CS 381 Homework 5 – Types

You may work in groups of up to three people on this assignment. You will need to sign-up for a group in Canvas at least 24 hours before the due date of the assignment.

A Rank-Based Type System for a Stack Language 2

We extend Stack Language 2 from Homework 4 by adding the following three operations

- DEC – decrements the topmost element (must be Int) on the stack
- SWAP exchanges the two topmost elements on the stack
- POP k – pops k elements off the stack

The abstract syntax of the extended language is as follows.

```
type Prog = [Cmd]
data Cmd  = LDI Int  | LDB Bool | LEQ  | ADD  | MULT  | DUP
          | IFELSE Prog Prog | DEC | SWAP | POP Int
          deriving Show
```

Again the types of the values on the Stack are Bool and Int. You should dynamically type check all operations during run time. Instead of using Either as in homework 4 you can define

```
data Val =  I Int | B Bool
data Stack = [Val]
```

You can assume that the initial contents of the stack will be in the form [I 5, B True] etc. Use the data types defined in HW5types.hs.

We will also define a rank type system for this language that assigns ranks to stacks and operations to ensure that a program does not result in a rank mismatch. A rank mismatch occurs when there are not enough elements on the stack to perform an operation in a program. For example, executing the program p1 = [ADD] with the stack = [I 1] will result in a rank mismatch and would return RankError.

The rank of a stack is given by the number of its elements. The rank of a single stack operation is given by a pair of numbers (n, m) where n indicates the number of elements operation takes off the top of the stack and m is the number of elements the operation puts onto the stack. The rank for a stack program is defined to be the rank of the stack that would be obtained if the program was run on an empty stack. The rank of a stack program p1 run with stack s1 is the rank of the stack s1 after the execution of program p1. A rank error occurs in a stack program when an operation with rank (n, m) is executed on a stack with rank $k < n$. For example:

p2 = [LDI 5, LDI 10, ADD] has rank 1 when run with the stack s1 = [] and rank 2 when run with s2 = [I 5].

Use the following types to represent the operation ranks.

```
type Rank        = Int
type CmdRank = (Int, Int)
```

Define a function rankC that maps each stack operation to its rank.

```
rankC :: Cmd → CmdRank
```

For example:

```
rankC (ADD)   = (2, 1)   since ADD removes two values from the stack and places one on.
rankC (DEC)   = (1, 1)    since DEC removes one value  from the stack and places one on
…..
```

The rank of Cmd (IFELSE Prog1 Prog2) will need to be calculated differently. Since we do not know which branch is executed we will need to verify that there is enough stack to execute either branch on the current stack contents. If either Prog1 or Prog2 produces a rank error then the entire IFELSE produces an error. If the IFELSE command does not produce a rank error then set the rank of the current stack used for subsequent command to the min {rank(Prog1) , rank (Prog2)}. For example, p2 = [IFELSE [ADD] [LDI 2, DUP]] ran with stack s2 = [B True, I 5] of rank 2 would result in a rank error. However, if p2 ran with stack s3 = [B False, I 10] the result would be [I 2, I 2, I 10]. Since we don't know which branch the IFELSE statement will take we will assign a Rank Error if the stack is rank 2.

Define a function rankP that computes the rank of a program when ran with a stack of rank r. The Maybe data type is used to capture rank error, that is, a program that contains a rank error should be mapped to Nothing whereas ranks of other programs are wrapped by the Just constructor.

```
rankP :: Prog → Rank → Maybe Rank
```

Hint: You may need an auxiliary function rank :: Prog → Rank → Maybe Rank and define rankP using rank.

Define a function run for evaluating a program with a given stack

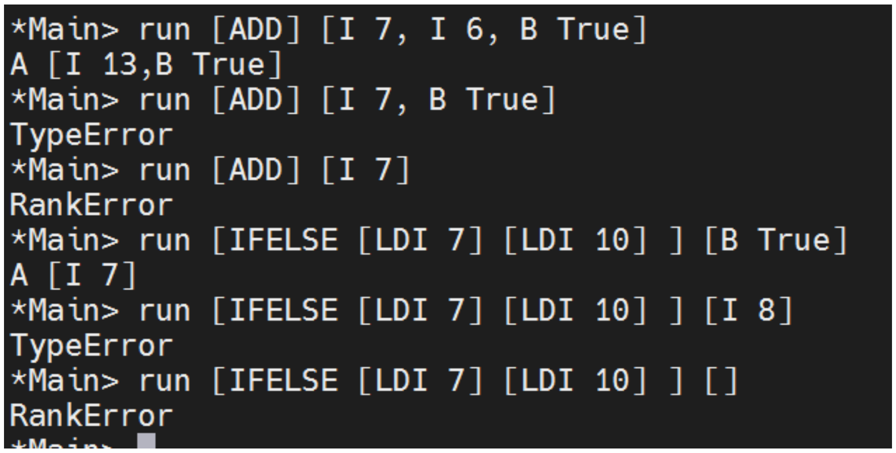
```
run :: Prog →Stack→Results
```

where Results is defined as

```
data Result = A Stack | RankError | TypeError
```

The run function should that first call the function rankP to check whether the stack program is rank correct. If the program with a given stack is rank correct then call semCmd to execute the program otherwise produce a RankError. If a type error occurs during execution of a program return a Result of TypeError. If the program runs without rank or type errors return a Result of A Stack.

Below are some simple examples.



```
*Main> run [ADD] [I 7, I 6, B True]
A [I 13,B True]
*Main> run [ADD] [I 7, B True]
TypeError
*Main> run [ADD] [I 7]
RankError
*Main> run [IFELSE [LDI 7] [LDI 10] ] [B True]
A [I 7]
*Main> run [IFELSE [LDI 7] [LDI 10] ] [I 8]
TypeError
*Main> run [IFELSE [LDI 7] [LDI 10] ] []
RankError
```

Submission Guidelines:

- Submit a single file named HW5sol.hs to Canvas.
- If you are working in a group you must join the group before submitting an assignment.
- Your code should import HW5types and contain detailed comments including the names of all students in a group.
- You should "comment out" any test cases you used to test your code
- Verify your program using hw5verifier.hs.