Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
Segundo Semestre de 2014

IIC 1103 - Introducción a la Programación

Laboratorio 11

Objetivo General

En este laboratorio reforzaremos los contenidos vistos en clases relacionados con recursión. Para llevar a cabo los ejercicios debes ser capaz de pensar recursivamente e implementar algoritmos recursivos.

Enunciado

1. En esta pregunta debes programar el algoritmo de Euclides para encontrar el MCD entre dos números. A continuación se describe el algoritmo suponiendo que se quiere calcular el MCD entre dos números `A` y `B`, con `A` $\leq$ `B` (notar que no hay pérdida de generalidad).

   - Si `A` divide a `B`, entonces el MCD entre `A` y `B` es `A`.
   - De lo contrario, el MCD entre `A` y `B` es igual al MCD entre `B % A` y `A`.

   Por ejemplo, si los números son 27 y 45, como 27 no divide a 45 entonces el MCD entre ellos se calcula como el MCD entre 18 (= 45`%` 27) y 27. Como 18 no divide a 27, el MCD se calcula como el MCD entre 9 (= 27`%` 18) y 18. Como 9 divide a 18, el número buscado es 9.

   Tu programa debe recibir como input los dos números sobre los cuáles se quiere calcular el MCD, en una línea y separados por un espacio. Debes considerar que el primer número no necesariamente es el menor. Como output debe imprimir los dos números correspondientes a cada paso del algoritmo, comenzando por el menor y separados por un espacio. Finalmente debe imprimir el número buscado. A continuación se muestran dos ejemplos de ejecuciones de este programa:

caso	input	output
1	27 45	27 45 18 27 9 18 9
2	167362 1234	1234 167362 772 1234 462 772 310 462 152 310 6 152 2 6 2

2. En esta pregunta desarrollarás un programa para detectar si un usuario está escribiendo sumas bien formadas o está comentiendo errores. Una suma bien formada corresponde a un string `s` que cumple con una de las siguientes dos condiciones:

   - `s` representa un entero positivo (para verificar esto puedes usar `s.isdigit()`).
   - `s` es un string de la forma `(s1+s2)`, donde `s1` y `s2` son sumas bien formadas.

   Por ejemplo, las expresiones `(1+7)`, `125`, `(23+(5+1))` y `((4+12)+(5+(6+7)))` son sumas bien formadas, mientras que las expresiones `(1+(2+4)`, `(a+2)`, `4+` y `3+6` no lo son (notar que `3+6` no es una suma bien formada por la ausencia de paréntesis).

   Tu programa recibirá como input un string, y debe imprimir `True` o `False` dependiendo de si el string representa o no una suma bien formada. A continuación se muestran ejecuciones que ejemplifican cómo debiese funcionar tu programa:

caso	input	output
1	(4+7)	`True`
2	(1	`False`
3	25	`True`
4	(2+(2+(2+2)))	`True`
5	(3+3+3)	`False`
6	((1+7)+(2+65))	`True`
7	(4+5+(123))	`False`
8	(34)	`False`
9	(a+4)	`False`
10	hola	`False`
11	(1+(7+3))	`True`

   Hint: Si el string `s` *no es un número, para verificar si es de la forma* `(s1+s2)` *puedes revisar cada uno de los signos* `+` *que aparecen en* `s`*.*

3. En esta pregunta debes escribir un programa para saber si un número natural se puede construir sumando elementos que pertenecen a una lista de números naturales, considerando que un elemento de la lista se puede sumar tantas veces como se necesite.

   Como input, tu programa recibirá en la primera línea la lista de los elementos separados por coma y sin espacios, la cual denominaremos `L`. En la segunda línea encontrarás el número que se desea sumar, que llamaremos `k`. El output debe ser `True` si es posible obtener el número `k` sumando elementos de `L` y `False` en caso contrario.

   Por ejemplo, si tu programa recibe la lista `L=[5, 3, 9]` y el número `31` entonces debe retornar `True`, pues $31 = 9 + 9 + 5 + 5 + 3$. Por otro lado, si tu programa recibe la lista `L=[5, 9]` y el número `31` entonces debe retornar `False`, pues no existen naturales a y b tales que $5a + 9b = 31$.

   A continuación se muestran dos diálogos que ejemplifican cómo debiese funcionar tu programa:

Caso	Entrada	Salida
1	5,8	
	33	True
2	30,27,51	
	100	False

4. El banco más importante de la región estará transportando objetos valiosos de una sucursal a otra durante esta noche. Su objetivo es transportar al menos una cierta cantidad de dinero en objetos. Sin

embargo, sólo tienen un camión, y no están seguros de si pueden meter objetos en la caja de seguridad de dicho camión para conseguir su objetivo.

Para ayudar al banco, deberás escribir un programa que indique si es posible transportar la cantidad de dinero que desean usando el camión y los objetos disponibles. Debes suponer que la caja de seguridad del camión tiene volumen `L`, que los objetos tienen volúmenes `v1, v2, ..., vn` y precios `d1, d2, ..., dn` respectivamente, y que el banco desdea transportar `D` pesos. Por ejemplo, supongamos que se quiere transportar al menos 35 pesos en objetos, que el volumen del camión es 66, y que hay cuatro objetos cuyos volúmenes son 12, 17, 21 y 32 y cuyos valores son 5, 13, 15 y 16 pesos respectivamente. En tal caso tu programa deberá retornar `True`, pues se puede meter al camión los objetos cuyos volúmenes son 12, 21 y 32, cuyos precios suman 36 pesos. Sin embargo, si cambiamos el volumen del primer objeto por 14, tu programa debiese retornar `False`.

Como primera línea de input tu programa recibirá un número que indica la cantidad mínima de dinero que se desea transportar. En la segunda línea recibirá otro número, que indicará el volumen de la caja de seguridad del camión. Luego, recibirá una cantidad arbitraria de posibles objetos para transportar. Cada objeto será codificado en una línea, la cual contendrá dos números separados por coma y sin espacios. El primero de estos números indica el volumen del objeto y el segundo su valor. A continuación se muestran dos ejemplos de posibles diálogos de tu programa:

caso	input	output

1	`35`
	`66`
	`12,5`
	`17,13`
	`21,15`
	`32,16`
		`True`

2	`10`
	`12`
	`5,5`
	`9,7`
	`6,3`
	`3,2`
		`False`