APS 105F – Computer Fundamentals

Assignment #3: Code Transformations and Modules

Fall 1997

(To be completed in your lab period Sept. 24/25)

Objective: To do some small code transformation exercises and start using modules.

1 Code Transformations

Often you are faced with converting the description of an algorithm into a programming language that does not have the constructs (usually looping constructs) that have been used to describe the algorithm. These small exercises will get you to work through some of these situations. In each case, you must produce a program that will implement the same algorithm.

Exercise 1

Consider the following program:

```
var x,y : int
loop
        put "x? "..
        get x
        put "y? "..
        get y
        exit when x = 0 and y = 0

        loop
                exit when y = 0
                put x*y," "..
                y := y-1
        end loop
        put ""
end loop
put "done",skip
```


Assume now that you only have the **do-while** looping construct available. A **do-while** statement has the form

```
    do
        <actions>
    while <condition>
```

Recode the program using **do-while** loops. Recall that you can simulate a **do-while** in Turing by putting the *exit* statement in the appropriate place. For you C programmers, C has this construct.

Exercise 2

Consider the following program that has multiple exits:

```
var x, xsquared, xroot : real

loop
        put "x? "..
        get x

        exit when x < 0

        put "x = ",x

        xsquared := x*x
        put "x**2 = ",xsquared
        exit when xsquared > 10000

        xroot := sqrt(x)
        put "sqrt(",x,") = ",xroot,skip

end loop

put skip,"Good bye!"
```

Recode the program using **while** loops. A **while** statement has the form

```
    while <condition>
        <actions>
    end while
```

Recall that you can simulate a **while** in Turing by putting the *exit* statement in the appropriate place. For you C programmers, C has this construct.

Exercise 3

Replace the **for** statement in the following program with a **do-while** style construct as described in Exercise 1.

```
var x,y : int

loop
    put "x? "..
    get x
    put "y? "..
    get y

    for Num : x .. y
        put Num," "..
    end for
    put skip
end loop
```

2 Procedures and Functions

Procedures and functions are used to improve the organization of your programs by separating individual tasks into "sub-programs". To illustrate the new concepts associated with procedures and functions, the quadratic equation solver program of Lab 2 has been rewritten and error checking added. The modified program is depicted in Figure 1. Please study it so that you fully understand it.

The `getvalue` procedure is used to prompt the user for input. The same procedure is used for all three coefficients.

The parameters of this procedure are declared as `"name:string; var coefficient:real"`. What is the difference between having `var` written before a parameter, and not having it? How would the procedure change if `var` was not written before `coefficient`?

What is the basic difference between a procedure and a function? In the function `discriminant` what does the `:real` at the end of the first line of the function declaration signify?

What are the global variables of the program? Identify some local variables. Could the `solvequad` procedure be rewritten without parameters? How?

3 A Car Fuel Efficiency Program

You just won the Super 7 Lotto and purchased two new cars. You would like to keep track of the fuel efficency as measured in litres/100 kilometres, as well as the total amount you have


```
function discriminant(a,b,c:real):real
  result b**2-4*a*c
end discriminant

procedure getvalue(name:string, var coefficient:real)
  put "Enter the value of ", name, ": "..
  get coefficient
end getvalue

procedure solvequad(a,b,c:real)
  var dis:real
  if a=0 then
    if b=0 then
      put "The equation has no solution"
    else
      put "The solution of the equation is: ", -c/b
    end if
  else
    dis:= discriminant(a,b,c)
    if dis>0 then
      var root1:real:=(-b+sqrt(dis))/(2*a)
      var root2:real:=(-b-sqrt(dis))/(2*a)
      put "The roots of the equation are: ", root1, " and ", root2
    elsif dis=0 then
      var root:real:=-b/(2*a)
      put "The solution of the equation is: ", root
    else
      var realroot:real:=-b/(2*a)
      var imagroot:real:=sqrt(abs(dis))/(2*a)
      put "The roots of the equation are: ", realroot, " + i*"..
      put imagroot, " and ", realroot, " - i*", imagroot
    end if
  end if
end solvequad

var A,B,C:real
getvalue("a",A)
getvalue("b",B)
getvalue("c",C)
solvequad(A,B,C)
```

Figure 1: The quadratic equation solver program

spent for fuel on each car. Write a program that allows you to input the current odometer reading, the number of litres that you put in, and the cost of the fuel after each time you fuel one of your cars. It is assumed that you fill-up the tank each time. For each car, the program will keep track of the initial and current odometer readings, the total amount of fuel consumed, and the total amount spent on fuel since the program was reset.

The program starts by initializing all values to zero and then continuously prompts the user to select an action. The choices are to input data after a fill-up, to request the fuel efficiency for the last tank of gas, the overall efficiency and cost since you started, and to reset the initial values of the odometer, total fuel used, and total cost. Since the actions are the same for either car, you should write only one module for each action.

Note that there is a special *boundary case* when you start your program because no data has been entered. You must handle this and any other problems reasonably. Assume that the correct type of data will be input, i.e., you will not get a letter when you are asking for a number.

The **I**nput module should check that the new odometer reading is greater than the previous one and that the cost was greater than zero. The **L**ast module will give you the fuel efficiency for the most recent fill-up, and the **A**verage module will compute the overall efficiency and cost of fuel. The **R**eset module will prompt for an odometer reading, and set total fuel usage and cost to zero.

An example session is shown in Figure 2. The program should accept either upper or lower case letters to specify actions, e.g., either **Q** or **q** for **(Q)uit**.

4 Optional Section – Exception Handling

I assume the input procedure of your fuel efficiency program checked the user input for errors. Well-engineered software will always anticipate problems that might come up during the execution of the program, and will always be prepared to handle them. In your program, you may have used if-then statements to detect improper input. However, some problems that might come up during the execution of the program may not be resolved using such program control statements. For example, if your program prompts the user to enter a real value, but s/he enters a non-numeric character, this will result in the generation of an exception, and if you have not specified in your program how to deal with an exception, the program will terminate.

A formal method of dealing with exceptions is provided in OOT by way of the exceptionHandler mechanism. Full instructions on the use of the exceptionHandler are provided in the OOT reference manual. Extend your fuel efficiency program by adding handler blocks to deal with exceptions that result from improper input. If the user provides a non-numeric input for the odometer, fuel, or cost amounts, the exception handler should post a warning to the user and request for input again.

A more human-readable description of exception handling is described on the home page.


```
     Fuel Efficiency Program

(I)nput, (L)ast, (A)verage, (R)eset, or (Q)uit? A
(M)aserati or (L)ambourghini? m
No values available.  You need to set the initial odometer value.

(I)nput, (L)ast, (A)verage, (R)eset, or (Q)uit? r
(M)aserati or (L)ambourghini? m
Enter odometer reading: 578

(I)nput, (L)ast, (A)verage, (R)eset, or (Q)uit? a
(M)aserati or (L)ambourghini? m
You travelled 0 kilometres at a rate of 0 litres/100 km.

(I)nput, (L)ast, (A)verage, (R)eset, or (Q)uit? I
(M)aserati or (L)ambourghini? m
Enter odometer reading: 1153
Enter litres: 47.8
Enter cost: 24.38

(I)nput, (L)ast, (A)verage, (R)eset, or (Q)uit? l
(M)aserati or (L)ambourghini? m
You travelled 575 kilometres at a rate of 8.31 litres/100 km.

(I)nput, (L)ast, (A)verage, (R)eset, or (Q)uit? i
(M)aserati or (L)ambourghini? m
Enter odometer reading: 1672
Enter litres: 42.3
Enter cost: 21.02

(I)nput, (L)ast, (A)verage, (R)eset, or (Q)uit? a
(M)aserati or (L)ambourghini? m
You have travelled a total of 1094 kilometres at a rate of
8.24 litres/100 km for a total cost of $45.40.

(I)nput, (L)ast, (A)verage, (R)eset, or (Q)uit? Q

End of Program
```

Figure 2: Fuel Efficiency Program