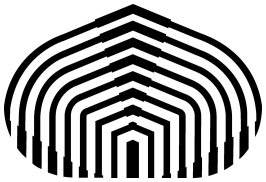

Universidad Simón Bolívar
Departamento de Computación y Tecnología de la información
CI-2691- Laboratorio de algoritmos I

Laboratorio 6

El objetivo de este laboratorio es construir programas en Python utilizando análisis descendente.

Contenido: análisis descendente, subprogramas no recursivos, pasajes de parámetros por valor y por referencia, su correspondencia con los parámetros de entrada, salida, entrada-salida de GCL y alcance de identificadores.

Análisis descendente y Subprogramas no recursivos

En los ejercicios que se han desarrollado a lo largo del curso, se puede observar una diferenciación natural de secciones dentro del bloque de instrucciones que conforma el código del programa. En casi todos ellos hay una sección de adquisición de datos (entrada), una sección de procesamiento algorítmico (cálculos) y una sección de despliegue de los resultados (salida).

Cuando la complejidad de un programa aumenta, es conveniente separar estas secciones en estructuras de programación que se conocen como subprogramas. Estos subprogramas pueden a la vez ser de cierta complejidad por lo que se requiere subdividirlos para poder resolver el problema de manera efectiva y legible. A esto se le conoce como la estrategia de "*Divide and Conquer*" que puede traducirse como "Divide y Vencerás".

Al proceso de concebir la solución de un problema con esta estrategia se le conoce como "Análisis Descendente". Este mismo término (análisis descendente) se usa para designar al diseño resultante de este proceso. El análisis descendente de un problema puede plasmarse en un diagrama llamado "Carta Estructurada". Éste ayuda en la comprensión y el desarrollo del programa.

Una ventaja de tener subprogramas es que ellos pueden ser reutilizados en distintas partes del código. El lenguaje de programación Python provee una estructura de subprograma,

conocida como función. Con esta estructura se pueden traducir los procedimientos y las funciones de GCL usados en el curso de Algoritmos y Estructuras 1.

La estructura general de una función en Python se inicia con la palabra reservada `def`, de manera similar a como se definen los predicados que habíamos visto en los laboratorios previos, seguida del nombre de la función con la lista de parámetros formales colocados entre paréntesis. Cada parámetro puede ser seguido de dos puntos (`:`), y el tipo de variable que se espera obtener. Para especificar el tipo de valor de retorno, coloque la símbolo `->` y luego el nombre del tipo. A continuación se coloca un ejemplo:

```
def perimetro(r: float, pi: float) -> float:
# PRECONDICION: r>0
# POSTCONDICION: perimetro = 2*pi*r

# var respuesta: float // variable auxiliar

   respuesta = pi * r * r

   return respuesta
```

Para realizar la llamada a una función, se coloca el nombre de ésta seguido de los parámetro reales (o actuales) separados por coma y encerrados entre paréntesis. Esta llamada debe colocarse dentro de una expresión del mismo tipo del valor de retorno. Por ejemplo, en una asignación a una variable, como se muestra a continuación:

```
# var a: float
a = perimetro( 20.5, 3.141 )
```

En el caso de los procedimientos que se pueden definir en GCL, también pueden traducirse a Python como funciones. Indicando que no tienen valor de retorno. Para esto se utiliza la palabra reservada `'void'`. En este caso, en la especificación del tipo de cada uno de los parámetros, se puede colocar un string con la palabra 'void'. Por ejemplo:

```
def inicializarLista (l:[int]) -> 'void':
# PRECONDICION: true
# POSTCONDICION: all (l[i]==0 for i in range(0,len(l))

# var i: int
  for i in range(0,len(l)):
     l[i] = 0
```

El lenguaje Python permite el uso recursivo de subprogramas, esto es que un subprograma tenga una llamada a sí mismo de forma directa o indirecta. Esto es algo de mayor complejidad al requerido en el taller de esta semana, pues será cubierto más adelante en este curso y en los cursos más avanzados. En este laboratorio sólo trabajaremos con subprogramas no recursivos. Igual que en los programas, los subprogramas debe tener especificadas las precondiciones y las postcondiciones, como se pudo observar en los ejemplos dados. Para más información en cuanto a funciones en python, se puede revisar la referencia [1].

Ejercicio: Tome el ejercicio 1 realizado en el prelaboratorio 5 (Prelab5ejercicio1.py),que calcula las raíces de una ecuación cuadrática. Escriba un subprograma que realice solamente dicho cálculo. Modifique el programa principal para que realice la llamada a este subprograma. Recuerde que cada función debe tener su precondición y postcondición. Guarde su programa con el nombre PreLab6ejercicio1.py y súbalo al aula virtual.

Pasaje de parámetros por valor y por referencia

Una diferencia notable entre los procedimientos y funciones del lenguaje de programación formal GCL, es la forma de pasaje de parámetros. En GCL se distinguen parámetros de entrada, parámetros de salida y parámetros de entrada y salida. El estudio de éstos es parte del contenido del curso teórico de Algoritmos y Estructuras I.

En los lenguajes de programación es muy común clasificar los parámetros en: por valor y por referencia. Los parámetros por valor son aquellos donde el valor del parámetro real es copiado al parámetro formal. Los parámetros por referencia son aquellos donde una dirección o referencia del parámetro real es copiada en el parámetro formal. Los parámetros por valor corresponden a los parámetros de entrada de GCL, mientras que los parámetros por referencia corresponden a los parámetros de entrada y salida de GCL.

En Python, los parámetros que son de algún tipo básico del lenguaje (como int, char, float o str) son tratados como parámetros por valor. Aquellos que corresponden a tipos no básicos como arreglos o estructuras son tratados como parámetros por referencia. También se puede devolver una tupla con varios valores de resultado. Esto es similar a los parámetros de salida de GCL, que se usan cuando se quiere devolver más de un valor.

Veamos un ejemplo.

```
def f(a: str, b: int) -> (str, int):
# PRECONDICION: b>0
# POSTCONDICION: f = (‘valor nuevo’,b+1)

   a = ‘valor nuevo’
   b = b + 1
```

```
    return a, b
```

Pruebe las siguientes acciones en el interpretador de Python.

```
x, y = 'valor viejo', 99
w, z = f (x, y)
print(x, y)
print(w, z)
```

Un ejemplo de pase de parámetros por referencia sería la función `inicializarLista` de la sección anterior. En este caso, el parámetro formal de la función se copia la dirección en memoria donde se ubica la lista (parámetro real), de modo que la función modifica la lista directamente y no una copia y por ende cualquier modificación sobre la misma es visible al salir de la función. Esto significa que si la lista `l` antes de la llamada a la función `inicializarLista` tiene 5 elementos, por ejemplo `l = [1,2,3,4,5]`, la lista al salir de la función tendrá los siguientes valores `l=[0,0,0,0,0]`. Para una ilustración de lo explicado hasta aquí descargue el archivo PreLab6Ejemplo1.py que aparece en el aula virtual junto a este enunciado.

Existen diferentes tratamientos para el pasaje de parámetros, pero serán vistas en cursos más avanzados.

Alcance de identificadores

Alcance es la palabra que se utiliza para describir la habilidad de un identificador para ser visible desde algún lugar de nuestro código. Hasta ahora hemos trabajado con dos tipos de alcances: global y local. Se explicarán los dos tipos de alcances de variables con el siguiente ejemplo:

```
globalVar = "Esta es de alcance global"

def miFuncion() -> 'void':
  localVar = "Esta es local"
  print("miFuncion - localVar:  " + localVar)
  print("miFuncion - globalVar: " + globalVar)

miFuncion()
print("global - globalVar: " + globalVar)
print("global - localVar:  " + localVar)
```

Coloque el PreLab6Ejemplo2.py y corra el programa. Observe lo que ocurre. Note que el programa falla porque el identificador `localVar`, está definido sólo dentro del subprograma `miFuncion`. Mientras que el identificador `globalVar` está definido en todo el programa.

Dentro de la declaración de un subprograma puede usarse un identificador que ya haya sido usado en el programa, con una definición nueva (diferente), es decir, se usa el mismo identificador, pero con otra definición. Esto sustituye la definición del identificador global (definido en el programa) por el identificador local (definido dentro del subprograma) hasta el final del bloque de instrucciones del subprograma. Los identificadores declarados dentro de un subprograma se dice que tienen alcance local.

En el cuerpo de un subprograma se pueden usar los identificadores definidos dentro del subprograma, es decir los identificadores locales. También se pueden usar los identificadores globales definidos antes de la declaración del subprograma que no hayan sido redefinidos dentro del mismo. En el ejemplo anterior esto último es el caso de `globalVar`. A estos últimos identificadores se les dice que son identificadores que designan elementos del "contexto global" del subprograma (o simplemente elementos o identificadores globales).

El uso de variables y parámetros globales tiene serios inconvenientes para la legibilidad de los programas y para la verificación de su corrección. Es aconsejable que se pase por parámetro cualquier dato que quiera compartirse entre subprogramas o entre el programa y un subprograma.

De manera que en este curso estará prohibido el uso de variables globales. Esto es, aunque las reglas de alcance permiten a un subprograma usar una variable o un parámetro de su contexto global, vamos a exigir que en el bloque de instrucciones se use sólo parámetros y variables declarados en el contexto local del propio subprograma.

Ejercicios: Tome los ejercicios 2 y 3 realizados en el prelaboratorio 5, y realice el cálculo del valor solicitado en un subprograma. Guárdelos con los nombres (PreLab6Ejercicio2.py y PreLab6Ejercicio3.py). No olvide respetar las sugerencias mencionadas sobre alcances de identificadores en este prelaboratorio.

Referencias

[1] Documentación oficial de Python, "Defining functions". Disponible en la web. https://docs.python.org/3/tutorial/controlflow.html#defining-functions

[2] Documentacion oficial de Python, "How do I write a function with output parameters". Disponible en la web. https://docs.python.org/3/faq/programming.html#how-do-i-write-a-function-with-output-parameters-call-by-reference