

Práctica 4

Ejercicio 1: a) Tome una de las variables de la línea 3 del siguiente código e indique y defina cuáles son sus atributos:

```
1. Procedure Practica4();  
2. var  
3. a,i:integer  
4. p:puntero  
5. Begin  
6. a:=0;  
7. new(p);  
8. p:= ^i  
9. for i:=1 to 9 do  
10.a:=a+i;  
11.end;  
12....  
13.p:= ^a;  
14....  
15.dispose(p);  
16.end;
```

Atributos de la variable a.

Nombre: a

Alcance: Estático, de la línea 3 hasta línea 16

Tipo: Predefinido Integer, ligadura estática.

L-Valor: Alocacion dinámica automática.

R-Valor: Dinámico 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

b) Compare los atributos de la variable del punto a) con los atributos de la variable de la línea 4. ¿Qué dato contiene esta variable?

Atributos de la variable p.

Nombre: p

Alcance: Estático, de la línea 4 hasta línea 16

Tipo: Tipos definidos por el usuario Puntero, ligadura estática.

L-Valor: Alocación dinámica explícita.

R-Valor: Dinámico l-valor de i (línea 8), l-valor de a (línea 13).

Ejercicio 2:

a. Indique cuales son las diferentes formas de inicializar una variable en el momento de la declaración de la misma.

- Ignorar el problema: la inicializo con lo que haya en memoria
- Estrategia de inicialización:
 - Inicialización por defecto: Enteros se inicializan en 0, los caracteres en blanco, etc.
 - Inicialización en la declaración: C int i =0, j= 1

b. Analice en los lenguajes: Java, C, Python y Ruby las diferentes formas de inicialización de variables que poseen. Realice un cuadro comparativo de esta característica.

Java	<ul style="list-style-type: none">- Declaración e inicialización en la misma línea: int num = 10;- Declaración e inicialización separadas: int num; num = 10;- Inicialización por defecto: int num; (inicializada a 0 automáticamente)
C	<ul style="list-style-type: none">- Declaración e inicialización en la misma línea: int num = 10;- Declaración e inicialización separadas: int num; num = 10;- No hay inicialización por defecto, el valor inicial es indeterminado
Python	<ul style="list-style-type: none">- Declaración e inicialización en la misma línea: num = 10- No es necesario declarar el tipo de variable antes de su inicialización
Ruby	<ul style="list-style-type: none">- Declaración e inicialización en la misma línea: num = 10- No es necesario declarar el tipo de variable antes de su inicialización

En resumen, Java y C tienen formas similares de inicialización de variables, mientras que Python y Ruby tienen una sintaxis más flexible y sencilla. En Python y Ruby no es necesario declarar el tipo de variable antes de su inicialización, lo que puede hacer que el código sea más fácil de escribir y entender. Por otro lado, Java y C no permiten la asignación de variables sin declarar su tipo previamente. Además, Java tiene la opción de inicialización por defecto, que es útil en ciertos casos, como cuando se desea inicializar un array.

Ejercicio 3: Explique (de al menos un ejemplo de cada uno) los siguientes conceptos asociados al atributo l-valor de una:

El atributo l-valor de una variable hace referencia a su capacidad de ser referenciada o manipulada mediante una expresión de asignación. En otras palabras, una variable con l-valor puede ser asignada a otra variable o valor.

a. Variable estática.

Una variable estática tiene un ámbito de visibilidad limitado a la función o archivo en el que se declara y su valor se mantiene constante durante toda la ejecución del programa. Es decir, es una variable que existe y se inicializa en tiempo de compilación y su valor persiste en memoria durante toda la ejecución del programa.

Ejemplo en C:

```
1 #include <stdio.h>
2
3 void static_example() {
4     static int num = 0;
5     printf("El valor de num es: %d\n", num);
6     num++;
7 }
8
9 int main() {
10     for (int i = 0; i < 5; i++) {
11         static_example();
12     }
13     return 0;
14 }
```

En este ejemplo, la variable num es una variable estática dentro de la función static_example().

La salida del programa sería:

```
1 El valor de num es: 0
2 El valor de num es: 1
3 El valor de num es: 2
4 El valor de num es: 3
5 El valor de num es: 4
```

b. Variable automática o semiestática.

Una variable automática o semiestática se declara dentro de un bloque y su ámbito de visibilidad se limita a ese bloque. Su valor se inicializa al entrar al bloque y se destruye al salir de él. Estas variables también se llaman locales, ya que solo son visibles dentro de la función en la que se declaran.

Ejemplo en Python:

```
1 def automatic_example():
2     num = 0
3     print("El valor de num es: ", num)
4     num += 1
5
6 for i in range(5):
7     automatic_example()
```

En este ejemplo, la variable num es una variable automática dentro de la función automatic_example().

La salida del programa sería:

```
1 El valor de num es: 0
2 El valor de num es: 0
3 El valor de num es: 0
4 El valor de num es: 0
5 El valor de num es: 0
```

c. Variable dinámica.

Una variable dinámica es una variable que se crea y se destruye en tiempo de ejecución. En la mayoría de los lenguajes de programación, estas variables se crean utilizando funciones o métodos específicos como `malloc()` o `new`. El valor de una variable dinámica puede cambiar a lo largo de la ejecución del programa.

Ejemplo en Python:

```
1 def dynamic_example():
2     num = input("Ingrese un número: ")
3     print("El valor ingresado es:", num)
4
5 for i in range(5):
6     dynamic_example()
```

En este ejemplo, la variable `num` es una variable dinámica que se inicializa mediante la entrada de usuario en cada llamada a la función `dynamic_example()`.

La salida del programa dependerá de lo que ingrese el usuario.

d. Variable semidinámica.

Una variable semidinámica es una variable que se inicializa en tiempo de ejecución pero que no cambia de tamaño después de ser creada.

```
1 def semidynamic_example():
2     num = [1, 2, 3]
3     print("El valor de num es:", num)
4     num[1] = 5
5     print("El valor modificado de num es:", num)
6
7 semidynamic_example()
```

En este ejemplo, la variable `num` es una variable semidinámica, ya que se crea en tiempo de ejecución como una lista con tres elementos y su tamaño no cambia después de ser creada. Sin embargo, su valor puede ser modificado, como se ve en la línea `num[1] = 5`.

La salida del programa sería:

```
1 El valor de num es: [1, 2, 3]
2 El valor modificado de num es: [1, 5, 3]
```

Investigue sobre que tipos de variables respecto de su l-valor hay en los lenguajes C y Ada.

En el lenguaje de programación C, se pueden clasificar las variables según su l-valor de la siguiente manera:

- a. Variable estática: se define usando la palabra clave "static" y se mantiene en memoria durante toda la ejecución del programa. Estas variables se inicializan automáticamente a cero si no se les asigna un valor explícitamente.
- b. Variable automática o semiestática: se definen dentro de una función y se eliminan de la memoria cuando la función termina. Estas variables no se inicializan automáticamente, por lo que su valor es indeterminado hasta que se les asigna uno.
- c. Variable dinámica: se asignan en tiempo de ejecución usando funciones como malloc() y calloc(). Estas variables se mantienen en la memoria hasta que se libera explícitamente usando la función free().
- d. Variable semidinámica: se crean usando un tamaño fijo en tiempo de compilación y su valor puede cambiar en tiempo de ejecución. Por ejemplo, un arreglo puede ser una variable semidinámica.

En el lenguaje de programación Ada, las variables se clasifican según su l-valor de la siguiente manera:

- a. Variable estática: se definen usando la palabra clave "constant" o "static" y se mantiene en memoria durante toda la ejecución del programa.
- b. Variable automática o semiestática: se definen dentro de un bloque y se eliminan de la memoria cuando el bloque termina.
- c. Variable dinámica: se asignan en tiempo de ejecución usando la palabra clave "new". Estas variables se mantienen en la memoria hasta que se libera explícitamente usando la palabra clave "delete".
- d. Variable semidinámica: no hay una clasificación explícita para las variables semidinámicas en Ada, pero se pueden crear utilizando arreglos de tamaño fijo. El tamaño del arreglo es fijo en tiempo de compilación, pero el valor de sus elementos puede cambiar en tiempo de ejecución.

Ejercicio 4:

a. ¿A qué se denomina variable local y a qué se denomina variable global?

Una variable local es una variable declarada dentro de una función o bloque de código y su ámbito de alcance se limita a esa función o bloque de código. Por otro lado, una variable global es una variable declarada fuera de cualquier función o bloque de código y su ámbito de alcance se extiende a todo el programa.

b. ¿Una variable local puede ser estática respecto de su l-valor? En caso afirmativo dé un ejemplo

Sí, una variable local puede ser estática respecto a su l-valor. En algunos lenguajes de programación, como C y C++, se pueden definir variables locales como estáticas para mantener su valor entre llamadas sucesivas a una función.

Un ejemplo en C podría ser:

```
1 #include <stdio.h>
2
3 void ejemploFuncion() {
4     static int variableLocal = 0;
5     variableLocal++;
6     printf("El valor de la variable estática es %d\n", variableLocal);
7 }
8
9 int main() {
10    ejemploFuncion(); // Imprime "El valor de la variable estática es 1"
11    ejemploFuncion(); // Imprime "El valor de la variable estática es 2"
12    ejemploFuncion(); // Imprime "El valor de la variable estática es 3"
13    return 0;
14 }
```

En este ejemplo, la variable `variableLocal` se declara como `static` dentro de la función `ejemploFuncion()`. Esto significa que su valor se mantiene entre llamadas sucesivas a la función. En cada llamada a la función, el valor de `variableLocal` se incrementa en uno y se imprime en pantalla. Como la variable es estática, su valor se mantiene entre llamadas y no se pierde cuando la función termina su ejecución.

c. Una variable global ¿siempre es estática? Justifique la respuesta.

No necesariamente. La estática es una propiedad que puede tener una variable independientemente de su alcance. En algunos lenguajes de programación, como C, las variables globales son, por defecto, estáticas en términos de su l-valor, lo que significa que su valor se mantiene durante toda la vida útil del programa y no puede ser modificado por otras funciones o bloques de código. Sin embargo, en otros lenguajes de programación como Java y Python, las variables globales no son estáticas en términos de su l-valor, ya que pueden ser modificadas en tiempo de ejecución y su valor puede ser accedido y modificado por cualquier función o bloque de código en el programa.

En resumen, la naturaleza de las variables globales respecto a su l-valor no depende del hecho de que sean globales o no, sino que depende del lenguaje de programación utilizado y de cómo se manejan las variables en ese lenguaje. Por lo tanto, es importante conocer las características del lenguaje de programación que se está utilizando para comprender el comportamiento de las variables globales en términos de su l-valor.

d. Indique qué diferencia hay entre una variable estática respecto de su l-valor y una constante

La diferencia principal entre una variable estática respecto a su l-valor y una constante es que el valor de una constante no puede ser modificado una vez que se ha definido, mientras que el valor de una variable estática puede ser modificado durante la ejecución del programa.

Una constante se define en el código del programa y su valor es fijo durante toda la ejecución. Es decir, el valor de una constante se define una sola vez y no se puede modificar en tiempo de ejecución. Las constantes se usan para representar valores fijos como pi, el número de días en una semana, entre otros.

Por otro lado, una variable estática se define en el código del programa y su valor se mantiene en memoria durante toda la ejecución del programa, pero el valor de la variable se puede modificar en cualquier momento. Las variables estáticas se utilizan para mantener un valor a lo largo de la vida útil del programa, como contar el número de veces que se llama a una función.

En resumen, la diferencia fundamental entre una variable estática y una constante es que el valor de una constante no puede ser modificado después de ser definido, mientras que el valor de una variable estática puede ser modificado en tiempo de ejecución.

Ejercicio 5:

a. En Ada hay dos tipos de constantes, las numéricas y las comunes. Indique a que se debe dicha clasificación.

La clasificación de las constantes en Ada en numéricas y comunes se debe a que las constantes numéricas son aquellas que se refieren a valores numéricos, mientras que las constantes comunes son aquellas que pueden contener caracteres y símbolos, como cadenas de texto.

b. En base a lo respondido en el punto a), determine el momento de ligadura de las constantes del siguiente código:

H: constant Float:= 3,5;

I: constant:= 2;

K: constant float:= H*I;

El momento de ligadura de las constantes en Ada es durante la compilación del programa, es decir, en tiempo de compilación. En el código proporcionado, el momento de ligadura de la constante "H" es cuando se le asigna el valor 3.5, el momento de ligadura de la constante "I" es cuando se le asigna el valor 2, y el momento de ligadura de la constante "K" es cuando se calcula su valor mediante la multiplicación de "H" e "I". Por lo tanto, el momento de ligadura de "K" es después del momento de ligadura de "H" e "I" durante el proceso de compilación. Como todas las constantes se definen en tiempo de compilación, su valor se conoce antes de que se ejecute el programa.

Ejercicio 6: Sea el siguiente archivo con funciones de C:

Archivo.c

```
{ int x=1; (1)
```

```
int func1();{
```

```

int i;

for (i:=0; i < 4; i++) x=x+1;
}

int func2();{
int i, j;

/*sentencias que contienen declaraciones y
sentencias que no contienen declaraciones*/
.....

for (i:=0; i < 3; i++) j=func1 + 1;
}
}

```

Analice si llegaría a tener el mismo comportamiento en cuanto a asignación de memoria, sacar la declaración (1) y colocar dentro de func1() la declaración static int x =1;

Si, llegaría a tener el mismo comportamiento ya que si se tiene 'static int x=1' en la func1(), la memoria se asignaría una sola vez cuando se llama por primera vez a la función y esta memoria se mantiene asignada durante toda la vida del programa, al igual que tener 'int x = 1' como variable global, por lo tanto, el comportamiento en cuanto a la asignación de memoria sería el mismo.

Ejercicio 7: Sea el siguiente segmento de código escrito en Java, indique para los identificadores si son globales o locales.

<pre> Clase Persona { public long id public string nombreApellido public Domicilio domicilio private string dni; public string fechaNac; public static int cantTotalPersonas; //Se tienen los getter y setter de cada una de las variables //Este método calcula la edad de la persona a partir de la fecha de nacimiento </pre>	<pre> public int getEdad(){ public int edad=0; public string fN = this.getFechaNac(); return edad; } Clase Domicilio { public long id; public static int nro public string calle public Localidad loc; //Se tienen los getter y setter de cada una de las variables } </pre>
--	--

Los identificadores se pueden clasificar de la siguiente manera:

Globales (variables estáticas de clase, aquellos que se declaran a nivel de la clase y pueden ser accedidos sin necesidad de crear una instancia de la clase):

- cantTotalPersonas (estático) en la clase Persona
- nro (estático) en la clase Domicilio

Variables de instancia (no estáticas, son variables de instancia que se declaran dentro de la clase y solo pueden ser accedidas a través de una instancia de la clase):

- id en las clases Persona y Domicilio
- nombreApellido en la clase Persona
- domicilio en la clase Persona
- dni en la clase Persona
- fechaNac en la clase Persona
- calle en la clase Domicilio
- loc en la clase Domicilio

Locales (son aquellos que se declaran dentro de un método y solo pueden ser accedidos dentro del mismo):

- edad en el método getEdad() de la clase Persona
- fN en el método getEdad() de la clase Persona

La diferencia entre variables globales, variables de instancia y variables locales en Java se basa en su alcance y su duración. Las variables globales son visibles en toda la clase y tienen una duración de toda la vida de la aplicación, mientras que las variables de instancia son específicas de cada objeto y duran tanto como el objeto exista. Por último, las variables locales solo son visibles en el método o bloque de código en el que se declaran y tienen una duración limitada a la ejecución de ese método o bloque.

Ejercicio 8: Sea el siguiente ejercicio escrito en Pascal

1- Program Uno;

2- type tpuntero= ^integer;

3- var mipuntero: tpuntero;

4- var i:integer;

5- var h:integer;

6- Begin

7- i:=3;

8- mipuntero:=nil;

```
9- new(mipuntero);
10- mipuntero^:=i;
11- h:= mipuntero^+i;
12- dispose(mipuntero);
13- write(h);
14- i:= h- mipuntero;
15- End.
```

a) Indique el rango de instrucciones que representa el tiempo de vida de las variables i, h y mipuntero.

i: línea 4 – línea 15

h: línea 5- línea 15

mipuntero: línea 9 – línea 12

b) Indique el rango de instrucciones que representa el alcance de las variables i, h y mipuntero.

i: línea 4 – línea 15

h: línea 5 – línea 15

mipuntero: línea 3 – línea 15

c) Indique si el programa anterior presenta un error al intentar escribir el valor de h. Justifique

No, porque a h se le asigna el valor apuntado por mipuntero + i antes de realizar el dispose (mipuntero), es decir, antes de que se libere la memoria.

d) Indique si el programa anterior presenta un error al intentar asignar a i la resta de h con mipuntero. Justifique

Si, se presenta un error porque a h se le intenta restar la dirección de memoria del valor al que apunta mipuntero, y ese valor es nil ya que antes se efectuó un dispose(mipuntero) y no se puede realizar una resta entre un entero (h) y un puntero nulo (nil), por lo tanto se genera un error en tiempo de ejecución.

e) Determine si existe otra entidad que necesite ligar los atributos de alcance y tiempo de vida para justificar las respuestas anteriores. En ese caso indique cuál es la entidad y especifique su tiempo de vida y alcance.

Si, la otra entidad que existe es el programa principal, el cual es una entidad que necesita ligar los atributos de alcance y tiempo de vida para justificar las respuestas anteriores. En el caso de Pascal, el programa principal tiene alcance global y su tiempo de vida es desde su declaración hasta el final del programa (línea 6 – línea 15)

f) Especifique el tipo de variable de acuerdo a la ligadura con el l-valor de las variables que encontró en el ejercicio.

i es de tipo entero, h es de tipo entero y mi puntero es de tipo tpuntero (puntero a entero).

Ejercicio 9: Elija un lenguaje y escriba un ejemplo:

a. En el cual el tiempo de vida de un identificador sea mayor que su alcance

Ejemplo en C:

```
1 #include <stdio.h>
2
3 void suma() {
4     static int num = 0;
5     num++;
6     printf ("El valor de num es: %d\n", num);
7 }
8
9 int main(void) {
10     int i;
11     for (i = 1; i <= 3; i++) {
12         suma();
13     }
14     return 0;
15 }
```

Se imprime en consola:

```
1 El valor de num es: 1
2 El valor de num es: 2
3 El valor de num es: 3
```

En el anterior programa, el alcance de num es solo dentro de la función suma, pero su tiempo de vida es desde la primera vez que se ejecuta la función suma hasta que finaliza el programa ya que se trata de una variable estática.

b. En el cual el tiempo de vida de un identificador sea menor que su alcance

Ejemplo en Pascal:

```
1 program ejemploB;
2
3 type tpuntero = ^integer;
4
5 var
```

```

6 mipuntero: tpuntero;
7 begin
8     new(mipuntero);
9     mipuntero^ := 10;
10    dispose(mipuntero);
11    writeln(mipuntero^);
12 end.

```

En el anterior programa, el alcance de mipuntero desde la línea 6 (su declaración) hasta la 12, pero su tiempo de vida es desde la 8 hasta la 10 (su tiempo de vida termina por el dispose()). Después de liberar la memoria asignada a mipuntero, el puntero deja de existir y no se puede acceder al entero apuntado por él, como se intenta hacer en la línea 11.

c. En el cual el tiempo de vida de un identificador sea igual que su alcance

Ejemplo en Python:

```

1 def ejercicioC():
2     x = 5
3     print(x)
4     y = 10
5     if x == 5:
6         z = 15
7         print(y, z)
8     print(y)
9
10 ejercicioC()

```

En el anterior programa, el alcance de x, y, y z está limitado a la función ejercicioC(). Además, su tiempo de vida coincide con el tiempo de ejecución de la función (desde que se ejecuta la función hasta que termina la función). Por lo tanto, el alcance y el tiempo de vida de x, y, y z son iguales.