Þýðendur
Þýðandi og milliÞula fyrir NanoMorpho með endurkvæmri ofanferð

Pétur Daníel Ámundason

March 8, 2018

```
import java.util.Vector;

import javafx.beans.binding.ObjectExpression;

import java.util.HashMap;

public class NanoMorphoParser
{
    final static int ERROR = -1;
    final static int IF = 1001;
    final static int ELSE = 1002;
    final static int ELSIF = 1003;
    final static int WHILE = 1004;
    final static int VAR = 1005;
    final static int RETURN = 1006;
    final static int NAME = 1007;
    final static int OPNAME = 1008;
    final static int LITERAL = 1009;

    static String advance() throws Exception
    {
        return NanoMorphoLexer.advance();
    }

    static String over( int tok ) throws Exception
    {
        return NanoMorphoLexer.over(tok);
    }

    static String over( char tok ) throws Exception
    {
        return NanoMorphoLexer.over(tok);
    }

    static int getToken1()
    {
        return NanoMorphoLexer.getToken1();
    }

    static public void main( String[] args ) throws Exception
    {
        try
        {
            NanoMorphoLexer.startLexer(args[0]);
            program(args[0].substring(0,args[0].lastIndexOf('.')));
        }
        catch( Throwable e )
```

```
    {

        System.out.println("villa_lina_" + NanoMorphoLexer.getLine());
        System.out.println(e.getMessage());
    }
}

static String getLexeme2() throws Exception{
    return NanoMorphoLexer.getLexeme2();
}

static String getLexeme() throws Exception{
    return NanoMorphoLexer.getLexeme();
}

static void program(String proName) throws Exception
{
    Vector<Object[]> res = new Vector<Object[]>();
    while( getToken1()!=0 ) res.add(function());
    generateProgram(proName, res.toArray());
}

static HashMap<String,Integer> varTable = new HashMap<String,Integer>();
static int hashCount = 0;
static int acount = 0;
static int vcount = 0;

static Object[] function() throws Exception
{
    acount = 0;
    vcount = 0;
    hashCount = 0;
    varTable.clear();
    Vector<Object[]> res = new Vector<Object[]>();
    String funName = getLexeme();
    over(NAME); over('(');
    if( getToken1()!=')' )
    {
        for(;;)
        {
            acount++;
            varTable.put(getLexeme(),hashCount++);
            over(NAME);
            if( getToken1()!=',' ) break;
            over(',');
        }
    }
    over(')'); over('{');
    while( getToken1()==VAR )
    {
        decl(); over(';');
    }
    while( getToken1()!='}' )
    {
        res.add(expr()); over(';');
    }
    over('}');
    return new Object[]{funName,acount,vcount,res.toArray()};
}

static void decl() throws Exception
{
    over(VAR);
    for(;;)
    {
        varTable.put(getLexeme(),hashCount++);
        vcount++;
        over(NAME);
```

```
            if( getToken1()!=',' ) break;
            over(',');
        }
    }
    static Object[] expr() throws Exception
    {
        Object[] res = new Object[]{};
        if( getToken1()==RETURN )
        {
            over(RETURN);
            res =  new Object[]{RETURN,expr()};
            return res;
        }
        else if( getToken1()==NAME && NanoMorphoLexer.getToken2()=='=' )
        {
            int i = varTable.get(getLexeme());
            over(NAME); over('=');
            res = new Object[]{SNAME,i,expr()};
            return res;
        }
        else
        {
            return binopexpr();
        }
    }

     static int CALL = 10;
    static Object[]  binopexpr() throws Exception
    {
        Object[] e = smallexpr();
        while( getToken1()==OPNAME )
        {
            String op = getLexeme();
            over(OPNAME);
            Object[] r = smallexpr();
            e = new Object[]{CALL,op,2,new Object[]{e,r}};
        }
        return e;
    }
    static int FCALL = 3012;
    static int SNAME = 1123;
    static int FNAME = 2299;
    static Object[] smallexpr() throws Exception
    {
        Object[] res = new Object[]{};
        switch( getToken1() )
        {
        case NAME:
            String name = getLexeme();
            int pos = varTable.get(name);
            int arg = 0;
            Vector<Object[]> st = new Vector<Object[]>();
            over(NAME);
            if( getToken1()=='(' )
            {
                over('(');
                if( getToken1()!=')' )
                {
                    for(;;)
                    {
                        arg++;
                        st.add(expr());
                        if( getToken1()==')' ) break;
                        over(',');
                    }
                }
                over(')');
                res= new Object[]{FCALL,name,arg,st.toArray()};
```

```
            return res;
        } else {
            res = new Object[]{FNAME, pos};
            return res;
        }
    case WHILE:
        over(WHILE); expr(); body(); return res;
    case IF:
        Vector<Object[]> bo = new Vector<Object[]>();
        Object[] e, b, s = new Object[]{};
        over(IF); e = expr(); b = body();
        while( getToken1()==ELSIF )
        {
            over(ELSIF);
            s = new Object[]{IF1,expr(),body()};
        }
        if( getToken1()==ELSE )
        {
            over(ELSE); body();
        }
        res = new Object[]{IF,e,b,s};
        return res;
    case LITERAL:
        res = new Object[]{LITERAL, getLexeme()};
        over(LITERAL); return res;
    case OPNAME:
        over(OPNAME); smallexpr(); return res;
    case '(':
        over('('); expr(); over(')'); return res;
    default:
        NanoMorphoLexer.expected("expression");
    }
    return res;
}

static int BODY =1235466;
static int IF1 = 12355313;

static Object[] body() throws Exception
{
    over('{');
    Vector<Object[]> st = new Vector<Object[]>();
    while( getToken1()!='}' )
    {
        st.add(expr());
        over(';');
    }
    over('}');
    return new Object[]{WHILE, st.toArray()};
}

static void generateFunction( Object[] f )
    {
            // f = {fname,acount,vcount,expr[]}
    acount = (Integer)f[1];
    vcount = (Integer)f[2];
            String fname = (String)f[0];
    int count = (Integer)f[1];
    Object[] args = (Object[])f[3];
            emit("#\""+fname+"[f"+count+"]\" =");
    emit("[");
    emit("(MakeVal null)");
    while(vcount != 0){
        emit("(Push)"); vcount--;
    }
    while(acount != 0){
        emit("(Push)"); acount--;
    }
```

```
    for(int i = 0; i < args.length; i++) generateExpr((Object[])args[i]);
    emit("];");
}

static void generateProgram( String name, Object[] p )
    {
            emit("\""+name+".mexe\" = main in");
            emit("!{{");
            for( int i=0 ; i!=p.length ; i++ ) generateFunction((Object[])p[i]);
            emit("}}*BASIS;");
    }

static void generateExpr(Object[] e){

    Object[] res;
    if (e == null) {
        return;
    }
    if((int)e[0] == NAME){
        // {NAME,i,expr()}
        emit("(Fetch "+e[1]+")");
        generateExpr((Object[])e[2]);

    }
    if((int)e[0] == SNAME){
        // {SNAME,i,expr()}
        generateExpr((Object[])e[2]);
        emit("(Store "+e[1]+")");
    }
    if((int)e[0] == FNAME){
        // {NAME,i,expr()}
        emit("(Fetch "+e[1]+")");
    }

    if((int)e[0] == LITERAL){
        emit("(MakeVal "+(String)e[1]+")");
        emit("(Push)");
    }
    if ((int)e[0] == RETURN) {
        res = (Object[])e[1];
        generateExpr((Object[])e[1]);
        emit("(CallR #\""+"writeln"+"[f1]\" 1)");
        // emit("(Return)");

    }
    // e = new Object[]{CALL,op,2,new Object[]{e,r}};
    if((int)e[0] == CALL){
        res = (Object[])e[3];
        generateExpr((Object[])res[0]);
        generateExpr((Object[])res[1]);
        emit("(Call #\""+e[1]+"[f"+e[2]+"]\" "+e[2]+")");
    }
    if((int)e[0] == FCALL){
        res = (Object[])e[3];
        for(int i = 0; i < res.length; i++) generateExpr((Object[])res[i]);
        emit("(Call #\""+e[1]+"[f"+e[2]+"]\" "+e[2]+")");
    }

    /**
    s = new Object[]{IF1,expr(),body()};
    res = new Object[]{IF,e,b,s};
     */
    if((int)e[0] == IF){

    }
    if((int)e[0] == IF1){

    }
```

```
    }

    static void emit(String s){
        System.out.println(s);
    }
}
```

jflex kóðinn

```
/**
java -jar JFlex-1.6.0.jar nanolexer.jflex
javac NanoLexer.java
java NanoLexer inntaksskra > uttaksskra
make test
*/

import java.io.*;

%%

%public
%class NanoLexer
%unicode
%byaccj

%{

// Skilgreiningar a tokum (tokens):
final static int ERROR = -1;
final static int IF = 1001;
final static int DEFINE = 1002;
final static int NAME = 1003;
final static int LITERAL = 1004;
final static int WHILE = 1005;
final static int VAR = 1006;
final static int ELSIF = 1007;
final static int RETURN = 1008;
final static int ELSE = 1009;
final static int OPNAME = 1010;


// Breyta sem mun innihalda les (lexeme):
public static String lexeme;

public static void main( String[] args ) throws Exception
{
NanoLexer lexer = new NanoLexer(new FileReader(args[0]));
int token = lexer.yylex();
while( token!=0 )
{
System.out.println(""+token+": \'"+lexeme+"\'");
token = lexer.yylex();
}
}

%}

/* 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} {
lexeme = yytext();
return yycharat(0);
}

{_OPNAME} {
lexeme = yytext();
return OPNAME;
}

{_STRING} | {_FLOAT} | {_CHAR} | {_INT} | null | true | false {
lexeme = yytext();
return LITERAL;
}

"return" {
lexeme = yytext();
return RETURN;
}

"else" {
lexeme = yytext();
return ELSE;
}

"elsif" {
lexeme = yytext();
return ELSIF;
}

"while" {
lexeme = yytext();
return WHILE;
}

"if" {
lexeme = yytext();
return IF;
}

"define" {
lexeme = yytext();
return DEFINE;
}

"var" {
lexeme = yytext();
return VAR;
}

{_NAME} {
lexeme = yytext();
return NAME;
}

";;;".*$ {
}
```

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

. {
lexeme = yytext();
return ERROR;
}
```