Variable a, sus atributos son:

* Su nombre: a
* Tipo: integer
* Alcance: 4-16
* l-valor: automatica
* r-valor: 0

Variable P

identificador: P

Tipo: Puntero

Alcance: 5-16

l-valor: Depende ya que al ser un puntero, este representa 2 variables, una del tipo automatica que es la propia variable del programa, y otra dinamica la cual es la referenciada en memoria.

r-valor: undefined



En la mayoría de los lenguajes de programación, las variables pueden inicializarse al declararse de las siguientes maneras:

1. Inicialización explícita con un valor literal: Se asigna un valor constante directamente (por ejemplo, int x = 5;).
2. Inicialización con una expresión: Se usa una operación o cálculo (por ejemplo, int x = 2 + 3;).
3. Inicialización con un constructor o función: Se invoca un constructor o función para asignar un valor (por ejemplo, String s = new String("hola"); en Java).
4. Inicialización por defecto: Si no se especifica un valor, el lenguaje puede asignar un valor predeterminado (por ejemplo, 0 para enteros en C con variables estáticas/globales).
5. Inicialización con otra variable: El valor inicial proviene de otra variable ya declarada (por ejemplo, int y = x;).
6. Inicialización dinámica o implícita: En lenguajes con tipado dinámico, el tipo y valor se infieren del contexto (por ejemplo, x = 5 en Python).

Estas formas varían según el lenguaje, su tipado (estático o dinámico) y las reglas de ámbito o almacenamiento.

b)

| **Característica** | **Java** | **C** | **Python** | **Ruby** |
| --- | --- | --- | --- | --- |
| **Tipado** | Estático, fuerte | Estático, débil | Dinámico, fuerte | Dinámico, fuerte |
| **Explícita con literal** | Sí (int x = 5;) | Sí (int x = 5;) | Sí (x = 5) | Sí (x = 5) |
| **Con expresión** | Sí (int x = 2 + 3;) | Sí (int x = 2 + 3;) | Sí (x = 2 + 3) | Sí (x = 2 + 3) |
| **Con constructor** | Sí (String s = new String();) | No | Sí (lista = list()) | Sí (array = Array.new) |
| **Por defecto** | Sí (campos: 0, null, etc.) | Sí (global/estática: 0) | No (no existe hasta asignar) | Sí (nil si no asignada) |
| **Con otra variable** | Sí (int y = x;) | Sí (int y = x;) | Sí (y = x) | Sí (y = x) |
| **Inicialización especial** | No | Sí (arreglos: {1, 2, 3}) | No | No |
| **Requiere inicialización** | Sí (locales) | No (basura si no se inicializa) | No (se crea al asignar) | No (se crea al asignar) |

1. 1. Una variable estatica, solo existen en el lenguaje C y son aquellas las cuales se alocan en el área de Codigo en vez de en el área de datos con el resto de las variables. Su tiempo de vida es de antes de que se aloque el bloque que las contiene, y termina despues.
      1. Static int a;
   2. Una variable automatica, es aquella que se aloca cuando se aloca la unidad que la contiene.
      1. int x;
   3. Una variable dinamica son la variable en memoria, que apunta a la dirección de memoria indicada.
      1. p = ^x;
   4. Las variables semidinamicas son aquellas que solo existen para los arreglos en ADA
2. 1. Una variable local es aquella que solo puede ser referenciada en un bloque en particular (ej una funcion) ya que fue definida dentro de este.

Mientras que una variable global es aquella que fue definida en el codigo principal y su alcance es desde la instruccion siguiente a su definicion hasta el final del programa mientras no se la enmascare.

* 1. Yo creo que si, ya que que sea local no significa que por cada llamada al bloque que la contiene un l-valor vaya a cambiar.

Ej:

Si yo tengo una funcion  
 int fun1(){

int x = 1;

print(x);

}

Independientemente de que yo llame varias veces a la funcion fun1, el espacio alocado para x, sera el mismo para todas las llamadas hasta que este se desaloque.

* 1. No, ya que estatica es un termino que utilizamos exclusivamente para describir como se comportan las variables en C. Una variable global en cualquier otro lenguaje sera Automatica
  2. La diferencia entre una variable estatica y una constante, es que en una variable estatica lo que no cambia es su L-Valor, mientras que en una constante lo que se mantiene siempre igual es su R-Valor.



### 1. Constantes Numéricas

Características:

* Son evaluadas en tiempo de compilación.
* No ocupan memoria, ya que el compilador sustituye su valor directamente en el código.
* Solo se pueden usar para tipos numéricos (Integer, Float, etc.).

### 2. Constantes Comunes

Características:

* Se almacenan en memoria en tiempo de ejecución.
* Se pueden usar con cualquier tipo de dato, como String, Record, etc.
* Se comportan más como variables de solo lectura.

Diferencia clave:

* Constantes numéricas → No ocupan memoria y se resuelven en compilación.
* Constantes comunes → Se almacenan en memoria en ejecución.
  1. H: Su binding time sera en tiempo de compilación
  2. I: Se hara el binding en tiempo de Compilacion. Ya que a pesar de no ser definida como constant Float, el compilador interpreta que el valor definido es un integer.
  3. K: Se hara el binding en compilacion

1. La diferencia sera, que la variable X, en vez de ser tomada como una variable automatica y alocada junto con el resto de las variables y funciones en el área de datos, sera alocada durante la compilación en el área de codigo.
2. En java, tecnicamente no existen las variables globales como si lo hacen en lenguajes como C.

Las variables definidas dentro del metodo calcularEdad(), son locales al metodo. Mientras que las variables de instancia de las Clases, serian locales a la clase pero al estar definidas como publicas, son accesibles por fuera del codigo, pero no por su identificador, si no que por medio de una referencia a la propia clase. Por ende no son realmente globables ya que con el identificador de la variable solo, no podrian ser invocadas.

1. 1. i,h:1-15

miPuntero: 1-12

* 1. 1. I: 5-15
     2. h: 6-15
     3. miPuntero:4-12
  2. No, no deberia presentar ningun error
  3. En la instruccion 13, no deberia tener ningun error, ahora en la instruccion siguiente a esta si lo hay ya que intenta hacer i:= h- mipuntero; lo cual es erroneo ya que mipuntero es un puntero a un valor, no el valor en si por lo que daria error.
  4. Si, esta otra entidad seria mipuntero^, la cual es la variable dinamica alocada en memoria que apunta a la posicion de memoria. su tiempo de vida es de 10-12 y su alcance es de 4 a 15
  5. i, h y mipuntero son automaticas, mientras que mipuntero^ es dinamica.

1. En Pascal
2. Program prueba

int x;

…

5int x := 1

End(linea 10);

Su alcance sin enmascaramiento sera de 3, a 5

mientras que su tiempo de vida sera de 1 a 10.

1. Un ejemplo de esto son los punteros ya que una vez hecho el dispose, termina su tiempo de vida pero si se los declara como globales, su alcance continua hasta el final del programa.

program TiempoVidaVsAlcance;

type

PEntero = ^Integer;

var

ptr: PEntero; // Puntero global (alcance amplio)

procedure ejemplo;

begin

new(ptr); // Se asigna memoria dinámica

ptr^ := 42; // Se almacena un valor

dispose(ptr); // Se libera la memoria (fin del tiempo de vida)

// Aquí ptr todavía es accesible pero apunta a memoria inválida

writeln(ptr^); // ERROR: Acceso a memoria liberada (comportamiento indefinido)

end;

begin

ejemplo;

end.

1. Esto sucede con las variables definidas dentro de funciones, ya que su tiempo de vida inicia cuando se aloca la funcion, y su alcance es dentro de esta misma

program VidaIgualAlcance;

procedure ejemplo;

var

x: integer;

begin

x := 5; // Se usa dentro de su ámbito

writeln('Valor de x:', x);

end;

begin

end.

10)

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 enmascararian y perderian parte de su alcance. Exceptuando Pascal ya que no se puede declarar variables fuera de funciones.

En pascal el tiempo de vida sera todo el bloque de codigo, mientras que en C y en ADA, si se declarase dentro de bloques declare(ada) o un bloque de codo independiente(C) el tiempo de vida seria lo que viva este bloque.

11)

I) F

II)F

III)F

IV)V

b) El tipo de dato de una variable es la característica que define qué tipo de valores puede almacenar una variable y qué operaciones se pueden realizar sobre ella.

12)

| IDENT. | TIPO | R-VALOR | ALCANCE | TV |
| --- | --- | --- | --- | --- |
| a(linea 4) | automatica | - | 5-14 | 1-14 |
| n (linea 4) | automatica | - | 5-14 | 1-14 |
| p (linea 4) | automatica | - | 5-11/8-14 | 1/14 |
| v1 (linea5) | SemiDinamica | - | 6-14 | 1-14 |
| c1 (linea6) | automatica | 10 | 7-10 / 8-14 | 1-14 |
| v2 (linea 2) | automatica | - | 3-7.6 | 7-7.6 |
| c1 (linea 3) | automatica | basura | 4-7.6 | 1(8)-7.6 |
| c2 (linea 3) | automatica | basura | 4-7.6 | 1(8)-7.6 |
| p(linea 4) | un p automatico, el otro dinamica | nil | 5-7.5.6 | (P)7.1-7.6  (p^) 7.5.3-7.5.6 |
| q(linea 4) | un q automatico, el otro dinamica | nil | 5-7.5.8 | (q)7.1-7.6  (q^)7.5.4-7.5.8 |

13)

1. Depende, ya que si existiese un lenguaje que defina reglas de nombrado de sus lenguajes(Ejemplo Constantes Todo en mayusculas) y tuviese reglas de alocacion para los distintos tipos de variables. En general no sucede.
2. Si, ya que si se definen dos variables con el mismo nombre, la definida más tarde enmascarará la definida primero, acortando su alcance.
3. No, no tendria porque tener influencia sobre lo almacenado en ella.
4. Podría considerarse que si ya que por ejemplo en fortran con un prefijo, asume que es de un tipo determinado.

14)

| ident | tipo | r-valor | alcance | tv |
| --- | --- | --- | --- | --- |
| v1 | automatica | 0 | 2-4/9-12/21-23 | 1-28 |
| \*a | Representa dos variables una automatica y otra dinamica | Undefined | 3-16 | 1-16 |
| v1 (linea 3) | automatica | 0 | 5-8 | 3-8 |
| y | automatica | 0 | 5-8 | 3-8 |
| var3 | estatica | 0 | 11-16 | <1-16> |
| v1(linea 12) | Automatica | 0 | 13-16 | 9-16 |
| y(linea 12) | automatica | 0 | 13-16 | 9-16 |
| var1 | automatica | basura | 15-16 | 9-16 |
| aux | estatica | 0 | 18-25/28 | <17-28> |
| v2 | automatica | 0 | 7-8/12-16/19-28 | 1-28 |
| aux(linea25) | automatica | 0 | 26-28 | 24-28 |
| fun2(Linea3) | - | - | 4-16 | 3-8 |
| fun2(linea19) | - | - | 20-28 | 19-23 |
| fun3(linea24) | - | - | 24-28 | 24-28 |