Problem Set #6 - Due 03/19/03 Total 150

The purpose of this problem set is to:

- Help you become familiar with material covered in week 6 of class.

Please turn in each problem on a separate page. Each page should have your Name, email id, and the problem number clearly printed/written on it. Keep track of how long time it takes to complete each problem. The time taken for each problem should be printed on the first page. If you use more than one page for one problem, please STAPLE the pages together. You will lose points if you do not document the time taken for each problem, which at the same time means that you will get points for documenting "time taken" A template (in PDF form) is available on the web.

Problem 1 - 50 points

Part 1: Write pseudo code to:

- Add two 3x3 matrices
- Multiply two 3x3 matrices.
- Find the transpose of a 3x3matrix
- Find the inverse of a 3x3 matrix

Turn in a **hard copy** of your pseudo code.

Part 2: Define your own package, with functions/procedures to

a. Create a 3x3 matrix.
b. Display a 3x3 matrix
c. Add two 3x3 matrices
d. Multiply two 3x3 matrices
e. Find the inverse of a 3x3 matrix
f. Find the transpose of a 3x3 matrix.

Write a program to use your package to create two 3x3 matrices, display the contents of the matrices to the user, add the matrices, and display the sum to the user, find the product of the matrices and display it to the user, find the inverse of the product, display the inverse to the user, find the transpose of the product of the matrices and display it to the user. Turn in a **hard copy** of your **code listing** and an **electronic copy** of your **code**. Please zip the related files (1 .ads and 2 .adb files) into a zip file as detailed in Gerry's email.

Part 3: What is the time complexity (Big O) of the procedures implemented above.

a. Create
b. Display
c. Add two 3x3 matrices
d. Multiply two 3x3 matrices
e. Find the inverse of a 3x3 matrix
f. Find the transpose of the matrix.

Explain how you derived your results. Turn in a **hard copy** of your solution.

Problem 2 - 25 points

What are strings? What are the operations allowed on strings? (List the various functions and procedures that define operations on strings). Add a single line explanation for each of the functions / procedures. Turn in a **hard copy** of your solution.

Hint : Look at Chapter 10 of Feldman and the Ada.Strings .ads and .adb files.

Problem 3 - 75 points

Part 1:

A mathematical expression such as: y + z, has an **operator** (+) and two **operands** (y and z). Such operators are called **binary** operators because they have two operands

Normally **infix notation** is used, requiring brackets to determine a sequence of operations in some expressions: such as 3 * (4 + 5)

Polish notation, also known as **prefix** notation, is a symbolic logic invented by Polish mathematician Jan Lukasiewicz in the 1920's. When using Polish notation, the instruction (operation) precedes the data (operands). In Polish notation, the order (and only the order) of operations and operands determines the result, making parentheses unnecessary. The notation for the expression 3 * (4 +5) is expressed as * 3 + 4 5

In the early days of the calculator, the end-user had to write down the results of their intermediate steps when using the algebraic Order of Operations. Not only did this slow things down, it provided an opportunity for the end-user to make errors and sometimes defeated the purpose of using a calculating machine. In the 1960's, engineers at Hewlett-Packard decided that it would be easier for end-users to learn Jan Lukasiewicz' logic system than to try and use the Order of Operations on a calculator. They modified Jan Lukasiewicz's system for a calculator keyboard by placing the instructions (operators) after the data. In homage to Jan Lukasiewicz' Polish logic system, the engineers at Hewlett-Packard called their modification reverse Polish notation (RPN).

The notation for the expression 3*(4+5) would now be expressed as

4 5 + 3 *
or it could be further simplified to
3 4 5 + *

Write pseudo code to transform an expression in infix form into postfix and prefix.

- Assume that variables and operators are represented by single characters.
- Your algorithm has to handle the binary operators '*', '/', '+', '-'.
- '*', '/' have greater precedence than '+', '-'
- '*', '/' have the same precedence. If '*', '/' are present in the expression, without parentheses, the expression is evaluated from left to right.
- '-', '+'have the same precedence. If '+', '-' are present in the expression, without parentheses, the expression is evaluated from left to right.

Part 2: Write a program in Ada95 to:

- Get an input expression in infix form from the user. (Use an array of characters, you much check to see if the user input is legal i.e. that he/she inputs only legal operands and operators).
- Convert the expression into postfix
- Display it to the user
- Convert the expression into prefix notation
- Display it to the user.

Turn in a **hard copy** of your solution in **case study format** and an **electronic copy** of your **code**. Please zip the related files (1 .ads and 2 .adb file) into a separate zip file (not the one used for Problem 1) as detailed in Gerry's email.

Hint:

- Use Enumerations to check legality of operators and operands.
- Use a stack to convert between the expressions.