

Þýðendur
Þáttari fyrir NanoMorpho með endurkvæmri ofanferð
Pétur Daníel Ámundason

March 15, 2018

nanoMorpho þáttarinn

```
import java.io.*;

public class nanoMorpho{

    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;

    public static NanoLexer lexer;
    public static int token;
    public static int nextToken;
    public static String lexem;
    public static String nextLexem;

    public static void init()
        throws Exception
    {
        lexer = new NanoLexer(new FileReader("test.s"));
        token = lexer.yylex();
        lexem = NanoLexer.getlexeme();
        nextToken = lexer.yylex();
        nextLexem = NanoLexer.getlexeme();
    }

    public static void advance(){
        try{
            if(token != 0){
                token = nextToken;
                lexem = nextLexem;
                nextToken = lexer.yylex();
                nextLexem = NanoLexer.getlexeme();
            }
        } catch(Exception e) {
            throw new Error(e);
        }
    }

    public static void println(String message){
        System.out.println(message);
    }
}
```

```

}

public static void over(String s){
    if(s.equals(lexem)){
        advance();
    } else {
        println("Villa_fann_" + lexem + "_bjost_vid_" + s);
    }
}

// <program> ::= <function>
public static void program(){
    if(token == DEFINE){
        while(token == DEFINE){
            advance();
            function();
        }
    } else {
        println("Bjost_vid_falli_fann_" + lexem);
    }
}

// <function> ::= NAME(<names>{
//                      [VAR [NAME,]*NAME ;]*
//                      <exp><express>}
public static void function(){
    if(NAME == token){
        advance();
        over("(");
        names();
        over(")");
    } else {
        println("Bjost_vid_nafni_a_falli_fann_" + lexem);
    }
    over("{");
    if(VAR == token){
        advance();
        if(NAME == token){
            advance();
            while(",".equals(lexem) && nextToken == NAME){
                over(",");
                advance();
            }
        } else {
            println("Villa_vanntar_breytunafn_fann_" + lexem);
        }
        over(";");
    }
    expr();
    over(";");
    express();
    over("}");
}

// <names> ::= <name>,<names> | name | ""
public static void names(){
    if(NAME == token){
        advance();
        if(",".equals(lexem) && nextToken == NAME){
            over(",");
            names();
        }
    }
}

// <express> ::= <expr>,<express> |
public static void express(){
    if(lexem.equals("{}")){

```

```

    return;
} else {
    expr();
    if (lexem.equals(";") && nextToken != 0) {
        over(";");
        express();
    }
}
}

// <expr> ::= RETURN<expr> | NAME = <expr> | <binopexpr>
public static void expr() {
    if (RETURN == token) {
        advance();
        expr();
    }
    if (NAME == token && nextLexem.equals("=")) {
        advance();
        over("=");
        expr();
    }
    binopexpr();
}

// <binopexpr> ::= <smallexp> | [<smallexp><op>]+<smallexp>
public static void binopexpr() {
    if (token == LITERAL && nextToken == OPNAME) {
        while (token == LITERAL && nextToken == OPNAME) {
            smallexpr();
        }
    } else {
        smallexpr();
    }
}

// <smallexp> ::= (<expr>) | LITERAL | NAME | OPNAME<smallexp> | <ifexpr> | NAME(<args>)
public static void smallexpr() {
    if ("(" .equals(lexem)) {
        over("(");
        expr();
        over(")");
    }
    if (NAME == token && nextLexem.equals("(")) {
        advance();
        over("(");
        args();
        over(")");
    }

    if (LITERAL == token) {
        advance();
    }
    if (OPNAME == token) {
        advance();
        smallexpr();
    }
    if (NAME == token) {
        advance();
    }
    if (IF == token) {
        advance();
        ifexpr();
    }
    if (WHILE == token) {
        advance();
        over("(");
        expr();
        over(")");
    }
}

```

```

        body ();
    }
}

// <ifexpr> ::= (<expr><body>
public static void ifexpr () {
    over("(");
    expr();
    over(")");
    body();
    while (ELSIF == token) {
        advance();
        ifexpr();
    }
    if (ELSE == token) {
        advance();
        body();
    }
}

// <body> ::= {<express><express>
public static void body () {
    over("{");
    express();
    over("}");
    express();
}

// <args> ::= <expr> | <expr>,<args>
public static void args () {
    if (lexem.equals(",")) {
        return;
    } else {
        expr();
        if (lexem.equals(",") && nextToken != 0) {
            over(",");
            args();
        }
    }
}

public static void main( String[] args )
    throws Exception
{
    init();
    program();
}
}

```

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 ycharat(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;
}

```