Compiladores - LCC
Práctica 0.
Introducción a Tiger

Ejercicio 1

Escribir un "Hola Mundo!" en Tiger.

Ejercicio 2

Las listas de enteros pueden representarse utilizando el siguiente record.

```
lista = {item: int, resto:lista}
```

Implemente las siguientes funciones:

- cons que recibe dos argumentos, un entero y una lista. Devuelve la lista formada por el entero seguido de la lista.
- length que recibe una lista de enteros, y devuelve la longitud de la misma.
- concat que recibe dos listas como argumentos y las concatena.
- snoc que recibe dos argumentos: un entero x y una lista xs. Devuelve la lista xs con el elemento x al final.
- filtra que recibe un entero n y una lista l. Devuelve una lista formada por los elementos de l que no son iguales a n, manteniendo el orden de l.
- isin que recibe dos argumentos: un entero x y una lista xs. Devolviendo verdadero si x aparece al menos una vez en xs.
- remove que recibe dos argumentos: un entero x y una lista xs. Devolviendo el resultado de borrar la primer aparición de x en xs en el caso que $isin(x, xs)$ sea verdadero.
- removeall que recibe dos argumentos similar a `remove` pero que elimina todas las apariciones de x en xs.
- reverse que recibe una lista de enteros, y retorna la lista en orden inverso.
- printlist que toma como argumento una lista de enteros y la muestra por pantalla.

```
let
    type lista = {item:int, resto:lista}
    /* Definir cons          */
    /* Definir length        */
    /* Definir concat        */
    /* Definir snoc          */
    /* Definir filtra        */
    /* Definir isin          */
    /* Definir remove        */
    /* Definir removeall     */
    /* Definir reverse       */
    /* Definir printlist     */
in
    nil
end
```

Ejercicio 3

De forma similar a las listas de enteros del ejercicio anterior, podemos definir a los árboles generales como:

```
let
  /* define a tree */
  type tree = {key : int, children : treelist}
  type treelist = {hd : tree, tl : treelist}
in
   0
end
```

Escribir las siguientes funciones:

- isBin que dado un árbol general responda con verdadero en el caso que sea un árbol binario o falso si no lo es.
- isBComplete que dado un árbol general responda con verdadero en el caso que sea binario completo o falso en cualquier otro caso.
- printInOrder que imprima todos los valores recorriendo el árbol en orden.
- printPosOrder que imprima todos los valores recorriendo el árbol en orden.

Ejercicio 4

Definir las siguientes funciones:

- **maxargs:** dado un *tigerabs.exp* devuelve la máxima cantidad de argumentos con los cuales se ha llamado a la función *print*.
- **cantplus:** dado un *tigerabs.exp*, contar la cantidad de veces que se utiliza la operación binaria de la suma.

Ejercicio 5

Encuentre el AST que corresponde a los siguientes fragmentos de código. Puede usar la primera versión del compilador, comentando la línea
`val _ = findEscape(expr)` en *tigermain.sml*.

a) `a := 10`

b) `for i := 0 to c do print (".")`

c) `f[a+1].data[0]`

d)
```
let
    var f := 10
in
    f(f, f); f
end
```

e) `type lista = {item:int, resto:lista}`

f) `if row[r]=0 & a<b then g(r)`

¿Todos los fragmentos pueden ser parte de un programa válido? ¿Por qué hay un problema al copiar directamente el fragmento *e*? Descomente la línea comentada anteriormente. ¿Qué error detecta ahora el compilador?

Ejercicio 6

Encuentre el código que genera los siguientes ASTs. No tome en cuenta el valor del campo `pos`.

a) `ArrayExp({init = IntExp(5, 0), size = IntExp(10, 0), typ = "a"}, 0)`

b) `VarExp(SubscriptVar(SimpleVar "a", IntExp(7, 0)), 0)`

c) `AssignExp({exp = NilExp 0, var = SimpleVar "a"}, 0)`

Ejercicio 7

Implementar *pretty printers* para los siguientes *datatypes*:

- `EnvEntry`
- `Tipo`

Implementar la función

```
val tabPrint : ('a -> string) * ('b -> string) * ('a, 'b) Tabla -> string
```