Experiment 7

RA2111026010354

YAGNESH S

CODE:

minijava.l

%{

#include "minijava.tab.h"

%}

%%

"class" { return CLASS; }

"public" { return PUBLIC; }

"static" { return STATIC; }

"void" { return VOID; }

"main" { return MAIN; }

"String" { return STRING; }

"args" { return ARGS; }

"{" { return OPEN\_BRACE; }

"}" { return CLOSE\_BRACE; }

"(" { return OPEN\_PAREN; }

")" { return CLOSE\_PAREN; }

";" { return SEMICOLON; }

"[" { return OPEN\_BRACKET; }

"]" { return CLOSE\_BRACKET; }

"=" { return ASSIGN; }

"int" { return INT; }

"boolean" { return BOOLEAN; }

"if" { return IF; }

"else" { return ELSE; }

"while" { return WHILE; }

"System.out.println" { return PRINTLN; }

"true"|"false" { return BOOL\_LITERAL; }

"this" { return THIS; }

"new" { return NEW; }

"length" { return LENGTH; }

"!" { return NOT; }

"&&" { return AND; }

"<" { return LESS\_THAN; }

"+" { return PLUS; }

"-" { return MINUS; }

"\*" { return MULTIPLY; }

"true" { return TRUE; }

"false" { return FALSE; }

"return" { return RETURN; }

[a-zA-Z][a-zA-Z0-9\_]\* { yylval.strval = strdup(yytext); return IDENTIFIER; }

[0-9]+ { yylval.numval = atoi(yytext); return INT\_LITERAL; }

"//".\* /\* Ignore comments \*/

[ \t\n] /\* Skip whitespace \*/

. { return yytext[0]; }

%%

int yywrap(void) {

return 1;

}

minijava.y

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

extern int yylex();

extern int yyparse();

extern FILE\* yyin;

void yyerror(const char\* s);

%}

%token CLASS PUBLIC STATIC VOID MAIN STRING ARGS

%token OPEN\_BRACE CLOSE\_BRACE OPEN\_PAREN CLOSE\_PAREN SEMICOLON

%token OPEN\_BRACKET CLOSE\_BRACKET ASSIGN

%token INT BOOLEAN IF ELSE WHILE PRINTLN RETURN

%token INT\_LITERAL BOOL\_LITERAL IDENTIFIER

%token THIS NEW LENGTH NOT AND LESS\_THAN PLUS MINUS MULTIPLY TRUE FALSE

%%

Program: ClassDeclaration

| MainClass ClassDeclaration

;

MainClass: CLASS IDENTIFIER OPEN\_BRACE PUBLIC STATIC VOID MAIN OPEN\_PAREN STRING OPEN\_BRACKET CLOSE\_BRACKET IDENTIFIER CLOSE\_PAREN OPEN\_BRACE Statement CLOSE\_BRACE

;

ClassDeclaration: CLASS IDENTIFIER OPEN\_BRACE ClassBody CLOSE\_BRACE

| ClassDeclaration CLASS IDENTIFIER OPEN\_BRACE ClassBody CLOSE\_BRACE

;

ClassBody: ClassMemberList

;

ClassMemberList: ClassMember

| ClassMemberList ClassMember

;

ClassMember: FieldDeclaration

| MethodDeclaration

;

FieldDeclaration: Type IDENTIFIER SEMICOLON

;

MethodDeclaration: PUBLIC Type IDENTIFIER OPEN\_PAREN FormalParameterList CLOSE\_PAREN OPEN\_BRACE StatementList RETURN Expression SEMICOLON CLOSE\_BRACE

;

FormalParameterList: /\* Empty \*/

| Type IDENTIFIER

| FormalParameterList COMMA Type IDENTIFIER

;

Type: INT

| BOOLEAN

| STRING

| IDENTIFIER

;

StatementList: /\* Empty \*/

| StatementList Statement

;

Statement: Block

| AssignmentStatement

| ArrayAssignmentStatement

| IfStatement

| WhileStatement

| PrintlnStatement

| ReturnStatement

;

Block: OPEN\_BRACE StatementList CLOSE\_BRACE

;

AssignmentStatement: IDENTIFIER ASSIGN Expression SEMICOLON

;

ArrayAssignmentStatement: IDENTIFIER OPEN\_BRACKET Expression CLOSE\_BRACKET ASSIGN Expression SEMICOLON

;

IfStatement: IF OPEN\_PAREN Expression CLOSE\_PAREN Statement ELSE Statement

;

WhileStatement: WHILE OPEN\_PAREN Expression CLOSE\_PAREN Statement

;

PrintlnStatement: PRINTLN OPEN\_PAREN Expression CLOSE\_PAREN SEMICOLON

;

ReturnStatement: RETURN Expression SEMICOLON

;

Expression: INT\_LITERAL

| BOOL\_LITERAL

| IDENTIFIER

| THIS

| NEW INT OPEN\_BRACKET Expression CLOSE\_BRACKET

| NEW IDENTIFIER OPEN\_PAREN CLOSE\_PAREN

| Expression DOT LENGTH

| Expression OPEN\_BRACKET Expression CLOSE\_BRACKET

| OPEN\_PAREN Expression CLOSE\_PAREN

| Expression MULTIPLY Expression

| Expression PLUS Expression

| Expression MINUS Expression

| Expression LESS\_THAN Expression

| Expression AND Expression

| Expression NOT

| Expression DOT IDENTIFIER OPEN\_PAREN ArgumentList CLOSE\_PAREN

| Expression DOT IDENTIFIER

| IDENTIFIER OPEN\_PAREN ArgumentList CLOSE\_PAREN

| IDENTIFIER OPEN\_BRACKET Expression CLOSE\_BRACKET

;

ArgumentList: /\* Empty \*/

| Expression

| ArgumentList COMMA Expression

;

%%

void yyerror(const char\* s) {

printf("Syntax error.\n");

}

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

if (argc < 2) {

printf("Usage: %s <input\_file>\n", argv[0]);

return 1;

}

FILE\* input = fopen(argv[1], "r");

if (!input) {

printf("Error: Could not open file %s\n", argv[1]);

return 1;

}

yyin = input;

yyparse();

printf("Program parsed successfully.\n");

fclose(input);

return 0;

}

**OUTPUT:**

