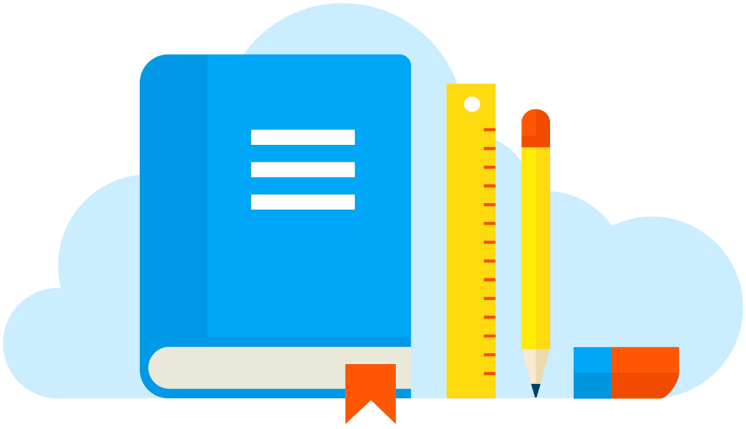


Compiler Assignment 2

Date: 06/04/2021

Soumalya Kundu

Roll: 001810501033

Question 1a:

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?

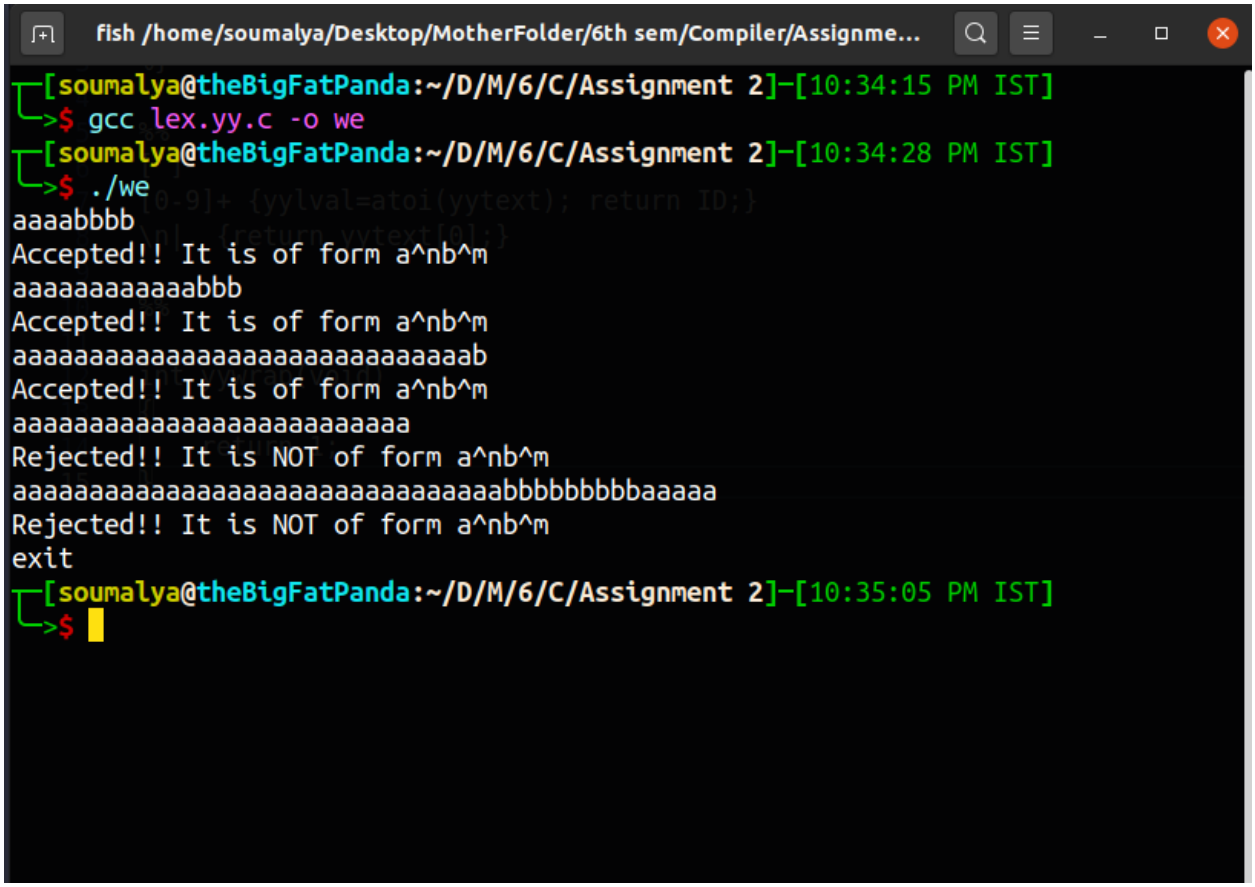
- **Code:**

  FileName q1a.l:

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

```
%%
\n      { return 1; }
[a]+[b]+\n  {  return 0; }
exit  { exit(0); }
.       ;

%%
int yywrap() { return 0; }

int main(){
        int temp;
        while(1) {
                temp = yylex();
                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:**



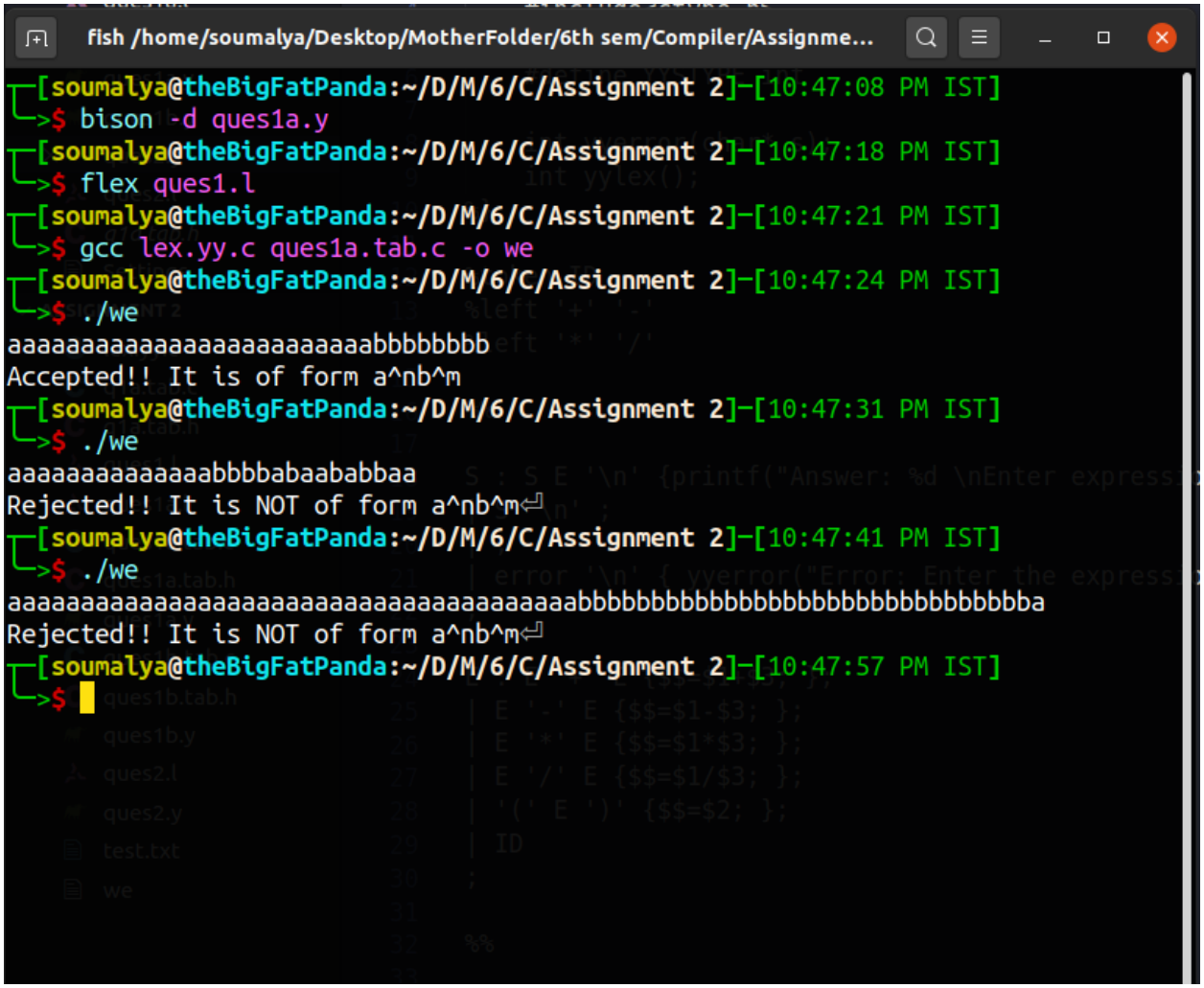
- **Code:**

FileName 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; }
int yyerror(char *s) {
        printf("Rejected!! It is NOT of form a^nb^m");
        exit(0);
}
int main() {
        yyparse(); // reads a token value pair from yylex()
        printf("Accepted!! It is of form a^nb^m\n");
        return 0;
}
```

- **Output:**



- **Conclusion:**

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

Question 1b:

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

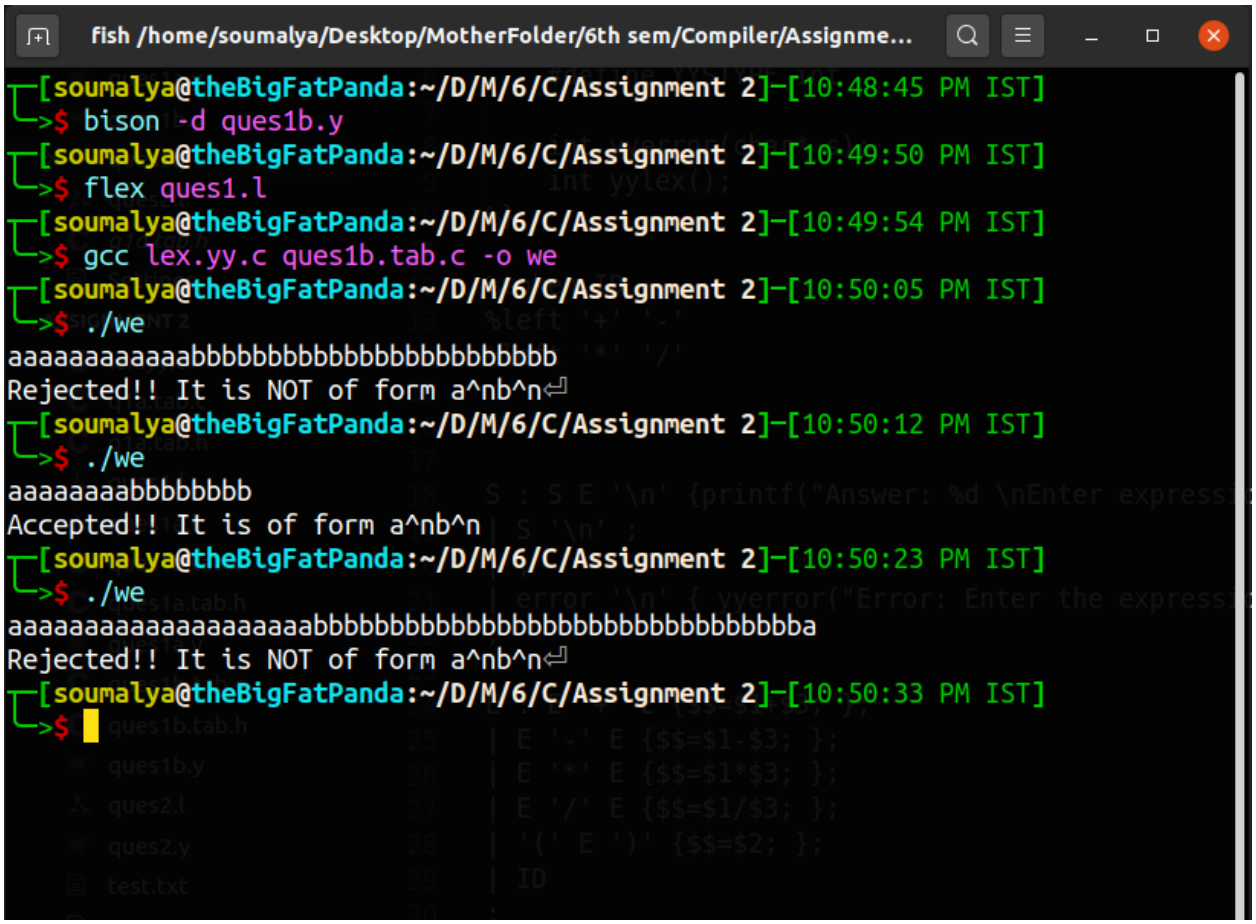
- **Code:**

  FileName q1b.y:

```
%{
#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:**



- **Conclusion:**

  To check strings of type ‘A nB m ’ where n is not equal to m then both YACC and LEX are almost the same as per 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.

Question 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.

- **Code:**

  FileName q2.l:

```
%{
 #include "ques2.tab.h"
 extern int yylval;
%}


%%


[0-9]+ {yylval = atoi(yytext); return id;}
```

```
\n { return 0;}
[' '\t]+ {;}
. { return yytext[0];}

%%
```

- **Code:**

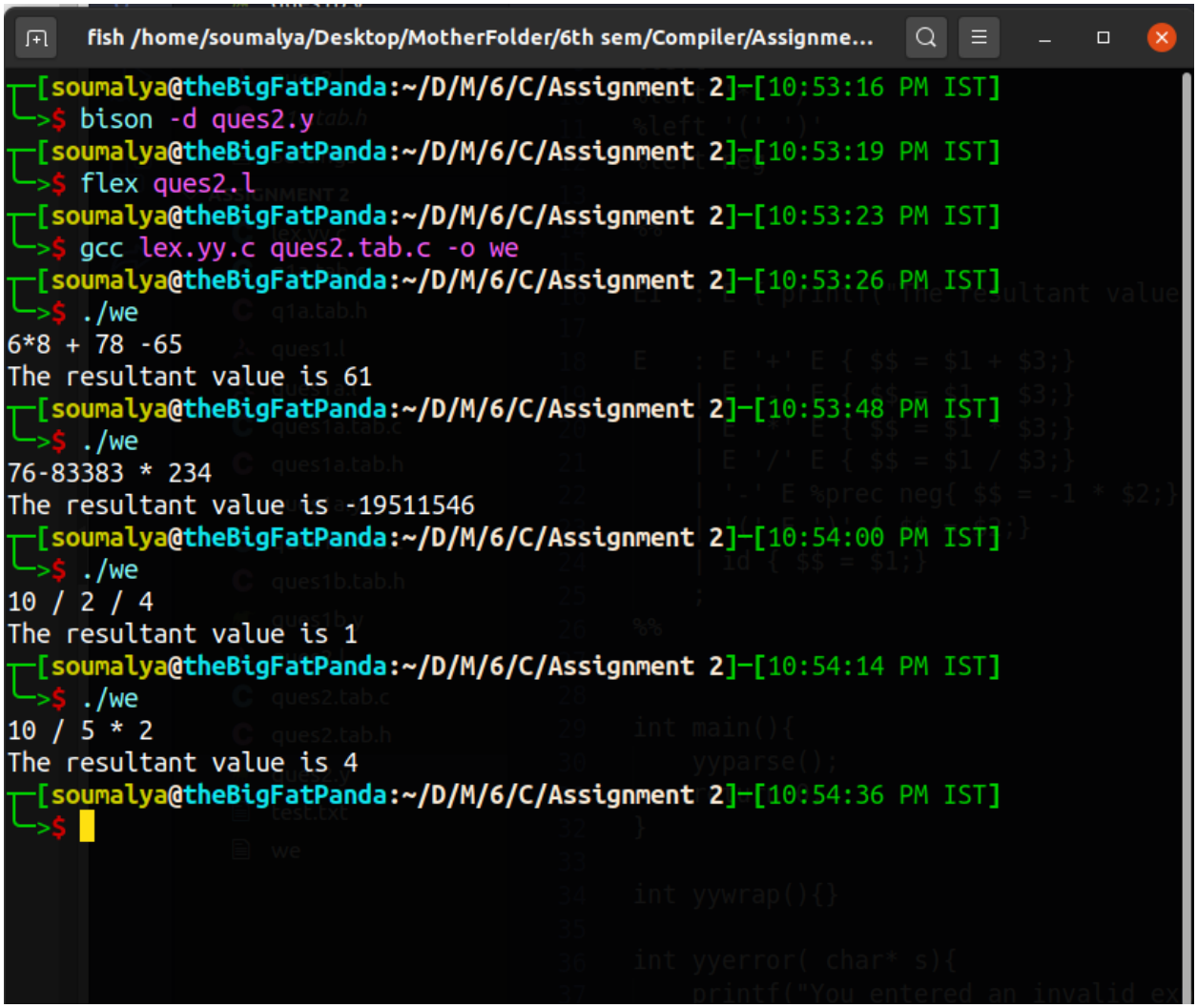
FIlename q2.y:

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

#define YYSTYPE int

int yyerror(char* s);
int yylex();
%}
```

```
%token ID

%left '+' '-'

%left '*' '/'



%%



S : S E '\n' {printf("Answer: %d \nEnter expression\n",$2);}

| S '\n' ;

| ;

| error '\n' { yyerror("Error: Enter the expression\n" );}

;



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

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

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

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

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

```
| ID
;



%%



int main()
{
        printf("Enter the expression:\n");
         yyparse();
}
        int yyerror (char *s)
{
         printf("%s\n", s);
          return 0;
}
```

- **Output:**


fish /home/soumalya/Desktop/MotherFolder/6th sem/Compiler/Assignme...
[soumalya@theBigFatPanda:~/D/M/6/C/Assignment 2]-[10:53:16 PM IST]
>$ bison -d ques2.y
[soumalya@theBigFatPanda:~/D/M/6/C/Assignment 2]-[10:53:19 PM IST]
>$ flex ques2.l
[soumalya@theBigFatPanda:~/D/M/6/C/Assignment 2]-[10:53:23 PM IST]
>$ gcc lex.yy.c ques2.tab.c -o we
[soumalya@theBigFatPanda:~/D/M/6/C/Assignment 2]-[10:53:26 PM IST]
>$ ./we
6*8 + 78 -65
The resultant value is 61
[soumalya@theBigFatPanda:~/D/M/6/C/Assignment 2]-[10:53:48 PM IST]
>$ ./we
76-83383 * 234
The resultant value is -19511546
[soumalya@theBigFatPanda:~/D/M/6/C/Assignment 2]-[10:54:00 PM IST]
>$ ./we
10 / 2 / 4
The resultant value is 1
[soumalya@theBigFatPanda:~/D/M/6/C/Assignment 2]-[10:54:14 PM IST]
>$ ./we
10 / 5 * 2
The resultant value is 4
[soumalya@theBigFatPanda:~/D/M/6/C/Assignment 2]-[10:54:36 PM IST]
>$