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;
```

b) Compare los atributos de la variable del punto a) con los atributos de la variable de la línea 4. Que dato contiene esta variable?,

Ejercicio 2:

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

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

Ejercicio 4:

- **a.** ¿A qué se denomina variable local y a qué se denomina variable global?
- **b.** ¿Una variable local puede ser estática respecto de su l-valor? En caso afirmativo dé un ejemplo
- **c.** Una variable global ¿siempre es estática? Justifique la respuesta.
- **d.** Indique qué diferencia hay entre una variable estática respecto de su l-valor y una constante

Ejercicio 5:

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

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;
```

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**;

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
}
```

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.
b) Indique el rango de instrucciones que representa el alcance de las variables i, h y mipuntero.
c) Indique si el programa anterior presenta un error al intentar escribir el valor de h. Justifique
d) Indique si el programa anterior presenta un error al intentar asignar a i la resta de h con mipuntero. Justifique
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.
f) Especifique el tipo de variable de acuerdo a la ligadura con el l-valor de las variables que encontró en el ejercicio.

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

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.

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.
II) Conjunto de valores que puede tomar y un rango de instrucciones en el que se conoce el nombre.
III) Conjunto de valores que puede tomar y lugar de memoria asociado con la variable.
IV) Conjunto de valores que puede tomar y conjunto de operaciones que se pueden realizar sobre esos valores.

b) Escriba la definición correcta de tipo de dato de una variable.

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;
1.   type puntero is access integer;
2.   v2:vector(0..n);
3.   c1, c2: character;
4.   p,q: puntero;
5.   begin
        7.5.1. n:=4;
        7.5.2. v2(n):= v2(1) + v1(5);
        7.5.3. p:= new puntero;
        7.5.4. q:= p;
        7.5.5. .......
        7.5.6. free p;
        7.5.7. ......
        7.5.8. free q;
        7.5.9. ......
  7.6. end;
8. begin
9.   n:=5;
10.  .....
11.  Uno;
12.  a:= n + 2;
13.  .....
14. end
```

Ident.	Tipo	r-valor	Alcance	T.V.
a (linea 4)	automática	basura	4-14	1-14

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:

a) Su tiempo de vida.
b) Su alcance.
c) Su r-valor.
d) Su tipo.

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 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.

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.

Ejercicio 1

Inciso a)

variable a	
Nombre	a
Alcance	4-16
Tipo	integer
L-Valor	Automático
R-Valor	Indefinido
Tiempo de Vida	1-16

Inciso b)

variable a	variable p
Nombre: a	**Nombre:** p
Alcance: 4-16	**Alcance: p** 5-16 - **p^** 5-16
Tipo: integer	**Tipo:** puntero
L-Valor: Automático	**L-Valor: p** Automático - **p^** Dinámico
R-Valor: Indefinido	**R-Valor: p** Nil - **p^** Indefinido
Tiempo de Vida: 1-16	**Tiempo de Vida: p** 1-16 - **p^** 7-15

Ejercicio 2

Inciso a)

- **Inicialización por defecto:** Las variables se inicializan con un valor por defecto, por ejemplo los enteros en 0, los caracteres en blanco, etc.
- **Inicialización en la declaración:** Las variables pueden inicializarse en el mismo momento que se declaran, por ejemplo “int i = 0;”.
- **Ignorar el problema:** La variable toma como valor inicial lo que hay en memoria (la cadena de bits asociados al área de almacenamiento). Puede llevar a errores y requiere chequeos adicionales.

Inciso b)

Lenguaje	Inicialización por defecto	Inicialización por declaración	Ignorar el Problema
Java	SI	SI	NO
C	SI (sólo para variables globales y estáticas)	SI	SI
Python	NO	SI	NO
Ruby	NO	SI	NO

Ejercicio 3

Variable	Descripción	Ejemplo en C	Ejemplo en Ada
Variable Estática	Esta definición se analiza desde la perspectiva del lenguaje C. Son variables que se clasifican con la palabra clave “static” y estas se alocan en memoria en tiempo de Compilación.	static int static_var = 10;	NO POSEE
Variable Automática	Son variables que se alocan en memoria cuando se aloca la unidad que las contiene. Estas son las más comunes como por ejemplo las variables locales a un programa o a un método.	int auto_var = 20;	Auto_Var : Integer := 20;
Variable Dinámica	Estas variables son los Punteros, estos contienen 2 variables, el Puntero en sí mismo que es una Variable	int *dynamic_var;	type Int_Ptr is access Integer; Dynamic_Variable : Int_Ptr := new Integer'(20);

	Automática, y la variable que contiene el Puntero que cuando se hace el “new()” sobre el mismo, la variable que contiene se aloca en la Heap y se vuelve Dinámica.		
Variable Semidinámica	Son variables cuyo tamaño puede ser dinámico, pero su dirección de memoria es estática. Esta definición se analiza desde la perspectiva del lenguaje Ada, en particular viendo los arreglos semidinámicos.	NO POSEE	Array : Array(1..n) of Integer;

Ejercicio 4

Inciso a)

- **Variable Local:** Son todas las referencias a variables que se han creado dentro de una unidad (programa o subprograma).
- **Variable Global:** Son todas las referencias a variables creadas en el programa principal.

Inciso b)

- Si, una variable local puede ser estática con respecto de su l-valor, ejemplo en C:

```c
#include <stdio.h>

void example_function() {
    static int static_local_var = 0; // Variable local estática
    static_local_var++; // Incrementamos su valor en cada llamada
    printf("Valor de la variable estática local: %d\n", static_local_var);
}

int main() {
    example_function(); // Llamada 1 "Valor de la variable estática local: 1"
    example_function(); // Llamada 2 "Valor de la variable estática local: 2"
    example_function(); // Llamada 3 "Valor de la variable estática local: 3"
    return 0;
}
```

Inciso c)

- No, una variable global puede ser o no estática, el alcance de las mismas no condiciona el momento en el que se alocan en memoria, por ejemplo, en C podemos tener variables globales estáticas mediante el uso de la palabra clave “static” o podemos tener variables globales no estáticas mediante el uso de la palabra clave “extern”.

Inciso d)

Variable Estática	Constante
L-Valor: El l-valor de una variable estática es la dirección de memoria donde se almacena su valor.	**L-Valor:** Las constantes no tienen un l-valor en el sentido tradicional. El l-valor se refiere a la dirección de memoria de una variable, pero las constantes no ocupan una posición en la memoria de la misma manera que las variables.
Comportamiento: Una variable estática conserva su valor entre las llamadas a la función en la que se declara. Esto significa que su valor persiste a lo largo de la ejecución del programa, pero su valor puede modificarse mediante operaciones de asignación dentro de la función. La variable estática se inicializa solo una vez durante toda la ejecución del programa.	**Comportamiento:** Una constante tiene un valor que no puede cambiar durante la ejecución del programa. Se define mediante la palabra clave “const”, y una vez que se le asigna un valor, ese valor no puede ser modificado posteriormente. Las constantes se utilizan para representar valores que son fijos y conocidos en tiempo de compilación.
En términos de **l-valor**, mientras que una variable estática tiene una dirección de memoria asociada donde se almacena su valor y que puede ser modificada, una constante no tiene una dirección de memoria explícita, ya que su valor es tratado directamente como un valor literal durante la compilación.	

En términos de **comportamiento**, una variable estática se utiliza cuando se necesita preservar su valor entre las llamadas a una función, pero aún así permitir que su valor cambie dentro del contexto de la función. Por otro lado, una constante se utiliza cuando se necesita un valor que no cambie en absoluto durante la ejecución del programa y que sea conocido en tiempo de compilación.

Ejercicio 5

Inciso a)

Constante Numérica	Constante Definida
Las constantes numéricas son aquellas que representan valores numéricos, ya sean enteros, reales o de otros tipos numéricos definidos por el usuario. Estas constantes se utilizan para asignar valores fijos a variables numéricas y para realizar cálculos matemáticos.	Las constantes comunes, también conocidas como constantes definidas, son aquellas que representan valores que no son necesariamente numéricos. Estas constantes se utilizan para asignar valores fijos a variables de tipos no numéricos, como cadenas de caracteres, tipos enumerados y otros tipos definidos por el usuario.
La clasificación de las constantes en numéricas y comunes en Ada ayuda a organizar y entender mejor el código, ya que refleja la naturaleza de los valores que representan y cómo se utilizan en el programa. Las constantes numéricas se utilizan principalmente para valores numéricos, mientras que las constantes comunes se utilizan para valores de otros tipos.	

Inciso b)

- La constante **H** se liga en el momento en que se encuentra su declaración, es decir, cuando se le asigna el valor **3.5**.
- La constante **I** también se liga en el momento de su declaración, cuando se le asigna el valor **2.**
- La constante **K** se liga después de que **H** e **I** ya hayan sido ligadas. En este caso, H y I ya han sido ligadas, por lo que se puede calcular el valor de K como H * Float(I) (3.5 * 2), que resulta en **7.0.** Por lo tanto, el momento de ligadura de K ocurre después de que H e I hayan sido ligadas, y su valor se calcula en base a esas constantes ya ligadas.

Ejercicio 6

- En C las variables globales tienen comportamiento estático en cuanto a su l-valor, por lo tanto, sería lo mismo declarar la variable **x** en el programa **main** que dentro de **func1.**

Ejercicio 7

Identificadores Globales	Identificadores Locales
public static int cantTotalPersonas; public static int nro;	public long id; (Clase Persona) public string nombreApellido; public Domicilio domicilio; private string dni; public string fechaNac; public long id; (Clase Domicilio) public string calle; public Localidad loc;

Ejercicio 8

Inciso a)

- **Tiempo de Vida de “i”:** 1-15.
- **Tiempo de Vida de “h”:** 1-15
- **Tiempo de Vida de “mipuntero”:** 1-15.
- **Tiempo de Vida de “mipuntero^”:** 9-12.

Inciso b)

- **Alcance de “i”:** 5-15.
- **Alcance de “h”:** 6-15.
- **Alcance de “mipuntero”:** 4-15.
- **Alcance de “mipuntero^”:** 4-15.

Inciso c)

- No, el programa no presenta error al querer imprimir el valor de “h” ya que se le asigna la suma del valor de la variable a la que apunta el puntero (“i”) antes de que se le haga dispose con el valor de la variable “i”.

Inciso d)

- Si, el programa generaría error ya que “h” tiene como valor nil porque se realizó un dispose, y no es una operación válida hacer una resta entre un entero y nil.

Inciso e)

- Si, la entidad faltante es el programa principal, este 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 6 –15.

Inciso f)

- **Variable “i”:** Automática, Integer.
- **Variable “h”:** Automática, Integer.
- **Variable “mipuntero”:** Automática, Puntero.
- **Variable “mipuntero^”:** Dinámica, Integer.

Ejercicio 9

Inciso a)

```c
#include <stdio.h>

void func() {
    static int x = 0; // initialized only once, and retains its value between function calls
    x++;
    printf("Inside func(), x = %d\n", x);
}

int main() {
    func();
    func();
    func();

    return 0;
}
```

Inciso b)

```
program ejemploB;
type intpuntero = ^integer;
var
mipuntero: intpuntero;
begin
 new(mipuntero);
 mipuntero^ := 10;
 dispose(mipuntero);
 writeln(mipuntero^); //El valor de mipuntero^ ya no es accesible
end.
```

Inciso c)

```python
def ejemplo():
    identificador = "Este es un identificador con vida igual a su alcance"
    print(identificador)

# Intentar acceder al identificador fuera de la función causará un error
# print(identificador)  # Esto causará un NameError
```

Ejercicio 10

- Si, se puede asegurar que el tiempo de vida y el alcance de la variable “c” es siempre todo el procedimiento en el que se encuentra definida ya que no existen subprocesos internos que puedan modificar el alcance o el tiempo de vida de la misma.

Ejercicio 11

Inciso a)

- La respuesta verdadera es la **IV)** las demás son falsas.

Inciso b)

- **Tipo de dato de una variable:**
  - Conjunto de valores que se le pueden asociar a una variable junto con el conjunto de operaciones permitidas para la misma.
  - El tipo de una variable ayuda a:
    - Proteger a las variables de operaciones no permitidas. ○ Chequear tipos.
    - Verificar el uso correcto de las variables.
    - Detectar errores en forma temprana.
    - Mejorar la confiabilidad del código.
  - Antes de que una variable pueda ser referenciada, debe ligarsele un tipo.
  - Existen 3 clases de Tipos: Predefinidos por el lenguaje, Definidos por el Usuario y los Tipos de Datos Abstractos.
  - En cuanto a los momentos de ligadura, puede ocurrir de dos formas: Estático o Dinámico.

Ejercicio 12

```
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
```

Identificador	Tipo/L-Valor	R-Valor	Alcance	Tiempo de Vida
Main (línea 2)	-	-	3-29	2-29
a (línea 4)	automática	basura	5-29	2-29
n (línea 4)	automática	basura	5-29	2-29
p (línea 4)	automática	basura	5-11 -> 23-29	2-29
v1 (línea 5)	automática	basura	6-29	2-29
c1 (línea 6)	automática	basura	7-10 -> 23-29	2-29
Uno (línea 7)	-	-	8-29	7-22
v2 (línea 9)	semidinámica	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	basura	12-22	7-22
q^	dinámica	nil	12-22	16-20

Ejercicio 13

Inciso a)

- No, el nombre de una variable no puede condicionar el tiempo de vida de la misma, el concepto de nombre es lo que se utiliza para referenciar a la variable mientras que el tiempo de vida de la misma es el periodo de tiempo que la variable está alocada en memoria y su binding existe.

Inciso b)

- Si, el nombre de una variable puede condicionar el alcance de la misma ya que el alcance es definido como el rango de instrucciones en el que es conocido el nombre de la variable, por lo tanto, este podría condicionar el factor del alcance.

Inciso c)

- No, el nombre no condiciona el r-valor de la misma, este último concepto es definido como el valor codificado almacenado en la locación asociada a la variable, es decir, a su l-valor. Nosotros podemos acceder al r-valor gracias al nombre ya que este es el que nos sirve para poder referenciar a la misma.

Inciso d)

- No, el nombre de una variable no condiciona el tipo de la misma, el tipo normalmente se define en la declaración de la variable y es el que condiciona que conjunto de valores y operaciones podemos hacer con la variable pero esta puede tener cualquier nombre y esto no va a interferir con el nombre de la misma.

Ejercicio 14

Identificador	Tipo/L-Valor	R-Valor	Alcance	Tiempo de Vida
v1 (línea 1)	automática	0	2-4 -> 9-12 -> 21-23	1-28
a (línea 2)	automática	null	3-16	1-28
a^ (línea 2)	dinámica	basura	3-16	15-16
fun2 (línea 3)	-	-	4-16	3-8
v1 (línea 4)	automática	basura	5-8	3-8
y (línea 4)	automática	basura	5-8	3-8
main (línea 9)	-	-	10-16	9-16
var3 (línea 10)	estática	0	11-16	1-28
v1 (línea 12)	automática	basura	13-16	9-16
y (línea 12)	automática	basura	13-16	9-16
var1 (línea 14)	automática	‘C’	15	13-15
aux (línea 17)	estática	0	18-25	1-28
v2 (línea 18)	automática	0	7 -> 12-16 -> 19-28	1-28
fun2 (línea 19)	-	-	20-23	19-23
fun3 (línea 24)	-	-	25-28	24-28
aux (línea 25)	automática	basura	26-28	24-28

Ejercicio 15

Modificador	Javascript	Python
“const”	Declara una variable con un valor constante, que no puede ser reasignado. Debe ser inicializado al declararlo.	No hay un equivalente directo en Python. Las variables en Python pueden ser reasignadas, pero una convención similar a “const” es usar nombres de variables en mayúsculas para indicar que no deben ser modificadas.
“var”	Declara una variable de alcance global o de función. No tiene restricciones de	No hay un equivalente directo en Python. En Python, todas las variables

	reasignación o redeclaración. Es menos comúnmente utilizado desde la introducción de let y const.	declaradas dentro de una función tienen un alcance local. Variables definidas fuera de las funciones tienen un alcance global.
“let”	Declara una variable de bloque con un alcance limitado al bloque de código en el que está definida. Puede ser reasignada pero no redeclarada en el mismo ámbito.	No hay un equivalente directo en Python. Sin embargo, las variables definidas en un bloque de código (como dentro de un bucle for o if) tienen un alcance limitado a ese bloque.
Ausencia	Si una variable se declara sin ninguno de los modificadores anteriores, su comportamiento depende del contexto: en el caso de variables declaradas fuera de cualquier función, se comportan como variables globales (var); si se declaran dentro de una función, se comportan como variables de función (var). Desde ECMAScript 6 (ES6), se considera una mala práctica declarar variables sin especificar un modificador.	Las variables sin declarar son consideradas variables locales en Python, y arrojarán un error si se intentan utilizar antes de ser inicializadas. Es una buena práctica inicializar todas las variables antes de usarlas.