COMPILER DESIGN

ASSIGNMENT 1

Name: **Anirban Das** Class: **BCSE-III** Roll No.: **001910501077** Group: **A3**

Problem 1

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

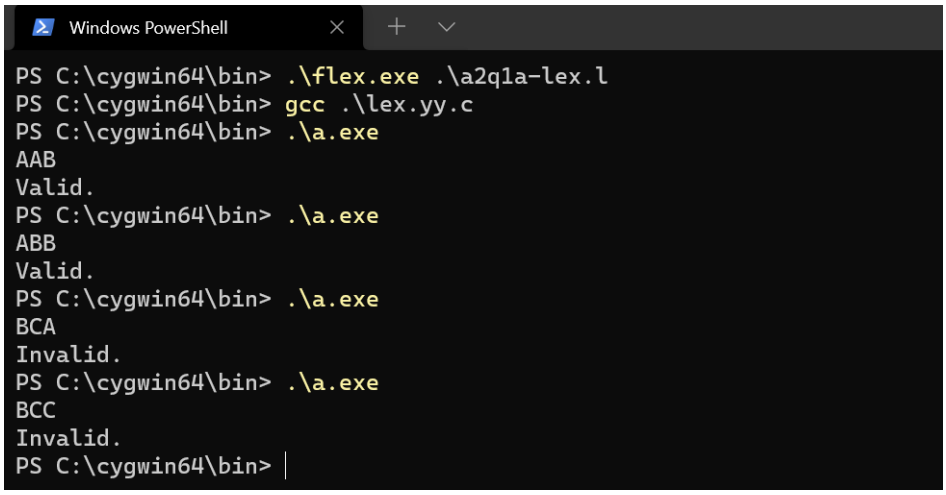
a. **Design a grammar to recognise a string of the form AA...ABB...B, i.e. any number of As followed by any number of Bs. Use LEX or YACC to recognise it. Which one is a better option?**

LEX Code

```
%{
#include <stdio.h>
%}

%%
\n {return 0;}
 [A]+[B]+ {return 1;}
. {;}
%%

int yywrap() {}

int main() {
      int ok = yylex();
      if (ok) {
            printf("Valid.\n");
```

```
    } else {
        printf("Invalid.\n");
    }

    return 0;
}
```

Output



YACC Code

.y file

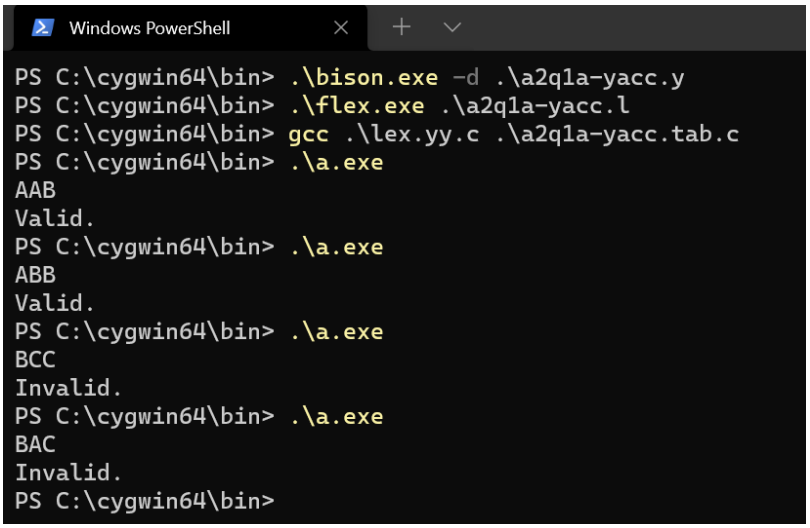
```
%{
#include <stdio.h>
#include <stdlib.h>

int yyerror(char *s);
int yylex();
%}
%start string
%token A B

%%
string : as bs ;
```

```
as : A
            | as A ;
bs : B
            | bs B;
%%

int yyerror(char *s) {
    printf("Invalid.\n");
    exit(0);
}

int main() {
    yyparse();
    printf("Valid.\n");
    return 0;
}
```

.l file

```
%{
#include "prob1a.tab.h"
int yyerror(char *s);
int yylex();
%}

%%
A {return A;}
B {return B;}
\n {return 0;}
. {return yytext[0];}
%%
int yywrap() {return 1;}
```

Output



For this problem, using LEX was a better option because it can be done using a simple regex, while in YACC, we've to construct a CFG.

b. Change your grammar to recognise strings with equal numbers of As and Bs - now which one is better?

YACC Code

.y file

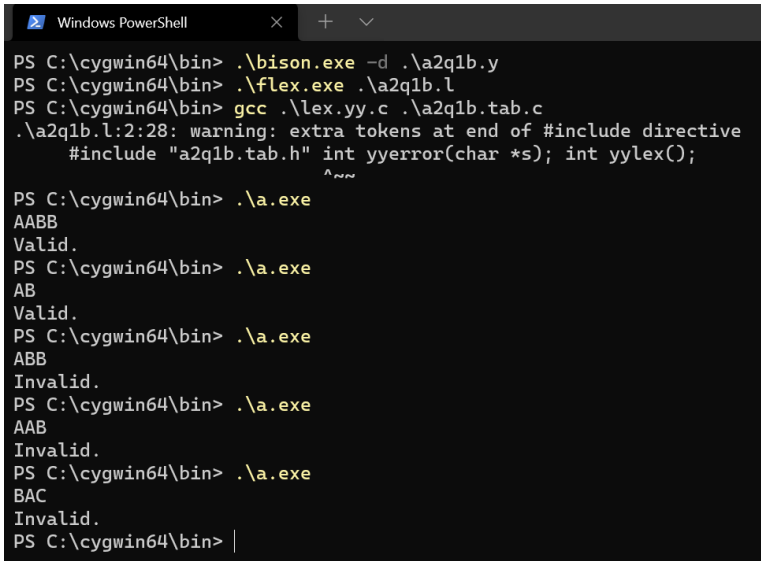
```
%{
#include <stdio.h>
#include <stdlib.h>

int yyerror(char *s); int
yylex();
%}
%start string
%token A B

%%
```

```
  string : A string B | A B ;
%%

int yyerror(char *s) {
      printf("Invalid.\n"); exit(0);
}

int main() {
      yyparse();
      printf("Valid.\n"); return
      0;
}
```

.l file

```
%{
#include "prob1b.tab.h" int
yyerror(char *s); int yylex();
%} %%
A {return A;}
B {return B;}
\n {return 0;}
. {return yytext[0];} %%
int yywrap() {return 1;}
```

Output



For this problem, using YACC was a better option because, in LEX, we had to keep track of counts of ‘A’ and ‘B’s when occurring in a continuous stream and then check if the count is equal or not, which seems a cumbersome process, whereas it can be done using YACC through one CFG.

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.

.y file

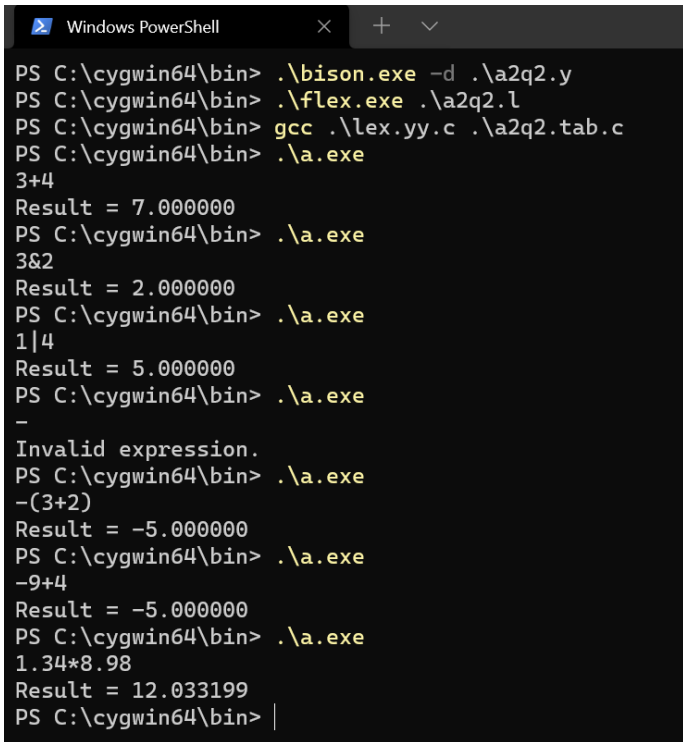
```
%{
#include <stdio.h>
#include <stdlib.h>
int yylex(void);
int yyerror(char *);
%}
%start Expression
%union {float num;}
%token <num> NUMBER
%type <num> Expression E
%left '+' '-'
%left '*'
%left '&' '|'
%left '(' ')'

%%
Expression: E {printf("Result = %f\n", $$); return 0;}; E: E '+' E {$$ =
$1 + $3;}
 | E '-' E {$$ = $1 - $3;}
 | E '*' E {$$ = $1 * $3;}
```

```
 | E '&' E {$$ = (int)($1) & (int)($3);}
 | E '|' E {$$ = (int)($1) | (int)($3);}
 | '-' E {$$ = -$2;}
 | '(' E ')' {$$ = $2;}
 | NUMBER {$$ = $1;}
;
%%

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

int yyerror(char* s){
    printf("Invalid expression.\n");
    exit(0);
}
```

.l file

```
%{
#include <stdio.h>
#include "prob2.tab.h"
%}

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

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

Output


Windows PowerShell
PS C:\cygwin64\bin> .\bison.exe -d .\a2q2.y
PS C:\cygwin64\bin> .\flex.exe .\a2q2.l
PS C:\cygwin64\bin> gcc .\lex.yy.c .\a2q2.tab.c
PS C:\cygwin64\bin> .\a.exe
3+4
Result = 7.000000
PS C:\cygwin64\bin> .\a.exe
3&2
Result = 2.000000
PS C:\cygwin64\bin> .\a.exe
1|4
Result = 5.000000
PS C:\cygwin64\bin> .\a.exe
-
Invalid expression.
PS C:\cygwin64\bin> .\a.exe
-(3+2)
Result = -5.000000
PS C:\cygwin64\bin> .\a.exe
-9+4
Result = -5.000000
PS C:\cygwin64\bin> .\a.exe
1.34*8.98
Result = 12.033199
PS C:\cygwin64\bin>