Project 1: syntax analysis (30pts)

Due Oct 14, 2010 (corrections may be made until final)

In this project, you are required to build a scanner and a parser for a subset of the C language (you may pick a different language other than C, but your language must support an equivalent set of terminals/tokens and non-terminals/concepts).

Assuming that you decide to parse a subset of the *C* language, your scanner need to recognize the following token types.

- Separators: SEMICOLON(";"), LPAREN("("), RPAREN(")"), COMMA(","), LBRACE("{"), RBRACE("}"), LBRACKET("[") and RBRACKET("]").
- Operators: ASSIGN("="), LT("<"), GT(">"), EQ("=="), PLUS("+"), MINUS("-"), MULT("*"), DIV("/"), NOT("!"), AND("&&"), OR("||")
- Keywords: IF("if"), ELSE("else"), WHILE("while"), INT("int"), FLOAT("float").
- Identifiers: ID, which includes all user-defined names in C.
- Integer constants: ICONST. Examples include 3, 235, etc.
- Floating point constants: FCONST. Examples include 3.2, 5E2, 2.3e-3, etc.
- Comments: COMMENT. All strings enclosed within "/*" and "*/".

Based on the above defined tokens, your parser needs to support the following language constructs.

- Variable declarations for integer, floating point values and arrays. e.g.

```
int x, y, z;
float a, b, c;
float arr[100], arr2[100][50];
```

- Integer and floating point expressions that support ASSIGN("="), LT("<"), GT(">"), EQ("=="), PLUS("+"), MINUS("-"), MULT("*"), DIV("/"), NOT("!"), AND("&&"), OR("||"). e.g.

```
!(a == b + 1 * (c - 1) / 2)
```

- Expression statements. For example, "a = b+c; " and "a+b;" are both expression statements, each of which contains a single expression.
- Block statements (sequences of statements enclosed within pairs of braces), e.g.

```
{a = b + c; d = a * 2; }
```

- If-else statements, e.g., "if (a < b) c = b; else c = a; ".
- While statements, e.g., "while (a < b) a = a + 1; ".

The follow is a sample input to be processed by your scanner and parser.

```
float a[100][100], b[100][100], c[100][100];
int i, j, k;
i = 0;
while (i < 100) {
  j = 0;
  while (j < 100) {
    if (!(c[i][j] == 0))
       c[i][j] = 0;
    k = 0;
    while (k < 100) {
       c[i][j] = c[i][j] + a[i][k] * b[k][j];
       k = k + 1;
    }
    j = j + 1;
  }
  i = i + 1;
}
```

You may choose to write the scanner/parser manually using C/C++/Java, or use automatic scanner/parser generators such as the *lex* and *yacc* under Unix/Linux. You can also the POET transformation scripting language to implement your scanner and parser. Using POET is encouraged as it is a research language under active development. You can find help on lex, yacc at the class web site or simply typing "man lex" and "man yacc" on your Linux or Unix machine. The POET software as well as its language manual can also be downloaded from the class web site.

Submit your project by packaging up your directory into a gzip file and then sending the file online to www.cs.utsa.edu/ cs5363 (submission web site will be set up soon). Suppose your entire project is in a directory named "project1". Goto the parent directory of *project1* and type

```
> tar -cf project1.tar project1
> gzip project1.tar
```

A file *project1.tar.gz* will be generated. Make sure that you document which approach you have chosen to implement the project, especially if you are using an approach different from any of the options listed in this assignment. If you used any tool beyond those mentioned in this assignment, you also need to make sure that we have access to the same tool (letting us know how to download it). And if you have chosen a language different than C, you must provide your own test file which is equivalent to the one given.