Programming D

SML Practice Excercise

Zehady Abdullah Khan

Student ID: 09B12702

Email: a_khanzehady@yahoo.com

Exercise 3(課題 3)

Basically the function compute in Exercise has the same structure as of exercise 2. I have added if else statements for 3 more operators "-" , "*","/" to perform subtraction, multiplication and division. The main operation is actually happening inside the **"in"** block where I have written

in

(v1 - v2,t2) or (v1 * v2,t2) or (v1 div v2,t2) for corresponding operation

end

I have checked my program with the following executions(* has been used to comment out in the main source codes):

(* **Execution Result**
* val compute = fn : string -> int
* - compute "* 4 2";
* val it = 8 : int
* - compute "/ 9 3";
* val it = 3 : int
* - compute "* - 6 3 / 16 4";
* val it = 12 : int
*)

Exercise 4(課題 4)

In exercise 4, it is asked to handle the brackets. I have written my SML codes so that it will work both with and without brackets for the function compute.

I have come up with 3 solutions. Only the 1st solution for the exercise 4 in the file "09B12702 Kadai 3~5.sml" has been written according to the function definition mentioned below. The 2nd and 3rd solution was done experimentally; actually just to have fun with SML syntax.

<EXP> ::= 非負整数 | <COMP>
<COMP> ::= "(""+"<EXP><EXP>")" | "(""-"<EXP><EXP>")"| "(""*"<EXP><EXP>")" | "(""/"<EXP><EXP>")"

In the 1st solution the bracket is being handled in the COMP function as it is defined above. The goal of EXP function is returning a tuple with a non-negative integer value and the string list that is left behind. In case it is not a integer after checking with the "isInt" function, it hands over the list to the COMP function. In the COMP function firstly it is being checked whether left

parentheses “(” has been used or not. If not used, then it simply follows same rule as of exercise 3. Otherwise, it gets into the if else statement blocks where the operator gets checked with another if else block, then finally gets computed in the let in end block according to the operator. For example,

```
if hd t = "+" then
    let
        val (v1,t1) = EXP (tl t)    (* tail the tail t is handed over*)
        val (v2,t2) = EXP t1
        val (v3,t3) = (v1 + v2,t2)
    in
        if hd t3 = ")" then  (* right parentheses check*)
            (v3,tl t3)
        else raise SyntaxError
    end
```

And before returning the result of the operation it is being checked whether the right parentheses “)” has been used or not. If not used, there occurs SyntaxError defined in lib.sml.

In the 2nd solution the brackets are being handled in the EXP function.

In the 3rd solution I have checked both the left and right parentheses in both of the functions EXP and COMP.

Apart from these differences, the 3 solutions are almost same and they produced the same result with the following executions.

```
(******Execution Result***********)
(* - compute "(+ 1 2)";
* val it = 3 : int
* - compute "+ 1 2";
* val it = 3 : int
* - compute "(+ * 1 2 (/ 8 - 3 2))";
* val it = 10 : int
* - compute "(+ ( * 1 2) (/8 (- 3 2)))";
* val it = 10 : int
* - compute "+ * 1 2 / 8 - 3 2";
* val it = 10 : int *)
```

Exercise 5(課題 5)

In this exercise EXP function is capable of handling factorials and Fibonacci sequences. It calls the function FUNC to perform those. I have also written the code so that it is possible to compute both with or without using parentheses “(“,“)”. Inside the FUNC function after checking the left parentheses, it is getting checked that whether the head of the tail t is “fact” or “fibo”. If it is true, then in the “let in end” block factorial value or element of fibanacci sequence is computed by calling the function “fact” or “fibo” defined in lib.sml. For example in the case of the factorial operation

```
if (hd t) = "fact" then
    let
      val (v1,t1) = EXP (tl t)
      val (factV,t2) = (fact v1,t1)  (*factorial getting computed*)
```

```
in
  if (hd t2) = ")" then  (*right parentheses check*)
        (factV,tl t2)
  else raise SyntaxError
end
```

If the beginning left parentheses is not used ,then right parentheses check statements are excluded.

```
(* Execution Results
* - compute "fibo 5";
* val it = 8 : int
* - compute "fact 5";
* val it = 120 : int
* - compute "+ fact fibo 5 - 9 8";
* val it = 40321 : int
* - compute "(+ (fact (fibo 5)) (- 9 8))";
* val it = 40321 : int
* - compute "(+ (fibo 3) 9)";
* val it = 12 : int
* - compute "(/ (fact (fibo 3)) (* (- (fibo 3) 9) (fact 4)))";
* val it = ~1 : int
*
*)*)
```

Exercise 6(課題 6)

Firstly I have defined the findVal function . The argument of findVal function is a list of (string*int) tuples. So I have used tuple (c,v) to get the header where the c will take the string value (those are actually the characters in the formula to operate with)and v will take the integer value.

```
let  val (c,v) = h  in  if s = c then v else findValue s end
```

Later if it matches, the function returns the integer value for the character , otherwise recursively call itself taking the tail t in the arguments.

Compute function now has one more argument mapL . The EXP function checks whether the head of the list h is alphabetic or not by using "isAlp" function.If it is an alphabet it calls the findVal function to get its value from the mapL list.

if isAlp h then (findValue h mapL,t)

Structurally this function otherwise is similar to the function of exercise 5.

```
(* execution result*
*
* - compute "(+ (* 10 a) (* b c))" [("a",1),("b",2),("c",3)];
* val it = 16 : int
* - findValue "b" [("a",1),("b",2)];
* val it = 2 : int
* - compute "(/ (* (fibo (* (fact a) 2)) (- b c)) a)" [("a",1),("b",2),("c",3)];
* val it = ~2 : int
```

)))*)*)*)

Extra 1(拡張課題 1)

Firstly I have defined “toStr” function which converts integer to string(Alternative to the SML prebuilt function Int.toStr(a:int)). For debugging I have created printList function to print Lists though during finalization I have deleted all the printList function call from compute function.

This function is also written in a way so that both with bracket or without bracket compute arguments will work.

To perform operation on more than two values in the compute function, the main modification is done inside the FUNC function. Firstly parentheses has been checked in the same way described previously. Then the desired operation is done and stored in a tuple. Then it is being checked whether the tail t3 is nil or not . If nil, then we are done, this is the same as previous exercise for two operands. Otherwise a new list is being created where the first two values for an operation is being replaced by the value created after doing the operation on only those first two values and the left is the same . Then this new list is handed over to COMP again. It is like doing operation on two values again and again.For example,

```
if hd t = "+" then
                    let
                        val (v1,t1) = EXP (tl t)
                        val (v2,t2) = EXP t1
                        val (v3,t3) = (v1 + v2,t2)
                    in
                        if t3 = [] then
                          raise SyntaxError
                        else if (hd t3) = ")" then
                          (v3,tl t3)
                        else if (hd t3) = "(" orelse isInt (hd t3) orelse isAlp
(hd t3) orelse (hd t3) = "fact" orelse (hd t3) = "fibo" orelse (hd t3) = "+" orelse
(hd t3) = "-" orelse (hd t3) = "*" orelse (hd t3) = "/" then
                          let
                              val lst = "("::"+"::toStr(v3)::t3   (*the new
list*)
                              val (v4,t4) = COMP lst (* recursive call to
COMP *)
                          in
                              (v4,t4)
                          end
                        else raise SyntaxError
                    end


(*execution result
*
* - compute "(+ 10 2)" [];
```

```
* val it = 16 : int
* - findValue "b" [("a",1),("b",2)];
* val it = 2 : int
* - compute "(/ (* (fibo (* (fact a) 2)) (- b c)) a)" [("a",1),("b",2),("c",3)];
* val it = ~2 : int
* - compute "(/ (* (fibo (* (+ (fact a) b c) 2)) (- b c)) a)" [("a",1),("b",2),("c",3)];
* val it = 5 : int
* - compute "(+ 3 4 (* 1 a b) 7 c d e (fact 4) (/ 18 6 3) (fibo 2) (- 10 2 8))"
[("a",3),("b",4),("c",1),("d",5),("e",8)];
* val it = 67 : int
* - compute "+ 3 4 7 c d e / 18 6 3" [("a",3),("b",4),("c",1),("d",5),("e",8)];
* val it = 28 : int
* - compute "+ 3 4 7 c d e / 18 6 3" [("a",3),("b",4),("c",1),("d",5),("e",8)];
* val it = 29 : int
* - compute "(+ 3 4 7 c d e (fact 4) (/ 18 6 3) (fibo 2) (- 10 2 8))"
[("a",3),("b",4),("c",1),("d",5),("e",8)];
* val it = 55 : int
* - compute "+ 3 4 7 c d e (/ 18 6 3) (- 10 2 8)"
[("a",3),("b",4),("c",1),("d",5),("e",8)];
* val it = 29 : int
* - compute "+ 3 4 7 c d e (/ 18 6 3) (- 10 2 3)"
[("a",3),("b",4),("c",1),("d",5),("e",8)];
* val it = 34 : int
* - compute "(+ 1 2 3 (* a b c))" [("a",3),("b",5),("c",2)];
* val it = 36 : int
*
*)*)*)*)*)*)*)
```

Extra 2(拡張課題 2)

I have modified the function definition for more general use :

<EXP> ::= <TERM>{{"+"|"-"|"*"|"/"}<TERM>}*

<TERM> ::= 非負整数 | 英字列 | "("<EXP>")"

In the EXP function there is two terms . Between those two terms any arithmetic operation is possible. And each term follows the definition of the TERM function

```
let
    val (term1,t1) = TERM (h::t)  (*the first term *)
        in
          if t1 = [] then  ()
              (term1,t1)  (*if nil return the term1 only)
    else if (hd t1) = "+" then
              let
                val (term2,t2) = TERM (tl t1) (* otherwise get the 2nd term*)
              in
                if t2 = [] then
                    (term1+term2,t2) (*do the operation on term1 and term2*)
```

```
                                    else if (hd t2) = "+" orelse (hd t2) = "-" orelse (hd t2) = "*" orelse (hd t2) = "/" then
            (*Otherwise check whether there is more operations to handle*)
                                        let
                                            val (res,tt) = EXP (Int.toString(term2)::t2)
        (*First do operations for term2 and the list t2 and store it in the res*)
                                                in
                                                  (term1 + res ,tt)
            (*then do the operation between term1 and res*)
            (*So in my code the operands in the farthest distance from the beginning are operated first*)
                                                end
                                      else
                                                (term1+term2,t2)
                                    end
```

The basic structure for other operations are quite the same as above.

```
(* execution result *) (*
* -compute "((1+a)-(b+3+c+2))" [("a",2),("b",3),("c",4)];
* val it = ~9 : int
* -compute "(b+3+c+2)" [("a",2),("b",3),("c",4)];
* val it = 12 : int
* -compute "((1+a)*(b+3+c+2))" [("a",2),("b",3),("c",4)];
* val it = 36 : int
* -compute "(1+a)*(b+3*(c+2))" [("a",2),("b",3),("c",4)];
* val it = 63 : int
* -compute "(b+(3*(c+2)))" [("a",2),("b",3),("c",4)];
* val it = 21 : int
* -compute "(1+2)*(3+15/(1+2))" [];
* val it = 45 : int
* -compute "(1+2)*(3+15/(1+2))" [];
* val it = 24 : int
*)
```

Extra 3(拡張課題 3)

I have modified the cisOpr function

```
fun cisOpr c = c = #"+" orelse c = #"-" orelse c = #"*" orelse c = #"/"
        orelse c = #"(" orelse c = #")" orelse c = #"^" orelse c = #"%" orelse c = #"=" orelse c = #"|" orelse c = #"<" orelse c = #">";
```

I have added modulus operation and power operation. I have created my own power function.

I have written two functions compute and compute_M . The first one is for prefix operation (^ 2 3) and the 2nd one is for infix operation (2 % 3)

I have also implemented the condition statements < , > , = for the function compute_M. I was very close to implement the if definition. But because of time shortage, I couldn't submit it.

(* **execution result**
* - compute_M "((3^2)+5*(10%7))" [];
* val it = 24 : int
* - compute_M "((3^a)+5*b*(10%c))" [("a",2),("b",3),("c",4)];
* val it = 39 : int
* - compute_M "2=3" [];
* val it = 0 : int
* - compute_M "2<3" [];
* val it = 1 : int
* compute_M "(a>(b=c))" [("a",5),("b",3),("c",2)];
* val it = 1 : int
*)