# 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:

# Procedure Practica4();

1. **var**
2. a,i:integer
3. p:puntero

# Begin

1. a:=0;
2. new(p);
3. p:= ^i
4. for i:=1 to 9 do **10.**a:=a+i; **11.**end;
5. ...
6. p:= ^a;
7. ...
8. dispose(p); **16.end**;

**RTA {**

Atributos de la variable a.

Nombre: a

Alcance: Estático (es local), de la línea 3 hasta línea 16

Tipo: Predefinido Integer, ligadura estática.

L-Valor: Alocacion dinámica automática. (¿En realidad esta alocacion no ocurre con strings largos o con ciertos arreglos dinámicos?**)** //Mi respuesta 🡪 Alocacion Estática.

R-Valor: Dinámico indefinido //Mi respuesta 🡪 Estático definido y vale 0 línea 6.

**}**

**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?

**RTA {**

Atributos de la variable p.

Nombre: p

Alcance: Estático (es local), 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:

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

**RTA {**

• 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

**}**

1. 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.

**RTA {**

|  |  |
| --- | --- |
| Java | - 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 | - Declaración e inicialización en la misma línea: int num = 10;  - Declaración e inicialización separadas: int num; num = 10;  - Las variables globales y estáticas se inicializan por defecto a 0 si no se les  asigna un valor explícitamente.  - Las variables locales no tienen un valor por defecto y su valor inicial es  impredecible, a menos que se les asigne un valor explícitamente. |
| Python | - Declaración e inicialización en la misma línea: num = 10  - No es necesario declarar el tipo de variable antes de su inicialización |
| Ruby | - Declaración e inicialización en la misma línea: num = 10  - No es necesario declarar el tipo de variable antes de su inicialización |

**}**

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

1. Variable estática.

**RTA {**

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:

**#include <stdio.h>**

**void static\_example() {**

**static int num = 0;**

**printf("El valor de num es: %d\n", num);**

**num++;**

**}**

**int main() {**

**for (int i = 0; i < 5; i++) {**

**static\_example();**

**}**

**return 0;**

**}**

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

La salida del programa sería:

El valor de num es: 0

El valor de num es: 1

El valor de num es: 2

El valor de num es: 3

El valor de num es: 4

//////////////////////////////

Extracto del PowerPoint de la catedra:

Momentos – Alocación de memoria:

Estático: se hace en compilación (antes de la ejecución) cuando se carga el programa en memoria en zona de datos y perdura hasta fin de la ejecución (sensible a la historia).

**}**

1. Variable automática o semiestática.

**RTA {**

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:

**def automatic\_example():**

**num = 0**

**print("El valor de num es: ", num)**

**num += 1**

**for i in range(5):**

**automatic\_example()**

En este ejemplo, la variable num es una variable automática dentro de la función

automatic\_example().

La salida del programa sería:

El valor de num es: 0

El valor de num es: 0

El valor de num es: 0

El valor de num es: 0

El valor de num es: 0

**}**

1. Variable dinámica.

**RTA {**

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:

**def dynamic\_example():**

**num = input("Ingrese un número: ")**

**print("El valor ingresado es:", num)**

**for i in range(5):**

**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.

//////////////////////////////

Extracto del PowerPoint de la catedra:

Momentos – Alocación de memoria:

Dinámico: se hace en tiempo de ejecución.

1. Automática: cuando aparece una declaración de una variable en la ejecución

2. Explícita: requerida por el programador con una sentencia de creación, a través de algún constructor (por ej.. algún puntero)

**}**

1. Variable semidinámica.

**RTA {**

Una variable semidinámica es aquella cuyo tamaño puede cambiar durante la ejecución del

programa, pero solo se puede ajustar su tamaño en ciertos puntos de control predefinidos. Un

ejemplo de una variable semidinámica en ADA podría ser una matriz cuyo tamaño se puede

ajustar en tiempo de ejecución, pero solo en un punto de control específico del programa.

Ejemplo en ADA:

**with Ada.Text\_IO; use Ada.Text\_IO;**

**procedure Ejemplo\_Variable\_Semidinamica is**

**type Matriz is array (Positive range <>, Positive range <>) of Integer;**

**-- Definición de una matriz de tipo semidinámico**

**Filas, Columnas : Positive := 5; -- Tamaño inicial de la matriz**

**M : Matriz (1..Filas, 1..Columnas); -- Inicialización de la matriz**

**begin**

**-- Realizar alguna operación con la matriz inicial**

**Put\_Line("El tamaño actual de la matriz es " & Positive'Image(Filas) & "x" &**

**Positive'Image(Columnas));**

**-- En algún punto de control en el programa, ajustar el tamaño de la matriz**

**Filas := 8;**

**Columnas := 8;**

**M := M (1..Filas, 1..Columnas); -- Ajustar el tamaño de la matriz**

**-- Realizar alguna operación con la matriz actualizada**

**Put\_Line("El tamaño actual de la matriz es " & Positive'Image(Filas) & "x"**

**& Positive'Image(Columnas));**

**end Ejemplo\_Variable\_Semidinamica;**

**}**

De al menos un ejemplo de cada uno.

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

**RTA {**

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

1. Variable estática: son variables que se almacenan en la memoria estática y mantienen su valor entre llamadas a funciones. Estas variables se declaran con la palabra clave "static" y se inicializan automáticamente en cero.

2. Variable automática: son variables que se almacenan en la memoria de la pila y se eliminan automáticamente al salir de su ámbito de definición. Estas variables se declaran sin la palabra clave "static" y su vida útil está limitada a su función o bloque de código.

3. Variable dinámica: son variables que se asignan en tiempo de ejecución y se almacenan en la memoria dinámica. Estas variables se crean utilizando las funciones "malloc" o "calloc" y se liberan utilizando la función "free". Estas variables tienen una vida útil que se extiende más allá de la función o bloque de código en el que se crean.

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

1. Variable estática: se definen usando la palabra clave "constant" o "static" y se mantiene en memoria durante toda la ejecución del programa.

2. Variable automática o semiestática: se definen dentro de un bloque y se eliminan de la memoria cuando el bloque termina.

3. 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".

4. Variable semidinámica: una matriz cuyo tamaño se puede ajustar en tiempo de ejecución, pero solo en un punto de control específico del programa.

**}**

# Ejercicio 4:

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

**RTA {**

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.

**}**

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

**RTA {**

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:

**#include <stdio.h>**

**void ejemploFuncion() {**

**static int variableLocal = 0;**

**variableLocal++;**

**printf("El valor de la variable estática es %d\n", variableLocal);**

**}**

**int main() {**

**ejemploFuncion(); // Imprime "El valor de la variable estática es 1"**

**ejemploFuncion(); // Imprime "El valor de la variable estática es 2"**

**ejemploFuncion(); // Imprime "El valor de la variable estática es 3"**

**return 0;**

**}**

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.

**}**

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

**RTA {**

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.

Ejemplo:

**public class Ejemplo {**

**// Esta variable es "global" al estar accesible desde toda la clase,**

**// pero NO es estática**

**public int contador = 0;**

**// Esta variable es global y también ES estática**

**public static int total = 0;**

**public void incrementar() {**

**contador++; // Afecta al contador de una instancia**

**total++; // Afecta al total de toda la clase**

**}**

**public static void main(String[] args) {**

**Ejemplo e1 = new Ejemplo();**

**Ejemplo e2 = new Ejemplo();**

**e1.incrementar(); // contador = 1, total = 1**

**e2.incrementar(); // contador = 1, total = 2**

**System.out.println("e1.contador: " + e1.contador); // 1**

**System.out.println("e2.contador: " + e2.contador); // 1**

**System.out.println("Ejemplo.total: " + Ejemplo.total); // 2**

**}**

**}**

**}**

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

**RTA {**

### ✅ **Diferencia entre una variable estática respecto de su L-Value y una constante**

| **Característica** | **Variable estática (respecto de su L-Value)** | **Constante** |
| --- | --- | --- |
| **L-Value** (dirección de memoria) | Tiene una **dirección fija** en memoria durante toda la ejecución del programa (es persistente). | También tiene una dirección fija, pero es de **solo lectura**. |
| **Mutabilidad** | ✅ **Puede cambiar su valor** durante la ejecución. | ❌ **No puede modificarse** una vez definida. |
| **Tiempo de vida** | Existe durante **toda la ejecución** del programa. | También existe durante toda la ejecución, pero **su valor no cambia**. |
| **Ejemplo en Java** | static int contador = 0; | final int MAX = 100; |
| **Propósito** | Conservar estado entre llamadas o compartir datos entre instancias. | Definir valores **constantes** que no deben cambiar. |

**}**

# Ejercicio 5:

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

**RTA {**

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.

**}**

1. 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;

**RTA {**

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

**RTA {**

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 seria el mismo.

Aunque en cuanto a visibilidad y semántica no es lo mismo. En variable global seria visible y modificable por cualquier parte dentro del archivo. Mientras que siendo estatica en la func1(), se la encapsula y solo puede ser accedida desde la misma. Evitando en el ultimo caso, posibles errores.

**}**

**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  } |
| **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

# Program Uno;

1. type tpuntero= ^integer;
2. var mipuntero: tpuntero;
3. var i:integer;
4. var h:integer;

# Begin

**7-** i:=3;

1. mipuntero:=nil;
2. new(mipuntero);
3. mipunterno^:=i;
4. h:= mipuntero^+i;
5. dispose(mipuntero);
6. write(h);
7. i:= h- mipuntero;
8. **End**.
   1. Indique el rango de instrucciones que representa el tiempo de vida de las variables i, h y mipuntero.

**RTA {**

* Tiempo de vida de i y h: Desde su declaración hasta el End. final (líneas 4–15 y 5–15, respectivamente).
* Tiempo de vida de mipuntero: Desde la línea 3 hasta la 15, pero su memoria dinámica válida solo entre las líneas 9–12.

**}**

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

**RTA {**

i: línea 4 – línea 15

h: línea 5 – línea 15

mipuntero: línea 3 – línea 15

^mipuntero: línea 3 – línea 15**}**

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

**RTA {**

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.

**}**

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

Justifique

**RTA {**

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 efectuo 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.

**}**

* 1. 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.

**RTA {**

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)

**}**

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

**RTA {**

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

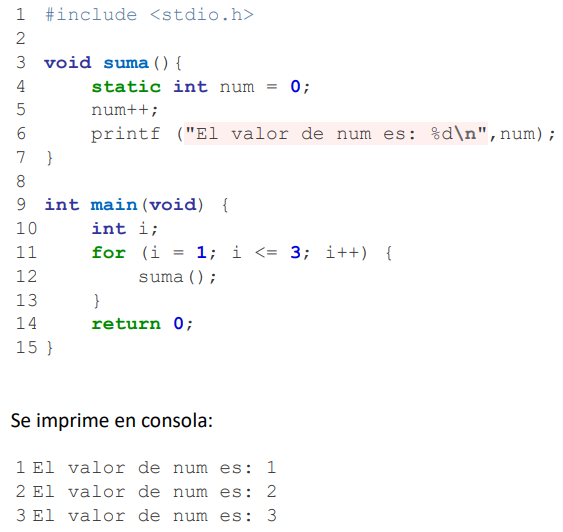
**}**

**Ejercicio 9:** Elija un lenguaje y escriba un ejemplo:

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

**RTA {**

Ejemplo en C:

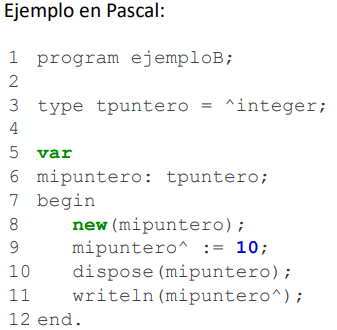


En el anterior programa, el alcance de num es solo dentro de la función suma, pero su tiempo de vida es desde la primera vez que se ejecuta la función suma hasta que finaliza el programa ya que se trata de una variable estática.

**}**

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

**RTA {**

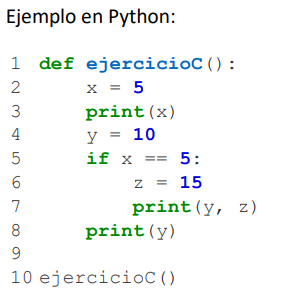
****

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.

**}**

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

**RTA {**

****

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.

**}**

**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.

**RTA {**Sí, en los tres lenguajes se puede asegurar que la variable c tiene alcance y tiempo de vida limitados al procedimiento en el que se encuentra definida, ya que no hay definiciones de procedimientos internos que puedan afectar su alcance o tiempo de vida. **}**

**Ejercicio 11: a)** Responda Verdadero o Falso para cada opción. El tipo de dato de una variable es?

1. Un string de caracteres que se usa para referenciar a la variable y operaciones que se pueden realizar sobre ella. **FALSO**
2. Conjunto de valores que puede tomar y un rango de instrucciones en el que se conoce el nombre. **FALSO**
3. Conjunto de valores que puede tomar y lugar de memoria asociado con la variable. **FALSO**
4. 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.

**RTA {**

• Definición:

Conjunto de valores

Conjunto de las operaciones

• Antes de que una variable pueda ser referenciada debe ligársele un tipo

Protege a las variables de operaciones no permitidas

• Chequeo de tipos: verifica el uso correcto de las variables

Extracto del powerpoint: Tipo: es el tipo de variables definidas, tiene asociadas rango de valores y conjunto de operaciones permitidas**.}**

**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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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** | **Alcan**  **ce** | **T.V.** |
| a (línea  4) | automática | basura | 5-29 | 1-29 |
| n (línea 4) | Automática | Basura | 5-29 | 1-29 |
| p (línea 4) | Automática | Basura | 5-11 23-29 | 1-29 |
| v1 (linea 5) | Automática | Basura | 6-29 | 1-29 |
| c1 (línea 6) | Automática | 10 | 7-10 23-29 | 1-29 |
| v2 (línea 9) | Semidinamica | 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 |
| Uno() (línea 7) |  |  | 8-29 | 7-22 |
| Main() (línea  2) |  |  | 3-29 | 2-29 |
|  | | | | | |

# 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:

1. Su tiempo de vida: **El nombre de una variable no condiciona su tiempo de vida. El tiempo de vida de una variable no depende del nombre que tenga.**
2. Su alcance: **El nombre una variable puede condicionar su alcance de, ya que el alcance de una variable es el rango de instrucciones en el que se conoce el nombre**.
3. Su r-valor: **El nombre de una variable no condiciona su r-valor, ya que el r-valor se determina por la asignación de un valor a la variable (al nombre). Lo que si permite el nombre es poder acceder, mediante ese nombre, a la variable.**
4. Su tipo. **El nombre de una variable no condiciona directamente su tipo, ya que el tipo se define explícitamente en la declaración de la variable. Sin embargo, es importante elegir un nombre descriptivo para la variable que refleje el tipo de dato que almacena, lo que puede hacer que sea más fácil para los programadores entender y trabajar con el código que utiliza la variable.**

Justifique la respuesta

**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. }   1. **main**() 2. {static int var3; 3. extern int v2; 4. int v1, y; 5. for(y=0; y<10; y++) 6. { char var1='C'; 7. a=&v1;} 8. }   **ARCHIVO2.C**   1. static int aux; 2. int v2; 3. **static int fun2**( ) 4. { extern int v1; 5. aux=aux+1; 6. …   23. }   1. **int fun3**( ) 2. { int aux; 3. aux=aux+1; 4. …   28. } |  | | | | | |
|  | **Ident.** | **Tipo** | **r-valor** | **Alcance** | **T.V.** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  | | | | | |

# 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.