# Laboratory 8 Documentation

## Documentation

%{

#include "y.tab.h"

#include <stdio.h>

#include <stdlib.h>

//#define YYDEBUG 1

%}

%token BN;

%token ARR;

%token VAR;

%token IF;

%token PRINT;

%token READ;

%token ELSE;

%token ELSEIF;

%token WHILE;

%token FOR;

%token SETNTH;

%token GETNTH;

%token RETURN;

%token IDENTIFIER;

%token INTCONSTANT;

%token STRINGCONSTANT;

%token LEN;

%token NEWLINE;

%token PLUS;

%token MINUS;

%token TIMES;

%token DIV;

%token MOD;

%token EQ;

%token BIGGER;

%token BIGGEREQ;

%token LESS;

%token LESSEQ;

%token EQQ;

%token NEQ;

%token AND;

%token OR;

%token NOT;

%token FALSE;

%token TRUE;

%token EMPTYSTRING;

%token OPEN;

%token CLOSE;

%token BRACKETOPEN;

%token BRACKETCLOSE;

%token COMMA;

%token DIEZ;

%token VERTICAL;

%token SEPARATOR;

%start compound\_statement

%%

compound\_statement : statement SEPARATOR {printf("compound\_statement -> statement \n");};

| statement SEPARATOR compound\_statement {printf("compound\_statement -> satement\_list\n");};

statement : assignment\_statement {printf("statement -> assignment\_statement\n");}

| if\_statement {printf("statement -> if\_statement\n");}

| while\_statement {printf("statement -> while\_statement\n");}

| for\_statement {printf("statement -> for\_statement\n");}

| function\_call\_statement {printf("statement -> function\_call\_statement\n");};

function\_call\_statement : READ OPEN IDENTIFIER CLOSE {printf("statement -> --> ( IDENTIFIER )\n");}

| PRINT OPEN IDENTIFIER CLOSE {printf("statement -> <-- ( IDENTIFIER )\n");}

| PRINT OPEN INTCONSTANT CLOSE {printf("statement -> <-- ( INTCONSTANT )\n");}

| PRINT OPEN STRINGCONSTANT CLOSE {printf("statement -> <-- ( STRINGCONSTANT )\n");};

if\_statement : IF OPEN condition CLOSE DIEZ compound\_statement DIEZ {printf("if\_statement -> IF ( condition ) # compound\_statement #\n");}

| IF OPEN condition CLOSE DIEZ compound\_statement DIEZ ELSE DIEZ compound\_statement DIEZ {printf("if\_statement -> IF ( condition ) # compound\_statement # else # compound\_statement #\n");};

while\_statement : WHILE OPEN condition CLOSE DIEZ compound\_statement DIEZ {printf("assignment\_statement : while ( condition ) # compound\_statement #\n");};

for\_statement : FOR for\_header DIEZ compound\_statement DIEZ {printf("for\_statement : for ( for\_header ) # compound\_statement #\n");};

for\_header : assignment\_statement VERTICAL condition VERTICAL assignment\_statement {printf("for\_header : assignment\_statement | condition | assignment\_statement\n");};

condition : expression relation expression {printf("condition : expression relation expression");};

relation : LESS {printf("relation : <");}

| LESSEQ {printf("relation : <=");}

| BIGGER {printf("relation : >");}

| BIGGEREQ {printf("relation : >=");}

| NEQ {printf("relation : !=");}

| EQQ {printf("relation : ==");};

assignment\_statement : IDENTIFIER EQ expression {printf("assignment\_statement -> IDENTIFIER = expression\n");}

| IDENTIFIER EQ array\_statement {printf("assignment\_statement -> IDENTIFIER = array\_statement\n");};

array\_statement : BRACKETOPEN BRACKETCLOSE {printf("array\_statement -> [ ]\n");}

| BRACKETOPEN expression\_list BRACKETCLOSE {printf("array\_statement -> [ expression\_list ]\n");};

expression : expression PLUS term {printf("expression -> expression + term\n");}

| expression MINUS term {printf("expression -> expression - term\n");}

| term {printf("expression -> term\n");};

term : term TIMES factor {printf("term -> term \* factor\n");}

| term DIV factor {printf("term -> term / factor\n");}

| factor {printf("term -> factor\n");};

factor : OPEN expression CLOSE {printf("factor -> ( expression )\n");}

| IDENTIFIER {printf("factor -> IDENTIFIER\n");}

| INTCONSTANT {printf("factor -> INTCONSTANT\n");};

expression\_list : expression {printf("expression\_list -> expression\n");}

| expression COMMA expression\_list {printf("expression -> expression , expression\_list\n");};

%%

int yyerror(char \*s)

{

printf("%s\n",s);

}

extern FILE \*yyin;

int main(int argc, char \*\*argv)

{

if(argc>1) yyin = fopen(argv[1],"r");

if(!yyparse()) fprintf(stderr, "\tOK\n");

}

## How to run the project

You have the lex from the previous project. Now we have to compile the yacc project.

Run the following commands:

yacc -d lang.y

After you have to do a cross compile between the lex and yacc source files.

gcc lex.yy.c y.tab.c -o scanner

Finally, run the application by providing a source file for your language:

./scanner p1.in