Quiz 1 CSCI 431 Fall 2005
23 September, 2005

Name: ____________________

This is an open book, open notes quiz. Be sure to show your work in order to get full credit for the problem. When possible place your answers in the provided boxes. There are 6 questions on this quiz.

The `BoolTests.atg` file distributed with Homework assignment 4 is included on the last page of this exam.

Problem 1 (5 points)

In the style of Coco, write two character set definitions: One for all letters and another for all digits.

```
CHARACTERS
  letters     = ____________________ .
  digits      = ____________________ .
```

Problem 2 (5 points)

Which of the following five strings could match the following token definition?

```
     ″ye″ { ″e″ } ″ha″ { ″a″ | ″rvard″}
```

yeha
yeeee
yehaarvard
yeeeeeeeeha
yha

Problem 3 (15 points)

The following Java "method" contains six compile-time errors: 0ne per line. State which of these errors are lexical, syntactic, and semantic errors. The six errors are all underlined and in bold face.

```
     public static void (X, int Y) {

          boolean b = Y <> 5 ;

          if (Y)

               System.out.println(″This is fun) ;

          return b ;

     ]
```

Problem 4 (30 points)
Using your character set definitions of Problem 1, write token definitions (regular expressions) that generate the languages described in the following subproblems. Give some justification for your answers, particularly if you hope to receive partial credit for answers which have some flaws.

Subproblem 4A:
Strings of the language start with the letters "`CSCI`" followed by three digits.
Typical members of the language include: "`CSCI201`", and "`CSCI431`".
Strings that are not in the language include: "`CSCI2000`", "`csci431`", and "`ARTS310`".

Subproblem 4B:
Strings of the language start with an even number of digits followed by any number of letters.
Typical members of the language include: "`47yucK`", "" (empty string), "`7777`", "`IOU`".
Strings that are not in the language include: "`431CSCI`", "`csci2000`", and "`12ABC5`".

Subproblem 4C:
In strings of the language, every single digit of the string is surrounded by letters.
Typical members of the language include: "`yuck`", "" (empty string), "`x7yzz7z7w`".
Strings that are not in the language include: "`CSCI431`", "`csci2`", and "`12ABC5`".
If you have an easy solution, it is probably wrong.

Problem 5 (15 points)
Using the `BoolTests atg` specification found on the last page of this exam, draw a parse three that matches the following `BoolEquiv`:

```
A + A B == A
```

Problem 6 (30 points)
In this last problem, you will define your own Coco-style productions for a rather useless language.

We'll use the following token definitions for `string`, a Java-style String literal that is composed only of letters, and `number`, a positive integer, in this problem.

```
TOKENS
  string  = letters { letters } .
  number  = digits  { digits } .
```

The simplest elements of our language are strings. These are simply our `string` tokens. Optionally strings may be sharped. A sharped string is followed by the symbol "#" and a number. Examples of sharped strings are shown below:

```
     XYZ # 15
     CSCI # 431
```

A "#" may be applied to a string only once. "`ABC # 5 # 17`" is **not** allowed in our language.

Finally, several optionally sharped strings may be dotted by connecting them with the "`.`" operator. We're now done with our language description. Here are some examples of dotted sharped strings that are allowed in our language.

```
     CSCI
     CSCI # 431
     CSCI # 431 .  UNCA
     A # 0 . B . C # 17 . FUN . EXAM # 2000000
```

Now you must define the productions required to generate our language.

```
PRODUCTIONS
  SharpedString =




  DottedSharpedExpr =
COMPILER BoolTests
$NCF

CHARACTERS
  letter     = 'A'..'Z' + 'a'..'z' .


TOKENS
  variable   = letter { letter } .
  literal    = '0' | '1' .

PRODUCTIONS
  BoolExpr   =
      BoolTerm
      {
        "+"
        BoolTerm
      }
      .
  BoolTerm   =
      BoolFactor
      {
        BoolFactor
      }
      .
  BoolFactor =
        variable
      |
        literal
      |
        "~"
        BoolFactor
      |
        "("
        BoolExpr
         ")"
      .
  BoolEquiv  =
      BoolExpr
      "=="
      BoolExpr
      '\n'
      .

  BoolEquivs = { BoolEquiv } .

  BoolAssign =
        variable
      ":="
        literal
      '\n'
      .

  BoolAssigns = { BoolAssign } .

  BoolTests  = BoolAssigns '\n' BoolEquivs .

END BoolTests .
```