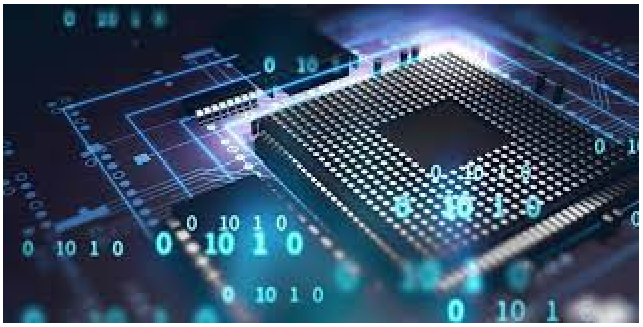
System Programming Practicals



Khushal Sachdeva
College roll no: 20/88044
University roll no: 20003570032
B.Sc. Hons. Computer Science

```
lex file.l
gcc lex.yy.c -lfl
./a.out

Input ←
Ctrl + d  // yylex() exit point
Output →
```

```
lex file.l
yacc file.y
cc lex.yy.c  y.tab.c
./a.out
```

1. Write a Lex program to count the number of lines and characters in the input file.

2. Write a Lex program that implements the Caesar cipher: it replaces every letter with the one three letters after in alphabetical order, wrapping around at Z. e.g. a is replaced by d, b by e, and so on z by c.

3. Write a Lex program that finds the longest word (defined as a contiguous string of upper and lower-case letters) in the input.

4. Write a Lex program that distinguishes keywords, integers, floats, identifiers, operators, and comments in any simple programming language.

5. Write a Lex program to count the number of identifiers in a C file.

6. Write a Lex program to count the number of words, characters, blank spaces and lines in a C file.

7. Write a Lex specification program that generates a C program which takes a string “abcd” and prints the following output.
abcd abc ab a

8. A program in Lex to recognize a valid arithmetic expression.

9. Write a YACC program to find the validity of a given expression (for operators + - * and /)

10. A Program in YACC which recognizes a valid variable which starts with a letter followed by a digit. The letter should be in lowercase only.

11. A Program in YACC to evaluate an expression (simple calculator program for addition and subtraction, multiplication, division).

12. Program in YACC to recognize the strings “ab”, “aabb”, ”aaabbb”,… of the language (a^n b^n , n>=1).

13. Program in YACC to recognize the language (a^n b , n>=10). (Output to say input is valid or not)

1. Write a Lex program to count the number of lines and characters in the input file.

```
%{
      #include<stdio.h>
      int lines = 0;
      int characters = 0;
%}

%%
([ ])+ characters++;
\n lines++;
%%

int main()
{
      printf("Opening the file.....");
      extern FILE *yyin;
      yyin = fopen("file.text", "r");
      yylex();
      printf("\nNo. of characters: %d\nNo. of lines: %d\n", words, lines);
      fclose(yyin);
      return 0;
}
```

2. Write a Lex program that implements the Caesar cipher: it replaces every letter with the one three letters after in alphabetical order, wrapping around at Z. e.g. a is replaced by d, b by e, and so on z by c.

```
%{
  int rot = 0;
%}
%%
[A-Z] { fprintf(yyout, "%c", (yytext[0] - 'A' + rot) % 26 + 'A'); }
[a-z] { fprintf(yyout, "%c", (yytext[0] - 'a' + rot) % 26 + 'a'); }
. { fprintf(yyout, "%s", yytext); }
%%
int main(void) {
  printf("Enter Key (ROT): ");
  scanf("%d", &rot);
  yyin = fopen("input.txt", "r");
  yyout = fopen("output.txt", "w");
  yylex();
  fclose(yyin);
  fclose(yyout);
  return 0;
}

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

3. Write a Lex program that finds the longest word (defined as a contiguous string of upper and lower-case letters) in the input.

```
%{
  int length = 0;
  char *word = NULL;
%}
%%
[a-zA-Z]+ {
  if (yyleng > length) {
    length = yyleng;
    word = yytext;
  }
}
[ |\n|\r|\t] { ; }
. { ; }
%%
int main(void) {
  yyin = fopen("input.txt", "r");
  yylex();
  fclose(yyin);
  printf("Longest Word: %.*s\n", length, word);
  printf("Length of Longest Word: %d\n", length);
  return 0;
}

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

4. Write a Lex program that distinguishes keywords, integers, floats, identifiers, operators, and comments in any simple programming language.

```
%{
  int integers = 0;
  int floats = 0;
  int identifiers = 0;
  int operators = 0;
  int comments = 0;
%}

%%
[#].* { printf("%s <- preprocessor directive\n", yytext); } // preprocessor directives

[ |\n|\t] { ; } // whitespaces

[,|;|"("|")"|"{"|"}"|"\["|"\]"] { ; } // brackets, delimiters

"//".* { comments++; printf("%s <- comment\n", yytext); } // single line comments

[0-9]+ { integers++; printf("%s <- integer\n", yytext); } // integers

[0-9]+("."[0-9]+) { floats++; printf("%s <- float\n", yytext); } // floats

void|int|main|char|for|while|continue|switch|case|break|if|else|return|true|false { printf("%s <- keyword\n", yytext); } // keywords

"<="|">="|"!="|"=="|"<"|">"|"&"|"|"|"^"|"<<"|">>"|"~"|"&&"|"||"|"!"|"++"|"--"|"="|"+"|"-"|"*"|"/"|"%" { operators++; printf("%s <- operator\n", yytext); } // operators

['](['^\\\']|\\.)?['] { ; } // characters

["]([^\\\"]|\\.)*["] { ; } // strings

[a-zA-Z_]+[a-zA-Z0-9_]* { identifiers++; printf("%s <- identifier\n", yytext); } // identifiers

%%
```

```
int main() {
  yyin = fopen("text.c", "r");
  yylex();
  printf("\n");
  printf("number of integers: %d\n", integers);
  printf("number of floats: %d\n", floats);
  printf("number of identifiers: %d\n", identifiers);
  printf("number of operators: %d\n", operators);
  printf("number of comments: %d", comments);
  return 0;
}

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

5. Write a Lex program to count the number of identifiers in a C file.

```
%{
    int identifiers = 0;
%}

%%
[#].* { ; } // preprocessor directives

[ |\n|\t] { ; } // whitespaces

[,|;|"("|")"|"{"|"}"|"\["|"\]"] { ; } // brackets, delimiters

"//".* { ; } // single line comment

-?[0-9]+("."[0-9]+)? { ; } // numbers

void|int|main|char|for|while|continue|switch|case|break|if|else|return|true
|false { ; } // keywords

"<="|">="|"!="|"=="|"<"|">" { ; }  // relational operators

"&"|"|"|"^"|"<<"|">>"|"~" { ; }  // bitwise operators

"&&"|"||"|"!" { ; } // logical operators
```

```
"++"|"--" { ; } // postfix/prefix operators

"="|"+"|"-"|"*"|"/"|"%" { ; } // other operators

['](["^\\\']|\\.)?['] { ; } // characters


["]([^\\\"]|\\.)*["] { ; } // strings
[a-zA-Z_]+[a-zA-Z0-9_]* { identifiers++; printf("%s <- identifier\n",
yytext); } // identifiers
%%

int main() {
  yyin = fopen("text.c", "r");
  yylex();
  printf("\nnumber of C identifiers: %d\n", identifiers);
  return 0;
}

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

6. Write a Lex program to count the number of words, characters, blank spaces and lines in a C file.

```
%{
  int words = 0;
  int lines = 0;
  int spaces = 0;
  int characters = 0;
%}

%%
[^ \t\n,\.:;]+ { words++; characters += yyleng; }
[\n] { lines++; characters += yyleng; }
[ |\t] { spaces++; characters += yyleng; }
. { characters++; }
%%
```

```
int main() {
  yyin = fopen("text.txt", "r");
  yylex();
  printf("number of words: %d\n", words);
  printf("number of blank spaces: %d\n", spaces);
  printf("number of lines: %d\n", lines);
  printf("number of characters: %d\n", characters);
  return 0;
}

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

7. Write a Lex specification program that generates a C program which takes a string “abcd” and prints the following output.

abcd   abc   ab   a

```
%{
  #include <stdio.h>
%}
%%
a|ab|abc|abcd { printf("%s\n", yytext); REJECT; }
.|\n { ; }
%%
int main() {
  yyin = fopen("input.txt", "r");
  yylex();
  return 0;
}

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

8. A program in Lex to recognize a valid arithmetic expression.

```
%{
  #include <stdio.h>
  int brackets = 0,
      operators = 0,
      numbersOridentifiers = 0,
      flag = 0;
%}
%%
[a-zA-Z_]+[a-zA-Z0-9_]* { numbersOridentifiers++; }
-?[0-9]+("."[0-9]+)? { numbersOridentifiers++; }
[+|\-|*|/|=|\^|%] { operators++; }
"(" { brackets++; }
")" { brackets--; }
";" { flag = 1; }
.|\n { ; }
%%
int main() {
  printf("Enter Arithmetic Expression: ");
  /* yyin = fopen("input.txt", "r"); */
  yylex();
  if (
    (operators + 1) == numbersOridentifiers
      && brackets == 0 && flag == 0
  ) {
    printf("Valid Expression\n");
  } else {
    printf("Invalid Expression\n");
  }
  return 0;
}

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

9. Write a YACC program to find the validity of a given expression (for operators + - * and /)

//lex.l

```
%{
#include "y.tab.h"
%}
letter [a-z]
digit [0-9]
newline [\n]
%%
{letter} { return letter ;}
{digit} { return digit ; }
{newline} { return newline ;}
['+'|'*'|'/'] {return operator;}
['\-'] {return minus;}
['('] {return ob;}
[')'] {return cb;}
. { printf("Invalid Variable\n");}
%%
int yywrap(){
    return 1;
}
```

//yacc.y

```
%{
#include<stdio.h>
#include<stdlib.h>
int yylex(void);
int yyerror(char *);
/*
    E production is to check if entered identifier is valid or not ie
letters then digits (as per question )
    E->letterT
    T->letterT|digit
    Now for valid expression S production taking care of it
    S->E (a single variable is also a valid expression means urinary)
        | S operator S and so on .................
    and newline when user press enter then show result
    Here, number i.e. valid Expression 0/1/2 etc are just for debugging
purpose ie to check which production is used
*/
%}
```

```
//tokens letter digit new line operator open bracket close bracket
%token letter digit newline operator minus ob cb
// for left associativity
%left '+' '-'
%left '*' '/'
%%


S : E { printf("Valid Identifiers 1\n");printf("Final result : valid
Expression \n");exit(0);};
    | S operator S newline {printf("Final result :  valid Expression
\n");exit(0);}
    | S minus S newline {printf("Final result : valid Expression \n");
exit(0);}
    | minus S newline {printf("Final result : valid Expression \n");
exit(0);}
    | S operator ob minus S cb {printf("Final result : valid Expression
\n"); exit(0);}
    | S operator ob S cb {printf("Final result : valid Expression \n");
exit(0);}
    | ob S ob {printf("Final result : valid Expression \n"); exit(0);}
    ;

E : letter T {printf("variable letter\n");};


T: letter T {printf("letter term\n");}| digit {printf("digit\n");};

%%

int yyerror(char *msg)
{
printf("Invalid Expression or identifier\n");
exit(0);
}

int main ()
{
    // main method
printf("Enter the expression: ");
yyparse();
}
```

10. A Program in YACC which recognizes a valid variable which starts with a letter followed by a digit. The letter should be in lowercase only.

// lex.l

```
%{
 #include <stdlib.h>
 #include "y.tab.h"
 void yyerror(char *);
%}
%%
 /* variables */
[a-z] {
 yylval = *yytext - 'a';
 return VARIABLE;
 }
 /* integers */
[0-9]+ {
 yylval = atoi(yytext);
 return INTEGER;
 }
 /* operators */
[-+()=/*\n] { return *yytext; }
 /* skip whitespace */
[ \t] ;

['$'] {exit(0);}
 /* anything else is an error */
. yyerror("invalid character");

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

//yacc .y

```
%token INTEGER VARIABLE
%left '+' '-'
%left '*' '/'
%{
#include <stdio.h>
 void yyerror(char *);
 int yylex(void);
 int sym[26];
%}
%%
program:
 program statement '\n'
 |
 ;
statement:
 expr { printf("Expresssion is Valid and result is : %d\n", $1); }
 ;
expr:
 INTEGER
 | expr '+' expr { $$ = $1 + $3; }
 | expr '-' expr { $$ = $1 - $3; }
 | expr '*' expr { $$ = $1 * $3; }
 | expr '/' expr { $$ = $1 / $3; }
 | '(' expr ')' { $$ = $2; }
 ;
%%
void yyerror(char *s) {
 printf( "Invalid Expresssion : %s\n", s);
}
int main(void) {
printf("Enter the Expresssion: \n");
printf("Press Enter to see result.\nPress $ to end.\n");
 yyparse();
 return 0;
}
```

11. A Program in YACC to evaluate an expression (simple calculator program for addition and subtraction, multiplication, division).

// lex.l

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

  #if __has_include("y.tab.h")
    #include "y.tab.h"
  #endif
%}

%option noyywrap
%%
[0-9]+(\.[0-9]+)? { yylval.f = atof(yytext); return NUM; }
[\-+()*/] { return yytext[0]; }
[ \t\n]+ { ; }
%%
```

// yacc.y

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

%union { float f; }
%token <f> NUM
%type <f> E T F
%%
S : E { printf("%f\n", $1); }
  ;
E : E '+' T { $$ = $1 + $3; }
  | E '-' T { $$ = $1 - $3; }
  | T
  ;
T : T '*' F { $$ = $1 * $3; }
  | T '/' F { $$ = $1 / $3; }
  | F
  ;
```

```
F : '(' E ')' { $$ = $2; }
  | '-' F { $$  = -$2; }
  | NUM
  ;
%%
int main()
{
   yyparse();
   return 0;
}

void yyerror(char *msg) {
  fprintf(stderr, "%s\n", msg);
  exit(1);
}
```

12. Program in YACC to recognize the strings “ab”, “aabb”, ”aaabbb”,… of the language (*a*^*n* *b*^*n* , n>=1).

// lex.l

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

  #if __has_include("y.tab.h")
    #include "y.tab.h"
  #endif
%}

%option noyywrap
%%
[a] { return A; }
[b] { return B; }
[ |\n|\t] { return yytext[0]; }
. { return yytext[0]; }
%%
```

// yacc.y

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

%token A B

%%
S : E '\n' { printf("VALID STRING\n"); exit(0); }
  ;
E : A E B
  | A B
  ;
%%
int main()
{
   yyparse();
   return 0;
}

void yyerror(char *msg) {
  fprintf(stderr, "INVALID STRING\n");
  exit(1);
}
```

13. Program in YACC to recognize the language (a^n b , n>=10). (Output to say input is valid or not)

// lex.l

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

  #if __has_include("y.tab.h")
    #include "y.tab.h"
  #endif
%}

%option noyywrap
%%
[a] { return A; }
[b] { return B; }
[ |\n|\t] { ; }
```

```
. { return yytext[0]; }
%%
```

// yacc.y

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

%token A B
%%
S : X Y B { printf("VALID STRING\n"); }
  ;
X : A A A A A A A A A A
  ;
Y : A Y
  |
  ;
%%
int main()
{
   yyparse();
   return 0;
}

void yyerror(char *msg) {
  fprintf(stderr, "INVALID STRING\n");
  exit(1);
}
```