T-501-FMAL Programming languages, Practice class 5
Spring 2020

1. Write a decompiler function `decomp : rcode -> texpr` from stack machine code of Expressions2.fs back to target expressions.

   You can disable `RPop`, `RDup`, `RSwap` as instructions since `tcomp` never produces them. But in fact it can be fun to keep them!

   Hint: Consider running stack machine code like `reval` does but keeping in the stack not integers but target expressions instead.

```
type rinstr =
    | RLoad of int                        // load value from runtime environment
    | RNum of int
    | RAdd
    | RSub
    | RMul
    | RPop
    | RDup
    | RSwap

type rcode = rinstr list

type texpr =
    | TVar of int                         // a number rather than a name
    | TNum of int
    | TOp of string * texpr * texpr

> decomp [RNum 3;RLoad 2; RAdd];;
val it : texpr = TOp ("+",TCstI 3,TVar 2)          // 3 + x2
> decomp [RNum 3;RLoad 2; RDup; RMul; RAdd];;
val it : texpr = TOp ("+",TCstI 3,TOp ("*",TVar 2,TVar 2))
                                                     // 3 + x2 * x2
> decomp [RNum 3;RLoad 2; RDup; RLoad 0; RSwap; RMul; RAdd; RAdd];;
val it : texpr =
  TOp ("+",TCstI 3,TOp ("+",TVar 2,TOp ("*",TVar 0,TVar 2)))
                                                     // 3 + (x2 + x0 * x2)
```

2. (i) Define a version of the type of `expr` from Expressions3.fs where a let can define multiple local variables at once. In concrete syntax, such a multi-let expression would have the form `let x1 = e1, ..., xn = en in e`.

   (ii) We can think of two different meanings for the multi-let construct.

   The first is that the right-hand sides `e1, ..., en` have only the variables defined outside of the multi-let in their scopes ("parallel multi-let").

   The second is that each next right-hand side `ei` has also the variables `x1, ..., x{i-1}` from the preceding definitions also its scope ("sequential multi-let").

   Extend `eval : expr -> envir -> int` accordingly in two ways.

   The expression `let x = 1 in let x = x + 2, y = x * 2 in x * y` should evaluate to 6 in the parallel and to 18 in the sequential intepretation of multi-let.

   (iii) (This one is meant as a paper-and-pencil problem.) How can one express parallel and sequential multi-let in terms of the simple let that only introduces one local variable?

   Hint: For parallel multi-let you need tuples.

3. (i) Write a function `removeBlockComm : char list -> char list` that takes the list-of-characters representation of a string and removes from it sections between matching `(*` and `*)` ("block comments"). Comments can be nested and `(*` and `*)` must pair up properly.

   If a started comment is unfinished by the end of the string, an exception should be raised.

An exception should be raised also if `*)` occurs before any `(*`.

Hint: You need to keep track of how deep you are in nested comments. Level 0 corresponds to being in code, level 1 means being inside a block comment, level 2 inside a block comment which is inside another block comment etc.

You can turn your function into a function `string -> string` using the `stringVersion` below that can wrap it between converters from a string to a list of characters and the other way around.

```
let chars2String (cs : char list) : string =
    List.fold (fun s c -> s + string c) "" cs

let string2Chars (s : string) : char list =
    let rec helper cs i =
        if i = 0 then cs else let i = i - 1 in helper (s.[i] :: cs) i
    helper [] (String.length s)

let stringVersion f s = chars2String (f (string2Chars s))
```

```
> stringVersion removeBlockComm "abc(*de*)fgh";;
val it : string = "abcfgh"
> stringVersion removeBlockComm "abc(*d(*01*)23e*)fgh";;
val it : string = "abcfgh"
> stringVersion removeBlockComm "abc(*d(*01*)23efgh";;
System.Exception: unfinished comment(s)
> stringVersion removeBlockComm "abc*)d(*01*)23efgh";;
System.Exception: unstarted comment
> stringVersion removeBlockComm "abc(*de*)fgh(*i*)kl";;
val it : string = "abcfghkl"
```

(ii) Write a function `removeLineComm : char list -> char list` that takes the list-of-characters representation of a string and removes from it sections between `//` and the nearest following `\n` (newline) ("line comments"). The `\n` itself must be kept!

Hint: Here levels are not needed. You just have to keep track of whether you are inside a comment or not.

```
> stringVersion removeLineComm "abc//de\nfgh";;
val it : string = "abc
fgh"
> stringVersion removeLineComm "abc//d//e\nfgh";;
val it : string = "abc
fgh"
> stringVersion removeLineComm "abc//de\nfg//h";;
val it : string = "abc
fg"
```

(iii) Write a function `removeComments : char list -> char list` that copes with both block and line comments.

`//` and `\n` that are inside block comments should not count as start and end markers of line comments.

And `(*` and `*)` in line comments should not count as start and end markers of block comments.

```
> stringVersion removeComments "abc(*de*)fgh";;
val it : string = "abcfgh"
> stringVersion removeComments "abc//de\nfgh";;
val it : string = "abc
fgh"
> stringVersion removeComments "abc(*d//e*)f\ngh";;
val it : string = "abcf
gh"                                  // notice that f has been kept
> stringVersion removeComments "abc//d(*e\ng*)h";;
System.Exception: unstarted block comment
```