**Conceptos y Paradigmas de Lenguajes de Programación 2024**
**Práctica Nro. 4**
**Variables**
**Objetivo:** Conocer el manejo de identificadores en memoria y como lo definen e implementan los diferentes lenguajes.

**Ejercicio 1:**

**a) Tome una de las variables de la línea 3 del siguiente código e indique y defina cuales 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;
```

Nombre: a
Tipo: integer
Alcance: Estático, 4-16
L-valor: Automatica
R-valor: Basura

**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?**

Nombre: p
Tipo: puntero
Alcance: Estatico, 5-15
L-Valor: Automatica
R-Valor: Basura
Esta variable contiene el valor la dirección de memoria de un Integer

p^ contendría el valor del integer y su L-valor sería dinamico

**Ejercicio 2:**

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

- **Inicialización por defecto:** Se inicializan con un valor por defecto, cómo 0 para los enteros, caracteres en blanco, funciones en void, etc

- **Inicialización en la declaración:** Se inicializan en el mismo momento que se declaran, por ejemplo: int n = 0;

- **Ignorar el problema:** Toma como valor inicial lo que hay en memoria

**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.**

| | Java | C | Python | Ruby |
|---|---|---|---|---|
| Inicialización por defecto | Si | Si | No | No |
| Inicialización en la misma línea | Si | Sólo globales y estaticas que se inicializan en 0 | Si | Si |
| Ignorar el problema | No | Si | No | No |

**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 estatica:** Se alocan en memoria antes de la ejecución y su tiempo de vida trasciende a la ejecución del programa. Para la cátedra, las únicas variables consideradas estaticas son las *static* de C

```
#include <stdio.h>

void funcion() {
    // Variable estática
    static int x = 0;
    x++;
    printf("Variable estática: %d\n", x);
}

int main() {
    funcion(); // Imprime Variable estática: 1
    funcion(); // Imprime Variable estática: 2
    funcion(); // Imprime Variable estática: 3
    return 0;
}
```

**Variable automática:** Se alocan en memoria cuando aparece una declaración en la ejecución y su tiempo de vida es desde la línea siguiente de la declaración hasta el fin del bloque que la contiene.

```
#include <stdio.h>

void funcion() {
    // Variable automática
    int y = 0;
    y++;
    printf("Variable automática: %d\n", y);
}

int main() {
    funcion(); // Imprime Variable automática: 1
    funcion(); // Imprime Variable automática: 1
    funcion(); // Imprime Variable automática: 1
    return 0;
}
```

**Variable dinámica:** Son punteros. Se alocan cuando una instrucción lo solicita de manera explícita y se desaloca con una sentencia explícita que lo solicita o cuando termina el programa o en caso de que el lenguaje implemente Garbage Collector, se pierde la referencia. Lo entendemos como dos variables, el puntero (contiene dirección de memoria) y el puntero^ (contiene el R-valor)

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int a = 5;

    int *p; // Se declara puntero a int

    p = (int *)malloc(sizeof(int)); // Se asigna memoria dinámica para p

    *p = a; // Guardamos el valor de a en la memoria asignada a través de p

    *p = 10; // Modificamos el valor de a a través de p

    printf("Variable dinámica: %d\n", *p); // Imprimimos el valor almacenado en la memoria asignada a través de p

    free(p); // Se desaloca la memoria asignada para p

    return 0;
}
```

**Variable semidinamica:** Variables con dirección de memoria que se aloca de manera estática (su dirección no cambia) pero tamaño no determinado (dinámico). Para la cátedra, el único caso de variable semidinamica son los arreglos semidinamicos de ADA.

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

procedure Main is
   -- Declaración de un arreglo semidinámico
   type Array_Type is array (Positive range <>) of Integer;

   -- Variable semidinámica
   Arr : Array_Type(1..10); -- Cambiar a (Positive range <>)

   -- Tamaño del arreglo
   N : Natural := 10; -- Cambiar a otro valor si se desea un tamaño diferente

begin
   -- Asignación de valores al arreglo
   for I in Arr'Range loop
      Arr(I) := I * 2;
   end loop;

   -- Impresión de valores del arreglo
   for I in 1..N loop
      Put_Line("Variable semidinámica: " & Integer'Image(Arr(I)));
   end loop;
end Main;
```

**Ejercicio 4:**

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

Esta definición está relacionada al alcance de la variable.

- Variable local es aquella que es conocida dentro del bloque que la contiene.
- Variable global es aquella que es conocida en 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 de su l-valor al utillizar la palabra clave static en su declaración. Al hacerlo, la variable conserva su valor entre llamadas sucesivas a la función en la que se declara, manteniendo su l-valor a lo largo del tiempo de ejecución del programa.

```
#include <stdio.h>

void funcionEstatica() {
    static int contador = 0; // Variable local estática

    contador++; // Incrementa el contador en cada llamada a la función
    printf("El valor de contador es: %d\n", contador);
}

int main() {
    funcionEstatica(); // Llamada a la función
    funcionEstatica(); // Llamada nuevamente
    funcionEstatica(); // Y otra llamada más

    return 0;
}
```

En este ejemplo, la variable contador se declara como estática dentro de la función funcionEstatica(). Cada vez que se llama a funcionEstatica(), contador conserva su valor entre llamadas y se incrementa en uno.

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

No necesariamente. La propiedad de ser estatica o no no depende ni se define en función del alcance de la varibale. Una variable global no estatica muere al terminar la ejecución del programa. Si es estática, su valor se mantendría sucesivamente entre diferentes ejecuciones del programa.

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

- Una variable estática conserva su dirección de memoria pero puede cambiar su R-Valordurante la ejecución.
- Una variable global conserva su R-Valor fijo, pero podría cambiar su L-Valor durante la ejecución.

**Ejercicio 5:**

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

Las constantes numéricas son aquellas que incluyen enteros, reales y complejos. Estas constantes siguen una notación estándar y se resuelven durante la etapa de compilación del programa. Sus valores son establecidos antes de la ejecución del código.

Las constantes no numéricas abarcan caracteres, cadenas de texto y boolean. En este caso no se evalúan durante la compilación, sino que esta evaluación se realiza durante la ejecución. Entonces los valores de las constantes no numéricas sólo se conocen una vez que el programa ha comenzado a ejecutarse.

**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;
```

Todas son de tipo numerico, por lo tanto, se evalúan y se realiza la ligadura en compilación. (Dado que H e I ya tienen valores asignados en tiempo de compilación, el momento de ligadura de K también es durante la compilación, ya que su valor se puede determinar en ese momento sin necesidad de esperar a la ejecución del 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 alocación de memoria, sacar la declaración (1) y colocar dentro de func1() la declaración static int x =1;**

Si, el resultado sería el mismo. En C las variables globales no cambian la dirección (su L-valor), por lo tanto, en cuanto a alocación de memoria, efectivamente se comportarían igual.

Tener en cuenta que con esa modificación no sería posible llamar directamente a la variable X desde otros lugares del código ya que la opción (1) es de alcance global y la opción (2) de alcance local.

**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
}
```

**Globales:**

- cantTotalPersonas
- nro

**Globales no estaticas:**

*debe crearse una instancia para poder acceder, y su tiempo de vida termina cuando se desaloca la clase

- id
- nombreApellido
- domicilio
- id
- calle
- loc

**Locales:**

- dni
- fechaNac

**Ni idea, eso no compila (por lo menos en Java 21):**

- public int edad
- public string fN

pero creo que ¿locales? están dentro de un método (?

**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.**

**Tiempo de vida:**

- i: 1 - 15
- h: 1 - 15
- mipuntero: 1 - 15

**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 - 14

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

No, no presenta un error.

En la línea 9 guarda un espacio de memoria para el entero
En la línea 10, se guarda en la celda a la que apunta mipuntero el mismo valor de ‘i’, un 3
En la línea 11 se suma el contenido de ‘mipuntero’ + i, o sea 3+3, y se le asigna a h

Al hacer el write en la línea 13, la variable ‘h’ ya tiene un valor, ergo se imprime correctamente

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

Si, presenta un error semantico.
Porque está restando un integer con una variable de tipo puntero.
*En el caso de que la linea 14 fuera 'i:= h- mipuntero**^**;' también presentaría un error, puesto que ya se desalocó de memoria el contenido del puntero con el dispose

**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.**

La entidad es **mipuntero^** (el contenido de la dirección a la que apunta el puntero)

Alcance:

- 4 - 14 *(es siempre igual a la variable puntero automatica)

Tiempo de vida:

- 9 - 12 *(del new al dispose)

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

- **i:** Automática
- **h:** Automática
- **mipuntero:** Automática
- **mipuntero^:** Dinámica

**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**

```
#include <stdio.h>

void funcion() {
    // Variable estática, tiempo de vida mayor al alcance
    // Alcance: todo el programa
    // Tiempo de vida: trasciende al programa
    static int x = 0;
    x++;
    printf("Variable estática: %d\n", x);
}

int main() {
    funcion(); // Imprime Variable estática: 1
    funcion(); // Imprime Variable estática: 2
    funcion(); // Imprime Variable estática: 3
    return 0;
}
```

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

```
#include <stdio.h>
#include <stdlib.h> // Necesario para la función malloc y free

int main() {
    // Paso 1: Asignar memoria dinámicamente a un puntero (para un int)
    int *puntero = (int *)malloc(sizeof(int));

    // Paso 2: Asignar un valor de 10 a *puntero
    *puntero = 10;

    // Paso 3: Liberar la memoria asignada al puntero
    free(puntero);

    // Paso 4: Imprimir el contenido de *puntero (después de liberar la memoria)
    // Nota: Después de liberar la memoria, *puntero ya no está definido y no debería ser
    // accedido. Esto imprime basura. Pero así se demuestra que *puntero
    // ya no tiene un valor válido (pues terminó su tiempo de vida) aunque todavía
    // tiene alcance
    printf("Contenido de *puntero después de liberar la memoria: %d\n", *puntero);

    return 0;
}
```

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

```
#include <stdio.h>
int main() {
    // Declaración de variables
    int num1, num2, suma;

    // Solicitar al usuario que ingrese los números
    printf("Ingrese el primer número: ");
    scanf("%d", &num1);
    printf("Ingrese el segundo número: ");
    scanf("%d", &num2);

    // Calcular la suma
    suma = num1 + num2;

    // Mostrar el resultado
    printf("La suma de %d y %d es: %d\n", num1, num2, suma);

    return 0;
} //Tiempo de vida y alcance: todo el programa :)
```

**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.**

Si, en los tres casos, al no haber procedimientos internos, podemos asegurar que el tiempo de vida y el alcance es igual a todo el procedimiento que la contiene.

En el caso de C y ADA es importante la aclaración del enunciado de que la variable se encuentra definida **al comienzo** del procedimiento (asumo primer línea), ya que si la declaración fuera a la mitad, el alcance y tiempo de vida sería desde la línea siguiente de la declaración hasta el final, siendo inaccesible en la primera mitad. Pascal, al tener una sección especial para declarar variables, la aclaración de al comienzo o a la mitad no modificaría la respuesta.

**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, string de caracteres para referenciar es el nombre

**II) Conjunto de valores que puede tomar y un rango de instrucciones en el que se conoce el nombre.**

Falso, rango de instrucciones donde se conoce es alcance

**III) Conjunto de valores que puede tomar y lugar de memoria asociado con la variable.**

Falso, lugar de memoria asociado es L-valor

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

Verdadero :)

**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 cual es su el tiempo de vida. Indicar para cada variable su r-valor al momento de alocación en memoria**

```
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
```

| IDENT | TIPO | R VALOR | ALCANCE | T.V. |
|---|---|---|---|---|
| a (4) | Automática | Basura | 4-14 | 2-29 |
| Main | | | 3-29 | 2-29 |
| n | Automática | Basura | 5-29 | 2-29 |
| p (4) | Automática | Basura | 5-10 -> 23-29 | 2-29 |
| v1 | Automática | Basura | 6-29 | 2-29 |
| c1 (6) | Automática | 10 | 7-10 -> 23-29 | 2-29 |
| Uno | | | 8-29 | 7-22 |
| v2 | Semi-dinamica | Basura | 10-22 | 7-22 |
| c1 (10) | Automática | Basura | 11-22 | 7-22 |
| c2 | Automática | Basura | 11-22 | 7-22 |
| p (11) | Automática | Nil | 12-22 | 7-22 |
| p^ | Dinámica | Basura | 12-22 | 15-18 |
| q | Automática | Nil | 12-22 | 7-22 |
| q^ | Dinámica | Basura | 12-22 | 16-20 |

**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**

Opción correcta: **(b)** Puede condicionar el alcance de la variable, ya que si hubiera otra variable con el mismo nombre en otro submódulo o bloque, la variable original dejaría de ser conocida mientras ese submódulo esté en ejecución.

**Ejercicio 14:** **Sean los siguientes archivos en C, los cuales se compilan juntos**
**Indicar para cada variable de qué tipo es en cuanto al momento de ligadura de su l-valor.**
**Indicar para cada identificador cuál es su alcance y cual 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 | Automatica | 0 | 2-4 -><br>9-12 -><br>21-23 | 1-28 |
| a | Automatica | Null | 2-16 | 1-16 |
| a^ | Dinamica | Basura | 2-16 | 15-16 |
| fun2 | | | 4-16 | 3-8 |
| v1 | Automatica | Basura | 5-8 | 3-7 |
| y | Automatica | Basura | 5-8 | 3-7 |
| var3 | Estatica | 0 | 11-16 | <1-28> |
| v1 | Automatica | Basura | 13-16 | 9-16 |
| y | Automatica | Basura | 13-16 | 9-16 |
| var1 | Automatica | C | 15-16 | 13-16 |
| aux | Estatica | 0 | 18-28 | <1-28> |
| v2 | Automatica | 0 | 19-28 -><br>7 -><br>12 - 16 | 1-28 |
| fun2 | | | 20-28 | 19-23 |
| fun3 | | | 25-28 | 24-28 |
| aux | Automatica | Basura | 26-28 | 24-28 |

**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**

**Var:**
La sentencia var declara una variable, opcionalmente inicializándola con un valor.

**Let:**
La instrucción let declara una variable de alcance local con ámbito de bloque(blockscope), la cual, opcionalmente, puede ser inicializada con algún valor

**Var vs Let**
Cuando usamos **let** dentro de un bloque, podemos limitar el alcance de la variable a dicho bloque (por ejemplo, dentro de un If o un For). Notemos la diferencia con var, cuyo alcance reside dentro de la función donde ha sido declarada la variable.

**Const**
Las variables constantes presentan un ámbito de bloque y sólo se pueden acceder dentro del mismo tal y como lo hacen las variables definidas usando la instrucción let, con la particularidad de que el valor de una constante no puede cambiarse a través de la reasignación.

**Ausencia de palabra clave**
Si no se usa ninguna de las palabras claves anteriormente mencionadas, el motor de JavaScript va a intentar buscar la variable en un contexto superior de forma recursiva. Esto es, primero busca en el ámbito local, si no se encuentra, lo intenta en su contexto inmediatamente superior, y así hasta que llegue al contexto global. Si no se ha encontrado en ningún contexto, esa variable en se crea automáticamente.
El problema es si esa variable sí que se encuentra en uno de esos contexto superior que se han ido comprobando y no se tuvo en cuenta, podría generar problemas no deseados.

**Comparación con Java:**
En java se debe indicar el tipo de variable si o si.
Si se declara un ‘int n’ dentro de un For, el mismo es accesible sólo dentro de ese bloque, como sucede con el For.
El proceso de buscar recursivamente en la jerarquía superior (method lookup) java lo realiza con métodos de las clases, pero no sucede con variables. En ese caso, Java tiraría una excepción.