ANKARA UNIVERSITY

COM2043

PROGRAMMING LANGUAGE CONCEPTS

PROJECT 1 REPORT

Sıla Şahin

21290215

Computer Engineering

Introduction

In this project, I aim to develop my programming language using Lex and Yacc. My programming language's name is "Opclo (Open-Close)". It is designed as an unusual language due to its syntax. The rules of the programming language and grammatical BNF notations were explained in the report. Lex and Yacc files contents are included in this report. Finally, this programing language is explained by giving some code examples.

Rules

- All programs must start with "open" and end with "close" and the program can be finished without typing codes.
- All commands must be ended by ".".
- There is only one type exist; "integer".
- Variables names must be start with ampersand "&", after that it must be a lowercase letter and after small letter there must be at least one lowercase/uppercase letter or digit. Variable names cannot include symbol except initial ampersand "&" symbol.
- Variable defining is like; "&value1 let 8."
- Expressions include printing function, if statement, while-do loop, for loop, and assignment.
- Logic includes "and" and "or" operations.
- There are some comparisons like; "smaller" is less than, "bigger" is greater than, "==" is equal, and "not==" is not equal. Logic covers these too.
- There is four arithmetic operations; plus is addition, minus is subtraction, divide is division, and multiply is multiplication.
- The precedence of operators given as: parenthesis>comparisons>and/or

Grammar Rules

In this part, I explained BNF rules of Opclo.

<program>→ <START>< exprs>< FINISH>

| <START>< FINISH>

<START> → open

<FINISH> → close

This function is main function. All programs starts with “open” and ends with “close”.

<exprs> → <expr>

| <expr><exprs>

Program can include one expression or more than one expression.

<expr>→< VARIABLE>< ASSIGN>< arithmetics1> <ENDLINE>

| <arithmetics1> <WRITE>< ENDLINE>

| <IF>< LOGIC> < BEGINBLOCK>< exprs>< FINISHBLOCK> <ENDLINE>

| <WHILE> <LOGIC>< DO>< BEGINBLOCK><exprs> <FINISHBLOCK>< ENDLINE>

|< INTEGER>< FOR> <BEGINBLOCK> <exprs>< FINISHBLOCK>< ENDLINE>

< VARIABLE>→[&][a-z][a-zA-Z0-9]+

<ASSIGN>→let

<ENDLINE>→.

<WRITE>→write

<IF>→whether

< BEGINBLOCK>→<

< FINISHBLOCK>→>

<WHILE>→when

< DO>→do

< INTEGER>→ [1-9][0-9]*

< FOR> → times

This part include assignment, printing, if statement, while-do loop and for loop operations.

<arithmetics1>→< arithmetics1>< PLUS>< arithmetics1>

| <arithmetics1>< MINUS>< arithmetics1>

| <arithmetics2>

< PLUS>→++

< MINUS>→--

<arithmetics2>→< arithmetics2>< DIVIDE> <values>

| <arithmetics2>< MULTIPLY>< values>

|< values>

< DIVIDE>→//

< MULTIPLY>→**

<values>→< OPENPAR> <arithmetics1> <CLOSEPAR>

| <INTEGER>

| <VARIABLE>

< OPENPAR>→ {

<CLOSEPAR>→ }

Arithmetic operations and presedences are defined in this part. Presedences is like: paranthesis > divide/multiply > plus/minus

<LOGIC>→< LOGIC> <AND> <LOGIC>

| <LOGIC> <OR> <LOGIC>

| <compare>

| <OPENPAR> <LOGIC> <CLOSEPAR>

<AND>→^

<OR>→v

Logical operations are defined here. “And”, “or”, and some comparison operations are used in logic function.

<compare>→< values><SMALLER> <arithmetics1>

| <values> <BIGGER ><arithmetics1>

| <values> <EQUAL> <arithmetics1>

| <values> <NOTEQUAL> <arithmetics1>

<SMALLER>→smaller

<BIGGER >→bigger

<EQUAL>→==

<NOTEQUAL>→not==

In this part, comparisons are defined. Smaller is less then, bigger is greater then.

Presedences is like: paranthesis > comparison > and/or

mpl.y File

```
%{
   #include <stdio.h>
   #include <stdlib.h>
   #include <string.h>
   int yylex(void);
   void yyerror(){
            printf("syntax error\n");
            exit(1)}
%}
%start program
%token VARIABLE INTEGER WHILE DO IF FOR WRITE PLUS MINUS DIVIDE MULTIPLY EQUAL NOTEQUAL AND OR
SMALLER BIGGER ASSIGN
%token BEGINBLOCK FINISHBLOCK OPENPAR CLOSEPAR ENDLINE START FINISH
%left PLUS MINUS
%left DIVIDE MULTIPLY
%left EQUAL NOTEQUAL
%left AND OR
%left SMALLER BIGGER
%%
program: START exprs FINISH
            | START FINISH;
exprs: expr
             | expr exprs;
expr: VARIABLE ASSIGN arithmetics1 ENDLINE
      | arithmetics1 WRITE ENDLINE
      | IF LOGIC BEGINBLOCK exprs FINISHBLOCK ENDLINE
      | WHILE LOGIC DO BEGINBLOCK exprs FINISHBLOCK ENDLINE
      | INTEGER FOR BEGINBLOCK exprs FINISHBLOCK ENDLINE;
```

```
arithmetics1: arithmetics1 PLUS arithmetics1
            | arithmetics1 MINUS arithmetics1
            | arithmetics2;
arithmetics2: arithmetics2 DIVIDE values
            | arithmetics2 MULTIPLY values
            | values;
values: OPENPAR arithmetics1 CLOSEPAR
      | INTEGER
      | VARIABLE;
LOGIC: LOGIC AND LOGIC
      | LOGIC OR LOGIC
      | compare
      | OPENPAR LOGIC CLOSEPAR;
compare: values SMALLER arithmetics1
         | values BIGGER arithmetics1
         | values EQUAL arithmetics1
         | values NOTEQUAL arithmetics1;
%%
int main(){
   yyparse();
   printf("OK\n");
   return 0;}
```

mpl.l File

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "y.tab.h"
%}
%%
"open"          {return START;}
"close"         {return FINISH;}
"<"             {return BEGINBLOCK;}
">"             {return FINISHBLOCK;}
"."             {return ENDLINE;}
"{"             {return OPENPAR;}
"}"             {return CLOSEPAR;}
"++"            {return PLUS;}
"--"            {return MINUS;}
"//"            {return DIVIDE;}
"**"            {return MULTIPLY;}
"=="            {return EQUAL;}
"not=="         {return NOTEQUAL;}
"^"             {return AND;}
"v"             {return OR;}
"smaller"       {return SMALLER;}
"bigger"        {return BIGGER;}
"when"          {return WHILE;}
"do"            {return DO;}
"whether"       {return IF;}
"times"         {return FOR;}
"write"         {return WRITE;}
"let"           {return ASSIGN;}
[&][a-z][a-zA-Z0-9]+   {return VARIABLE;}
[1-9][0-9]*     {return INTEGER;}
```

[\t\n] ;

. {printf("syntax error \n"); exit(1);}

%%

Code Examples

Example1

open

&value1 let 8. (&value1 is 8)

&value2 let 16. (&value2 is 16)

&value3 let 2. (&value3 is 2)

&value4 let 5. (&value4 is 5)

close

Example2

open

&result1 let &value1 ++ &value3. (&result1 is 8+2= 10)

&result2 let &value2 -- &value1. (&result2 is 16-8= 8)

&result3 let &value2 // &value3. (&result3 is 16/2=8)

*&result4 let &value1 ** &value3.* (&result4 is 8*2=16)

&result1 write.

&result2 write.

&result3 write.

&result4 write.

close

Output:

10

8

8

16

Example3

```
open
&value2 let 16.                          (&value2 is 16)
&value3 let 2.                           (&value3 is 2)
&value4 let 5.                           (&value4 is 5)
&result3 let &value2 // &value3.         (&result3 is 16/2=8)
whether{5 smaller &result3 ^ &value4 not== 6} <          (if(5<8 and 4!=6))
        &result3 write.
>.
close
```

Output:

8

Example4

```
open
&value1 let 8.           (&value1 is 8)
&value4 let 5.           (&value4 is 5)
when &value1 bigger 2 do <       (while 8>2 do)
        &value4 let &value4 ++ 5.
        &value1 let &value1 -- 1.
        &value4 write.
>.
close
```

Output:

10

15

20

25

30

35

Example5

```
open
&value3 let 2.                    (&value3 is 2)
6 times <
        &value3 let &value3 ** 2.
        &value3 write.
>.
close
```

Output:

4

8

16

32

64

128

Example6

```
open
whether 8 smaller 10 <            (if 8<10)
        100 write.
>.
close
```

Output:

100