Práctica 4

1)

a) Nombre: a.
Alcance: desde línea 4 a 16, local al procedimiento.
Tipo: integer/entero. Ligado estáticamente.
L-Valor: automático (dirección de memoria en la que se almacena a).
R-Valor: dinámico, comienza indefinidamente, luego en la línea 6 se le asigna el valor 0, etc.

b) Variable p:
- Nombre: p.
- Alcance: desde línea 5 a 16.
- Tipo: puntero, luego ^p es de tipo integer.
- L-Valor: p automático y ^p dinámico.
- R-Valor: p indefinido y ^p nil.

Variable a mencionada en el inciso anterior.

2)

a)
- Inicializar por defecto.
- Inicializar explícitamente.
- No inicializar.

b)

Modo de inicialización	Python	Java	C	Ruby
Por defecto	---	int x; //queda en 0	int glob; //solo funciona con variables globales, queda en 0.	---
En la misma línea	x = 0;	int x = 0;	int x = 0;	x = 0;
Declaración e inicialización	---	int x; x = 0;	int x; x = 0;	---

3)

a) Variable Estática: La memoria se reserva **una sola vez**, **en tiempo de compilación** o al iniciar el programa, y se mantiene durante **toda la ejecución del programa**.
Ej en C: int x = 10; // variable global: estática

b) Variable Automática o Semiestática: La memoria se reserva **en tiempo de ejecución**, al **entrar a un bloque o función**, y se libera automáticamente al salir.

Ej en C: void func() {
int x = 5; // variable automática
}

c) Variable Dinámica: La memoria se reserva **en tiempo de ejecución** mediante llamadas explícitas como `malloc` en C o `new` en Pascal o Ada.
Ej en C: int* p = malloc(sizeof(int)); // variable dinámica
*p = 10;

d) Variable Semidinámica: Se refiere a estructuras como **arrays dinámicos** o estructuras cuya **dimensión se define en tiempo de ejecución**, pero el almacenamiento puede estar en la pila o el heap, dependiendo del lenguaje.
Ej en C: void func(int n) {
int arr[n]; // Variable Length Array (VLA): semidinámica
}

Tipo	C	Ada
Estática	Variables globales, `static` locales	Variables globales, o con `pragma`
Automática	Variables locales en funciones	Variables declaradas en procedimientos
Dinámica	Asignadas con `malloc`, `calloc`	Asignadas con `new`, tipos `access`
Semidinámica	Arreglos con tamaño definido en tiempo de ejecución (VLA)	Arreglos con rango definido dinámicamente

4)

a) Variable Local: Es una variable **declarada dentro de una función, procedimiento o bloque**, y **solo se puede usar dentro de ese bloque**.
Variable Global: Es una variable **declarada fuera de todas las funciones o procedimientos**, y **puede ser accedida desde cualquier parte del programa** (salvo que haya restricciones).

b) Si, por ejemplo en C podemos definir una variable de tipo static para que mantenga su l-valor entre ejecuciones.
Ej en C:
```
void contadorLlamadas() {
    static int veces = 0;  // Local pero estática
    veces++;
    printf("Esta función fue llamada %d veces\n", veces);
}
```

c) No necesariamente, la propiedad “estática” es independiente del alcance de la variable. En algunos lenguajes, como Python o Java, las variables globales pueden cambiar su l-valor durante la ejecución del programa, mientras que en otros lenguajes, como C, las variables globales son por defecto estáticas. Por ende, depende del lenguaje.

d) Ninguna puede cambiar su l-valor, pero las variables estáticas si pueden modificarse durante la ejecución del programa, mientras que las constantes no.

5)

a) Constantes numéricas: Son **valores literales numéricos** que **no tienen un tipo explícito al momento de su escritura**, y que **pueden adoptar el tipo que se necesite** en una expresión determinada.
   Constantes comunes: Son **constantes declaradas explícitamente** con un **tipo fijo** y un **valor constante asignado**, definido por el programador. Pueden ser caracteres o cadenas.
b) El momento de ligadura de Ada se realiza en tiempo de compilación. La primera constante es ligada cuando se asigna 3,5. La segunda cuando se le asigna 2 y la última cuando se calcula la multiplicación entre ambas constantes.

6) Es igual, ya que en C las variables globales se declaran por defecto estáticas, por lo que se comporta de la misma manera que una variable static, ninguna cambiaría su l-valor durante la ejecución del programa.

7) Clase Persona:
   - public long id : Local
   - public string nombreApellido : Local
   - public Domicilio domicilio : Local
   - private string dni; : Local
   - public string fechaNac; : Local
   - public static int cantTotalPersonas; : Global

   Método getEdad():
   - public int edad=0; : Local
   - public string fN = this.getFechaNac(); : Local

   Clase Domicilio:
   - public long id; : Local
   - public static int nro : Global
   - public string calle : Local
   - public Localidad loc; : Local

8)
   a) El tiempo de vida de las tres variables va desde la línea 1 a la 15
   b) Para mipuntero de 4-15, para i de 5-15 y para h de 6-15
   c) No, se realiza un new del puntero y una asignación antes de usar el valor para la operación, por lo que queda h:= i+i.
   d) Si, se realiza un dispose del puntero antes de la operación, por lo que el puntero queda en nil.
   e) Si, la otra entidad que debe ligarse es el programa principal. En este caso, por ser Pascal, el tiempo de vida va desde la declaración, hasta el final del programa.
   f) i y h son de tipo entero, mientras que mipuntero es de tipo puntero (a entero)

9) d
   a) Ej en C:

```
#include <stdio.h>
void contadorLlamadas() {
    static int veces = 0;  // Local pero estática
```

```
    veces++;
    printf("Esta función fue llamada %d veces\n", veces);
}

int main() {
    contadorLlamadas(); // 1
    contadorLlamadas(); // 2
    contadorLlamadas(); // 3
    return 0;
}
```

En este caso el alcance de la variable es el método pero el tiempo de vida va desde que se ejecuta por primera vez el método hasta que termina el programa.

b) Ej en Pascal

```
program EjB;
type tpuntero = ^integer
var
mipuntero: tpuntero;
begin
    new(mipuntero);
    ^mipunterpo:= 10;
    dispose(mipuntero);
    writeln(mipuntero^);
end;
```

En este caso el alcance va desde que es declarada hasta que termina el programa, mientras que el tiempo de vida solo desde que se hace el new hasta que se realiza el dispose.

c) Ej en Python

```
def ejercicioC():
    x = 5
    print(x)
    y = 10
    if x == 5:
        z = 15
        print(y, z)
    print(y)
ejercicioC()
```

En este caso las variables tienen el alcance limitado por la función ejercicicioC() y el tiempo de vida coincide con el del método.

10) Sí, en los tres lenguajes se puede asegurar que el tiempo de vida y el alcance están limitados por el procedimiento en el que se encuentran definidas, ya que no hay ningún otro procedimiento interno que intervenga.

11)

a)

- F
- F

- F
- V

b) El tipo de dato define qué valores puede tomar la variable y qué operaciones puede realizar. Para poder referenciarla primero debe ligarse un tipo, además puede chequearse el correcto uso a través del chequeo de tipos.

12)

Nombre	Tipo	R-Valor	Alcance	Tiempo de Vida
a	automática(integer)	basura	5-14	1-14
n	automática(integer)	basura	5-14	1-14
p	automática(integer)	basura	5-14	1-14
v1	automática	basura	6-14	1-14
c1	automática	10	7-7.2 y 8-14	1-14
v2	semidinámica	basura	3-7.6	7-7.6
c1	automática	basura	4-7.6	7-7.6
c2	automática	basura	4-7.6	7-7.6
p	automática	nil	5-7.6	7-7.6
q	automática	nil	5-7.6	7-7.6
p^	dinámica	basura	7-7.6	7.5.4-7.5.6
q^	dinámica	basura	7-7.6	7.5.5-7.5.8

13) El nombre de una variable no condiciona su tiempo de vida, ni su tipo, ni su r-valor. Podría llegar a condicionar su alcance, ya que el alcance es el rango de instrucciones en el que se conoce el nombre y puede llegar a entrar en conflicto con el nombre de otras variables.

14)

Nombre	Tipo	R-Valor	Alcance	Tiempo de Vida
v1	automática	0	2-4 ; 9-12 ; 21-23	1-28
a	automática	null	3-16	1-28
*a	dinámica	basura	3-16	1-28
v1	automática	0	5-8	3-8
y	automática	0	5-8	3-8
var3	estática	0	11-16	1-28

v1	automática	0	13-16	13-16
y	automática	0	13-16	9-16
var1	automática	C	15	9-15
aux	estática	0	18-28	1-28
v2	automática	0	7 ; 12-16 ; 19-28	1-28
aux	automática	0	26-28	24-28

15)

- **var:** alcance global o de función (no por bloque), se puede redeclarar y se puede reasignar.
  Ej en Javascript:
  ```
  function test() {
    if (true) {
      var x = 10;
    }
    console.log(x); // 10, ¡aunque está fuera del bloque!
  }
  ```

- **let:** alcance de bloque ({}), no puede redeclararse en su ámbito pero sí puede reasignarse.
  Ej en Javascript:
  ```
  if (true) {
    let x = 10;
  }
  console.log(x); // ReferenceError: x is not defined
  ```

- **const:** alcance de bloque, no puede ni redeclararse y reasignarse.
  Ej en Javascript:
  ```
  const x = 5;
  x = 10; // Error: Assignment to constant variable
  const obj = { a: 1 };
  obj.a = 2; // Válido, el objeto es mutable
  ```

- **sin modificador:** alcance global dentro de una función
  Ej en Javascript:
  ```
  function test() {
    x = 100; // Se vuelve global
  }
  test();
  console.log(x); // 100, ¡aunque nunca la declaramos!
  ```

Característica	`var` / `let` / `const` (JS)	Python
Declaración obligatoria	✅ Sí, excepto sin modificador	✅ Sí, se debe declarar antes de usar
Alcance por bloque	`let` / `const` = sí, `var` = no	❌ No, el alcance es por función
Reasignación	`var` / `let` = sí, `const` = no	✅ Sí (todas las variables son reasignables)
Inmutabilidad	`const` previene reasignación (no objetos)	❌ No existe `const`, solo convención (`ALL_CAPS`)
Hoisting	`var` sí, `let` / `const` con restricciones	❌ No existe hoisting en Python