Zuname _____ Matr. Nr. _____

Übungsgruppe

- ❑ 1 (Wöß) Do 10^{15}-11^{45}
- ❑ 2 (Wöß) Do 12^{45}-14^{15}
- ❑ 3 (Rammerstorfer) Do 14^{30}-16^{15}

Punkte _____ korr. _____

Letzter Abgabetermin
Donnerstag, 25.10.2001, 8^{15} Uhr

Lexikalischer Analysator

a) Implementierung (17 Punkte)

Beginnen Sie die Arbeit an Ihrem *MicroJava*-Compiler, indem Sie einen lexikalischen Analysator für die Sprache *MicroJava* schreiben. Lesen Sie sich dazu die Sprachdefinition genau durch. Welche Terminalsymbole und Terminalklassen werden in der Grammatik gebraucht? Wie sehen die Kommentare, wie die Zeichenkonstanten aus? Welche Schlüsselwörter und vordeklarierten Namen gibt es?

Folgende Schnittstelle ist vorgegeben:

Datei *Token.java*:

```
package MJ;
public class Token {
  // token codes
  public static final int
    none   =  0, ident     =  1,
    number =  2, charConst =  3, …
    ifK    = 33, // 'if' is Java keyword
    newK   = 34,… endFile   = 40;

    public int kind;
    public int line;
    public int col;
    public int val;
    public String string;
}
```

Datei *Scanner.java*:

```
package MJ;
  import …;
public class Scanner {
  // static variables
  private static InputStream in; // source file
  private static char ch;        // lookahead character
  public  static int col;        // current column
  public  static int line;       // current line

  //--------- print error message
  private static void error(String msg) { … }
  //--------- initialize scanner
  public static void init(InputStream s) { … }
  //--------- return next input token
  public static Token next() { … }
}
```

Beachten Sie:

- Bei jedem Aufruf von *next()* muss das nächste erkannte Terminalsymbol (= *Token*) zurückgegeben werden. Wenn das Ende des Eingabestroms erreicht ist, wird das Terminalsymbol *endFile* zurückgegeben.

- Schlüsselwörter, Bezeichner, (positive ganze) Zahlen, Zeichenkonstanten, sowie Operatoren müssen erkannt werden.
- Leerzeichen, Tabulatoren und Zeilenumbrüche müssen überlesen werden.
- Kommentare ('//') bis zum Zeilenende müssen überlesen werden.
- Folgende Ereignisse haben einen lexikalischen Fehler zur Folge:
  - Das Auftreten ungültiger Zeichen.
  - Bei Zeichenkonstanten:
    - Fehlende schließende Anführungszeichen.
    - Leere Zeichenkonstanten.
  - Zu große Zahlenkonstanten (der Wertebereich von *int* ist wie in *Java* von –2147483648 bis 2147483647 definiert). **Achtung**: der lexikalische Analysator muss nur positive ganze Zahlen liefern (also von 0 bis 2147483647). Negative Zahlen werden erkannt, indem ein Minuszeichen und eine positive ganze Zahl als zwei separate Tokens bei wiederholtem Aufruf von *next()* geliefert werden. (*Anm.*: -2147483648 kann daher nicht als Konstante im Programm direkt angegeben werden. Will man diese Zahl tatsächlich verwenden, müsste man -2147483647-1 schreiben).

  Lexikalische Fehler sollen Fehlermeldungen nach sich ziehen. Diese müssen unter Angabe von Zeilen- und Spaltenposition der Fehlerstelle ausgegeben werden.

  Beispiel: `-- line 10, col 8: invalid character 'Œ'!`

b) Test (7 Punkte)

Schreiben Sie einen Treiber, der den Scanner initialisiert und solange *next()* aufruft, bis das Dateiende erreicht ist. Der Treiber soll jedes *Token* mitsamt seinen Attributen ausgeben.

Beispiel: Eingabedatei *TuNix.mic*

```
class TuNix
  int dummy;
{
  int tuNix() { dummy=0; return dummy; }

  void main() {
    dummy= tuNix();
  }
}
```

Ausgabe:

```
line 1, col 1:     classKind
line 1, col 7:     ident          TuNix
line 2, col 3:     ident          int
line 2, col 7:     ident          dummy
line 2, col 12:    semicolon
line 3, col 1:     lbrace
line 4, col 3:     ident          int
line 4, col 7:     ident          tuNix
line 4, col 12:    lpar
line 4, col 13:    rpar
line 4, col 15:    lbrace
line 4, col 17:    ident          dummy
line 4, col 22:    assign
line 4, col 23:    number         0
...
```

Testen Sie den Scanner ausführlich mit einem umfangreichen *MicroJava*-Programm! Testen Sie auch die lexikalischen Fehler ab.