CS 335 - Homework 10

Deadline:

Due by 11:59 pm on Friday, May 6th

How to submit:

Submit your files for this homework using `~st10/335submit` on nrs-labs, with a homework number of `10`

Purpose:

to practice with some older versions of if statements; to consider a classic problem with the venerable if-else statement; to practice with several older approaches to repetition; to consider two different approaches to the for-statement; to comment on each other's Prolog games; to reflect on Scheme, Prolog, and Squeak

Important notes:

- Put your answers for this homework into a file `hw10.txt`; make sure the file includes your name, and precedes each answer with the problem/part number for which it is the answer.

The Problems:

Problem 1 - 20 points:

(adapted from MacLennan Exercise 2-1, p. 48)

If you look over the early-FORTRAN examples in Chapter 2 of MacLennan, you should be able to see how FORTRAN assignment statements and basic GOTO statements were written; and in class, we discussed its arithmetic-IF and its (added a bit later) logical-IF.

(Or, if you'd like another reference, I like this FORTRAN-IV reference page:

http://www.math-cs.gordon.edu/courses/cs323/FORTRAN/fortran.html

)

And, FORTRAN has optional automatic declaration of variables; so, for the parts below, assume that variables `S` and `N` have been automatically declared. (For this problem, do not use any FORTRAN features from versions of FORTRAN later than FORTRAN IV.)

1 part a

Write a fragment of FORTRAN that uses an arithmetic-IF, assignment statements, and GOTO's to accomplish the following:

- store 13.5 in `S` if `N = 0`
- store -5.1 in `S` if `N > 0`
- store +5.1 in `S` if `N < 0`

1 part b

Now write a fragment of FORTRAN that uses logical-IF's and assignment statements to accomplish the same thing.

Problem 2 - 10 points

Consider that classic selection statement in many languages, the if-else statement.

Depending on the syntax, an if-else statement can be prone to a problem called the **dangling else** problem. That is, consider the following "pseudocode" statement (written using absolutely atrocious style to make a point):

```
if condition1 then if condition2 then statement1 else statement2
```

When will `statement2` be executed -- when `condition1` is false, or when `(condition1 is true AND condition2 is false)`? That is, which `if` is the `else statement2` paired with?

Algol-60, Pascal, and Ada addressed the dangling else problem in three different ways. We briefly discussed these in class, and MacLennan discusses two of these in more detail; you can find out more detail about the third on the web, or by using the ACM Digital Library.

2 part a

Just to make sure you know -- explain how each of these languages either handled or prevented this problem.

2 part b

Which of these three approaches do you prefer, and why?

Problem 3 - 12 points

(adapted from MacLennan Exercise 2-2, p. 48)

Assume that a variable `N` has been automatically declared and that a value has been read into it.

Write in FORTRAN IV a fragment of code with a leading-decision loop that doubles the value in variable `N` until it is greater than 250. (If it is already greater than 250, it should **not** change the value in `N`)

Problem 4 - 12 pts

(adapted from MacLennan Exercise 2-10, p. 51)

Write a FORTRAN IV fragment that uses a DO-loop to compute into a variable `SUM` the sum of the array elements `A(1) + A(2) + ... + A(250)`

Problem 5 - 12 points

Algol-60 was a language renowned for its elegance in a number of areas -- which made its less-elegant features stand out all the more. Several of its features took the goal of generality too far, to the point of being "cluttered with features of extreme generality and doubtful utility" (MacLennan, p. 138), or baroque.

Algol-60's extremely general for-loop has been described as being baroque -- you can read about it in MacLennan, on pp. 137-138.

To give you a taste of how baroque the Algol for-loop could be, determine and write out what the following fragment of Algol-60 code would print (assuming that `Print` prints its argument's value on its own line):

```
for i := 3 step 2 until 11,
         5, 14, 6,
         i*2 while i < 40
do
    Print i
```

Problem 6 - 10 points

For a more modern and different variation of a for-loop (different from the C++ for-loop, anyway), look up a description of Python's for-loop on the web.

Assuming that a Python list can be expressed as a comma-separated list of expressions within parentheses (for example,

```
(1, "dog", True)
```

and that:

```
print <expression>
```

...prints the value of `<expression>` on its own line, write an example of a Python for-loop printing out the value of each of a list of at least seven expressions on its own line.

(Note: you can test your for-loop using Python on nrs-labs, if you like -- you can reach the Python interpreter there by simply typing:

```
python
```

and you can use `exit()` or ^D to exit the interpreter when you are done.)

Problem 7 - 12 pts

Consider the Prolog games you have written. You will see them in action in class on Tuesday, May 3rd;

those latest versions will be posted on the course Moodle site on that day as well.

Play each game at least a little, and look at its source code if you'd like; then, for each game, write either something you liked about it or some Prolog feature used in its code that you found particularly interesting.

It is possible that these comments may be sent to each game's creator.

Problem 8 - 12 pts

Consider Scheme, Prolog, and Squeak.

For each, write the following:

- something that you find to be easier to do in that language than in C++
- something that you find to be harder to do in that language than in C++
- something you liked (or, if more appropriate, disliked least... 8-)) about that language
- something you disliked (or, if more appropriate, liked least... 8-)) about that language

It is possible that these may be collected and posted to the course Moodle site.