SimpleCharStream.java

```java
/* Generated By:JavaCC: Do not edit this line. SimpleCharStream.java Version 4.0 */
package jminusminus;

/**
 * An implementation of interface CharStream, where the stream is assumed to
 * contain only ASCII characters (without unicode processing).
 */

public class SimpleCharStream
{
  public static final boolean staticFlag = false;
  int bufsize;
  int available;
  int tokenBegin;
  public int bufpos = -1;
  protected int bufline[];
  protected int bufcolumn[];

  protected int column = 0;
  protected int line = 1;

  protected boolean prevCharIsCR = false;
  protected boolean prevCharIsLF = false;

  protected java.io.Reader inputStream;

  protected char[] buffer;
  protected int maxNextCharInd = 0;
  protected int inBuf = 0;
  protected int tabSize = 8;

  protected void setTabSize(int i) { tabSize = i; }
  protected int getTabSize(int i) { return tabSize; }


  protected void ExpandBuff(boolean wrapAround)
  {
     char[] newbuffer = new char[bufsize + 2048];
     int newbufline[] = new int[bufsize + 2048];
     int newbufcolumn[] = new int[bufsize + 2048];

     try
     {
        if (wrapAround)
        {
           System.arraycopy(buffer, tokenBegin, newbuffer, 0, bufsize - tokenBegin);
           System.arraycopy(buffer, 0, newbuffer,
                                             bufsize - tokenBegin, bufpos);
           buffer = newbuffer;

           System.arraycopy(bufline, tokenBegin, newbufline, 0, bufsize - tokenBegin);
           System.arraycopy(bufline, 0, newbufline, bufsize - tokenBegin, bufpos);
           bufline = newbufline;

           System.arraycopy(bufcolumn, tokenBegin, newbufcolumn, 0, bufsize - tokenBegin);
           System.arraycopy(bufcolumn, 0, newbufcolumn, bufsize - tokenBegin, bufpos);
           bufcolumn = newbufcolumn;

           maxNextCharInd = (bufpos += (bufsize - tokenBegin));
        }
```

```
         else
         {
            System.arraycopy(buffer, tokenBegin, newbuffer, 0, bufsize - tokenBegin);
            buffer = newbuffer;

            System.arraycopy(bufline, tokenBegin, newbufline, 0, bufsize - tokenBegin);
            bufline = newbufline;

            System.arraycopy(bufcolumn, tokenBegin, newbufcolumn, 0, bufsize - tokenBegin);
            bufcolumn = newbufcolumn;

            maxNextCharInd = (bufpos -= tokenBegin);
         }
      }
      catch (Throwable t)
      {
         throw new Error(t.getMessage());
      }


      bufsize += 2048;
      available = bufsize;
      tokenBegin = 0;
   }

   protected void FillBuff() throws java.io.IOException
   {
      if (maxNextCharInd == available)
      {
         if (available == bufsize)
         {
            if (tokenBegin > 2048)
            {
               bufpos = maxNextCharInd = 0;
               available = tokenBegin;
            }
            else if (tokenBegin < 0)
               bufpos = maxNextCharInd = 0;
            else
               ExpandBuff(false);
         }
         else if (available > tokenBegin)
            available = bufsize;
         else if ((tokenBegin - available) < 2048)
            ExpandBuff(true);
         else
            available = tokenBegin;
      }

      int i;
      try {
         if ((i = inputStream.read(buffer, maxNextCharInd,
                                     available - maxNextCharInd)) == -1)
         {
            inputStream.close();
            throw new java.io.IOException();
         }
         else
            maxNextCharInd += i;
         return;
      }
      catch(java.io.IOException e) {
         --bufpos;
         backup(0);
         if (tokenBegin == -1)
            tokenBegin = bufpos;
         throw e;
      }
   }

   public char BeginToken() throws java.io.IOException
   {
      tokenBegin = -1;
      char c = readChar();
      tokenBegin = bufpos;

      return c;
   }

   protected void UpdateLineColumn(char c)
   {
      column++;

      if (prevCharIsLF)
      {
         prevCharIsLF = false;
         line += (column = 1);
      }
      else if (prevCharIsCR)
      {
         prevCharIsCR = false;
         if (c == '\n')
         {
            prevCharIsLF = true;
         }
         else
            line += (column = 1);
      }

      switch (c)
      {
         case '\r' :
            prevCharIsCR = true;
            break;
         case '\n' :
            prevCharIsLF = true;
            break;
         case '\t' :
            column--;
            column += (tabSize - (column % tabSize));
            break;
         default :
            break;
      }

      bufline[bufpos] = line;
      bufcolumn[bufpos] = column;
   }

   public char readChar() throws java.io.IOException
   {
      if (inBuf > 0)
      {
         --inBuf;

         if (++bufpos == bufsize)
            bufpos = 0;

         return buffer[bufpos];
      }

      if (++bufpos >= maxNextCharInd)
         FillBuff();

      char c = buffer[bufpos];

      UpdateLineColumn(c);
      return (c);
   }

   /**
    * @deprecated
    * @see #getEndColumn
    */

   public int getColumn() {
      return bufcolumn[bufpos];
   }

   /**
    * @deprecated
    * @see #getEndLine
    */

   public int getLine() {
      return bufline[bufpos];
   }

   public int getEndColumn() {
      return bufcolumn[bufpos];
   }

   public int getEndLine() {
      return bufline[bufpos];
   }

   public int getBeginColumn() {
      return bufcolumn[tokenBegin];
   }

   public int getBeginLine() {
      return bufline[tokenBegin];
   }

   public void backup(int amount) {

     inBuf += amount;
     if ((bufpos -= amount) < 0)
        bufpos += bufsize;
   }

   public SimpleCharStream(java.io.Reader dstream, int startline,
   int startcolumn, int buffersize)
   {
     inputStream = dstream;
     line = startline;
     column = startcolumn - 1;

     available = bufsize = buffersize;
     buffer = new char[buffersize];
     bufline = new int[buffersize];
     bufcolumn = new int[buffersize];
   }

   public SimpleCharStream(java.io.Reader dstream, int startline,
                           int startcolumn)
   {
      this(dstream, startline, startcolumn, 4096);
   }

   public SimpleCharStream(java.io.Reader dstream)
   {
      this(dstream, 1, 1, 4096);
   }
```

```java
  public void ReInit(java.io.Reader dstream, int startline,
  int startcolumn, int buffersize)
  {
    inputStream = dstream;
    line = startline;
    column = startcolumn - 1;

    if (buffer == null || buffersize != buffer.length)
    {
      available = bufsize = buffersize;
      buffer = new char[buffersize];
      bufline = new int[buffersize];
      bufcolumn = new int[buffersize];
    }
    prevCharIsLF = prevCharIsCR = false;
    tokenBegin = inBuf = maxNextCharInd = 0;
    bufpos = -1;
  }

  public void ReInit(java.io.Reader dstream, int startline,
                     int startcolumn)
  {
     ReInit(dstream, startline, startcolumn, 4096);
  }

  public void ReInit(java.io.Reader dstream)
  {
     ReInit(dstream, 1, 1, 4096);
  }
  public SimpleCharStream(java.io.InputStream dstream, String encoding, int startline,
  int startcolumn, int buffersize) throws java.io.UnsupportedEncodingException
  {
     this(encoding == null ? new java.io.InputStreamReader(dstream) : new java.io.InputStreamReader(dstream, encoding), startline, startcolumn, buffersize);
  }

  public SimpleCharStream(java.io.InputStream dstream, int startline,
  int startcolumn, int buffersize)
  {
     this(new java.io.InputStreamReader(dstream), startline, startcolumn, buffersize);
  }

  public SimpleCharStream(java.io.InputStream dstream, String encoding, int startline,
                          int startcolumn) throws java.io.UnsupportedEncodingException
  {
     this(dstream, encoding, startline, startcolumn, 4096);
  }

  public SimpleCharStream(java.io.InputStream dstream, int startline,
                          int startcolumn)
  {
     this(dstream, startline, startcolumn, 4096);
  }

  public SimpleCharStream(java.io.InputStream dstream, String encoding) throws java.io.UnsupportedEncodingException
  {
     this(dstream, encoding, 1, 1, 4096);
  }

  public SimpleCharStream(java.io.InputStream dstream)
  {
     this(dstream, 1, 1, 4096);
  }
```

```
   public void ReInit(java.io.InputStream dstream, String encoding, int startline,
                           int startcolumn, int buffersize) throws
java.io.UnsupportedEncodingException
   {
      ReInit(encoding == null ? new java.io.InputStreamReader(dstream) : new
java.io.InputStreamReader(dstream, encoding), startline, startcolumn, buffersize);
   }

   public void ReInit(java.io.InputStream dstream, int startline,
                           int startcolumn, int buffersize)
   {
      ReInit(new java.io.InputStreamReader(dstream), startline, startcolumn,
buffersize);
   }

   public void ReInit(java.io.InputStream dstream, String encoding) throws
java.io.UnsupportedEncodingException
   {
      ReInit(dstream, encoding, 1, 1, 4096);
   }

   public void ReInit(java.io.InputStream dstream)
   {
      ReInit(dstream, 1, 1, 4096);
   }
   public void ReInit(java.io.InputStream dstream, String encoding, int startline,
                      int startcolumn) throws java.io.UnsupportedEncodingException
   {
      ReInit(dstream, encoding, startline, startcolumn, 4096);
   }
   public void ReInit(java.io.InputStream dstream, int startline,
                      int startcolumn)
   {
      ReInit(dstream, startline, startcolumn, 4096);
   }
   public String GetImage()
   {
      if (bufpos >= tokenBegin)
         return new String(buffer, tokenBegin, bufpos - tokenBegin + 1);
      else
         return new String(buffer, tokenBegin, bufsize - tokenBegin) +
                               new String(buffer, 0, bufpos + 1);
   }

   public char[] GetSuffix(int len)
   {
      char[] ret = new char[len];

      if ((bufpos + 1) >= len)
         System.arraycopy(buffer, bufpos - len + 1, ret, 0, len);
      else
      {
         System.arraycopy(buffer, bufsize - (len - bufpos - 1), ret, 0,
                                                           len - bufpos - 1);
         System.arraycopy(buffer, 0, ret, len - bufpos - 1, bufpos + 1);
      }

      return ret;
   }

   public void Done()
   {
      buffer = null;
      bufline = null;
      bufcolumn = null;
   }

   /**
    * Method to adjust line and column numbers for the start of a token.
    */
   public void adjustBeginLineColumn(int newLine, int newCol)
   {
      int start = tokenBegin;
      int len;

      if (bufpos >= tokenBegin)
      {
         len = bufpos - tokenBegin + inBuf + 1;
      }
      else
      {
         len = bufsize - tokenBegin + bufpos + 1 + inBuf;
      }

      int i = 0, j = 0, k = 0;
      int nextColDiff = 0, columnDiff = 0;

      while (i < len &&
             bufline[j = start % bufsize] == bufline[k = ++start % bufsize])
      {
         bufline[j] = newLine;
         nextColDiff = columnDiff + bufcolumn[k] - bufcolumn[j];
         bufcolumn[j] = newCol + columnDiff;
         columnDiff = nextColDiff;
         i++;
      }

      if (i < len)
      {
         bufline[j] = newLine++;
         bufcolumn[j] = newCol + columnDiff;

         while (i++ < len)
         {
            if (bufline[j = start % bufsize] != bufline[++start % bufsize])
               bufline[j] = newLine++;
            else
               bufline[j] = newLine;
         }
      }

      line = bufline[j];
      column = bufcolumn[j];
   }

 }
```