```
import java.io.*;

%%

%public
%class NanoMorphoLexer
%unicode
%byaccj
%line
%column

%{


private NanoMorphoParser yyparser;

public NanoMorphoLexer(Reader r, NanoMorphoParser a)
{
      this(r);
      yyparser = a;
}


public int getLine()
{
      return yyline+1;
}

public int getColumn()
{
      return yycolumn+1;
}



%}

  /* Reglulegar skilgreiningar */

  /* Regular definitions */

_DIGIT=[0-9]
_FLOAT={_DIGIT}+\.{_DIGIT}+([eE][+-]?{_DIGIT}+)?
_INT={_DIGIT}+
_STRING=\"([^\"\\]|\\b|\\t|\\n|\\f|\\r|\\\"|\\\'|\\\\|(\\[0-3][0-7][0-
7])|\\[0-7][0-7]|\\[0-7])*\"
_CHAR=\'([^\'\\]|\\b|\\t|\\n|\\f|\\r|\\\"|\\\'|\\\\|(\\[0-3][0-7][0-
7])|(\\[0-7][0-7])|(\\[0-7]))\'
_DELIM=[(){},;=]
_NAME=([:letter:]|{_DIGIT})+
_OPNAME=[\+\-*/!%&=><\:\^\~&|?]+

%%

  /* Lesgreiningarreglur */

{_DELIM} {
      yyparser.yylval = new NanoMorphoParserVal(yytext());
      return yycharat(0);
}
```

```
{_STRING} | {_FLOAT} | {_CHAR} | {_INT} | null | true | false {
      yyparser.yylval = new NanoMorphoParserVal(yytext());
      return NanoMorphoParser.LITERAL;
}

"if" {
      yyparser.yylval = new NanoMorphoParserVal(yytext());
      return NanoMorphoParser.IF;
}

"else" {
      yyparser.yylval = new NanoMorphoParserVal(yytext());
      return NanoMorphoParser.ELSE;
}

"elsif" {
      yyparser.yylval = new NanoMorphoParserVal(yytext());
      return NanoMorphoParser.ELSIF;
}

"while" {
      yyparser.yylval = new NanoMorphoParserVal(yytext());
      return NanoMorphoParser.WHILE;
}

"var" {
      yyparser.yylval = new NanoMorphoParserVal(yytext());
      return NanoMorphoParser.VAR;
}

"return" {
      yyparser.yylval = new NanoMorphoParserVal(yytext());
      return NanoMorphoParser.RETURN;
}

{_NAME} {
      yyparser.yylval = new NanoMorphoParserVal(yytext());
      return NanoMorphoParser.NAME;
}

{_OPNAME} {
      yyparser.yylval = new NanoMorphoParserVal(yytext());
      return NanoMorphoParser.OPNAME;
}

";;;".*$ {
}

[ \t\r\n\f] {
}

. {
      yyparser.yylval = new NanoMorphoParserVal(yytext());
      return NanoMorphoParser.ERROR;
}


%{
        import java.io.*;
```

```
        import java.util.*;
%}
%token <sval> NAME, LITERAL, OPNAME, ERROR
%type <obj> program function exprs expr binopexpr smallexpr optexpr
nonemptyoptexpr elseifexpr elseexpr body
%type <ival> decls decl parlist optparlist
%token IF, ELSE, ELSIF, VAR, WHILE, RETURN
%%

start
  :   program    {
generateProgram(name,((Vector<Object>)($1)).toArray()); }
  ;

program
  : program function  { ((Vector<Object>)($1)).add($2); $$=$1; }
  | function  { $$ = new Vector<Object>();
((Vector<Object>)($$)).add($1); }
  ;

function
  : NAME  {
              varCount = 0;
              varTable = new HashMap<String,Integer>();
          }
    '(' optparlist ')' '{' decls exprs '}'
          {
            $$ = new Object[]{$1, $4, $7,
((Vector<Object>)($8)).toArray()};
          }
  ;

parlist
  : NAME { addVar($1);$$ = $$+1;}
  | parlist ',' NAME { addVar($3);$$ = 1+$1;}
  ;

optparlist
  : {$$ = 0;}
  | parlist { $$ = $1;}
  ;

decls
  : {$$ = 0;}
  | decls decl ';' {$$ = $1+$2;}
  ;

decl
  :   decl ',' NAME          { addVar($3); $$=$1+1; }
  |   VAR NAME {addVar($2); $$=1;}
  ;

exprs
  : expr ';' {$$ = new Vector<Object>(); ((Vector<Object>)($$)).add($1);}
  | exprs expr ';' { ((Vector<Object>)($1)).add($2); $$=$1;}
  ;

expr
  : RETURN expr {$$ = new Object[]{"RETURN", $2};}
  | NAME '=' expr {$$ = new Object[]{"STORE", findVar($1), $3};}
```

```
  | binopexpr {$$ = $1;}
  ;

binopexpr
  : smallexpr {$$=$1;}
  | binopexpr OPNAME smallexpr {$$ = new Object[]{"CALL", $2, new
Object[]{$1, $3}};}
  ;

smallexpr
  : NAME {$$ = new Object[]{"NAME", findVar($1)};}
  | NAME '(' optexpr ')' {$$ = new Object[]{"CALL", $1,
((Vector<Object>)($3)).toArray()};}
  | OPNAME smallexpr {$$ = new Object[]{"CALL", $1, new Object[]{$2}};}
  | LITERAL { $$ = new Object[]{"LITERAL", $1};}
  | '(' expr ')' {$$ = $2;}
  | IF expr body elseifexpr elseexpr {$$ = new Object[]{"IF", $2,
((Vector<Object>)($3)).toArray(), $4, $5};}
  | WHILE expr body {$$ = new Object[]{"WHILE", $2,
((Vector<Object>)($3)).toArray()};}
  ;

optexpr
  : {$$ = new Vector<Object>();}
  | optexpr expr {((Vector<Object>)($1)).add($2); $$=$1;}
  ;

elseifexpr
  : {$$ = null;}
  | ELSIF expr body elseifexpr {$$ = new Object[]{"ELSIF", $2,
((Vector<Object>)($3)).toArray(), $4};}
  ;

elseexpr
  : ELSE body  {$$ = new Object[]{"ELSE",
((Vector<Object>)($2)).toArray()};}
  | {$$ = null;}
  ;

body
  : '{' exprs '}' {$$ = $2;}
  ;

%%

private static int varCount;
private static HashMap<String,Integer> varTable;
private NanoMorphoLexer lexer;
private static String name;

private void addVar( String name )
{
  if( varTable.get(name) != null )
      throw new Error("Variable "+name+" already exists, near line
"+lexer.getLine());
  varTable.put(name,varCount++);
}

private int findVar( String name )
{
```

```
      Integer res = varTable.get(name);
      if( res == null )
            throw new Error("Variable "+name+" does not exist, near line
"+lexer.getLine());
      return res;
}

public NanoMorphoParser(Reader r) {
      lexer = new NanoMorphoLexer(r,this);
}

private int yylex()
{
      int yyl_return = -1;
      try
      {
            yylval = null;
            yyl_return = lexer.yylex();
            if( yylval==null )
                  yylval = new
NanoMorphoParserVal(NanoMorphoParser.yyname[yyl_return]);
      }
      catch (IOException e)
      {
            System.err.println("IO error: "+e);
      }
      return yyl_return;
}

public void yyerror( String error )
{
      System.err.println("Error: "+error);
      System.err.println("Line: "+lexer.getLine());
      System.err.println("Column: "+lexer.getColumn());
      System.exit(1);
}

public static void main( String[] args) throws IOException,
FileNotFoundException
{
      NanoMorphoParser par = new NanoMorphoParser(new
FileReader(args[0]));
  name = args[0].substring(0,args[0].lastIndexOf('.'));
      par.yyparse();
}

static void generateProgram( String filename, Object[] funs )
{
  String programname = filename.substring(0,filename.indexOf('.'));
  System.out.println("\""+programname+".mexe\" = main in");
  System.out.println("!");
  System.out.println("{{");
  for( Object f: funs )
  {
      generateFunction((Object[])f);
  }
  System.out.println("}}");
  System.out.println("*");
  System.out.println("BASIS;");
}
```

```
static void generateFunction( Object[] fun )
{
  //fun = {fname, argcount, varcount, res[]};
  String fname = (String)fun[0];

  int argCount = (int)fun[1];
  int varCount = (int)fun[2];
  System.out.println("#\""+fname+"[f"+argCount+"]\" =");

  System.out.println("[");
  for(int k = 0; k<varCount;k++){
    System.out.println("(MakeVal null)");
    System.out.println("(Push)");
  }

  for(Object e:(Object[])fun[3]){
    generateExpr((Object[])e);
  }
  System.out.println("(Return)");
  System.out.println("];");
}

static int nextLab = 0;


static void generateExpr( Object[] e )
{
  switch((String)e[0]){
    case "NAME":
      System.out.println("(Fetch "+e[1]+")");
      return;
    case "LITERAL":
      System.out.println("(MakeVal "+(String)e[1]+")");
      return;
    case "RETURN":
      generateExpr((Object[])e[1]);
      System.out.println("(Return)");
      return;
    case "OPNAME":
      generateExpr((Object[])e[2]);
      System.out.println("(Call \""+e[1]+"[f1]\" "+1+")");
      return;
    case "IF":
      //e = {"IF" expr body elseifexpr elseexpr}
      int labElse = nextLab++;
      int labEnd = nextLab++;
      generateExpr((Object[])e[1]);
      System.out.println("(GoFalse _"+labElse+")");
      generateBody((Object[])e[2]);
      System.out.println("(Go _"+labEnd+")");
      if(e[3]!=null){
        Object[] argu = (Object[])e[3];
        for(int i = 0; i<argu.length-1;i+=3){
          System.out.println("_"+labElse+":");
          generateExpr((Object[])argu[i+1]);
          labElse = nextLab++;
          System.out.println("(GoFalse _"+labElse+")");
          generateBody((Object[])argu[i+2]);
          System.out.println("(Go _"+labEnd+")");
```

```
        }
      }
        System.out.println("_"+labElse+":");
        if(e[4]!=null){
          Object[] bod = (Object[])e[4];
          generateBody((Object[])bod[1]);
        }
        System.out.println("_"+labEnd+":");
        return;
      case "WHILE":
        int labStart = nextLab++;
        int labQuit = nextLab++;
        System.out.println("_"+labStart+":");
        generateExpr((Object[])e[1]);
        System.out.println("(GoFalse _"+labQuit+")");
        generateBody((Object[])e[2]);
        System.out.println("(Go _"+labStart+")");
        System.out.println("_"+labQuit+":");
        return;
      case "CALL":
        //e = {"CALL", name, [expr,...,expr]}
        if(e[2]==null){
            System.out.println("(Call #\""+e[1]+"[f"+0+"]\" "+0+")");
            return;
        }
        Object[] args = (Object[])e[2];
        if( args.length!=0){
          generateExpr((Object[])args[0]);
        }
        for (int i = 1; i<args.length; i++){
          System.out.println("(Push)");
          generateExpr((Object[])args[i]);
        }
        System.out.println("(Call #\""+e[1]+"[f"+args.length+"]\"
"+args.length+")");
        return;
      case "STORE":
        generateExpr((Object[])e[2]);
        System.out.println("(Store "+e[1]+")");
        return;
  }
}

static void generateBody( Object[] bod )
{
     for(int i=0; i<bod.length; i++) {
          generateExpr((Object[])bod[i]);
  }
}
```