COM2043
Programming Language Concepts

Project 1 Report

Serkan Yılmaz
21290681
Computer Engineering

Table Of Contents

Introduction

In this report, my aim is to introduce my programming language named “Olsun” developed using lex and yacc. It is designed for teaching programming to kids in Turkey with the Turkish words. It is explained in the report that BNF descriptions and rules ol the programming language. It is in report that lex and yacc files. Finally, programming in this language is explained in example codes.

Rules

- All programs must end with “bitir.”
- All commands must ended by “,”
- Variable names be any alphanumeritic string, but they must start witjh a letter
- Variable names cannot include symbols except underscore “_”. However “_” cannot be an initial of a variable name.
- Only variable type is integer.
- Variable definition is like:

  a 5 olsun,

  This code makes value of a, 5.
- Statements include assignment, printing function, logic operations and loops.
- Assignment includes addition, subtraction,division and multiplication.
- The presedence of “*” and “/” is higher than “+” and “-”. Paranthesis have the highest presedence.
- Logic includes and and or gates and also comparisons such that less than, greater than, is equal to and is not equal to.
- Precedence in logic:
  - Paranthesis>Comparisons>And/Or

Grammar Rules

This part contains BNF rules of Olsun.

<program> → <statements> <FINISH>
<FINISH> → bitir.

This part creates the main funtion. All programs must ent with “bitir.”

<statements> → <statement> | <statement> <statements>

Program can contain one startement or more than one statement. This is recursive.

<statement> → <VARIABLE> <A> <ASSIGN> <ENDCOMMAND>
|<D> <PRINT> <ENDCOMMAND>
|<LOGIC> <IF> <STARTBLOCK> <statements> <ENDBLOCK>
|<LOGIC> <IF> <statement>
|<LOGIC> <WHILE> <STARTBLOCK> <statements> <ENDBLOCK>
|<LOGIC> <WHILE> <statement>
|<NUMBER> <FOR> <STARTBLOCK> <statements> <ENDBLOCK>
|<NUMBER> <FOR> <statement>

<VARIABLE> → <VARIABLE> <A> <ASSIGN> <ENDCOMMAND>
<ASSIGN> → olsun
<ENDCOMMAND> → ,
<PRINT> → yaz
<IF> → ise
<STARTBLOCK> → {
<ENDBLOCK> → }
<WHILE> → iken
<FOR> → kere
<NUMBER> → [1-9][0-9]*

Assignment, printing, logic and loop operations defined here.

<A> → <A> <SUBTRACT> <B>
|<A> <ADD> <B>
|<B>
<SUBTRACT> → -
<ADD> → +

<B> → <B> <DIVIDE> <D>
|<B> <MULTIPLY> <D>
|<D>
<DIVIDE> → /
<MULTIPLY> → *

<D> → <STARTPAR> <A> <ENDPAR>
|<NUMBER>
|<VARIABLE>
<STARTPAR> → (
<ENDPAR> →)

Presedences and basic math operations are defined here.
Paranthesis > Division/Multiplication > Subtraction/Additon

<LOGIC> → <TRUEVAL>|<FALSEVAL>
|<LOGIC> <AND ><LOGIC >
|<LOGIC> <OR> <LOGIC >
|<comparison>
<TRUEVAL> → dogru
<FALSEVAL> → yanlis
<AND> → ve
<OR> → veya

<comparison> → <D> <LESSTHAN> <A>
|<D> <GREATERTHAN> <A>
|<D> <ISEQUAL> <A>
|<D> <ISNOTEQUAL> <A>
|<F>

<LESSTHAN> → <
<GREATERTHAN> → >
<ISEQUAL> → ==
<ISNOTEQUAL> → !=

<F> → <STARTPAR> <LOGIC> <ENDPAR>

Presedences and logic and comparison operations are defined here.
Paranthesis > Comparison > And/Or

mpl.l

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "y.tab.h"
%}
%option noyywrap
%%
"{"  {return STARTBLOCK;}
"}"  {return ENDBLOCK;}
"("  {return STARTPAR;}
")"  {return ENDPAR;}

"+"  {return ADD;}
"-"  {return SUBTRACT;}
"*"  {return MULTIPLY;}
"/"  {return DIVIDE;}

","  {return ENDCOMMAND;}
"bitir." {return FINISH;}

"olsun"  {return ASSIGN;}
"yaz"  {return PRINT;}

"iken"  {return WHILE;}
"kere"  {return FOR;}
"ise"  {return IF;}

"dogru"  {return TRUEVAL;}
"yanlis" {return FALSEVAL;}

"ve"  {return AND;}
"veya"  {return OR;}
"=="  {return ISEQUAL;}
"!="  {return ISNOTEQUAL;}
"<"  {return LESSTHAN;}
">"  {return GREATERTHAN;}

[a-zA-Z][_a-zA-z0-9]*  {return VARIABLE;}
[1-9][0-9]*  {return NUMBER;}
[ \t\n] ;
.  {printf("syntax error \n"); exit(1);}
%%
```

mpl.y

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int yylex(void);
void yyerror(){
 printf("syntax error\n");
 exit(1);
}
%}
%start program
%token NUMBER VARIABLE
%token ADD SUBTRACT MULTIPLY DIVIDE
%token AND OR
%token TRUEVAL FALSEVAL
%token ISEQUAL ISNOTEQUAL LESSTHAN GREATERTHAN
%token ASSIGN IF WHILE FOR PRINT
%token STARTPAR ENDPAR STARTBLOCK ENDBLOCK ENDCOMMAND FINISH
%left ADD SUBTRACT MULTIPLY DIVIDE
%left AND OR
%left LESSTHAN GREATERTHAN ISEQUAL ISNOTEQUAL
%right ASSIGN
%%
program : statements FINISH;
statements: statement | statement statements;
statement :VARIABLE A ASSIGN ENDCOMMAND
  |D PRINT ENDCOMMAND
  |LOGIC IF STARTBLOCK statements ENDBLOCK
  |LOGIC IF statement
  |LOGIC WHILE STARTBLOCK statements ENDBLOCK
       |LOGIC WHILE statement
  |NUMBER FOR STARTBLOCK statements ENDBLOCK
       |NUMBER FOR statement;
A :A SUBTRACT B
 |A ADD B
 |B;
B :B DIVIDE D
 |B MULTIPLY D
 |D;
D :STARTPAR A ENDPAR
 |NUMBER
 |VARIABLE;
LOGIC :TRUEVAL|FALSEVAL
    |LOGIC AND LOGIC
    |LOGIC OR LOGIC
    |comparison;
comparison :D LESSTHAN A
        |D GREATERTHAN A
        |D ISEQUAL A
        |D ISNOTEQUAL A
        |F;
F :STARTPAR LOGIC ENDPAR;
%%
int main() {  yyparse();  printf("OK\n");  return 0; }
```

Code Examples

Example - 1

```
a 5 olsun,          a is 5.
b 7 olsun,          b is 7.
c 9 olsun,          c is 9

d a+b olsun,        d is a+b so 12.
yaz d,              print d so print 12.

bitir.              end ef the program
```

output:
12

Example - 2 : Finding Maximum of Elements

```
a 5 olsun,              a is 5.
b 7 olsun,              b is 7.
c 9 olsun,              c is 9
max a olsun,            max is a so max is 5.
a<b ise max b olsun,    if a<b max is b so max is 7.
b<c ise max c olsun,    if b<c max is c so max is 9.
max yaz,                print max so print 9.
bitir.                  end ef the program
```

output:
9

Example - 3

```
(dogru veya yanlis) ise 1 yaz,
bitir.
```

if True or False print 1. True or False is True so print 1. end ef the program

output:
1

Example - 4 : iken Loop (while)

```
x 1 olsun,
sayac 1 olsun,
sayac<5 iken {
 x x*2 olsun,
 x yaz,
 sayac sayac+1 olsun,
}
bitir.
```

x is 1.
sayac is 1.
while (sayac<5){
x is x*2
print x
sayac is sayac+1
}
end ef the program

output:
2
4
8
16

- sayac is increasing 1 in every loop. Prints powers of 2.

Example - 5 : kere Loop (for)

```
yas 19 olsun,                yas is 19.
3 kere yas yas+1 olsun,      3 times add 1 to yas. yas is 22.
yas yaz,                     print yas so print 22.

yas 19 olsun,                yas is 19.
3 kere {                     3 times{
 yas yas+1 olsun,            add 1 to yas.
 yas yaz,                    print yas.
}                            }
bitir.                       end ef the program
```

output:

22
20
21
22