COSC 1030 Introduction to Computer Science II
Assignment 3: Recursion
Due: Wednesday, March 19, 1997, 10:30 am (SHARP)

1.1) Learning Objectives
* Recursion
* More on Pointers
* More on ADTs and Classes
* Formal Proofs

1.2) What is given
* recPic.dem : A program that draws pictures recursively, a tutorial for it, and help for it.
* polyTree.tu : Most of the code for a class that handles polynomials represented as a binary tree. One of your tasks is to complete the remaining routines.
* poly2.tu : You modify this file to complete the definition of PolyTree.
* polyTree.dem : A Demonstration program that you can use to test your program
* sample.in : An example input file that you can use with polyTree.dem
* sample.out : The output from the sample input
* rundemo : A script for running the polyTree.dem and producing a file to hand in.
* Cover page To be the first page handed in.

1.3) What to hand in:

When working with a partner, each person should attempt to solve every problem on his/her own before comparing results and writing up the solution together. It is crucial that you LEARN this material.

A report for this assignment documents what you did for tasks in this specification. The order of the sections and their contents is the following.

* Task 3 -- A print out of the reproduced and the self created figures.
* Task 4 -- Answers to the 4 sets of 10 questions. You do not need to answer them one after the other. Organize the data in a concise easy to read way such as a table.
* Task 6 -- Completed poly2.tu. including complete specifications, and comments. Use OOT automatic paragraphing style. Clearly let us know about any incomplete or buggy programs.
* Task 6 -- The answers to the questions.
* Task 7 -- The full proof, including the parts given in Task 7.
* Task 8 -- The trace diagram. Hand drawn is acceptable.
* Task 9 -- rundemo.out -- the output from the shell script rundemo.
* Task 10 -- Your no input demonstration program, polyTreeAssert.dem and a script showing its execution.

---

2) Learning about Recursive Figures
* Follow the tutorial for the program recPic.dem.
  - o The tutorial is linked to this page.
  - o In your home directory, make a directory called recPic,
  - o Execute oot &
  - o In the OOT Directory View, click on Chdir:%oot
  - o Double click on 1030 , double click on recPic.dem, and run.

---

3) Drawing Recursive Figures
* Within the program recPic, the <Reproduce> menu shows you thee recursive pictures. You are to reproduce them.
* If you are working, on your own design two of your own recursive figures. Groups of two design three. These will be marked not on the number of lines drawn, but on originality and on the extent to which

they demonstrate an understanding of recursion.
* To hand in a recursive picture:
  - o select <Mode:Hand In>. Draw a single copy of your recursive picture.
  - o In your xterm window type the command "snapwin filename.ps"
  - o Click on the window containing the picture to be saved.
  - o Use "ghostview filename.ps" to check that you captured the picture.

---

4) Finding the Connection Between Recursive Pictures and Recursive Programs.
* Consider the following four recursive programs.

```
procedure RecA( m:int )          procedure RecB( m:int )
    if m=1 then                      if m=1 then
        put "Bye"                        put "Bye"
    else                             else
        put "Hi"                         put "Hi"
        RecA(m-1)                        RecB(m-1)
    end if                               RecB(m-1)
end RecA                             end if
                                 end RecB

procedure RecC( m:int )          procedure RecD( m:int )
    if m=1 then                      if m=1 then
        put "Bye"                        put "Bye"
    else                             else
        put "Hi"                         put "Hi"
        RecC(m div 2)                    RecD (m div 2)
    end if                               RecD((m+1) div 2)
end RecC                             end if
                                 end RecD
```

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

* Use the RecPicture program to draw the <Example> figures "rec1" and "rec2". Then, separately for EACH of the above four programs answer the following questions.

1) Which of the two recursive pictures "rec1" or "rec2" corresponds best to this recursive program.

2) What in this picture corresponds to the program parameter m?

For all m>=1, let B(m) be the number of times "Bye" is printed and H(m) be the number of times "Hi" is printed by the program on input m.

3) What does B(m) relate to in the picture?

4) Give a recurrence relation for B(m), i.e. give B(1) and B(m) in terms of B(m') for smaller m'.

5) Give a closed expression for B(m), i.e. expressed in terms of m.

6) Show that your closed form in (5) satisfies the recurrence relation in (4).

7-10) Repeat (3-6) for H(m).

---

5) Study the recursive code

We provide the bulk of the code that handles multivariate polynomials in polyTree.tu. Study the specifications and the program text.

In particular note how the recursive programs FinalizeRec and PPutRec solve an instance of the problem by first solving two smaller instances (namely the left and right sub-polynomials) and combining these solutions into a solution for the original instance.

---

6) Write code for the programs Copy, Eval and Put.

The specifications for these routines are found at the top of polyTree.tu. For Put the hard part is to print the minimum number of brackets for a polynomial.

Questions to answer:

- o The operation Operator sets its input parameters, f and g, to nil instead of calling finalize. Why?
- o Could the operation be made without copying nodes while keeping f and g pointing at the same polynomials? What is the danger of this?

---

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 break the different possibilities into cases.

Case 1) k=0
Case 2) ...
Your task is to prove the subsubGoal for enough cases to cover all possibilities. Be explicitly clear where and how you use the assumption [S(0) and ... and S(k-1)].

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 a recursive program

Study the code provided for the routine PrettyPrint. Trace the execution of this program with the following polynomial as input:

```
          |-- 1
      --*-|
          |       |-- 2
          |   |-*-|
          |   |   |   |-- 3
          |   |   |-+-|
          |   |       |-- 4
          |-+-|
              |-- 5
```

Each time a routine (PrettyPrint or PPrintRec) is called, draw a box. This is referred to as an "instance" of the routine. (The block of memory storing the local variables for this instance is called a "stack frame".) Connect the instances with arrows from the calling to the called instance. Because each instance potentially calls two other instances, the trace diagram will look like a tree. Include with each such instance:

* The input values prefix and dir for this instance.
* The complete output of this instance. This includes the output of all called subprograms, not just the put statement for the instance.

---

9) Running the Demo Program

Create your own example demonstration input file in "mysample.in". It should be approximately the same length as sample.in.

Copy and execute the shell script program rundemo. Make sure you have set its protection mode to executable. The script runs the demonstration program polyTree.dem on sample.in and outputs a comparision of your output with sample.out. It then runs polyTree.dem on your test file "mysample.in". The output is put into the file "rundemo.out".

---

10) Assertion Testing

Write a short, no more than one page, no input demonstration program, polyTreeAssert.dem. As done in assignment 1, if no bugs are found, then it should only output a single statement "Example assertions are true."

```
p->InitVal("x")
assert( p->Eval(x,y,z) = x )

s := f->Eval(x,y,z) + g->Eval(x,y,z)
h->Sum( f, g )
assert( h->Eval(x,y,z) = s )
```