# 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identificador | Lvalor | Rvalor | alcance | TV |
| A | Automática | undefined | 4 – 16 | 1 – 16 |
| P | Automática | undefined | 5 – 16 | 1 – 16 |
| ^P | Dinámica | Undefined | 5 – 16 | 7 – 15 |

La P es un puntero por lo que una parte es automática “el puntero como tal” y por otro lado esta lo apuntado por este, esta parte es la dinámica.

b) contiene una dirección a una posición de memoria (a una variable dinámica)

## 2

1. Depende del lenguaje, hay lenguajes en los que hay una sección de declaración y luego una de código, entonces en la sección de declaración únicamente se indica el tipo y nombre de la variable y luego se inicializa en la sección de código como ADA o PASCAL,

Hay otros que dejan declara e inicializar al mismo tiempo ya que no tienen una sección para la declaración, (como C o cualquier derivado de este)

Y algunos otros al declarar, el compilador le da un valor por defecto como C



|  |  |  |  |
| --- | --- | --- | --- |
| C | Java | Python | Ruby |
| Se debe indicar el tipo, nombre y el valor es opcional en variables globales | Java en las variables globales para los datos primitivos las inicializa con un valor y los objetos en null | Se tiene que especificar solo el nombre y el valor asignado, ninguna es opcional | Same as python |

# 3

1. la variable estática se aloca en memoria en tiempo de compilación por lo que su tiempo de vida dura más que el programa, el único lenguaje que posee este tipo es “C”
2. la variable automática se aloca cuando la función, proceso o unidad que la contiene se aloca, la mayoría de las variables funcionan así
3. las variables dinámicas son aquellas que se alocan en la memoria dinámica, el único ejemplo son las celdas de memoria apuntadas (la segunda parte de un puntero) / se alocan en tiempo de ejecución
4. las variables semidinamicos son aquellas que en principio se alocan en compilación (automática) y en tiempo de ejecución se les agrega más memoria de ser necesario (dinámica), el único ejemplo de esto son los arreglos semidinamicos de “ADA”

C

* Automática
* Dinámica
* Estática
* Dudable de acá para abajo
* parámetros de funciones y variables
* strings
* compound / una suerte de constructor
* expresion entre parentesis si lo que esta afuera es un lvalue
* (variable.algo) solo si “variable” es un lvalue
* El resultado de algo accedido a traves de un puntero (con ->)
* El resultado de un operador de subscripcion
* Un lvalue entre parentesis retorna un lvalue
* <https://www.quora.com/What-is-an-Lvalue-in-C>

ADA

* Automática
* Semidinámica
* Dinámica

# 4

1. las variables locales son aquellas declaradas o definidas dentro de la unidad en ejecución, las globales son independientes de las unidades, son aquellas declaradas en el código principal (si lo hay)
2. puede ser local y estática si la variable se define dentro de una unidad y se la declara con la palabra clave “static”, solo en C, de esta manera la variable se alocará en compilación, pero su alcance será limitado a la unidad que la contiene.
3. No, la palabra clave “static” solo modifica en que momento se alocara la variable en memoria, mientras que la palabra clave “global” solo modifica su alcance, por lo que no son dependientes.
4. Las constantes son variables en las que el Rvalor asignado en su definición no variara, mientras que las estáticas son variables alocadas en tiempo de compilación

# 5

1. Las constantes numéricas a diferencia de las normales tienen la capacidad de no necesitar la especificación del tipo y dejan usar el “\_” para separar los “miles” en números así es más legible

const numéricas: se asigna el r-valor en tiempo de compilación (no es estática)

const comunes: se asigna el r-valor en tiempo de ejecución

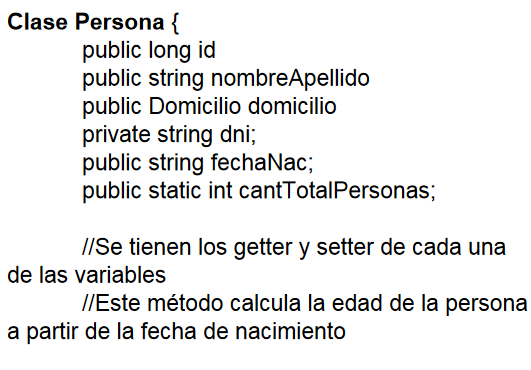
todas las constantes son automáticas

1. 1. Ejecucion(Rvalue)
   2. Compilacion(Rvalue)
   3. Ejecucion(Rvalue)

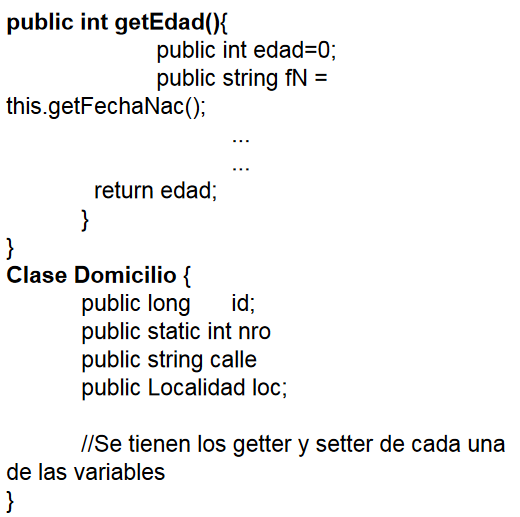
# 6

En cuanto a alocación de memoria, tal y como está (asumiendo que “int x=1;” está en el main), se alocara de manera automática cuando la unidad que la contiene se ejecute, en este caso el main. Si la variable se declara como “static” aunque esto se realiza dentro de una función (haciendo que la variable sea local), la variable se alocara en tiempo de compilación.

# 7

en este caso las variables declaradas en la clase, seran globales (en esa clase), por lo que podran se accedidas desde los metodos declarados en esta clase

otra interpretacion es que las variables “static” son las globales, puesto que a estas se puede acceder desde cualquier parte del programa (mientras se tenga acceso a la clase y siempre y cuando la variable static sea public)

en este caso las variables globales (con respecto a la clase) serian las declaradas en la clase “Domicilio” puesto que seran accesibles desde los metodos (getters y setters)

las variables declaradas en el metodo “getEdad()” serian locales puesto que solo se pueden acceder dentro de ese metodo

si gloabales en java son las static, aca no hay ninguna variable global

# 8

1. mipuntero: 1-12;

^mipuntero : 9-12

i: 1:15

h:1-15

1. mipuntero: 4-15;

^mipuntero: 4-15;

i: 5-15

h:6-15

1. no presenta ningún error ni sintáctico ni semántico, porque al hacer la suma accede al valor apuntado por el puntero y realiza una suma entre enteros
2. presenta un error de tipo (semántica dinámica), puesto que intenta hacer una resta entre un entero y un tipo puntero
3. se refiere al “program uno;” tiene alcance y tiempo de vida todo el programa (1, 15)
4. mipuntero: dinámica y automática

h: automática

i: automática

# 9

Lenguaje usado “C”

1. variables estáticas (de C) puesto que se alocan en compilación y el programa inicia una vez compilado
2. variables dinámicas, el alcance es desde la declaración, pero su tiempo de vida comienza una vec hecho el “new” o instrucción en cuestión que aloque la variable
3. Variables automáticas, ya que se alocan una vez alocada la unidad que las contiene y cuando esta termina se desalocan, y como su alcance también es acotado a la unidad en cuestión estos valores son iguales.

# 10

1. en C no se puede asegurar, puesto que no hay una sección de definición, por lo que se podría hacer de nuevo “int C” y se crearía una nueva variable C.
2. en pascal si podemos asegurarlo, puesto que en el codigo no se puede realizar de nuevo una definicion, solo asignarle un nuevo valor a la variable.
3. En ADA es igual que pascal pero sin el “var” osea, las declaraciones tienen que ir antes del begin, por lo que si, se puede asegurar.

# 11

1. False
2. False
3. False
4. True

Definicion: valores y operaciones que se pueden realizar

# 12

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

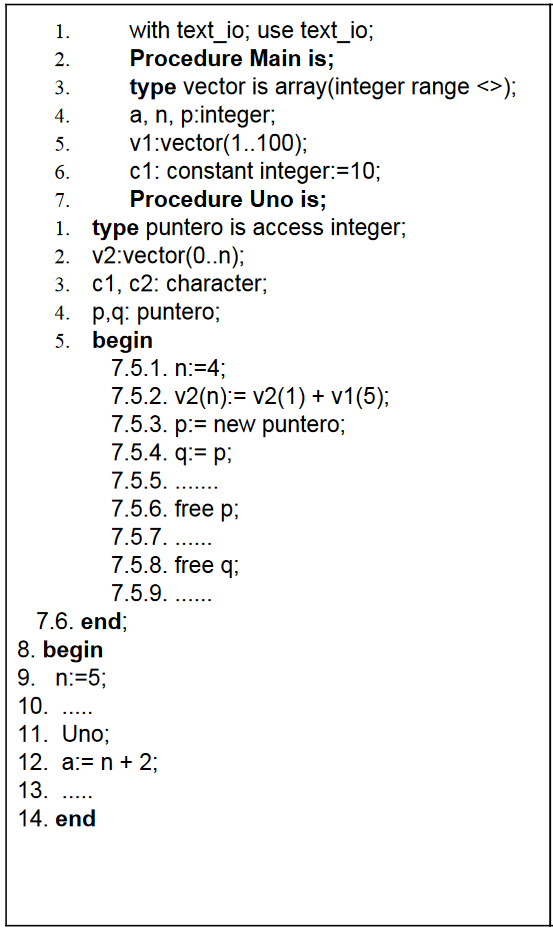
26

27

28

29

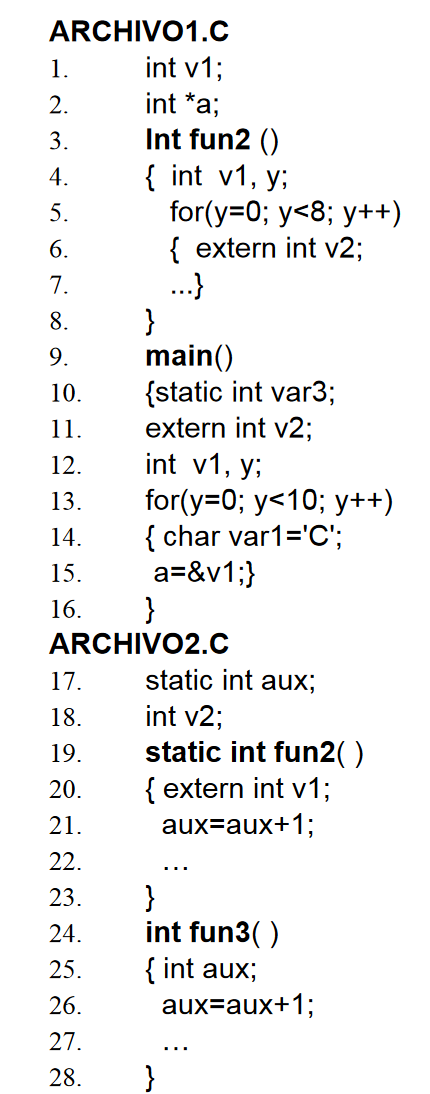
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id | Tipo/lvalor | Rvalor | Alcance | tv |
| Main |  |  | 2,29 | 2,29 |
| a | Automatica | Undefined | 5,29 | 2,29 |
| n | Automatica | Undefined | 5,29 | 2,29 |
| p | Automatica | Undefined | 5,11-23,29 | 2,29 |
| V1 | SemiDinam | Undefined | 6,29 | 2,29 |
| C1 | Automatica | Undefined | 7,10-23,29 | 2,29 |
| Uno |  |  | 7,22 | 7,22 |
| V2 | SemiDinam | Undefined | 10,22 | 7-22 |
| C1-uno() | Automatica | Undefined | 11,22 | 7,22 |
| C2 | Automatica | Undefined | 11,22 | 7,22 |
| p-uno() | Automatica | Undefined | 12,22 | 7,22 |
| ^p | Dinamica | Undefiend | 12,22 | 15,18 |
| q-uno() | Automatica | Undefined | 12,22 | 7,22 |
| ^q | Dinamica | Undefiend | 12,22 | 16,20 |



# 13

1. NO – aunque haya otra variable con el mismo nombre esta primera se desalocara cuando su unidad lo haga
2. SI – si se declara una variable con el mismo nombre más adelante esta perderá alcance en la unidad en donde este declarada la segunda variable
3. NO
4. NO

# 14



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identificador | Tipo | Rvalor | Alcance | TV |
| V1 | Automática | 0 | 2-4; 9-12;21-23 | 1-28 |
| a | Automática | nil | 3-16 | 1-16 |
| ^a | Dinámica | nil | 3-16 | 15-16 |
| Fun2() |  |  | 3-16 | 3-8 |
| V1-fun2() | automática | 0 | 5-8 | 3-8 |
| y-fun2() | Automática | 0 | 5-8 | 3-8 |
| Main() |  |  | 9-16 | 9-16 |
| Var3 | Estática | 0 | 11-16 | <1-16> |
| V1-main() | Automática | 0 | 13-16 | 9-16 |
| y-main() | Automática | 0 | 13-16 | 9-16 |
| Var1 | Automática | “” | 15-15 | 13-15 |
| Aux | Estática | 0 | 18-25 | <17-28> |
| V2 | Automática | 0 | 19-28  7-7  12-16 | 17-28  5-7  9-16 |
| Fun2() |  |  | 19-28 | <17-28> |
| fun3() |  |  | 24-28 | 24-28 |
| Aux-fun3() | Automática | 0 | 26-28 | 24-28 |

# 15

1. **const** hace que la variable sea una constante y tiene un alcance limitado al bloque que la define
2. **var** es una variable automática local o global
3. **let** es una variable con alcance limitado al bloque que la contiene