Práctica 9

Ejercicio 1: Una sentencia puede ser simple o compuesta, ¿Cuál es la diferencia?

Simple:

- Es el flujo de control más simple.
- Ejecución de una sentencia a continuación de otra.
- El delimitador más general y más usado es el ";"
    - Sentencia 1;
    - …
    - Sentencia n;
- Hay lenguajes que NO tienen delimitador
- Establecen que por cada línea haya sólo 1 instrucción. Se los llaman orientados a línea.
- Ej: Fortran, Basic, Ruby, Phyton

Compuesta:

- Se pueden agrupar varias sentencias en una con el uso delimitadores:
    - Begin y End. En Ada, Pascal
    - { } en C, C++, Java, etc.

Ejercicio 2: Analice como C implementa la asignación.

Asignación en C:

- A = B
- Define a la sentencia de asignación como una expresión con efectos colaterales.
- C evalúa de derecha a izquierda.
    - Ejemplo a=b=c=0; El 0 se devuelve como r-valor a todos, primero se asigna a C, luego C asigna a B, luego B asigna a A y todos valen 0
- Requiere del lado izquierdo de la asignación un l-valor

Ejercicio 3: ¿Una expresión de asignación puede producir efectos laterales que afecten al resultado final, dependiendo de cómo se evalúe? De ejemplos.

Sí, una expresión de asignación puede producir efectos laterales que afecten al resultado final de un programa, dependiendo de cómo se evalúe. Los efectos laterales ocurren cuando una expresión tiene impacto en algo más que simplemente calcular un valor y asignarlo a una variable. Aquí hay algunos ejemplos de cómo los efectos laterales pueden influir en el resultado final:

1. Expresiones con incremento o decremento:

```
1 int a = 5;
2 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.

2. Expresiones con operaciones de asignación compuestas:

```
1 int c = 10;
2 c += 5;
```

En este caso, la expresión c += 5 incrementa el valor de c en 5 y lo asigna de nuevo a c. El efecto lateral es que el valor final de c será 15.

3. Asignación en condición:

```
1 if (i = 30) {...}
```

Si se usa el operador de asignación (=) en la condición del if, la expresión se evaluará como verdadera siempre que el valor asignado no sea cero.
El efecto lateral en este caso es que el valor de i se modifica a 30 como resultado de la asignación. Además la condición del if siempre se evaluará como verdadera.

Ejercicio 4: Qué significa que un lenguaje utilice circuito corto o circuito largo para la evaluación de una expresión. De un ejemplo en el cual por un circuito de error y por el otro no.

Circuitos cortos (short-circuit):

En un lenguaje que utiliza el circuito corto, la evaluación de una expresión lógica se detiene tan pronto como se determina el resultado final sin necesidad de evaluar el resto de la expresión.

- Operador "y" (&&): Si la primera parte de la expresión es falsa, el resultado final será falso sin importar el valor de la segunda parte. Por lo tanto, en un circuito corto, si la primera parte es falsa, la segunda parte no se evalúa.
- Operador "o" (||): Si la primera parte de la expresión es verdadera, el resultado final será verdadero sin importar el valor de la segunda parte. Por lo tanto, en un circuito corto, si la primera parte es verdadera, la segunda parte no se evalúa.

Ejemplo de circuito corto:

```
1 int a = 10;
2 int b = 0;
3
4 if (b != 0 && a/b > 5) {
5     // Código que se ejecuta si ambas partes son verdaderas
6 }
```

En este ejemplo, debido al circuito corto, la evaluación de a/b > 5 no se realiza si b es igual a 0. Esto evita una posible división por cero y evita un error en tiempo de ejecución.

Circuitos largos (non-short-circuit):

En un lenguaje que utiliza el circuito largo, la evaluación de una expresión lógica continúa evaluando todas las partes de la expresión, incluso si el resultado final ya se ha determinado.

- Operador "y" (&&): En un circuito largo, ambas partes de la expresión se evalúan independientemente del valor de la primera parte. Esto asegura que se realicen todas las comprobaciones necesarias.
- Operador "o" (||): En un circuito largo, ambas partes de la expresión se evalúan independientemente del valor de la primera parte. Esto asegura que se realicen todas las comprobaciones necesarias.

Ejemplo de circuito largo:

```
1 int a = 10;
2 int b = 0;
3
4 if (b != 0 & a/b > 5) {
5     // Código que se ejecuta si ambas partes son verdaderas
6 }
```

En este ejemplo, debido al circuito largo, la evaluación de a/b > 5 se realiza incluso si b es igual a 0. Esto puede conducir a una división por cero y a un error en tiempo de ejecución si no se toman precauciones adicionales.

Ejercicio 5: ¿Qué regla define Delphi, Ada y C para la asociación del else con el if correspondiente? ¿Cómo lo maneja Python?

En lenguajes como Delphi, ADA y C, la regla de asociación del else se basa en la coincidencia de las llaves o palabras clave correspondientes al if más cercano sin un else.

En el caso de C, se utiliza la coincidencia de las llaves para determinar la asociación del else con el if correspondiente más cercano. El else se asocia con el if que tiene la misma llave de apertura ({}) más cercana y que no tiene un else asociado.

En el caso de Ada y Delphi, se utiliza la coincidencia de las palabras clave begin y end para determinar la asociación del else con el if correspondiente más cercano. El else se asocia con el if que tiene la misma palabra clave begin y end más cercana y que no tiene un else asociado.

En ambos casos, la asociación del else se basa en la estructura de anidación de los bloques de código y la coincidencia de las llaves o palabras clave correspondientes.

Es importante tener en cuenta que esta regla se aplica cuando no se utilizan llaves o palabras clave adicionales para delimitar explícitamente los bloques de código. Si se utilizan llaves o palabras clave adicionales, la asociación del else se determina por la coincidencia de esas delimitaciones adicionales.

En cuanto a Python, utiliza una indentación especial para delimitar bloques de código en lugar de llaves o palabras clave como begin y end. En Python, la asociación del else con el if correspondiente se basa en la indentación. El else se asocia con el if anterior que tenga la misma indentación.

Ejercicio 6: ¿Cuál es la construcción para expresar múltiples selección que implementa C? ¿Trabaja de la misma manera que la de Pascal, ADA o Python?

En C se utiliza el 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. (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

```
1  switch (expresion) {
2      case valor1:
3          // código a ejecutar si la expresion es igual a valor1
4          break;
5      case valor2:
6          // código a ejecutar si la expresion es igual a valor2
7          break;
8      case valor3:
9          // código a ejecutar si la expresion es igual a valor3
10         break;
11     default:
12         // si la expresion no coincide con ninguno de los casos anteriores
13         break;
14 }
```

Pascal y ADA tienen una construcción switch-case para expresar múltiples selecciones similar a C, Python utiliza estructuras if-elif-else y, a partir de la versión 3.10, ofrece la opción de usar match-case, aunque con diferencias significativas respecto al switch-case de C.

Pascal:

- La expresión a evaluar es llamada “selector"
- Lleva palabra case seguida de variable de tipo ordinal y la palabra reservada of.
- Luego lista de sentencias de acuerdo a 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).
- Es inseguro cuando no establece qué sucede cuando un valor no cae dentro de las alternativas puestas.
- Para finalizar la estructura del case se coloca un “end;” (no se corresponde con ningún “begin”).
- Si una expresión en el case coincide con el valor evaluado, se ejecutará el bloque de código correspondiente y luego se saldrá del switch-case. El control no pasará automáticamente al siguiente case, a menos que se especifique explícitamente mediante la instrucción "fallthrough" (que no es parte de Pascal estándar).

ADA:

- Constructor Case expresión is, con when y end case
- 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
- No pasa la compilación si:
    - NO se coloca la rama para un posible valor
    - NO aparece la opción Others en esos casos
- Si una expresión en el case coincide con el valor evaluado, se ejecutará el bloque de código correspondiente y luego se saldrá del switch-case. Se puede utilizar la instrucción "fallthrough" para indicar que se desea ejecutar el siguiente case sin importar si hay una coincidencia en el valor evaluado. Sin embargo, esta instrucción no es requerida por defecto como en C, y su uso debe ser explícito.

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.

Ejercicio 7: Sea el siguiente código:

```
.....
var i, z:integer;
Procedure A;
begin
  i:= i +1;
end;
begin
z:=5;
```

```
for i:=1..5 do
 begin
  z:=z*5;
  A;
  z:=z + i;
 end;
end;
```

a- Analice en las versiones estándar de ADA y Pascal, si este código puede llegar a traer problemas. Justifique la respuesta.

   En ADA y Pascal estándar no se puede modificar manualmente los valores de las variables de los iteradores. Por lo que habría error.

b- Comente qué sucedería con las versiones de Pascal y ADA, que Ud. Utilizó

En ADA y Pascal no estándar se pueden modificar los valores sin que salte un error pero sin embargo el código tendría muchos problemas.

Ejercicio 8 - Sea el siguiente código en Pascal:

var puntos: integer;

begin

...

case puntos

1..5: write(“No puede continuar”);

10:write(“Trabajo terminado”)

end;

..

Analice, si esto mismo, con la sintaxis correspondiente, puede trasladarse así a los lenguajes ADA, C. ¿Provocaría error en algún caso? Diga cómo debería hacerse en cada lenguaje y explique el por qué. Codifíquelo.

Como el bloque else es opcional en Pascal y en este caso no esta, podría haber un error si el valor de puntos no coincide con ninguna de las sentencias en la lista.

ADA:

```
1  procedure Example is
 2    puntos: integer;
 3 begin
 4    -- ...
 5    case puntos is
 6       when 1 .. 5 =>
 7          put_line("No puede continuar");
 8       when 10 =>
 9          put_line("Trabajo terminado");
10       when others => // No hago nada
11    end case;
12    -- ...
13 end Example;
```

La clausula others debe estar porque si no no pasaría la compilación.

C:

```
1  #include <stdio.h>
2
3  int main() {
4     int puntos;
5     // ...
```

```
6     switch (puntos) {
7        case 1:
8        case 2:
9        case 3:
10       case 4:
11       case 5:
12          printf("No puede continuar");
13          break;
14       case 10:
15          printf("Trabajo terminado");
16          break;
17    }
18    // ...
19    return 0;
20 }
```

No se puede especificar directamente un rango en el case. En su lugar, se deben listar explícitamente los valores que corresponden al rango deseado. Esto significa que en C se deben enumerar los casos para cada número del 1 al 5 en lugar de especificar el rango 1..5 directamente. A vez como en C luego de ejecutarse una sentencia asociada a un case se continua por las siguientes ramas, habría que agregar un break para que esto no se haga (en Pascal se hace por si defecto entro a una rama no entro al resto)

Ejercicio 9: Qué diferencia existe entre el generador YIELD de Python y el return de una función. De un ejemplo donde sería útil utilizarlo

La diferencia principal entre el generador yield de Python y el return de una función es que yield permite generar una secuencia de valores en lugar de retornar un solo valor como lo hace return.

Cuando se utiliza yield en una función, esta se convierte en un generador. El generador no devuelve un resultado inmediatamente como lo haría una función con return, sino que genera un valor cada vez que se le solicita, manteniendo su estado y la capacidad de continuar desde donde se detuvo.

Un ejemplo útil de uso de yield sería una función que genere una secuencia infinita de números pares. En lugar de calcular todos los números pares hasta un límite y almacenarlos en una lista, podemos utilizar un generador con yield para obtener los números pares uno a uno, evitando así almacenar todos los números en memoria.

Ejercicio 10: Describa brevemente la instrucción map en Javascript y sus alternativas.

La función map en JavaScript es una función de orden superior que se utiliza para transformar los elementos de un arreglo original y generar un nuevo arreglo con los resultados de aplicar una función a cada elemento del arreglo original.

```
1 const nuevoArreglo = arregloOriginal
2 .map(funcion(elemento, indice, arreglo) {
3   // Código de transformación
4   return nuevoElemento;
5 });
```

Algunas alternativas a map en JavaScript incluyen:

- forEach: La función forEach también recorre cada elemento del arreglo original, pero no crea un nuevo arreglo con los resultados de la transformación. En su lugar, se utiliza para realizar operaciones o acciones en cada elemento del arreglo sin modificar el arreglo original ni generar un nuevo arreglo.
- filter: La función filter se utiliza para crear un nuevo arreglo que contenga solo los elementos que cumplan una determinada condición. En lugar de transformar cada elemento como lo hace map, filter decide si un elemento se incluye en el nuevo arreglo basado en una condición booleana.
- reduce: La función reduce se utiliza para combinar todos los elementos de un arreglo en un solo valor. A diferencia de map, que genera un nuevo arreglo con la misma longitud que el original, reduce puede generar un resultado de cualquier tipo, como un número, una cadena o un objeto.

Ejercicio 11: Determine si el lenguaje que utiliza frecuentemente implementa instrucciones para el manejo de espacio de nombres. Mencione brevemente qué significa este concepto y enuncie la forma en que su lenguaje lo implementa. Enuncie las características más importantes de este concepto en lenguajes como PHP o Python.

Un espacio de nombres es un contenedor lógico que se utiliza para agrupar y organizar elementos, como variables, funciones, clases u otros identificadores, dentro de un programa. Su propósito principal es evitar colisiones de nombres y proporcionar un contexto separado para los elementos con el mismo nombre.

Un espacio de nombres permite que múltiples elementos con el mismo nombre coexistan en un programa sin conflicto, ya que cada elemento se encuentra dentro de un espacio de nombres único.

Al utilizar espacios de nombres, se puede acceder a los elementos dentro de ellos mediante una notación de calificación, utilizando el nombre del espacio de nombres seguido de un separador (.) y luego el nombre del elemento específico. Esto ayuda a evitar ambigüedades y asegura que se esté haciendo referencia al elemento correcto.

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:

```
1  var miEspacioDeNombres = {
 2   funcion1: function() {
```

```
    // código de la función 1
  },
  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.