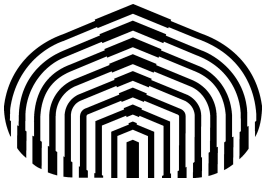

Universidad Simón Bolívar
Departamento de Computación y Tecnología de la información
CI-2691- Laboratorio de algoritmos I

Laboratorio 4

El objetivo de este laboratorio es el estudio de arreglos unidimensionales y multidimensionales, ciclos for, ciclos anidados, constructor de tipos estructurados y arreglos de estructuras.

Arreglos

Son estructuras de datos que permiten implementar secuencias de elementos de un mismo tipo. Pueden ser de una o varias dimensiones (los llamados multidimensionales). En GCL los arreglos se declaran de la forma `array [E1)x...x[En) of T`, en donde los `Ei` corresponden a expresiones que representan el tamaño de la dimensión `i`, generalmente expresadas como un rango de valores `0..ki`, `n` es el número de dimensiones y `T` es el tipo de los elementos del arreglo. Los arreglos permiten representar objetos muy útiles y conocidos como son los vectores y matrices. A continuación se muestra un ejemplo de cómo declarar en GCL un arreglo bidimensional de tamaño 3x5, con elementos de tipo entero:

```
CONST
   A: array[0..3)[0..5) of int
```

La forma más sencilla de implementar arreglos en Python es a través de las listas. Por ello, este curso sólo se usará esta implementación. Hay otra implementación que puede ser consultada en [1], sin embargo, está fuera del alcance de este curso.

Listas en Python

En Python las listas pueden ser inicializadas como una secuencia de elementos separados por comas y colocados entre corchetes, como se muestra a continuación:

```
>>> a = [0, 14, 100, 1234]
```

Para accesar los elementos del arreglo individuales del arreglo, se hace de forma similar que en GCL a través del índice, es decir, se coloca una expresión entera encerrada entre corchetes

a continuación del nombre del arreglo, siendo el 0 el primer elemento. Por ejemplo, para el arreglo anterior, pruebe en el interpretador de Python las siguientes acciones :

```
>>> a[0]
>>> a[5]  // aquí se produce un error pues el arreglo sólo tiene 4 elementos
>>> a[1+1]
```

En Python también se pueden concatenar arreglos con el operador `+`, y también se puede concatenar **n** veces la lista en sí misma con el operador `*`**.** Por ejemplo, pruebe las siguientes acciones en el interpretador de Python:

```
>>> [1,2,3]+[5,6,7,9]
>>> [5,6,7,9]*2
>>> [1,2,3]+[5,6,7,9]*2
```

Listas Anidadas en Python

A través de las listas anidadas se implementan los arreglos multidimensionales de GCL mencionados anteriormente, en Python. Por ejemplo, para inicializar un arreglo de enteros de 3 dimensiones de tamaño 3, similar a una matriz 3x3, podemos realizarlo de la siguiente manera.

```
>>> mat = [
...         [1, 2, 3],
...         [4, 5, 6],
...         [7, 8, 9],
...        ]
```

Para accesar los elementos de un arreglo multidimensional, se deben colocar tantos índices como dimensiones tenga el arreglo, de forma similar que GCL, pero rodeando cada dimensión con los corchetes, siendo el `0` el primer elemento en cada dimensión y `n-1` el último elemento si la dimensión es de tamaño `n`. Por ejemplo, para el arreglo `mat`, pruebe en el interpretador de Python las siguientes acciones :

```
>>> mat[0][0]
>>> mat[1][2]
>>> mat[2][2]
>>> mat[3][3] //en este caso da error, por qué?
```

Ciclos FOR

En GCL solo se utiliza el lazo `do` (estudiado en el Prelaboratorio 3). Sin embargo, en Python tenemos otra opción de lazo, que es el `for`. Básicamente, el `for` se usa para el recorrido en listas [2]. La sintaxis de este lazo comienza con la palabra reservada `for` seguida de la

variable que recibirá el valor de cada elemento de la lista, luego la palabra reservada `in`, y la expresión que pueda ser evaluada como una lista seguida de dos puntos (`:`). Igual que en el lazo `while`, las acciones que se deseen ejecutar en cada iteración, se deben colocar con un nivel de indentación (o sangría) dentro del `for`. Por ejemplo, pruebe el siguiente lazo en el interpretador de Python:

```
>>>for x in range(1,4):
...    print(x)
...
```

De esta forma se tiene dos maneras de implementar el recorrido en arreglos (listas) de Python, dependiendo del ciclo que se utilice: `while` o `for`. El `while` es completamente equivalente al do de GCL. El `for` da un recorrido alternativo sin manejo directo del contador del ciclo. Por ejemplo, para imprimir los elementos del arreglo `a` podemos hacerlo de las siguientes dos maneras:

`k=0`
`while ( k < 4 ):`
`  print(a[k])`
`  k=k+1`
	`for k in range(0,4):`
	`  print(a[k])`

También se puede utilizar el lazo `for` para la declaración de listas de una manera más abreviada. Su estructura es bastante parecida a los cuantificadores realizados en los laboratorios anteriores.

Por ejemplo, si queremos declarar una lista cuyos elementos sean los números desde el 5 al 10, se escribe:

```
>>> [ x for x in range(5,11) ]
```

Si necesitamos agregarle algún tipo de condición, podemos colocarla al final. Por ejemplo, el cuadrado de todos los números pares desde el 5 al 10. Tenemos:

```
[ x*x for x in range(5,11) if ( x % 2 == 0) ]
```

Ciclos Anidados

Cuando una de las instrucciones que aparece dentro de un ciclo es también un ciclo, estamos ante la presencia de ciclos anidados.

Un caso muy común donde aparecen es en los recorridos de matrices. Por ejemplo el siguiente trozo de programa en GCL muestra la suma de los elementos de una matriz m de tamaño NxN o arreglo bidimensional.

```
f,suma := 0,0;
do f < N ->
   c:=0;
   do c < N ->
      suma := suma+m[f,c];
      c:=c+1
   od;
   f:=f+1
od
```

En este ejemplo el ciclo externo realiza N iteraciones, y el interno N iteraciones por cada iteración del ciclo externo, por tal razón el número total de iteraciones realizadas en N*N.

Si se usa un while, un trozo de programa en Python equivalente es

```
f,suma = 0,0
while f < N :
   c=0
   while c < N :
      suma = suma+m[f][c]
      c=c+1
   f=f+1
```

Con el ciclo `for` también se pueden realizar ciclos anidados. El ciclo anterior usando for quedaría

```
suma = 0
for f in range(0,N) :
   for c in range(0, N) :
      suma = suma+m[f][c]
```

Por ello, usando `for` se pueden hacer recorridos sobre listas multidimensionales. Por ejemplo, pruebe en el interpretador de Python las siguientes acciones.

```
>>> mat = [ [1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> for f in range(0,3):
...   for c in range(0,3):
...     print (mat[f][c])
...
```

Invariantes y Cotas en Ciclos FOR

De igual manera que los lazos `while`, pueden colocarse invariantes y cotas a los lazos `for`. Sin embargo, el uso de la función `range` dentro de un for garantiza que el lazo tiene una función de cota que decrece y es positivo por lo cual no es necesario estas condiciones. Sólo indicar cual es la función de cota y verificar el invariante. Para ello seguiremos el mismo esquema que para el lazo `while`, como se indica a continuación:

```
# Cota N - i
# Invariante
assert ( ... condiciones que se deben cumplir ... )

for i in range (0,N):

   # operaciones del lazo

   # Invariante
   assert ( ... condiciones que se deben cumplir ... )
```

Tipos de Datos Estructurados

En Python, para construir estructuras de datos que agrupen atributos de diferentes tipos se utilizan las clases. Con ellas se puede definir un nuevo tipo de datos, que contiene uno o varios atributos con tipos de datos diferentes.

La clase se define utilizando la palabra reservada `class`, seguida del nombre de la clase, y luego dos puntos (`:`). Luego se colocan los atributos o campos de la clase con un valor inicial que indica el tipo de dicho atributo. Para indicar que los atributos pertenecen a la clase en necesario colocar un nivel de indentación o sangría con respecto a la palabra `class`. Por ejemplo:

```
class Persona:
   edad = 0      # tipo entero
   altura = 0.0  # tipo real
```

Luego para indicar que una variable es del tipo Persona, se le asigna se utiliza la sintaxis `<var>=<tipo>()`. A continuación se pueden usar o modificar los atributos, teniendo acceso a ellos con la notación punto (`.`), de manera similar como se usaba la parte real e imaginaria de un complejo. Por ejemplo, realice las siguientes pruebas en el interpretador de Python

```
>>> yo = Persona()
>>> yo.edad = 29     # cambia el valor de la edad
```

```
>>> yo.altura = 1.75 # cambia el valor de la altura
```

Para comparar dos estructuras es necesario preguntar individualmente por cada campo, pues cada estructura es un objeto diferente. Por ejemplo realice las siguientes acciones en su interpretador de Python.

```
>>> tu = Persona()
>>> tu.edad = 29
>>> tu.altura = 1.75
>>> yo == tu  # aquí el resultado es false pues son objetos distintos
>>> yo.edad == tu.edad 29 and yo.altura == tu.altura #esto si es true
```

ADVERTENCIA: no es correcto asignar directamente por el nombre una estructura a otra, pues esto genera un efecto de borde que puede producir que su programa no funcione bien más adelante. Observe lo que pasa con las siguientes acciones

```
>>> yo.edad = 15
>>> yo.altura = 1.60
>>> tu.edad = 29
>>> tu.altura = 1.75
>>> yo = tu # Prohibido realizar
>>> yo.altura
>>> yo.edad
>>> tu.altura = 1.45
>>> yo.altura
```

Por ello, la manera correcta de asignar los valores de una estructura a otra es la siguiente:

```
yo.edad = tu.edad
yo.altura = tu.altura
```

Listas de Estructuras en Python

Para declarar un arreglo de estructuras se hace de manera similar que la declaración de listas, pero ahora sus elementos son instancias de un tipo estructurado. Por ejemplo, si se quiere crear un arreglo de tres personas se escribiría

```
tresPersonas = [Persona(),Persona(),Persona()]
```

Para usar inicializar los valores de esta lista de estructuras debe acceder individualmente cada posición de la lista y dentro de ella cada campo, como se observa a continuación

```
tresPersonas[0].edad = 15
tresPersonas[0].altura = 1.65
tresPersonas[1].edad = 25
tresPersonas[1].altura = 1.75
tresPersonas[2].edad = 35
tresPersonas[2].altura = 1.55
```

Si se quiere realizar un cálculo sobre el arreglo de estructuras, por ejemplo, calcular el promedio de edades, se puede usar un for de la siguiente manera

```
suma = 0
for i in range(0,3):
   suma = suma + tresPersonas[i].edad
promedio = suma / 3
```

ADVERTENCIA: De igual manera, que con las variables declaradas como estructuras, no es correcto asignar directamente las posiciones de un arreglo de estructuras, pues también genera efectos de borde. Es necesario asignar cada campo individualmente. Por ejemplo:

```
tresPersonas[0] = trespersonas[2] # Prohibido, es incorrecto
```

La manera correcta es

```
tresPersonas[0].edad = tresPersonas[2].edad
tresPersonas[0].altura = tresPersonas[2].edad
```

Ejercicios:

1. PreLab4Ejercicio1.gcl: Escriba un algoritmo en GCL que dado un arreglo A de N enteros, calcule la suma de los elementos del arreglo. La postcondición es `{suma = (%sigma i: 0<=i<N : A[i])}`

2. PreLab4Ejercicio2.py: Traduzca el algoritmo del ejercicio 1 a un programa en Python. Para ello, lea N enteros, los coloque en un arreglo A y luego calcule la suma de los elementos. Utilice la instrucción **FOR** para hacer el lazo. Se sugiere utilizar, la siguiente inicialización del arreglo A:
   `A = [ int(input("A[" + str(i) + "]= ")) for i in range(N) ]`

3. PreLab4Ejercicio3.gcl: Escriba un algoritmo en GCL que dada una matriz cuadrada NxN, calcule si la matriz es diagonal. La postcondición del algoritmo es `{diagonal == (%forall i,j: 0<=i<N /\ 0<=j<N /\ i≠j: M[i][j]=0)}`

4. PreLab4Ejercicio4.py: Traduzca el algoritmo del ejercicio 3 a un programa en Python. Para ello, lea los elementos de una matriz cuadrada 3 x 3, los coloque en un arreglo multidimensional y luego calcule si es diagonal. Utilice la instrucción **FOR** para hacer los lazos. Se sugiere utilizar, la siguiente inicialización de la matriz M:

```
M=[ [int(input("M["+str(i)+","+str(j)+"]=")) for i in range(3)]
    for j in range(3)]
```

5. PreLab4Ejercicio5.py: Escriba un programa en Python que construya una clase Estudiante que almacene la edad, el nombre e índice académico de un estudiante. El programa lee los datos de un grupo de N estudiantes, con N leido de la entrada. Estos datos deben ser guardados en un arreglo de estudiantes, llamado **`grupo`**. Finalmente, calcule el promedio de la edad del grupo y el promedio del índice. Utilice la instrucción **FOR** para los ciclos. Como precondiciones se debe satisfacer que el nombre debe tener al menos 1 caracter, la edad positiva y el índice debe estar en el intervalo [1,5]. Se sugiere utilizar la función **`len`** de Python para calcular el tamaño de un string. Inicialice el arreglo **`grupo`** de la siguiente manera:

```
grupo = [ Estudiante() for x in range(N) ]
```

Referencias

[1] Arrays, Documentación oficial de Python. Disponible en la Web.
https://docs.python.org/3.3/library/array.html

[2] ForLoop, Documentación oficial de Python. Disponible en la Web.
https://wiki.python.org/moin/ForLoop