Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
Primer Semestre de 2015

Exámen de Medio Semestre

Tiempo: 150 minutos – Con apuntes (1 Hoja)

1. Los computadores trabajan con números binarios (compuestos exclusivamente por unos y ceros). Un byte está compuesto por 8 bits, y cada bit puede ser un 1 o un 0. Para transformar un número binario b (de n bits) a decimal se utiliza la siguiente fórmula:

$$d = \sum_{i=0}^{n-1} b_i * 2^i$$

Donde b_i es el i-ésimo bit de b, contados de **derecha a izquierda** y **comenzando desde cero** ($b = b_{n-1}b_{n-2}...b_0$). Por ejemplo: $00010010 = 0\times2^7+0\times2^6+0\times2^5+1\times2^4+0\times2^3+0\times2^2+1\times2^1+0\times2^0 = 18$.

Realiza un programa que pida al usuario un número binario y muestre el valor decimal de cada uno de sus bytes. Por ejemplo:

- Si el usuario ingresa 11101101 (8 dígitos), el programa deberá mostrar 237.
- Si el usuario ingresa 10100101011011011101, el binario se divide en 3 bytes (de derecha a izquierda):

1010	01010110	11011101
10	86	221

... y se debería mostrar el valor decimal de cada uno de ellos por separado: 221, 86 y 10.

Para pedir un número binario utiliza la función `pedir_binario()`, que retorna un `int` con el número binario que ingresó el usuario. **Importante**: No puedes usar ni strings ni listas en este ejercicio.

2. a) Escribe una función divisor(x,i) que entregue el i-ésimo divisor de x (sin incluir a x). Si i está fuera de rango, debe devolver -1. Por ejemplo, los divisores de 12 son 1, 2, 3, 4 y 6, y los divisores de 9 son 1 y 3, por lo que la función entregaría los siguientes valores:

`divisor(12,1)` = 1	`divisor(12,6)` = -1	`divisor(9,3)` = -1
`divisor(12,5)` = 6	`divisor(9,2)` = 3	`divisor(9,0)` = -1

b) Un número abundante es un número para el cual la suma de todos sus divisores (excluido él mismo) suma más que el número.

divisores de 12: 1,2,3,4,6	suma: 16	$16 > 12$	`es_abundante(16)` = `True`
divisores de 10: 1,2,5	suma:8	$8 < 10$	`es_abundante(8)` = `False`
divisores de 30: 1,2,3,5,6,10,15	suma:42	$42 > 30$	`es_abundante(42)` = `True`

Usa la función anterior (divisor) para escribir una función `es_abundante(x)` que entregue `True` si `x` es abundante, `False` si no.

c) Se sabe que cualquier número entero mayor a 20161 se puede escribir como la suma de dos números abundantes. Usa la/s función/es anteriores (`divisor` y/o `es_abundante`) para escribir un programa que pregunte un número al usuario y en caso de ser posible, lo muestre como la suma de dos números abundantes. Si hay dos posibles respuestas (e.g. 60 puede escribirse como 12+48 o como 30+30) puedes mostrar cualquiera de las dos.

Número? 20222	Número? 60	Número? 34
20222 = 12 + 20150	60 = 30 + 30	No es posible

3. Recibes de un amigo código escrito en Python, pero resulta que lo escribió para Python 2.7 y tú ya usas Python 3.4. La mayor diferencia entre ambas versiones es la función `print`: en Python 2.7 va sin paréntesis.

   Escribe un programa que modifique el archivo `juego_python27.py` y lo guarde en un nuevo archivo `juego_python34.py`, en el que cada instrucción print que encuentres se modifique por la versión más nueva de la instrucción. Por supuesto, no debes cambiar las otras instrucciones. Puedes considerar que los strings siempre tienen comillas dobles. Las reglas se resumen de la siguiente manera:

   - Instrucciones print de texto o variables, con cualquier indentación, sin paréntesis → Instrucciones print de texto o variables, con la misma indentación, usando paréntesis. Puedes considerar que cada print solo tendrá un string o una variable.
   - Cualquier instrucción (sea print o no), seguida de un comentario al final de la linea → Si la instrucción es print, se debe agregar los paréntesis correspondientes
   - Otras instrucciones y comentarios de linea completa → no se modifican.

   La siguiente tabla muestra ejemplos de líneas y cómo deben quedar en el archivo final.

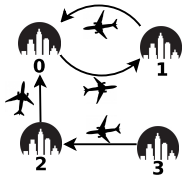
Python 2.7	**Python 3.4**
`print "hola!"`	`print("hola!")`
`  print x`	`  print(x)`
`#esta linea no tiene print, solo comentario`	`#esta linea no tiene print, solo comentario`
`print x #imprimi x con print`	`print(x) #imprimi x con print`
`print "hola"#imprimi "hola"con print`	`print("hola")#imprimi "hola"con print`
`print "comentario es con #"`	`print("comentario es con #")`

4. Una aerolínea representa sus vuelos directos mediante una tabla con unos y ceros. La tabla es un cuadrado tal que cada fila y columna está asociada a una ciudad particular. Si la casilla `(i,j)` tiene un 1, significa que existe un vuelo directo que lleva desde la ciudad `i` hasta la ciudad `j`, mientras que un cero significa que no existe. El orden de las ciudades en filas y columnas es el mismo, es decir, la ciudad asociada a la *fila* `i` es la misma que la asociada a la *columna* `i`. Por ejemplo, considera que la aerolínea llega a 4 ciudades (cuyos índices son: 0, 1, 2 y 3) y tiene los siguientes vuelos: $3 \to 2$, $2 \to 0$, $0 \to 1$ y $1 \to 0$ (figura 1a). Su tabla de vuelos en Python sería la figura 1b, que denominaremos `t`. Implementa las siguientes funciones:

   a) `vuelosSinVuelta(tabla)`: Esta función recibe la tabla de vuelos y retorna una lista con los vuelos sin vuelta directa. Los elementos de la lista retornada deben contener los índices de la ciudad de origen y destino del vuelo. Por ejemplo, el retorno de la función para `t` debería ser `[[2,0],[3,2]]`.

   b) `vuelosConUnaEscala(tabla)`: Esta función recibe la tabla de vuelos y retorna una nueva tabla (de igual dimensión) que tiene un `1` en la casilla `(i,j)` si y solo si puedo ir desde la ciudad `i` a la ciudad `j` en forma directa, o realizando una escala (i.e. puedo ir de `i` a `k`, y luego de `k` a `j`). Por ejemplo, en `t` es posible ir de `3` a `0` haciendo escala en `2`, y de `2` a `1` haciendo escala en `0`. No consideres casos en que `i` sea igual a `j`. La figura 1c muestra el resultado esperado para `t`.

   c) `vuelosConEscala(tabla)`: Esta función recibe la tabla de vuelos y retorna una nueva tabla que tiene un `1` en la casilla `(i,j)` si y solo si puedo ir desde la ciudad `i` a la ciudad `j` en forma directa, o realizando algún número de escalas. Por ejemplo, en `t` es posible ir de `3` a `1` realizando escalas en `2` y `0`. La figura 1d muestra el resultado esperado para `t`. **Hint**: Utilice la función `vuelosConUnaEscala(t)`.



(a)

```
t = [[0,1,0,0],
     [1,0,0,0],
     [1,0,0,0],
     [0,0,1,0]]
```

(b)

```
vuelosConUnaEscala(t)
# Resultado esperado:
# [[0, 1, 0, 0],
#  [1, 0, 0, 0],
#  [1, 1, 0, 0],
#  [1, 0, 1, 0]]
```

(c)

```
vuelosConEscala(t)
# Resultado esperado:
# [[0, 1, 0, 0],
#  [1, 0, 0, 0],
#  [1, 1, 0, 0],
#  [1, 1, 1, 0]]
```

(d)

Figura 1: Ejemplo con 4 ciudades.