Conceptos y Paradigmas de Lenguajes de Programación - Práctica 4

según la práctica el alcance de una variable comienza en la linea siguiente a su declaración

EJERCICIO 1

a)

variable a	
Nombre	a
Alcance	3-16
Tipo	integer
L-valor	automática
R-valor	indefinido
Tiempo de vida	1-16

b)

variables	variable a	variable p
Nombre	a	p
Alcance	3-16	p: 4-16 / p^: 4-16
Tipo	integer	puntero
L-valor	automática	p: automática / p^: dinámica
R-valor	indefinido	p: nil / p^: indefinido
Tiempo de vida	1-16	p: 1-16 / p^: 7-15

el alcance de p y p^ son el mismo
y el tiempo de vida de p^ es desde el new al dispose

EJERCICIO 2

- **Inicialización por defecto:** las variables se inicializan con un valor por defecto de manera automática. Ej: enteros en 0, caracteres en blanco, etc.
- **Inicialización en la declaración:** las variables pueden inicializarse en el mismo momento que se declaran. Ej: “int i = 0;”.
- **Ignorar el problema:** la variable toma como valor inicial lo que hay en memoria (la cadena de bits asociados al área de almacenamiento). Puede llevar a errores y requiere chequeos adicionales.

Lenguaje	Inicialización por defecto	Inicialización en la declaración	Ignorar el problema
Java	SI (solo en variables de instancia)	SI	NO
C	SI (solo en variables globales/estáticas)	SI	SI (en locales -> valor basura)
Python	SI (al declarar ser inicializa)	SI	NO
Ruby	SI (al declarar ser inicializa)	SI	NO

EJERCICIO 3

Variable	Descripción	Ejemplo en C	Ejemplo en ADA
Estática	**C** es el único lenguaje que tiene variables estáticas puras. Se declaran con la palabra clave **static**. Se reserva su espacio de memoria previo a la ejecución del programa (en compilación). Persiste luego del programa.	static int static_var = 10;	NO TIENE
Automática	Son variables que se alocan en memoria cuando se aloca la unidad que las contiene. Son las más comunes. Ej: var locales a un programa o un método.	int auto_var = 10;	Auto_var: Integer := 20;
Dinámica	Son los punteros. Contienen dos variables: el puntero mismo (p) que es una var automática, y la var que contiene el puntero (p^). Al hacer new(p), p^ se aloca en la memoria Heap y se vuelve dinámica.	int *dynamic_var;	type Int_Ptr is access Integer; Dynamic_Variable: Int_Ptr := new Integer'(20);
Semidinámica	Son variables cuyo tamaño puede ser dinámico, pero su dirección de memoria es estática. Solamente **ADA** posee este tipo de variables (arreglos semidinámicos).	NO TIENE	Array : Array (1..n) of Integer;

EJERCICIO 4

a)

- **Variable local**: son todas las referencias a variables que se han creado dentro de una unidad (programa o subprograma).

- **Variable global**: son todas las referencias a variables que se han creado en el programa principal.

b) Si, una variable local puede ser estática respecto de su l-valor. Ejemplo: solamente C posee la posibilidad de definir variables estáticas puras. static int x = 0;

```
#include <stdio.h>

void contador() {
    static int x = 0;  // variable local, pero estática
    x++;
    printf("x = %d\n", x);
}

int main() {
    contador(); // x = 1
    contador(); // x = 2
    contador(); // x = 3
    return 0;
}
```

CONSULTAR !

c) No, una variable global puede o no ser estática. El alcance de la misma no condiciona el momento en el que se alocan en memoria. Por ejemplo en C podemos tener variables globales estáticas (static int a = 0;) y variables globales no estáticas o automáticas (extern int b = 0;). En el caso de las estáticas, su Tiempo De Vida es mayor.

d)

	variable estática	**constante**
l-valor	es la dirección de memoria donde se almacena su valor	no tiene un l-valor tradicional. no ocupa espacio de memoria accesible como las variables.
comportamiento	persiste a lo largo del programa pero su valor **puede** modificarse durante la ejecución.	persiste a lo largo del programa pero su valor **no** puede modificarse.
Una variable **estática** se utiliza cuando se necesita preservar su valor entre las llamadas a una función, pero aún así permitir que su valor cambie dentro del contexto de la función. Por otro lado, una **constante** se utiliza cuando se necesita un valor que no cambie en absoluto durante la ejecución del programa y que sea conocido en tiempo de compilación.		

EJERCICIO 5

a) La clasificación de constantes en **numéricas** y **comunes** en el lenguaje de programación **ADA** se debe al **tipo de datos** que representan y **cómo se manejan por el compilador**.

constante numérica	**constante definida**
representan valores numéricos (enteros, reales, tipos numéricos definidos por el usuario). se utilizan para asignar valores fijos a variables numéricas y para realizar cálculos matemáticos.	representan valores que no son necesariamente numéricos. se utilizan para asignar valores fijos a variables de tipos no numéricos (cadenas de caracteres, tipos enumerados, y tipos definidos por el usuario).

b) La constante H se liga en el momento en que se encuentra su declaración, es decir, cuando se le asigna el valor 3.5
La constante I también se liga en el momento de su declaración, asignándole el valor 2.
La constante K se liga luego de que H e I ya hayan sido ligadas. En este caso, tanto H como I fueron ligadas previamente por lo que se puede calcular el valor de K = 7.

EJERCICIO 6

En C las variables globales tienen comportamiento estático en cuanto a su l-valor, por lo tanto, sería lo mismo (en cuanto a alocación de memoria) sacar la variable global ***int x = 1;*** y declararla como ***static int x = 1;*** dentro de **func1()**. Ambas se alocan una vez previo a la ejecución del programa y persisten durante toda la ejecución.

EJERCICIO 7

En Java las variables globales como tal no existen, pero una variable definida como **static** dentro de una clase se considera global ya que se comparte entre todas las instancias de esa clase.

variables globales	variables locales
public static int cantTotalPersonas; public static int nro;	**Clase Persona** public long id; public string nombreApellido; public Domicilio domicilio; private string dni; public string fechaNac; *(var locales del método getEdad)* *public int edad;* *public string fN;* **Clase Domicilio** public long id; public string calle; public Localidad loc;

EJERCICIO 8

a)

variable	tiempo de vida
i	1-15
h	1-15
mipuntero	1-15
mipuntero^	9-12

b)

variable	alcance
i	5-15

h	6-15
mipuntero	4-15
mipuntero^	4-15

alcance de mipuntero^ ???

c) No presenta un error al intentar escribir el valor de h porque su valor fue calculado a través de mipuntero^ y de i previo al dispose de mipuntero.

d) En este caso se presenta un error ya que se intenta restarle nil a la variable h.

e) Podría pensarse que una entidad faltante es mipuntero^, pero ya fue definida anteriormente.
La entidad faltante sería el programa principal, este necesita ligar los atributos de alcance y tiempo de vida para justificar las respuestas anteriores. Su alcance es global y su tiempo de vida es desde su declaración hasta el final del programa (de 1 a 15).

f) CONSULTAR: mipuntero es tipo puntero y mipuntero^ tipo integer?

variable	tipo de acuerdo a la ligadura con el l-valor
i	automática, integer
h	automática, integer
mipuntero	automática, puntero
mipuntero^	dinámica, integer

EJERCICIO 9

a) Una variable tipo static en una función. Su tiempo de vida excede la función pero su alcance es solamente dentro de la misma.

```
int* func() {
   static int x = 5;
   return &x;
}
```

b) Una variable tipo puntero p. Alcance de p^ es menor que su tiempo de vida ya que su alcance es desde new hasta dispose.

```
program ejemploB;

var
  p: ^Integer;

begin
  new(p);
  p^ := 42;
  dispose(p);
end.
```

c) Una variable local de una función. Su tiempo de vida va a ser la función y su alcance también.

```
void func() {
    int x = 10;
    // x vive y es accesible
solo dentro de func()
}
```

EJERCICIO 10

Si, se puede asegurar que el tiempo de vida y el alcance de la variable **c** en los 3 lenguajes es siempre todo el proceso donde se encuentra definida, ya que, no existen subprocesos internos que puedan modificar el alcance o el tiempo de vida de la misma.

Esto se debe a que las variables están declaradas dentro del procedimiento, y en todos estos lenguajes, las variables locales en un procedimiento tienen su **alcance limitado** al procedimiento y su **tiempo de vida** abarca desde la entrada al procedimiento hasta su salida, momento en el cual la variable es destruida.

EJERCICIO 11

a) La opción verdadera es la **IV**: El tipo de dato de una variable es el conjunto de valores que puede tomar y el conjunto de operaciones que se pueden realizar sobre esos valores.

b) Tipo de dato de una variable:

- Conjunto de valores que se le pueden asociar a una variable y el conjunto de operaciones permitidas para la misma.
- El tipo de dato ayuda a:
  - Proteger a las variables de operaciones no permitidas -> chequear tipos.
  - Verificar el uso correcto de variables.
  - Detectar errores en forma temprana.
  - Mejorar la confiabilidad del código.
- Antes de que una variable pueda ser referenciada, debe ser ligada a un tipo.
- Existen 3 tipos: predefinidos por el lenguaje, definidos por el usuario, y los tipos de datos abstractos.
- Su ligadura puede ser estática o dinámica.

EJERCICIO 12

```
1. with text_io; use text_io;
2. Procedure Main is;
3. type vector is array(integer range <>);
4. a, n, p:integer;
5. v1:vector(1..100);
6. c1: constant integer:=10;
7. Procedure Uno is;
8. type puntero is access integer;
9. v2:vector(0..n);
10. c1, c2: character;
11. p,q: puntero;
12. begin
13.       n:=4;
14.       v2(n):= v2(1) + v1(5);
15.       p:= new puntero;
16.       q:= p;
17.       .......
18.       free p;
19.       ......
20.       free q;
21.       ......
22.       end;
23.begin
24.       n:=5;
25.       ....
26.       Uno;
27.       a:= n + 2;
28.       ....
29. end
```

identificador	tipo (l-valor)	r-valor	alcance	tiempo de vida
main (linea 2)	-	-	3-29	2-29
uno (linea 8)	-	-	8 -29	7-22
a (linea 4)	automática	basura	5-29	1-29
n (linea 4)	automática	basura	5-29	1-29
p (linea 4)	automática	basura	5-11 -> 23-29	1-29
v1 (linea 5)	automática	basura	6-29	1-29
c1 (linea 6)	automatica	basura	7-10 -> 23-29	1-29
v2 (linea 9)	semidinamica	basura	10-22	7-22
c1 (linea 10)	automatica	basura	11-22	7-22
c2 (linea 10)	automatica	basura	11-22	7-22
p (linea 11)	automatica	nil	12-22	12-22
p^	dinamica	basura	12-22	15-18
q (linea 11)	automatica	nil	12-22	12-22
q^	dinamica	basura	12-22	16-20

CONSULTAR LOS CAMPOS EN ROSA (consultado: están bien)

en ADA hay dos tipos de constantes

1. c1: constant integer:=10; es constante común (r-valor = indefinido) (en ejecución)

2. c1: constant := 10; es constante numérica (r-valor = 10) (en compilación)

EJERCICIO 13

a) No, el nombre de una variable no puede condicionar el tiempo de vida de la misma, el concepto de nombre es lo que se utiliza para referenciar a la variable mientras que el tiempo de vida de la misma es el periodo de tiempo que la variable está alocada en memoria y su binding existe.

b) El nombre de una variable no define por si mismo su alcance, pero si puede condicionar el alcance de la misma ya que el alcance es definido como el rango de instrucciones en el que es conocido el nombre de la variable, por lo tanto, este podría condicionar el factor del alcance. (Ej: podría haber otra variable con el mismo nombre definida en una función posterior y que se enmascare modificando su alcance).

c) No, el nombre no condiciona el r-valor de la misma, este último concepto es definido como el valor codificado almacenado en la locación asociada a la variable, es decir, a su l-valor. Nosotros podemos acceder al r-valor gracias al nombre ya que este es el que nos sirve para poder referenciar a la misma.

d) No, el nombre de una variable no condiciona el tipo de la misma, el tipo normalmente se define en la declaración de la variable y es el que condiciona el conjunto de valores y operaciones que podemos hacer con la variable pero esta puede tener cualquier nombre y esto no va a interferir con el tipo de la misma.

EJERCICIO 14

EN C:

el r-valor es 0 o blanco si la variable es global o estatica
el r-valor es basura/indefinido si es local
el r- valor de punteros es nil

identificador	tipo (l-valor)	r-valor	alcance	tiempo de vida
v1 (linea 1)	automatica	0	2-14 9-12 21-23	1-28
a (linea 2)	automatica	nil	3-16	1-28
a^	dinamica	basura	3-16	15-16
v1 (linea 4)	automatica	basura	5-8	3-8
y (linea 4)	automatica	basura	5-8	3-8
var 3 (linea 10)	estatica	0	11-16	<1-28>
v1 (linea 12)	automatica	basura	13-16	9-16
y (linea 12)	automatica	basura	13-16	9-16
var1 (linea 14)	automatica	basura	15	13-15
aux (linea 17)	estatica	0	18-25	<1-28>
v2 (linea 18)	automatica	0	7-8 12-16 19-28	1-28
aux (linea 25)	automatica	basura	26-28	24-28
main (linea 9)	-	-	10-16	9-16
fun 2 (linea 3)	-	-	4-16	3-8
fun 2 (linea 19)	-	-	20-28	19-23
fun 3 (linea 24)	-	-	25-28	24-28

lo que se extiende con el extern es el ALCANCE. no el tiempo de vida. para una var global el tiempo de vida es todo el programa, para una local es solo el método donde esta definida. (para una local puede ser método o una estructura tipo for)

el r-valor de funciones/métodos es indefinido.

EJERCICIO 15

modificador	javascript	python
const	declara una variable con un valor constante que no puede ser reasignado. debe ser inicializado al declararlo.	no hay un equivalente directo. las variables en Python pueden ser reasignadas, pero una convención similar a “const” es usar nombres de variables en mayúsculas para indicar que no deben ser modificadas.
var	declara una variable de alcance global o de función. no tiene restricciones de reasignación o redeclaración. es menos comúnmente utilizado desde la introducción de let y const.	no hay un equivalente directo. en Python, todas las variables declaradas dentro de una función tiene un alcance local. variables definidas fuera de las funciones tienen alcance global.
let	declara una variable de bloque con un alcance limitado al bloque de código en el que está definida. puede ser reasignada pero no redeclarada en el mismo ámbito.	no hay un equivalente directo. sin embargo, las variables definidas en un bloque de código (como dentro de un for o un if) tienen un alcance limitado a ese bloque.
ausencia	su comportamiento depende del contexto: • variables declaradas fuera de una función: se comportan como variables globales (var). • variables declaradas dentro de una función: se comportan como variables locales de la función (var) de todas maneras, se considera una mala práctica declarar variables sin especificar un modificador.	las variables sin declarar son consideradas variables locales en Python, y arrojarán un error si se intentan utilizar antes de ser inicializadas. es una buena práctica inicializar todas las variables antes de usarlas.