Compiler Design Report

Shuvayan Ghosh Dastidar
001810501044
BCSE-III
Third Year Second Semester
2021

Assignment 2
YACC and using it to parse different grammars.

Problem Statement

1. 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?

2. 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.

Question 1

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?

First part is recognising a grammar of the type A..A..B..B ,ie, that is any number of As followed by any number of Bs. For this we can design a regular expression to recognise strings of the type which is implemented in a aabb.l Lex file.

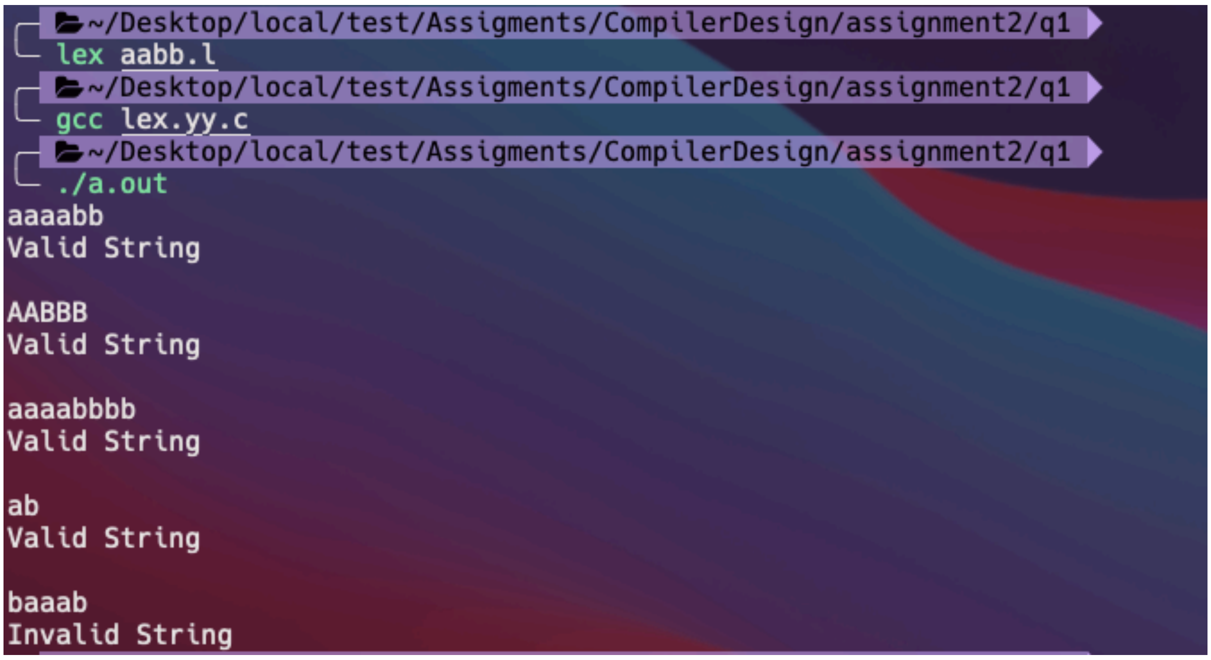
aabb.l

```
%%

[Aa]+[Bb]+ {printf("Valid String\n");}
.         {printf("Invalid String\n”); exit(0);}
%%
int yywrap(){
    return 0;
}

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

The steps for compilation are

- lex aabb.l
- gcc lex.yy.c
- ./a.out

The output of the program is shown below



```
┌ ~/Desktop/local/test/Assigments/CompilerDesign/assignment2/q1
└ lex aabb.l
┌ ~/Desktop/local/test/Assigments/CompilerDesign/assignment2/q1
└ gcc lex.yy.c
┌ ~/Desktop/local/test/Assigments/CompilerDesign/assignment2/q1
└ ./a.out
aaaabb
Valid String

AABBB
Valid String

aaaabbbb
Valid String

ab
Valid String

baaab
Invalid String
```

It's difficult to parse strings of type A_nB_n by just regular expressions so we need some kind of grammar to parse this string with equal number of As followed by equal number of Bs.

The grammar for this parsing is -

SS -> S NL
S -> A S B

Where NL represents a new line character '\n'.

It has two files ab.l and ab.y. One for parsing the tokens and the other for parsing the grammar and checking if the tokens are accepted by the grammar.

ab.l

```
%{
  #include "y.tab.h"
%}

%%
[aA] {return A;}
[bB] {return B;}
\n {return NL;}
.  {return yytext[0];}
%%

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

ab.y

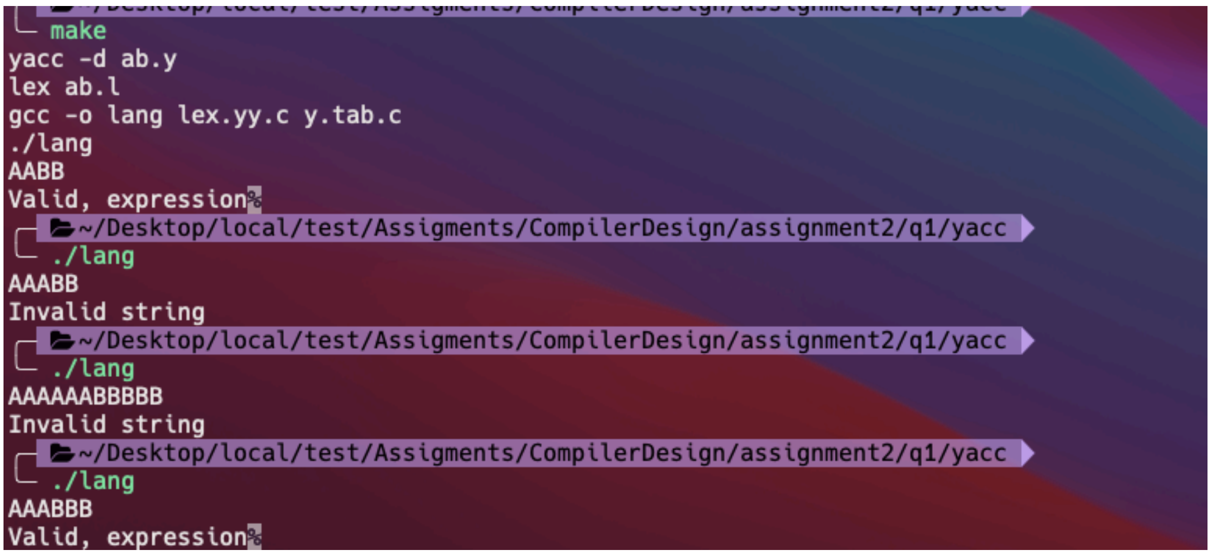
```
%{
     #include <stdio.h>
     #include <math.h>
     #include <stdlib.h>
     int yylex();
     void yyerror(char *);
 %}

%token A B NL

 %%
ss : S NL  { printf("Valid, expression"); exit(0); }
S : A S B |
 %%

int main(){
     yyparse();
}

void yyerror(char *s){
     printf("Invalid string\n");
}
```

There is a makefile for compiling and running the program.

Makefile

```
lang :
	yacc -d ab.y
	lex ab.l
	gcc -o $@ lex.yy.c y.tab.c
	./$@


.PHONY: clean

# ~ means to ignore all the warings if file is not present

clean:
	rm -f *~ core $(INCDIR)/*~
	rm lex.yy.c y.tab.c y.tab.h
	rm -f lang
```

Here is the output of the above program,



```
└─ make
yacc -d ab.y
lex ab.l
gcc -o lang lex.yy.c y.tab.c
./lang
AABB
Valid, expression%
┌─ ~/Desktop/local/test/Assigments/CompilerDesign/assignment2/q1/yacc
└─ ./lang
AAABB
Invalid string
┌─ ~/Desktop/local/test/Assigments/CompilerDesign/assignment2/q1/yacc
└─ ./lang
AAAAAABBBBB
Invalid string
┌─ ~/Desktop/local/test/Assigments/CompilerDesign/assignment2/q1/yacc
└─ ./lang
AAABBB
Valid, expression%
```

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.?

The question requires to write a lex file and an yacc file for parsing arithmetic expressions and perform calculations based on them.

For recognizing numbers of the form decimal and fractional types, we define a number.h file where we define the properties of the tokens parsed by the lex file.

number.h

```
#ifndef DEFN
#define DEFN
struct number{

        int ival;
        double fval;
        char type;
};

#endif
```

The type stores 1 for integer tokens and 2 for fractional tokens. fval stores the value of fractional tokens whereas rival stores the value of the integer tokens.

Then we have the lex file for recognizing different tokens in the given input.

cal.l

```
%{
    #include<math.h>
    #include"number.h"
    char INT_TYPE=1;
    char FLOAT_TYPE=2;
    int error_flag=0;
    #include"y.tab.h"
%}
%%
[0-9]+\.[0-9][0-9][0-9]+  {
                            printf("Float type match with more than 2 digs %s\n",
yytext);
                            printf("ONLY 2 DECIMAL DIGITS SUPPORTED\n");
                            return ERROR;
                         }
[0-9]+\.[0-9][0-9]? {
                            yylval.p.fval = atof(yytext);
                            yylval.p.type = 2;
                            yylval.p.ival = 0;
                            return num;
                        }
[0-9]+                  {
                            yylval.p.ival = atoi(yytext);
                            yylval.p.fval = 0;
                            yylval.p.type =1;
                            return num;
                        }
[ \t]*\+[ \t]* { return PLUS;}
[ \t]*\-[ \t]* { return MINUS;}
[ \t]*\*[ \t]* { return MUL;}
[ \t]*\|[ \t]* { return OR;}
[ \t]*\&[ \t]* { return AND;}
[\t] ;
\n return 0;
. return yytext[0];
%%
int yywrap(){
    return 0;
}
```

The lex file parses and returns the tokens or stores the value for further computation as required.

The tokens are passed on to a cal.y file.

The left associativity is done by the %left symbol in yacc and lower symbols have more precedence.

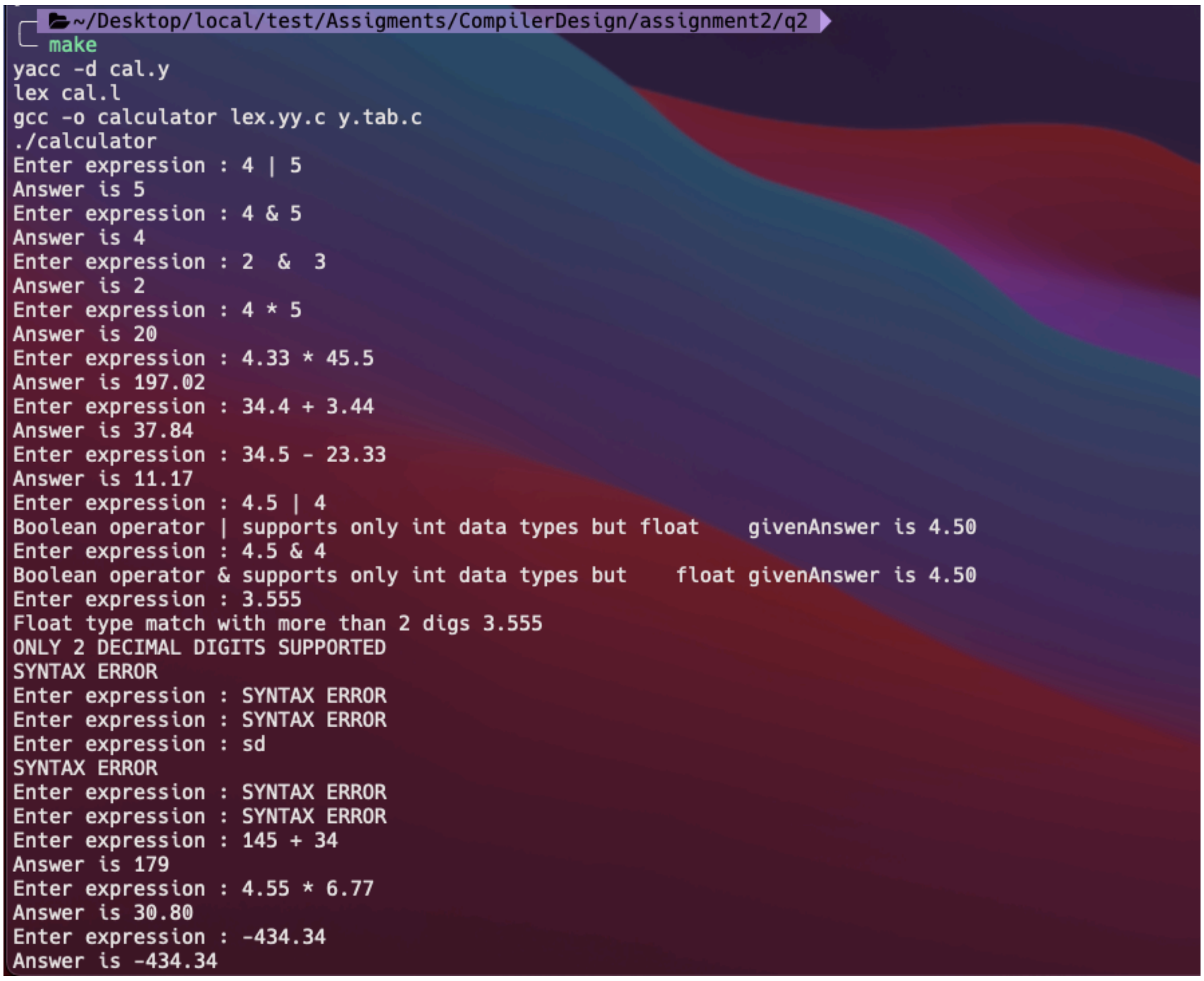
cal.y

```
%{
    #include <stdio.h>
    #include <math.h>
    int yylex();
    void yyerror(char *);
    #include "number.h"
    extern char FLOAT_TYPE;
    extern char INT_TYPE;
    extern int error_flag;
%}

%union {
    struct number p;
}
%token<p> num
%token PLUS MINUS
%token MUL AND OR
%token ERROR
%left OR AND
%left PLUS MINUS
%left MUL '/'
%nonassoc uminu
%type<p>exp

%%


ss: exp  {
                if ( $1.type == INT_TYPE ){
                    if(!error_flag) printf("Answer is %d\n" , $1.ival );
                }
                else{
                    if(!error_flag) printf("Answer is %.2f\n", $1.fval );

                }
            }


exp: exp PLUS exp      {

                        if ( $1.type == FLOAT_TYPE || $3.type == FLOAT_TYPE ) $$.type =
FLOAT_TYPE;
                        else $$.type = INT_TYPE;

                        if ($$.type == FLOAT_TYPE){
                            $$.fval = $1.fval + $3.fval + $1.ival + $3.ival;
                            $$.ival = 0;
                        }
                        else {
                            $$.ival = $1.ival + $3.ival;
                            $$.fval = 0;
                        }
                    }
    | exp MINUS exp     {
                         if ( $1.type == FLOAT_TYPE || $3.type == FLOAT_TYPE ) $$.type =
FLOAT_TYPE;
                         else $$.type = INT_TYPE;

                         if ($$.type == FLOAT_TYPE) $$.fval = $1.ival + $1.fval - ( $3.ival +
$3.fval ), $$.ival = 0;
                         else $$.ival = $1.ival - $3.ival, $$.fval = 0;
                    }
    | exp MUL exp     {
                         if ( $1.type == FLOAT_TYPE || $3.type == FLOAT_TYPE ) $$.type =
FLOAT_TYPE;
                         else $$.type = INT_TYPE;

                         if ($$.type == FLOAT_TYPE) $$.fval = ($1.ival + $1.fval) * ($3.ival
+ $3.fval), $$.ival = 0;
                         else $$.ival = $1.ival * $3.ival, $$.fval = 0;
                    }
    | exp OR exp     {
                         if ( ($1.type != INT_TYPE) || ($3.type != INT_TYPE )){
                             printf("Boolean operator | supports only int data types but
float    given");
                         }
                         else{
                             $$.type = INT_TYPE;
                             $$.fval = 0; $$.ival = $1.ival |  $3.ival;
                         }
                    }
    | exp AND exp      {
                          if ( ($1.type != INT_TYPE) || ($3.type != INT_TYPE )){
                              printf("Boolean operator & supports only int data types but
float given");
                          }
                          else{
                              $$.type = INT_TYPE;
                              $$.fval = 0; $$.ival = $1.ival &  $3.ival;
                          }
                    }
    | MINUS exp         {
                         $$.type = $2.type;
                         if ( $$.type == FLOAT_TYPE ) $$.fval = -$2.fval, $$.ival = 0;
                         else $$.ival = -$2.ival, $$.fval = 0;

                    }
    | num            ;
    | '(' exp')'     {
                         $$.type = $2.type;
                         if ( $$.type == FLOAT_TYPE ) $$.fval = $2.fval, $$.ival = 0;
                         else $$.ival = $2.ival, $$.fval = 0;
                    }

%%

int main(){
    do{
        error_flag = 0;
        printf("Enter expression : ");
        yyparse();
    }while(1);
}


void yyerror(char * s){
    error_flag=1;
    printf("SYNTAX ERROR\n");
}
```

This is compiled by a Makefile

```
calculator :
        yacc -d cal.y
        lex cal.l
        gcc -o $@ lex.yy.c y.tab.c
        ./$@


.PHONY: clean

# ~ means to ignore all the warings if file is not present

clean:
        rm -f *~ core $(INCDIR)/*~
        rm lex.yy.c y.tab.c y.tab.h
        rm -f calculator
```

The output of the calculator is shown below.



```
~/Desktop/local/test/Assigments/CompilerDesign/assignment2/q2
make
yacc -d cal.y
lex cal.l
gcc -o calculator lex.yy.c y.tab.c
./calculator
Enter expression : 4 | 5
Answer is 5
Enter expression : 4 & 5
Answer is 4
Enter expression : 2  &  3
Answer is 2
Enter expression : 4 * 5
Answer is 20
Enter expression : 4.33 * 45.5
Answer is 197.02
Enter expression : 34.4 + 3.44
Answer is 37.84
Enter expression : 34.5 - 23.33
Answer is 11.17
Enter expression : 4.5 | 4
Boolean operator | supports only int data types but float    givenAnswer is 4.50
Enter expression : 4.5 & 4
Boolean operator & supports only int data types but    float givenAnswer is 4.50
Enter expression : 3.555
Float type match with more than 2 digs 3.555
ONLY 2 DECIMAL DIGITS SUPPORTED
SYNTAX ERROR
Enter expression : SYNTAX ERROR
Enter expression : SYNTAX ERROR
Enter expression : sd
SYNTAX ERROR
Enter expression : SYNTAX ERROR
Enter expression : SYNTAX ERROR
Enter expression : 145 + 34
Answer is 179
Enter expression : 4.55 * 6.77
Answer is 30.80
Enter expression : -434.34
Answer is -434.34
```