CMPE160 PROJECT#2

“Simple Compiler”

Deadline: April 16*th*, 2004, 17:00

1. Project Description:

In this project you are going to implement a simple “compiler” for a primitive programming language. Your program will accept commands of the following form:

- **INPUT** *variable name*
- **PRINT** *variable name*
- *variable name* = infix arithmetic expression involving variable names and arithmetic operators +, -, *, /, ^
- **GOTO** *line*
    - **ALWAYS**, or
    - **IF** *infix logical expression* involving variable names and operators +, -, *, /, ^ (exponentiation operator), & (for AND), | (for OR), ~(for NOT), <, >, =
- **STOP**
- **RUN**

These commands are to be stored in an array of strings until the **RUN** command is entered. Upon encountering **RUN**, your program should execute the previously stored commands. "Execute" here means

- For an **INPUT** command: send a question mark to the terminal and allow the user to enter a real number, which is stored in variable name.
- For a **PRINT** command: write to the terminal the numerical contents of the specified variable name.
- For an assignment command: parse the expression into postfix form and then evaluate it. Store the result in the variable name on the left of the equality sign.
- For a **GOTO** command: branch to the line number specified when ALWAYS follows the line number or when the infix expression that follows the IF evaluates to true. Here "line number" refers to the relative position of the line in the sequence of lines that were entered prior to the **RUN** command. The first line number in this sequence is "00."
- For a **STOP** command: halt execution.

To make things relatively easy, you may assume a syntax that

- specifies that one and only one blank space follows **INPUT**, **PRINT**, **GOTO**, and line number. No other blanks appear anywhere.
- allows only one variable name to follow **INPUT** or **PRINT**.
- only allows variable names consisting of one uppercase alphabetical character.
- only allows line numbers consisting of two digits: 00 through 99.

The order of precedence of operators is as following:

Arithmetic operators	^
	*, /
	+, -
Relational operators	<, >, =
Logical operators	~ (NOT)
	& (AND)
	| (OR)

The exponentiation operator (^) has the highest precedence, furthermore it is right associative. In handling more than one consecutive exponentiation operations, the following example should illustrate the need for care:

3^2^3 = 3^8 not 9^3.

In this project, you should use Dynamic Array Stacks.

2. Material to Submit

- **SDD:** You are going to deliver a software design document until April 9, Friday, 17:00 to Ömer Korçak at ETA 104. The document should follow the guidelines explained in the PS. You are encouraged to use flowcharts, diagrams, and graphical aids to make our document easier to read and understand.
- **CODE:** The complete working code will be delivered on April 16, Friday, until 17:00. Your code should work in Visual C++ 6,0. You are expected to write a clear code, with lots of comments, good variable and class names, and as object-oriented as possible. If you make changes to the SDD please resubmit it.

We are going to check your code for

- -Good programming style
- -Compliance with project specifications
- -Compliance with design document

NOTE:

- Our department has very strict codes for student "collaboration" on projects.
- Remember that the **only** aim of this project is to teach you to have good programming skills that you are going to use for the rest of your lives. There is no way of learning programming other than hands-on experience.
- Both the document and the project will require your full attention for at least a couple of days. If you start them one day before the deadline, it will not possible to complete them in a satisfactory manner.
- Please work on your own, and ask your assistants immediately if there is a problem you cannot handle.