COM241

PROGRAMMING LANGUAGES CONCEPTS

# LEX & YACC

>MUSTAFA MURAT 16290109

>MURAT ÇOBAN 16290087

>RULES OF MY PROGRAMMING LANGUAGE

- All programs must end with "exit".
- Only one type exist; "int".
- Variables names only can be alphabetic string.
- An assignment includes five types of operators; "+", "-", "*", "/".
- An comparison includes all operators; "<", ">", "<=", ">=", "==", "!=".
- There are two logical operators; "||", "&&".
- It has one conditional statement; if else
- It has one loop structure; while
- The number of variables or constants in an expression is unlimited.

>BNF Notation

```
<lines>    ::= exit_command
             | <lines> ; <line>

<line>     ::= <assignment>
             | print <exp>
             | IF (<condition>) <line> ELSE <line>
             | IF (<condition>) <line>
             | WHILE (<condition>) <line>

<assignment> ::= <identifier> = <exp>

<condition> ::= <condition> > <term>
              | <condition> < <term>
              | <condition> >= <term>
              | <condition> <= <term>
              | <condition> == <term>
              | <condition> != <term>
              | <condition> || <term>
              | <condition> && <term>
              | <exp>

<exp> ::= <term>
        | <exp> + <term>
        | <exp> - <term>

<term> ::= <factor>
         | <term> * <factor>
         | <term> / <factor>

<factor> ::= <number>
           | <identifier>

<number> ::= 1 | 2 | … | 9

<identifier> ::= <digit> | <identifier> ; <digit>

<digit> ::=  a | b | … | z
```

>EBNF Notation

```
<lines>     ::= exit_command
              | {<line>}

<line>     ::= assignment
             | print exp
             | IF (<condition>) <line> [ELSE <line>]
             | WHILE (<condition>) <line>
             ;

<assignment> ::= <identifier> = <exp>

<condition> ::= <exp> {( > | < | >= | <= | == | != | || |
&& )<term>}

<exp> ::= <term> {( + | - ) <term>}

<term> ::= <factor> {( * | / ) <factor>}

<factor> ::= <number>
          | <identifier>

<number> ::= 1 | 2 | … | 9

<identifier> ::= {<digit>}

<digit> ::=  a | b | … | z
```

>Lex Input

```
%{
     #include "y.tab.h"
     void yyerror (char *s);
%}

%%

"print"           {return print; }
"exit"            {return exit_command;}
"if"              {return IF;}
"else"            {return ELSE;}
"while"           {return WHILE;}
"<="              {return LE;}
">="              {return GE;}
"=="              {return EQ;}
"!="              {return NE;}
"||"              {return OR;}
"&&"              {return AND;}
[a-zA-Z]          {yylval.id = yytext[0]; return identifier;}
[0-9]+            {yylval.num = atoi(yytext); return number;}
[ \t\n]           ;  /* Ignore Whitespace */
[-+*/<>!(){}=;]   {return yytext[0];}
.                 {ECHO; yyerror ("Unexpected character\n");}

%%

int yywrap (void)
{
     return 1;
}
```

>Yacc Input

```
%{
	#include<stdio.h>
	int yylex();
	int symbols[52];
	int symbolVal(char symbol);			/* Symbol Table */
	void updateSymbolVal(char symbol, int val);
	void yyerror(char *s);
%}

%union {
	int num;	/* Integer Value */
	char id;	/* Char Value */
}
%start lines					/* Yacc Definitions */
%token print exit_command
%token <num> IF ELSE WHILE LE GE EQ NE OR AND
%token <num> number
%token <id>  identifier
%type  <num> line exp term factor condition
%type  <id>  assignment print
%right '='
%left  AND OR
%left  '<' '>' LE GE EQ NE
%left  '*''/'
%left  '+''-'
%right  '!'

%%

lines		: exit_command		{ printf("OK\n"); }
		| line lines		{ ; }
		;

line		: assignment				{ ; }
		| print exp				{ printf("%d\n",$2); }
		| IF '('condition')' line ELSE line	{ ; }
		| IF '('condition')' line		{ ; }
		| WHILE '('condition')' line		{ ; }
		;

assignment	: identifier '=' exp		{updateSymbolVal($1, $3);}
		;

condition	: condition '>' term		{ $$ = $1 > $3  ? 1 : 0; }
		| condition '<' term		{ $$ = $1 < $3  ? 1 : 0; }
		| condition GE  term		{ $$ = $1 >= $3 ? 1 : 0; }
		| condition LE  term		{ $$ = $1 <= $3 ? 1 : 0; }
		| condition EQ  term		{ $$ = $1 == $3 ? 1 : 0; }
		| condition NE  term		{ $$ = $1 != $3 ? 1 : 0; }
		| condition OR  term		{ $$ = $1 || $3 ? 1 : 0; }
		| condition AND term		{ $$ = $1 && $3 ? 1 : 0; }
		| exp				{ $$ = $1; }
		;

exp		: term				{ $$ = $1; }
		| exp '+' term			{ $$ = $1 + $3; }
		| exp '-' term			{ $$ = $1 - $3; }
		;
```

```
term         : factor               { $$ = $1; }
             | term '*' factor            { $$ = $1 * $3; }
             | term '/' factor            { $$ = $1 / $3; }
             ;

factor       : number               { $$ = $1; }
             | identifier           { $$ = symbolVal($1); }
             ;

%%

int computeSymbolIndex(char token)
{
      int idx = -1;

      if(token >= 'a' && token <= 'z')
      {
            idx = token - 'a' + 26;
      }

      else if(token >= 'A' && token <= 'Z')
      {
            idx = token - 'A';
      }

      return idx;
}

int symbolVal(char symbol)          /* Returns the value of a given symbol */
{
      int bucket = computeSymbolIndex(symbol);

      return symbols[bucket];
}

void updateSymbolVal(char symbol, int val)      /* Updates the value of a given
symbol */
{
      int bucket = computeSymbolIndex(symbol);

      symbols[bucket] = val;
}

int main (void)
{
      yyparse();

      return 0;
}

void yyerror (char *s)
{
      fprintf (stderr, "%s\n", s);
}
```

```
a = 9

while(a > 8 || a < 10)
     if(a >= 6 && a != 10)
     print a / 3

a = 15 + 9 * 8

print a

exit
```

```
> 3
  87
  OK
```

```
a = 9

while(a > 8 || a < 10)
     else
     if(a >= 6 && a != 10)
     print a / 3;

a = 15 + 9 * 8

print a

exit
```

```
> syntax error
```