Scanner assignment

Overview of the Overall Projects

The projects in this course consist of the development of four phases of a compiler construction. This Scanner is the first phase of it. Later phases include a Parser, an intro-function code generator, and a inter-function code generator. You may program in C/C++/Java/Python.

Description of this Project

In this particular project, you are asked to write a scanner, the first phase of your compiler. The scanner converts an input program into tokens and rewrites the program with some tokens modified. If the input program is a correct C program, the output program should also be a correct program that has identical behavior. The specific requirements for code generation are:

- The scanner should recognize all the tokens of the input program. The token definitions are here.
- The scanner to add string "cs512" to the beginning of every <identifier> token except the name of function "main".
- The '#include' statements and C++ style comments are treated as special tokens. They are defined as <meta statement> in our token language. Your scanner should simply copy them to the generated program (each in a separate line) without modification. Your future compiler will be asked to do the same.
- The scanner takes one arguement, which is the name of the input program, and puts the generated program in a new file named with an extension "_gen". So "foo.c" will become "foo_gen.c".
- The output program does not need to be formatted in any way. The scanner can write all statements in one line, except that the meta-statements should be put in separate lines.

For example, the following program should be converted as follows.

example program : foo.c

```
#include <stdio.h>
#define read(x) scanf("%d\n", &x)
#define write(x) printf("%d\n", x)

// function foo
void foo() {
   int a;
   read(a);
   write(a);
}

int main() {
   foo();
}
```

running your scanner:

% scanner foo.c

generated program (no formatting needed): foo_gen.c

```
#include <stdio.h>
#define read(x) scanf("%d\n", &x)
#define write(x) printf("%d\n", x)

// function foo
void cs512foo() {  int cs512a;  read(cs512a);  write(cs512a);  }  int main() { cs512foo(); }
```

Your scanner will be tested by whether the generated program gives the same execution result as the input program.

Recommended implementation:

- Encapsulate states and functions of tokens in token objects.
- Encapsulate the scanning process in a scanner object that has the following three functions:
  - A constructor that takes in the name of the input program
  - A test function that tells whether there is more token left in the input program
  - A access function that returns the next token

With this interface, you can scan through and print a program as follows:

```
Scanner scanner("test1.c");  // Initialize the scanner.
While (scanner.HasMoreTokens()) {
   Token t = scanner.GetNextToken();
   if (t.GetTokenType()==ID && t.GetTokenName()!="main")
        ... ... // print token with cs512 attached
   else
        t.Print();
}
```

Token language

The tokens include **<identifier>, <number>, <reserved word>, <symbol>, <string>, and <meta statement>**. They are defined as regular expressions as follows.

<letter> --> a | b | ... | y | z | A | B | ... | Z | **underscore**
<identifier> --> <letter> (<letter> | <digit>)*

<digit> --> 0 | 1 | ... | 9
<number> --> <digit>+

<reserved word> --> **int | void | if | while | return | read | write | print | continue | break | binary | decimal**

<symbol> --> **left_parenthesis | right_parenthesis | left_brace | right_brace | left_bracket | right_bracket | comma | semicolon | plus_sign | minus_sign | star_sign | forward_slash | == | != | > | >= | < | <= | equal_sign | double_and_sign | double_or_sign**
(The symbols are: **( ) { } [ ] , ; + - * / == != > >= < <= = && ||**)

<string> --> any string between (and including) the closest pair of quotation marks.
<meta statement> --> any string begins with '#' or '//' and ends with the end of line ('\n').