```
/**************************************************************
  JLex: A Lexical Analyzer Generator for Java(TM)
  Written by Elliot Berk <ejberk@cs.princeton.edu>. Copyright 1996.
  Maintained by C. Scott Ananian <cananian@alumni.princeton.edu>.
  See below for copyright notice, license, and disclaimer.
  New releases from http://www.cs.princeton.edu/~appel/modern/java/JLex/

  Version 1.2.6, 2/7/03, [C. Scott Ananian]
   Renamed 'assert' function 'ASSERT' to accomodate Java 1.4's new keyword.
   Fixed a bug which certain forms of comment in the JLex directives section
     (which are not allowed) to be incorrectly parsed as macro definitions.
  Version 1.2.5, 7/25/99-5/16/00, [C. Scott Ananian]
   Stomped on one more 8-bit character bug.  Should work now (really!).
   Added unicode support, including unicode escape sequences.
   Rewrote internal JavaLexBitSet class as SparseBitSet for efficient
     unicoding.
   Added an NFA character class simplification pass for unicode efficiency.
   Changed byte- and stream-oriented I/O routines to use characters and
     java.io.Reader and java.io.Writer instead --- which means we read in
     unicode specifications correctly and write out a proper unicode java
     source file.  As a happy side-effect, the output java file is written
     with your platform's preferred newline character(s).
   Rewrote CInput to fix bugs with line-counting in the specification file
     and "unusual behaviour" when the last line of the specification wasn't
     terminated with a newline. Thanks to Matt Hanna <mhanna@cs.caltech.edu>
     for pointing out the bug.
   Fixed a bug that would cause JLex not to terminate given certain input
     specifications.  Thanks to Mark Greenstreet <mrg@cs.ubc.ca> and
     Frank B. Brokken <frank@suffix.icce.rug.nl> for reporting this.
   CUP parser integration improved according to suggestions made by
     David MacMahon <davidm@smartsc.com>.  The %cup directive now tells
     JLex to generate a parser conforming to the java_cup.runtime.Scanner
     interface; see manual for more details.
   Fixed bug with null string literals ("") in regexps.  Reported by
     Charles Fischer <fischer@cs.wisc.edu>.
   Rewrote start-of-line and end-of-line handling, closing active bug #5.
     Also fixed line-counting code, closing active bug #12.  All
     new-line handling is now platform-independent.
   Used unpackFromString more extensively to allow larger cmap, etc,
     tables.  This helps unicode support work reliably.  It's also
     prettier now if you happen to read the source to the generated
     lexer.
   Generated lexer now accepts unicode LS (U+2028) and PS (U+2029) as
     line separators for strict unicode compliance; see
     http://www.unicode.org/unicode/reports/tr18/
   Fixed bug with character constants in action strings.  Reported by
     Andrew Appel against 1.2.5b3.
   Fixed bug with illegal \^C-style escape sequences.  Reported by
     Toshiya Iwai <iwai@isdnet.co.jp> against 1.2.5b4.
   Fixed "newline in quoted string" error when unpaired single- or
     double-quotes were present in comments in the action phrase.
     Reported by Stephen Ostermiller <1010JLex@ostermiller.com>
     against 1.2.5b4.  Reported by Eric Esposito <eric.esposito@unh.edu>
     against 1.2.4 and 1.2.5b2.
   Fixed "newline in quoted string" error when /* or // appeared
     in quoted strings in the action phrase.  Reported by
     David Eichmann <david-eichmann@uiowa.edu> against 1.2.5b5.
   Fixed 'illegal constant' errors in case statements caused by
     Sun's JDK 1.3 more closely adhering to the Java Language
     Specification.  Reported by a number of people, but
     Harold Grovesteen <hgrovesteen@home.com> was the first to direct me to
     a Sun bug report (4119776) which quoted the relevant section of the
     JLS (15.27) to convince me that the JLex construction actually was
     illegal.  Reported against 1.2.5b6, but this bit of code has been
     present since the very first version of JLex (1.1.1).

  Version 1.2.4, 7/24/99, [C. Scott Ananian]
   Correct the parsing of '-' in character classes, closing active
     bug #1.  Behaviour follows egrep: leading and trailing dashes in
     a character class lose their special meaning, so [-+] and [+-] do
     what you would expect them to.
   New %ignorecase directive for generating case-insensitive lexers by
     expanding matched character classes in a unicode-friendly way.
   Handle unmatched braces in quoted strings or comments within
```

```
      action code blocks.
    Fixed input lexer to allow whitespace in character classes, closing
      active bug #9.  Whitespace in quotes had been previously fixed.
    Made Yylex.YYEOF and %yyeof work like the manual says they should.

   Version 1.2.3, 6/26/97, [Raimondas Lencevicius]
    Fixed the yy_nxt[][] assignment that has generated huge code
    exceeding 64K method size limit. Now the assignment
    is handled by unpacking a string encoding of integer array.
    To achieve that, added
    "private int [][] unpackFromString(int size1, int size2, String st)"
    function and coded the yy_nxt[][] values into a string
    by printing integers into a string and representing
    integer sequences as "value:length" pairs.
    Improvement: generated .java file reduced 2 times, .class file
      reduced 6 times for sample grammar. No 64K errors.
    Possible negatives: Some editors and OSs may not be able to handle
      the huge one-line generated string. String unpacking may be slower
      than direct array initialization.

   Version 1.2.2, 10/24/97, [Martin Dirichs]
   Notes:
     Changed yy_instream to yy_reader of type BufferedReader. This reflects
      the improvements in the JDK 1.1 concerning InputStreams. As a
      consequence, changed yy_buffer from byte[] to char[].
      The lexer can now be initialized with either an InputStream
      or a Reader. A third, private constructor is called by the other
      two to execute user specified constructor code.

   Version 1.2.1, 9/15/97 [A. Appel]
    Fixed bugs 6 (character codes > 127) and 10 (deprecated String constructor).

   Version 1.2, 5/5/97, [Elliot Berk]
   Notes:
     Simply changed the name from JavaLex to JLex.  No other changes.

   Version 1.1.5, 2/25/97, [Elliot Berk]
   Notes:
     Simple optimization to the creation of the source files.
      Added a BufferedOutputStream in the creation of the DataOutputStream
      field m_outstream of the class CLexGen.  This helps performance by
      doing some buffering, and was suggested by Max Hailperin,
      Associate Professor of Computer Science, Gustavus Adolphus College.

   Version 1.1.4, 12/12/96, [Elliot Berk]
   Notes:
     Added %public directive to make generated class public.

   Version 1.1.3, 12/11/96, [Elliot Berk]
   Notes:
     Converted assertion failure on invalid character class
      when a dash '-' is not preceded with a start-of-range character.
      Converted this into parse error E_DASH.

   Version 1.1.2, October 30, 1996 [Elliot Berk]
     Fixed BitSet bugs by installing a BitSet class of my own,
      called JavaLexBitSet.  Fixed support for '\r', non-UNIX
      sequences.  Added try/catch block around lexer generation
      in main routine to moderate error information presented
      to user.  Fixed macro expansion, so that macros following
      quotes are expanded correctly in regular expressions.
      Fixed dynamic reallocation of accept action buffers.

   Version 1.1.1, September 3, 1996 [Andrew Appel]
     Made the class "Main" instead of "JavaLex",
      improved the installation instructions to reflect this.

   Version 1.1, August 15, 1996  [Andrew Appel]
     Made yychar, yyline, yytext global to the lexer so that
      auxiliary functions can access them.
   **************************************************************/

 /***************************************************************
        JLEX COPYRIGHT NOTICE, LICENSE, AND DISCLAIMER
   Copyright 1996-2000 by Elliot Joel Berk and C. Scott Ananian
```

```
  Permission to use, copy, modify, and distribute this software and its
  documentation for any purpose and without fee is hereby granted,
  provided that the above copyright notice appear in all copies and that
  both the copyright notice and this permission notice and warranty
  disclaimer appear in supporting documentation, and that the name of
  the authors or their employers not be used in advertising or publicity
  pertaining to distribution of the software without specific, written
  prior permission.

  The authors and their employers disclaim all warranties with regard to
  this software, including all implied warranties of merchantability and
  fitness. In no event shall the authors or their employers be liable
  for any special, indirect or consequential damages or any damages
  whatsoever resulting from loss of use, data or profits, whether in an
  action of contract, negligence or other tortious action, arising out
  of or in connection with the use or performance of this software.
  **************************************************************/

/***************************************************************
  Package Declaration
  **************************************************************/
package JLex;

/***************************************************************
  Imported Packages
  **************************************************************/
import java.lang.System;
import java.lang.Integer;
import java.lang.Character;

import java.util.Enumeration;
import java.util.Stack;
import java.util.Hashtable;
import java.util.Vector;

/******************************
  Questions:
  2) How should I use the Java package system
  to make my tool more modularized and
  coherent?

  Unimplemented:
  !) Fix BitSet issues -- expand only when necessary.
  2) Repeated accept rules.
  6) Clean up the CAlloc class and use buffered
  allocation.
  9) Add to spec about extending character set.
  11) m_verbose -- what should be done with it?
  12) turn lexical analyzer into a coherent
  Java package
  13) turn lexical analyzer generator into a
  coherent Java package
  16) pretty up generated code
  17) make it possible to have white space in
  regular expressions
  18) clean up all of the class files the lexer
  generator produces when it is compiled,
  and reduce this number in some way.
  24) character format to and from file: writeup
  and implementation
  25) Debug by testing all arcane regular expression cases.
  26) Look for and fix all UNDONE comments below.
  27) Fix package system.
  28) Clean up unnecessary classes.
  *****************************/

/***************************************************************
  Class: CSpec
 **************************************************************/
class CSpec
{
  /***************************************************************
    Member Variables
    **************************************************************/
  /* Lexical States. */
  Hashtable m_states; /* Hashtable taking state indices (Integer)
                         to state name (String). */

  /* Regular Expression Macros. */
  Hashtable m_macros; /* Hashtable taking macro name (String)
                                to corresponding char buffer that
                                holds macro definition. */

  /* NFA Machine. */
  CNfa m_nfa_start; /* Start state of NFA machine. */
  Vector m_nfa_states; /* Vector of states, with index
                                 corresponding to label. */

  Vector m_state_rules[]; /* An array of Vectors of Integers.
                                    The ith Vector represents the lexical state
                                    with index i.  The contents of the ith
                                    Vector are the indices of the NFA start
                                    states that can be matched while in
                                    the ith lexical state. */


  int m_state_dtrans[];

  /* DFA Machine. */
  Vector m_dfa_states; /* Vector of states, with index
                                 corresponding to label. */
  Hashtable m_dfa_sets; /* Hashtable taking set of NFA states
                                  to corresponding DFA state,
                                  if the latter exists. */

  /* Accept States and Corresponding Anchors. */
  Vector m_accept_vector;
  int m_anchor_array[];

  /* Transition Table. */
  Vector m_dtrans_vector;
  int m_dtrans_ncols;
  int m_row_map[];
  int m_col_map[];

  /* Special pseudo-characters for beginning-of-line and end-of-file. */
  static final int NUM_PSEUDO=2;
  int BOL; // beginning-of-line
  int EOF; // end-of-line

  /** NFA character class minimization map. */
  int m_ccls_map[];

  /* Regular expression token variables. */
  int m_current_token;
  char m_lexeme;
  boolean m_in_quote;
  boolean m_in_ccl;

  /* Verbose execution flag. */
  boolean m_verbose;

  /* JLex directives flags. */
  boolean m_integer_type;
  boolean m_intwrap_type;
  boolean m_yyeof;
  boolean m_count_chars;
  boolean m_count_lines;
  boolean m_cup_compatible;
  boolean m_unix;
  boolean m_public;
  boolean m_ignorecase;

  char m_init_code[];
  int m_init_read;

  char m_init_throw_code[];
  int m_init_throw_read;
  char m_class_code[];
  int m_class_read;

  char m_eof_code[];
  int m_eof_read;

  char m_eof_value_code[];
  int m_eof_value_read;

  char m_eof_throw_code[];
  int m_eof_throw_read;

  char m_yylex_throw_code[];
  int m_yylex_throw_read;

  /* Class, function, type names. */
  char m_class_name[] = {
    'Y', 'y', 'l',
    'e', 'x'
    };
  char m_implements_name[] = {};
  char m_function_name[] = {
    'y', 'y', 'l',
    'e', 'x'
    };
  char m_type_name[] = {
    'Y', 'y', 't',
    'o', 'k', 'e',
    'n'
    };

  /* Lexical Generator. */
  private CLexGen m_lexGen;

  /***************************************************************
    Constants
    ***********************************************************/
  static final int NONE = 0;
  static final int START = 1;
  static final int END = 2;

  /***************************************************************
    Function: CSpec
    Description: Constructor.
    **************************************************************/
  CSpec
    (
     CLexGen lexGen
     )
      {
        m_lexGen = lexGen;

        /* Initialize regular expression token variables. */
        m_current_token = m_lexGen.EOS;
        m_lexeme = '\0';
        m_in_quote = false;
        m_in_ccl = false;

        /* Initialize hashtable for lexer states. */
        m_states = new Hashtable();
        m_states.put(new String("YYINITIAL"),new Integer(m_states.size()));

        /* Initialize hashtable for lexical macros. */
        m_macros = new Hashtable();

        /* Initialize variables for lexer options. */
        m_integer_type = false;
        m_intwrap_type = false;
        m_count_lines = false;
        m_count_chars = false;
        m_cup_compatible = false;
        m_unix = true;
        m_public = false;
        m_yyeof = false;
```

```
	m_ignorecase = false;

	/* Initialize variables for JLex runtime options. */
	m_verbose = true;

	m_nfa_start = null;
	m_nfa_states = new Vector();

	m_dfa_states = new Vector();
	m_dfa_sets = new Hashtable();

	m_dtrans_vector = new Vector();
	m_dtrans_ncols = CUtility.MAX_SEVEN_BIT + 1;
	m_row_map = null;
	m_col_map = null;

	m_accept_vector = null;
	m_anchor_array = null;

	m_init_code = null;
	m_init_read = 0;

	m_init_throw_code = null;
	m_init_throw_read = 0;

	m_yylex_throw_code = null;
	m_yylex_throw_read = 0;

	m_class_code = null;
	m_class_read = 0;

	m_eof_code = null;
	m_eof_read = 0;

	m_eof_value_code = null;
	m_eof_value_read = 0;

	m_eof_throw_code = null;
	m_eof_throw_read = 0;

	m_state_dtrans = null;

	m_state_rules = null;
      }
}

/***************************************************************
  Class: CEmit
  **************************************************************/
class CEmit
{
  /***************************************************************
    Member Variables
    **************************************************************/
  private CSpec m_spec;
  private java.io.PrintWriter m_outstream;

  /***************************************************************
    Constants: Anchor Types
    **************************************************************/
  private final int START = 1;
  private final int END = 2;
  private final int NONE = 4;

  /***************************************************************
    Constants
    **************************************************************/
  private final boolean EDBG = true;
  private final boolean NOT_EDBG = false;

  /***************************************************************
    Function: CEmit
    Description: Constructor.
    **************************************************************/
  CEmit
```

```
    (
     )
      {
        reset();
      }

  /***************************************************************
    Function: reset
    Description: Clears member variables.
    **************************************************************/
  private void reset
    (
     )
      {
        m_spec = null;
        m_outstream = null;
      }

  /***************************************************************
    Function: set
    Description: Initializes member variables.
    **************************************************************/
  private void set
    (
     CSpec spec,
     java.io.PrintWriter outstream
     )
      {
        if (CUtility.DEBUG)
          {
            CUtility.ASSERT(null != spec);
            CUtility.ASSERT(null != outstream);
          }

        m_spec = spec;
        m_outstream = outstream;
      }

  /***************************************************************
    Function: emit_imports
    Description: Emits import packages at top of
    generated source file.
    **************************************************************/
  /*void emit_imports
    (
     CSpec spec,
     OutputStream outstream
     )
      throws java.io.IOException
        {
          set(spec,outstream);

          if (CUtility.DEBUG)
            {
              CUtility.ASSERT(null != m_spec);
              CUtility.ASSERT(null != m_outstream);
            }*/

          /*m_outstream.println("import java.lang.String;");
          m_outstream.println("import java.lang.System;");
          m_outstream.println("import java.io.BufferedReader;");
          m_outstream.println("import java.io.InputStream;");*/
        /*
          reset();
        }*/

  /***************************************************************
    Function: print_details
    Description: Debugging output.
    **************************************************************/
  private void print_details
    (
     )
      {
        int i;
        int j;
        int next;
        int state;
        CDTrans dtrans;
        CAccept accept;
        boolean tr;

        System.out.println("--------------------- Transition Table "
                           + "---------------------");

        for (i = 0; i < m_spec.m_row_map.length; ++i)
          {
            System.out.print("State " + i);

            accept = (CAccept) m_spec.m_accept_vector.elementAt(i);
            if (null == accept)
              {
                System.out.println(" [nonaccepting]");
              }
            else
              {
                System.out.println(" [accepting, line "
                                 + accept.m_line_number
                                 + " <"
                                 + (new java.lang.String(accept.m_action,0,
                                               accept.m_action_read))
                                 + ">]");
              }
            dtrans = (CDTrans) m_spec.m_dtrans_vector.elementAt(m_spec.m_row_map[i]);

            tr = false;
            state = dtrans.m_dtrans[m_spec.m_col_map[0]];
            if (CDTrans.F != state)
              {
                tr = true;
                System.out.print("\tgoto " + state + " on [" + ((char) 0));
              }
            for (j = 1; j < m_spec.m_dtrans_ncols; ++j)
              {
                next = dtrans.m_dtrans[m_spec.m_col_map[j]];
                if (state == next)
                  {
                    if (CDTrans.F != state)
                      {
                        System.out.print((char) j);
                      }
                  }
                else
                  {
                    state = next;
                    if (tr)
                      {
                        System.out.println("]");
                        tr = false;
                      }
                    if (CDTrans.F != state)
                      {
                        tr = true;
                        System.out.print("\tgoto " + state + " on [" + ((char) j));
                      }
                  }
              }
            if (tr)
              {
                System.out.println("]");
              }
          }

        System.out.println("--------------------- Transition Table "
                           + "---------------------");
      }

  /***************************************************************
    Function: emit
    Description: High-level access function to module.
    ***************************************************************/
  void emit
    (
     CSpec spec,
     java.io.PrintWriter outstream
     )
      throws java.io.IOException
        {
          set(spec,outstream);

          if (CUtility.DEBUG)
            {
              CUtility.ASSERT(null != m_spec);
              CUtility.ASSERT(null != m_outstream);
            }

          if (CUtility.OLD_DEBUG) {
            print_details();
          }

          emit_header();
          emit_construct();
          emit_helpers();
          emit_driver();
          emit_footer();

          reset();
        }

  /***************************************************************
    Function: emit_construct
    Description: Emits constructor, member variables,
    and constants.
    ***************************************************************/
  private void emit_construct
    (
     )
      throws java.io.IOException
        {
          if (CUtility.DEBUG)
          {
            CUtility.ASSERT(null != m_spec);
            CUtility.ASSERT(null != m_outstream);
          }

          /* Constants */
          m_outstream.println("\tprivate final int YY_BUFFER_SIZE = 512;");

          m_outstream.println("\tprivate final int YY_F = -1;");
          m_outstream.println("\tprivate final int YY_NO_STATE = -1;");

          m_outstream.println("\tprivate final int YY_NOT_ACCEPT = 0;");
          m_outstream.println("\tprivate final int YY_START = 1;");
          m_outstream.println("\tprivate final int YY_END = 2;");
          m_outstream.println("\tprivate final int YY_NO_ANCHOR = 4;");

          // internal
          m_outstream.println("\tprivate final int YY_BOL = "+m_spec.BOL+";");
          m_outstream.println("\tprivate final int YY_EOF = "+m_spec.EOF+";");
          // external
          if (m_spec.m_integer_type || true == m_spec.m_yyeof)
            m_outstream.println("\tpublic final int YYEOF = -1;");

          /* User specified class code. */
          if (null != m_spec.m_class_code)
            {
              m_outstream.print(new String(m_spec.m_class_code,0,
                                           m_spec.m_class_read));
            }

          /* Member Variables */
          m_outstream.println("\tprivate java.io.BufferedReader yy_reader;");
          m_outstream.println("\tprivate int yy_buffer_index;");
          m_outstream.println("\tprivate int yy_buffer_read;");
          m_outstream.println("\tprivate int yy_buffer_start;");
          m_outstream.println("\tprivate int yy_buffer_end;");
          m_outstream.println("\tprivate char yy_buffer[];");
          if (m_spec.m_count_chars)
            {
              m_outstream.println("\tprivate int yychar;");
            }
          if (m_spec.m_count_lines)
            {
              m_outstream.println("\tprivate int yyline;");
            }
          m_outstream.println("\tprivate boolean yy_at_bol;");
          m_outstream.println("\tprivate int yy_lexical_state;");
          /*if (m_spec.m_count_lines || true == m_spec.m_count_chars)
            {
              m_outstream.println("\tprivate int yy_buffer_prev_start;");
            }*/
          m_outstream.println();


          /* Function: first constructor (Reader) */
          m_outstream.print("\t");
          if (true == m_spec.m_public) {
            m_outstream.print("public ");
          }
          m_outstream.print(new String(m_spec.m_class_name));
          m_outstream.print(" (java.io.Reader reader)");

          if (null != m_spec.m_init_throw_code)
            {
              m_outstream.println();
              m_outstream.print("\t\tthrows ");
              m_outstream.print(new String(m_spec.m_init_throw_code,0,
                                                m_spec.m_init_throw_read));
              m_outstream.println();
              m_outstream.println("\t\t{");
            }
          else
            {
              m_outstream.println(" {");
            }

          m_outstream.println("\t\tthis ();");
          m_outstream.println("\t\tif (null == reader) {");
          m_outstream.println("\t\t\tthrow (new Error(\"Error: Bad input "
                                 + "stream initializer.\"));");
          m_outstream.println("\t\t}");
          m_outstream.println("\t\tyy_reader = new java.io.BufferedReader(reader);");
          m_outstream.println("\t}");
          m_outstream.println();


          /* Function: second constructor (InputStream) */
          m_outstream.print("\t");
          if (true == m_spec.m_public) {
            m_outstream.print("public ");
          }
          m_outstream.print(new String(m_spec.m_class_name));
          m_outstream.print(" (java.io.InputStream instream)");

          if (null != m_spec.m_init_throw_code)
            {
              m_outstream.println();
              m_outstream.print("\t\tthrows ");
              m_outstream.println(new String(m_spec.m_init_throw_code,0,
                                                m_spec.m_init_throw_read));
              m_outstream.println("\t\t{");
            }
          else
            {
              m_outstream.println(" {");
            }

          m_outstream.println("\t\tthis ();");
          m_outstream.println("\t\tif (null == instream) {");
          m_outstream.println("\t\t\tthrow (new Error(\"Error: Bad input "
                                 + "stream initializer.\"));");
          m_outstream.println("\t\t}");
          m_outstream.println("\t\tyy_reader = new java.io.BufferedReader(new java.io.InputSt
          m_outstream.println("\t}");
          m_outstream.println();


          /* Function: third, private constructor - only for internal use */
          m_outstream.print("\tprivate ");
          m_outstream.print(new String(m_spec.m_class_name));
          m_outstream.print(" ()");

          if (null != m_spec.m_init_throw_code)
            {
              m_outstream.println();
              m_outstream.print("\t\tthrows ");
              m_outstream.println(new String(m_spec.m_init_throw_code,0,
                                                m_spec.m_init_throw_read));
              m_outstream.println("\t\t{");
            }
          else
            {
              m_outstream.println(" {");
            }

          m_outstream.println("\t\tyy_buffer = new char[YY_BUFFER_SIZE];");
          m_outstream.println("\t\tyy_buffer_read = 0;");
          m_outstream.println("\t\tyy_buffer_index = 0;");
          m_outstream.println("\t\tyy_buffer_start = 0;");
          m_outstream.println("\t\tyy_buffer_end = 0;");
          if (m_spec.m_count_chars)
            {
              m_outstream.println("\t\tyychar = 0;");
            }
          if (m_spec.m_count_lines)
            {
              m_outstream.println("\t\tyyline = 0;");
            }
          m_outstream.println("\t\tyy_at_bol = true;");
          m_outstream.println("\t\tyy_lexical_state = YYINITIAL;");
          /*if (m_spec.m_count_lines || true == m_spec.m_count_chars)
            {
              m_outstream.println("\t\tyy_buffer_prev_start = 0;");
            }*/

          /* User specified constructor code. */
          if (null != m_spec.m_init_code)
            {
              m_outstream.print(new String(m_spec.m_init_code,0,
                                                m_spec.m_init_read));
            }

          m_outstream.println("\t}");
          m_outstream.println();

        }

  /***************************************************************
    Function: emit_states
    Description: Emits constants that serve as lexical states,
    including YYINITIAL.
    **************************************************************/
  private void emit_states
    (
     )
      throws java.io.IOException
        {
          Enumeration states;
          String state;
          int index;

          states = m_spec.m_states.keys();
          /*index = 0;*/
          while (states.hasMoreElements())
            {
```

```
            state = (String) states.nextElement();

            if (CUtility.DEBUG)
              {
                CUtility.ASSERT(null != state);
              }

            m_outstream.println("\tprivate final int "
                                + state
                                + " = "
                                + (m_spec.m_states.get(state)).toString()
                                + ";");
            /*++index;*/
          }

        m_outstream.println("\tprivate final int yy_state_dtrans[] = {");
        for (index = 0; index < m_spec.m_state_dtrans.length; ++index)
          {
            m_outstream.print("\t\t" + m_spec.m_state_dtrans[index]);
            if (index < m_spec.m_state_dtrans.length - 1)
              {
                m_outstream.println(",");
              }
            else
              {
                m_outstream.println();
              }
          }
        m_outstream.println("\t};");
      }

  /***************************************************************
    Function: emit_helpers
    Description: Emits helper functions, particularly
    error handling and input buffering.
    **************************************************************/
  private void emit_helpers
    (
     )
      throws java.io.IOException
      {
        if (CUtility.DEBUG)
          {
            CUtility.ASSERT(null != m_spec);
            CUtility.ASSERT(null != m_outstream);
          }

        /* Function: yy_do_eof */
        m_outstream.println("\tprivate boolean yy_eof_done = false;");
        if (null != m_spec.m_eof_code)
          {
            m_outstream.print("\tprivate void yy_do_eof ()");

            if (null != m_spec.m_eof_throw_code)
              {
                m_outstream.println();
                m_outstream.print("\t\tthrows ");
                m_outstream.println(new String(m_spec.m_eof_throw_code,0,
                                                  m_spec.m_eof_throw_read));
                m_outstream.println("\t\t{");
              }
            else
              {
                m_outstream.println(" {");
              }

            m_outstream.println("\t\tif (false == yy_eof_done) {");
            m_outstream.print(new String(m_spec.m_eof_code,0,
                                          m_spec.m_eof_read));
            m_outstream.println("\t\t}");
            m_outstream.println("\t\tyy_eof_done = true;");
            m_outstream.println("\t}");
          }

        emit_states();
```

```
/* Function: yybegin */
m_outstream.println("\tprivate void yybegin (int state) {");
m_outstream.println("\t\tyy_lexical_state = state;");
m_outstream.println("\t}");

/* Function: yy_initial_dtrans */
/*m_outstream.println("\tprivate int yy_initial_dtrans (int state) {");
m_outstream.println("\t\treturn yy_state_dtrans[state];");
m_outstream.println("\t}");*/

/* Function: yy_advance */
m_outstream.println("\tprivate int yy_advance ()");
m_outstream.println("\t\tthrows java.io.IOException {");
/*m_outstream.println("\t\t{");*/
m_outstream.println("\t\tint next_read;");
m_outstream.println("\t\tint i;");
m_outstream.println("\t\tint j;");
m_outstream.println();

m_outstream.println("\t\tif (yy_buffer_index < yy_buffer_read) {");
m_outstream.println("\t\t\treturn yy_buffer[yy_buffer_index++];");
/*m_outstream.println("\t\t\t++yy_buffer_index;");*/
m_outstream.println("\t\t}");
m_outstream.println();

m_outstream.println("\t\tif (0 != yy_buffer_start) {");
m_outstream.println("\t\t\ti = yy_buffer_start;");
m_outstream.println("\t\t\tj = 0;");
m_outstream.println("\t\t\twhile (i < yy_buffer_read) {");
m_outstream.println("\t\t\t\tyy_buffer[j] = yy_buffer[i];");
m_outstream.println("\t\t\t\t++i;");
m_outstream.println("\t\t\t\t++j;");
m_outstream.println("\t\t\t}");
m_outstream.println("\t\t\tyy_buffer_end = yy_buffer_end - yy_buffer_start;");
m_outstream.println("\t\t\tyy_buffer_start = 0;");
m_outstream.println("\t\t\tyy_buffer_read = j;");
m_outstream.println("\t\t\tyy_buffer_index = j;");
m_outstream.println("\t\t\tnext_read = yy_reader.read(yy_buffer,");
m_outstream.println("\t\t\t\t\tyy_buffer_read,");
m_outstream.println("\t\t\t\t\tyy_buffer.length - yy_buffer_read);");
m_outstream.println("\t\t\tif (-1 == next_read) {");
m_outstream.println("\t\t\t\treturn YY_EOF;");
m_outstream.println("\t\t\t}");
m_outstream.println("\t\t\tyy_buffer_read = yy_buffer_read + next_read;");
m_outstream.println("\t\t}");
m_outstream.println();

m_outstream.println("\t\twhile (yy_buffer_index >= yy_buffer_read) {");
m_outstream.println("\t\t\tif (yy_buffer_index >= yy_buffer.length) {");
m_outstream.println("\t\t\t\tyy_buffer = yy_double(yy_buffer);");
m_outstream.println("\t\t\t}");
m_outstream.println("\t\t\tnext_read = yy_reader.read(yy_buffer,");
m_outstream.println("\t\t\t\t\tyy_buffer_read,");
m_outstream.println("\t\t\t\t\tyy_buffer.length - yy_buffer_read);");
m_outstream.println("\t\t\tif (-1 == next_read) {");
m_outstream.println("\t\t\t\treturn YY_EOF;");
m_outstream.println("\t\t\t}");
m_outstream.println("\t\t\tyy_buffer_read = yy_buffer_read + next_read;");
m_outstream.println("\t\t}");

m_outstream.println("\t\treturn yy_buffer[yy_buffer_index++];");
m_outstream.println("\t}");

/* Function: yy_move_end */
m_outstream.println("\tprivate void yy_move_end () {");
m_outstream.println("\t\tif (yy_buffer_end > yy_buffer_start &&");
m_outstream.println("\t\t    '\\n' == yy_buffer[yy_buffer_end-1])");
m_outstream.println("\t\t\tyy_buffer_end--;");
m_outstream.println("\t\tif (yy_buffer_end > yy_buffer_start &&");
m_outstream.println("\t\t    '\\r' == yy_buffer[yy_buffer_end-1])");
m_outstream.println("\t\t\tyy_buffer_end--;");
m_outstream.println("\t}");

/* Function: yy_mark_start */
```

```
m_outstream.println("\tprivate boolean yy_last_was_cr=false;");
m_outstream.println("\tprivate void yy_mark_start () {");
if (m_spec.m_count_lines || true == m_spec.m_count_chars)
  {
    if (m_spec.m_count_lines)
      {
        m_outstream.println("\t\tint i;");
        m_outstream.println("\t\tfor (i = yy_buffer_start; "
                            + "i < yy_buffer_index; ++i) {");
        m_outstream.println("\t\t\tif ('\\n' == yy_buffer[i] && !yy_last_was_cr) {");
        m_outstream.println("\t\t\t\t++yyline;");
        m_outstream.println("\t\t\t}");
        m_outstream.println("\t\t\tif ('\\r' == yy_buffer[i]) {");
        m_outstream.println("\t\t\t\t++yyline;");
        m_outstream.println("\t\t\t\tyy_last_was_cr=true;");
        m_outstream.println("\t\t\t} else yy_last_was_cr=false;");
        m_outstream.println("\t\t}");
      }
    if (m_spec.m_count_chars)
      {
        m_outstream.println("\t\tyychar = yychar");
        m_outstream.println("\t\t\t+ yy_buffer_index - yy_buffer_start;");
      }
  }
m_outstream.println("\t\tyy_buffer_start = yy_buffer_index;");
m_outstream.println("\t}");

/* Function: yy_mark_end */
m_outstream.println("\tprivate void yy_mark_end () {");
m_outstream.println("\t\tyy_buffer_end = yy_buffer_index;");
m_outstream.println("\t}");

/* Function: yy_to_mark */
m_outstream.println("\tprivate void yy_to_mark () {");
m_outstream.println("\t\tyy_buffer_index = yy_buffer_end;");
m_outstream.println("\t\tyy_at_bol = "+
                    "(yy_buffer_end > yy_buffer_start) &&");
m_outstream.println("\t\t            "+
                    "('\\r' == yy_buffer[yy_buffer_end-1] ||");
m_outstream.println("\t\t            "+
                    " '\\n' == yy_buffer[yy_buffer_end-1] ||");
m_outstream.println("\t\t            "+ /* unicode LS */
                    " 2028/*LS*/ == yy_buffer[yy_buffer_end-1] ||");
m_outstream.println("\t\t            "+ /* unicode PS */
                    " 2029/*PS*/ == yy_buffer[yy_buffer_end-1]);");
m_outstream.println("\t}");

/* Function: yytext */
m_outstream.println("\tprivate java.lang.String yytext () {");
m_outstream.println("\t\treturn (new java.lang.String(yy_buffer,");
m_outstream.println("\t\t\tyy_buffer_start,");
m_outstream.println("\t\t\tyy_buffer_end - yy_buffer_start));");
m_outstream.println("\t}");

/* Function: yylength */
m_outstream.println("\tprivate int yylength () {");
m_outstream.println("\t\treturn yy_buffer_end - yy_buffer_start;");
m_outstream.println("\t}");

/* Function: yy_double */
m_outstream.println("\tprivate char[] yy_double (char buf[]) {");
m_outstream.println("\t\tint i;");
m_outstream.println("\t\tchar newbuf[];");
m_outstream.println("\t\tnewbuf = new char[2*buf.length];");
m_outstream.println("\t\tfor (i = 0; i < buf.length; ++i) {");
m_outstream.println("\t\t\tnewbuf[i] = buf[i];");
m_outstream.println("\t\t}");
m_outstream.println("\t\treturn newbuf;");
m_outstream.println("\t}");

/* Function: yy_error */
m_outstream.println("\tprivate final int YY_E_INTERNAL = 0;");
m_outstream.println("\tprivate final int YY_E_MATCH = 1;");
m_outstream.println("\tprivate java.lang.String yy_error_string[] = {");
m_outstream.println("\t\t\"Error: Internal error.\\n\",");
```

```
        m_outstream.println("\t\t\"Error: Unmatched input.\\n\"");
        m_outstream.println("\t};");
        m_outstream.println("\tprivate void yy_error (int code,boolean fatal) {");
        m_outstream.println("\t\tjava.lang.System.out.print(yy_error_string[code]);");
        m_outstream.println("\t\tjava.lang.System.out.flush();");
        m_outstream.println("\t\tif (fatal) {");
        m_outstream.println("\t\t\tthrow new Error(\"Fatal Error.\\n\");");
        m_outstream.println("\t\t}");
        m_outstream.println("\t}");

        /* Function: yy_next */
        /*m_outstream.println("\tprivate int yy_next (int current,char lookahead) {");
        m_outstream.println("\t\treturn yy_nxt[yy_rmap[current]][yy_cmap[lookahead]];");
        m_outstream.println("\t}");*/

        /* Function: yy_accept */
        /*m_outstream.println("\tprivate int yy_accept (int current) {");
        m_outstream.println("\t\treturn yy_acpt[current];");
        m_outstream.println("\t}");*/


        // Function: private int [][] unpackFromString(int size1, int size2, String st)
        // Added 6/24/98 Raimondas Lencevicius
        // May be made more efficient by replacing String operations
        // Assumes correctly formed input String. Performs no error checking
        m_outstream.println("\tprivate int[][] unpackFromString"+
                            "(int size1, int size2, String st) {");
        m_outstream.println("\t\tint colonIndex = -1;");
        m_outstream.println("\t\tString lengthString;");
        m_outstream.println("\t\tint sequenceLength = 0;");
        m_outstream.println("\t\tint sequenceInteger = 0;");
        m_outstream.println();
        m_outstream.println("\t\tint commaIndex;");
        m_outstream.println("\t\tString workString;");
        m_outstream.println();
        m_outstream.println("\t\tint res[][] = new int[size1][size2];");
        m_outstream.println("\t\tfor (int i= 0; i < size1; i++) {");
        m_outstream.println("\t\t\tfor (int j= 0; j < size2; j++) {");
        m_outstream.println("\t\t\t\tif (sequenceLength != 0) {");
        m_outstream.println("\t\t\t\t\tres[i][j] = sequenceInteger;");
        m_outstream.println("\t\t\t\t\tsequenceLength--;");
        m_outstream.println("\t\t\t\t\tcontinue;");
        m_outstream.println("\t\t\t\t}");
        m_outstream.println("\t\t\t\tcommaIndex = st.indexOf(',');");
        m_outstream.println("\t\t\t\tworkString = (commaIndex==-1) ? st :");
        m_outstream.println("\t\t\t\t\tst.substring(0, commaIndex);");
        m_outstream.println("\t\t\t\tst = st.substring(commaIndex+1);");
        m_outstream.println("\t\t\t\tcolonIndex = workString.indexOf(':');");
        m_outstream.println("\t\t\t\tif (colonIndex == -1) {");
        m_outstream.println("\t\t\t\t\tres[i][j]=Integer.parseInt(workString);");
        m_outstream.println("\t\t\t\t\tcontinue;");
        m_outstream.println("\t\t\t\t}");
        m_outstream.println("\t\t\t\tlengthString =");
        m_outstream.println("\t\t\t\t\tworkString.substring(colonIndex+1);");
        m_outstream.println("\t\t\t\tsequenceLength="+
                            "Integer.parseInt(lengthString);");
        m_outstream.println("\t\t\t\tworkString="+
                            "workString.substring(0,colonIndex);");
        m_outstream.println("\t\t\t\tsequenceInteger="+
                            "Integer.parseInt(workString);");
        m_outstream.println("\t\t\t\tres[i][j] = sequenceInteger;");
        m_outstream.println("\t\t\t\tsequenceLength--;");
        m_outstream.println("\t\t\t}");
        m_outstream.println("\t\t}");
        m_outstream.println("\t\treturn res;");
        m_outstream.println("\t}");
      }

  /***************************************************************
    Function: emit_header
    Description: Emits class header.
    **************************************************************/
  private void emit_header
    (
     )
```

```
      throws java.io.IOException
      {
        if (CUtility.DEBUG)
          {
            CUtility.ASSERT(null != m_spec);
            CUtility.ASSERT(null != m_outstream);
          }

        m_outstream.println();
        m_outstream.println();
        if (true == m_spec.m_public) {
          m_outstream.print("public ");
        }
        m_outstream.print("class ");
        m_outstream.print(new String(m_spec.m_class_name,0,
                                     m_spec.m_class_name.length));
        if (m_spec.m_implements_name.length > 0) {
           m_outstream.print(" implements ");
           m_outstream.print(new String(m_spec.m_implements_name,0,
                                     m_spec.m_implements_name.length));
        }
        m_outstream.println(" {");
      }

  /***************************************************************
    Function: emit_table
    Description: Emits transition table.
    **************************************************************/
  private void emit_table
    (
     )
      throws java.io.IOException
      {
        int i;
        int elem;
        int size;
        CDTrans dtrans;
        boolean is_start;
        boolean is_end;
        CAccept accept;

        if (CUtility.DEBUG)
          {
            CUtility.ASSERT(null != m_spec);
            CUtility.ASSERT(null != m_outstream);
          }

        m_outstream.println("\tprivate int yy_acpt[] = {");
        size = m_spec.m_accept_vector.size();
        for (elem = 0; elem < size; ++elem)
          {
            accept = (CAccept) m_spec.m_accept_vector.elementAt(elem);

            m_outstream.print("\t\t/* "+elem+" */ ");
            if (null != accept)
              {
                is_start = (0 != (m_spec.m_anchor_array[elem] & CSpec.START));
                is_end = (0 != (m_spec.m_anchor_array[elem] & CSpec.END));

                if (is_start && true == is_end)
                  {
                    m_outstream.print("YY_START | YY_END");
                  }
                else if (is_start)
                  {
                    m_outstream.print("YY_START");
                  }
                else if (is_end)
                  {
                    m_outstream.print("YY_END");
                  }
                else
                  {
                    m_outstream.print("YY_NO_ANCHOR");
                  }
              }
            else
              {
                m_outstream.print("YY_NOT_ACCEPT");
              }

            if (elem < size - 1)
              {
                m_outstream.print(",");
              }

            m_outstream.println();
          }
        m_outstream.println("\t};");

        // CSA: modified yy_cmap to use string packing 9-Aug-1999
        int[] yy_cmap = new int[m_spec.m_ccls_map.length];
        for (i = 0; i < m_spec.m_ccls_map.length; ++i)
            yy_cmap[i] = m_spec.m_col_map[m_spec.m_ccls_map[i]];
        m_outstream.print("\tprivate int yy_cmap[] = unpackFromString(");
        emit_table_as_string(new int[][] { yy_cmap });
        m_outstream.println(")[0];");
        m_outstream.println();

        // CSA: modified yy_rmap to use string packing 9-Aug-1999
        m_outstream.print("\tprivate int yy_rmap[] = unpackFromString(");
        emit_table_as_string(new int[][] { m_spec.m_row_map });
        m_outstream.println(")[0];");
        m_outstream.println();

        // 6/24/98 Raimondas Lencevicius
        // modified to use
        //    int[][] unpackFromString(int size1, int size2, String st)
        size = m_spec.m_dtrans_vector.size();
        int[][] yy_nxt = new int[size][];
        for (elem=0; elem<size; elem++) {
            dtrans = (CDTrans) m_spec.m_dtrans_vector.elementAt(elem);
            CUtility.ASSERT(dtrans.m_dtrans.length==m_spec.m_dtrans_ncols);
            yy_nxt[elem] = dtrans.m_dtrans;
        }
        m_outstream.print
          ("\tprivate int yy_nxt[][] = unpackFromString(");
        emit_table_as_string(yy_nxt);
        m_outstream.println(");");
        m_outstream.println();
      }

  /***************************************************************
    Function: emit_driver
    Description: Output an integer table as a string.  Written by
    Raimondas Lencevicius 6/24/98; reorganized by CSA 9-Aug-1999.
    From his original comments:
           yy_nxt[][] values are coded into a string
           by printing integers and representing
           integer sequences as "value:length" pairs.
    **************************************************************/
  private void emit_table_as_string(int[][] ia) {
        int sequenceLength = 0; // RL - length of the number sequence
        boolean sequenceStarted = false; // RL - has number sequence started?
        int previousInt = -20; // RL - Bogus -20 state.

        // RL - Output matrix size
        m_outstream.print(ia.length);
        m_outstream.print(",");
        m_outstream.print(ia.length>0?ia[0].length:0);
        m_outstream.println(",");

        StringBuffer outstr = new StringBuffer();

        //  RL - Output matrix
        for (int elem = 0; elem < ia.length; ++elem)
          {
            for (int i = 0; i < ia[elem].length; ++i)
              {
                int writeInt = ia[elem][i];
                if (writeInt == previousInt) // RL - sequence?
                  {
                    if (sequenceStarted)
                      {
                        sequenceLength++;
                      }
                    else
                      {
                        outstr.append(writeInt);
                        outstr.append(":");
                        sequenceLength = 2;
                        sequenceStarted = true;
                      }
                  }
                else // RL - no sequence or end sequence
                  {
                    if (sequenceStarted)
                      {
                        outstr.append(sequenceLength);
                        outstr.append(",");
                        sequenceLength = 0;
                        sequenceStarted = false;
                      }
                    else
                      {
                        if (previousInt != -20)
                          {
                            outstr.append(previousInt);
                            outstr.append(",");
                          }
                      }
                  }
                previousInt = writeInt;
                // CSA: output in 75 character chunks.
                if (outstr.length() > 75) {
                  String s = outstr.toString();
                  m_outstream.println("\""+s.substring(0,75)+"\" +");
                  outstr = new StringBuffer(s.substring(75));
                }
              }
          }
        if (sequenceStarted)
          {
            outstr.append(sequenceLength);
          }
        else
          {
            outstr.append(previousInt);
          }
        // CSA: output in 75 character chunks.
        if (outstr.length() > 75) {
          String s = outstr.toString();
          m_outstream.println("\""+s.substring(0,75)+"\" +");
          outstr = new StringBuffer(s.substring(75));
        }
        m_outstream.print("\""+outstr+"\"");
  }

  /***************************************************************
    Function: emit_driver
    Description: 
    **************************************************************/
  private void emit_driver
    (
     )
      throws java.io.IOException
        {
          if (CUtility.DEBUG)
            {
              CUtility.ASSERT(null != m_spec);
              CUtility.ASSERT(null != m_outstream);
            }
          
          emit_table();
          if (m_spec.m_integer_type)
            {
              m_outstream.print("\tpublic int ");
              m_outstream.print(new String(m_spec.m_function_name));
              m_outstream.println(" ()");
            }
          else if (m_spec.m_intwrap_type)
            {
              m_outstream.print("\tpublic java.lang.Integer ");
              m_outstream.print(new String(m_spec.m_function_name));
              m_outstream.println(" ()");
            }
          else
            {
              m_outstream.print("\tpublic ");
              m_outstream.print(new String(m_spec.m_type_name));
              m_outstream.print(" ");
              m_outstream.print(new String(m_spec.m_function_name));
              m_outstream.println(" ()");
            }

          /*m_outstream.println("\t\tthrows java.io.IOException {");*/
          m_outstream.print("\t\tthrows java.io.IOException");
          if (null != m_spec.m_yylex_throw_code)
            {
              m_outstream.print(", ");
              m_outstream.print(new String(m_spec.m_yylex_throw_code,0,
                                                m_spec.m_yylex_throw_read));
              m_outstream.println();
              m_outstream.println("\t\t{");
            }
          else
            {
              m_outstream.println(" {");
            }

          m_outstream.println("\t\tint yy_lookahead;");
          m_outstream.println("\t\tint yy_anchor = YY_NO_ANCHOR;");
          /*m_outstream.println("\t\tint yy_state "
            + "= yy_initial_dtrans(yy_lexical_state);");*/
          m_outstream.println("\t\tint yy_state "
                                 + "= yy_state_dtrans[yy_lexical_state];");
          m_outstream.println("\t\tint yy_next_state = YY_NO_STATE;");
          /*m_outstream.println("\t\tint yy_prev_stave = YY_NO_STATE;");*/
          m_outstream.println("\t\tint yy_last_accept_state = YY_NO_STATE;");
          m_outstream.println("\t\tboolean yy_initial = true;");
          m_outstream.println("\t\tint yy_this_accept;");
          m_outstream.println();

          m_outstream.println("\t\tyy_mark_start();");
          /*m_outstream.println("\t\tyy_this_accept = yy_accept(yy_state);");*/
          m_outstream.println("\t\tyy_this_accept = yy_acpt[yy_state];");
          m_outstream.println("\t\tif (YY_NOT_ACCEPT != yy_this_accept) {");
          m_outstream.println("\t\t\tyy_last_accept_state = yy_state;");
          m_outstream.println("\t\t\tyy_mark_end();");
          m_outstream.println("\t\t}");

          if (NOT_EDBG)
            {
              m_outstream.println("\t\tjava.lang.System.out.println(\"Begin\");");
            }

          m_outstream.println("\t\twhile (true) {");

          m_outstream.println("\t\t\tif (yy_initial && yy_at_bol) "+
                                         "yy_lookahead = YY_BOL;");
          m_outstream.println("\t\t\telse yy_lookahead = yy_advance();");
          m_outstream.println("\t\t\tyy_next_state = YY_F;");
          /*m_outstream.println("\t\t\t\tyy_next_state = "
                                 + "yy_next(yy_state,yy_lookahead);");*/
          m_outstream.println("\t\t\tyy_next_state = "
           + "yy_nxt[yy_rmap[yy_state]][yy_cmap[yy_lookahead]];");

          if (NOT_EDBG)
            {
              m_outstream.println("java.lang.System.out.println(\"Current state: \""
                                     + " + yy_state");
              m_outstream.println("+ \"\tCurrent input: \"");
              m_outstream.println(" + ((char) yy_lookahead));");
            }
          if (NOT_EDBG)
            {
              m_outstream.println("\t\t\tjava.lang.System.out.println(\"State = \""
                                     + "+ yy_state);");
              m_outstream.println("\t\t\tjava.lang.System.out.println(\"Accepting status = \'
                                     + "+ yy_this_accept);");
              m_outstream.println("\t\t\tjava.lang.System.out.println(\"Last accepting state
                                     + "+ yy_last_accept_state);");
              m_outstream.println("\t\t\tjava.lang.System.out.println(\"Next state = \""
                                     + "+ yy_next_state);");
              m_outstream.println("\t\t\tjava.lang.System.out.println(\"Lookahead input = \"'
                                     + "+ ((char) yy_lookahead));");
            }

          // handle bare EOF.
          m_outstream.println("\t\t\tif (YY_EOF == yy_lookahead "
                                 + "&& true == yy_initial) {");
          if (null != m_spec.m_eof_code)
            {
              m_outstream.println("\t\t\t\tyy_do_eof();");
            }
          if (true == m_spec.m_integer_type)
            {
              m_outstream.println("\t\t\t\treturn YYEOF;");
            }
          else if (null != m_spec.m_eof_value_code)
            {
              m_outstream.print(new String(m_spec.m_eof_value_code,0,
                                                m_spec.m_eof_value_read));
            }
          else
            {
              m_outstream.println("\t\t\t\treturn null;");
            }
          m_outstream.println("\t\t\t}");

          m_outstream.println("\t\t\tif (YY_F != yy_next_state) {");
          m_outstream.println("\t\t\t\tyy_state = yy_next_state;");
          m_outstream.println("\t\t\t\tyy_initial = false;");
          /*m_outstream.println("\t\t\t\tyy_this_accept = yy_accept(yy_state);");*/
          m_outstream.println("\t\t\t\tyy_this_accept = yy_acpt[yy_state];");
          m_outstream.println("\t\t\t\tif (YY_NOT_ACCEPT != yy_this_accept) {");
          m_outstream.println("\t\t\t\t\tyy_last_accept_state = yy_state;");
          m_outstream.println("\t\t\t\t\tyy_mark_end();");
          m_outstream.println("\t\t\t\t}");
          /*m_outstream.println("\t\t\t\tyy_prev_state = yy_state;");*/
          /*m_outstream.println("\t\t\t\tyy_state = yy_next_state;");*/
          m_outstream.println("\t\t\t}");

          m_outstream.println("\t\t\telse {");

          m_outstream.println("\t\t\t\tif (YY_NO_STATE == yy_last_accept_state) {");


          /*m_outstream.println("\t\t\t\t\tyy_error(YY_E_MATCH,false);");
          m_outstream.println("\t\t\t\t\tyy_initial = true;");
          m_outstream.println("\t\t\t\t\tyy_state "
                                 + "= yy_state_dtrans[yy_lexical_state];");
          m_outstream.println("\t\t\t\t\tyy_next_state = YY_NO_STATE;");*/
          /*m_outstream.println("\t\t\t\t\tyy_prev_state = YY_NO_STATE;");*/
          /*m_outstream.println("\t\t\t\t\tyy_last_accept_state = YY_NO_STATE;");
          m_outstream.println("\t\t\t\t\tyy_mark_start();");*/
          /*m_outstream.println("\t\t\t\t\tyy_this_accept = yy_accept(yy_state);");*/
          /*m_outstream.println("\t\t\t\t\tyy_this_accept = yy_acpt[yy_state];");
          m_outstream.println("\t\t\t\t\tif (YY_NOT_ACCEPT != yy_this_accept) {");
          m_outstream.println("\t\t\t\t\t\tyy_last_accept_state = yy_state;");
          m_outstream.println("\t\t\t\t\t}");*/

          m_outstream.println("\t\t\t\t\tthrow (new Error(\"Lexical Error: Unmatched Input.\'
          m_outstream.println("\t\t\t\t}");
          m_outstream.println("\t\t\t\telse {");

          m_outstream.println("\t\t\t\t\tyy_anchor = yy_acpt[yy_last_accept_state];");
          /*m_outstream.println("\t\t\t\t\tyy_anchor "
            + "= yy_accept(yy_last_accept_state);");*/
          m_outstream.println("\t\t\t\t\tif (0 != (YY_END & yy_anchor)) {");
          m_outstream.println("\t\t\t\t\t\tyy_move_end();");
          m_outstream.println("\t\t\t\t\t}");
          m_outstream.println("\t\t\t\t\tyy_to_mark();");

          m_outstream.println("\t\t\t\t\tswitch (yy_last_accept_state) {");

          emit_actions("\t\t\t\t\t");

          m_outstream.println("\t\t\t\t\tdefault:");
          m_outstream.println("\t\t\t\t\t\tyy_error(YY_E_INTERNAL,false);");
          /*m_outstream.println("\t\t\t\t\t\treturn null;");*/
          m_outstream.println("\t\t\t\t\tcase -1:");
          m_outstream.println("\t\t\t\t\t}");

          m_outstream.println("\t\t\t\t\tyy_initial = true;");
          m_outstream.println("\t\t\t\t\tyy_state "
                                 + "= yy_state_dtrans[yy_lexical_state];");
          m_outstream.println("\t\t\t\t\tyy_next_state = YY_NO_STATE;");
          /*m_outstream.println("\t\t\t\t\tyy_prev_state = YY_NO_STATE;");*/
          m_outstream.println("\t\t\t\t\tyy_last_accept_state = YY_NO_STATE;");

          m_outstream.println("\t\t\t\t\tyy_mark_start();");

          /*m_outstream.println("\t\t\t\t\tyy_this_accept = yy_accept(yy_state);");*/
          m_outstream.println("\t\t\t\t\tyy_this_accept = yy_acpt[yy_state];");
          m_outstream.println("\t\t\t\t\tif (YY_NOT_ACCEPT != yy_this_accept) {");
          m_outstream.println("\t\t\t\t\t\tyy_last_accept_state = yy_state;");
          m_outstream.println("\t\t\t\t\t\tyy_mark_end();");
          m_outstream.println("\t\t\t\t\t}");

          m_outstream.println("\t\t\t\t}");
          m_outstream.println("\t\t\t}");
          m_outstream.println("\t\t}");
          m_outstream.println("\t}");

          /*m_outstream.println("\t\t\t\t");
          m_outstream.println("\t\t\t");
          m_outstream.println("\t\t\t");
          m_outstream.println("\t\t\t");
          m_outstream.println("\t\t\t");
          m_outstream.println("\t\t}");*/
        }

  /***************************************************************
    Function: emit_actions
    Description:
    **************************************************************/
  private void emit_actions
    (
     String tabs
     )
      throws java.io.IOException
        {
          int elem;
          int size;
          int bogus_index;
          CAccept accept;

          if (CUtility.DEBUG)
            {
              CUtility.ASSERT(m_spec.m_accept_vector.size()
                              == m_spec.m_anchor_array.length);
            }

          bogus_index = -2;
          size = m_spec.m_accept_vector.size();
          for (elem = 0; elem < size; ++elem)
            {
              accept = (CAccept) m_spec.m_accept_vector.elementAt(elem);
              if (null != accept)
                {
                  m_outstream.println(tabs + "case " + elem
                                      + ":");
                  m_outstream.print(tabs + "\t");
                  m_outstream.print(new String(accept.m_action,0,
                                               accept.m_action_read));
                  m_outstream.println();
                  m_outstream.println(tabs + "case " + bogus_index + ":");
                  m_outstream.println(tabs + "\tbreak;");
                  --bogus_index;
                }
            }
        }

  /***************************************************************
    Function: emit_footer
    Description:
    **************************************************************/
  private void emit_footer
    (
     )
      throws java.io.IOException
      {
        if (CUtility.DEBUG)
          {
            CUtility.ASSERT(null != m_spec);
            CUtility.ASSERT(null != m_outstream);
          }

        m_outstream.println("}");
      }
}

/***************************************************************
  Class: CBunch
  **************************************************************/
class CBunch
{
  /***************************************************************
    Member Variables
    **************************************************************/
  Vector m_nfa_set; /* Vector of CNfa states in dfa state. */
  SparseBitSet m_nfa_bit; /* BitSet representation of CNfa labels. */
  CAccept m_accept; /* Accepting actions, or null if nonaccepting state. */
  int m_anchor; /* Anchors on regular expression. */
  int m_accept_index; /* CNfa index corresponding to accepting actions. */

  /***************************************************************
    Function: CBunch
    Description: Constructor.
    **************************************************************/
  CBunch
    (
     )
      {
        m_nfa_set = null;
        m_nfa_bit = null;
        m_accept = null;
        m_anchor = CSpec.NONE;
        m_accept_index = -1;
      }
}

/***************************************************************
  Class: CMakeNfa
  **************************************************************/
class CMakeNfa
{
  /***************************************************************
    Member Variables
    **************************************************************/
  private CSpec m_spec;
  private CLexGen m_lexGen;
```

```
    private CInput m_input;

    /***************************************************************
      Function: CMakeNfa
      Description: Constructor.
      **************************************************************/
    CMakeNfa
      (
       )
        {
          reset();
        }

    /***************************************************************
      Function: reset
      Description: Resets CMakeNfa member variables.
      **************************************************************/
    private void reset
      (
       )
        {
          m_input = null;
          m_lexGen = null;
          m_spec = null;
        }

    /***************************************************************
      Function: set
      Description: Sets CMakeNfa member variables.
      **************************************************************/
    private void set
      (
       CLexGen lexGen,
       CSpec spec,
       CInput input
       )
        {
          if (CUtility.DEBUG)
            {
              CUtility.ASSERT(null != input);
              CUtility.ASSERT(null != lexGen);
              CUtility.ASSERT(null != spec);
            }

          m_input = input;
          m_lexGen = lexGen;
          m_spec = spec;
        }

    /***************************************************************
      Function: allocate_BOL_EOF
      Description: Expands character class to include special BOL and
      EOF characters.  Puts numeric index of these characters in
      input CSpec.
      **************************************************************/
    void allocate_BOL_EOF
      (
       CSpec spec
       )
          {
            CUtility.ASSERT(CSpec.NUM_PSEUDO==2);
            spec.BOL = spec.m_dtrans_ncols++;
            spec.EOF = spec.m_dtrans_ncols++;
          }

    /***************************************************************
      Function: thompson
      Description: High level access function to module.
      Deposits result in input CSpec.
      **************************************************************/
    void thompson
      (
       CLexGen lexGen,
       CSpec spec,
       CInput input
       )
        throws java.io.IOException
          {
            int i;
            CNfa elem;
            int size;

            /* Set member variables. */
            reset();
            set(lexGen,spec,input);

            size = m_spec.m_states.size();
            m_spec.m_state_rules = new Vector[size];
            for (i = 0; i < size; ++i)
              {
                m_spec.m_state_rules[i] = new Vector();
              }

            /* Initialize current token variable
               and create nfa. */
            /*m_spec.m_current_token = m_lexGen.EOS;
            m_lexGen.advance();*/

            m_spec.m_nfa_start = machine();

            /* Set labels in created nfa machine. */
            size = m_spec.m_nfa_states.size();
            for (i = 0; i < size; ++i)
              {
                elem = (CNfa) m_spec.m_nfa_states.elementAt(i);
                elem.m_label = i;
              }

            /* Debugging output. */
            if (CUtility.DO_DEBUG)
              {
                m_lexGen.print_nfa();
              }

            if (m_spec.m_verbose)
              {
                System.out.println("NFA comprised of "
                                   + (m_spec.m_nfa_states.size() + 1)
                                   + " states.");
              }

            reset();
          }

  /***************************************************************
    Function: discardCNfa
    Description:
    **************************************************************/
  private void discardCNfa
    (
     CNfa nfa
     )
      {
        m_spec.m_nfa_states.removeElement(nfa);
      }

  /***************************************************************
    Function: processStates
    Description:
    **************************************************************/
  private void processStates
    (
     SparseBitSet states,
     CNfa current
     )
      {
        int size;
        int i;

        size = m_spec.m_states.size();
      for (i = 0; i <  size; ++i)
        {
          if (states.get(i))
            {
              m_spec.m_state_rules[i].addElement(current);
            }
        }
    }

  /***************************************************************
    Function: machine
    Description: Recursive descent regular expression parser.
    **************************************************************/
  private CNfa machine
    (
     )
      throws java.io.IOException
      {
        CNfa start;
        CNfa p;
        SparseBitSet states;

        if (CUtility.DESCENT_DEBUG)
          {
            CUtility.enter("machine",m_spec.m_lexeme,m_spec.m_current_token);
          }

        start = CAlloc.newCNfa(m_spec);
        p = start;

        states = m_lexGen.getStates();

        /* Begin: Added for states. */
        m_spec.m_current_token = m_lexGen.EOS;
        m_lexGen.advance();
        /* End: Added for states. */

        if (m_lexGen.END_OF_INPUT != m_spec.m_current_token) // CSA fix.
          {
            p.m_next = rule();

            processStates(states,p.m_next);
          }

        while (m_lexGen.END_OF_INPUT != m_spec.m_current_token)
          {
            /* Make state changes HERE. */
            states = m_lexGen.getStates();

            /* Begin: Added for states. */
            m_lexGen.advance();
            if (m_lexGen.END_OF_INPUT == m_spec.m_current_token)
              {
                break;
              }
            /* End: Added for states. */

            p.m_next2 = CAlloc.newCNfa(m_spec);
            p = p.m_next2;
            p.m_next = rule();

            processStates(states,p.m_next);
          }

        // CSA: add pseudo-rules for BOL and EOF
        SparseBitSet all_states = new SparseBitSet();
        for (int i = 0; i < m_spec.m_states.size(); ++i)
                all_states.set(i);
        p.m_next2 = CAlloc.newCNfa(m_spec);
        p = p.m_next2;
        p.m_next = CAlloc.newCNfa(m_spec);
        p.m_next.m_edge = CNfa.CCL;
        p.m_next.m_next = CAlloc.newCNfa(m_spec);
        p.m_next.m_set = new CSet();
        p.m_next.m_set.add(m_spec.BOL);
```

```
	p.m_next.m_set.add(m_spec.EOF);
	p.m_next.m_next.m_accept = // do-nothing accept rule
	    new CAccept(new char[0], 0, m_input.m_line_number+1);
	processStates(all_states,p.m_next);
	// CSA: done.

	if (CUtility.DESCENT_DEBUG)
	  {
	    CUtility.leave("machine",m_spec.m_lexeme,m_spec.m_current_token);
	  }

	return start;
      }

  /***************************************************************
    Function: rule
    Description: Recursive descent regular expression parser.
    **************************************************************/
  private CNfa rule
    (
     )
      throws java.io.IOException
      {
	CNfaPair pair;
	CNfa p;
	CNfa start = null;
	CNfa end = null;
	int anchor = CSpec.NONE;

	if (CUtility.DESCENT_DEBUG)
	  {
	    CUtility.enter("rule",m_spec.m_lexeme,m_spec.m_current_token);
	  }

	pair = CAlloc.newCNfaPair();

	if (m_lexGen.AT_BOL == m_spec.m_current_token)
	  {
	    anchor = anchor | CSpec.START;
	    m_lexGen.advance();
	    expr(pair);

	    // CSA: fixed beginning-of-line operator. 8-aug-1999
	    start = CAlloc.newCNfa(m_spec);
	    start.m_edge = m_spec.BOL;
	    start.m_next = pair.m_start;
	    end = pair.m_end;
	  }
	else
	  {
	    expr(pair);
	    start = pair.m_start;
	    end = pair.m_end;
	  }

	if (m_lexGen.AT_EOL == m_spec.m_current_token)
	  {
	    m_lexGen.advance();
	    // CSA: fixed end-of-line operator. 8-aug-1999
	    CNfaPair nlpair = CAlloc.newNLPair(m_spec);
	    end.m_next = CAlloc.newCNfa(m_spec);
	    end.m_next.m_next = nlpair.m_start;
	    end.m_next.m_next2 = CAlloc.newCNfa(m_spec);
	    end.m_next.m_next2.m_edge = m_spec.EOF;
	    end.m_next.m_next2.m_next = nlpair.m_end;
	    end = nlpair.m_end;
	    anchor = anchor | CSpec.END;
	  }

	/* Check for null rules. Charles Fischer found this bug. [CSA] */
	if (end==null)
	    CError.parse_error(CError.E_ZERO, m_input.m_line_number);

	/* Handle end of regular expression.  See page 103. */
	end.m_accept = m_lexGen.packAccept();
	end.m_anchor = anchor;

	/* Begin: Removed for states. */
	/*m_lexGen.advance();*/
	/* End: Removed for states. */

	if (CUtility.DESCENT_DEBUG)
	  {
	    CUtility.leave("rule",m_spec.m_lexeme,m_spec.m_current_token);
	  }

	return start;
      }

  /***************************************************************
    Function: expr
    Description: Recursive descent regular expression parser.
    **************************************************************/
  private void expr
    (
     CNfaPair pair
     )
      throws java.io.IOException
      {
	CNfaPair e2_pair;
	CNfa p;

	if (CUtility.DESCENT_DEBUG)
	  {
	    CUtility.enter("expr",m_spec.m_lexeme,m_spec.m_current_token);
	  }

	if (CUtility.DEBUG)
	  {
	    CUtility.ASSERT(null != pair);
	  }

	e2_pair = CAlloc.newCNfaPair();

	cat_expr(pair);

	while (m_lexGen.OR == m_spec.m_current_token)
	  {
	    m_lexGen.advance();
	    cat_expr(e2_pair);

	    p = CAlloc.newCNfa(m_spec);
	    p.m_next2 = e2_pair.m_start;
	    p.m_next = pair.m_start;
	    pair.m_start = p;

	    p = CAlloc.newCNfa(m_spec);
	    pair.m_end.m_next = p;
	    e2_pair.m_end.m_next = p;
	    pair.m_end = p;
	  }

	if (CUtility.DESCENT_DEBUG)
	  {
	    CUtility.leave("expr",m_spec.m_lexeme,m_spec.m_current_token);
	  }
      }

  /***************************************************************
    Function: cat_expr
    Description: Recursive descent regular expression parser.
    **************************************************************/
  private void cat_expr
    (
     CNfaPair pair
     )
      throws java.io.IOException
      {
	CNfaPair e2_pair;
	if (CUtility.DESCENT_DEBUG)
	  {
	    CUtility.enter("cat_expr",m_spec.m_lexeme,m_spec.m_current_token);
	  }

	if (CUtility.DEBUG)
	  {
	    CUtility.ASSERT(null != pair);
	  }

	e2_pair = CAlloc.newCNfaPair();

	if (first_in_cat(m_spec.m_current_token))
	  {
	    factor(pair);
	  }

	while (first_in_cat(m_spec.m_current_token))
	  {
	    factor(e2_pair);

	    /* Destroy */
	    pair.m_end.mimic(e2_pair.m_start);
	    discardCNfa(e2_pair.m_start);

	    pair.m_end = e2_pair.m_end;
	  }

	if (CUtility.DESCENT_DEBUG)
	  {
	    CUtility.leave("cat_expr",m_spec.m_lexeme,m_spec.m_current_token);
	  }
      }

  /***************************************************************
    Function: first_in_cat
    Description: Recursive descent regular expression parser.
    **************************************************************/
  private boolean first_in_cat
    (
     int token
     )
      {
	switch (token)
	  {
	  case CLexGen.CLOSE_PAREN:
	  case CLexGen.AT_EOL:
	  case CLexGen.OR:
	  case CLexGen.EOS:
	    return false;

	  case CLexGen.CLOSURE:
	  case CLexGen.PLUS_CLOSE:
	  case CLexGen.OPTIONAL:
	    CError.parse_error(CError.E_CLOSE,m_input.m_line_number);
	    return false;

	  case CLexGen.CCL_END:
	    CError.parse_error(CError.E_BRACKET,m_input.m_line_number);
	    return false;

	  case CLexGen.AT_BOL:
	    CError.parse_error(CError.E_BOL,m_input.m_line_number);
	    return false;

	  default:
	    break;
	  }

	return true;
      }

  /***************************************************************
    Function: factor
    Description: Recursive descent regular expression parser.
    **********************************************************/
  private void factor
    (
     CNfaPair pair
     )
      throws java.io.IOException
      {
        CNfa start = null;
        CNfa end = null;

        if (CUtility.DESCENT_DEBUG)
          {
            CUtility.enter("factor",m_spec.m_lexeme,m_spec.m_current_token);
          }

        term(pair);

        if (m_lexGen.CLOSURE == m_spec.m_current_token
            || m_lexGen.PLUS_CLOSE == m_spec.m_current_token
            || m_lexGen.OPTIONAL == m_spec.m_current_token)
          {
            start = CAlloc.newCNfa(m_spec);
            end = CAlloc.newCNfa(m_spec);

            start.m_next = pair.m_start;
            pair.m_end.m_next = end;

            if (m_lexGen.CLOSURE == m_spec.m_current_token
                || m_lexGen.OPTIONAL == m_spec.m_current_token)
              {
                start.m_next2 = end;
              }

            if (m_lexGen.CLOSURE == m_spec.m_current_token
                || m_lexGen.PLUS_CLOSE == m_spec.m_current_token)
              {
                pair.m_end.m_next2 = pair.m_start;
              }

            pair.m_start = start;
            pair.m_end = end;
            m_lexGen.advance();
          }

        if (CUtility.DESCENT_DEBUG)
          {
            CUtility.leave("factor",m_spec.m_lexeme,m_spec.m_current_token);
          }
      }

  /***************************************************************
    Function: term
    Description: Recursive descent regular expression parser.
    **************************************************************/
  private void term
    (
     CNfaPair pair
     )
      throws java.io.IOException
      {
        CNfa start;
        boolean isAlphaL;
        int c;

        if (CUtility.DESCENT_DEBUG)
          {
            CUtility.enter("term",m_spec.m_lexeme,m_spec.m_current_token);
          }

        if (m_lexGen.OPEN_PAREN == m_spec.m_current_token)
          {
            m_lexGen.advance();
            expr(pair);

            if (m_lexGen.CLOSE_PAREN == m_spec.m_current_token)
              {
                m_lexGen.advance();
              }
            else
              {
                CError.parse_error(CError.E_SYNTAX,m_input.m_line_number);
              }
          }
        else
          {
            start = CAlloc.newCNfa(m_spec);
            pair.m_start = start;

            start.m_next = CAlloc.newCNfa(m_spec);
            pair.m_end = start.m_next;

            if (m_lexGen.L == m_spec.m_current_token &&
                Character.isLetter(m_spec.m_lexeme))
              {
                isAlphaL = true;
              }
            else
              {
                isAlphaL = false;
              }
            if (false == (m_lexGen.ANY == m_spec.m_current_token
                          || m_lexGen.CCL_START == m_spec.m_current_token
                          || (m_spec.m_ignorecase && isAlphaL)))
              {
                start.m_edge = m_spec.m_lexeme;
                m_lexGen.advance();
              }
            else
              {
                start.m_edge = CNfa.CCL;

                start.m_set = new CSet();

                /* Match case-insensitive letters using character class. */
                if (m_spec.m_ignorecase && isAlphaL)
                  {
                    start.m_set.addncase(m_spec.m_lexeme);
                  }
                /* Match dot (.) using character class. */
                else if (m_lexGen.ANY == m_spec.m_current_token)
                  {
                    start.m_set.add('\n');
                    start.m_set.add('\r');
                    // CSA: exclude BOL and EOF from character classes
                    start.m_set.add(m_spec.BOL);
                    start.m_set.add(m_spec.EOF);
                    start.m_set.complement();
                  }
                else
                  {
                    m_lexGen.advance();
                    if (m_lexGen.AT_BOL == m_spec.m_current_token)
                      {
                        m_lexGen.advance();

                        // CSA: exclude BOL and EOF from character classes
                        start.m_set.add(m_spec.BOL);
                        start.m_set.add(m_spec.EOF);
                        start.m_set.complement();
                      }
                    if (false == (m_lexGen.CCL_END == m_spec.m_current_token))
                      {
                        dodash(start.m_set);
                      }
                    /*else
                      {
                        for (c = 0; c <= ' '; ++c)
                          {
                            start.m_set.add((byte) c);
                          }
                      }*/
                  }
                m_lexGen.advance();
              }
          }

        if (CUtility.DESCENT_DEBUG)
          {
            CUtility.leave("term",m_spec.m_lexeme,m_spec.m_current_token);
          }
      }

  /***************************************************************
    Function: dodash
    Description: Recursive descent regular expression parser.
    **************************************************************/
  private void dodash
    (
     CSet set
     )
      throws java.io.IOException
        {
          int first = -1;

          if (CUtility.DESCENT_DEBUG)
            {
              CUtility.enter("dodash",m_spec.m_lexeme,m_spec.m_current_token);
            }

          while (m_lexGen.EOS != m_spec.m_current_token
                 && m_lexGen.CCL_END != m_spec.m_current_token)
            {
              // DASH loses its special meaning if it is first in class.
              if (m_lexGen.DASH == m_spec.m_current_token && -1 != first)
                {
                  m_lexGen.advance();
                  // DASH loses its special meaning if it is last in class.
                  if (m_spec.m_current_token == m_lexGen.CCL_END)
                    {
                      // 'first' already in set.
                      set.add('-');
                      break;
                    }
                  for ( ; first <= m_spec.m_lexeme; ++first)
                    {
                      if (m_spec.m_ignorecase)
                        set.addncase((char)first);
                      else
                        set.add(first);
                    }
                }
              else
                {
                  first = m_spec.m_lexeme;
                  if (m_spec.m_ignorecase)
                    set.addncase(m_spec.m_lexeme);
                  else
                    set.add(m_spec.m_lexeme);
                }

              m_lexGen.advance();
            }

        if (CUtility.DESCENT_DEBUG)
          {
            CUtility.leave("dodash",m_spec.m_lexeme,m_spec.m_current_token);
          }
      }
}

/**
 * Extract character classes from NFA and simplify.
 * @author C. Scott Ananian 25-Jul-1999
 */
class CSimplifyNfa
```

```
{
  private int[] ccls; // character class mapping.
  private int original_charset_size; // original charset size
  private int mapped_charset_size; // reduced charset size

  void simplify(CSpec m_spec) {
    computeClasses(m_spec); // initialize fields.

    // now rewrite the NFA using our character class mapping.
    for (Enumeration e=m_spec.m_nfa_states.elements(); e.hasMoreElements(); ) {
      CNfa nfa = (CNfa) e.nextElement();
      if (nfa.m_edge==CNfa.EMPTY || nfa.m_edge==CNfa.EPSILON)
        continue; // no change.
      if (nfa.m_edge==CNfa.CCL) {
        CSet ncset = new CSet();
        ncset.map(nfa.m_set, ccls); // map it.
        nfa.m_set = ncset;
      } else { // single character
        nfa.m_edge = ccls[nfa.m_edge]; // map it.
      }
    }

    // now update m_spec with the mapping.
    m_spec.m_ccls_map = ccls;
    m_spec.m_dtrans_ncols = mapped_charset_size;
  }
  /** Compute minimum set of character classes needed to disambiguate
   *  edges.  We optimistically assume that every character belongs to
   *  a single character class, and then incrementally split classes
   *  as we see edges that require discrimination between characters in
   *  the class. [CSA, 25-Jul-1999] */
  private void computeClasses(CSpec m_spec) {
    this.original_charset_size = m_spec.m_dtrans_ncols;
    this.ccls = new int[original_charset_size]; // initially all zero.

    int nextcls = 1;
    SparseBitSet clsA = new SparseBitSet(), clsB = new SparseBitSet();
    Hashtable h = new Hashtable();

    System.out.print("Working on character classes.");
    for (Enumeration e=m_spec.m_nfa_states.elements(); e.hasMoreElements(); ) {
      CNfa nfa = (CNfa) e.nextElement();
      if (nfa.m_edge==CNfa.EMPTY || nfa.m_edge==CNfa.EPSILON)
        continue; // no discriminatory information.
      clsA.clearAll(); clsB.clearAll();
      for (int i=0; i<ccls.length; i++)
        if (nfa.m_edge==i || // edge labeled with a character
            nfa.m_edge==CNfa.CCL && nfa.m_set.contains(i)) // set of characters
          clsA.set(ccls[i]);
        else
          clsB.set(ccls[i]);
      // now figure out which character classes we need to split.
      clsA.and(clsB); // split the classes which show up on both sides of edge
      System.out.print(clsA.size()==0?".":":");
      if (clsA.size()==0) continue; // nothing to do.
      // and split them.
      h.clear(); // h will map old to new class name
      for (int i=0; i<ccls.length; i++)
        if (clsA.get(ccls[i])) // a split class
          if (nfa.m_edge==i ||
              nfa.m_edge==CNfa.CCL && nfa.m_set.contains(i)) { // on A side
            Integer split = new Integer(ccls[i]);
            if (!h.containsKey(split))
              h.put(split, new Integer(nextcls++)); // make new class
            ccls[i] = ((Integer)h.get(split)).intValue();
          }
    }
    System.out.println();
    System.out.println("NFA has "+nextcls+" distinct character classes.");

    this.mapped_charset_size = nextcls;
  }
}

/***************************************************************
```

```
  Class: CMinimize
 **************************************************************/
class CMinimize
{
  /***************************************************************
    Member Variables
    **************************************************************/
  CSpec m_spec;
  Vector m_group;
  int m_ingroup[];

  /***************************************************************
    Function: CMinimize
    Description: Constructor.
    **************************************************************/
  CMinimize
    (
     )
      {
        reset();
      }

  /***************************************************************
    Function: reset
    Description: Resets member variables.
    **************************************************************/
  private void reset
    (
     )
      {
        m_spec = null;
        m_group = null;
        m_ingroup = null;
      }

  /***************************************************************
    Function: set
    Description: Sets member variables.
    **************************************************************/
  private void set
    (
     CSpec spec
     )
      {
        if (CUtility.DEBUG)
          {
            CUtility.ASSERT(null != spec);
          }

        m_spec = spec;
        m_group = null;
        m_ingroup = null;
      }

  /***************************************************************
    Function: min_dfa
    Description: High-level access function to module.
    **************************************************************/
  void min_dfa
    (
     CSpec spec
     )
      {
        set(spec);

        /* Remove redundant states. */
        minimize();

        /* Column and row compression.
           Save accept states in auxilary vector. */
        reduce();

        reset();
      }
```

```
/***************************************************************
  Function: col_copy
  Description: Copies source column into destination column.
  **************************************************************/
private void col_copy
  (
   int dest,
   int src
   )
    {
      int n;
      int i;
      CDTrans dtrans;

      n = m_spec.m_dtrans_vector.size();
      for (i = 0; i < n; ++i)
        {
          dtrans = (CDTrans) m_spec.m_dtrans_vector.elementAt(i);
          dtrans.m_dtrans[dest] = dtrans.m_dtrans[src];
        }
    }

/***************************************************************
  Function: trunc_col
  Description: Truncates each column to the 'correct' length.
  **************************************************************/
private void trunc_col
  (
   )
    {
      int n;
      int i;
      CDTrans dtrans;

      n = m_spec.m_dtrans_vector.size();
      for (i = 0; i < n; ++i)
        {
          int[] ndtrans = new int[m_spec.m_dtrans_ncols];
          dtrans = (CDTrans) m_spec.m_dtrans_vector.elementAt(i);
          System.arraycopy(dtrans.m_dtrans, 0, ndtrans, 0, ndtrans.length);
          dtrans.m_dtrans = ndtrans;
        }
    }
/***************************************************************
  Function: row_copy
  Description: Copies source row into destination row.
  **************************************************************/
private void row_copy
  (
   int dest,
   int src
   )
    {
      CDTrans dtrans;

      dtrans = (CDTrans) m_spec.m_dtrans_vector.elementAt(src);
      m_spec.m_dtrans_vector.setElementAt(dtrans,dest);
    }

/***************************************************************
  Function: col_equiv
  Description:
  **************************************************************/
private boolean col_equiv
  (
   int col1,
   int col2
   )
    {
      int n;
      int i;
      CDTrans dtrans;

      n = m_spec.m_dtrans_vector.size();
      for (i = 0; i < n; ++i)
```

```
        {
          dtrans = (CDTrans) m_spec.m_dtrans_vector.elementAt(i);
          if (dtrans.m_dtrans[col1] != dtrans.m_dtrans[col2])
            {
              return false;
            }
        }

      return true;
    }

  /***************************************************************
    Function: row_equiv
    Description:
    **************************************************************/
  private boolean row_equiv
    (
     int row1,
     int row2
     )
      {
        int i;
        CDTrans dtrans1;
        CDTrans dtrans2;

        dtrans1 = (CDTrans) m_spec.m_dtrans_vector.elementAt(row1);
        dtrans2 = (CDTrans) m_spec.m_dtrans_vector.elementAt(row2);

        for (i = 0; i < m_spec.m_dtrans_ncols; ++i)
          {
            if (dtrans1.m_dtrans[i] != dtrans2.m_dtrans[i])
              {
                return false;
              }
          }

        return true;
      }

  /***************************************************************
    Function: reduce
    Description:
    **************************************************************/
  private void reduce
    (
     )
      {
        int i;
        int j;
        int k;
        int nrows;
        int reduced_ncols;
        int reduced_nrows;
        SparseBitSet set;
        CDTrans dtrans;
        int size;

        set = new SparseBitSet();

        /* Save accept nodes and anchor entries. */
        size = m_spec.m_dtrans_vector.size();
        m_spec.m_anchor_array = new int[size];
        m_spec.m_accept_vector = new Vector();
        for (i = 0; i < size; ++i)
          {
            dtrans = (CDTrans) m_spec.m_dtrans_vector.elementAt(i);
            m_spec.m_accept_vector.addElement(dtrans.m_accept);
            m_spec.m_anchor_array[i] = dtrans.m_anchor;
            dtrans.m_accept = null;
          }

        /* Allocate column map. */
        m_spec.m_col_map = new int[m_spec.m_dtrans_ncols];
        for (i = 0; i < m_spec.m_dtrans_ncols; ++i)
          {
```

```
	    m_spec.m_col_map[i] = -1;
	  }

	/* Process columns for reduction. */
	for (reduced_ncols = 0; ; ++reduced_ncols)
	  {
	    if (CUtility.DEBUG)
	      {
	        for (i = 0; i < reduced_ncols; ++i)
		  {
		    CUtility.ASSERT(-1 != m_spec.m_col_map[i]);
		  }
	      }

	    for (i = reduced_ncols; i < m_spec.m_dtrans_ncols; ++i)
	      {
		if (-1 == m_spec.m_col_map[i])
		  {
		    break;
		  }
	      }

	    if (i >= m_spec.m_dtrans_ncols)
	      {
		break;
	      }

	    if (CUtility.DEBUG)
	      {
		CUtility.ASSERT(false == set.get(i));
		CUtility.ASSERT(-1 == m_spec.m_col_map[i]);
	      }

	    set.set(i);

	    m_spec.m_col_map[i] = reduced_ncols;

	    /* UNDONE: Optimize by doing all comparisons in one batch. */
	    for (j = i + 1; j < m_spec.m_dtrans_ncols; ++j)
	      {
		if (-1 == m_spec.m_col_map[j] && true == col_equiv(i,j))
		  {
		    m_spec.m_col_map[j] = reduced_ncols;
		  }
	      }
	  }

	/* Reduce columns. */
	k = 0;
	for (i = 0; i < m_spec.m_dtrans_ncols; ++i)
	  {
	    if (set.get(i))
	      {
		++k;

		set.clear(i);

		j = m_spec.m_col_map[i];

		if (CUtility.DEBUG)
		  {
		    CUtility.ASSERT(j <= i);
		  }

		if (j == i)
		  {
		    continue;
		  }

		col_copy(j,i);
	      }
	  }
	m_spec.m_dtrans_ncols = reduced_ncols;
	/* truncate m_dtrans at proper length (freeing extra) */
	trunc_col();
```

```
        if (CUtility.DEBUG)
          {
            CUtility.ASSERT(k == reduced_ncols);
          }

        /* Allocate row map. */
        nrows = m_spec.m_dtrans_vector.size();
        m_spec.m_row_map = new int[nrows];
        for (i = 0; i < nrows; ++i)
          {
            m_spec.m_row_map[i] = -1;
          }

        /* Process rows to reduce. */
        for (reduced_nrows = 0; ; ++reduced_nrows)
          {
            if (CUtility.DEBUG)
              {
                for (i = 0; i < reduced_nrows; ++i)
                  {
                    CUtility.ASSERT(-1 != m_spec.m_row_map[i]);
                  }
              }

            for (i = reduced_nrows; i < nrows; ++i)
              {
                if (-1 == m_spec.m_row_map[i])
                  {
                    break;
                  }
              }

            if (i >= nrows)
              {
                break;
              }

            if (CUtility.DEBUG)
              {
                CUtility.ASSERT(false == set.get(i));
                CUtility.ASSERT(-1 == m_spec.m_row_map[i]);
              }

            set.set(i);

            m_spec.m_row_map[i] = reduced_nrows;

            /* UNDONE: Optimize by doing all comparisons in one batch. */
            for (j = i + 1; j < nrows; ++j)
              {
                if (-1 == m_spec.m_row_map[j] && true == row_equiv(i,j))
                  {
                    m_spec.m_row_map[j] = reduced_nrows;
                  }
              }
          }

        /* Reduce rows. */
        k = 0;
        for (i = 0; i < nrows; ++i)
          {
            if (set.get(i))
              {
                ++k;

                set.clear(i);

                j = m_spec.m_row_map[i];

                if (CUtility.DEBUG)
                  {
                    CUtility.ASSERT(j <= i);
                  }
                if (j == i)
                  {
                    continue;
                  }

                row_copy(j,i);
              }
          }
        m_spec.m_dtrans_vector.setSize(reduced_nrows);

        if (CUtility.DEBUG)
          {
            /*System.out.println("k = " + k + "\nreduced_nrows = " + reduced_nrows + "");*/
            CUtility.ASSERT(k == reduced_nrows);
          }
      }

  /***************************************************************
    Function: fix_dtrans
    Description: Updates CDTrans table after minimization
    using groups, removing redundant transition table states.
    **************************************************************/
  private void fix_dtrans
    (
     )
      {
        Vector new_vector;
        int i;
        int size;
        Vector dtrans_group;
        CDTrans first;
        int c;

        new_vector = new Vector();

        size = m_spec.m_state_dtrans.length;
        for (i = 0; i < size; ++i)
          {
            if (CDTrans.F != m_spec.m_state_dtrans[i])
              {
                m_spec.m_state_dtrans[i] = m_ingroup[m_spec.m_state_dtrans[i]];
              }
          }

        size = m_group.size();
        for (i = 0; i < size; ++i)
          {
            dtrans_group = (Vector) m_group.elementAt(i);
            first = (CDTrans) dtrans_group.elementAt(0);
            new_vector.addElement(first);

            for (c = 0; c < m_spec.m_dtrans_ncols; ++c)
              {
                if (CDTrans.F != first.m_dtrans[c])
                  {
                    first.m_dtrans[c] = m_ingroup[first.m_dtrans[c]];
                  }
              }
          }

        m_group = null;
        m_spec.m_dtrans_vector = new_vector;
      }

  /***************************************************************
    Function: minimize
    Description: Removes redundant transition table states.
    **************************************************************/
  private void minimize
    (
     )
      {
        Vector dtrans_group;
        Vector new_group;
        int i;
        int j;
        int old_group_count;
        int group_count;
        CDTrans next;
        CDTrans first;
        int goto_first;
        int goto_next;
        int c;
        int group_size;
        boolean added;

        init_groups();

        group_count = m_group.size();
        old_group_count = group_count - 1;

        while (old_group_count != group_count)
          {
            old_group_count = group_count;

            if (CUtility.DEBUG)
              {
                CUtility.ASSERT(m_group.size() == group_count);
              }

            for (i = 0; i < group_count; ++i)
              {
                dtrans_group = (Vector) m_group.elementAt(i);

                group_size = dtrans_group.size();
                if (group_size <= 1)
                  {
                    continue;
                  }

                new_group = new Vector();
                added = false;

                first = (CDTrans) dtrans_group.elementAt(0);
                for (j = 1; j < group_size; ++j)
                  {
                    next = (CDTrans) dtrans_group.elementAt(j);

                    for (c = 0; c < m_spec.m_dtrans_ncols; ++c)
                      {
                        goto_first = first.m_dtrans[c];
                        goto_next = next.m_dtrans[c];

                        if (goto_first != goto_next
                            && (goto_first == CDTrans.F
                                || goto_next == CDTrans.F
                                || m_ingroup[goto_next] != m_ingroup[goto_first]))
                          {
                            if (CUtility.DEBUG)
                              {
                                CUtility.ASSERT(dtrans_group.elementAt(j) == next);
                              }

                            dtrans_group.removeElementAt(j);
                            --j;
                            --group_size;
                            new_group.addElement(next);
                            if (false == added)
                              {
                                added = true;
                                ++group_count;
                                m_group.addElement(new_group);
                              }
                            m_ingroup[next.m_label] = m_group.size() - 1;

                            if (CUtility.DEBUG)
                              {
                                CUtility.ASSERT(m_group.contains(new_group)
                                                == true);
                                CUtility.ASSERT(m_group.contains(dtrans_group)
                                                    == true);
                                CUtility.ASSERT(dtrans_group.contains(first)
                                                    == true);
                                CUtility.ASSERT(dtrans_group.contains(next)
                                                    == false);
                                CUtility.ASSERT(new_group.contains(first)
                                                    == false);
                                CUtility.ASSERT(new_group.contains(next)
                                                    == true);
                                CUtility.ASSERT(dtrans_group.size() == group_size);
                                CUtility.ASSERT(i == m_ingroup[first.m_label]);
                                CUtility.ASSERT((m_group.size() - 1)
                                                    == m_ingroup[next.m_label]);
                              }

                            break;
                          }
                      }
                  }
              }
          }

        System.out.println(m_group.size() + " states after removal of redundant states.");

        if (m_spec.m_verbose
            && true == CUtility.OLD_DUMP_DEBUG)
          {
            System.out.println();
            System.out.println("States grouped as follows after minimization");
            pgroups();
          }

        fix_dtrans();
      }

  /***************************************************************
    Function: init_groups
    Description:
    **************************************************************/
  private void init_groups
    (
     )
      {
        int i;
        int j;
        int group_count;
        int size;
        CAccept accept;
        CDTrans dtrans;
        Vector dtrans_group;
        CDTrans first;
        boolean group_found;

        m_group = new Vector();
        group_count = 0;

        size = m_spec.m_dtrans_vector.size();
        m_ingroup = new int[size];

        for (i = 0; i < size; ++i)
          {
            group_found = false;
            dtrans = (CDTrans) m_spec.m_dtrans_vector.elementAt(i);

            if (CUtility.DEBUG)
              {
                CUtility.ASSERT(i == dtrans.m_label);
                CUtility.ASSERT(false == group_found);
                CUtility.ASSERT(group_count == m_group.size());
              }

            for (j = 0; j < group_count; ++j)
              {
                dtrans_group = (Vector) m_group.elementAt(j);
                if (CUtility.DEBUG)
                  {
                    CUtility.ASSERT(false == group_found);
                    CUtility.ASSERT(0 < dtrans_group.size());
                  }

                first = (CDTrans) dtrans_group.elementAt(0);

                if (CUtility.SLOW_DEBUG)
                  {
                    CDTrans check;
                    int k;
                    int s;

                    s = dtrans_group.size();
                    CUtility.ASSERT(0 < s);

                    for (k = 1; k < s; ++k)
                      {
                        check = (CDTrans) dtrans_group.elementAt(k);
                        CUtility.ASSERT(check.m_accept == first.m_accept);
                      }
                  }

                if (first.m_accept == dtrans.m_accept)
                  {
                    dtrans_group.addElement(dtrans);
                    m_ingroup[i] = j;
                    group_found = true;

                    if (CUtility.DEBUG)
                      {
                        CUtility.ASSERT(j == m_ingroup[dtrans.m_label]);
                      }

                    break;
                  }
              }

            if (false == group_found)
              {
                dtrans_group = new Vector();
                dtrans_group.addElement(dtrans);
                m_ingroup[i] = m_group.size();
                m_group.addElement(dtrans_group);
                ++group_count;
              }
          }

        if (m_spec.m_verbose
            && true == CUtility.OLD_DUMP_DEBUG)
          {
            System.out.println("Initial grouping:");
            pgroups();
            System.out.println();
          }
      }

  /***************************************************************
    Function: pset
    **************************************************************/
  private void pset
    (
     Vector dtrans_group
     )
      {
        int i;
        int size;
        CDTrans dtrans;

        size = dtrans_group.size();
        for (i = 0; i < size; ++i)
          {
            dtrans = (CDTrans) dtrans_group.elementAt(i);
            System.out.print(dtrans.m_label + " ");
```

```java
          }
      }

    /***************************************************************
      Function: pgroups
      **************************************************************/
    private void pgroups
      (
       )
        {
          int i;
          int dtrans_size;
          int group_size;

          group_size = m_group.size();
          for (i = 0; i < group_size; ++i)
            {
              System.out.print("\tGroup " + i + " {");
              pset((Vector) m_group.elementAt(i));
              System.out.println("}");
              System.out.println();
            }

          System.out.println();
          dtrans_size = m_spec.m_dtrans_vector.size();
          for (i = 0; i < dtrans_size; ++i)
            {
              System.out.println("\tstate " + i
                                 + " is in group "
                                 + m_ingroup[i]);
            }
        }
}

/***************************************************************
  Class: CNfa2Dfa
 **************************************************************/
class CNfa2Dfa
{
  /***************************************************************
    Member Variables
    **************************************************************/
  private CSpec m_spec;
  private int m_unmarked_dfa;
  private CLexGen m_lexGen;

  /***************************************************************
    Constants
    **************************************************************/
  private static final int NOT_IN_DSTATES = -1;

  /***************************************************************
    Function: CNfa2Dfa
    **************************************************************/
  CNfa2Dfa
    (
     )
      {
        reset();
      }

  /***************************************************************
    Function: set
    Description:
    **************************************************************/
  private void set
    (
     CLexGen lexGen,
     CSpec spec
     )
      {
        m_lexGen = lexGen;
        m_spec = spec;
        m_unmarked_dfa = 0;
      }
```

```
/***************************************************************
  Function: reset
  Description:
  **************************************************************/
private void reset
  (
   )
    {
      m_lexGen = null;
      m_spec = null;
      m_unmarked_dfa = 0;
    }

/***************************************************************
  Function: make_dfa
  Description: High-level access function to module.
  **************************************************************/
void make_dfa
  (
   CLexGen lexGen,
   CSpec spec
   )
    {
      int i;

      reset();
      set(lexGen,spec);

      make_dtrans();
      free_nfa_states();

      if (m_spec.m_verbose && true == CUtility.OLD_DUMP_DEBUG)
        {
          System.out.println(m_spec.m_dfa_states.size()
                             + " DFA states in original machine.");
        }

      free_dfa_states();
    }

 /***************************************************************
  Function: make_dtrans
  Description: Creates uncompressed CDTrans transition table.
  **************************************************************/
private void make_dtrans
  (
   )
   /* throws java.lang.CloneNotSupportedException*/
    {
      CDfa next;
      CDfa dfa;
      CBunch bunch;
      int i;
      int nextstate;
      int size;
      CDTrans dtrans;
      CNfa nfa;
      int istate;
      int nstates;

      System.out.print("Working on DFA states.");

      /* Reference passing type and initializations. */
      bunch = new CBunch();
      m_unmarked_dfa = 0;

      /* Allocate mapping array. */
      nstates = m_spec.m_state_rules.length;
      m_spec.m_state_dtrans = new int[nstates];

      for (istate = 0; nstates > istate; ++istate)
        {
          /* CSA bugfix: if we skip all zero size rules, then
             an specification with no rules produces an illegal
             lexer (0 states) instead of a lexer that rejects
             everything (1 nonaccepting state). [27-Jul-1999]
          if (0 == m_spec.m_state_rules[istate].size())
            {
              m_spec.m_state_dtrans[istate] = CDTrans.F;
              continue;
            }
          */

          /* Create start state and initialize fields. */
          bunch.m_nfa_set = (Vector) m_spec.m_state_rules[istate].clone();
          sortStates(bunch.m_nfa_set);

          bunch.m_nfa_bit = new SparseBitSet();

          /* Initialize bit set. */
          size = bunch.m_nfa_set.size();
          for (i = 0; size > i; ++i)
            {
              nfa = (CNfa) bunch.m_nfa_set.elementAt(i);
              bunch.m_nfa_bit.set(nfa.m_label);
            }

          bunch.m_accept = null;
          bunch.m_anchor = CSpec.NONE;
          bunch.m_accept_index = CUtility.INT_MAX;

          e_closure(bunch);
          add_to_dstates(bunch);

          m_spec.m_state_dtrans[istate] = m_spec.m_dtrans_vector.size();

          /* Main loop of CDTrans creation. */
          while (null != (dfa = get_unmarked()))
            {
              System.out.print(".");
              System.out.flush();

              if (CUtility.DEBUG)
                {
                  CUtility.ASSERT(false == dfa.m_mark);
                }

              /* Get first unmarked node, then mark it. */
              dfa.m_mark = true;

              /* Allocate new CDTrans, then initialize fields. */
              dtrans = new CDTrans(m_spec.m_dtrans_vector.size(),m_spec);
              dtrans.m_accept = dfa.m_accept;
              dtrans.m_anchor = dfa.m_anchor;

              /* Set CDTrans array for each character transition. */
              for (i = 0; i < m_spec.m_dtrans_ncols; ++i)
                {
                  if (CUtility.DEBUG)
                    {
                      CUtility.ASSERT(0 <= i);
                      CUtility.ASSERT(m_spec.m_dtrans_ncols > i);
                    }

                  /* Create new dfa set by attempting character transition. */
                  move(dfa.m_nfa_set,dfa.m_nfa_bit,i,bunch);
                  if (null != bunch.m_nfa_set)
                    {
                      e_closure(bunch);
                    }

                  if (CUtility.DEBUG)
                    {
                      CUtility.ASSERT((null == bunch.m_nfa_set
                                       && null == bunch.m_nfa_bit)
                                      || (null != bunch.m_nfa_set
                                          && null != bunch.m_nfa_bit));
                    }
```

```
                  /* Create new state or set state to empty. */
                  if (null == bunch.m_nfa_set)
                    {
                      nextstate = CDTrans.F;
                    }
                  else
                    {
                      nextstate = in_dstates(bunch);

                      if (NOT_IN_DSTATES == nextstate)
                        {
                          nextstate = add_to_dstates(bunch);
                        }
                    }

                  if (CUtility.DEBUG)
                    {
                      CUtility.ASSERT(nextstate < m_spec.m_dfa_states.size());
                    }

                  dtrans.m_dtrans[i] = nextstate;
                }

              if (CUtility.DEBUG)
                {
                  CUtility.ASSERT(m_spec.m_dtrans_vector.size() == dfa.m_label);
                }

              m_spec.m_dtrans_vector.addElement(dtrans);
            }
        }

      System.out.println();
    }

  /***************************************************************
    Function: free_dfa_states
    **************************************************************/
  private void free_dfa_states
    (
     )
      {
        m_spec.m_dfa_states = null;
        m_spec.m_dfa_sets = null;
      }

  /***************************************************************
    Function: free_nfa_states
    **************************************************************/
  private void free_nfa_states
    (
     )
      {
        /* UNDONE: Remove references to nfas from within dfas. */
        /* UNDONE: Don't free CAccepts. */

        m_spec.m_nfa_states = null;
        m_spec.m_nfa_start = null;
        m_spec.m_state_rules = null;
      }

  /***************************************************************
    Function: e_closure
    Description: Alters and returns input set.
    **************************************************************/
  private void e_closure
    (
     CBunch bunch
     )
      {
        Stack nfa_stack;
        int size;
        int i;
        CNfa state;
        /* Debug checks. */
        if (CUtility.DEBUG)
          {
            CUtility.ASSERT(null != bunch);
            CUtility.ASSERT(null != bunch.m_nfa_set);
            CUtility.ASSERT(null != bunch.m_nfa_bit);
          }

        bunch.m_accept = null;
        bunch.m_anchor = CSpec.NONE;
        bunch.m_accept_index = CUtility.INT_MAX;

        /* Create initial stack. */
        nfa_stack = new Stack();
        size = bunch.m_nfa_set.size();
        for (i = 0; i < size; ++i)
          {
            state = (CNfa) bunch.m_nfa_set.elementAt(i);

            if (CUtility.DEBUG)
              {
                CUtility.ASSERT(bunch.m_nfa_bit.get(state.m_label));
              }

            nfa_stack.push(state);
          }

        /* Main loop. */
        while (false == nfa_stack.empty())
          {
            state = (CNfa) nfa_stack.pop();

            if (CUtility.OLD_DUMP_DEBUG)
              {
                if (null != state.m_accept)
                  {
                    System.out.println("Looking at accepting state " + state.m_label
                                       + " with <"
                                       + (new String(state.m_accept.m_action,0,
                                                     state.m_accept.m_action_read))
                                       + ">");
                  }
              }

            if (null != state.m_accept
                && state.m_label < bunch.m_accept_index)
              {
                bunch.m_accept_index = state.m_label;
                bunch.m_accept = state.m_accept;
                bunch.m_anchor = state.m_anchor;

                if (CUtility.OLD_DUMP_DEBUG)
                  {
                    System.out.println("Found accepting state " + state.m_label
                                       + " with <"
                                       + (new String(state.m_accept.m_action,0,
                                                     state.m_accept.m_action_read))
                                       + ">");
                  }

                if (CUtility.DEBUG)
                  {
                    CUtility.ASSERT(null != bunch.m_accept);
                    CUtility.ASSERT(CSpec.NONE == bunch.m_anchor
                                    || 0 != (bunch.m_anchor & CSpec.END)
                                    || 0 != (bunch.m_anchor & CSpec.START));
                  }
              }

            if (CNfa.EPSILON == state.m_edge)
              {
                if (null != state.m_next)
                  {
                    if (false == bunch.m_nfa_set.contains(state.m_next))
                      {
                        if (CUtility.DEBUG)
                          {
                            CUtility.ASSERT(false == bunch.m_nfa_bit.get(state.m_next.m_label
                          }

                        bunch.m_nfa_bit.set(state.m_next.m_label);
                        bunch.m_nfa_set.addElement(state.m_next);
                        nfa_stack.push(state.m_next);
                      }
                  }

                if (null != state.m_next2)
                  {
                    if (false == bunch.m_nfa_set.contains(state.m_next2))
                      {
                        if (CUtility.DEBUG)
                          {
                            CUtility.ASSERT(false == bunch.m_nfa_bit.get(state.m_next2.m_labe
                          }

                        bunch.m_nfa_bit.set(state.m_next2.m_label);
                        bunch.m_nfa_set.addElement(state.m_next2);
                        nfa_stack.push(state.m_next2);
                      }
                  }
              }
          }

        if (null != bunch.m_nfa_set)
          {
            sortStates(bunch.m_nfa_set);
          }

        return;
      }

  /***************************************************************
    Function: move
    Description: Returns null if resulting NFA set is empty.
    **************************************************************/
  void move
    (
     Vector nfa_set,
     SparseBitSet nfa_bit,
     int b,
     CBunch bunch
     )
      {
        int size;
        int index;
        CNfa state;

        bunch.m_nfa_set = null;
        bunch.m_nfa_bit = null;

        size = nfa_set.size();
        for (index = 0; index < size; ++index)
          {
            state = (CNfa) nfa_set.elementAt(index);

            if (b == state.m_edge
                || (CNfa.CCL == state.m_edge
                    && true == state.m_set.contains(b)))
              {
                if (null == bunch.m_nfa_set)
                  {
                    if (CUtility.DEBUG)
                      {
                        CUtility.ASSERT(null == bunch.m_nfa_bit);
                      }

                    bunch.m_nfa_set = new Vector();
                    /*bunch.m_nfa_bit
                        = new SparseBitSet(m_spec.m_nfa_states.size());*/
                    bunch.m_nfa_bit = new SparseBitSet();
                  }

                bunch.m_nfa_set.addElement(state.m_next);
                /*System.out.println("Size of bitset: " + bunch.m_nfa_bit.size());
                System.out.println("Reference index: " + state.m_next.m_label);
                System.out.flush();*/
                bunch.m_nfa_bit.set(state.m_next.m_label);
              }
          }

        if (null != bunch.m_nfa_set)
          {
            if (CUtility.DEBUG)
              {
                CUtility.ASSERT(null != bunch.m_nfa_bit);
              }

            sortStates(bunch.m_nfa_set);
          }

        return;
      }

  /***************************************************************
    Function: sortStates
    ***************************************************************/
  private void sortStates
    (
     Vector nfa_set
     )
      {
        CNfa elem;
        int begin;
        int size;
        int index;
        int value;
        int smallest_index;
        int smallest_value;
        CNfa begin_elem;

        size = nfa_set.size();
        for (begin = 0; begin < size; ++begin)
          {
            elem = (CNfa) nfa_set.elementAt(begin);
            smallest_value = elem.m_label;
            smallest_index = begin;

            for (index = begin + 1; index < size; ++index)
              {
                elem = (CNfa) nfa_set.elementAt(index);
                value = elem.m_label;

                if (value < smallest_value)
                  {
                    smallest_index = index;
                    smallest_value = value;
                  }
              }

            begin_elem = (CNfa) nfa_set.elementAt(begin);
            elem = (CNfa) nfa_set.elementAt(smallest_index);
            nfa_set.setElementAt(elem,begin);
            nfa_set.setElementAt(begin_elem,smallest_index);
          }

        if (CUtility.OLD_DEBUG)
          {
            System.out.print("NFA vector indices: ");

            for (index = 0; index < size; ++index)
              {
                elem = (CNfa) nfa_set.elementAt(index);
                System.out.print(elem.m_label + " ");
              }
            System.out.println();
        }

      return;
    }

  /***************************************************************
    Function: get_unmarked
    Description: Returns next unmarked DFA state.
    **************************************************************/
  private CDfa get_unmarked
    (
     )
      {
      int size;
      CDfa dfa;

      size = m_spec.m_dfa_states.size();
      while (m_unmarked_dfa < size)
        {
          dfa = (CDfa) m_spec.m_dfa_states.elementAt(m_unmarked_dfa);

          if (false == dfa.m_mark)
            {
              if (CUtility.OLD_DUMP_DEBUG)
                {
                  System.out.print("*");
                  System.out.flush();
                }

              if (m_spec.m_verbose && true == CUtility.OLD_DUMP_DEBUG)
                {
                  System.out.println("---------------");
                  System.out.print("working on DFA state "
                                   + m_unmarked_dfa
                                   + " = NFA states: ");
                  m_lexGen.print_set(dfa.m_nfa_set);
                  System.out.println();
                }

              return dfa;
            }

          ++m_unmarked_dfa;
        }

      return null;
    }

  /***************************************************************
    function: add_to_dstates
    Description: Takes as input a CBunch with details of
    a dfa state that needs to be created.
    1) Allocates a new dfa state and saves it in
    the appropriate CSpec vector.
    2) Initializes the fields of the dfa state
    with the information in the CBunch.
    3) Returns index of new dfa.
    **************************************************************/
  private int add_to_dstates
    (
     CBunch bunch
     )
      {
      CDfa dfa;

      if (CUtility.DEBUG)
        {
          CUtility.ASSERT(null != bunch.m_nfa_set);
          CUtility.ASSERT(null != bunch.m_nfa_bit);
          CUtility.ASSERT(null != bunch.m_accept
                          || CSpec.NONE == bunch.m_anchor);
        }

      /* Allocate, passing CSpec so dfa label can be set. */
      dfa = CAlloc.newCDfa(m_spec);
```

```
        /* Initialize fields, including the mark field. */
        dfa.m_nfa_set = (Vector) bunch.m_nfa_set.clone();
        dfa.m_nfa_bit = (SparseBitSet) bunch.m_nfa_bit.clone();
        dfa.m_accept = bunch.m_accept;
        dfa.m_anchor = bunch.m_anchor;
        dfa.m_mark = false;

        /* Register dfa state using BitSet in CSpec Hashtable. */
        m_spec.m_dfa_sets.put(dfa.m_nfa_bit,dfa);
        /*registerCDfa(dfa);*/

        if (CUtility.OLD_DUMP_DEBUG)
          {
            System.out.print("Registering set : ");
            m_lexGen.print_set(dfa.m_nfa_set);
            System.out.println();
          }

        return dfa.m_label;
      }

  /***************************************************************
    Function: in_dstates
    **************************************************************/
  private int in_dstates
    (
     CBunch bunch
     )
      {
        CDfa dfa;

        if (CUtility.OLD_DEBUG)
          {
            System.out.print("Looking for set : ");
            m_lexGen.print_set(bunch.m_nfa_set);
          }

        dfa = (CDfa) m_spec.m_dfa_sets.get(bunch.m_nfa_bit);

        if (null != dfa)
          {
            if (CUtility.OLD_DUMP_DEBUG)
              {
                System.out.println(" FOUND!");
              }

            return dfa.m_label;
          }

        if (CUtility.OLD_DUMP_DEBUG)
          {
            System.out.println(" NOT FOUND!");
          }
        return NOT_IN_DSTATES;
      }

}

/***************************************************************
  Class: CAlloc
  **************************************************************/
class CAlloc
{
  /***************************************************************
    Function: newCDfa
    **************************************************************/
  static CDfa newCDfa
    (
     CSpec spec
     )
      {
        CDfa dfa;

        dfa = new CDfa(spec.m_dfa_states.size());
        spec.m_dfa_states.addElement(dfa);

        return dfa;
      }

  /***************************************************************
    Function: newCNfaPair
    Description:
    **************************************************************/
  static CNfaPair newCNfaPair
    (
     )
      {
        CNfaPair pair = new CNfaPair();

        return pair;
      }
  /***************************************************************
    Function: newNLPair
    Description: return a new CNfaPair that matches a new
                 line: (\r\n?|[\n\uu2028\uu2029])
                 Added by CSA 8-Aug-1999, updated 10-Aug-1999
    **************************************************************/
  static CNfaPair newNLPair(CSpec spec) {
    CNfaPair pair = newCNfaPair();
    pair.m_end=newCNfa(spec); // newline accepting state
    pair.m_start=newCNfa(spec); // new state with two epsilon edges
    pair.m_start.m_next = newCNfa(spec);
    pair.m_start.m_next.m_edge = CNfa.CCL;
    pair.m_start.m_next.m_set = new CSet();
    pair.m_start.m_next.m_set.add('\n');
    if (spec.m_dtrans_ncols-CSpec.NUM_PSEUDO > 2029) {
      pair.m_start.m_next.m_set.add(2028); /*U+2028 is LS, the line separator*/
      pair.m_start.m_next.m_set.add(2029); /*U+2029 is PS, the paragraph sep.*/
    }
    pair.m_start.m_next.m_next = pair.m_end; // accept '\n', U+2028, or U+2029
    pair.m_start.m_next2 = newCNfa(spec);
    pair.m_start.m_next2.m_edge = '\r';
    pair.m_start.m_next2.m_next = newCNfa(spec);
    pair.m_start.m_next2.m_next.m_next = pair.m_end; // accept '\r';
    pair.m_start.m_next2.m_next.m_next2 = newCNfa(spec);
    pair.m_start.m_next2.m_next.m_next2.m_edge = '\n';
    pair.m_start.m_next2.m_next.m_next2.m_next = pair.m_end; // accept '\r\n';
    return pair;
  }

  /***************************************************************
    Function: newCNfa
    Description:
    **************************************************************/
  static CNfa newCNfa
    (
     CSpec spec
     )
      {
        CNfa p;

        /* UNDONE: Buffer this? */

        p = new CNfa();

        /*p.m_label = spec.m_nfa_states.size();*/
        spec.m_nfa_states.addElement(p);
        p.m_edge = CNfa.EPSILON;

        return p;
      }
}

/***************************************************************
  Class: Main
  Description: Top-level lexical analyzer generator function.
 **************************************************************/
public class Main
{
  /***************************************************************
    Function: main
    **************************************************************/
  public static void main
    (
     String arg[]
     )
    throws java.io.IOException
      {
        CLexGen lg;

        if (arg.length < 1)
          {
            System.out.println("Usage: JLex.Main <filename>");
            return;
          }

        /* Note: For debuging, it may be helpful to remove the try/catch
           block and permit the Exception to propagate to the top level.
           This gives more information. */
        try
          {
            lg = new CLexGen(arg[0]);
            lg.generate();
          }
        catch (Error e)
          {
            System.out.println(e.getMessage());
          }
      }
}

/***************************************************************
  Class: CDTrans
  **************************************************************/
class CDTrans
{
  /*************************************************************
    Member Variables
    ***********************************************************/
  int m_dtrans[];
  CAccept m_accept;
  int m_anchor;
  int m_label;

  /*************************************************************
    Constants
    ***********************************************************/
  static final int F = -1;

  /*************************************************************
    Function: CTrans
    ***********************************************************/
  CDTrans
    (
     int label,
     CSpec spec
     )
      {
        m_dtrans = new int[spec.m_dtrans_ncols];
        m_accept = null;
        m_anchor = CSpec.NONE;
        m_label = label;
      }
}

/***************************************************************
  Class: CDfa
  **************************************************************/
class CDfa
{
  /***************************************************************
    Member Variables
    ***********************************************************/
  int m_group;
  boolean m_mark;
  CAccept m_accept;
  int m_anchor;
  Vector m_nfa_set;
  SparseBitSet m_nfa_bit;
  int m_label;

  /***************************************************************
    Function: CDfa
    **************************************************************/
  CDfa
    (
     int label
     )
      {
        m_group = 0;
        m_mark = false;

        m_accept = null;
        m_anchor = CSpec.NONE;

        m_nfa_set = null;
        m_nfa_bit = null;

        m_label = label;
      }
}

/***************************************************************
  Class: CAccept
 **************************************************************/
class CAccept
{
  /***************************************************************
    Member Variables
    **************************************************************/
  char m_action[];
  int m_action_read;
  int m_line_number;

  /***************************************************************
    Function: CAccept
    **************************************************************/
  CAccept
    (
     char action[],
     int action_read,
     int line_number
     )
      {
        int elem;

        m_action_read = action_read;

        m_action = new char[m_action_read];
        for (elem = 0; elem < m_action_read; ++elem)
          {
            m_action[elem] = action[elem];
          }

        m_line_number = line_number;
      }

  /***************************************************************
    Function: CAccept
    **************************************************************/
  CAccept
    (
     CAccept accept
     )
      {
        int elem;

        m_action_read = accept.m_action_read;
	m_action = new char[m_action_read];
	for (elem = 0; elem < m_action_read; ++elem)
	  {
	    m_action[elem] = accept.m_action[elem];
	  }

	m_line_number = accept.m_line_number;
      }

  /***************************************************************
    Function: mimic
    **************************************************************/
  void mimic
    (
     CAccept accept
     )
      {
	int elem;

	m_action_read = accept.m_action_read;

	m_action = new char[m_action_read];
	for (elem = 0; elem < m_action_read; ++elem)
	  {
	    m_action[elem] = accept.m_action[elem];
	  }
      }
}

/***************************************************************
  Class: CAcceptAnchor
  **************************************************************/
class CAcceptAnchor
{
  /***************************************************************
    Member Variables
    **************************************************************/
  CAccept m_accept;
  int m_anchor;

  /***************************************************************
    Function: CAcceptAnchor
    **************************************************************/
  CAcceptAnchor
    (
     )
      {
	m_accept = null;
	m_anchor = CSpec.NONE;
      }
}

/***************************************************************
  Class: CNfaPair
  **************************************************************/
class CNfaPair
{
  /***************************************************************
    Member Variables
    **************************************************************/
  CNfa m_start;
  CNfa m_end;

  /***************************************************************
    Function: CNfaPair
    **************************************************************/
  CNfaPair
    (
     )
      {
	m_start = null;
	m_end = null;
      }
}
```

```
/***************************************************************
  Class: CInput
  Description: 
 **************************************************************/
class CInput
{
  /***************************************************************
    Member Variables
    **************************************************************/
  private java.io.BufferedReader m_input; /* JLex specification file. */

  boolean m_eof_reached; /* Whether EOF has been encountered. */
  boolean m_pushback_line; 

  char m_line[]; /* Line buffer. */
  int m_line_read; /* Number of bytes read into line buffer. */
  int m_line_index; /* Current index into line buffer. */

  int m_line_number; /* Current line number. */

  /***************************************************************
    Constants
    **************************************************************/
  static final boolean EOF = true;
  static final boolean NOT_EOF = false;
  
  /***************************************************************
    Function: CInput
    Description: 
    **************************************************************/
  CInput
    (
     java.io.Reader input
     )
      {
        if (CUtility.DEBUG)
          {
            CUtility.ASSERT(null != input);
          }

        /* Initialize input stream. */
        m_input = new java.io.BufferedReader(input);

        /* Initialize buffers and index counters. */
        m_line = null;
        m_line_read = 0;
        m_line_index = 0;

        /* Initialize state variables. */
        m_eof_reached = false;
        m_line_number = 0;
        m_pushback_line = false;
      }

  /***************************************************************
    Function: getLine
    Description: Returns true on EOF, false otherwise.
    Guarantees not to return a blank line, or a line
    of zero length.
    **************************************************************/
  boolean getLine 
    (
     )
      throws java.io.IOException
      {
        String lineStr;
        int elem;
        
        /* Has EOF already been reached? */
        if (m_eof_reached)
          {
            return EOF;
          }
        
        /* Pushback current line? */
        if (m_pushback_line)
          {
            m_pushback_line = false;

            /* Check for empty line. */
            for (elem = 0; elem < m_line_read; ++elem)
              {
                if (false == CUtility.isspace(m_line[elem]))
                  {
                    break;
                  }
              }

            /* Nonempty? */
            if (elem < m_line_read)
              {
                m_line_index = 0;
                return NOT_EOF;
              }
          }

        while (true)
          {
            if (null == (lineStr = m_input.readLine()))
              {
                m_eof_reached = true;
                m_line_index = 0;
                return EOF;
              }
            m_line = (lineStr + "\n").toCharArray();
            m_line_read=m_line.length;
            ++m_line_number;

            /* Check for empty lines and discard them. */
            elem = 0;
            while (CUtility.isspace(m_line[elem]))
              {
                ++elem;
                if (elem == m_line_read)
                  {
                    break;
                  }
              }

            if (elem < m_line_read)
              {
                break;
              }
          }

        m_line_index = 0;
        return NOT_EOF;
      }
}

/********************************************************
  Class: Utility
  *******************************************************/
class CUtility
{
  /********************************************************
    Constants
    *******************************************************/
  static final boolean DEBUG = true;
  static final boolean SLOW_DEBUG = true;
  static final boolean DUMP_DEBUG = true;
  /*static final boolean DEBUG = false;
  static final boolean SLOW_DEBUG = false;
  static final boolean DUMP_DEBUG = false;*/
  static final boolean DESCENT_DEBUG = false;
  static final boolean OLD_DEBUG = false;
  static final boolean OLD_DUMP_DEBUG = false;
  static final boolean FOODEBUG = false;
  static final boolean DO_DEBUG = false;
```

```
/********************************************************
  Constants: Integer Bounds
  ********************************************************/
static final int INT_MAX = 2147483647;

static final int MAX_SEVEN_BIT = 127;
static final int MAX_EIGHT_BIT = 255;
static final int MAX_SIXTEEN_BIT=65535;

/********************************************************
  Function: enter
  Description: Debugging routine.
  ********************************************************/
static void enter
  (
   String descent,
   char lexeme,
   int token
   )
    {
      System.out.println("Entering " + descent
                         + " [lexeme: " + lexeme
                         + "] [token: " + token + "]");
    }

/********************************************************
  Function: leave
  Description: Debugging routine.
  ********************************************************/
static void leave
  (
   String descent,
   char lexeme,
   int token
   )
    {
      System.out.println("Leaving " + descent
                         + " [lexeme:" + lexeme
                         + "] [token:" + token + "]");
    }

/********************************************************
  Function: ASSERT
  Description: Debugging routine.
  ********************************************************/
static void ASSERT
  (
   boolean expr
   )
    {
      if (DEBUG && false == expr)
        {
          System.out.println("Assertion Failed");
          throw new Error("Assertion Failed.");
        }
    }

/***************************************************************
  Function: doubleSize
  **************************************************************/
static char[] doubleSize
  (
   char oldBuffer[]
   )
    {
      char newBuffer[] = new char[2 * oldBuffer.length];
      int elem;

      for (elem = 0; elem < oldBuffer.length; ++elem)
        {
          newBuffer[elem] = oldBuffer[elem];
        }

      return newBuffer;
    }
```

```
/********************************************************
  Function: doubleSize
  ********************************************************/
static byte[] doubleSize
  (
   byte oldBuffer[]
   )
    {
      byte newBuffer[] = new byte[2 * oldBuffer.length];
      int elem;

      for (elem = 0; elem < oldBuffer.length; ++elem)
        {
          newBuffer[elem] = oldBuffer[elem];
        }

      return newBuffer;
    }

/*************************************************
  Function: hex2bin
  *************************************************/
static char hex2bin
  (
   char c
   )
    {
      if ('0' <= c && '9' >= c)
        {
          return (char) (c - '0');
        }
      else if ('a' <= c && 'f' >= c)
        {
          return (char) (c - 'a' + 10);
        }
      else if ('A' <= c && 'F' >= c)
        {
          return (char) (c - 'A' + 10);
        }

      CError.impos("Bad hexidecimal digit" + c);
      return 0;
    }

/*************************************************
  Function: ishexdigit
  *************************************************/
static boolean ishexdigit
  (
   char c
   )
    {
      if (('0' <= c && '9' >= c)
          || ('a' <= c && 'f' >= c)
          || ('A' <= c && 'F' >= c))
        {
          return true;
        }

      return false;
    }

/*************************************************
  Function: oct2bin
  *************************************************/
static char oct2bin
  (
   char c
   )
    {
      if ('0' <= c && '7' >= c)
        {
          return (char) (c - '0');
        }
      CError.impos("Bad octal digit " + c);
      return 0;
    }

/********************************************************
  Function: isoctdigit
  *******************************************************/
static boolean isoctdigit
  (
   char c
   )
    {
      if ('0' <= c && '7' >= c)
        {
          return true;
        }

      return false;
    }

/********************************************************
  Function: isspace
  *******************************************************/
static boolean isspace
  (
   char c
   )
    {
      if ('\b' == c
          || '\t' == c
          || '\n' == c
          || '\f' == c
          || '\r' == c
          || ' ' == c)
        {
          return true;
        }

      return false;
    }

/********************************************************
  Function: isnewline
  *******************************************************/
static boolean isnewline
  (
   char c
   )
    {
      if ('\n' == c
          || '\r' == c)
          {
          return true;
        }

      return false;
    }

/********************************************************
  Function: bytencmp
  Description: Compares up to n elements of
  byte array a[] against byte array b[].
  The first byte comparison is made between
  a[a_first] and b[b_first].  Comparisons continue
  until the null terminating byte '\0' is reached
  or until n bytes are compared.
  Return Value: Returns 0 if arrays are the
  same up to and including the null terminating byte
  or up to and including the first n bytes,
  whichever comes first.
  *******************************************************/
static int bytencmp
  (
   byte a[],
      int a_first,
      byte b[],
      int b_first,
      int n
      )
       {
         int elem;

         for (elem = 0; elem < n; ++elem)
           {
             /*System.out.print((char) a[a_first + elem]);
             System.out.print((char) b[b_first + elem]);*/

             if ('\0' == a[a_first + elem] && '\0' == b[b_first + elem])
               {
                 /*System.out.println("return 0");*/
                 return 0;
               }
             if (a[a_first + elem] < b[b_first + elem])
               {
                 /*System.out.println("return 1");*/
                 return 1;
               }
             else if (a[a_first + elem] > b[b_first + elem])
               {
                 /*System.out.println("return -1");*/
                 return -1;
               }
           }

         /*System.out.println("return 0");*/
         return 0;
       }

  /********************************************************
    Function: charncmp
    *******************************************************/
  static int charncmp
     (
      char a[],
      int a_first,
      char b[],
      int b_first,
      int n
      )
       {
         int elem;

         for (elem = 0; elem < n; ++elem)
           {
             if ('\0' == a[a_first + elem] && '\0' == b[b_first + elem])
               {
                 return 0;
               }
             if (a[a_first + elem] < b[b_first + elem])
               {
                 return 1;
               }
             else if (a[a_first + elem] > b[b_first + elem])
               {
                 return -1;
               }
           }

         return 0;
       }
}

/********************************************************
  Class: CError
  *******************************************************/
class CError
{
  /********************************************************
    Function: impos
  Description:
  *****************************************************/
static void impos
  (
   String message
   )
    {
      System.out.println("JLex Error: " + message);
    }

/*****************************************************
  Constants
  Description: Error codes for parse_error().
  *****************************************************/
static final int E_BADEXPR = 0;
static final int E_PAREN = 1;
static final int E_LENGTH = 2;
static final int E_BRACKET = 3;
static final int E_BOL = 4;
static final int E_CLOSE = 5;
static final int E_NEWLINE = 6;
static final int E_BADMAC = 7;
static final int E_NOMAC = 8;
static final int E_MACDEPTH = 9;
static final int E_INIT = 10;
static final int E_EOF = 11;
static final int E_DIRECT = 12;
static final int E_INTERNAL = 13;
static final int E_STATE = 14;
static final int E_MACDEF = 15;
static final int E_SYNTAX = 16;
static final int E_BRACE = 17;
static final int E_DASH = 18;
static final int E_ZERO = 19;
static final int E_BADCTRL = 20;

/*****************************************************
  Constants
  Description: String messages for parse_error();
  *****************************************************/
static final String errmsg[] =
  {
    "Malformed regular expression.",
    "Missing close parenthesis.",
    "Too many regular expressions or expression too long.",
    "Missing [ in character class.",
    "^ must be at start of expression or after [.",
    "+ ? or * must follow an expression or subexpression.",
    "Newline in quoted string.",
    "Missing } in macro expansion.",
    "Macro does not exist.",
    "Macro expansions nested too deeply.",
    "JLex has not been successfully initialized.",
    "Unexpected end-of-file found.",
    "Undefined or badly-formed JLex directive.",
    "Internal JLex error.",
    "Unitialized state name.",
    "Badly formed macro definition.",
    "Syntax error.",
    "Missing brace at start of lexical action.",
    "Special character dash - in character class [...] must\n"
      + "\tbe preceded by start-of-range character.",
    "Zero-length regular expression.",
    "Illegal \\^C-style escape sequence (character following caret must\n"
      + "\tbe alphabetic).",
  };

/*****************************************************
  Function: parse_error
  Description:
  *****************************************************/
static void parse_error
  (
   int error_code,
   int line_number
```

```
      )
       {
         System.out.println("Error: Parse error at line "
                            + line_number + ".");
         System.out.println("Description: " + errmsg[error_code]);
         throw new Error("Parse error.");
       }
}

/********************************************************
  Class: CSet
  ********************************************************/
class CSet
{
  /********************************************************
    Member Variables
    ********************************************************/
  private SparseBitSet m_set;
  private boolean m_complement;

  /********************************************************
    Function: CSet
    ********************************************************/
  CSet
    (
     )
    {
      m_set = new SparseBitSet();
      m_complement = false;
    }

  /********************************************************
    Function: complement
    ********************************************************/
  void complement
    (
     )
      {
        m_complement = true;
      }

  /********************************************************
    Function: add
    ********************************************************/
  void add
    (
     int i
     )
      {
        m_set.set(i);
      }

  /********************************************************
    Function: addncase
    ********************************************************/
  void addncase // add, ignoring case.
    (
     char c
     )
      {
        /* Do this in a Unicode-friendly way. */
        /* (note that duplicate adds have no effect) */
        add(c);
        add(Character.toLowerCase(c));
        add(Character.toTitleCase(c));
        add(Character.toUpperCase(c));
      }

  /********************************************************
    Function: contains
    ********************************************************/
  boolean contains
    (
     int i
     )
```

```
      {
        boolean result;

        result = m_set.get(i);

        if (m_complement)
          {
            return (false == result);
          }

        return result;
      }

  /********************************************************
    Function: mimic
    *******************************************************/
  void mimic
    (
     CSet set
     )
      {
        m_complement = set.m_complement;
        m_set = (SparseBitSet) set.m_set.clone();
      }

  /** Map set using character classes [CSA] */
  void map(CSet set, int[] mapping) {
    m_complement = set.m_complement;
    m_set.clearAll();
    for (Enumeration e=set.m_set.elements(); e.hasMoreElements(); ) {
      int old_value =((Integer)e.nextElement()).intValue();
      if (old_value<mapping.length) // skip unmapped characters
        m_set.set(mapping[old_value]);
    }
  }
}

/********************************************************
  Class: CNfa
  *******************************************************/
class CNfa
{
  /********************************************************
    Member Variables
    *******************************************************/
  int m_edge;  /* Label for edge type:
                          character code,
                          CCL (character class),
                          [STATE,
                          SCL (state class),]
                          EMPTY,
                          EPSILON. */

  CSet m_set;  /* Set to store character classes. */
  CNfa m_next;  /* Next state (or null if none). */

  CNfa m_next2;  /* Another state with type == EPSILON
                            and null if not used.
                            The NFA construction should result in two
                            outgoing edges only if both are EPSILON edges. */

  CAccept m_accept;  /* Set to null if nonaccepting state. */
  int m_anchor;  /* Says if and where pattern is anchored. */

  int m_label;

  SparseBitSet m_states;

  /********************************************************
    Constants
    *******************************************************/
  static final int NO_LABEL = -1;

  /********************************************************
    Constants: Edge Types
    Note: Edge transitions on one specific character
    are labelled with the character Ascii (Unicode)
    codes.  So none of the constants below should
    overlap with the natural character codes.
    *****************************************************/
  static final int CCL = -1;
  static final int EMPTY = -2;
  static final int EPSILON = -3;

  /*****************************************************
    Function: CNfa
    *****************************************************/
 CNfa
    (
     )
    {
      m_edge = EMPTY;
      m_set = null;
      m_next = null;
      m_next2 = null;
      m_accept = null;
      m_anchor = CSpec.NONE;
      m_label = NO_LABEL;
      m_states = null;
    }

  /*****************************************************
    Function: mimic
    Description: Converts this NFA state into a copy of
    the input one.
    *****************************************************/
  void mimic
    (
     CNfa nfa
     )
      {
        m_edge = nfa.m_edge;

        if (null != nfa.m_set)
          {
            if (null == m_set)
              {
                m_set = new CSet();
              }
            m_set.mimic(nfa.m_set);
          }
        else
          {
            m_set = null;
          }

        m_next = nfa.m_next;
        m_next2 = nfa.m_next2;
        m_accept = nfa.m_accept;
        m_anchor = nfa.m_anchor;

        if (null != nfa.m_states)
          {
            m_states = (SparseBitSet) nfa.m_states.clone();
          }
        else
          {
            m_states = null;
          }
      }
}

/***************************************************************
  Class: CLexGen
  ***************************************************************/
class CLexGen
{
  /***************************************************************
    Member Variables
    ***************************************************************/
```

```
private java.io.Reader m_instream; /* JLex specification file. */
private java.io.PrintWriter m_outstream; /* Lexical analyzer source file. */

private CInput m_input; /* Input buffer class. */

private Hashtable m_tokens; /* Hashtable that maps characters to their
                               corresponding lexical code for
                               the internal lexical analyzer. */
private CSpec m_spec; /* Spec class holds information
                         about the generated lexer. */
private boolean m_init_flag; /* Flag set to true only upon
                                successful initialization. */

private CMakeNfa m_makeNfa; /* NFA machine generator module. */
private CNfa2Dfa m_nfa2dfa; /* NFA to DFA machine (transition table)
                               conversion module. */
private CMinimize m_minimize; /* Transition table compressor. */
private CSimplifyNfa m_simplifyNfa; /* NFA simplifier using char classes */
private CEmit m_emit; /* Output module that emits source code
                         into the generated lexer file. */


/********************************************************
  Constants
  *******************************************************/
private static final boolean ERROR = false;
private static final boolean NOT_ERROR = true;
private static final int BUFFER_SIZE = 1024;

/********************************************************
  Constants: Token Types
  *******************************************************/
static final int EOS = 1;
static final int ANY = 2;
static final int AT_BOL = 3;
static final int AT_EOL = 4;
static final int CCL_END = 5;
static final int CCL_START = 6;
static final int CLOSE_CURLY = 7;
static final int CLOSE_PAREN = 8;
static final int CLOSURE = 9;
static final int DASH = 10;
static final int END_OF_INPUT = 11;
static final int L = 12;
static final int OPEN_CURLY = 13;
static final int OPEN_PAREN = 14;
static final int OPTIONAL = 15;
static final int OR = 16;
static final int PLUS_CLOSE = 17;

/***************************************************************
  Function: CLexGen
  **************************************************************/
CLexGen
  (
   String filename
   )
    throws java.io.FileNotFoundException, java.io.IOException
    {
      /* Successful initialization flag. */
      m_init_flag = false;

      /* Open input stream. */
      m_instream = new java.io.FileReader(filename);
      if (null == m_instream)
        {
          System.out.println("Error: Unable to open input file "
                             + filename + ".");
          return;
        }

      /* Open output stream. */
      m_outstream
        = new java.io.PrintWriter(new java.io.BufferedWriter(
              new java.io.FileWriter(filename + ".java")));
```

```
	if (null == m_outstream)
	  {
	    System.out.println("Error: Unable to open output file "
			       + filename + ".java.");
	    return;
	  }

	/* Create input buffer class. */
	m_input = new CInput(m_instream);

	/* Initialize character hash table. */
	m_tokens = new Hashtable();
	m_tokens.put(new Character('$'),new Integer(AT_EOL));
	m_tokens.put(new Character('('),new Integer(OPEN_PAREN));
	m_tokens.put(new Character(')'),new Integer(CLOSE_PAREN));
	m_tokens.put(new Character('*'),new Integer(CLOSURE));
	m_tokens.put(new Character('+'),new Integer(PLUS_CLOSE));
	m_tokens.put(new Character('-'),new Integer(DASH));
	m_tokens.put(new Character('.'),new Integer(ANY));
	m_tokens.put(new Character('?'),new Integer(OPTIONAL));
	m_tokens.put(new Character('['),new Integer(CCL_START));
	m_tokens.put(new Character(']'),new Integer(CCL_END));
	m_tokens.put(new Character('^'),new Integer(AT_BOL));
	m_tokens.put(new Character('{'),new Integer(OPEN_CURLY));
	m_tokens.put(new Character('|'),new Integer(OR));
	m_tokens.put(new Character('}'),new Integer(CLOSE_CURLY));

	/* Initialize spec structure. */
	m_spec = new CSpec(this);

	/* Nfa to dfa converter. */
	m_nfa2dfa = new CNfa2Dfa();
	m_minimize = new CMinimize();
	m_makeNfa = new CMakeNfa();
	m_simplifyNfa = new CSimplifyNfa();

	m_emit = new CEmit();

	/* Successful initialization flag. */
	m_init_flag = true;
      }

  /***************************************************************
    Function: generate
    Description:
    **************************************************************/
  void generate
    (
     )
      throws java.io.IOException, java.io.FileNotFoundException
      {
	if (false == m_init_flag)
	  {
	    CError.parse_error(CError.E_INIT,0);
	  }

	if (CUtility.DEBUG)
	  {
	    CUtility.ASSERT(null != this);
	    CUtility.ASSERT(null != m_outstream);
	    CUtility.ASSERT(null != m_input);
	    CUtility.ASSERT(null != m_tokens);
	    CUtility.ASSERT(null != m_spec);
	    CUtility.ASSERT(m_init_flag);
	  }

	/*m_emit.emit_imports(m_spec,m_outstream);*/

	if (m_spec.m_verbose)
	  {
	    System.out.println("Processing first section -- user code.");
	  }
	userCode();
	if (m_input.m_eof_reached)
	  {
	    CError.parse_error(CError.E_EOF,m_input.m_line_number);
	  }

	if (m_spec.m_verbose)
	  {
	    System.out.println("Processing second section -- "
			       + "JLex declarations.");
	  }
	userDeclare();
	if (m_input.m_eof_reached)
	  {
	    CError.parse_error(CError.E_EOF,m_input.m_line_number);
	  }

	if (m_spec.m_verbose)
	  {
	    System.out.println("Processing third section -- lexical rules.");
	  }
	userRules();
	if (CUtility.DO_DEBUG)
	  {
	    print_header();
	  }

	if (m_spec.m_verbose)
	  {
	    System.out.println("Outputting lexical analyzer code.");
	  }
	m_emit.emit(m_spec,m_outstream);

	if (m_spec.m_verbose && true == CUtility.OLD_DUMP_DEBUG)
	  {
	    details();
	  }

	m_outstream.close();
      }

  /***************************************************************
    Function: userCode
    Description: Process first section of specification,
    echoing it into output file.
    **************************************************************/
  private void userCode
    (
     )
      throws java.io.IOException
      {
	int count = 0;

	if (false == m_init_flag)
	  {
	    CError.parse_error(CError.E_INIT,0);
	  }

	if (CUtility.DEBUG)
	  {
	    CUtility.ASSERT(null != this);
	    CUtility.ASSERT(null != m_outstream);
	    CUtility.ASSERT(null != m_input);
	    CUtility.ASSERT(null != m_tokens);
	    CUtility.ASSERT(null != m_spec);
	  }

	if (m_input.m_eof_reached)
	  {
	    CError.parse_error(CError.E_EOF,0);
	  }

	while (true)
	  {
	    if (m_input.getLine())
	      {
		/* Eof reached. */
		CError.parse_error(CError.E_EOF,0);
```

```
            }

          if (2 <= m_input.m_line_read
              && '%' == m_input.m_line[0]
              && '%' == m_input.m_line[1])
            {
              /* Discard remainder of line. */
              m_input.m_line_index = m_input.m_line_read;
              return;
            }

          m_outstream.print(new String(m_input.m_line,0,
                                            m_input.m_line_read));
        }
    }

  /***************************************************************
    Function: getName
    **************************************************************/
  private char[] getName
    (
     )
      {
        char buffer[];
        int elem;

        /* Skip white space. */
        while (m_input.m_line_index < m_input.m_line_read
               && true == CUtility.isspace(m_input.m_line[m_input.m_line_index]))
          {
            ++m_input.m_line_index;
          }

        /* No name? */
        if (m_input.m_line_index >= m_input.m_line_read)
          {
            CError.parse_error(CError.E_DIRECT,0);
          }

        /* Determine length. */
        elem = m_input.m_line_index;
        while (elem < m_input.m_line_read
               && false == CUtility.isnewline(m_input.m_line[elem]))
          {
            ++elem;
          }

        /* Allocate non-terminated buffer of exact length. */
        buffer = new char[elem - m_input.m_line_index];

        /* Copy. */
        elem = 0;
        while (m_input.m_line_index < m_input.m_line_read
               && false == CUtility.isnewline(m_input.m_line[m_input.m_line_index]))
          {
            buffer[elem] = m_input.m_line[m_input.m_line_index];
            ++elem;
            ++m_input.m_line_index;
          }

        return buffer;
      }

  private final int CLASS_CODE = 0;
  private final int INIT_CODE = 1;
  private final int EOF_CODE = 2;
  private final int INIT_THROW_CODE = 3;
  private final int YYLEX_THROW_CODE = 4;
  private final int EOF_THROW_CODE = 5;
  private final int EOF_VALUE_CODE = 6;

  /***************************************************************
    Function: packCode
    Description:
    **************************************************************/
```

```
private char[] packCode
  (
   char start_dir[],
   char end_dir[],
   char prev_code[],
   int prev_read,
   int specified
   )
    throws java.io.IOException
    {
      if (CUtility.DEBUG)
        {
          CUtility.ASSERT(INIT_CODE == specified
                          || CLASS_CODE == specified
                          || EOF_CODE == specified
                          || EOF_VALUE_CODE == specified
                          || INIT_THROW_CODE == specified
                          || YYLEX_THROW_CODE == specified
                          || EOF_THROW_CODE == specified);
        }

      if (0 != CUtility.charncmp(m_input.m_line,
                                 0,
                                 start_dir,
                                 0,
                                 start_dir.length - 1))
        {
          CError.parse_error(CError.E_INTERNAL,0);
        }

      if (null == prev_code)
        {
          prev_code = new char[BUFFER_SIZE];
          prev_read = 0;
        }

      if (prev_read >= prev_code.length)
        {
          prev_code = CUtility.doubleSize(prev_code);
        }

      m_input.m_line_index = start_dir.length - 1;
      while (true)
        {
          while (m_input.m_line_index >= m_input.m_line_read)
            {
              if (m_input.getLine())
                {
                  CError.parse_error(CError.E_EOF,m_input.m_line_number);
                }

              if (0 == CUtility.charncmp(m_input.m_line,
                                         0,
                                         end_dir,
                                         0,
                                         end_dir.length - 1))
                {
                  m_input.m_line_index = end_dir.length - 1;

                  switch (specified)
                    {
                    case CLASS_CODE:
                      m_spec.m_class_read = prev_read;
                      break;

                    case INIT_CODE:
                      m_spec.m_init_read = prev_read;
                      break;

                    case EOF_CODE:
                      m_spec.m_eof_read = prev_read;
                      break;

                    case EOF_VALUE_CODE:
                      m_spec.m_eof_value_read = prev_read;
                      break;

                    case INIT_THROW_CODE:
                      m_spec.m_init_throw_read = prev_read;
                      break;

                    case YYLEX_THROW_CODE:
                      m_spec.m_yylex_throw_read = prev_read;
                      break;

                    case EOF_THROW_CODE:
                      m_spec.m_eof_throw_read = prev_read;
                      break;

                    default:
                      CError.parse_error(CError.E_INTERNAL,m_input.m_line_number);
                      break;
                    }

                  return prev_code;
                }
            }

          while (m_input.m_line_index < m_input.m_line_read)
            {
              prev_code[prev_read] = m_input.m_line[m_input.m_line_index];
              ++prev_read;
              ++m_input.m_line_index;

              if (prev_read >= prev_code.length)
                {
                  prev_code = CUtility.doubleSize(prev_code);
                }
            }
        }
    }

  /***************************************************************
    Member Variables: JLex directives.
    **************************************************************/
  private char m_state_dir[] = {
    '%', 's', 't',
    'a', 't', 'e',
    '\0'
    };

  private char m_char_dir[] = {
    '%', 'c', 'h',
    'a', 'r',
    '\0'
    };

  private char m_line_dir[] = {
    '%', 'l', 'i',
    'n', 'e',
    '\0'
    };

  private char m_cup_dir[] = {
    '%', 'c', 'u',
    'p',
    '\0'
    };

  private char m_class_dir[] = {
    '%', 'c', 'l',
    'a', 's', 's',
    '\0'
    };

  private char m_implements_dir[] = {
    '%', 'i', 'm', 'p', 'l', 'e', 'm', 'e', 'n', 't', 's',
    '\0'
    };
```

```
	private char m_function_dir[] = {
		'%', 'f', 'u',
		'n', 'c', 't',
		'i', 'o', 'n',
		'\0'
		};

	private char m_type_dir[] = {
		'%', 't', 'y',
		'p', 'e',
		'\0'
		};

	private char m_integer_dir[] = {
		'%', 'i', 'n',
		't', 'e', 'g',
		'e', 'r',
		'\0'
		};

	private char m_intwrap_dir[] = {
		'%', 'i', 'n',
		't', 'w', 'r',
		'a', 'p',
		'\0'
		};

	private char m_full_dir[] = {
		'%', 'f', 'u',
		'l', 'l',
		'\0'
		};

	private char m_unicode_dir[] = {
		'%', 'u', 'n',
		'i', 'c', 'o',
		'd', 'e',
		'\0'
		};

	private char m_ignorecase_dir[] = {
		'%', 'i', 'g',
		'n', 'o', 'r',
		'e', 'c', 'a',
		's', 'e',
		'\0'
		};

	private char m_notunix_dir[] = {
		'%', 'n', 'o',
		't', 'u', 'n',
		'i', 'x',
		'\0'
		};

	private char m_init_code_dir[] = {
		'%', 'i', 'n',
		'i', 't', '{',
		'\0'
		};

	private char m_init_code_end_dir[] = {
		'%', 'i', 'n',
		'i', 't', '}',
		'\0'
		};

	private char m_init_throw_code_dir[] = {
		'%', 'i', 'n',
		'i', 't', 't',
		'h', 'r', 'o',
		'w', '{',
		'\0'
		};
```

```
        private char m_init_throw_code_end_dir[] = {
          '%', 'i', 'n',
          'i', 't', 't',
          'h', 'r', 'o',
          'w', '}',
          '\0'
          };

        private char m_yylex_throw_code_dir[] = {
          '%', 'y', 'y', 'l',
          'e', 'x', 't',
          'h', 'r', 'o',
          'w', '{',
          '\0'
          };

        private char m_yylex_throw_code_end_dir[] = {
          '%', 'y', 'y', 'l',
          'e', 'x', 't',
          'h', 'r', 'o',
          'w', '}',
          '\0'
          };

        private char m_eof_code_dir[] = {
          '%', 'e', 'o',
          'f', '{',
          '\0'
          };

        private char m_eof_code_end_dir[] = {
          '%', 'e', 'o',
          'f', '}',
          '\0'
          };

        private char m_eof_value_code_dir[] = {
          '%', 'e', 'o',
          'f', 'v', 'a',
          'l', '{',
          '\0'
          };

        private char m_eof_value_code_end_dir[] = {
          '%', 'e', 'o',
          'f', 'v', 'a',
          'l', '}',
          '\0'
          };

        private char m_eof_throw_code_dir[] = {
          '%', 'e', 'o',
          'f', 't', 'h',
          'r', 'o', 'w',
          '{',
          '\0'
          };

        private char m_eof_throw_code_end_dir[] = {
          '%', 'e', 'o',
          'f', 't', 'h',
          'r', 'o', 'w',
          '}',
          '\0'
          };

        private char m_class_code_dir[] = {
          '%', '{',
          '\0'
          };

        private char m_class_code_end_dir[] = {
          '%', '}',
          '\0'
          };
```

```
  private char m_yyeof_dir[] = {
    '%', 'y', 'y',
    'e', 'o', 'f',
    '\0'
    };

  private char m_public_dir[] = {
    '%', 'p', 'u',
    'b', 'l', 'i',
    'c', '\0'
    };

  /***************************************************************
    Function: userDeclare
    Description:
    **************************************************************/
  private void userDeclare
    (
     )
      throws java.io.IOException
        {
          int elem;

          if (CUtility.DEBUG)
            {
              CUtility.ASSERT(null != this);
              CUtility.ASSERT(null != m_outstream);
              CUtility.ASSERT(null != m_input);
              CUtility.ASSERT(null != m_tokens);
              CUtility.ASSERT(null != m_spec);
            }

          if (m_input.m_eof_reached)
            {
              /* End-of-file. */
              CError.parse_error(CError.E_EOF,
                                 m_input.m_line_number);
            }

          while (false == m_input.getLine())
            {
              /* Look for double percent. */
              if (2 <= m_input.m_line_read
                  && '%' == m_input.m_line[0]
                  && '%' == m_input.m_line[1])
                {
                  /* Mess around with line. */
                  m_input.m_line_read -= 2;
                  System.arraycopy(m_input.m_line, 2,
                                   m_input.m_line, 0, m_input.m_line_read);

                  m_input.m_pushback_line = true;
                  /* Check for and discard empty line. */
                  if (0 == m_input.m_line_read
                      || '\n' == m_input.m_line[0])
                    {
                      m_input.m_pushback_line = false;
                    }

                  return;
                }

              if (0 == m_input.m_line_read)
                {
                  continue;
                }

              if ('%' == m_input.m_line[0])
                {
                  /* Special lex declarations. */
                  if (1 >= m_input.m_line_read)
                    {
                      CError.parse_error(CError.E_DIRECT,
                                         m_input.m_line_number);
                      continue;
                    }

                  switch (m_input.m_line[1])
                    {
                    case '{':
                      if (0 == CUtility.charncmp(m_input.m_line,
                                                 0,
                                                 m_class_code_dir,
                                                 0,
                                                 m_class_code_dir.length - 1))
                        {
                          m_spec.m_class_code = packCode(m_class_code_dir,
                                                         m_class_code_end_dir,
                                                         m_spec.m_class_code,
                                                         m_spec.m_class_read,
                                                         CLASS_CODE);
                          break;
                        }

                      /* Bad directive. */
                      CError.parse_error(CError.E_DIRECT,
                                         m_input.m_line_number);
                      break;

                    case 'c':
                      if (0 == CUtility.charncmp(m_input.m_line,
                                                 0,
                                                 m_char_dir,
                                                 0,
                                                 m_char_dir.length - 1))
                        {
                          /* Set line counting to ON. */
                          m_input.m_line_index = m_char_dir.length;
                          m_spec.m_count_chars = true;
                          break;
                        }
                      else if (0 == CUtility.charncmp(m_input.m_line,
                                                      0,
                                                      m_class_dir,
                                                      0,
                                                      m_class_dir.length - 1))
                        {
                          m_input.m_line_index = m_class_dir.length;
                          m_spec.m_class_name = getName();
                          break;
                        }
                      else if (0 == CUtility.charncmp(m_input.m_line,
                                                      0,
                                                      m_cup_dir,
                                                      0,
                                                      m_cup_dir.length - 1))
                        {
                          /* Set Java CUP compatibility to ON. */
                          m_input.m_line_index = m_cup_dir.length;
                          m_spec.m_cup_compatible = true;
                          // this is what %cup does: [CSA, 27-Jul-1999]
                          m_spec.m_implements_name =
                              "java_cup.runtime.Scanner".toCharArray();
                          m_spec.m_function_name =
                              "next_token".toCharArray();
                          m_spec.m_type_name =
                              "java_cup.runtime.Symbol".toCharArray();
                          break;
                        }

                      /* Bad directive. */
                      CError.parse_error(CError.E_DIRECT,
                                         m_input.m_line_number);
                      break;

                    case 'e':
                      if (0 == CUtility.charncmp(m_input.m_line,
                                                 0,
                                                 m_eof_code_dir,
                                                 0,
                                                 m_eof_code_dir.length - 1))
                        {
                          m_spec.m_eof_code = packCode(m_eof_code_dir,
                                                       m_eof_code_end_dir,
                                                       m_spec.m_eof_code,
                                                       m_spec.m_eof_read,
                                                       EOF_CODE);
                          break;
                        }
                      else if (0 == CUtility.charncmp(m_input.m_line,
                                                      0,
                                                      m_eof_value_code_dir,
                                                      0,
                                                      m_eof_value_code_dir.length - 1))
                        {
                          m_spec.m_eof_value_code = packCode(m_eof_value_code_dir,
                                                             m_eof_value_code_end_dir,
                                                             m_spec.m_eof_value_code,
                                                             m_spec.m_eof_value_read,
                                                             EOF_VALUE_CODE);
                          break;
                        }
                      else if (0 == CUtility.charncmp(m_input.m_line,
                                                      0,
                                                      m_eof_throw_code_dir,
                                                      0,
                                                      m_eof_throw_code_dir.length - 1))
                        {
                          m_spec.m_eof_throw_code = packCode(m_eof_throw_code_dir,
                                                       m_eof_throw_code_end_dir,
                                                       m_spec.m_eof_throw_code,
                                                       m_spec.m_eof_throw_read,
                                                       EOF_THROW_CODE);
                          break;
                        }

                      /* Bad directive. */
                      CError.parse_error(CError.E_DIRECT,
                                         m_input.m_line_number);
                      break;

                    case 'f':
                      if (0 == CUtility.charncmp(m_input.m_line,
                                                 0,
                                                 m_function_dir,
                                                 0,
                                                 m_function_dir.length - 1))
                        {
                          /* Set line counting to ON. */
                          m_input.m_line_index = m_function_dir.length;
                          m_spec.m_function_name = getName();
                          break;
                        }
                      else if (0 == CUtility.charncmp(m_input.m_line,
                                                      0,
                                                      m_full_dir,
                                                      0,
                                                      m_full_dir.length - 1))
                        {
                          m_input.m_line_index = m_full_dir.length;
                          m_spec.m_dtrans_ncols = CUtility.MAX_EIGHT_BIT + 1;
                          break;
                        }

                      /* Bad directive. */
                      CError.parse_error(CError.E_DIRECT,
                                         m_input.m_line_number);
                      break;

                    case 'i':
                      if (0 == CUtility.charncmp(m_input.m_line,
                                                 0,
                                                 m_integer_dir,
                                                 0,
                                                 m_integer_dir.length - 1))
                        {
                          /* Set line counting to ON. */
                          m_input.m_line_index = m_integer_dir.length;
                          m_spec.m_integer_type = true;
                          break;
                        }
                      else if (0 == CUtility.charncmp(m_input.m_line,
                                                      0,
                                                      m_intwrap_dir,
                                                      0,
                                                      m_intwrap_dir.length - 1))
                        {
                          /* Set line counting to ON. */
                          m_input.m_line_index = m_integer_dir.length;
                          m_spec.m_intwrap_type = true;
                          break;
                        }
                      else if (0 == CUtility.charncmp(m_input.m_line,
                                                      0,
                                                      m_init_code_dir,
                                                      0,
                                                      m_init_code_dir.length - 1))
                        {
                          m_spec.m_init_code = packCode(m_init_code_dir,
                                                        m_init_code_end_dir,
                                                        m_spec.m_init_code,
                                                        m_spec.m_init_read,
                                                        INIT_CODE);
                          break;
                        }
                      else if (0 == CUtility.charncmp(m_input.m_line,
                                                      0,
                                                      m_init_throw_code_dir,
                                                      0,
                                                      m_init_throw_code_dir.length - 1))
                        {
                          m_spec.m_init_throw_code = packCode(m_init_throw_code_dir,
                                                       m_init_throw_code_end_dir,
                                                       m_spec.m_init_throw_code,
                                                       m_spec.m_init_throw_read,
                                                       INIT_THROW_CODE);
                          break;
                        }
                      else if (0 == CUtility.charncmp(m_input.m_line,
                                                      0,
                                                      m_implements_dir,
                                                      0,
                                                      m_implements_dir.length - 1))
                        {
                          m_input.m_line_index = m_implements_dir.length;
                          m_spec.m_implements_name = getName();
                          break;
                        }
                      else if (0 == CUtility.charncmp(m_input.m_line,
                                                      0,
                                                      m_ignorecase_dir,
                                                      0,
                                                      m_ignorecase_dir.length-1))
                        {
                          /* Set m_ignorecase to ON. */
                          m_input.m_line_index = m_ignorecase_dir.length;
                          m_spec.m_ignorecase = true;
                          break;
                        }

                      /* Bad directive. */
                      CError.parse_error(CError.E_DIRECT,
                                         m_input.m_line_number);
                      break;

                    case 'l':
                      if (0 == CUtility.charncmp(m_input.m_line,
                                                 0,
                                                 m_line_dir,
                                                 0,
                                                 m_line_dir.length - 1))
                        {
                          /* Set line counting to ON. */
                          m_input.m_line_index = m_line_dir.length;
                          m_spec.m_count_lines = true;
                          break;
                        }

                      /* Bad directive. */
                      CError.parse_error(CError.E_DIRECT,
                                         m_input.m_line_number);
                      break;

                    case 'n':
                      if (0 == CUtility.charncmp(m_input.m_line,
                                                 0,
                                                 m_notunix_dir,
                                                 0,
                                                 m_notunix_dir.length - 1))
                        {
                          /* Set line counting to ON. */
                          m_input.m_line_index = m_notunix_dir.length;
                          m_spec.m_unix = false;
                          break;
                        }

                      /* Bad directive. */
                      CError.parse_error(CError.E_DIRECT,
                                         m_input.m_line_number);
                      break;

                    case 'p':
                      if (0 == CUtility.charncmp(m_input.m_line,
                                                 0,
                                                 m_public_dir,
                                                 0,
                                                 m_public_dir.length - 1))
                        {
                          /* Set public flag. */
                          m_input.m_line_index = m_public_dir.length;
                          m_spec.m_public = true;
                          break;
                        }

                      /* Bad directive. */
                      CError.parse_error(CError.E_DIRECT,
                                         m_input.m_line_number);
                      break;

                    case 's':
                      if (0 == CUtility.charncmp(m_input.m_line,
                                                 0,
                                                 m_state_dir,
                                                 0,
                                                 m_state_dir.length - 1))
                        {
                          /* Recognize state list. */
                          m_input.m_line_index = m_state_dir.length;
                          saveStates();
                          break;
                        }

                      /* Undefined directive. */
                      CError.parse_error(CError.E_DIRECT,
                                         m_input.m_line_number);
                      break;

                    case 't':
                      if (0 == CUtility.charncmp(m_input.m_line,
                                                 0,
                                                 m_type_dir,
                                                 0,
                                                 m_type_dir.length - 1))
                        {
                          /* Set Java CUP compatibility to ON. */
                          m_input.m_line_index = m_type_dir.length;
                          m_spec.m_type_name = getName();
                          break;
                        }

                      /* Undefined directive. */
                      CError.parse_error(CError.E_DIRECT,
                                         m_input.m_line_number);
                      break;

                    case 'u':
                      if (0 == CUtility.charncmp(m_input.m_line,
                                                 0,
                                                 m_unicode_dir,
                                                 0,
                                                 m_unicode_dir.length - 1))
                        {
                          m_input.m_line_index = m_unicode_dir.length;
                          m_spec.m_dtrans_ncols= CUtility.MAX_SIXTEEN_BIT + 1;
                          break;
                        }

                      /* Bad directive. */
                      CError.parse_error(CError.E_DIRECT,
                                         m_input.m_line_number);
                      break;

                    case 'y':
                      if (0 == CUtility.charncmp(m_input.m_line,
                                                 0,
                                                 m_yyeof_dir,
                                                 0,
                                                 m_yyeof_dir.length - 1))
                        {
                          m_input.m_line_index = m_yyeof_dir.length;
                          m_spec.m_yyeof = true;
                          break;
                        } else if (0 == CUtility.charncmp(m_input.m_line,
                                                          0,
                                                          m_yylex_throw_code_dir,
                                                          0,
                                                          m_yylex_throw_code_dir.length - 1))
                        {
                          m_spec.m_yylex_throw_code = packCode(m_yylex_throw_code_dir,
                                                               m_yylex_throw_code_end_dir,
                                                       m_spec.m_yylex_throw_code,
                                                       m_spec.m_yylex_throw_read,
                                                       YYLEX_THROW_CODE);
                          break;
                        }


                      /* Bad directive. */
                      CError.parse_error(CError.E_DIRECT,
                                         m_input.m_line_number);
                      break;

                    default:
                      /* Undefined directive. */
                      CError.parse_error(CError.E_DIRECT,
                                         m_input.m_line_number);
                      break;
                    }
                }
              else
                {
                  /* Regular expression macro. */
                  m_input.m_line_index = 0;
                  saveMacro();
                }

              if (CUtility.OLD_DEBUG)
                {
                  System.out.println("Line number "
                                    + m_input.m_line_number + ":");
                        System.out.print(new String(m_input.m_line,
                                                    0,m_input.m_line_read));
                    }
                }
            }

    /***************************************************************
      Function: userRules
      Description: Processes third section of JLex
      specification and creates minimized transition table.
      **************************************************************/
    private void userRules
      (
       )
        throws java.io.IOException
        {
          int code;

          if (false == m_init_flag)
            {
              CError.parse_error(CError.E_INIT,0);
            }

          if (CUtility.DEBUG)
            {
              CUtility.ASSERT(null != this);
              CUtility.ASSERT(null != m_outstream);
              CUtility.ASSERT(null != m_input);
              CUtility.ASSERT(null != m_tokens);
              CUtility.ASSERT(null != m_spec);
            }

          /* UNDONE: Need to handle states preceding rules. */

          if (m_spec.m_verbose)
            {
              System.out.println("Creating NFA machine representation.");
            }
          m_makeNfa.allocate_BOL_EOF(m_spec);
          m_makeNfa.thompson(this,m_spec,m_input);

          m_simplifyNfa.simplify(m_spec);

          /*print_nfa();*/

          if (CUtility.DEBUG)
            {
              CUtility.ASSERT(END_OF_INPUT == m_spec.m_current_token);
            }

          if (m_spec.m_verbose)
            {
              System.out.println("Creating DFA transition table.");
            }
          m_nfa2dfa.make_dfa(this,m_spec);

          if (CUtility.FOODEBUG) {
            print_header();
          }

          if (m_spec.m_verbose)
            {
              System.out.println("Minimizing DFA transition table.");
            }
          m_minimize.min_dfa(m_spec);
        }

    /***************************************************************
      Function: printccl
      Description: Debugging routine that outputs readable form
      of character class.
      **************************************************************/
    private void printccl
      (
	 CSet set
	 )
	  {
	    int i;

	    System.out.print(" [");
	    for (i = 0; i < m_spec.m_dtrans_ncols; ++i)
	      {
		if (set.contains(i))
		  {
		    System.out.print(interp_int(i));
		  }
	      }
	    System.out.print(']');
	  }

  /***************************************************************
    Function: plab
    Description:
    **************************************************************/
  private String plab
    (
     CNfa state
     )
      {
	int index;

	if (null == state)
	  {
	    return (new String("--"));
	  }

	index = m_spec.m_nfa_states.indexOf(state);

	return ((new Integer(index)).toString());
      }

  /***************************************************************
    Function: interp_int
    Description:
    **************************************************************/
  private String interp_int
    (
     int i
     )
      {
	switch (i)
	  {
	  case (int) '\b':
	    return (new String("\\b"));

	  case (int) '\t':
	    return (new String("\\t"));

	  case (int) '\n':
	    return (new String("\\n"));

	  case (int) '\f':
	    return (new String("\\f"));

	  case (int) '\r':
	    return (new String("\\r"));

	  case (int) ' ':
	    return (new String("\\ "));

	  default:
	    return ((new Character((char) i)).toString());
	  }
      }

  /***************************************************************
    Function: print_nfa
    Description:
    **************************************************************/
  void print_nfa
    (
     )
      {
        int elem;
        CNfa nfa;
        int size;
        Enumeration states;
        Integer index;
        int i;
        int j;
        int vsize;
        String state;

        System.out.println("--------------------- NFA -----------------------");

        size = m_spec.m_nfa_states.size();
        for (elem = 0; elem < size; ++elem)
          {
            nfa = (CNfa) m_spec.m_nfa_states.elementAt(elem);

            System.out.print("Nfa state " + plab(nfa) + ": ");

            if (null == nfa.m_next)
              {
                System.out.print("(TERMINAL)");
              }
            else
              {
                System.out.print("--> " + plab(nfa.m_next));
                System.out.print("--> " + plab(nfa.m_next2));

                switch (nfa.m_edge)
                  {
                  case CNfa.CCL:
                    printccl(nfa.m_set);
                    break;

                  case CNfa.EPSILON:
                    System.out.print(" EPSILON ");
                    break;

                  default:
                    System.out.print(" " + interp_int(nfa.m_edge));
                    break;
                  }
              }

            if (0 == elem)
              {
                System.out.print(" (START STATE)");
              }

            if (null != nfa.m_accept)
              {
                System.out.print(" accepting "
                                 + ((0 != (nfa.m_anchor & CSpec.START)) ? "^" : "")
                                 + "<"
                                 + (new String(nfa.m_accept.m_action,0,
                                               nfa.m_accept.m_action_read))
                                 + ">"
                                 + ((0 != (nfa.m_anchor & CSpec.END)) ? "$" : ""));
              }

            System.out.println();
          }

        states = m_spec.m_states.keys();
        while (states.hasMoreElements())
          {
            state = (String) states.nextElement();
            index = (Integer) m_spec.m_states.get(state);

            if (CUtility.DEBUG)
              {
```

```
              CUtility.ASSERT(null != state);
              CUtility.ASSERT(null != index);
            }

          System.out.println("State \"" + state
                             + "\" has identifying index "
                             + index.toString() + ".");
          System.out.print("\tStart states of matching rules: ");

          i = index.intValue();
          vsize = m_spec.m_state_rules[i].size();

          for (j = 0; j < vsize; ++j)
            {
              nfa = (CNfa) m_spec.m_state_rules[i].elementAt(j);

              System.out.print(m_spec.m_nfa_states.indexOf(nfa) + " ");
            }

          System.out.println();
        }

      System.out.println("-------------------- NFA ----------------------");
    }

  /***************************************************************
    Function: getStates
    Description: Parses the state area of a rule,
    from the beginning of a line.
    < state1, state2 ... > regular_expression { action }
    Returns null on only EOF.  Returns all_states,
    initialied properly to correspond to all states,
    if no states are found.
    Special Notes: This function treats commas as optional
    and permits states to be spread over multiple lines.
    **************************************************************/
  private SparseBitSet all_states = null;
  SparseBitSet getStates
    (
     )
      throws java.io.IOException
      {
        int start_state;
        int count_state;
        SparseBitSet states;
        String name;
        Integer index;
        int i;
        int size;

        if (CUtility.DEBUG)
          {
            CUtility.ASSERT(null != this);
            CUtility.ASSERT(null != m_outstream);
            CUtility.ASSERT(null != m_input);
            CUtility.ASSERT(null != m_tokens);
            CUtility.ASSERT(null != m_spec);
          }

        states = null;

        /* Skip white space. */
        while (CUtility.isspace(m_input.m_line[m_input.m_line_index]))
          {
            ++m_input.m_line_index;

            while (m_input.m_line_index >= m_input.m_line_read)
              {
                /* Must just be an empty line. */
                if (m_input.getLine())
                  {
                    /* EOF found. */
                    return null;
                  }
              }
          }

        /* Look for states. */
        if ('<' == m_input.m_line[m_input.m_line_index])
          {
            ++m_input.m_line_index;

            states = new SparseBitSet();

            /* Parse states. */
            while (true)
              {
                /* We may have reached the end of the line. */
                while (m_input.m_line_index >= m_input.m_line_read)
                  {
                    if (m_input.getLine())
                      {
                        /* EOF found. */
                        CError.parse_error(CError.E_EOF,m_input.m_line_number);
                        return states;
                      }
                  }

                while (true)
                  {
                    /* Skip white space. */
                    while (CUtility.isspace(m_input.m_line[m_input.m_line_index]))
                      {
                        ++m_input.m_line_index;

                        while (m_input.m_line_index >= m_input.m_line_read)
                          {
                            if (m_input.getLine())
                              {
                                /* EOF found. */
                                CError.parse_error(CError.E_EOF,m_input.m_line_number);
                                return states;
                              }
                          }
                      }

                    if (',' != m_input.m_line[m_input.m_line_index])
                      {
                        break;
                      }

                    ++m_input.m_line_index;
                  }

                if ('>' == m_input.m_line[m_input.m_line_index])
                  {
                    ++m_input.m_line_index;
                    if (m_input.m_line_index < m_input.m_line_read)
                      {
                        m_advance_stop = true;
                      }
                    return states;
                  }

                /* Read in state name. */
                start_state = m_input.m_line_index;
                while (false == CUtility.isspace(m_input.m_line[m_input.m_line_index])
                       && ',' != m_input.m_line[m_input.m_line_index]
                       && '>' != m_input.m_line[m_input.m_line_index])
                  {
                    ++m_input.m_line_index;

                    if (m_input.m_line_index >= m_input.m_line_read)
                      {
                        /* End of line means end of state name. */
                        break;
                      }
                  }
                count_state = m_input.m_line_index - start_state;
```

```
		/* Save name after checking definition. */
		name = new String(m_input.m_line,
				  start_state,
				  count_state);
		index = (Integer) m_spec.m_states.get(name);
		if (null == index)
		  {
		    /* Uninitialized state. */
		    System.out.println("Uninitialized State Name: " + name);
		    CError.parse_error(CError.E_STATE,m_input.m_line_number);
		  }
		states.set(index.intValue());
	      }
	  }
	
	if (null == all_states)
	  {
	    all_states = new SparseBitSet();

	    size = m_spec.m_states.size();
	    for (i = 0; i < size; ++i)
	      {
		all_states.set(i);
	      }
	  }
	
	if (m_input.m_line_index < m_input.m_line_read)
	  {
	    m_advance_stop = true;
	  }
	return all_states;
      }

  /********************************************************
    Function: expandMacro
    Description: Returns false on error, true otherwise. 
    *******************************************************/
  private boolean expandMacro
    (
     )
      {
	int elem;
	int start_macro;
	int end_macro;
	int start_name;
	int count_name;
	String def;
	int def_elem;
	String name;
	char replace[];
	int rep_elem;

	if (CUtility.DEBUG)
	  {
	    CUtility.ASSERT(null != this);
	    CUtility.ASSERT(null != m_outstream);
	    CUtility.ASSERT(null != m_input);
	    CUtility.ASSERT(null != m_tokens);
	    CUtility.ASSERT(null != m_spec);
	  }

	/* Check for macro. */
	if ('{' != m_input.m_line[m_input.m_line_index])
	  {
	    CError.parse_error(CError.E_INTERNAL,m_input.m_line_number);
	    return ERROR;
	  }
	
	start_macro = m_input.m_line_index;
	elem = m_input.m_line_index + 1;
	if (elem >= m_input.m_line_read)
	  {
	    CError.impos("Unfinished macro name");
	    return ERROR;
	  }
```

```
          /* Get macro name. */
          start_name = elem;
          while ('}' != m_input.m_line[elem])
            {
              ++elem;
              if (elem >= m_input.m_line_read)
                {
                  CError.impos("Unfinished macro name at line "
                               + m_input.m_line_number);
                  return ERROR;
                }
            }
          count_name = elem - start_name;
          end_macro = elem;

          /* Check macro name. */
          if (0 == count_name)
            {
              CError.impos("Nonexistent macro name");
              return ERROR;
            }

          /* Debug checks. */
          if (CUtility.DEBUG)
            {
              CUtility.ASSERT(0 < count_name);
            }

          /* Retrieve macro definition. */
          name = new String(m_input.m_line,start_name,count_name);
          def = (String) m_spec.m_macros.get(name);
          if (null == def)
            {
              /*CError.impos("Undefined macro \"" + name + "\".");*/
              System.out.println("Error: Undefined macro \"" + name + "\".");
              CError.parse_error(CError.E_NOMAC, m_input.m_line_number);
              return ERROR;
            }
          if (CUtility.OLD_DUMP_DEBUG)
            {
              System.out.println("expanded escape: " + def);
            }

          /* Replace macro in new buffer,
             beginning by copying first part of line buffer. */
          replace = new char[m_input.m_line.length];
          for (rep_elem = 0; rep_elem < start_macro; ++rep_elem)
            {
              replace[rep_elem] = m_input.m_line[rep_elem];

              if (CUtility.DEBUG)
                {
                  CUtility.ASSERT(rep_elem < replace.length);
                }
            }

          /* Copy macro definition. */
          if (rep_elem >= replace.length)
            {
              replace = CUtility.doubleSize(replace);
            }
          for (def_elem = 0; def_elem < def.length(); ++def_elem)
            {
              replace[rep_elem] = def.charAt(def_elem);

              ++rep_elem;
              if (rep_elem >= replace.length)
                {
                  replace = CUtility.doubleSize(replace);
                }
            }

          /* Copy last part of line. */
          if (rep_elem >= replace.length)
            {
              replace = CUtility.doubleSize(replace);
            }
          for (elem = end_macro + 1; elem < m_input.m_line_read; ++elem)
            {
              replace[rep_elem] = m_input.m_line[elem];

              ++rep_elem;
              if (rep_elem >= replace.length)
                {
                  replace = CUtility.doubleSize(replace);
                }
            }

          /* Replace buffer. */
          m_input.m_line = replace;
          m_input.m_line_read = rep_elem;

          if (CUtility.OLD_DEBUG)
            {
              System.out.println(new String(m_input.m_line,0,m_input.m_line_read));
            }
          return NOT_ERROR;
        }

    /***************************************************************
      Function: saveMacro
      Description: Saves macro definition of form:
      macro_name = macro_definition
      **************************************************************/
    private void saveMacro
      (
       )
        {
          int elem;
          int start_name;
          int count_name;
          int start_def;
          int count_def;
          boolean saw_escape;
          boolean in_quote;
          boolean in_ccl;

          if (CUtility.DEBUG)
            {
              CUtility.ASSERT(null != this);
              CUtility.ASSERT(null != m_outstream);
              CUtility.ASSERT(null != m_input);
              CUtility.ASSERT(null != m_tokens);
              CUtility.ASSERT(null != m_spec);
            }

          /* Macro declarations are of the following form:
             macro_name macro_definition */

          elem = 0;

          /* Skip white space preceding macro name. */
          while (CUtility.isspace(m_input.m_line[elem]))
            {
              ++elem;
              if (elem >= m_input.m_line_read)
                {
                  /* End of line has been reached,
                     and line was found to be empty. */
                  return;
                }
            }

          /* Read macro name. */
          start_name = elem;
          while (false == CUtility.isspace(m_input.m_line[elem])
                 && '=' != m_input.m_line[elem])
            {
              ++elem;
	    if (elem >= m_input.m_line_read)
	      {
		/* Macro name but no associated definition. */
		CError.parse_error(CError.E_MACDEF,m_input.m_line_number);
	      }
	  }
	count_name = elem - start_name;

	/* Check macro name. */
	if (0 == count_name)
	  {
	    /* Nonexistent macro name. */
	    CError.parse_error(CError.E_MACDEF,m_input.m_line_number);
	  }

	/* Skip white space between name and definition. */
	while (CUtility.isspace(m_input.m_line[elem]))
	  {
	    ++elem;
	    if (elem >= m_input.m_line_read)
	      {
		/* Macro name but no associated definition. */
		CError.parse_error(CError.E_MACDEF,m_input.m_line_number);
	      }
	  }

	if ('=' == m_input.m_line[elem])
	  {
	    ++elem;
	    if (elem >= m_input.m_line_read)
	      {
		/* Macro name but no associated definition. */
		CError.parse_error(CError.E_MACDEF,m_input.m_line_number);
	      }
	  }
	else /* macro definition without = */
		CError.parse_error(CError.E_MACDEF,m_input.m_line_number);

	/* Skip white space between name and definition. */
	while (CUtility.isspace(m_input.m_line[elem]))
	  {
	    ++elem;
	    if (elem >= m_input.m_line_read)
	      {
		/* Macro name but no associated definition. */
		CError.parse_error(CError.E_MACDEF,m_input.m_line_number);
	      }
	  }

	/* Read macro definition. */
	start_def = elem;
	in_quote = false;
	in_ccl = false;
	saw_escape = false;
	while (false == CUtility.isspace(m_input.m_line[elem])
	       || true == in_quote
	       || true == in_ccl
	       || true == saw_escape)
	  {
	    if ('\"' == m_input.m_line[elem] && false == saw_escape)
	      {
		in_quote = !in_quote;
	      }

	    if ('\\' == m_input.m_line[elem] && false == saw_escape)
	      {
		saw_escape = true;
	      }
	    else
	      {
		saw_escape = false;
	      }
	    if (false == saw_escape && false == in_quote) { // CSA, 24-jul-99
	      if ('[' == m_input.m_line[elem] && false == in_ccl)
		in_ccl = true;
```

```
          if (']' == m_input.m_line[elem] && true == in_ccl)
            in_ccl = false;
        }

        ++elem;
        if (elem >= m_input.m_line_read)
          {
            /* End of line. */
            break;
          }
      }
    count_def = elem - start_def;

    /* Check macro definition. */
    if (0 == count_def)
      {
        /* Nonexistent macro name. */
        CError.parse_error(CError.E_MACDEF,m_input.m_line_number);
      }

    /* Debug checks. */
    if (CUtility.DEBUG)
      {
        CUtility.ASSERT(0 < count_def);
        CUtility.ASSERT(0 < count_name);
        CUtility.ASSERT(null != m_spec.m_macros);
      }

    if (CUtility.OLD_DEBUG)
      {
        System.out.println("macro name \""
                           + new String(m_input.m_line,start_name,count_name)
                           + "\".");
        System.out.println("macro definition \""
                           + new String(m_input.m_line,start_def,count_def)
                           + "\".");
      }

    /* Add macro name and definition to table. */
    m_spec.m_macros.put(new String(m_input.m_line,start_name,count_name),
                        new String(m_input.m_line,start_def,count_def));
  }

/***************************************************************
  Function: saveStates
  Description: Takes state declaration and makes entries
  for them in state hashtable in CSpec structure.
  State declaration should be of the form:
  %state name0[, name1, name2 ...]
  (But commas are actually optional as long as there is
  white space in between them.)
  **************************************************************/
private void saveStates
  (
   )
    {
      int start_state;
      int count_state;

      if (CUtility.DEBUG)
        {
          CUtility.ASSERT(null != this);
          CUtility.ASSERT(null != m_outstream);
          CUtility.ASSERT(null != m_input);
          CUtility.ASSERT(null != m_tokens);
          CUtility.ASSERT(null != m_spec);
        }

      /* EOF found? */
      if (m_input.m_eof_reached)
        {
          return;
        }

      /* Debug checks. */
```

```
	if (CUtility.DEBUG)
	  {
	    CUtility.ASSERT('%' == m_input.m_line[0]);
	    CUtility.ASSERT('s' == m_input.m_line[1]);
	    CUtility.ASSERT(m_input.m_line_index <= m_input.m_line_read);
	    CUtility.ASSERT(0 <= m_input.m_line_index);
	    CUtility.ASSERT(0 <= m_input.m_line_read);
	  }

	/* Blank line?  No states? */
	if (m_input.m_line_index >= m_input.m_line_read)
	  {
	    return;
	  }

	while (m_input.m_line_index < m_input.m_line_read)
	  {
	    if (CUtility.OLD_DEBUG)
	      {
		System.out.println("line read " + m_input.m_line_read
				   + "\tline index = " + m_input.m_line_index);
	      }

	    /* Skip white space. */
	    while (CUtility.isspace(m_input.m_line[m_input.m_line_index]))
	      {
		++m_input.m_line_index;
		if (m_input.m_line_index >= m_input.m_line_read)
		  {
		    /* No more states to be found. */
		    return;
		  }
	      }

	    /* Look for state name. */
	    start_state = m_input.m_line_index;
	    while (false == CUtility.isspace(m_input.m_line[m_input.m_line_index])
		   && ',' != m_input.m_line[m_input.m_line_index])
	      {
		++m_input.m_line_index;
		if (m_input.m_line_index >= m_input.m_line_read)
		  {
		    /* End of line and end of state name. */
		    break;
		  }
	      }
	    count_state = m_input.m_line_index - start_state;

	    if (CUtility.OLD_DEBUG)
	      {
		System.out.println("State name \""
				   + new String(m_input.m_line,start_state,count_state)
				   + "\".");
		System.out.println("Integer index \""
				   + m_spec.m_states.size()
				   + "\".");
	      }

	    /* Enter new state name, along with unique index. */
	    m_spec.m_states.put(new String(m_input.m_line,start_state,count_state),
				new Integer(m_spec.m_states.size()));

	    /* Skip comma. */
	    if (',' == m_input.m_line[m_input.m_line_index])
	      {
		++m_input.m_line_index;
		if (m_input.m_line_index >= m_input.m_line_read)
		  {
		    /* End of line. */
		    return;
		  }
	      }
	  }
      }
```

```
/********************************************************
  Function: expandEscape
  Description: Takes escape sequence and returns
  corresponding character code.
  *******************************************************/
private char expandEscape
  (
   )
    {
      char r;

      /* Debug checks. */
      if (CUtility.DEBUG)
        {
          CUtility.ASSERT(m_input.m_line_index < m_input.m_line_read);
          CUtility.ASSERT(0 < m_input.m_line_read);
          CUtility.ASSERT(0 <= m_input.m_line_index);
        }

      if ('\\' != m_input.m_line[m_input.m_line_index])
        {
          ++m_input.m_line_index;
          return m_input.m_line[m_input.m_line_index - 1];
        }
      else
        {
          boolean unicode_escape = false;
          ++m_input.m_line_index;
          switch (m_input.m_line[m_input.m_line_index])
            {
            case 'b':
              ++m_input.m_line_index;
              return '\b';

            case 't':
              ++m_input.m_line_index;
              return '\t';

            case 'n':
              ++m_input.m_line_index;
              return '\n';

            case 'f':
              ++m_input.m_line_index;
              return '\f';

            case 'r':
              ++m_input.m_line_index;
              return '\r';

            case '^':
              ++m_input.m_line_index;
              r=Character.toUpperCase(m_input.m_line[m_input.m_line_index]);
              if (r<'@' || r>'Z') // non-fatal
                  CError.parse_error(CError.E_BADCTRL,m_input.m_line_number);
              r = (char) (r - '@');
              ++m_input.m_line_index;
              return r;

            case 'u':
              unicode_escape = true;
            case 'x':
              ++m_input.m_line_index;
              r = 0;
              for (int i=0; i<(unicode_escape?4:2); i++)
                if (CUtility.ishexdigit(m_input.m_line[m_input.m_line_index]))
                  {
                    r = (char) (r << 4);
                    r = (char) (r | CUtility.hex2bin(m_input.m_line[m_input.m_line_index]))
                    ++m_input.m_line_index;
                  }
                else break;

              return r;
            default:
              if (false == CUtility.isoctdigit(m_input.m_line[m_input.m_line_index]))
                {
                  r = m_input.m_line[m_input.m_line_index];
                  ++m_input.m_line_index;
                }
              else
                {
                  r = 0;
                  for (int i=0; i<3; i++)
                    if (CUtility.isoctdigit(m_input.m_line[m_input.m_line_index]))
                      {
                        r = (char) (r << 3);
                        r = (char) (r | CUtility.oct2bin(m_input.m_line[m_input.m_line_inde
                        ++m_input.m_line_index;
                      }
                    else break;
                }
              return r;
            }
        }
    }

  /***************************************************************
    Function: packAccept
    Description: Packages and returns CAccept
    for action next in input stream.
    **************************************************************/
  CAccept packAccept
    (
     )
      throws java.io.IOException
      {
        CAccept accept;
        char action[];
        int action_index;
        int brackets;
        boolean insinglequotes;
        boolean indoublequotes;
        boolean instarcomment;
        boolean inslashcomment;
        boolean escaped;
        boolean slashed;

        action = new char[BUFFER_SIZE];
        action_index = 0;

        if (CUtility.DEBUG)
          {
            CUtility.ASSERT(null != this);
            CUtility.ASSERT(null != m_outstream);
            CUtility.ASSERT(null != m_input);
            CUtility.ASSERT(null != m_tokens);
            CUtility.ASSERT(null != m_spec);
          }

        /* Get a new line, if needed. */
        while (m_input.m_line_index >= m_input.m_line_read)
          {
            if (m_input.getLine())
              {
                CError.parse_error(CError.E_EOF,m_input.m_line_number);
                return null;
              }
          }

        /* Look for beginning of action. */
        while (CUtility.isspace(m_input.m_line[m_input.m_line_index]))
          {
            ++m_input.m_line_index;

            /* Get a new line, if needed. */
            while (m_input.m_line_index >= m_input.m_line_read)
              {
                if (m_input.getLine())
                  {
                    CError.parse_error(CError.E_EOF,m_input.m_line_number);
                    return null;
                  }
              }
          }

        /* Look for brackets. */
        if ('{' != m_input.m_line[m_input.m_line_index])
          {
            CError.parse_error(CError.E_BRACE,m_input.m_line_number);
          }

        /* Copy new line into action buffer. */
        brackets = 0;
        insinglequotes = indoublequotes = inslashcomment = instarcomment =
        escaped  = slashed = false;
        while (true)
          {
            action[action_index] = m_input.m_line[m_input.m_line_index];

            /* Look for quotes. */
            if ((insinglequotes || indoublequotes) && escaped)
                escaped=false; // only protects one char, but this is enough.
            else if ((insinglequotes || indoublequotes) &&
                     '\\' == m_input.m_line[m_input.m_line_index])
                escaped=true;
            else if (!(insinglequotes || inslashcomment || instarcomment) &&
                     '\"' == m_input.m_line[m_input.m_line_index])
                indoublequotes=!indoublequotes; // unescaped double quote.
            else if (!(indoublequotes || inslashcomment || instarcomment) &&
                     '\'' == m_input.m_line[m_input.m_line_index])
                insinglequotes=!insinglequotes; // unescaped single quote.
            /* Look for comments. */
            if (instarcomment) { // inside "/*" comment; look for "*/"
                if (slashed && '/' == m_input.m_line[m_input.m_line_index])
                    instarcomment = slashed = false;
                else // note that inside a star comment, slashed means starred
                    slashed = ('*' == m_input.m_line[m_input.m_line_index]);
            } else if (!inslashcomment && !insinglequotes && !indoublequotes) {
                // not in comment, look for /* or //
                inslashcomment =
                    (slashed && '/' == m_input.m_line[m_input.m_line_index]);
                instarcomment =
                    (slashed && '*' == m_input.m_line[m_input.m_line_index]);
                slashed = ('/' == m_input.m_line[m_input.m_line_index]);
            }

            /* Look for brackets. */
            if (!insinglequotes && !indoublequotes &&
                !instarcomment && !inslashcomment) {
              if ('{' == m_input.m_line[m_input.m_line_index])
                {
                  ++brackets;
                }
              else if ('}' == m_input.m_line[m_input.m_line_index])
                {
                  --brackets;

                  if (0 == brackets)
                    {
                      ++action_index;
                      ++m_input.m_line_index;

                      break;
                    }
                }
            }

            ++action_index;
            /* Double the buffer size, if needed. */
            if (action_index >= action.length)
              {
                action = CUtility.doubleSize(action);
              }
            ++m_input.m_line_index;
            /* Get a new line, if needed. */
            while (m_input.m_line_index >= m_input.m_line_read)
              {
                inslashcomment = slashed = false;
                if (insinglequotes || indoublequotes) { // non-fatal
                    CError.parse_error(CError.E_NEWLINE,m_input.m_line_number);
                    insinglequotes = indoublequotes = false;
                }
                if (m_input.getLine())
                  {
                    CError.parse_error(CError.E_SYNTAX,m_input.m_line_number);
                    return null;
                  }
              }
          }

        accept = new CAccept(action,action_index,m_input.m_line_number);

        if (CUtility.DEBUG)
          {
            CUtility.ASSERT(null != accept);
          }

        if (CUtility.DESCENT_DEBUG)
          {
            System.out.print("Accepting action:");
            System.out.println(new String(accept.m_action,0,accept.m_action_read));
          }

        return accept;
      }

  /********************************************************
    Function: advance
    Description: Returns code for next token.
    *******************************************************/
  private boolean m_advance_stop = false;
  int advance
    (
     )
      throws java.io.IOException
      {
        boolean saw_escape = false;
        Integer code;

        /*if (m_input.m_line_index > m_input.m_line_read) {
          System.out.println("m_input.m_line_index = " + m_input.m_line_index);
          System.out.println("m_input.m_line_read = " + m_input.m_line_read);
          CUtility.ASSERT(m_input.m_line_index <= m_input.m_line_read);
        }*/

        if (m_input.m_eof_reached)
          {
            /* EOF has already been reached,
               so return appropriate code. */

            m_spec.m_current_token = END_OF_INPUT;
            m_spec.m_lexeme = '\0';
            return m_spec.m_current_token;
          }

        /* End of previous regular expression?
           Refill line buffer? */
        if (EOS == m_spec.m_current_token
            /* ADDED */
            || m_input.m_line_index >= m_input.m_line_read)
            /* ADDED */
          {
            if (m_spec.m_in_quote)
              {
                CError.parse_error(CError.E_SYNTAX,m_input.m_line_number);
              }
            while (true)
              {
                if (false == m_advance_stop
                    || m_input.m_line_index >= m_input.m_line_read)
                  {
                    if (m_input.getLine())
                      {
                        /* EOF has already been reached,
                           so return appropriate code. */

                        m_spec.m_current_token = END_OF_INPUT;
                        m_spec.m_lexeme = '\0';
                        return m_spec.m_current_token;
                      }
                    m_input.m_line_index = 0;
                  }
                else
                  {
                    m_advance_stop = false;
                  }

                while (m_input.m_line_index < m_input.m_line_read
                       && true == CUtility.isspace(m_input.m_line[m_input.m_line_index]))
                  {
                    ++m_input.m_line_index;
                  }

                if (m_input.m_line_index < m_input.m_line_read)
                  {
                    break;
                  }
              }
          }

        if (CUtility.DEBUG) {
          CUtility.ASSERT(m_input.m_line_index <= m_input.m_line_read);
        }

        while (true)
          {
            if (false == m_spec.m_in_quote
                && '{' == m_input.m_line[m_input.m_line_index])
              {
                if (false == expandMacro())
                  {
                    break;
                  }

                if (m_input.m_line_index >= m_input.m_line_read)
                  {
                    m_spec.m_current_token = EOS;
                    m_spec.m_lexeme = '\0';
                    return m_spec.m_current_token;
                  }
              }
            else if ('\"' == m_input.m_line[m_input.m_line_index])
              {
                m_spec.m_in_quote = !m_spec.m_in_quote;
                ++m_input.m_line_index;

                if (m_input.m_line_index >= m_input.m_line_read)
                  {
                    m_spec.m_current_token = EOS;
                    m_spec.m_lexeme = '\0';
                    return m_spec.m_current_token;
                  }
              }
            else
              {
                break;
              }
          }

        if (m_input.m_line_index > m_input.m_line_read) {
          System.out.println("m_input.m_line_index = " + m_input.m_line_index);
```

```
	  System.out.println("m_input.m_line_read = " + m_input.m_line_read);
	  CUtility.ASSERT(m_input.m_line_index <= m_input.m_line_read);
	}

	/* Look for backslash, and corresponding 
	   escape sequence. */
	if ('\\' == m_input.m_line[m_input.m_line_index])
	  {
	    saw_escape = true;
	  }
	else
	  {
	    saw_escape = false;
	  }

	if (false == m_spec.m_in_quote)
	  {
	    if (false == m_spec.m_in_ccl &&
		CUtility.isspace(m_input.m_line[m_input.m_line_index]))
	      {
		/* White space means the end of 
		   the current regular expression. */

		m_spec.m_current_token = EOS;
		m_spec.m_lexeme = '\0';
		return m_spec.m_current_token;
	      }

	    /* Process escape sequence, if needed. */
	    if (saw_escape)
	      {
		m_spec.m_lexeme = expandEscape();
	      }
	    else
	      {
		m_spec.m_lexeme = m_input.m_line[m_input.m_line_index];
		++m_input.m_line_index;
	      }
	  }
	else
	  {
	    if (saw_escape 
		&& (m_input.m_line_index + 1) < m_input.m_line_read
		&& '\"' == m_input.m_line[m_input.m_line_index + 1])
	      {
		m_spec.m_lexeme = '\"';
		m_input.m_line_index = m_input.m_line_index + 2;
	      }
	    else
	      {
		m_spec.m_lexeme = m_input.m_line[m_input.m_line_index];
		++m_input.m_line_index;
	      }
	  }

	code = (Integer) m_tokens.get(new Character(m_spec.m_lexeme));
	if (m_spec.m_in_quote || true == saw_escape)
	  {
	    m_spec.m_current_token = L;
	  }
	else
	  {
	    if (null == code)
	      {
		m_spec.m_current_token = L;
	      }
	    else
	      {
		m_spec.m_current_token = code.intValue();
	      }
	  }

	if (CCL_START == m_spec.m_current_token) m_spec.m_in_ccl = true;
	if (CCL_END   == m_spec.m_current_token) m_spec.m_in_ccl = false;
	if (CUtility.FOODEBUG)
	  {
	    System.out.println("Lexeme: " + m_spec.m_lexeme
			       + "\tToken: " + m_spec.m_current_token
			       + "\tIndex: " + m_input.m_line_index);
	  }

	return m_spec.m_current_token;
      }

  /***************************************************************
    Function: details
    Description: High level debugging routine.
    **************************************************************/
  private void details
    (
     )
      {
	Enumeration names;
	String name;
	String def;
	Enumeration states;
	String state;
	Integer index;
	int elem;
	int size;

	System.out.println();
	System.out.println("\t** Macros **");
	names = m_spec.m_macros.keys();
	while (names.hasMoreElements())
	  {
	    name = (String) names.nextElement();
	    def = (String) m_spec.m_macros.get(name);

	    if (CUtility.DEBUG)
	      {
		CUtility.ASSERT(null != name);
		CUtility.ASSERT(null != def);
	      }

	    System.out.println("Macro name \"" + name
			       + "\" has definition \""
			       + def + "\".");
	  }

	System.out.println();
	System.out.println("\t** States **");
	states = m_spec.m_states.keys();
	while (states.hasMoreElements())
	  {
	    state = (String) states.nextElement();
	    index = (Integer) m_spec.m_states.get(state);

	    if (CUtility.DEBUG)
	      {
		CUtility.ASSERT(null != state);
		CUtility.ASSERT(null != index);
	      }

	    System.out.println("State \"" + state
			       + "\" has identifying index "
			       + index.toString() + ".");
	  }

	System.out.println();
	System.out.println("\t** Character Counting **");
	if (false == m_spec.m_count_chars)
	  {
	    System.out.println("Character counting is off.");
	  }
	else
	  {
	    if (CUtility.DEBUG)
	      {
              CUtility.ASSERT(m_spec.m_count_lines);
            }

          System.out.println("Character counting is on.");
        }

      System.out.println();
      System.out.println("\t** Line Counting **");
      if (false == m_spec.m_count_lines)
        {
          System.out.println("Line counting is off.");
        }
      else
        {
          if (CUtility.DEBUG)
            {
              CUtility.ASSERT(m_spec.m_count_lines);
            }

          System.out.println("Line counting is on.");
        }

      System.out.println();
      System.out.println("\t** Operating System Specificity **");
      if (false == m_spec.m_unix)
        {
          System.out.println("Not generating UNIX-specific code.");
          System.out.println("(This means that \"\\r\\n\" is a "
                             + "newline, rather than \"\\n\".)");
        }
      else
        {
          System.out.println("Generating UNIX-specific code.");
          System.out.println("(This means that \"\\n\" is a "
                             + "newline, rather than \"\\r\\n\".)");
        }

      System.out.println();
      System.out.println("\t** Java CUP Compatibility **");
      if (false == m_spec.m_cup_compatible)
        {
          System.out.println("Generating CUP compatible code.");
          System.out.println("(Scanner implements "
                             + "java_cup.runtime.Scanner.)");
        }
      else
        {
          System.out.println("Not generating CUP compatible code.");
        }

      if (CUtility.FOODEBUG) {
        if (null != m_spec.m_nfa_states && null != m_spec.m_nfa_start)
          {
            System.out.println();
            System.out.println("\t** NFA machine **");
            print_nfa();
        }
      }

      if (null != m_spec.m_dtrans_vector)
        {
          System.out.println();
          System.out.println("\t** DFA transition table **");
          /*print_header();*/
        }

      /*if (null != m_spec.m_accept_vector && null != m_spec.m_anchor_array)
        {
          System.out.println();
          System.out.println("\t** Accept States and Anchor Vector **");
          print_accept();
        }*/
    }

  /***************************************************************
```

```
    function: print_set
    **************************************************************/
  void print_set
    (
     Vector nfa_set
     )
      {
        int size;
        int elem;
        CNfa nfa;

        size = nfa_set.size();

        if (0 == size)
          {
            System.out.print("empty ");
          }

        for (elem = 0; elem < size; ++elem)
          {
            nfa = (CNfa) nfa_set.elementAt(elem);
            /*System.out.print(m_spec.m_nfa_states.indexOf(nfa) + " ");*/
            System.out.print(nfa.m_label + " ");
          }
      }

   /***************************************************************
     Function: print_header
     **************************************************************/
  private void print_header
    (
     )
      {
        Enumeration states;
        int i;
        int j;
        int chars_printed=0;
        CDTrans dtrans;
        int last_transition;
        String str;
        CAccept accept;
        String state;
        Integer index;

        System.out.println("/*---------------------- DFA -----------------------");

        states = m_spec.m_states.keys();
        while (states.hasMoreElements())
          {
            state = (String) states.nextElement();
            index = (Integer) m_spec.m_states.get(state);

            if (CUtility.DEBUG)
              {
                CUtility.ASSERT(null != state);
                CUtility.ASSERT(null != index);
              }

            System.out.println("State \"" + state
                               + "\" has identifying index "
                               + index.toString() + ".");

            i = index.intValue();
            if (CDTrans.F != m_spec.m_state_dtrans[i])
              {
                System.out.println("\tStart index in transition table: "
                                   + m_spec.m_state_dtrans[i]);
              }
            else
              {
                System.out.println("\tNo associated transition states.");
              }
          }

        for (i = 0; i < m_spec.m_dtrans_vector.size(); ++i)
```

```
              {
                dtrans = (CDTrans) m_spec.m_dtrans_vector.elementAt(i);

                if (null == m_spec.m_accept_vector && null == m_spec.m_anchor_array)
                  {
                    if (null == dtrans.m_accept)
                      {
                        System.out.print(" * State " + i + " [nonaccepting]");
                      }
                    else
                      {
                        System.out.print(" * State " + i
                                         + " [accepting, line "
                                         + dtrans.m_accept.m_line_number
                                         + " <"
                                         + (new String(dtrans.m_accept.m_action,0,
                                                       dtrans.m_accept.m_action_read))
                                         + ">]");
                        if (CSpec.NONE != dtrans.m_anchor)
                          {
                            System.out.print(" Anchor: "
                                             + ((0 != (dtrans.m_anchor & CSpec.START))
                                                ? "start " : "")
                                             + ((0 != (dtrans.m_anchor & CSpec.END))
                                                ? "end " : ""));
                          }
                      }
                  }
                else
                  {
                    accept = (CAccept) m_spec.m_accept_vector.elementAt(i);

                    if (null == accept)
                      {
                        System.out.print(" * State " + i + " [nonaccepting]");
                      }
                    else
                      {
                        System.out.print(" * State " + i
                                         + " [accepting, line "
                                         + accept.m_line_number
                                         + " <"
                                         + (new String(accept.m_action,0,
                                                       accept.m_action_read))
                                         + ">]");
                        if (CSpec.NONE != m_spec.m_anchor_array[i])
                          {
                            System.out.print(" Anchor: "
                                             + ((0 != (m_spec.m_anchor_array[i] & CSpec.START))
                                                ? "start " : "")
                                             + ((0 != (m_spec.m_anchor_array[i] & CSpec.END))
                                                ? "end " : ""));
                          }
                      }
                  }

                last_transition = -1;
                for (j = 0; j < m_spec.m_dtrans_ncols; ++j)
                  {
                    if (CDTrans.F != dtrans.m_dtrans[j])
                      {
                        if (last_transition != dtrans.m_dtrans[j])
                          {
                            System.out.println();
                            System.out.print(" *    goto " + dtrans.m_dtrans[j]
                                             + " on ");
                            chars_printed = 0;
                          }

                        str = interp_int((int) j);
                        System.out.print(str);

                        chars_printed = chars_printed + str.length();
                        if (56 < chars_printed)
                          {
```

```
                        System.out.println();
                        System.out.print(" *             ");
                        chars_printed = 0;
                      }

                    last_transition = dtrans.m_dtrans[j];
                  }
              }
            System.out.println();
          }
        System.out.println(" */");
        System.out.println();
      }
}

/*
 * SparseBitSet 25-Jul-1999.
 * C. Scott Ananian <cananian@alumni.princeton.edu>
 *
 * Re-implementation of the standard java.util.BitSet to support sparse
 * sets, which we need to efficiently support unicode character classes.
 */

/**
 * A set of bits. The set automatically grows as more bits are
 * needed.
 *
 * @version     1.00, 25 Jul 1999
 * @author C. Scott Ananian
 */
final class SparseBitSet implements Cloneable {
    /** Sorted array of bit-block offsets. */
    int  offs[];
    /** Array of bit-blocks; each holding BITS bits. */
    long bits[];
    /** Number of blocks currently in use. */
    int size;
    /** log base 2 of BITS, for the identity: x/BITS == x >> LG_BITS */
    static final private int LG_BITS = 6;
    /** Number of bits in a block. */
    static final private int BITS = 1<<LG_BITS;
    /** BITS-1, using the identity: x % BITS == x & (BITS-1) */
    static final private int BITS_M1 = BITS-1;

    /**
     * Creates an empty set.
     */
    public SparseBitSet() {
        bits = new long[4];
        offs = new int [4];
        size = 0;
    }

    /**
     * Creates an empty set with the specified size.
     * @param nbits the size of the set
     */
    public SparseBitSet(int nbits) {
        this();
    }

    /**
     * Creates an empty set with the same size as the given set.
     */
    public SparseBitSet(SparseBitSet set) {
        bits = new long[set.size];
        offs = new int [set.size];
        size = 0;
    }

    private void new_block(int bnum) {
        new_block(bsearch(bnum), bnum);
    }
    private void new_block(int idx, int bnum) {
        if (size==bits.length) { // resize
```

```
            long[] nbits = new long[size*3];
            int [] noffs = new int [size*3];
            System.arraycopy(bits, 0, nbits, 0, size);
            System.arraycopy(offs, 0, noffs, 0, size);
            bits = nbits;
            offs = noffs;
        }
        CUtility.ASSERT(size<bits.length);
        insert_block(idx, bnum);
    }
    private void insert_block(int idx, int bnum) {
        CUtility.ASSERT(idx<=size);
        CUtility.ASSERT(idx==size || offs[idx]!=bnum);
        System.arraycopy(bits, idx, bits, idx+1, size-idx);
        System.arraycopy(offs, idx, offs, idx+1, size-idx);
        offs[idx]=bnum;
        bits[idx]=0; //clear them bits.
        size++;
    }
    private int bsearch(int bnum) {
        int l=0, r=size; // search interval is [l, r)
        while (l<r) {
            int p = (l+r)/2;
            if (bnum<offs[p]) r=p;
            else if (bnum>offs[p]) l=p+1;
            else return p;
        }
        CUtility.ASSERT(l==r);
        return l; // index at which the bnum *should* be, if it's not.
    }

    /**
     * Sets a bit.
     * @param bit the bit to be set
     */
    public void set(int bit) {
        int bnum = bit >> LG_BITS;
        int idx  = bsearch(bnum);
        if (idx >= size || offs[idx]!=bnum)
            new_block(idx, bnum);
        bits[idx] |= (1L << (bit & BITS_M1) );
    }

    /**
     * Clears a bit.
     * @param bit the bit to be cleared
     */
    public void clear(int bit) {
        int bnum = bit >> LG_BITS;
        int idx  = bsearch(bnum);
        if (idx >= size || offs[idx]!=bnum)
            new_block(idx, bnum);
        bits[idx] &= ~(1L << (bit & BITS_M1) );
    }

    /**
     * Clears all bits.
     */
    public void clearAll() {
        size = 0;
    }

    /**
     * Gets a bit.
     * @param bit the bit to be gotten
     */
    public boolean get(int bit) {
        int bnum = bit >> LG_BITS;
        int idx  = bsearch(bnum);
        if (idx >= size || offs[idx]!=bnum)
            return false;
        return 0 != ( bits[idx] & (1L << (bit & BITS_M1) ) );
    }

    /**
```

```
     * Logically ANDs this bit set with the specified set of bits.
     * @param set the bit set to be ANDed with
     */
    public void and(SparseBitSet set) {
        binop(this, set, AND);
    }

    /**
     * Logically ORs this bit set with the specified set of bits.
     * @param set the bit set to be ORed with
     */
    public void or(SparseBitSet set) {
        binop(this, set, OR);
    }

    /**
     * Logically XORs this bit set with the specified set of bits.
     * @param set the bit set to be XORed with
     */
    public void xor(SparseBitSet set) {
        binop(this, set, XOR);
    }

    // BINARY OPERATION MACHINERY
    private static interface BinOp {
        public long op(long a, long b);
    }
    private static final BinOp AND = new BinOp() {
        public final long op(long a, long b) { return a & b; }
    };
    private static final BinOp OR = new BinOp() {
        public final long op(long a, long b) { return a | b; }
    };
    private static final BinOp XOR = new BinOp() {
        public final long op(long a, long b) { return a ^ b; }
    };
    private static final void binop(SparseBitSet a, SparseBitSet b, BinOp op) {
        int  nsize = a.size + b.size;
        long[] nbits;
        int [] noffs;
        int a_zero, a_size;
        // be very clever and avoid allocating more memory if we can.
        if (a.bits.length < nsize) { // oh well, have to make working space.
            nbits = new long[nsize];
            noffs = new int [nsize];
            a_zero  = 0; a_size = a.size;
        } else { // reduce, reuse, recycle!
            nbits = a.bits;
            noffs = a.offs;
            a_zero = a.bits.length - a.size; a_size = a.bits.length;
            System.arraycopy(a.bits, 0, a.bits, a_zero, a.size);
            System.arraycopy(a.offs, 0, a.offs, a_zero, a.size);
        }
        // ok, crunch through and binop those sets!
        nsize = 0;
        for (int i=a_zero, j=0; i<a_size || j<b.size; ) {
            long nb; int no;
            if (i<a_size && (j>=b.size || a.offs[i] < b.offs[j])) {
                nb = op.op(a.bits[i], 0);
                no = a.offs[i];
                i++;
            } else if (j<b.size && (i>=a_size || a.offs[i] > b.offs[j])) {
                nb = op.op(0, b.bits[j]);
                no = b.offs[j];
                j++;
            } else { // equal keys; merge.
                nb = op.op(a.bits[i], b.bits[j]);
                no = a.offs[i];
                i++; j++;
            }
            if (nb!=0) {
                nbits[nsize] = nb;
                noffs[nsize] = no;
                nsize++;
            }
        }
        a.bits = nbits;
        a.offs = noffs;
        a.size = nsize;
    }

    /**
     * Gets the hashcode.
     */
    public int hashCode() {
        long h = 1234;
        for (int i=0; i<size; i++)
            h ^= bits[i] * offs[i];
        return (int)((h >> 32) ^ h);
    }

    /**
     * Calculates and returns the set's size
     */
    public int size() {
        return (size==0)?0:((1+offs[size-1]) << LG_BITS);
    }

    /**
     * Compares this object against the specified object.
     * @param obj the object to commpare with
     * @return true if the objects are the same; false otherwise.
     */
    public boolean equals(Object obj) {
        if ((obj != null) && (obj instanceof SparseBitSet))
            return equals(this, (SparseBitSet)obj);
        return false;
    }
    /**
     * Compares two SparseBitSets for equality.
     * @return true if the objects are the same; false otherwise.
     */
    public static boolean equals(SparseBitSet a, SparseBitSet b) {
        for (int i=0, j=0; i<a.size || j<b.size; ) {
            if (i<a.size && (j>=b.size || a.offs[i] < b.offs[j])) {
                if (a.bits[i++]!=0) return false;
            } else if (j<b.size && (i>=a.size || a.offs[i] > b.offs[j])) {
                if (b.bits[j++]!=0) return false;
            } else { // equal keys
                if (a.bits[i++]!=b.bits[j++]) return false;
            }
        }
        return true;
    }

    /**
     * Clones the SparseBitSet.
     */
    public Object clone() {
        try {
            SparseBitSet set = (SparseBitSet)super.clone();
            set.bits = (long[]) bits.clone();
            set.offs = (int []) offs.clone();
            return set;
        } catch (CloneNotSupportedException e) {
            // this shouldn't happen, since we are Cloneable
            throw new InternalError();
        }
    }

    /**
     * Return an <code>Enumeration</code> of <code>Integer</code>s
     * which represent set bit indices in this SparseBitSet.
     */
    public Enumeration elements() {
        return new Enumeration() {
            int idx=-1, bit=BITS;
            { advance(); }
            public boolean hasMoreElements() {
                return (idx<size);
            }
            public Object nextElement() {
                int r = bit + (offs[idx] << LG_BITS);
                advance();
                return new Integer(r);
            }
            private void advance() {
                while (idx<size) {
                    while (++bit<BITS)
                        if (0!=(bits[idx] & (1L<<bit)))
                            return;
                    idx++; bit=-1;
                }
            }
        };
    }
    /**
     * Converts the SparseBitSet to a String.
     */
    public String toString() {
        StringBuffer sb = new StringBuffer();
        sb.append('{');
        for (Enumeration e=elements(); e.hasMoreElements(); ) {
            if (sb.length() > 1) sb.append(", ");
            sb.append(e.nextElement());
        }
        sb.append('}');
        return sb.toString();
    }

    /** Check validity. */
    private boolean isValid() {
        if (bits.length!=offs.length) return false;
        if (size>bits.length) return false;
        if (size!=0 && 0<=offs[0]) return false;
        for (int i=1; i<size; i++)
            if (offs[i] < offs[i-1])
                    return false;
        return true;
    }
    /** Self-test. */
    public static void main(String[] args) {
        final int ITER = 500;
        final int RANGE= 65536;
        SparseBitSet a = new SparseBitSet();
        CUtility.ASSERT(!a.get(0) && !a.get(1));
        CUtility.ASSERT(!a.get(123329));
        a.set(0); CUtility.ASSERT(a.get(0) && !a.get(1));
        a.set(1); CUtility.ASSERT(a.get(0) && a.get(1));
        a.clearAll();
        CUtility.ASSERT(!a.get(0) && !a.get(1));
        java.util.Random r = new java.util.Random();
        java.util.Vector v = new java.util.Vector();
        for (int n=0; n<ITER; n++) {
            int rr = ((r.nextInt()>>>1) % RANGE) << 1;
            a.set(rr); v.addElement(new Integer(rr));
            // check that all the numbers are there.
            CUtility.ASSERT(a.get(rr) && !a.get(rr+1) && !a.get(rr-1));
            for (int i=0; i<v.size(); i++)
                CUtility.ASSERT(a.get(((Integer)v.elementAt(i)).intValue()));
        }
        SparseBitSet b = (SparseBitSet) a.clone();
        CUtility.ASSERT(a.equals(b) && b.equals(a));
        for (int n=0; n<ITER/2; n++) {
            int rr = (r.nextInt()>>>1) % v.size();
            int m = ((Integer)v.elementAt(rr)).intValue();
            b.clear(m); v.removeElementAt(rr);
            // check that numbers are removed properly.
            CUtility.ASSERT(!b.get(m));
        }
        CUtility.ASSERT(!a.equals(b));
        SparseBitSet c = (SparseBitSet) a.clone();
        SparseBitSet d = (SparseBitSet) a.clone();
        c.and(a);
        CUtility.ASSERT(c.equals(a) && a.equals(c));
        c.xor(a);
        CUtility.ASSERT(!c.equals(a) && c.size()==0);
        d.or(b);
        CUtility.ASSERT(d.equals(a) && !b.equals(d));
        d.and(b);
        CUtility.ASSERT(!d.equals(a) && b.equals(d));
        d.xor(a);
        CUtility.ASSERT(!d.equals(a) && !b.equals(d));
        c.or(d); c.or(b);
        CUtility.ASSERT(c.equals(a) && a.equals(c));
        c = (SparseBitSet) d.clone();
        c.and(b);
        CUtility.ASSERT(c.size()==0);
        System.out.println("Success.");
    }
}

/************************************************************************
  JLEX COPYRIGHT NOTICE, LICENSE AND DISCLAIMER.

  Copyright 1996 by Elliot Joel Berk

  Permission to use, copy, modify, and distribute this software and its
  documentation for any purpose and without fee is hereby granted,
  provided that the above copyright notice appear in all copies and that
  both the copyright notice and this permission notice and warranty
  disclaimer appear in supporting documentation, and that the name of
  Elliot Joel Berk not be used in advertising or publicity pertaining
  to distribution of the software without specific, written prior permission.

  Elliot Joel Berk disclaims all warranties with regard to this software,
  including all implied warranties of merchantability and fitness.  In no event
  shall Elliot Joel Berk be liable for any special, indirect or consequential
  damages or any damages whatsoever resulting from loss of use, data or
  profits, whether in an action of contract, negligence or other
  tortious action, arising out of or in connection with the use or
  performance of this software.
  ***********************************************************************/
// set emacs indentation
// Local Variables:
// c-basic-offset:2
// End:
```