```
/*
 * 作成日: 2004/04/01
 *
 * Copyright (c) 2004 重村哲至
 * All rights reserved.
 */
//package jp.ac.tokuyama.pico.tec6;
import java.io.*;

/**
 * @author sigemura
 * アセンブラ本体部分
 */
class Asm {
  // アセンブルリストの整形に関係のある定数
  static final private int labelMax = 8; // ラベル欄の長さ
  static final private int mnemoMax = 8; // オペレーション欄の長さ
  static final private int operMax = 15; // オペランド欄の長さ
  private String nextSrc; // nextTok のソース
  private Token nextTok; // 着目しているトークン
  private TokenType nextType; // nextTok の型
  private Object nextValue; // nextTok の値(トークンが数なら数値等)

  private Lexer lexer; // 字句解析器
  private byte[] size = new byte[1]; // プログラムサイズ(パス１終了時に決定)
  private byte[] start = new byte[1]; // プログラムアドレス(パス１終了時に決定)

  private SymTbl symTbl = new SymTbl(); // 記号表
  private String fileName;
  private BufferedOutputStream binFile;

  // コンストラクタ
  Asm(String f) {
    fileName = f;
  }

  // 機械語の書き込み
  private void output(byte[] obj) {
    try {
      binFile.write(obj, 0, obj.length);
    } catch (IOException e) {
      System.err.println("機械語ファイルの書き込みエラー");
    }
  }

  // エラー発生時にメッセージを表示して終了する。
  private void errorExit(int ln, Err err) {
    setNext();
    System.err.println("＊＊＊ エラー[" + err.getMsg() + "] ＊＊＊");
    System.err.println("エラー発生場所 : ファイル [" + fileName + "] の " + ln + "行
で");
    System.err.println("エラー行の内容 : [" + lexer.getLine() + "]");
    System.err.println("エラートークン : [" + nextTok.getSrc() + "]");
    System.exit(1);
  }

  // 新しいトークンを取り込む
  private void getNext(boolean rsvWrd) throws AsmException {
    lexer.setNextTok(rsvWrd);
    setNext();
  }

  // パス１(シンボルテーブルを完成しプログラムサイズを決める。)
  void pass1(Lexer p) {
    lexer = p;
    int lc = 0; // ロケーションカウンタ
```



```
    int ln = 0; // 行番号

    try {
      getNext(false);

      // １行処理する毎に以下のループを１回実行する。
      while (nextType != TokenType.EOF) {
        String label = null;
        boolean err = false;
        int len = 0; // 命令長
        ln++;

        // ラベルの処理
        if (nextType == TokenType.NAM) {
          label = (String) nextValue; // ラベルあり
          getNext(true);
        } else if (nextType == TokenType.SPC) {
          getNext(true); // 空白あり
        } else if (nextType != TokenType.COM && nextType != TokenType.EOL) {
          // 行頭がラベルでも空白でもコメントでもEOLでもない。
          throw new AsmException(Err.BAD_Lab);
        }

        // 命令の処理
        InstTblEntry e;
        if (nextType == TokenType.RSV
          && (e = InstTbl.getEntry((RsvWords) nextValue)) != null) {
          // 命令のどれか

          int v = lc;
          len = e.getLength(); // 機械語命令は命令長が決まる。
          if (len == 0) { // ゼロなら疑似命令
            if (nextValue == RsvWords.EQU) {
              getNext(false);
              v = SyntaxAnalyzer.expr(lexer, symTbl, false);
            } else if (nextValue == RsvWords.ORG) {
              getNext(false);
              v = SyntaxAnalyzer.expr(lexer, symTbl, false);
              if (v < lc) {
                throw new AsmException(Err.BAD_Org);
              }
              len = v - lc;
              if (lc == 0) start[0] = (byte) v;
            } else if (nextValue == RsvWords.DS) {
              getNext(false);
              len = SyntaxAnalyzer.expr(lexer, symTbl, false);
            } else if (nextValue == RsvWords.DC) {
              getNext(false);
              len = SyntaxAnalyzer.dcAnalyzer((byte) 0, lexer, symTbl, true).length;
            } else { // 機械語命令
              getNext(false);
            }
            setNext();
          }

          // ラベルがあるなら登録する。
          if (label != null) {
            symTbl.insert(label, new Integer(v));
          }
        } else if (nextType == TokenType.EOL || nextType == TokenType.COM) {
          // ラベルがあるなら登録する。
          if (label != null) {
            symTbl.insert(label, new Integer(lc));
          }
        } else {
          throw new AsmException(Err.UND_Op); // 未知のニーモニック
```

```
      }

      // 次の行に進む。
      while (nextType != TokenType.EOL && nextType != TokenType.EOF) {
        getNext(false);
      }
      getNext(false);

      //    ロケーションカウンタを進める。
      lc += len;
    }
  } catch (AsmException exc) {
    errorExit(ln, exc.getErr());
  }

  //System.out.println("\n" + symTbl.toString());
  size[0] = (byte) (lc - start[0]);
}

// パス２(アセンブルリストと機械語プログラムを出力する。)
void pass2(Lexer p, ListFormatter lf, BufferedOutputStream os) {
  int lc = 0; // ロケーションカウンタ
  int ln = 0; // 行番号
  lexer = p;
  binFile = os;

  lexer.setListOutput(true); // アセンブルリスト整形モード

  // プログラムの開始アドレスとサイズの出力
  output(start);
  output(size);

  try {
    getNext(false);

    // 一行毎に以下のループを１回実行する。
    while (nextType != TokenType.EOF) {
      byte[] bin = null;
      int len = 0;
      byte adr = (byte) lc;
      ln++;

      // ラベルの処理
      if (nextType == TokenType.NAM || nextType == TokenType.SPC) {
        getNext(true);
      }

      InstTblEntry e; // 命令表のエントリー

      if (nextType == TokenType.RSV
        && (e = InstTbl.getEntry((RsvWords) nextValue)) != null) {
        // 命令のどれか
        // リスト行のカーソルをオペレーション欄に進める。
        lexer.setListCur(labelMax);

        // 命令の長さ
        len = e.getLength();
        if (len == 0) {
          // 長さゼロは疑似命令
          if (nextValue == RsvWords.EQU) { // EQU 命令
            getNext(false);
            // オペランド欄に移動
            lexer.setListCur(labelMax + mnemoMax);
            adr = (byte) SyntaxAnalyzer.expr(lexer, symTbl, false);
            setNext();
          } else if (nextValue == RsvWords.ORG) {
```



```
            getNext(false);
            // オペランド欄に移動
            lexer.setListCur(labelMax + mnemoMax);
            int v = SyntaxAnalyzer.expr(lexer, symTbl, false);
            // プログラム先頭のORG以外ならオブジェクトを出力する。
            if (lc != 0) {
              output(new byte[v - lc]);
            }
            lc = v;
            len = 0;
            adr = (byte) v;
            setNext();
          } else if (nextValue == RsvWords.DS) { // DS 命令
            getNext(false);
            // オペランド欄に移動
            lexer.setListCur(labelMax + mnemoMax);

            // オペランドを解析し0で初期化されたデータを作る。
            len = SyntaxAnalyzer.expr(lexer, symTbl, false);
            bin = new byte[len];

            // オブジェクトの出力
            output(bin);

            setNext();
          } else if (nextValue == RsvWords.DC) {
            getNext(false);
            // オペランド欄に移動
            lexer.setListCur(labelMax + mnemoMax);

            // オペランドを解析しデータを作る。
            bin = SyntaxAnalyzer.dcAnalyzer((byte) 0, lexer, symTbl, false);
            len = bin.length;

            // オブジェクトの出力
            output(bin);

            setNext();
          }
        } else { // 機械語命令
          SyntaxAnalyzer sa = e.getAnalyzer();

          // M1はNO命令等、M6はジャンプ命令
          getNext(sa != SyntaxAnalyzer.M1 && sa != SyntaxAnalyzer.M6);

          // オペランド欄に移動
          lexer.setListCur(labelMax + mnemoMax);

          // オペランドを解析し機械語を作る。
          bin = sa.analyzer(e.getCode(), lexer, symTbl);
          len = bin.length;

          // オブジェクトの出力
          output(bin);

          setNext();
        }
        // コメント欄に移動
        lexer.setListCur(labelMax + mnemoMax + operMax);
      }

      // 正しく構文解析が終了したか？
      if (nextType != TokenType.COM
        && nextType != TokenType.EOL
        && nextType != TokenType.EOF) {
        throw new AsmException(Err.BAD_Opr); // 行の後部が解析できない。
```

```
        }

        // 行末まで進む。
        while (nextType != TokenType.EOL && nextType != TokenType.EOF) {
          getNext(false);
        }

        // リスト出力
        lf.output(adr, bin, lexer.getListLine(), len);

        // 次の行に進む。
        getNext(false);

        // ロケーションカウンタを進める。
        lc += len;
      }
    } catch (AsmException exc) {
      errorExit(ln, exc.getErr());
    }
  }

  // 着目しているトークンを上の変数にセットする。
  private void setNext() {
    nextTok = lexer.getNextTok();
    nextType = nextTok.getType();
    nextSrc = nextTok.getSrc();
    nextValue = nextTok.getValue();
  }
}
```