Compiler Lab Assignment - 2

Name: Bitanu Chatterjee

Class: BCSE-III

Roll No: 001810501075

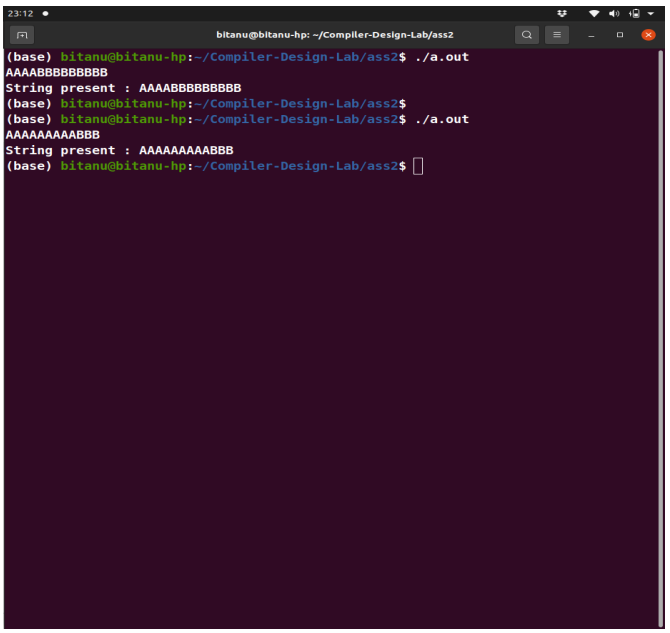
1. Learn how to use YACC (several tutorials are available on the Internet.)

Design a grammar to recognise a string of the form AA...ABB...B, i.e. any number of As followed by any number of Bs. Use LEX or YACC to recognise it. Which one is a better option? Change your grammar to recognise strings with equal numbers of As and Bs - now which one is better?

For A^nB^m

Lex file:

```
%{
  #include<stdio.h>
%}
%%
A+B+   { printf("String present : %s",yytext);}
[ \n\t] {;}
[.]    {printf("Error");}
%%
int yywrap(void)
{
  return 1;
}
int main(void)
{
  yylex();
  return 0;
}
```



Character generator:

```
 #include "q1a.tab.h"

%}

 /* Rule Section */

%%

[aA]        {return A;}

[bB]        {return B;}

\n          {return NL;}

.           {return yytext[0];}

%%

 int yywrap(void)

{

 return 1;

}
```

Yacc file:

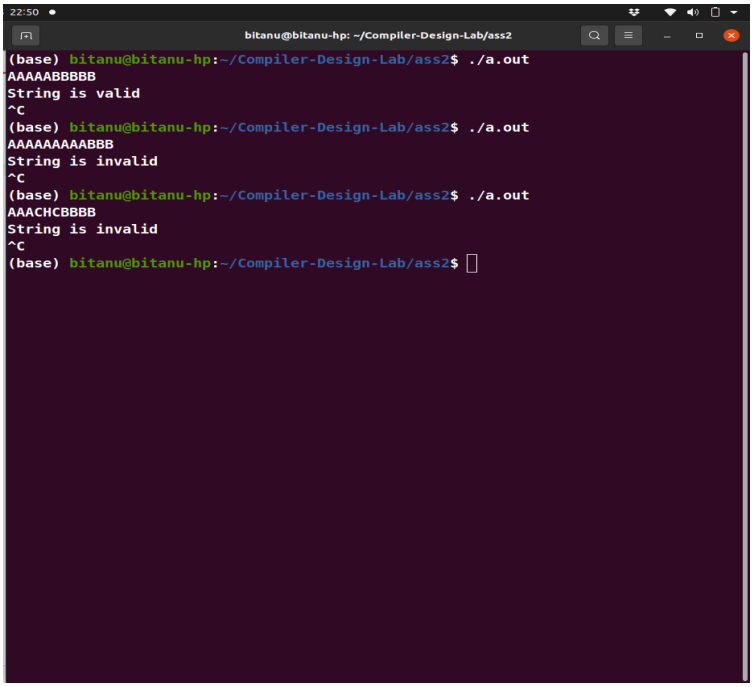
```
%{

 #include<stdio.h>
```

```
 #include<stdlib.h>

%}

 %token A B NL



%%

stmt: S NL  { printf("It is a valid string\n"); exit(0); }

;

S: a b|

;

a: A a | A;

b: b B | B;

%%

 int yyerror(char *msg)

{

 printf("It is an invalid string\n");

 exit(0);

}

 int main()

{

 yyparse();

 return 0;

}
```

For A^nB^n

Lex file:

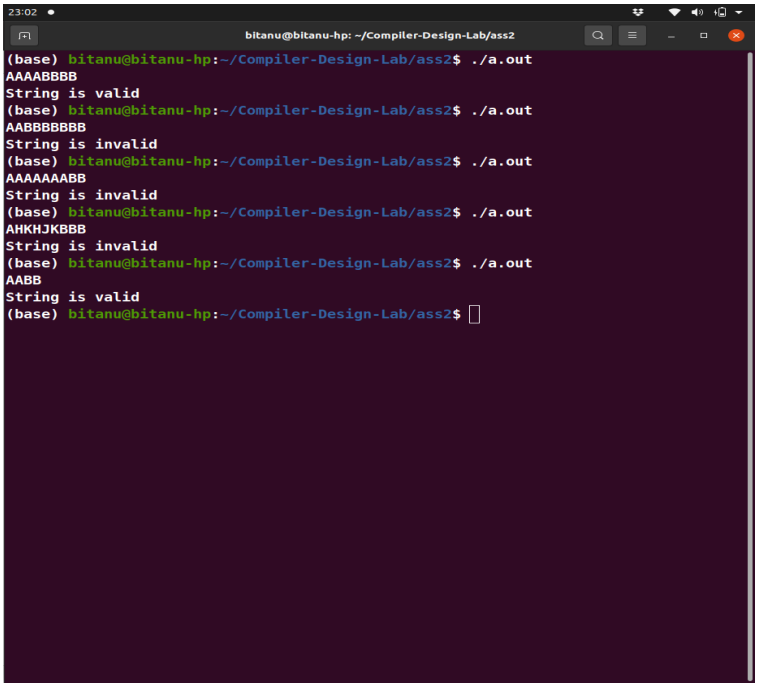
```
%{
  #include<stdio.h>
  int state = 0;
  int cnt = 0;
%}
%%
[ \n\t]     { if(state == 2 && cnt == 0)  printf("String is valid\n"); else printf("String is invalid\n"); state =
0; cnt = 0; }
A         { if(state < 2) {state = 1; cnt++;} else{ state = 3;cnt = 0;}}
B         { if(state == 1){ state = 2; cnt--;} else if(state == 2){if(cnt) cnt--; else{ state = 3; cnt = 0;}}}
.         { state = 3; }
%%
int yywrap(void)
{
  return 0;
}
int main(void)
{
  yylex();
  return 0;
}
```



Yacc file:

```
%{
 #include<stdio.h>
 #include<stdlib.h>
%}
 %token A B NL
 %%
stmt: S NL  { printf("String is valid\n"); exit(0); }
;
S:A S B|
;
%%
 int yyerror(char *msg)
{
 printf("String is invalid\n");
 exit(0);
}
```

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



CONCLUSION:

To check strings of type 'A^nB^m' where n is not equal to m then both YACC and LEX are almost same as per as convenience is concerned.

But in case 'A^nB^n' then YACC is far easier than LEX. In LEX we have to use different states by using the concept of DFA, along with a counter, since only DFA cannot determine the given string. In YACC however we got to use CFG, which solves the issue.

2. Write the lex file and the yacc grammar for an expression calculator. You need to deal with

i) binary operators '+', '*', '-';

ii) unary operator '-';

iii) boolean operators '&', '|'

iv) Expressions will contain both integers and floating point numbers (up to 2 decimal places). Consider left associativity and operator precedence by order of specification in yacc.

Lex file:

```
%{
 /* Definition section */
#include<stdio.h>
#include "q2.tab.h"
extern int yylval;
%}
/* Rule Section */
%%
[0-9]+ {
      yylval=atoi(yytext);
      return NUMBER;
     }
[\t] ;
 [\n] return 0;
. return yytext[0];
 %%
 int yywrap()
{
```

```
return 1;

}
```

Yacc file:

```
%{

 /* Definition section */

 #include<stdio.h>

 int flag=0;

%}

 %token NUMBER

 %left '+' '-'

 %left '*' '/' '%'


%left '&' '|'

 %left '(' ')'


 /* Rule Section */

%%

 ArithmeticExpression: E{

        printf("\nResult=%d\n", $$);

        return 0;

       };


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

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

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

```
|E'/'E {$$=$1/$3;}
 |E'%'E {$$=$1%$3;}

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

|E'|'E {$$=$1|$3;}
 |'('E')' {$$=$2;}

|'('+'NUMBER')' {$$=$3;}

|'('-'NUMBER')' {$$=-$3;}
 |NUMBER {$$=$1;}
 ;
 %%
//driver code
void main()
{
  printf("\nEnter an erithmetic expression\nAllowed operators: Addition, Subtraction, Multiplication, Division, Modulus and Round brackets\n\n");
  yyparse();
 if(flag==0)
 printf("\nValid expression\n\n");
}
 void yyerror()
{
 printf("\nInvalid expression\n\n");
 flag=1;
}
```

OUTPUT:

```
(base) bitanu@bitanu-hp:~/Compiler-Design-Lab/ass2$ ./a.out

Enter an erithmetic expression
Allowed operators: Addition, Subtraction, Multiplication, Division, Modulus and Round brackets

3*24

Result=72

Valid expression

(base) bitanu@bitanu-hp:~/Compiler-Design-Lab/ass2$ ./a.out

Enter an erithmetic expression
Allowed operators: Addition, Subtraction, Multiplication, Division, Modulus and Round brackets

78***^&&^^&

Invalid expression

(base) bitanu@bitanu-hp:~/Compiler-Design-Lab/ass2$ ./a.out

Enter an erithmetic expression
Allowed operators: Addition, Subtraction, Multiplication, Division, Modulus and Round brackets

6+(5|3)

Result=13

Valid expression

(base) bitanu@bitanu-hp:~/Compiler-Design-Lab/ass2$
```