Practica 8

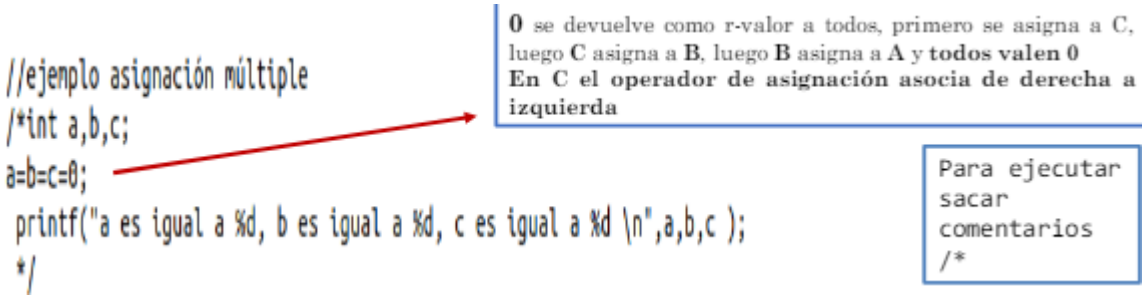
1.Diferencias

Sentencia Simple	Sentencia compuesta
Flujo de control mas simple	
Ejecución de una sentencia a continuación de otra	Se pueden agrupar varias sentencias en una con el uso de delimitadores (begin end)
Delimitador general= ;	Begin, end, { }

2.**Asignación en C**

Define a la sentencia de asignación como una expresión con efectos colaterales

A=B



3.Sí, una expresión de asignación puede producir efectos secundarios que afecten al resultado final, dependiendo de cómo se evalúe. Esto se debe a que el orden en que se evalúan las expresiones no siempre está definido por el lenguaje de programación, lo que puede dar lugar a diferentes resultados en función del orden en que se ejecutan las operaciones.

C:

```
int x = 10;
int y = x + 1;
x = y;
```

En este ejemplo, el valor final de `x` depende del orden en que se evalúen las expresiones `x + 1` y `x = y`. Si `x + 1` se evalúa primero, entonces `x` se establecerá en 11 antes de que se le asigne el valor de `y`, que también es 11.

Sin embargo, si `x = y` se evalúa primero, entonces `x` se establecerá en 10 antes de que se le asigne el valor de `y`, que es 11. En este caso, el valor final de `x` será 10.

```
int a=5;
int b= ++a;
```

En este ejemplo, la expresión ++a incrementa el valor de a en 1 antes de asignarlo a b.
El efecto lateral es que el valor final de b será 6, no 5

4.

Circuito corto:

Un lenguaje usa circuito corto para la evaluación de una expresión cuando dicha evaluación se detiene al determinar el resultado final sin tener que evaluar el resto de la expresión

- La conjunción (y o and) da verdadero solo cuando ambos son verdaderos, si el primer término es falso no necesita evaluar el segundo⇒ resultado falso
- La disyunción (o u or) da falso solo cuando ambos términos son falsos, si el primer término es verdadero no necesita evaluar el segundo⇒ resultado verdadero

Circuito largo:

Un lenguaje usa circuito largo para la evaluación de una expresión cuando dicha evaluación sigue evaluando todas las partes de la expresión aunque ya se haya determinado el resultado final.

Ejemplo donde da error por un circuito y por otro no

si hay un IF(A and B) y B es nulo..

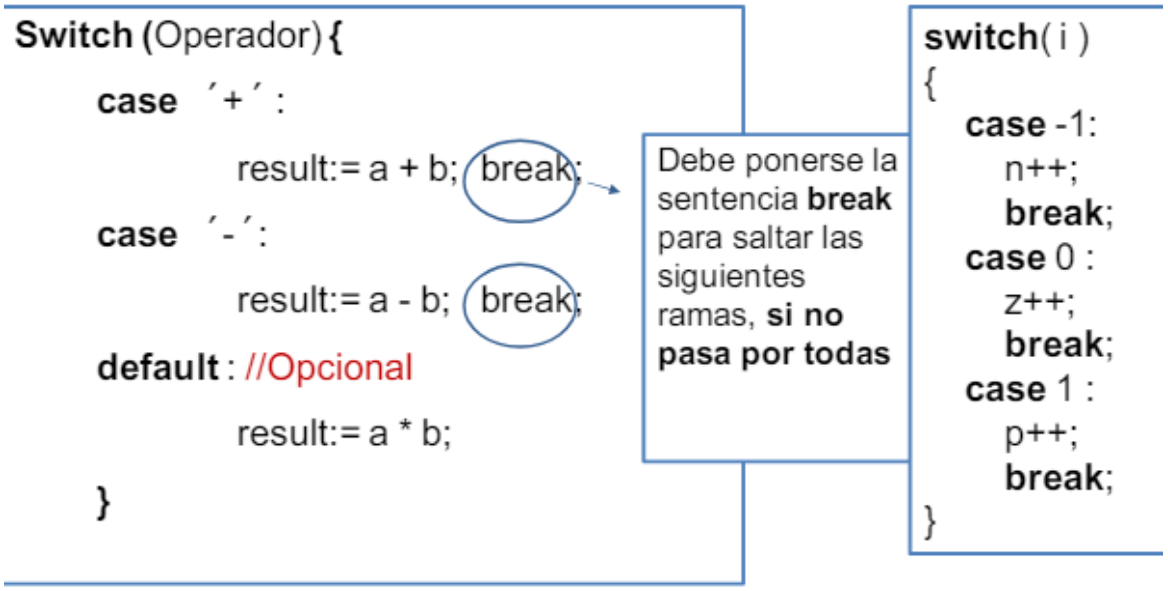
Si A es falso, por circuito corto termina la evaluación y no se llega a verificar B por lo que se evita el error. Por circuito largo, se evaluaría B también y generaría error.

5.Reglas para asociar el else con el if correspondiente

- Delphi, ADA, C: coincidencia de las llaves o palabras claves correspondientes al if más cercano sin un else.
  - en C con las llaves
  - En Ada y Delphi con begin y end.
  - Si se utilizan llaves o palabras clave adicionales, la asociación del else se determina por la coincidencia de esas delimitaciones adicionales.
- Python: usa la indentación, el else se asocia con el if anterior que tenga la misma indentación

6.Construcción para expresar múltiples selección

c



🔹 Constructor Switch seguido de (expresión)
🔹 Cada rama Case es "etiquetada" por uno o más valores constantes (enteros o char)
🔹 Si coincide con una etiqueta del Switch se ejecutan las sentencias asociadas, y se continúa con las sentencias de las otras entradas. (chequea todas salvo exista un break)
🔹 Existe la sentencia break, que provoca la salida de cada rama (sino continúa)
🔹 Existe opción default que sirve para los casos

que el valor no coincida con ninguna de las opciones establecidas, es opcional
▪ El orden en que aparecen las ramas no tiene importancia

Pascal

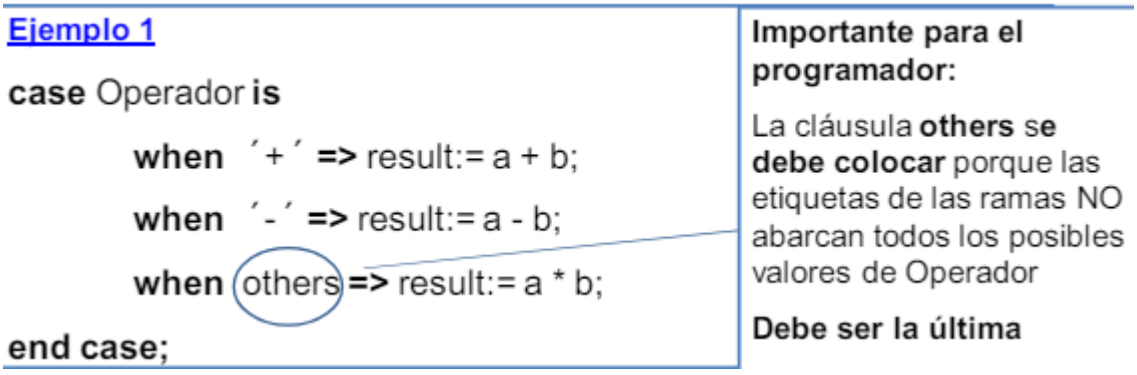
Ejemplo:

```
var opcion : char;
begin
 readln(opcion);
case opcion of
     '1' : nuevaEntrada;
     '2' : cambiarDatos;
     '3' : borrarEntrada
     else
      writeln('Opcion no valida!!')
     end;
end
```

Usa palabra reservada case seguida de variable de tipo ordinal y la palabra reservada of.
▪ La variable-expresión a evaluar es llamada “selector"
▪ Lista las sentencias de acuerdo con diferentes valores que puede adoptar la variable (los “casos”). Llevan etiquetas.
▪ No importan el orden en que aparecen
▪ bloque else para el caso que la variable adopte un valor que no coincida con ninguna de las sentencias de la lista. (opcional)
▪ Para finalizar se coloca un “end;” (no

se corresponde con ningún “begin” que exista).
▪ Es inseguro porque no establece qué sucede cuando un valor no cae dentro delas alternativas puestas

ADA



Las expresiones pueden ser solamente de tipo entero o enumerativas
En las selecciones del case se deben estipular "todos" los valores posibles que puede tomar la expresión
El when se acompaña con ⇒ para indicar la acción a ejecutar si se cumple la condición.
Tiene la cláusula Others que se puede utilizar para representar a aquellos valores que no se especificaron explícitamente
Others "debe" ser la última opción antes del end;
Después que una rama es ejecutada el Case entero finaliza. (no pasa por otras ramas)

Python

En Python, no hay una construcción específica para expresar múltiples selecciones como el switch-case en C. En su lugar, se utilizan estructuras if-

elif-else para lograr un comportamiento
similar. Cada expresión en el if-elif-else se evalúa secuencialmente y se ejecuta el bloque de código correspondiente a la primera expresión que sea verdadera. A partir de Python 3.10, se introdujo una nueva característica llamada "match-case" que proporciona una sintaxis más cercana al switch-case de C. Sin embargo, aún existen diferencias en comparación con el switch-case de C en términos de funcionalidad y flexibilidad.

7.En pascal estándar da error porque en este "no permite" que se modifiquen los valores del límite inferior, límite superior, ni del **valor de la variable de control. En este caso se modifica i en el procedimiento A.**

En ADA..

- **Versiones antiguas de ADA:** Es probable que generen errores de compilación debido a la modificación de una variable global dentro de un procedimiento.
- **Versiones modernas de ADA:** Podrían permitir la compilación del código, pero podrían generar resultados inesperados debido al comportamiento no determinista de la modificación de una variable global dentro de un ciclo

8. En ADA

```
with Ada.Text_IO;                              use Ada.Text_IO;

procedure Ejemplo_Case is
   type Puntos is Integer;
   Puntos_Variable : Puntos := 5;  -- Valor de ejemplo

begin
   Select Puntos_Variable
      when 1..5 => Put_Line ("No puede continuar");
      when 10 => Put_Line ("Trabajo terminado");
      when others => Put_Line ("Valor no válido");
   end Select;
end Ejemplo_Case;
```

```
#include <stdio.h>

int main() {
  int puntos = 5;  // Valor de ejemplo

  switch (puntos) {
    case 1 ... 5:
      printf("No puede continuar\n");
      break;
    case 10:
      printf("Trabajo terminado\n");
      break;
    default:
      printf("Valor no válido\n");
  }

  return 0;
}
```

9.

Diferencias entre `yield` y `return` en Python:

1. Comportamiento:

- `return`: Termina la ejecución de la función y devuelve un valor.
- `yield`: Suspende la ejecución de la función y devuelve un valor. **La función no termina y puede continuar más tarde desde el mismo punto.**

2. Uso:

- `return`: Se utiliza para devolver un único valor al final de la ejecución de la función.
- `yield`: Se utiliza para devolver una secuencia de valores uno a uno, **permitiendo iterar sobre la secuencia**.

3. Analogía:

- `return`: Es como entregar un paquete a la vez.

- `yield`: Es como entregar una serie de paquetes uno a uno, **permitiendo al receptor procesar cada paquete antes de recibir el siguiente**.

4. Ejemplo:

Función que genera números pares:

Python

```
def generador_pares(limite):
  for numero in range(limite):
    if numero % 2 == 0:
      yield numero  # Devuelve un valor usando yield
```

Uso del generador:

```
generador = generador_pares(10)  # Crea un generador

# Itera sobre el generador y muestra los valores (números par
for numero in generador:
  print(numero)
```

10. Map en JavaScript

La instrucción `map` en JavaScript es una **función de orden superior** que permite **transformar cada elemento de un array en otro valor**. Recibe dos argumentos:

1. **Función de transformación:** Una función que se aplica a cada elemento del array.
2. **Array original:** El array sobre el que se aplica la transformación.

Retorno:

Devuelve un **nuevo array** que contiene los resultados de aplicar la función de transformación a cada elemento del array original.

Ejemplo:

JavaScript

```
const numeros = [1, 2, 3, 4, 5];

const cuadrados = numeros.map(function(numero) {
  return numero * numero;
});

console.log(cuadrados); // [1, 4, 9, 16, 25]
```

Alternativas a `map`:

Existen otras formas de transformar arrays en JavaScript, aunque `map` es la más común y concisa:

- **Bucle `for`:** Se puede recorrer el array original y aplicar la transformación a cada elemento dentro del bucle.
- `forEach`: Similar a `map`, pero no devuelve un nuevo array, solo ejecuta la función de transformación para cada elemento.
- `reduce`: Permite reducir un array a un único valor, aplicando una operación acumulada a cada elemento.
- **Librerías de terceros:** Existen librerías como `lodash` que ofrecen funciones más avanzadas para trabajar con arrays.

11. Espacio de nombres: contenedor lógico usado para agrupar elementos como variables, funciones, clases, etc dentro de un programa, con el objetivo de evitar colisiones de nombres y dar un contexto separado para elementos con el mismo nombre. Permite que muchos elementos con el mismo nombre coexistan en un programa sin problemas porque cada elemento esta en un espacio de nombres únicos

JavaScript no implementa instrucciones nativas para el manejo de espacios de nombres de la
misma manera que otros lenguajes como PHP o Python. Sin embargo, se pueden simular
espacios de nombres en JavaScript utilizando objetos y módulos.
En JavaScript, los objetos actúan como contenedores y se pueden utilizar para agrupar
funciones, variables y otros elementos relacionados. Esto permite crear una estructura similar

a un espacio de nombres para organizar el código.
Por ejemplo, se puede utilizar un objeto para agrupar funciones relacionadas:

```
var miEspacioDeNombres = {
funcion1: function() {
},
funcion2: function() {
// código de la función 2
},
variable1: "Hola",
variable2: 42
};
```

En este ejemplo, miEspacioDeNombres actúa como un espacio de nombres y contiene las funciones funcion1 y funcion2, así como las variables variable1 y variable2. Para acceder a estos elementos, se utiliza la notación de punto:
miEspacioDeNombres.funcion1();
console.log(miEspacioDeNombres.variable1);

En lenguajes como PHP y Python, los espacios de nombres tienen características más avanzadas, como la capacidad de importar y exportar elementos entre espacios de nombres, y la posibilidad de tener múltiples niveles de anidamiento. Esto facilita la organización y el control más sofisticado de los elementos dentro de un programa.
En PHP, por ejemplo, se puede definir y utilizar espacios de nombres utilizando la palabra clave namespace y se pueden importar elementos utilizando use. Además, los espacios de nombres pueden estar organizados en una estructura de directorios que refleja la estructura de los
archivos fuente.
En Python, los módulos actúan como espacios de nombres, y se pueden importar utilizando la declaración import. Python también permite la creación de paquetes, que son directorios que
contienen módulos relacionados y proporcionan una forma de organizar y agrupar elementos.
A su vez, en un programa de Python hay tres tipos de espacios de nombres:
• Integrado: contiene los nombres de todos los objetos integrados de Python.
• Global: contiene cualquier nombre definido en el nivel del programa principal.
• Local: espacio de nombres local para la función y permanece hasta que la función
finaliza.