

Ejercicios de Programación - Sebesta

Alvarado J., Lasso H. & Lozano E.

13 de febrero de 2014

1. Introducción

Las respuestas propuestas en este repositorio son producto del trabajo de los estudiantes de la materia “Lenguajes de Programación” de la ESPOL, correspondientes a las preguntas del libro de Robert Sebesta, Concepts of Programming Languages.

2. Preguntas y Respuestas

2.1. Capítulo 5: Nombres, Enlaces y Alcances

- **Pregunta 4:** Write a Python function that has subprograms nested three deep and in which each nested subprogram references variables defined in all of its enclosing subprogram

```
def funcion0():  
    def funcion1():  
        nombre1="hola soy 1"  
        def funcion2():  
            nombre2="hola soy 2"  
            def funcion3():  
                nombre3="hola soy 3"  
                funcion3()  
            print("%s en f2"%(nombre3))  
        funcion2()  
        print("%s en f1"%(nombre2))  
        print("%s en f1"%(nombre3))  
    funcion1()  
    print("%s en f0"%(nombre1))  
    print("%s en f0"%(nombre2))  
    print("%s en f0"%(nombre3))  
funcion0()
```

```
C:\Users\NiAS>C:\Users\NiAS\Documents\Python\test3.py  
Traceback (most recent call last):  
  File "C:\Users\NiAS\Documents\Python\test3.py", line 17, in <module>  
    funcion0()  
  File "C:\Users\NiAS\Documents\Python\test3.py", line 13, in funcion0  
    funcion1()  
  File "C:\Users\NiAS\Documents\Python\test3.py", line 10, in funcion1  
    funcion2()  
  File "C:\Users\NiAS\Documents\Python\test3.py", line 9, in funcion2  
    print("%s en f2"%(nombre3))  
NameError: global name 'nombre3' is not defined  
C:\Users\NiAS>
```

NO EJECUTADO EN PYTHON: Variables no son declaradas globales

- **Pregunta 5:** Write a C function that includes the following sequence of statements: `x = 21;int x;x = 42;` Run the program and explain the results. Rewrite the same code in C++ and Java and compare the results.

- Función en C

```
#include<conio.h>
#include<stdlib.h>

int main(){

    x = 21;
    int x;
    x = 42;

}
```

NO COMPILA: error C2065: 'x' : identificador no declarado

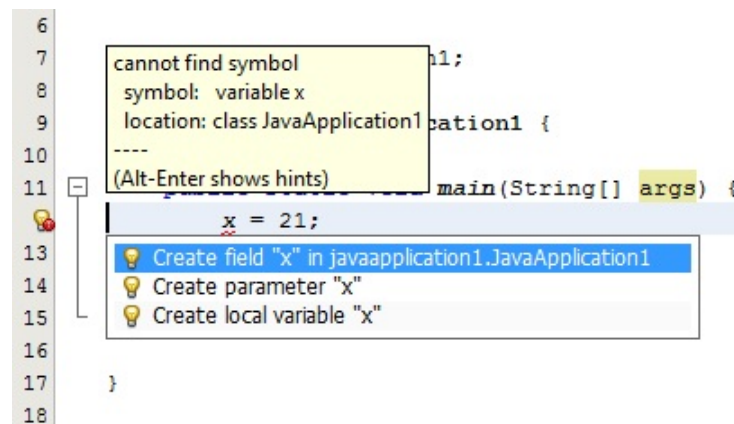
ARGUMENTO: En el lenguaje C es necesario crear la variable o instanciarla de las siguientes formas antes de poder usarla o asignarle un valor:

- (tipo) (nombreVariable);
Ejemplo: int x;
- (tipo) (nombreVariable)=(valor Inicial);
Ejemplo: int x=10;

- Función en C++

Sucede lo mismo que en C, el compilador de visual presenta los mismos errores, como si las tres lineas estuvieran mal escritas, aunque realmente es la primera.

- Función en Java



NO COMPILA: error: cannot find symbol

ARGUMENTO: En el lenguaje Java, el intérprete antes de compilar sugiere al programador crear la variable X como una variable de clase, y una vez compilado mostrar el error de que no puede encontrar el simbolo que se esta intentado usar, en este caso X.

- **Pregunta 6: Write test programs in C++, Java, and C# to determine the scope of a variable declared in a for statement. Specifically, the code must determine whether such a variable is visible after the body of the for statement.**

- Función en C++

```
#include<stdio.h>

int main(){
    int i=5;
    int length=0;
    for (i = 0; i < 5; i++)
    {
        int cont;
        length++;
        cont=length+5;
    }
    printf("%i",cont);
}
```

Error: el identificador "cont" no está definido

NO COMPILA: No es visible despues de la sentencia for.

- Función en C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            int i=100;
            for (int j = 0; j < 5; j++)
            {
                int holamundo= 5;
                holamundo++;
            }
            Console.WriteLine("Hello World! {1}",holamundo,i);
            Console.WriteLine("Press any key to e
            Console.ReadKey();
        }
    }
}
```

El nombre 'holamundo' no existe en el contexto actual

NO COMPILA: No es visible despues de la sentencia for.

Es de notar que los avisos que muestra el intérprete son diferentes en los dos casos anteriores, en C++ nos dice que no existe definición de la

variable que se quiere usar y por tanto no hay ningún valor que mostrar. En la siguiente, en C# en cambio nos informa que no estamos en el mismo contexto y que por tanto la variable no es visible ni alcanzable.

- **Pregunta 7: Write three functions in C or C++: one that declares a large array statically, one that declares the same large array on the stack, and one that creates the same large array from the heap. Call each of the subprograms a large number of times (at least 100,000) and output the time required by each. Explain the results.**

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include <time.h>

void funcion1();
void funcion2();
void funcion3();

int main(){
    clock_t start = clock();
    int i;
    for(i=0;i<100000;i++){
        funcion1();
    }
    printf("Tiempo transcurrido: %f", ((double)clock() - start) / CLOCKS_PER_SEC);
    getch();
}

void funcion1(){
    static int array1[100]={0};
}

void funcion2(){
    int array2[100]={0};
}

void funcion3(){
    int* array3;
    array3 = (int*)malloc (100*sizeof(int));
    free(array3);
}
```

Tiempo transcurrido: 0.002000

Tiempo transcurrido: 0.005000

Tiempo transcurrido: 0.214000

EN C: Diferentes tiempos para diferentes funciones

ARGUMENTO: Las variables y vectores en C ocupan un tamaño fijo, el cuál es establecido antes de ejecutar el programa, por tanto no pueden variarlo durante esta ejecución. Por medio de punteros se puede reservar o liberar memoria dinámicamente, es decir, según se necesite. La Función malloc sirve para solicitar un bloque de memoria del tamaño suministrado como parámetro. Es notable que la petición MALLOC al querer reservar memoria toma más tiempo en su ejecución, pero también es importante entender que es una forma segura de programar, ya que estaremos seguros de que obtendremos la memoria que necesitamos para las variables que usemos, sin tener errores de memoria en tiempo de ejecución. Así que aunque en la imagen veamos que toma más tiempo ejecutar MALLOC es preferible su uso, para no arriesgarnos a errores de memoria.

2.2. Capítulo 6: Tipos de Datos

- **Pregunta 1: Design a set of simple test programs to determine the type compatibility rules of a C compiler to which you have access. Write a report of your findings.**

La estructura jerárquica de un programa normalmente se expresa utilizando reglas recursivas. Por ejemplo, se pueden dar las siguientes reglas como parte de la definición de expresiones:

Cualquier identificador es una expresión. Cualquier número es una expresión. Si expresión1 y expresión2 son expresiones, entonces también lo son: expresión1 + expresión2 expresión1 * expresión2 (expresión1)

Las reglas 1 y 2 son reglas básicas (no recursivas), en tanto que la regla 3 define expresiones en función de operadores aplicados a otras expresiones.

La división entre análisis léxico y análisis sintáctico es algo arbitraria. Un factor para determinar la división es si una construcción del lenguaje fuente es inherentemente recursiva o no. Las construcciones léxicas no requieren recursión, mientras que las construcciones sintácticas suelen requerirla. No se requiere recursión para reconocer los identificadores, que suelen ser cadenas de letras y dígitos que comienzan con una letra. Normalmente, se reconocen los identificadores por el simple examen del flujo de entrada, esperando hasta encontrar un carácter que no sea ni letra ni dígito, y agrupando después todas las letras y dígitos encontrados hasta ese punto en un componente léxico llamado identificador.

```
int min(int a, int b) {
    if (a < b) return a;
    return b;
}
int min(string a, string b) {
    if (strcmp(a, b) < 0) return a;
    return b;
}
double x,y;
int a,b;
char c[ ],d[ ];

u = min(x,y);
v = min(a,b);
w = min(c,d);
```

```
t = min(x, c);
```

Error al tratar de comparar un tipo de dato double con un char[]

- **Pregunta 2: Determine whether some C compiler to which you have access implements the free function** Memoria Dinámica Es memoria que se reserva en tiempo de ejecución. Su principal ventaja frente a la estática, es que su tamaño puede variar durante la ejecución del programa. (En C, el programador es encargado de liberar esta memoria cuando no la utilice más). El uso de memoria dinámica es necesario cuando a priori no conocemos el número de datos/elementos a tratar.

Memoria Estática Es el espacio en memoria que se crea al declarar variables de cualquier tipo de dato (primitivas [int, char...] o derivados [struct, matrices, punteros...]). La memoria que estas variables ocupan no puede cambiarse durante la ejecución y tampoco puede ser liberada manualmente.

La función free sirve para liberar memoria que se asignó dinámicamente. Si el puntero es nulo, free no hace nada.

- **Pregunta 7: Write a C program that does a large number of references to elements of two-dimensioned arrays, using only subscripting. Write a second program that does the same operations but uses pointers and pointer arithmetic for the storage-mapping function to do the array references. Compare the time efficiency of the two programs. Which of the two programs is likely to be more reliable? Why?**

Programa 1

```
#include <stdio.h>
#include <conio.h>
#include <string.h>

void main (void){
    char cadena[10][70];
    for (i = 0; i <= 9; i++) {
        for (j = 0; j <= 69; j++) {
            printf("El valor es :%d", cadena[i][j]);
        }
    }
}
```

Accedimos a los elementos de la matriz por medio de subíndice Programa 2

```

#define FILAS 10
#define COLS 70
int **matriz;
matriz = (int **)malloc (FILAS*sizeof(int *)*COLS);

for (i = 0; i <= FILAS; i++) {
    for (j = 0; j<=COLS; j++) {
        printf("El valor es :%d",matriz** );
    }
}
matriz[i] = (int *) malloc (COLS*sizeof(int));

```

Accedimos a cada elemento de la matriz por medio de punteros Lo mismo que en el caso de matrices unidimensionales, existen dos formas para acceder los elementos de matrices multidimensionales: mediante subíndices y mediante punteros. En cuanto a la primera, la forma de designar un elemento es: $a[f][c]$, el primer subíndice indica la fila y el segundo la columna.

En caso de una matriz bidimensional $m[A][B]$, la posición $P_{a,b}$ respecto del comienzo del elemento $m[a][b]$ viene determinado por: $P_{a,b} = (a * B) + b$. Es decir: (fila x total de columnas) + columna

2.3. Capítulo 7: Expresiones e Instrucciones de asignación

- **Pregunta 1: Run the code given in Problem 13 (in the Problem Set) on some system that supports C to determine the values of sum1 and sum2. Explain the results.**

El código:

```

1  int fun(int *k) {
2      *k += 4;
3      return 3 * (*k) - 1;
4  }
5
6  void main() {
7      int i = 10, j = 10, sum1, sum2;
8      sum1 = (i / 2) + fun(&i);
9      sum2 = fun(&j) + (j / 2);
10 }

```

Nos devuelve $sum1 = 46$ y $sum2 = 48$

```

1 sum1 = (i/2) + fun(&i) = 5 + 41 = 46
2 //fun se calcula despues de obtener el valor de i/2
3 sum2 = fun(&j) + (j/2) = 41 + 7 = 48
4 //fun se calcula antes de obtener el valor de j/2

```

■ **Pregunta 2: Rewrite the program of Programming Exercise 1 in C++, Java, and C#, run them, and compare the results**

● Programa en C++, C#

```

1 class Program
2 {
3     static void Main(string[] args)
4     {
5         Program p = new Program();
6         int i = 10, j=10, sum1, sum2;
7         sum1 = (i / 2) + p.fun(ref i);
8         sum2 = p.fun(ref j) + (j / 2);
9         System.Console.WriteLine("sum1 = "+sum1);
10        System.Console.WriteLine("sum2 = " + sum2);
11        Console.Read();
12    }
13    public int fun(ref int k)
14    {
15        k = 4 + k;
16        return 3*(k) - 1;
17    }
18 }

```

El programa retorna el mismo resultado que C

● Programa en Java

```

1 public class TestLP {
2     public static void main(String[] args) {
3         TestLP test = new TestLP();
4         int i = 10, j = 10, sum1, sum2;
5         sum1 = (i / 2) + test.fun(i);
6         sum2 = test.fun(j) + (j / 2);
7         System.out.println("sum1= " + sum1);
8         System.out.println("sum2= " + sum2);
9     }
10    public int fun(int k) {
11        k += 4;

```



```

12         return 3 * (k) - 1;
13     }
14 }

```

El programa retorna: sum1= 46 y sum2= 46, ya que no se pueden utilizar punteros en Java.

- **Pregunta 3: Write a test program in your favorite language that determines and outputs the precedence and associativity of its arithmetic and Boolean operators.**

```

1 public class TestLP {
2     public static void main(String[] args) {
3         boolean a;
4         double e;
5         a = true || false && false || false;
6         System.out.println(a);
7         a = false || true && false || false && false;
8         System.out.println(a);
9         e = 1 + 2 + 3 * 3 / 6.0;
10        System.out.println(e);
11        e = (1 + 2) + ((3 * 3) / 6.0);
12        System.out.println(e);
13    }
14 }

```

Como resultado obtenemos lo siguiente por consola

```

1 true
2 false
3 4.5
4 4.5

```

Lo que nos indica que en las operaciones booleanas el operador AND (&&) tiene mayor precedencia que el operador OR (||). En las operaciones aritmeticas el operador * tiene mayor precedencia que / y +

- **Pregunta 4: Write a Java program that exposes Java's rule for operand evaluation order when one of the operands is a method call.**

```

1 public class TestLP {
2     public static void main(String[] args) {
3         TestLP test = new TestLP();
4         int a = 0, c;

```

```

5      c = (a) + test.func(--a);
6      System.out.println("c = " + c);
7      System.out.println("a = " + a);
8      a = 0;
9      c = test.func(--a) + (a);
10     System.out.println("c = " + c);
11     System.out.println("a = " + a);
12 }
13 public int func(int c) {
14     return c + 10;
15 }
16 }

```

El programa retorna:

```

1  c = 9
2  a = -1
3  c = 8
4  a = -1

```

Lo que nos muestra que las funciones se evalúan de izquierda a derecha

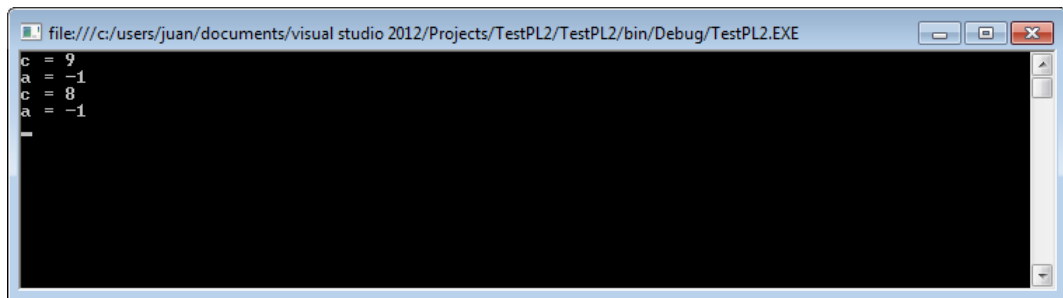
■ Pregunta 5: Repeat Programming Exercise 5 with C++.

```

1  class Program
2  {
3      static void Main(string[] args)
4      {
5          Program p = new Program();
6          int a = 0, c;
7          c = (a) + p.func(--a);
8          System.Console.WriteLine("c = " + c);
9          System.Console.WriteLine("a = " + a);
10         a = 0;
11         c = p.func(--a) + (a);
12         System.Console.WriteLine("c = " + c);
13         System.Console.WriteLine("a = " + a);
14         Console.Read();
15     }
16     public int func(int c)
17     {
18         return c + 10;
19     }
20 }

```

El programa retorna:

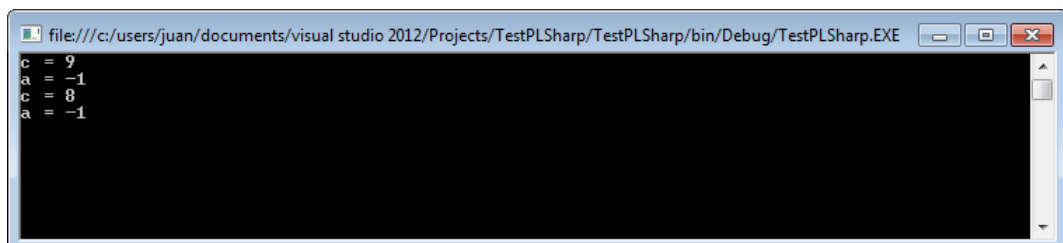


```
file:///c:/users/juan/documents/visual studio 2012/Projects/TestPL2/TestPL2/bin/Debug/TestPL2.EXE
c = 9
a = -1
c = 8
a = -1
-
```

■ Pregunta 6: Repeat Programming Exercise 6 with C#.

```
1 class Program
2 {
3     static void Main(string[] args)
4     {
5         Program p = new Program();
6         int a = 0, c;
7         c = (a) + p.func(--a);
8         System.Console.WriteLine("c = " + c);
9         System.Console.WriteLine("a = " + a);
10        a = 0;
11        c = p.func(--a) + (a);
12        System.Console.WriteLine("c = " + c);
13        System.Console.WriteLine("a = " + a);
14        Console.Read();
15    }
16    public int func(int c)
17    {
18        return c + 10;
19    }
20 }
```

El programa retorna:



```
file:///c:/users/juan/documents/visual studio 2012/Projects/TestPLSharp/TestPLSharp/bin/Debug/TestPLSharp.EXE
c = 9
a = -1
c = 8
a = -1
```

- **Pregunta 9:** Write a program in either Java, C++, or C# that performs a large number of floating-point operations and an equal number of integer operations and compare the time required.

```
1 public class TestLP {
2     public static void main(String[] args) {
3         int ae[] = new int[500000];
4         float af[] = new float[500000];
5         float accf = 0;
6         long t1, t2, t3, te, tf;
7         int acce = 0;
8         Random r = new Random();
9         for(int i=0; i< 500000; i++){
10             af[i]=r.nextFloat();
11             ae[i]=r.nextInt(40);
12         }
13         t1= System.nanoTime();
14         for(int e:ae){
15             acce+=e;
16         }
17         t2=System.nanoTime();
18         for(float f:af){
19             accf+=f;
20         }
21         t3=System.nanoTime();
22         te=t2-t1;
23         tf=t3-t2;
24         System.out.println("Total Entero = " + acce);
25         System.out.println("Total Flotante = " + accf);
26         System.out.println("Tiempo Entero = " + te);
27         System.out.println("Tiempo Flotante = " + tf);
28         System.out.println(t1);
29         System.out.println(t2);
30         System.out.println(t3);
31     }
32 }
```

El tiempo Entero es de alrededor de 7 milisegundos, mientras que el tiempo de flotantes es de alrededor de 0.5 milisegundos

2.4. Capítulo 8: Estructuras de Control

- **Pregunta 3:** Rewrite the following code segment using a multiple-selection statement in the following languages:

```
if ((k == 1) || (k == 2)) j = 2 * k - 1
```

```
if ((k == 3) || (k == 5)) j = 3 * k + 1
```

```
if (k == 4) j = 4 * k - 1
```

```
if ((k == 6) || (k == 7) || (k == 8)) j = k - 2
```

a. Fortran 95 (you'll probably need to look this one up)

b. Ada

c. C, C++, Java, or C

d. Python

e. Ruby

Assume all variables are integer type. Discuss the relative merits of the use of these languages for this particular code.

Fortran 95

```
if ( (k == 1) .or. (k == 2) ) then
  j = 2 * k - 1
elseif ( (k == 3) .or. (k == 5) ) then
  j = 3 * k + 1
elseif (k == 4)
then j = 4 * k - 1
elseif ( (k == 6) .or. (k == 7) .or. (k == 8) )
then j = k - 2
end if
```

C,C++,Java,or C

```
if ((k == 1) || (k == 2))
  j = 2 * k - 1
elseif ((k == 3) || (k == 5))
  j = 3 * k + 1
elseif (k == 4)
  j = 4 * k - 1
elseif ((k == 6) || (k == 7) || (k == 8))
  j = k - 2
```

Python

```
if ((k == 1) or (k == 2)) :
  j = 2 * k - 1
elseif ((k == 3) or (k == 5)):
```

```

j = 3 * k + 1
elseif (k == 4) :
j = 4 * k - 1
elseif ((k == 6) or (k == 7) or (k == 8)):
j = k - 2

```

Ruby

```

if ((k == 1) || (k == 2))
j = 2 * k - 1
elseif ((k == 3) || (k == 5))
j = 3 * k + 1
elseif (k == 4)
j = 4 * k - 1
elseif ((k == 6) || (k == 7) || (k == 8))
j = k - 2
end

```

- **Pregunta 4:** Consider the following C program segment. Rewrite it using no gotos or breaks.

```

j = -3;
for (i = 0; i < 3; i++)
switch (j + 2)
case 3:
case 2: j--; break;
case 0: j += 2; break;
default: j = 0;

if (j > 0) break;
j = 3 - i

```

```

j = -3;
for (i = 0; i < 3; i++) {
    if (j + 2) == 3

        elseif (j + 2) == 2 {
            j--;
            return ;
        }
    elseif (j + 2) == 0{

```

```

                                j += 2;
                                return;
                        }
                        else
                                j = 0;
                }
                if ((j + 2) > 0)
                        return;
                j = 3 - i
        }

```

- **Pregunta 5:** In a letter to the editor of CACM, Rubin (1987) uses the following code segment as evidence that the readability of some code with gotos is better than the equivalent code without gotos. This code finds the first row of an n by n integer matrix named x that has nothing but zero values.

```

for (i = 1; i <= n; i++)
for (j = 1; j <= n; j++)
if (x[i][j] != 0)
goto reject;
println ('First all-zero row is:', i);
break;
reject:

```

Rewrite this code without gotos in one of the following languages: C, C++, Java, C, or Ada. Compare the readability of your code to that of the example code.

```

for (i = 1; i <= n; i++) {
    for (j = 1; j <= n; j++){
        if (x[i][j] != 0)
            continue;
    }
    System.out.Println('First all-zero row is:' + i);
    break;
}

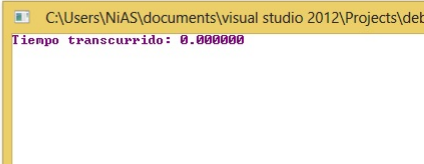
```

2.5. Capítulo 9: SubProgramas

- **Pregunta 1:** Write a program in a language that you know to determine the ratio of the time required to pass a large array by reference and the time required to pass the same array by value. Make the array as large as possible on the machine and implementation you use. Pass the array as many times as necessary to get reasonably accurate timings of the passing operations.

```
int main(){
    clock_t start = clock();
    int array[100000]={1};
    Referencia(array);
    printf("Tiempo transcurrido: %f", ((double)clock() - start) / CLOCKS_PER_SEC);
    getch();
}

int Referencia(int *r)
{
    int i;
    for(i=0; i<100000; i++){
        r[i]=5;
    }
}
```



POR REFERENCIA: Es notable que es más eficiente la ejecución al pasar un arreglo por referencia.

```
int main(){
    clock_t start = clock();
    int array[100000]={1,1};
    for(i=0; i<100000; i++){
        array[i]=Valor(array[i]);
    }
    printf("Tiempo transcurrido: %f", ((double)clock() - start) / CLOCKS_PER_SEC);
    getch();
}

int Valor(int v)
{
    v=5;
    return v;
}
```



POR VALOR: Se modificó la función para que hiciera lo mismo que la anterior, y notamos un cambio en los tiempos pero pasando cada elemento del arreglo por valor.