

LAB REPORT

COMPILER DESIGN

NIKHIL BADYAL

001810501069

BCSE III(A2)

Q1a. 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.

q1a.l

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

%%

\n    { return 1; }

[a]+[b]+\n { return 0; } /*one or more matches and * means
zero or more matches*/

exit { exit(0); }

.      ;

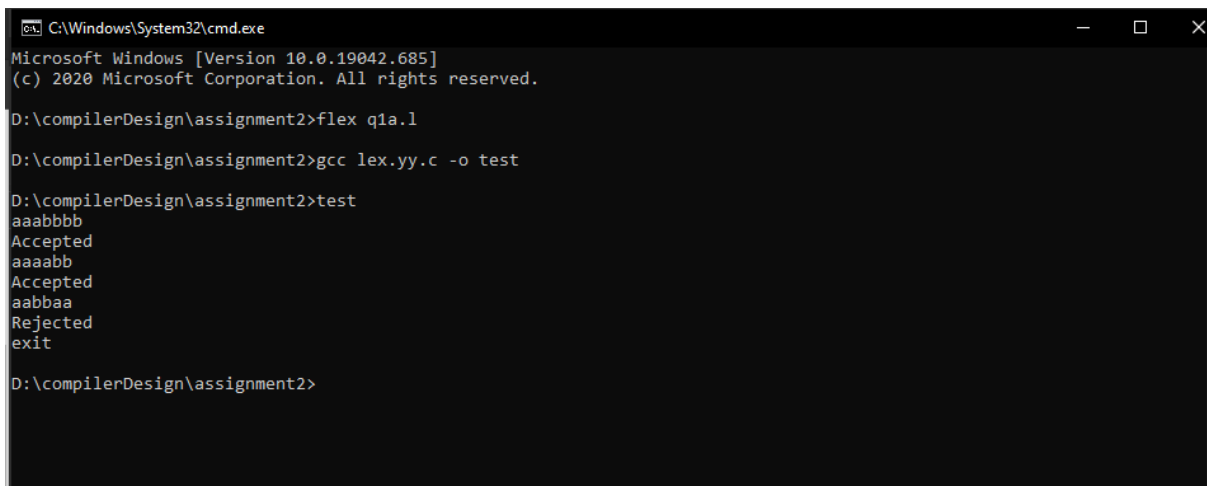
%%

int yywrap() { return 0; }

int main(){
    int temp;
    while(1) {
        temp = yylex(); /*reads characters from file* file
pointer called yyin.*/
        001810501069
```

```
        if(!temp) printf("Accepted!! It is of form  
a^nb^m\n");  
  
        else printf("Rejected!! It is NOT of form a^nb^m\n");  
  
    }  
  
    return 0;  
  
}
```

OUTPUT



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19042.685]
(c) 2020 Microsoft Corporation. All rights reserved.

D:\compilerDesign\assignment2>flex q1a.l

D:\compilerDesign\assignment2>gcc lex.yy.c -o test

D:\compilerDesign\assignment2>test
aaabbbb
Accepted
aaaabb
Accepted
aabbaa
Rejected
exit

D:\compilerDesign\assignment2>
```

q1a.y

```
%{  
#include<stdio.h>  
#include<stdlib.h>  
int yyerror (char *s);  
int yylex();  
%}  
%token a b  
  
%%  
  
s : as bs ;  
as  : a | as a ;  
bs  : b | bs b ;  
  
%%  
  
int yywrap() { return 1; } /*yywrap is called at the end of the file*/  
int yyerror(char *s) {  
    printf("Rejected!! It is NOT of form a^nb^m");  
    exit(0);  
}
```

```

}

int main() {
    yyparse(); /* yyparse() reads a stream of token/value pairs
from yylex(), */
    printf("Accepted!! It is of form a^n b^m\n");
    return 0;
}

```

OUTPUT

```
D:\compilerDesign\assignment2>bison -d q1a.y  
D:\compilerDesign\assignment2>flex q1.1  
  
D:\compilerDesign\assignment2>gcc lex.yy.c q1a.tab.c -o test  
q1.1:2:24: fatal error: ques1a.tab.h: No such file or directory  
compilation terminated.  
  
D:\compilerDesign\assignment2>bison -d q1a.y  
D:\compilerDesign\assignment2>flex q1.1  
  
D:\compilerDesign\assignment2>gcc lex.yy.c q1a.tab.c -o test  
  
D:\compilerDesign\assignment2>test  
aabbbb  
Accepted  
  
D:\compilerDesign\assignment2>test  
aba  
Rejected  
  
D:\compilerDesign\assignment2>test  
aaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb  
Accepted  
  
D:\compilerDesign\assignment2>
```

Conclusion- I think the lex version is much simpler, so I would use lex.

Q1b. Change your grammar to recognise strings with equal numbers of As and Bs - now which one is better?

q1b.l

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

%token a b

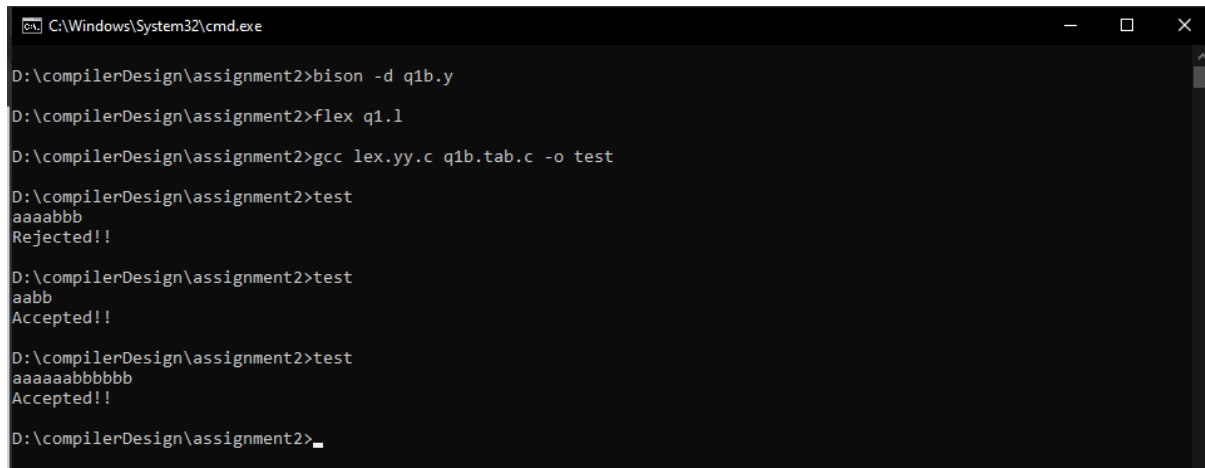
%%

s : a s b | a b ;
%%

int yywrap() { return 1; }
int yyerror(char *s) {
    printf("Rejected!! It is NOT of form a^nb^n");
    exit(0);
}
```

```
int main() {  
    yyparse();  
    printf("Accepted!! It is of form a^nb^n\n");  
    return 0;  
}
```

OUTPUT



```
C:\Windows\System32\cmd.exe  
D:\compilerDesign\assignment2>bison -d q1b.y  
D:\compilerDesign\assignment2>flex q1.l  
D:\compilerDesign\assignment2>gcc lex.yy.c q1b.tab.c -o test  
D:\compilerDesign\assignment2>test  
aaaabbb  
Rejected!!  
D:\compilerDesign\assignment2>test  
aabb  
Accepted!!  
D:\compilerDesign\assignment2>test  
aaaaaabbabbbb  
Accepted!!  
D:\compilerDesign\assignment2>
```

Conclusion-The best Lex can do is as done in previous part, and then count the 'a's and 'b's in the action to make sure they match.

Yacc e.g.:

```
a_b : 'a' a_b 'b' | 'a' 'b' ;
```

I would use the Yacc version.

Q2. Write the lex file and the yacc grammar for an expression calculator. You need to deal with

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

ii) Uniary 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.

q2.l

```
%{  
    #include "ques2.tab.h"  
    extern int yylval;  
}%  
  
%%  
  
[0-9]+ {yylval = atoi(yytext); return id;}  
\n { return 0;}  
[' \"\t]+ {;}  
. { return yytext[0];}  
  
%%
```

q2.y


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

```
%token id  
%left '+' '-'  
%left '*' '/'  
%left '(' ')'  
%left neg
```

```
%%
```

```
E1 : E { printf("The resultant value is %d\n", $1);}
```

```
E  : E '+' E { $$ = $1 + $3;}  
    | E '-' E { $$ = $1 - $3;}  
    | E '*' E { $$ = $1 * $3;}  
    | E '/' E { $$ = $1 / $3;}  
    | '-' E %prec neg{ $$ = -1 * $2;}  
    | '(' E ')' { $$ = $2;}  
    | id { $$ = $1;}
```

```
        ;  
    %%  
  
int main(){  
    yyparse();  
    return 0;  
}  
  
int yywrap(){}  
  
int yyerror( char* s){  
    printf("You entered an invalid expression\n");  
    exit(0);  
}
```

OUTPUT

```
D:\compilerDesign\assignment2>bison -d q2.y
D:\compilerDesign\assignment2>flex q2.l
D:\compilerDesign\assignment2>gcc lex.yy.c q2.tab.c -o test
D:\compilerDesign\assignment2>test
-6 * -11
The resultant value is 66

D:\compilerDesign\assignment2>test
98 - 90 + 5 * 70
The resultant value is 358

D:\compilerDesign\assignment2>test
68 * 69
The resultant value is 4692

D:\compilerDesign\assignment2>_
```