

LANGUAGE PROCESSORS

ASSIGNMENT - 1

Name : Deep Walke
Roll Number : BT19CSE122
Date of Sub. : 25/01/2022

Problem Statement:

1. Lex/Yacc or Flex/Bison: You should be able to use lex/flex to identify certain words, strings having certain words, strings having fifth element as 'a', strings having last letter as 'p', strings having exactly 2 letters as 'ab', strings having one Capital, one special and one numerical, etc.
2. You should be able to use Yacc/Bison for identifying a simple grammar for while-loop which has just one condition "==" and which has just one statement. The statement can be another while-loop or "var = var;"
3. Create a small calculator programme using Yacc. You should be able to use provide associative rules and precedence rules for calculator program. You should enhance the calculator to include an exponential which can accept $a^b^c^d$ and it should be right-associative and of highest order. Marks are reserved for proper error handling - giving as many errors as possible in one go - and as detailed and correct error messages as possible.

Question-1:

- Part a :strings having certain words'-
 - `^[a-zA-Z]*deep|walke|vnit[a-zA-Z]*$`
- Part b :Code to identify strings having fifth element as 'a'-
 - `^[a-zA-Z]{4}a[a-zA-Z]*$`
- Part c :Code to identify strings having last letter as 'p' -
 - `^.*p$`
- Part d :Code to identify strings having exactly 2 letters as 'ab'-
 - `^ab$`
- Part e :Code to identify strings having one Capital, one special and one numerical
 - `^[A-Z].*[@#$%^&-+=\(\)].*[0-9].*$`
 - `^[A-Z].*[0-9].*[@#$%^&-+=\(\)].*$`
 - `^[A-Z].*[0-9].*[A-Z].*$`
 - `^[A-Z].*[0-9].*[A-Z].*[0-9].*$`
 - `^[0-9].*[@#$%^&-+=\(\)].*[A-Z].*$`
 - `^[0-9].*[A-Z].*[@#$%^&-+=\(\)].*$`
- One extra string : Code to identify if a given string is time.
 - `^(0?[1-9]|1[0-2]):[0-5][0-9]$`

CODE

q1.1

Lex file

lex file will be having three parts as shown below and these parts are separated by '%%'

```
%{
#include<stdio.h>
#include<string.h>
int i = 0;
%}

/* rules */
%%

/* strings having certain words */
^[a-zA-Z]*deep|walke|vnit[a-zA-Z]*$ {printf("string having either
'deep', 'walke', 'vnit'");}
```

```

    /* strings having fifth element as 'a' */
^[a-zA-Z]{4}a[a-zA-Z ]*$ {printf("%s having fifth element as
'a'\n",yytext);}

    /* strings having last letter as 'p' */
^.*p$ {printf("string having last letter as 'p'\n");}

    /* strings having exactly 2 letters as 'ab' */
^ab$ {printf("string having exactly 2 letters as 'ab'\n");}

    /* strings having one Capital, one special and one numerical */

^.*[0-9].*[@#$%^&+=\(\)].*[A-Z].*$ {printf("one caps one special one
numerical\n");}
^.*[0-9].*[A-Z].*[@#$%^&+=\(\)].*$ {printf("one caps one special one
numerical\n");}
^.*[A-Z].*[@#$%^&+=\(\)].*[0-9].*$ {printf("one caps one special one
numerical\n");}
^.*[A-Z].*[0-9].*[@#$%^&+=\(\)].*$ {printf("one caps one special one
numerical\n");}
^.*[@#$%^&+=\(\)].*[0-9].*[A-Z].*$ {printf("one caps one special one
numerical\n");}
^.*[@#$%^&+=\(\)].*[A-Z].*[0-9].*$ {printf("one caps one special one
numerical\n");}

^(0?[1-9]|1[0-2]):[0-5][0-9]$ {printf("strings have time\n");}
.+

%%

int yywrap(void){}

int main()
{
    // The function that starts the analysis
    yylex();

    return 0;
}

```

Definitions

- The lex Definitions section contains external definitions, preprocessor instructions like #include, and abbreviations.

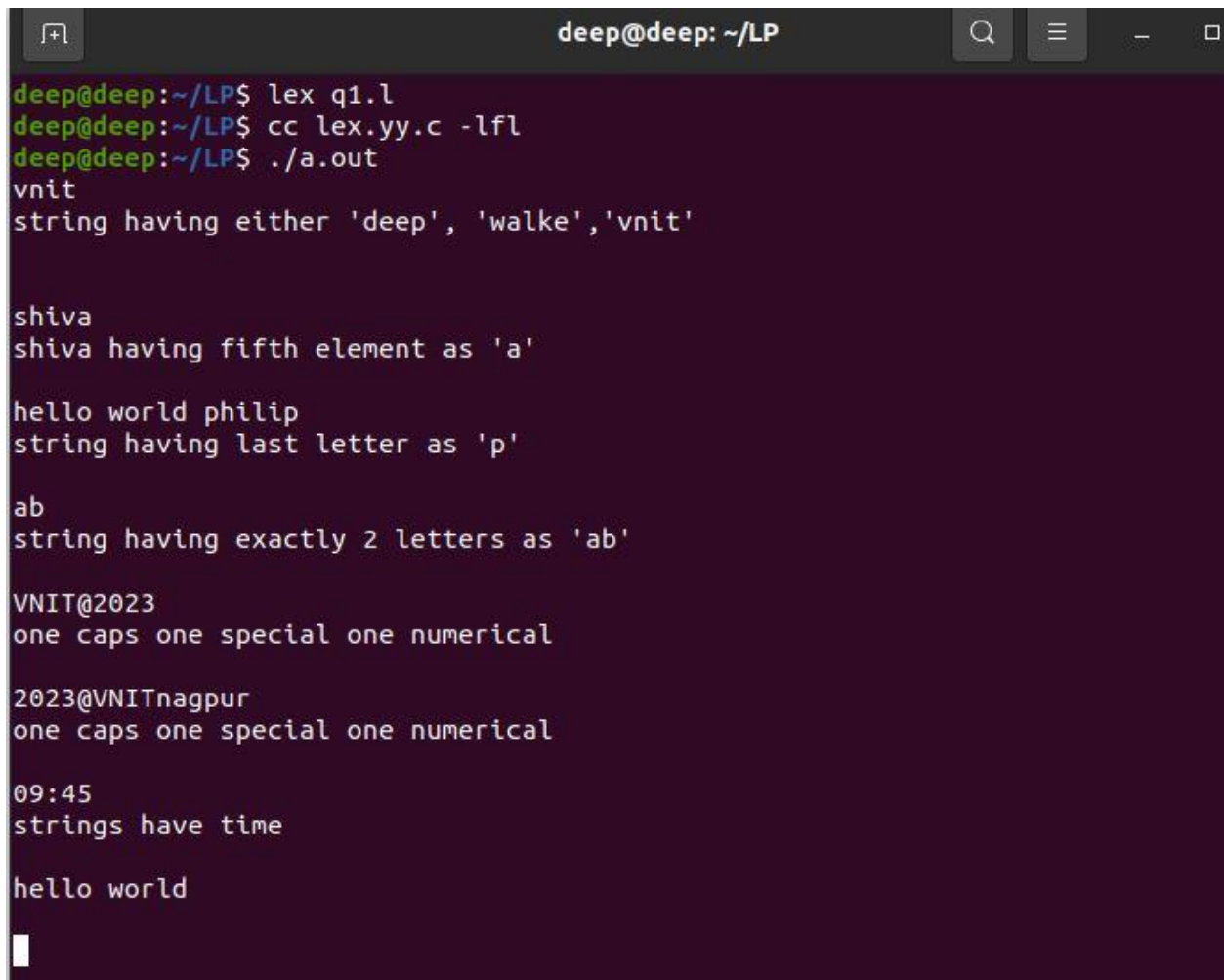
Rules

- I created regex patterns in order to meet the conditions that were given..

Subroutines

- The yywrap() function, which is part of the Lex library, should be redefined. This function is called when the scanner reaches the end of the file. If yywrap() returns 1, the scanner does regular wrap up at the conclusion of the input.
- The lexical analyzer function, yylex(), recognises and returns tokens from the input stream to the parser.

Execution Screenshots

A terminal window with a dark purple background. The title bar shows 'deep@deep: ~/LP'. The terminal contains the following text:

```
deep@deep:~/LP$ lex q1.l
deep@deep:~/LP$ cc lex.yy.c -lfl
deep@deep:~/LP$ ./a.out
vnit
string having either 'deep', 'walke','vnit'

shiva
shiva having fifth element as 'a'

hello world philip
string having last letter as 'p'

ab
string having exactly 2 letters as 'ab'

VNIT@2023
one caps one special one numerical

2023@VNITnagpur
one caps one special one numerical

09:45
strings have time

hello world
```

Question-2:

CODE

q2.l

```
%{
#include "y.tab.h"
void yyerror(char *);
}%
alpha [A-Za-z]
digit [0-9]

%%
"while" {return WHILE;}
"==" {return EQ;}
"=" {return ASIGN;}

[,{}()];] {return *yytext;}
[ \t\n]+ ;

{digit}+ return NUM;
{alpha}({alpha}|{digit})* return ID;

. {printf("\n\nlex err");}
%%
int yywrap()
{
return 1;
}
```

Yacc file

yacc file will be having three parts as shown below and these parts are separated by '%%'

q2.y

```
%{
#include<stdio.h>
void yyerror(char *);
int yylex();
%}

%start whileloop
%token WHILE EQ ASIGN ID NUM
%%
whileloop    : WHILE '(' cond ')' body {printf("parsing successful\n");} ;

cond         : VARIABLE EQ VARIABLE ;

VARIABLE     : ID
              | NUM ;

body         : '{' statement '}'
              | statement ;

statement    : ID ASIGN VARIABLE ';'
              | whileloop ;

%%

void yyerror(char *s)
{
    printf("\nparsing err\n");
}

int main()
{
    yyparse();
    return 0;
}
```

Declarations

- The methods `void yyerror(char*);` and `int yylex();` are included, as well as the relevant C header files.
- The tokens 'WHILE' 'EQ' 'ASSIGN' 'ID' 'NUM' will be stated in the table that will be produced when you run the bison command with the `-d` option.

Productions

- CFG productions with actions are referred to as translation rules.

Subroutines

- `yyerror` is used to determine the nature of an error by passing it a string. If no string is provided, a syntax error will be displayed.
- The main function calls the routine `yyparse()`

Execution Screenshots



```
deep@deep: ~/LP/Q2
deep@deep:~$ cd LP/Q2/
deep@deep:~/LP/Q2$ lex q2.l
deep@deep:~/LP/Q2$ yacc -d q2.y
deep@deep:~/LP/Q2$ cc lex.yy.c y.tab.c
deep@deep:~/LP/Q2$ ./a.out
while(a==b){
a=b;
}
parsing successful
^C
deep@deep:~/LP/Q2$ ./a.out
while(a==b){
    while(c==d)
    {
        c=d;
    }
}
parsing successful
parsing successful
^C
deep@deep:~/LP/Q2$ ./a.out
a=b;

parsing err
deep@deep:~/LP/Q2$ ./a.out
```

Question-3: Calculator Program

CODE

calculator.l

```
%{
/* Definition section */
#include<stdio.h>

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

%%
[0-9]+(\.[0-9]+)?  {
    yylval.dtype=atof(yytext);
    return NUMBER;

}
[\t] ;

[\n] return 0;

. return yytext[0];

%%

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


calculator.y

```
%{
/* Definition section */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int yylex(void);
void yyerror(char *s);

int flag=0;
%}

%union {double dtype;}

%token <dtype> NUMBER

%type <dtype> E ArithmeticExpression

%left '+' '-'

%left '*' '/' '%'

%right '^'

%left '(' ')'

%%

ArithmeticExpression: E{

    printf("\nResult=%lf\n", $$);
    return 0;

};
```

```

E:'+'E {$$=$1+$3;}

|E'-'E {$$=$1-$3;}

|E'*'E {$$=$1*$3;}

|E'/'E {if($3==0)yyerror("Division by 0"),exit(0); else $$=$1/$3;}

|E'%'E {$$=($1*$3)/100;}

|E'^'E {
    if($1==0 && $3<=0)yyerror("Negative or zero exponent not allowed
for base value zero"),exit(0);
    else if(1.0*((int)$3) != $3 && $1<0)yyerror("Exponent cannot be
fraction value if base is negative"),exit(0) ;
    else $$=pow($1,$3);
}

|('E')' {$$=$2;}

| '-' NUMBER {$$=-$2;}

| NUMBER {$$=$1;}

;

%%

//driver code
void main()
{
printf("\nEnter Any Arithmetic Expression \n");

yyparse();

}

```

```

void yyerror(char *s)
{
printf("\n Invalid Expression: %s \n",s);
flag=1;
}

```

The calculator performs arithmetic operations such as '+','-','*','/' (percentage) as well as calculating the exponent using the '^' operator.

This calculator employs associative and precedence rules, with left associative being used for arithmetic operations such as '+','-','*','/' (using percent left) and right associative being used for '^' (using percent right)

Error management is carried out for the following purposes:

- Division operator - If the denominator is 0, the operation is regarded invalid.
- If the base is zero and the exponent is zero or negative, the exponent operator will fail.
- If the exponent is fractional and the base is a negative integer.

Execution Screenshots

```

deep@deep: ~/LP/calculator
deep@deep:~/LP$ cd calculator/
deep@deep:~/LP/calculator$ lex calculator.l
deep@deep:~/LP/calculator$ yacc -d calculator.y
deep@deep:~/LP/calculator$ cc lex.yy.c y.tab.c -ln
deep@deep:~/LP/calculator$ ./a.out

Enter Any Arithmetic Expression
34+42

Result=76.000000
deep@deep:~/LP/calculator$ ./a.out

Enter Any Arithmetic Expression
2*67

Result=134.000000
deep@deep:~/LP/calculator$ ./a.out

Enter Any Arithmetic Expression
57/3

Result=19.000000
deep@deep:~/LP/calculator$ ./a.out

```

```
deep@deep:~/LP/calculator$ ./a.out
```

```
Enter Any Arithmetic Expression
```

```
3+100/25+3*4-11
```

```
Result=8.000000
```

```
deep@deep:~/LP/calculator$ ./a.out
```

```
Enter Any Arithmetic Expression
```

```
12*3-23+32/4+3^3
```

```
Result=48.000000
```

```
deep@deep:~/LP/calculator$ █
```

```
deep@deep:~/LP/calculator$ ./a.out
```

```
Enter Any Arithmetic Expression
```

```
7/0
```

```
Invalid Expression: Division by 0
```

```
deep@deep:~/LP/calculator$ ./a.out
```

```
Enter Any Arithmetic Expression
```

```
0^0
```

```
Invalid Expression: Negative or zero exponent not allowed for base value zero
```

```
deep@deep:~/LP/calculator$ ./a.out
```

```
Enter Any Arithmetic Expression
```

```
(-4)^3.233
```

```
Invalid Expression: Exponent cannot be fraction value if base is negative
```

```
deep@deep:~/LP/calculator$ ./a.out
```