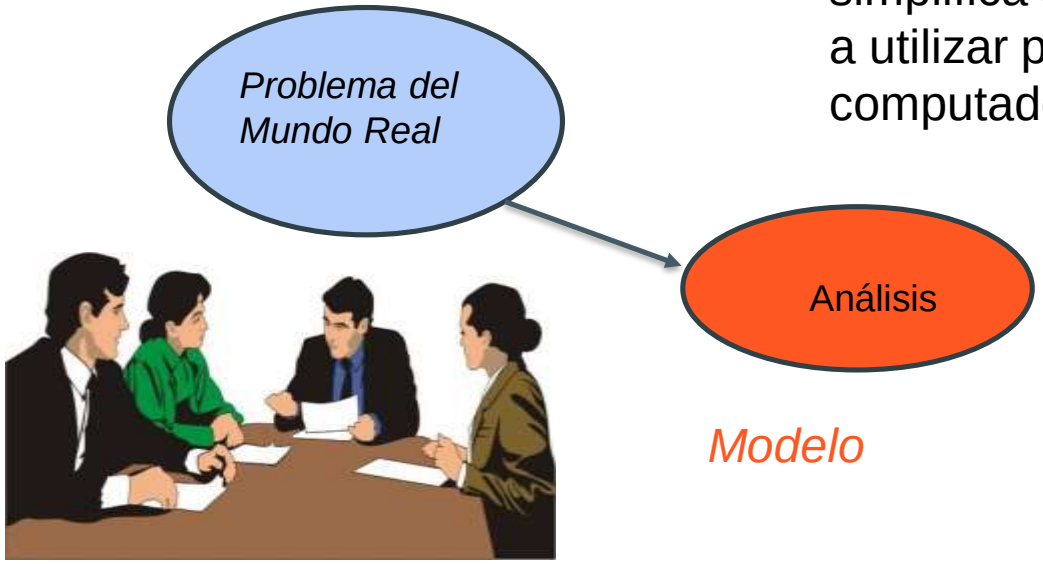
La ***resolución de problemas*** aplicaciones en áreas muy diferentes tales como biología, comercio, control industrial, educación, administración, robótica, arquitectura, etc.

Computadora máquina digital y sincrónica, con capacidad de cálculo numérico y lógico y de comunicación con el mundo exterior. Ayuda al hombre a realizar tareas repetitivas en menor tiempo y con mayor exactitud.

Etapas de resolución de un problema por computadora

Del Problema a la Solución

1er etapa: se sintetizan los requerimientos del problema y se simplifica el contexto y los datos a utilizar por el programa en la computadora.



Etapas de resolución de un problema por computadora

Del Problema a la Solución

Problema del Mundo Real

Analisis

Diseño

2da etapa:
La descomposición funcional nos ayudará a reducir la complejidad, a distribuir el trabajo y en el futuro a re-utilizar los módulos.
Algoritmos.

Algoritmos

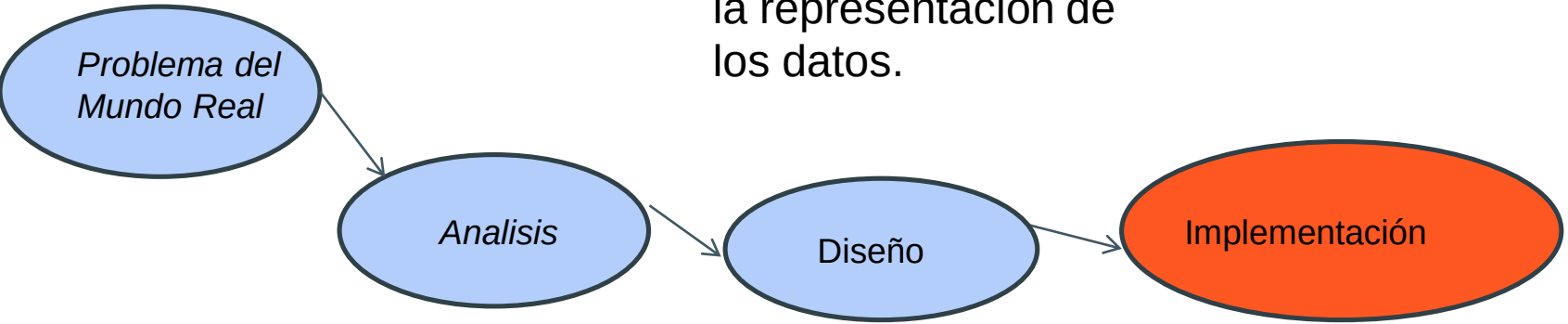
↓

Solución Modularizada

Etapas de resolución de un problema por computadora

Del Problema a la Solución

3er etapa: escribir algoritmos en un lenguaje de programación y elegir la representación de los datos.



Programa

Etapas de resolución de un problema por computadora

Del Problema a la Solución

4ta etapa: verificar que el programa conduce al resultado deseado, utilizando datos representativos del problema real

¿ Que Puede ocurrir?

Problema del Mundo Real → *Analisis* → Diseño → Implementación → Verificación

Etapa de Analisis del problema

- En una primera etapa, se analiza el problema en su contexto del mundo real. Deben obtenerse los **requerimientos** del usuario. El resultado del análisis del problema es un modelo preciso del ambiente del problema y del objetivo a resolver.

Etapa de Diseño de la solución

- Suponiendo que el problema es computable, a partir del modelo se debe diseñar una solución. En el paradigma procedural, esta etapa involucra entre otras tareas la **modularización del problema**, considerando la descomposición del mismo y los datos necesarios para cumplir su objetivo.

Esta etapa involucra la **especificación de los algoritmos**:

Etapa de Diseño de la solución

- Cada uno de los módulos del sistema diseñado tiene una función que podemos traducir en un algoritmo (que puede no ser único). La elección del algoritmo adecuado para la función del módulo es muy importante para la eficiencia posterior del sistema de software.

Etapa de Implementación de la solución

- Esta etapa involucra la **escritura de los programas**.

Los algoritmos definidos en la etapa de diseño se convierten en programas escritos en un lenguaje de programación concreto.

Etapa Verificación de la solución

- Una vez que se tienen los programas escritos y depurados de errores de sintaxis, se debe **verificar que su ejecución conduce al resultado deseado**, utilizando datos representativos del problema real.

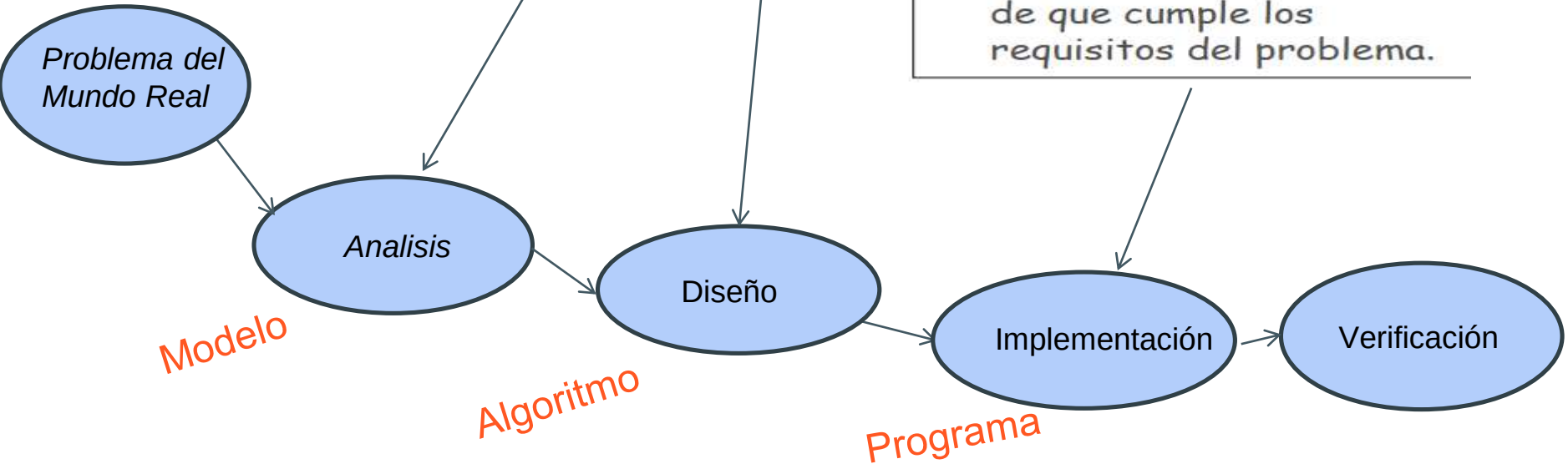
En resumen

Análisis y Diseño
(*NO depende del lenguaje*)

- Entender el problema
- Modelizarlo.
- Modularizar.
- Escribir los algoritmos.

Implementación
(*depende del lenguaje*)

- Codificación de los algoritmos en un lenguaje de programación.
- Prueba en ejecución de cada módulo.
- Prueba general del programa y verificación de que cumple los requisitos del problema.



Concepto de Algoritmo

Definiremos **algoritmo** como la especificación rigurosa de la secuencia de pasos (instrucciones) a realizar sobre un **autómata** para alcanzar un resultado deseado en un tiempo finito.

✓ **Especificación rigurosa** significa que debemos expresar un algoritmo en forma clara y unívoca.

✓ Si el autómata es una computadora, tendremos que escribir el algoritmo en un lenguaje “entendible” y ejecutable por la máquina.

✓ Alcanzar el resultado en **tiempo finito** significa que suponemos que un algoritmo **comienza y termina.** Por lo tanto, el número de instrucciones debe ser también finito. instrucciones debe ser también finito.

Programa

Concepto de Programa

Definiremos **programa** como el conjunto de instrucciones u órdenes **ejecutables** sobre una **computadora**, que permite cumplir con una función específica (dichas órdenes están expresadas en un lenguaje de programación concreto).

✓ Normalmente los programas alcanzan su función objetivo en un **tiempo finito**.

✓ Los **programas de aplicación** constituyen el verdadero valor que da utilidad a las computadoras (programas WEB, de administración, cálculo, comunicaciones, control industrial, sistemas expertos etc.)

✓ Los **programas** se escriben en un **lenguaje de programación** siguiendo un conjunto de reglas sintácticas y semánticas.

Escribir un programa exige:

✓ **Elegir la representación** adecuada de los datos del problema**.**

✓ **Elegir el lenguaje de programación** a utilizar, según el problema y la máquina a emplear.

✓**Definir el conjunto de instrucciones** (en el lenguaje elegido) cuya ejecución ordenada conduce a la solución.

Programa = Instrucciones + Datos

Programación Imperativa

El modelo que siguen los lenguajes de programación para **DEFINIR** y **OPERAR** la información, permite asociarlos a un paradigma de programación particular.

Vamos a trabajar con el paradigma imperativo/procedural.
El lenguaje de Programación que vamos a usar es C

Programa = Instrucciones + Datos

Las *instrucciones* (*acciones*) representan las operaciones que ejecutará la computadora al interpretar el programa.

Se escriben en un lenguaje de programación determinado

Lenguaje C

Los *datos* son los valores de información de los que se necesita disponer y en ocasiones transformar para ejecutar la función del programa.

Cada lenguaje de programación tiene los propios

Concepto de Dato

Un **Dato** es una representación de un objeto del mundo real. Los datos permiten modelizar los aspectos del problema que se quieren resolver mediante un programa ejecutable en una computadora.

Datos constantes
datos que no cambian durante la ejecución del programa.

Datos variables
datos que durante la ejecución del programa pueden cambiar.

En C cada dato debe tener asociado un tipo de dato

Definición de Tipo de Dato

Un *tipo de dato* es una clase de objetos de datos ligados a un conjunto de operaciones para crearlos y manipularlos.

Los tipos de datos se caracterizan por:

- ✓ Un rango de valores posibles
- ✓Un conjunto de operaciones realizables sobre ese tipo
- ✓ Una representación interna

¿Para que sirven?

¿Qué valores puedo usar?

¿Qué operaciones puedo aplicar?

¿Cuánta memoria utiliza?

Clasificación de Tipos de Datos

Los tipos de datos simples son aquellos que toman un único valor, en un momento determinado, de todos los permitidos para ese tipo.

Clasificación de Tipos de Datos Simples

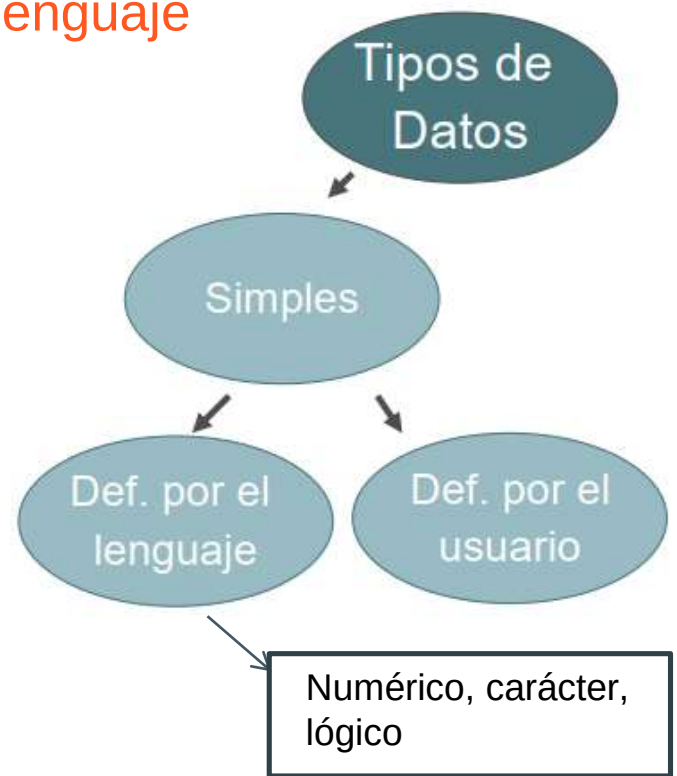
A su vez, los tipos simples, pueden clasificarse en:

Los tipos de datos definidos por el lenguaje (primitivos o estándar) son provistos por el lenguaje y tanto la representación como sus operaciones y valores son reservadas al mismo.

Los tipos definidos por el usuario, permiten definir nuevos tipos de datos a partir de los tipos simples.

Clasificación de Tipos de Datos Simples Definidos por el lenguaje

Comenzaremos presentando los tipos simples y definidos por el lenguaje



¿Valores posibles?

¿Operaciones?

¿Representación en memoria?

Estructuras de control

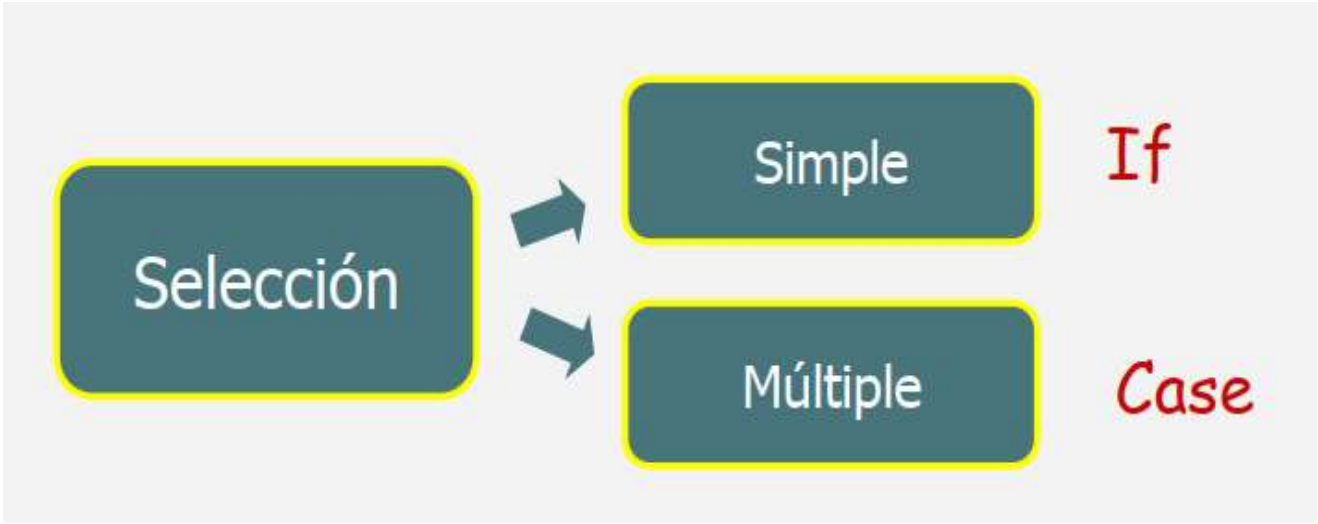
Todos los lenguajes de programación tienen un conjunto mínimo de instrucciones que permiten especificar el **control** del algoritmo que se quiere implementar.

Veremos que este conjunto **debe contener como mínimo:**

- ✓**Selección**
- ✓**Iteración**

¿Para qué nos sirven las estructuras de control?
Las estructuras de control permiten modificar el flujo de ejecución de las instrucciones de un programa.

Clasificación de las Estructuras de control

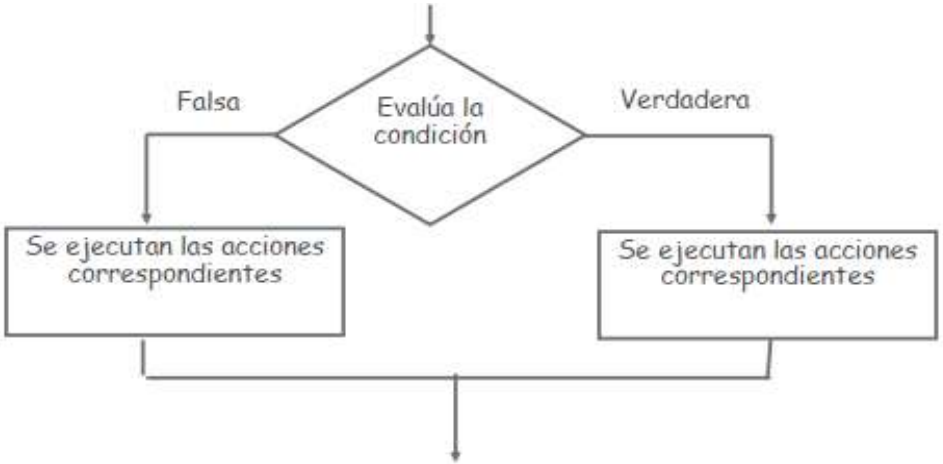



Iterativas
Precondicional
While ... do
Postcondicional
Repeat...until
Repetitivas
For

Estructura de control: selección simple

Puede ocurrir que en un problema real sea necesario elegir una alternativa entre 2 posibles.

La estructura de selección simple se representa simbólicamente:



Estructura de control: selección simple

Ejemplo sin else

```
#include <stdio.h>

int main() {
    int numero;

    printf("Ingresa un número: ");
    scanf("%d", &numero);

    // Estructura de control IF
    if (numero > 0) {
        printf("El número es positivo.\n");
    }

    return 0;
}
```

1-Se declara una variable numero de tipo entero.
2- Se le pide al usuario que ingrese un número.
3-La estructura if verifica si el número ingresado es mayor que cero.
4- Si la condición es verdadera (es decir, si el número es positivo), se imprime "El número es positivo."

Estructura de control: selección simple

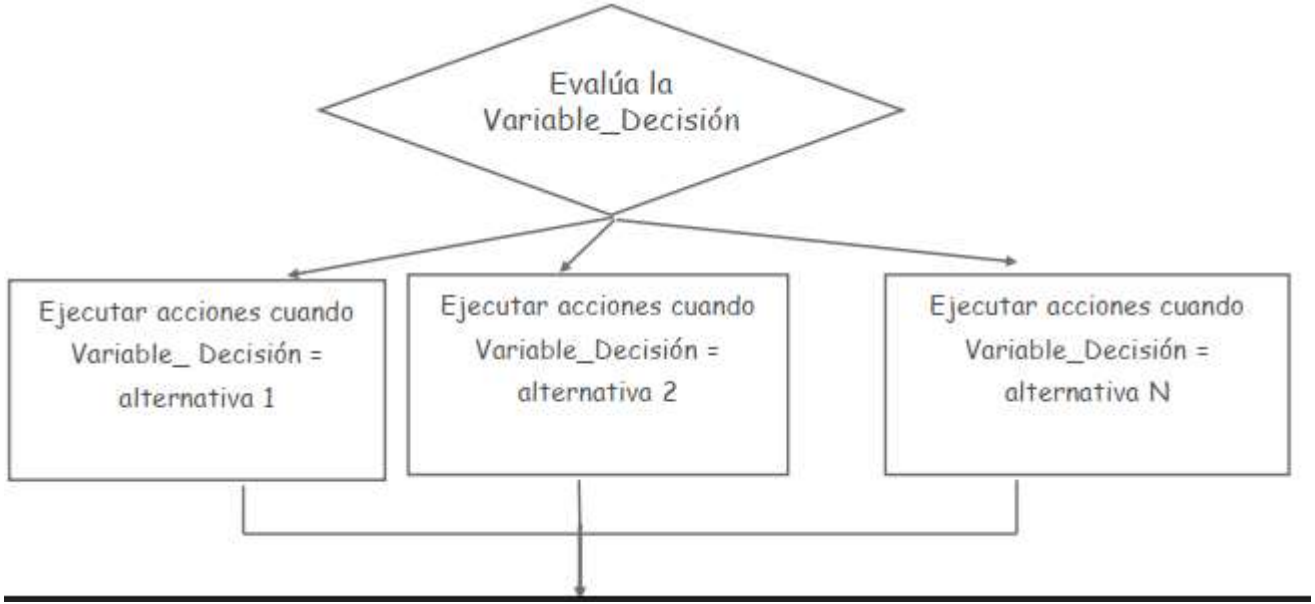
Podemos modificar la condición o añadir un bloque else si quieres realizar una acción diferente cuando la condición no se cumpla. Aquí te dejo un ejemplo con else:

```
#include <stdio.h>
int main() {
    int numero;

    printf("Ingresa un número: ");
    scanf("%d", &numero);

    // Estructura de control IF-ELSE
    if (numero > 0) {
        printf("El número es positivo.\n");
    } else {
        printf("El número no es positivo.\n");
    }

    return 0;
}
```

En este caso, si el número no es positivo, se ejecuta el bloque else y se imprime "El número no es positivo."

Estructura de Control de selección múltiple

Puede ocurrir que en un problema real sea necesario elegir una alternativa entre varias posibles en función del problema a resolver.
La estructura de selección múltiple se representa simbólicamente:



Estructura de Control de selección múltiple

Puede ocurrir que en un problema real sea necesario elegir una alternativa entre varias posibles en función del problema a resolver.

```
#include <stdio.h>
int main() {
    char letra = 'A';
    switch (letra) {
        case 'A':
            printf("Es la letra A\n");
            break;
        case 'B':
            printf("Es la letra B\n");
            break;
        case 'C':
            printf("Es la letra C\n");
            break;
        default:
            printf("Letra no reconocida\n");
    }
        return 0;
}
```

- En este caso, la variable letra tiene el valor 'A'.
-El switch compara el valor de letra con cada case.
-Como el valor de letra es 'A', se ejecuta el bloque correspondiente a case 'A' y se imprime "Es la letra A".
-Si se hubiera ingresado cualquier letra que no sea 'A', 'B' o 'C', se ejecutaría el bloque default que imprime "Letra no reconocida".

Estructuras de Control iterativas

❑ Puede ocurrir que se desee ejecutar un bloque de instrucciones desconociendo el número exacto de veces que se ejecutan.

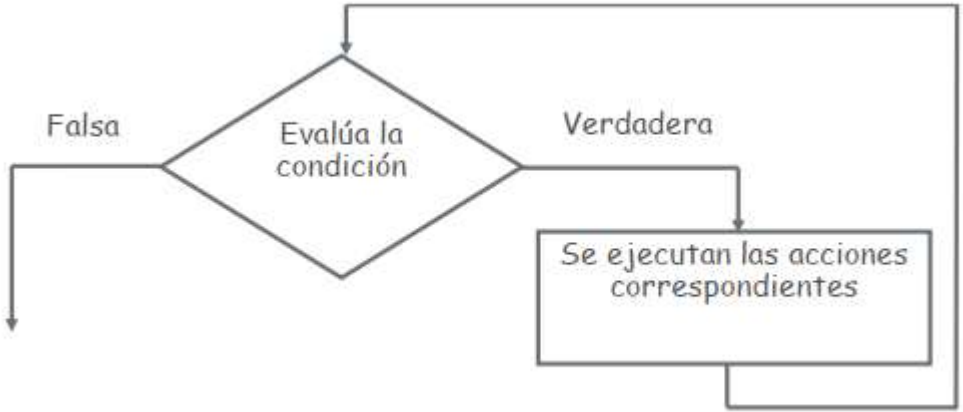
❑Para estos casos existen en la mayoría de los lenguajes de programación estructurada las **estructuras de control iterativas condicionales.**

❑Como su nombre lo indica las acciones se ejecutan dependiendo de la evaluación de la condición.

❑Estas estructuras se clasifican en **pre-condicionales** y **postcondicionales**.

❑Las estructuras de control iterativas precondicionales primero evalúan la condición y si es verdadera se ejecuta el bloque de acciones. Dicho bloque se puede ejecutar 0, 1 ó más veces.

❑ Importante: el valor inicial de la condición debe ser conocido o evaluable antes de la evaluación de la condición.



Estructura de Control iterativa precondicional

Estructura WHILE

```
#include <stdio.h>

int main() {
    int contador = 1;
while (contador <= 5) {
        printf("Contador: %d\n", contador);
        contador++;
    }

    return 0;
}
```

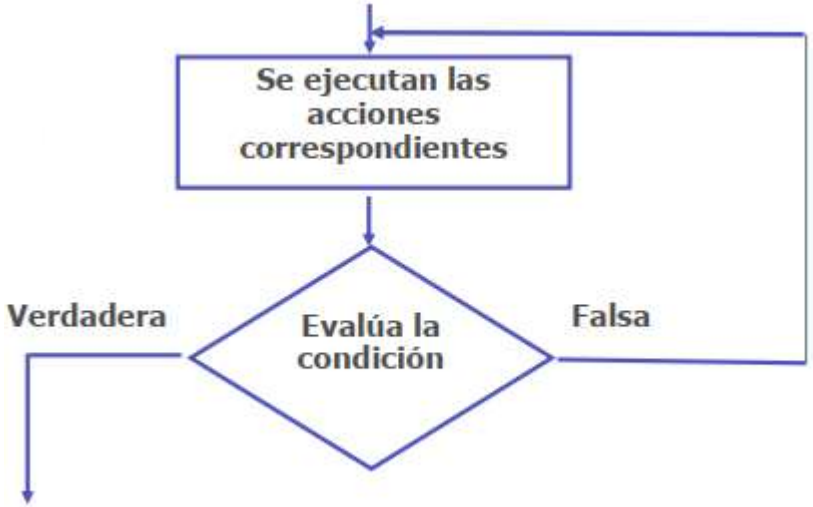
Imprime:
Contador: 1
Contador: 2
Contador: 3
Contador: 4
Contador: 5

Explicación:

- La variable contador empieza con el valor 1.
- La condición contador <= 5 se evalúa antes de cada iteración. Mientras sea verdadera, el ciclo while seguirá ejecutándose.
- En cada iteración, se imprime el valor actual de contador y luego se incrementa en 1.
- El ciclo se repite hasta que contador sea mayor que 5, momento en el cual el ciclo termina.

Estructura de Control iterativa postcondicional

❑Las **estructuras de control iterativas postcondicionales** primero ejecutan el bloque de acciones y luego evalúan la condición.
A diferencia de la estructura iterativa precondicional, el bloque de acciones se ejecuta 1 ó más veces.
Importante: el valor inicial de la condición debe ser conocido o evaluable antes de la evaluación de la condición.



Estructura de Control iterativa precondicional

Estructura DO (Repetir Hasta)

```
#include <stdio.h>

int main() {
    int contador = 1;
do {
        printf("Contador: %d\n", contador);
        contador++;
    } while (contador <= 5);

    return 0;
}
```

Imprime:
Contador: 1
Contador: 2
Contador: 3
Contador: 4
Contador: 5

Explicación

- La variable contador empieza con el valor 1.
- El bloque de código dentro del do se ejecuta al menos una vez.
- Después de cada ejecución, la condición contador <= 5 se evalúa. Si es verdadera, el ciclo se repite.
- Cuando el contador llega a 6, la condición se vuelve falsa y el ciclo termina.

Estructura de control repetitiva

- Puede ocurrir que se desee ejecutar un bloque de instrucciones conociendo el número exacto de veces que se ejecutan. Es decir repetir N veces un bloque de acciones.
- Este número de veces que se deben ejecutar las acciones es fijo y conocido de antemano.

Se muestra el diagrama esquemático de la repetición

```c
#include <stdio.h>

int main() {
    // Estructura for para imprimir
números del 1 al 5
    for (int i = 1; i <= 5; i++) {
        printf("Número: %d\n", i);
    }

    return 0;
}
```

El ciclo imprimirá los números del 1 al 5.

Explicación:

- La estructura for tiene tres partes:

1-Inicialización: int i = 1; — se establece el valor inicial de la variable i.

2-Condición: i <= 5; — el ciclo se repite mientras esta condición sea verdadera. En este caso, mientras i sea menor o igual a 5.

3- Incremento: i++ — en cada iteración, i se incrementa en 1.