APS 105F – Computer Fundamentals

Assignment #5: Recursion

Fall 1997

(To be completed in your lab period Oct. 8/9)

Objective: To understand how recursion works.

1 Factorial Function

The *factorial* can be implemented iteratively (meaning with loops) and recursively. An iterative definition of *factorial* is:

```
function factorial (N : int) : int
    var x : int := 1

    for i : 1 .. N
        x := x * i
    end for

    result x
end factorial
```

What is the recursive definition of *factorial* and what is the stopping condition?
A recursive version of *factorial* can be found in

`/usr/copy/aps105/factorial.t`

This version includes code that allows you to trace recursive calls as they occur in the evaluation of a factorial. Copy this file to your own directory. Study it, and test it with various input values. You will discover that you can compute factorials only up to a certain value. Determine this unlucky limit, and be able to explain why higher factorials cause the program to fail.

2 Printing A Wordlist in Reverse Order

Design, implement and test a Turing program that can read a list of words and print them in reverse order. The word "end" will end the list, and is not to be included in the output. You are to use recursion to do this rather than a data structure (i.e. no arrays or collections).
Hint: Design a procedure "getmore" that reads a word (a **string** in Turing), returns if the word is "end", and otherwise calls itself (to process the other words in the list) and then prints the word. Your code will look something like this.

```
procedure getmore
        var s: string
        put "Enter word: "..
        get s
        .
        .
        finish this code
        .
        .
end getmore

put "Enter a list of words, ending with \"end\"."
getmore
```

The execution of the program will produce something like this. The input supplied by the user is shown in **bold**.

```
Enter a list of words, ending with "end".

Enter word: recursion
Enter word: is
Enter word: useful
Enter word: end
useful
is
recursion
```

3 Finding Roots

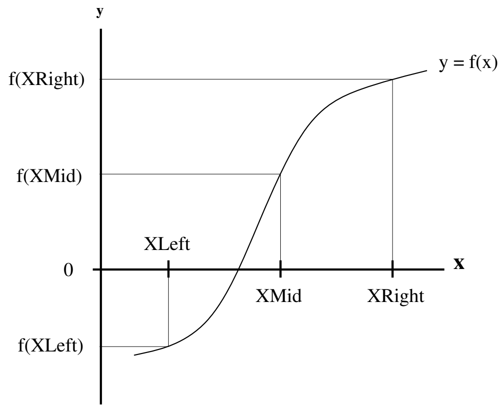
The bisection method is a means of finding an approximate root for the equation $f(x) = 0$ on the interval **XLeft** to **XRight** inclusive. The function is assumed to be continuous in this interval.

The user inputs are:

- the interval endpoints: **XLeft** and **XRight**

- the tolerance for the approximation: **Epsilon**

The figure below shows an example situation:



The goal is to find an interval [**XLeft**,**XRight**] that is less than **Epsilon** in length over which the function changes sign. The midpoint of the interval

$$\mathbf{XMid} = \frac{\mathbf{XLeft} + \mathbf{XRight}}{2.0}$$

will then be an approximation of the root of the equation. If you happen to find a value of *f(***XMid***)* that does equal zero then you are done and the algorithm should stop.

If the sign change happens in the interval [**XLeft**,**XMid**] then let **XRight** = **XMid**, and similarly if the change is in the other interval. Then repeat the process.

Part A

Find an approximation to the equation

$$5x^3 - 2x^2 + 3 = 0$$

over the interval (-1,1) using **Epsilon** = 0.0001. Your program should use a *recursive* implementation of the bisection method.

Part B

Redo **Part A** but use an *iterative* implementation of the bisection method.
Optional: Can you devise an experiment to measure the relative speeds of the algorithms in **Part A** and **Part B**?

Part C

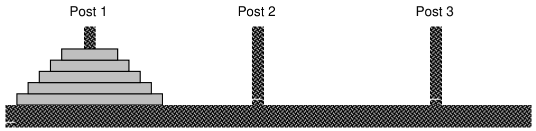
Find approximations to the three roots of the equation

$$2x^3 - x^2 - 13x - 6 = 0$$

You may assume that all roots are real numbers. Your may use any implementation of the bisection method.

4 Optional Section — Towers of Hanoi

The Towers of Hanoi game is the classic problem suited to a recursive programming solution. The set-up of the problem is illustrated in the figure below.



There are three upright posts in a line that are named post 1, post 2, and post 3. A number of rings of different sizes could be stacked up around the posts. To start, all of the rings are stacked on post 1, in order of decreasing size from the bottom up. The object of the game is to relocate the entire stack of rings from post 1 to post 2. The movement of rings is governed by two rules: 1. only one ring can be moved at any time from one of the posts to any of the other posts; and 2. at no time can a ring be placed on top of another ring of smaller size.

Determine the algorithm required to solve the Towers of Hanoi problem and implement it in a Turing program. The program should accept from the user the number of rings in the stack, and the program will list the directions to solve the puzzle. Here is an example of the output of the program:

```
       TOWERS OF HANOI
How many rings are in the stack?  3

Solution:
Move ring from post 1 to post 2.
Move ring from post 1 to post 3.
Move ring from post 2 to post 3.
Move ring from post 1 to post 2.
Move ring from post 3 to post 1.
Move ring from post 3 to post 2.
Move ring from post 1 to post 2.
```

You can verify that this is the solution for a stack of three rings.