Práctica 4: Variables

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;
```

Variable “a”:

- Nombre: a.
- Alcance: 4-16.
- Tipo: integer.
- L-Valor: automática.
- R-Valor: indefinido.
- Tiempo de vida: 1-16.

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?

Variable: “p”:

- Nombre: p.
- Alcance: p: 5-16 - p^: 5-16.
- Tipo: puntero.
- L-Valor: p: automático - p^: dinámico.
- R-Valor: p: nil - p^: indefinido.
- Tiempo de vida: p: 1-16 - p^: 7-15

Ejercicio 2:

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

- Inicialización por defecto: Las variables se inicializan con un valor por defecto, por ejemplo, los enteros en 0, los caracteres en blanco, etc.
- Inicialización en la declaración: Las variables pueden inicializarse en el mismo momento que se declaran, por ejemplo “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.

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

Lenguaje	Inicialización por defecto	Inicialización por declaración	Ignorar el problema
Java	✓	✓	✗
C	✓ (sólo para variables globales y estáticas)	✓	✓
Python	✗	✓	✗
Ruby	✗	✓	✗

Ejercicio 3: Explique los siguientes conceptos asociados al atributo l-valor de una:

- **a.** Variable estática.
- **b.** Variable automática o semiestática.
- **c.** Variable dinámica.
- **d.** Variable semidinámica.

De al menos un ejemplo de cada uno.

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

Variable	Definición	Ejemplo en C	Ejemplo en Ada
Variable estática	Su tiempo de vida abarca toda la ejecución del programa. Se asigna al inicio y se libera al final. Conserva su valor entre llamadas si es local.	static int x = 0;	No corresponde
Variable automática o semiestática	Su tiempo de vida está limitado al bloque donde se declara. Se crea al entrar y se destruye al salir.	int x = 10;	Auto_Var:Integer:= 20;
Variable dinámica	Se crea y destruye durante la ejecución, generalmente en la heap. Su tiempo de vida es controlado explícitamente por el programador.	int*dynamic_var;	P : access Integer := new Integer'(20);
Variable semidinámica	Son variables cuyo tamaño puede ser dinámico, pero su dirección de memoria es estática. (Visto desde la perspectiva de Ada)	No corresponde	Array: Array(1..n) of Integer;

Ejercicio 4:

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

Global: Son todas las referencias a variables creadas en el programa principal.

Local: Son todas las referencias a variables creadas dentro de una unidad (programa o subprograma).

- **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 de su L-valor si se declara como static (en lenguajes como C) o si se gestiona su almacenamiento de manera estática (como en Ada a nivel de paquete). Esto significa

que su dirección en memoria (L-valor) no cambia durante la ejecución del programa, a pesar de que su alcance es local.

```c
#include <stdio.h>

void funcion() {
    static int numero = 0; // Variable local con almacenamiento estático
    printf("Valor: %d, Dirección: %p\n", numero, (void*)&numero);
    numero++;
}

int main() {
    funcion(); // Primera llamada
    funcion(); // Segunda llamada
    funcion(); // Tercera llamada
    return 0;
}
```

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

No, una variable global puede ser o no estática, el alcance de las mismas no condiciona el momento en el que se alocan en memoria, por ejemplo, en C podemos tener variables globales estáticas mediante el uso de la palabra clave “static” o podemos tener variables globales no estáticas mediante el uso de la palabra clave “extern”.

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

Variable estática	Constante
L-Valor: El l-valor de una variable estática es la dirección de memoria donde se almacena su valor.	L-Valor: Las constantes no tienen un l-valor en el sentido tradicional. El l-valor se refiere a la dirección de memoria de una variable, pero las constantes no ocupan una posición en la memoria de la misma manera que las variables.
Comportamiento: Una variable estática conserva su valor entre las llamadas a la función en la que se declara. Esto significa que su valor persiste a lo largo de la ejecución del programa, pero su valor puede modificarse mediante operaciones de asignación dentro de la función.	Comportamiento: Una constante tiene un valor que no puede cambiar durante la ejecución del programa. Se define mediante la palabra clave “const”, y una vez que se le asigna un valor, ese valor no puede ser modificado posteriormente.
En términos de l-valor, mientras que una variable estática tiene una dirección de memoria asociada donde se almacena su valor y que puede ser modificada, una constante no tiene una dirección de memoria explícita, ya que su valor es tratado directamente como un valor literal durante la compilación.	
En términos de comportamiento, una variable estática se utiliza cuando se necesita preservar su valor entre las llamadas a una función, pero aun 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. En Ada hay dos tipos de constantes, las numéricas y las comunes. Indique a que se debe dicha clasificación.

En Ada, la clasificación de constantes en numéricas y comunes se debe a la forma en que se definen y utilizan. Las constantes numéricas son aquellas que representan valores numéricos, como enteros, reales, complejos, etc. Estas constantes se definen utilizando una notación numérica estándar y la ligadura se produce durante la compilación; esto significa que los valores de las constantes numéricas se conocen antes de que el programa se ejecute y se incorporan directamente en el código objeto generado por el compilador.

Por otro lado, las constantes comunes son aquellas que representan valores que no son numéricos, como caracteres, cadenas de texto, tipos enumerados, etc. Estas constantes se definen utilizando una notación específica para cada tipo de constante. No se evalúan durante la compilación. En su lugar, se evalúan en tiempo de ejecución, es decir, cuando el programa se está ejecutando. Esto significa que los valores de las constantes comunes se conocen solo después de que el programa se haya iniciado.

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;

- H: es una constante numérica con el valor 3.5 del tipo float. La ligadura se realiza en ejecución.
- I: es una constante numérica con el valor 2, sin tipo especificado así que se asume que es integer. La ligadura se realiza en la compilación. Es una constate no tipada.
- K: se define como una expresión que utiliza las constantes H e I. Dado que tanto H como I son constantes numéricas, su valor se evalúa durante la compilación. Por lo tanto, la constante K también se evalúa durante la ejecución, en el primer momento de ligadura.

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 alocación de memoria, sacar la declaración **(1)** y colocar dentro de **func1()** la declaración **static int x =1;**

En ambos casos, x se asigna en el segmento de datos (no en la pila), y su tiempo de vida es estático (toda la ejecución del programa). Por lo tanto, el comportamiento en cuanto a alocación de memoria es el mismo en términos de dónde y cómo se almacena x.
La diferencia está en el alcance: en el código original, x es global y accesible desde cualquier función, mientras que, en el código modificado, x es local a func1() en cuanto a visibilidad, pero su almacenamiento sigue siendo estático.

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

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
	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
	}

Identificadores globales:

- public static int cantTotalPersonas
- public static int nro

Identificadores locales:

- public long id
- public string nombreApellido
- public Domicilio domicilio
- private string dni
- public string fechaNac
- public long id
- public string calle
- public Localidad loc

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-     mipunterno^:=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: 1-15

h: 1-15

mipuntero: 1-15
mipuntero^: 9-12

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

i: 5-15
h: 6-15
mipuntero: 4-15
mipuntero^: 4-15

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

No, el programa no presenta error al querer imprimir el valor de “h” ya que se le asigna la suma del valor de la variable a la que apunta el puntero (“i”) antes de que se le haga dispose con el valor de la variable “i”.

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

Si, el programa generaría error ya que “h” tiene como valor nil porque se realizó un dispose, y no es una operación válida hacer una resta entre un tipo de dato entero y el valor nil.

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.

El programa principal puede considerarse una "otra entidad" que 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 la línea 1 hasta la 15.

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

Variable “i”: Automática, Integer.
Variable “h”: Automática, Integer.
Variable “mipuntero”: Automática, Puntero.
Variable “mipuntero^”: Dinámica, Integer.

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

```
1. static int aux;
2. int v2;
3. static int fun2( )
4. { extern int v1;
5. aux=aux+1;
6. }
7. int fun3( )
8. { int aux;
9. aux=aux+1;
10. }
```

Identificador	Lvalor	Rvalor	Alcance	T. vida	Inciso
aux	estática	0-1	1-8 10->	1-10	B
v2	automática	0	2-10	1-10	A
fun2			3-10	3-6	B
v1	automática		4-6	3-6	A
fun3			7-10	7-10	C
aux	automática		8-10	7-10	A

Ejercicio 10: Si tengo la siguiente declaración al comienzo de un procedimiento:

int c; **en C**

var c:integer; **en Pascal**
c: integer; **en ADA**

Y ese procedimiento NO contiene definiciones de procedimientos internos. ¿Puedo asegurar que el alcance y el tiempo de vida de la variable “c” es siempre todo el procedimiento en donde se encuentra definida? Analícelo y justifique la respuesta, para todos los casos.

Aunque no haya definiciones de procedimientos, no se me asegura que no haya definiciones de variables nuevas, y en caso de suceder esto, si se definiese una con el mismo identificador que las previamente definidas, se me enmascararían y perderían parte de su alcance. Exceptuando Pascal ya que no se puede declarar variables fuera de funciones.
En Pascal el tiempo de vida será todo el bloque de código, mientras que en C y en Ada, si se declarase dentro de bloques “declare” (Ada) o un bloque de código independiente (C) el tiempo de vida sería lo que viva este bloque.

Ejercicio 11: a) Responda Verdadero o Falso para cada opción. El tipo de dato de una variable es:
I) Un string de caracteres que se usa para referenciar a la variable y operaciones que se pueden realizar sobre ella.
Falso. El tipo de dato no es un string de caracteres que referencia a la variable; eso es el identificador de la variable.

II) Conjunto de valores que puede tomar y un rango de instrucciones en el que se conoce el nombre.
Falso. Aunque la primera parte es correcta, la segunda parte (rango de instrucciones) no pertenece a la definición del tipo de dato

III) Conjunto de valores que puede tomar y lugar de memoria asociado con la variable.
Falso. El tipo de dato no incluye el "lugar de memoria asociado con la variable". Esa es una propiedad de la variable en tiempo de ejecución, no del tipo de dato.

IV) Conjunto de valores que puede tomar y conjunto de operaciones que se pueden realizar sobre esos valores.
Verdadero.

b) Escriba la definición correcta de tipo de dato de una variable.
Conjunto de valores que se le pueden asociar a una variable junto con el conjunto de operaciones permitidas para la misma.

Ejercicio 12: Sea el siguiente programa en ADA, completar el cuadro siguiente indicando para cada variable de que tipo es en cuanto al momento de ligadura de su l-valor, su r-valor al momento de alocación en memoria y para todos los identificadores cuál es su alcance y cuál es su el tiempo de vida. Indicar para cada variable su r-valor al momento de alocación en memoria

Ident.	Tipo	r-valor	Alcance	T.V.
a (linea 4)	automática	basura	5-29	2-29
n (línea 4)	automática	basura	5-29	2-29

```
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
```

p (línea 4)	automática	basura	5-11 / 23-29	2-29
v1 (línea 5)	automática	basura	6-29	2-29
c1 (línea 6)	automática	basura	7-10 / 23-29	2-29
v2 (línea 9)	semidinámico	basura	10-22	7-22
c1 (línea 10)	automática	basura	11-22	7-22
c2 (línea 10)	automática	basura	11-22	7-22
p (línea 11)	automática	nil	12-22	7-22
p^	dinámica	basura	12-22	15-18
q (línea 11)	automática	nil	12-22	7-22
q^	dinámica	basura	12-22	16-20
Main (línea 2)	-	-	3-29	2-29
Uno (línea 7)	-	-	8-29	7-22

Aclaración:
Ident.= Identificador / **Tipo** es el tipo de la variable respecto del l-value
T.V. = Tiempo de Vida / **r-valor** debe ser tomado al momento de la alocación en memoria.
El alcance de los identificadores debe indicarse desde la línea siguiente a su declaración.

Ejercicio 13: El nombre de una variable puede condicionar:

- **a)** Su tiempo de vida.
- **b)** Su alcance.
- **c)** Su r-valor.
- **d)** Su tipo.

Justifique la respuesta

a) Su tiempo de vida: Falso. El tiempo de vida depende de dónde se declara la variable, no de su nombre. Podría ser verdadero si existen un lenguaje que define reglas para constantes, por ejemplo: mayúsculas para constantes.
b) Su alcance: Verdadero.
c) Su r-valor: Falso. El r-valor depende de las asignaciones y operaciones, no del nombre.

d) Su tipo: Falso. El tipo se define en la declaración de la variable, no por su nombre. Las convenciones de nombres son prácticas humanas, pero no afectan la definición del tipo. Podría considerarse verdadero por ejemplo en Fortran, según su prefijo se asume que es de un tipo determinado.

Ejercicio 14: Sean los siguientes archivos en C, los cuales se compilan juntos
Indicar para cada variable de que tipo es en cuanto al momento de ligadura de su l-valor.
Indicar para cada identificador cuál es su alcance y cuál es su el tiempo de vida.
Indicar para cada variable su r-valor al momento de alocación en memoria

ARCHIVO1.C

```
1.   int v1;
2.   int *a;
3.   Int fun2 ()
4.   { int  v1, y;
5.     for(y=0; y<8; y++)
6.     { extern int v2;
7.     ...}
8.   }
9.   main()
10.  {static int var3;
11.  extern int v2;
12.  int  v1, y;
13.  for(y=0; y<10; y++)
14.  { char var1='C';
15.   a=&v1;}
16.  }
```

ARCHIVO2.C

```
17.  static int aux;
18.  int v2;
19.  static int fun2( )
20.  { extern int v1;
21.    aux=aux+1;
22.    …
23.  }
24.  int fun3( )
25.  { int aux;
26.    aux=aux+1;
27.    …
28.  }
```

Ident.	Tipo	r-valor	Alcance	T.V.
v1 (línea 1)	automática	0	2-4 -> 9-12 -> 21-23	1-28
a (línea 2)	automática	null	3-16	1-28
a^	dinámica	basura	3-16	15-16
fun2 (línea 3)	-	-	4-16	3-16
v1 (línea 4)	automática	basura	5-8	3-8
y (línea 4)	automática	basura	5-8	3-8
main (línea 9)	-	-	10-16	9-16
var3 (línea 10)	estática	0	11-16	<1-28>
v1 (línea 12)	automática	basura	13-16	9-16
y (línea 12)	automática	basura	13-16	9-16
var1 (línea 14)	automática	basura	15	13-15
aux (línea 17)	estática	0	18-25	<1-28>
v2 (línea 18)	automática	0	7 -> 12-16 -> 19-28	1-28
fun2 (línea 19)	-	-	20-23	19-23
fun3 (línea 24)	-	-	25-28	24-28
aux (línea 25)	automática	basura	26-28	24-28

Aclaración:
Ident.= Identificador
T.V. = Tiempo de Vida
r-valor debe ser tomado al momento de la alocación en memoria
El *alcance* de los *identificadores* debe indicarse desde la línea siguiente a su declaración.

Ejercicio 15: Para javascript investigue la diferencia semántica para declarar una variable utilizando los modificadores const, var, let y la ausencia de cualquiera de estos. Compárelo con un lenguaje de su preferencia.

En JavaScript, existen cuatro formas de declarar variables: const, var, let y la ausencia de cualquiera de estos.

- const: se utiliza para declarar una variable que no cambiará de valor en todo el programa. Una vez que se asigna un valor a una variable constante, no se puede cambiar.
- var: se utilizaba antes de la introducción de let en ECMAScript 6, y aún es compatible con versiones antiguas de JavaScript. var tiene un alcance de función o global, lo que significa que una variable declarada con var dentro de una función estará disponible en cualquier lugar dentro de esa función. También se puede utilizar para declarar variables globales.
- let se utiliza para declarar una variable cuyo valor puede cambiar. A diferencia de var, tiene un alcance de bloque, lo que significa que una variable declarada con let dentro de un bloque (por ejemplo, dentro de un if o un for) no estará disponible fuera de ese bloque.
- Si declaramos una variable sin utilizar ninguna palabra clave, como var, let o const, se convierte en una variable global si se declara fuera de una función.

Comparando con un lenguaje como Python, la diferencia semántica es similar en cuanto a la asignación de variables. Python tiene una palabra clave const llamada final, que se utiliza para declarar variables constantes en Python 3.8 y versiones posteriores. En Python, la declaración de variables se realiza automáticamente al asignar un valor a una variable, no es necesario utilizar ninguna palabra clave para declarar una variable.

En general, la principal diferencia semántica entre const, var y let en JavaScript es el alcance y la capacidad de cambiar el valor de la variable. Mientras que, en Python, la declaración de variables es más simple, ya que no es necesario utilizar ninguna palabra clave para declarar una variable.