COSC 1030 Introduction to Computer Science II
Assignment 3: Recursion
Solution

---

4) Finding the Connection Between Recursive Pictures and Recursive Programs.

```
                                                  procedure RecC(
               procedure RecA(  procedure RecB(   m:int )          procedure RecD( m:int )
               m:int )          m:int )               if m=1 then      if m=1 then
                   if m=1 then      if m=1 then           put              put "Bye"
                       put "Bye"        put "Bye" "Bye"               else
      Prog         else             else              else                 put "Hi"
                       put "Hi"         put "Hi"          put "Hi"         RecD (m div 2)
                       RecA(m-1)        RecB(m-1)         RecC(m           RecD((m+1) div 2)
                   end if               RecB(m-1) div 2)               end if
               end RecA             end if            end if           end RecD
                                end RecB
                                                  end RecC
      1) Rec ~ rec1             rec2              rec1             rec2
      20 m ~   n                n                 lenght of box    lenght of box
      3) B(m)
      ~        number of leaves number of leaves  number of leaves number of leaves
      4)                                          B(1) = 1         B(1) = 1
      Recur.   B(1) = 1         B(1) = 1          B(m) =           B(m) =
      Relation B(m) = B(m-1)    B(m) = 2* B(m-1)  B(floor(m/2))    B(floor(m/2))+B(ceil(m/2
)
      5)
      Closed   B(m) = 1         B(m) = 2^(m-1)    B(m) = 1         B(m) = m
                                2^(1-1) = 1
      6) Check 1 = 1            2^([m]-1) = 2 *   1 = 1            1 = 1
               1 = 1                              1 = 1            m = floor(m/2)+ceil(m/2)
                                2^([m-1]-1)
      7) H(m)  number of interalnumber of interal number of
      ~        nodes            nodes             interal nodes    number of interal nodes
      8)                        H(1) = 0          H(1) = 0         H(1) = 0
      Recur.   H(1) = 0         H(m) = 1 + 2*     H(m) = 1 +       H(m) = 1 +
      Relation H(m) = 1 + H(m-1)H(m-1)            H(floor(m/2))    H(floor(m/2))+H(ceil(m/2
)
      9) H(m)                   H(m) = 2^(m-1) -  H(m) =
      =        H(m) = m-1       1                 floor(log_2(m))  H(m) = m - 1
                                2^(1-1) - 1 = 0
      10)      1-1 = 0          2^([m]-1) - 1 =   log(1) = 0       1 = 1
      Check    [m]-1 = 1 +      1 + 2             log([m]) = 1 +   m = 1 + [floor(m/2)-1] +
               [m-1]-1          *[2^([m-1]-1) -   log([m/2])       [ceil(m/2)-1]
                                1]
```

Note that (m div 2) = floor(m/2), ((m+1) div 2) = ceiling(m/2), and (m div 2) + ((m+1) div 2) = m.

---

6) For the complete code, see ass4.

* Question: Sum and Mult are designed to finalize the input polynomials f and g. Why is this done? Could the sum be made, without copying records, while keeping f and g pointing at the same polynomials? What are the dangers of this?

  Answered in handout for ass4.

---

7) Formal Proof of Correctness

The following is an outline of a proof of correctness for a recursive



program. Fill in the missing details required to prove the correctness of your program EvalRec.

For every integer k >=0, let S(k) be the formal statement that "The routine EvalRec returns the correct value for every value of x, y, and z and for every polynomial (pointed to by n) that has k nodes in it."

The overall goal is to prove that EvalRec returns the correct answer on every input. It is sufficient to prove that "forall k>=0 S(k)".

The method is proof by Strong Induction:
i.e. prove that for every k>=0, [S(0) and ... and S(k-1)] => S(k)
Note for k=0, the assumption is empty and you must prove S(0).

Let k be an arbitrary integer >= 0. Assume by way of strong induction that [S(0) and ... and S(k-1)].

SubGoal: Use the assumption to prove S(k).
Let x', y', and z' be arbitrary integers and let n' point at an arbitrary polynomial with k nodes.

SubSubGoal: Prove that EvalRec( x',y',z',n') returns the correct answer.
Because we have limited information about k, x', y', z', and n', it is best to brake the different possibilities into cases.

If an undefined variable is found oot will give an error message. Hence, assume all values are defined.

Case k=0 ie n=nil) The polynomial is not initialize. In this case, the program prints and error mesage as required by the post-conditions.

Case k=1) A polynomial consisting of single node will have n->left=nil. Hence, the Operand() test will succeed. If this operand is "x", "y", "z", or a real, then the progam returns the appropriate value. Otherwise, an appropriate error message is printed.

Case k>1 and n->left~=nil) Because n->left~=nil, the Operand() test will fail and the code will recurse on EvalRec(n->left) and EvalRec(n->right). These sub-trees will have strictly fewer than k nodes, because they do not include the root. Hence by the induction assumption, we know that EvalRec returns the correct evaluation of these sub-polynomials (or gives the correct error message if the sub-polynomial is not valid). To be valid, the given polynomial must be the +, -, *, or / of these two sub-polynomials. In these cases, the program returns the correct +, -, *, or / of the sub-anwers. If the root is other than +, -, *, or /, then the appropriate error message is given.

Case k>1 and n->left=nil) For there to be more then one node, n->right must point at them. This is not a valid polynomial. Ideally an error message should be given. In the program, the Operand() tests will pass. If the root node is "x", "y", "z", or a real, then give an answer, not noticing that n->right is pointing at something. Otherwise, an appropriate error message is printed.

Because all the possibilities have been proven, the SubSubGoal is complete.

Because we proved the SubGoal for arbitrary values of x, y, z, and n, it must be true for all values. This completes the SubGoal.

Because we assumed [S(0) and ... and S(k-1)] and then proved S(k), it follows that [S(0) and ... and S(k-1)] => S(k). Because we did this for an arbitrary value of k>=0, it must be true for all values of k>=0.

By the method of Strong Induction, we can conclude that "forall k>=0 S(k)", i.e. EvalRec works.

---

8) Tracing the execution of the recursive program PrettyPut

```
      PrettyPut  ________________________________
                | no parameters                  |
                | puts the polynomial pointed    |
                | to by poly                     |
                |________________________________|
                              |
      PPutRec   ______________V_______________
                |dir = root                     |
                |prefix =                       |
                | ""                            |
                |print:                         |
                |       |-- 1                   |
                | -- * -|                       |
                |       |           |-- 2       |
                |       |     |- * -|           |
                |       |     |     |     |-- 3 |
                |       |     |     |- + -|     |
                |       |     |           |-- 4 |
                |       |- + -|                 |
                |             |-- 5             |
                |_______________________________|
                    |                      |
____________________V_______   ____________V_________________
|dir = left                 |  |dir = right                  |
|prefix =                   |  |prefix =                     |
| bbbbbb                    |  | bbbbbb                      |
|print:                     |  |print:                       |
|       |-- 1               |  |       |           |-- 2     |
|___________________________|  |       |     |- * -|         |
                               |       |     |     |     |-- 3
                               |       |     |     |- + -|   |
                               |       |     |           |-- 4
                               |       |- + -|               |
                               |             |-- 5           |
                               |_____________________________|
                                        |                |
                                ________|                |
______________________________V____  ____________________V____
|dir = left                        |  |dir = right             |
|prefix =                          |  |prefix =                |
| bbbbbb|bbbbb                     |  | bbbbbbbbbbbb           |
|print:                            |  |print:                  |
|       |           |-- 2          |  |             |-- 5      |
|       |     |- * -|              |  |________________________|
|       |     |     |     |-- 3    |
|       |     |     |- + -|        |
|       |     |           |-- 4    |
|__________________________________|
    |                       |
    |                       |__________
____V_______________________   ____V_______________________
|dir = left                 |  |dir = right                 |
|prefix =                   |  |prefix =                    |
```



```
| bbbbbb|bbbbbbbbbbb           |  | bbbbbb|bbbbb|bbbbb            |
|print:                        |  |print:                         |
|       |           |-- 2      |  |       |     |     |     |-- 3 |
|______________________________|  |       |     |     |- + -|     |
                                  |       |     |           |-- 4 |
                                  |_______________________________|
                                       |                      |
                              _________|                      |
_____________________________V____  __________________________V____
|dir = left                      |  |dir = right                    |
|prefix =                        |  |prefix =                       |
| bbbbbb|bbbbb|bbbbb|bbbbb       |  | bbbbbb|bbbbb|bbbbbbbbbbb      |
|print:                          |  |print:                         |
|       |     |     |     |-- 3  |  |       |     |           |-- 4|
|________________________________|  |_________________________________|
```

---