APS 105F – Computer Fundamentals

Assignment #4: More Procedures and Functions
File Handling

Fall 1997

(To be completed in your lab period Oct. 1/2)

Objective: Think about rules for variable scope, do more practice with functions, and learn how to handle files.

Take Note: One of the most important tasks as a programmer is to make your program understandable to others. A program that cannot be maintained is of no use to anyone. Therefore, starting this week, all programs that you write for assignments, tests, and exams should exhibit good programming style. This means that the programs should be formatted in a sensible way and there should be sufficient comments to help someone else understand what the program is doing. In *OOT*, you can use the **Paragraph** command under the **File** menu to do the formatting for you. Another factor will be how your program is structured, including things like whether you use functions and procedures reasonably, and whether your procedures modify global variables.

Also, using obviously poor algorithms will be reflected in the grading.

1 Variable Scope

These are some small exercises for you to examine how variable scope works.

For you C programmers, you can try translating the programs, but it may be easier to fire up *OOT*, read in the program and hit the the run button!

Exercise 1

Consider the following Turing program:

```
var xm := 5

put "xm = ",xm

loop
    var y : int := 5
    loop
        var x : int := 0

        put "x loop = ", x, ", y = ",y
        x := x+1

        y := y - 1
        exit when y = 0
    end loop

    if xm = 0 then
        put "done"
        exit
    else
        var x : int := xm
        put "x else = ",x
        xm := xm - 1
    end if

end loop

put "xm end  = ",xm
```

Determine the scope of each variable and draw lines (as done in class) to indicate the regions where the variables can be referenced. What is the output of this program? You can check your answers by obtaining a copy of the program at

http://www.eecg.toronto.edu/~pc/courses/aps105f/assign4/loop1.t.

Exercise 2

Consider the following Turing Program:

```
var A, B : int

var Q, R, S, T : int

const Smallest := - 999

procedure swap (var A, B : int)
    var T : int

    T := A
    A := B
    B := T
end swap

procedure P (var Q, R, S : int)
    var T : int

    if Q > R then
        var A : int := 0
        swap (Q, R)

        T := R
        for i : Q .. R
            A := A + i
        end for
        put "Sum 1 = ", A
    else
        var A : int := 0
        T := R
        for i : Q .. R
            A := A + i
        end for
        put "Sum 2 = ", A
    end if

    if R > S then
        swap (R, S)
        if Q > R then
            swap (Q, R)
        end if
```



```
    end if

end P

function GetNum (S : string) : int
    var T : int

    loop
        put S, "? " ..
        get T
        exit when T > Smallest
    end loop

    result T
end GetNum


loop
    T := GetNum ("T")
    exit when T = 0

    Q := GetNum ("Q")
    R := GetNum ("R")
    S := GetNum ("S")

    put "Before (Q,R,S) = (", Q, ",", R, ",", S, ")"

    P (Q, R, S)

    put "After (Q,R,S) = (", Q, ",", R, ",", S, ")"

    put ""
end loop

put "Done!"
```

Determine the scope of each variable and draw lines (as done in class) to indicate the regions where the variables can be referenced. What does this program do? You can check your answers by obtaining a copy of the program at

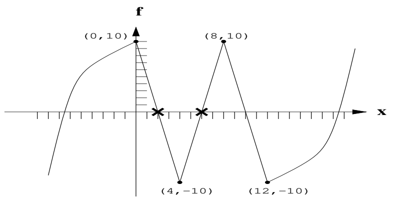
http://www.eecg.toronto.edu/~pc/courses/aps105f/assign4/loop1.t.

2 Finding the Zero Crossings of a Function

You are to write a program that counts the number of times that a function f crosses the X-axis in a range of X values specified by the user. In essence, the program counts the number of zeroes of the function in a given range. The function f is defined as follows:

$$\begin{array}{ll} f = 10 - x^2 & x < 0 \\ f = -5x + 10 & 0 \le x < 4 \\ f = 5x - 30 & 4 \le x < 8 \\ f = -5x + 50 & 8 \le x < 12 \\ f = (x-12)^2 - 10 & x \ge 12 \end{array}$$

Below is a plot of the function f:



For example, if the input range was specified as 0.0 to 8.0, the output of the program should be 2, because the function f crosses the x-axis twice between 0 and 8, as indicated by the crosses in the figure.

Your program should have two functions called **Func** and **Cross**. The **Func** module calculates and returns the value of the function f, as defined above. The **Cross** module takes two values of the function as input and returns a boolean value that indicates if an x-axis crossing has occurred between those two values.

3 Input/Output Redirection

Part of this exercise is to learn how to use new features in a language.

The programs you have designed so far have input data from the user at the computer terminal and have output text to the screen. For this exercise, you will experiment with reading in data from a text file and writing output to a text file. The Turing statements you will have to reference to complete this exercise are **open**, **close**, **get**, **put**, and **eof**. You should read Section 9.4, *Text Files* in Koffman if you are not familiar with handling files.

If you do not know how to use the file handling constructs, do not just read the book and attempt to complete the program for this assignment. The proper way to learn how to use any new feature in a programming language is to write a very simple program. For example, in this case, you can try the following steps:

1. Learn how to use the **open** and **get** commands for files. Create a text file with a simple message in it. Write a program to open that file, read the contents and print them on the screen.

2. You may encounter an error in the above program because you do not know how to detect when you have reached the end of the file. Now you need to learn about the *end of file* flag. Figure out how to use **eof** to detect the end of the file.

3. To clean things up, you should *close* your files when you are finished with them. In this trivial case, this is not really necessary as all files are closed when the program exits. Learn what *close* does and how to use it. Add this to your program.

4. Once you are comfortable with reading a file, detecting the end of the file, and closing the file, modify your program to output into a file instead of on the screen. Do not forget to close your file before your program exits.

When you have done all of the above, you have mastered all the basics of file handling!

Remember, when trying to learn a new feature try it on a small program first. That way you can learn how the language does it, without worrying about other problems. You may even want to keep this small program around so that you can refer to it later when you have forgotten what you just learned. Even if you are an experienced programmer trying to learn a new language, this is good practice because you can work out the details of the syntax.

Now for the real problem...

Write a new version of your quadratic equation solver program from Assignment 2 that reads in values of **a**, **b**, and **c** from a file named *coefs.txt*. This file (which you must first create) will contain one or more rows of data, each row having three numbers corresponding to **a**, **b**, and **c**. A sample *coefs.txt* file might look like this:

```
  2     7    -1
  4.9  12     3
 -2.71  3.14  0.98
```

Your program is to read in the data from the file, one row at a time, solve for the roots, and then display the results on the screen. The program should repeat these steps until all rows of data have been read. Display your results in tabular form to make the results clear. For the sample *coefs.txt* file shown above, your program should produce output that looks something like the following:

```
            Quadratic Root Solver

    a        b       c        root1     root2
------------------------------------------------
    2        7      -1      0.137459  -3.637459
    4.9     12       3     -0.282614  -2.166366
   -2.71     3.14    0.98  -0.255681   1.414353
```

Modify your program so that it sends the output to another text file, *roots.txt*. This modification should require only a few small changes.

4 Optional Section: UNIX I/O Redirection

Using **toot**, generate an executable version of your original quadratic solver that took input from the keyboard and output the results on the screen. Use UNIX I/O redirection to read data from *coefs.txt* and put the output into *roots2.txt*.

Recall that **toot** allows you to run your program from the command line:

```
spark1.ecf% toot quad.t
```

If **toot** complains, it may be because you have used some commands such as **locate()** in your program that try to do some screen-related things. You should remove those.