```
lex filename.l
gcc lex.yy.c
./a.out

lex filename.l
yacc filename.y -d
gcc lex.yy.c y.tab.c
./a.out
```

Assignment 2

1. Lex program to count comments, keywords, identifiers, words, lines, and spaces
2. Lex program to count total characters, white spaces, and tabs
3. Lex program to count total tokens
4. Lex program to count frequency of a given word

Assignment 3

1. Lex program to count words starting with ‘A’
2. Lex program to count words beginning and ending with ‘a’
3. Lex program to count words starting and ending with vowels

Assignment 4

1. Lex program for lowercase to uppercase and vice versa conversion
2. Lex program to check whether the character is uppercase, lowercase, or non-alphabetic
3. Lex program to count uppercase and lowercase characters from a file
4. Lex program for case conversion using file handling

Assignment 5

1. Lex program for decimal → hexadecimal conversion
2. Lex program for decimal → binary conversion
3. Lex program for hexadecimal → decimal conversion
4. Lex program to count lines ending with .com and URLs ending with .org
5. Lex program to count URLs ending with .edu

Assignment 6

1. YACC program for postfix expression evaluation
2. YACC program for infix → postfix conversion
3. YACC program for evaluating postfix expressions (with floating-point numbers)

Assignment 8

1. YACC desk calculator with error recovery
2. YACC desk calculator supporting parentheses

Assignment 9

1. YACC program for syntax checking of **for** loop
2. YACC program for syntax checking of **while** loop
3. YACC program for syntax checking of **switch–case**
4. YACC program for syntax checking of **if / if-else** statements

Assignment 10

1. YACC program for **Intermediate Code (IC)** generation for arithmetic expressions
2. YACC program for **IC generation** with parentheses
3. YACC program for **IC generation** for programming constructs

Assignment 11

1. Lex program for a simple calculator
2. Lex program for identifying digits and operators
3. Lex calculator for three operands and two operators (with precedence handling)

Assignment 12

1. Lex program to check valid email address
2. Lex program to check all domain-type email addresses (e.g., .com, .org, .co.in)
3. Lex program for email validation with file handling

Assignment 2

1)Lex program counts the number of comments, keywords, identifiers, words, lines, and spaces from the input file.

```
%{
int comment=0;
int keyword=0;int identifier=0;
int word=0;
int lines=0;
int spaces=0;
%}
%%
[/][/].* {comment++;}
[/][*][\n].*[*][/] {comment++;}
[' '] { spaces++; }
["].*["] {word++;}
"printf"|"scanf"|"while"|"for"|"if"|"else" { keyword++; word++; }
[A-Za-z_][A-Za-z]* { identifier++; word++;}
[a-z_][A-Za-z0-9_]* { identifier++; }
\n { lines++; }
. { }
%%
int main(int argc, char* argv[])
```

```
{
if (argc==2)
{
yyin=fopen(argv[1],"r");
}
else
{
printf("Enter the Input\n");
yyin=stdin;
}
yylex();
printf("comments = %d \nspaces= %d \nlines= %d \nwords= %d
\nkeywords= %d\nidentifier=%d\n",comment, spaces, lines, word,
keyword, identifier);
return 0;
}
int yywrap() {
return 1;
}
```

2)Lex Program to print the total characters, white spaces, tabs in the given input file.

```
%{
int chars=0;
int space=0;int tabs=0;
%}
%%
[\t] {tabs++;chars++;}
[ ] {space++;chars++;}
. {chars++;}
%%
int main(int argc, char* argv[])
{
if (argc==2)
{
yyin=fopen(argv[1],
"r");
}
else
{
printf("Enter the Input\n");
yyin=stdin;
}
yylex();
printf("Total characters = %d\n"
, chars);
printf("Spaces = %d\n"
, space);
printf("Tabs = %d\n"
, tabs);
```

```
return 0;
}
int yywrap() {
return 1;
}
```

3) Lex code to count the total number of tokens.

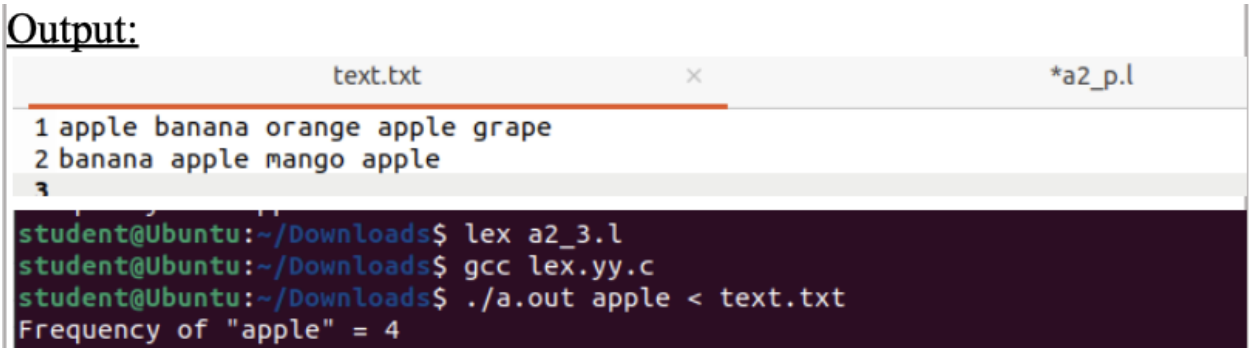
```
%{
int t=0;
%}
%%
["].
*["] {t++;}
[a-zA-Z
_][a-zA-Z0-9
_]* { t++; }
[0-9]+ { t++; }
[ \t\n]+ { }
. { t++; }
%%int main(int argc, char* argv[])
{
if (argc==2)
{
yyin=fopen(argv[1],
"r");
}
else
{
printf("Enter the Input\n");
yyin=stdin;
}
yylex();
printf("comments = %d "
,t);
return 0;
}
int yywrap() {
return 1;
}
Input file for all 3 is run.c
#include <stdio.h>
int main() {
        int num, reversed = 0, remainder, original;
printf("Enter an integer: ");
scanf ("%d", &num);
original = num;
while (num != 0) {
        remainder = num % 10;
        reversed = reversed * 10 + remainder;
```

```
        num /= 10;
}
if (original == reversed) {
        printf("%d is a palindrome.(n", original);
} else {
        printf("%d is not a palindrome.\n", original);
}
return 0;
}
```

4. Lex program to count the frequency of the given word in a file.

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

char *word_to_count;   // word to search for
int count = 0;         // frequency counter
%}

%%
[a-zA-Z0-9_]+ {
   if (strcmp(yytext, word_to_count) == 0)
      count++;
}
.|\n    { /* ignore other characters */ }
%%

int main(int argc, char *argv[])
{
   if (argc != 2) {
      fprintf(stderr, "Usage: %s <word_to_count>\n", argv[0]);
      return 1;
   }

   word_to_count = argv[1];  // store the word to count
   yylex();                  // start lexical analysis
   printf("Frequency of \"%s\" = %d\n", word_to_count, count);
   return 0;
}

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

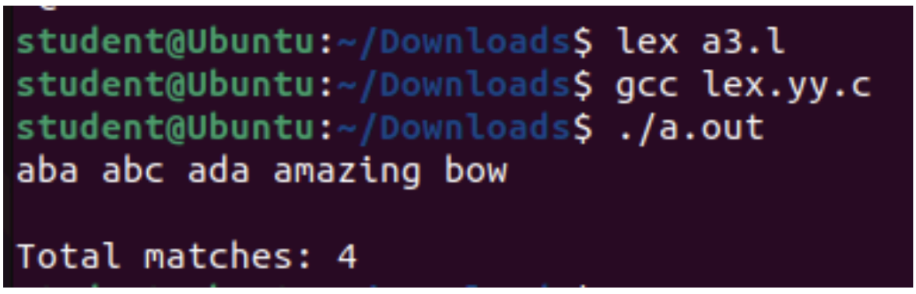
Output:



Assignment 3

1. Implement a LEX program to count the number of words starting with 'A'.

```
%{
#include<stdio.h>
int count =0;
%}
%%
^[Aa] {count++;}
" "[Aa] {count++;}
. {}
%%int main() {
yylex();
printf("Total matches: %d\n"
, count);
return 0;
}
int yywrap() {
return 1;
}
```



2. Implement a LEX program for words beginning and ending with 'a'.

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

```
%%
[b-zB-Z][a-zA-Z]*[ \n] { }
[a-zA-Z]*[b-zB-Z][ \n] { }
^[Aa][a-zA-Z]*[Aa][ \n] {count++;}
[Aa][a-zA-Z]*[Aa][ \n] {count++;}
. {}
%%
int main() {
yylex();
printf("Total matches: %d\n"
, count);
return 0;
int yywrap() {
return 1;
}
}
```

```
student@Ubuntu:~/Downloads$ lex a3.l
student@Ubuntu:~/Downloads$ gcc lex.yy.c
student@Ubuntu:~/Downloads$ ./a.out
aba abba baa abb
abb
aab aaa aa
Total matches: 4
```

3. Implement a LEX program to find the frequency of words which start and end with vowels in the input file.

```
%{
#include<stdio.h>
int count =0;
%}
%%
[b-df-hj-np-tv-zB-DF-HJ-NP-TV-Z][a-zA-Z]*[ \n] { }
[a-zA-Z]*[b-df-hj-np-tv-zB-DF-HJ-NP-TV-Z][ \n] { }
^[AEIOUaeiou][a-zA-Z]*[AEIOUaeiou][ \n] {count++;}
[AEIOUaeiou][a-zA-Z]*[aeiouAEIOU][ \n] {count++;}
Post lab questions
. {}
%%
int main() {
yylex();
printf("Total matches: %d\n"
, count);
return 0;
}
int yywrap() {
```

return 1;
}

```
student@Ubuntu:~/Downloads$ lex a3.l
student@Ubuntu:~/Downloads$ gcc lex.yy.c
student@Ubuntu:~/Downloads$ ./a.out
bade aeiou amu ab

aba uba
Total matches: 4
```

Assignment 4

1. LEX code for conversion of lowercase to uppercase and vice versa.

```
%{
%}
%%
[a-z] { printf("%c"
, yytext[0]- 32); }
[A-Z] { printf("%c"
, yytext[0]+32); }
%%
int main() {
yylex();return 0;
}
int yywrap() {
return 1;
}
```

```
student@Ubuntu:~/Downloads$ lex a3_1.l
student@Ubuntu:~/Downloads$ gcc lex.yy.c
student@Ubuntu:~/Downloads$ ./a.out
Abc ABc aBc
aBC abC AbC
```

2. LEX code to check whether the given character is in upper case, or in lower case or non-alphabetic character.

```
%{
%}
%%
[a-z] { printf("Lowercase %s\n"
, yytext); }
[A-Z] { printf("Uppercase %s\n"
, yytext); }
[\n] {printf("Non-Alphabetic Character new line");}
```

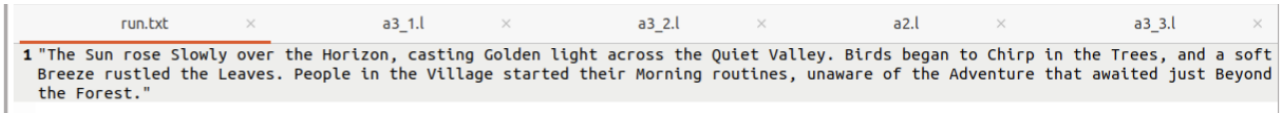
```
[\t] {printf("Non-Alphabetic Character tab space");}
[ ] {printf("Non-Alphabetic Character space");}
. {printf("Non-Alphabetic Character %s\n"
,yytext);}
%%
int main() {
yylex();
return 0;
int yywrap() {
return 1;
}
}
```

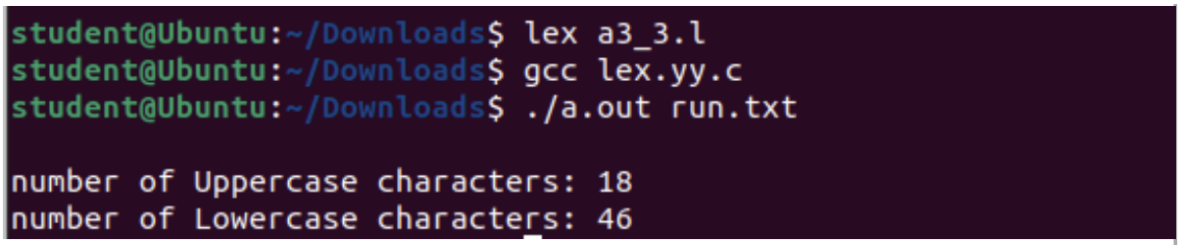
```
student@Ubuntu:~/Downloads$ lex a3_2.l
student@Ubuntu:~/Downloads$ gcc lex.yy.c
student@Ubuntu:~/Downloads$ ./a.out
WORD Abc        ccl
Uppercase W
Uppercase O
Uppercase R
Uppercase D
Non-Alphabetic Character space
Uppercase A
Lowercase b
Lowercase c
Non-Alphabetic Character tab space
Lowercase c
Lowercase c
Lowercase l
Non-Alphabetic Character new line
&abc
Non-Alphabetic Character &
Lowercase a
Lowercase b
Lowercase c
Non-Alphabetic Character new line
S
```

3. LEX code to count the lowercase, upper case characters in the given input file.

```
%{
int upper=0;
int lower=0;
%}
%%
[A-Z] { upper++; }
[a-z]+ { lower++; }
. { }
%%
int main(int argc, char* argv[])
{
if (argc==2)
{
yyin=fopen(argv[1],
```

```
"r");
}
else
{
printf("Enter the Input\n");
yyin=stdin;
}
yylex();
printf("number of Uppercase characters: %d\nnumber of Lowercase
characters: %d\n"
,upper,lower);
return 0;
}
int yywrap() {
return 1;
}
```





4. LEX code for case conversion of alphabets using file handling.

```
%{
#include <ctype.h>
#include <stdio.h>
%}
%%
[a-z] { fprintf(yyout,
[A-Z] {fprintf (yyout,
. { fprintf(yyout,
%%
"%c"
, toupper(yytext[0])); }
"%c"
, tolower(yytext[0])); }
"%c"
, yytext[0]); }
int main()
{
```

```
extern FILE *yyin,
*yyout;
yyin = fopen("run.txt"
,
"r");
yyout = fopen("Output.txt"
,
"w");
yylex();
fclose(yyin);
fclose(yyout);
return 0;
}
int yywrap() {
return 1;
}
```

Assignment 5

1.Write a LEX program for conversion of decimal to hexadecimal number in a file.

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

void convert_to_hex(const char *text) {
    double num = atof(text);
    int int_part = (int)num;
    double frac_part = num - int_part;

    printf("Decimal: %s\tHex: %X", text, int_part);
```

```
    if (frac_part > 0.0) {
        printf(".");
        for (int i = 0; i < 4 && frac_part > 0.0; i++) {
            frac_part *= 16;
            int digit = (int)frac_part;
            printf("%X", digit);
            frac_part -= digit;
        }
    }
    printf("\n");
}
%}

%%
[0-9]+\.[0-9]+   { convert_to_hex(yytext); }
[0-9]+           { convert_to_hex(yytext); }
[ \t\n]+         { /* Ignore whitespace */ }
.                { printf("Invalid input: %s\n", yytext); }
%%

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

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

```
student@Ubuntu:~/Downloads$ lex a5.l
student@Ubuntu:~/Downloads$ gcc lex.yy.c
student@Ubuntu:~/Downloads$ ./a.out
5.2
Decimal: 5.2    Hex: 5.3333
5
Decimal: 5      Hex: 5
11
Decimal: 11     Hex: B
11.4
Decimal: 11.4   Hex: B.6666
11.16
Decimal: 11.16  Hex: B.28F5
```

2. Write a LEX program for decimal to binary conversion.

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

```
#include <stdlib.h>

void print_binary_int(int n) {
    if (n == 0) {
        printf("0");
        return;
    }
    int bits[32], i = 0;
    while (n > 0) {
        bits[i++] = n % 2;
        n /= 2;
    }
    for (int j = i - 1; j >= 0; j--)
        printf("%d", bits[j]);
}

void print_binary_frac(double frac) {
    if (frac <= 0) return;
    printf(".");
    for (int i = 0; i < 8 && frac > 0; i++) {
        frac *= 2;
        if (frac >= 1) {
            printf("1");
            frac -= 1;
        } else {
            printf("0");
        }
    }
}

void convert(const char *text) {
    double num = atof(text);
    int int_part = (int)num;
    double frac_part = num - int_part;
    printf("Decimal: %s\tBinary: ", text);
    print_binary_int(int_part);
    print_binary_frac(frac_part);
    printf("\n");
}
%}

%%
[0-9]+(\.[0-9]+)?    { convert(yytext); }
[ \t\n]+             ;
.                    { printf("Invalid input: %s\n", yytext); }
%%

int main() {
    yylex();
```

```
    return 0;
}

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

```
student@Ubuntu:~/Downloads$ lex a5_2.l
student@Ubuntu:~/Downloads$ gcc lex.yy.c
student@Ubuntu:~/Downloads$ ./a.out
2.5
Decimal: 2.5    Binary: 10.1
2
Decimal: 2      Binary: 10
10
Decimal: 10     Binary: 1010
32
Decimal: 32     Binary: 100000
```

3. Write a LEX program for Hexadecimal to Decimal conversion.

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

void hex_to_decimal(const char *text) {
    long decimal = strtol(text, NULL, 16);
    printf("Hex: %s\tDecimal: %ld\n", text, decimal);
}
%}

%%
[0-9a-fA-F]+    { hex_to_decimal(yytext); }
[ \t\n]+        ;
.               ;
%%

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

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

```
student@Ubuntu:~/Downloads$ lex a5_3.l
student@Ubuntu:~/Downloads$ gcc lex.yy.c
student@Ubuntu:~/Downloads$ ./a.out
52
Hex: 52 Decimal: 82
B5
Hex: B5 Decimal: 181
```

Assignment 5

1. Lex program to count lines ending with "com" Lex program to count URLs ending with "org".

```
%{
int com=0;
int org=0;
%}
%%
.com$ {com++;}
http[s]?[:][/][/][^ ][^\n]*[.]org {org++;}
. {}
%%
int main(int argc, char* argv[])
{
if (argc==2)
{yyin=fopen(argv[1],
"r");
}
else
{
printf("Enter the Input\n");
yyin=stdin;
}
yylex();
printf("lines ending with com = %d \nurls ending with org = %d\n"
,com,
org);
return 0;
}
int yywrap() {
return 1;
}
```

text.txt × a6.l

```
1 http://example.com
2 http://example.com
3 email@domain.com
4 https://nonprofit.org
5 Just some text here.
6 http://anotherexample.org
7 example.net
8 visit http://opensource.org for more info
9 localhost.localdomain
10 email@domain.com
11 email@domain.com
12 http://example.com
13 email@domain.com
```

```
student@Ubuntu:~/Downloads$ lex a6.l
student@Ubuntu:~/Downloads$ gcc lex.yy.c
student@Ubuntu:~/Downloads$ ./a.out text.txt
```

```
lines ending with com = 7
urls ending with org = 3
```

2. Lex program to count URLs ending with "edu".

```
%{
int edu=0;
%}
%%
http[s]?[:][/][/][^ ][^\n]*[.]edu {edu++;}
. {}
%%
int main(int argc, char* argv[])
{
if (argc==2)
{
yyin=fopen(argv[1],
"r");
}
else
{
printf("Enter the Input\n");
yyin=stdin;
}
yylex();
printf("urls ending with org = %d\n"
,edu);
return 0;
}
int yywrap() {
```

return 1;
}

```
1 http://www.harvard.edu
2 https://mit.edu for information about MIT
3 http://example.com
4 https://openai.org
5 http://university.edu/path
6 http://notedu.com
7 https://subdomain.school.edu
8 http://something.edu
9
```

Terminal Output:

Assignment 6

1.YACC program for Postfix Expression Evaluation.

LEX code:

```
%{
#include<stdio.h>
#include "y.tab.h"
%}
digit [0-9]
number {digit}+
operator [+\-
*/]
%%
{number}  {yylval.n=atoi(yytext); return oprnd;}
{operator}  {return yytext[0];}
.
%%
int yywrap(){return 1;
}
```

Yacc code:

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

#define MAX 100
int stack[MAX];
int top = -1;

extern int yylex();
int yyerror(const char*);
int pop();
void push(int);
%}

%union {
   int n;
}

%token <n> oprnd

%%
S:
   | E { printf("\nResult = %d\n", pop()); }
;

E:
     E E '+' { int a = pop(); int b = pop(); push(b + a); }
   | E E '-' { int a = pop(); int b = pop(); push(b - a); }
   | E E '*' { int a = pop(); int b = pop(); push(b * a); }
   | E E '/' { int a = pop(); int b = pop(); push(b / a); }
   | oprnd   { push($1); }
;
%%

void push(int val) {
   stack[++top] = val;
}

int pop() {
   return stack[top--];
}

int main() {
   printf("\nEnter the postfix expression: ");
   yyparse();
   return 0;
}
```

```
int yyerror(const char *s) {
   fprintf(stderr, "\nError: %s\n", s);
   return 0;
}
```

```
student@Ubuntu:~/Downloads$ lex a7.l
student@Ubuntu:~/Downloads$ yacc a7.y -d
a7.y:14 parser name defined to default :"parse"
student@Ubuntu:~/Downloads$ gcc lex.yy.c y.tab.c
student@Ubuntu:~/Downloads$ ./a.out

Enter the postfix expression: 3 4 +


Result=7
student@Ubuntu:~/Downloads$ ./a.out

Enter the postfix expression: 12 4 + 3 2 * - 2 /


Result=5
```

2. YACC program for Conversion of Infix to Postfix expression.

LEX Code

```
%{
#include "y.tab.h"
%}
%%
[0-9] { yylval = yytext[0]; return DIGIT; }
[+\-
*/()] { return yytext[0]; }
[ \t\n] ; /* skip whitespace */
. { printf("Invalid character: %s\n"
, yytext); }
%%
int yywrap() {
return 1;
}
```

YACC Code

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

int yylex(void);
void yyerror(const char *s) { printf("Error: %s\n", s); }
%}
```

```
%token DIGIT
%left '+' '-'
%left '*' '/'

%%
expr:
      expr '+' expr   { printf("+ "); }
    | expr '-' expr   { printf("- "); }
    | expr '*' expr   { printf("* "); }
    | expr '/' expr   { printf("/ "); }
    | '(' expr ')'    { /* grouping - no output */ }
    | DIGIT           { printf("%c ", $1); }
;
%%

int main() {
   printf("Enter infix expression:\n");
   yyparse();
   printf("\n");
   return 0;
}
```

```
student@Ubuntu:~/Downloads$ lex a7.l
student@Ubuntu:~/Downloads$ yacc a71.y -d
a71.y:23 parser name defined to default :"parse"
student@Ubuntu:~/Downloads$ gcc lex.yy.c y.tab.c
student@Ubuntu:~/Downloads$ ./a.out
Enter infix expression:
(1+2)*(3-4)/5
1 2 + 3 4 - * 5 /
```

3. YACC program for evaluating postfix expressions containing floating point numbers.

LEX Code

```
%{
#include <stdio.h>
#include <stdlib.h> /* for atof */
#include "y.tab.h"
%}
digit [0-9]
number {digit}+(\.{digit}+)? /* integers or floats */
operator [+\-
*/]%%
{number} { yylval.f = atof(yytext); return oprnd; }
{operator} { return yytext[0]; }
. { ; } /* catch all other characters and ignore */
[ \t\n] { ; } /* ignore whitespace */
%%
```

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

Yacc code

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

extern int yylex();
int yyerror(const char *s);
float pop();
void push(float);

float stack[100];
int top = -1;
%}

%union {
   float f;
}

%token <f> oprnd

%%
S:
     /* empty */
   | E { printf("\nResult = %f\n", pop()); }
;

E:
     E E '+' { float a = pop(); float b = pop(); push(b + a); }
   | E E '-' { float a = pop(); float b = pop(); push(b - a); }
   | E E '*' { float a = pop(); float b = pop(); push(b * a); }
   | E E '/' { float a = pop(); float b = pop(); push(b / a); }
   | oprnd   { push($1); }
;
%%

void push(float val) {
   stack[++top] = val;
}

float pop() {
   return stack[top--];
}

int main() {
   printf("\nEnter the postfix expression: ");
```

```
    yyparse();
    return 0;
}

int yyerror(const char *s) {
    fprintf(stderr, "\nError: %s\n", s);
    return 0;
}
```

```
satviki@sandip:~/lexyacc_demo$ lex file.l
satviki@sandip:~/lexyacc_demo$ yacc -d file.y
satviki@sandip:~/lexyacc_demo$ gcc lex.yy.c y.tab.c
satviki@sandip:~/lexyacc_demo$ ./a.ouy
-bash: ./a.ouy: No such file or directory
satviki@sandip:~/lexyacc_demo$ ./a.out

Enter the postfix expression: 12.5 3.5 +

Result = 16.000000
```

Assignment 8

1.YACC program for desk calculator with error recovery.

Lex file

```
%{
#include "y.tab.h"
%}
%%
[0-9]+ { yylval = atoi(yytext); return oprnd; }
[+\-
*/] { return yytext[0]; }
[ \t\n]+ ; // Skip whitespace
. { printf("Invalid character: %s\n"
, yytext); }
%%
int yywrap() { return 1; }
```

Yacc file

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

extern int yylex();
int yyerror(const char *s);
int pop();
void push(int);
```

```
int stack[100];
int top = -1;
%}

%union {
   int n;
}

%token <n> oprnd

%%
S:
     /* empty */
   | E { printf("\nResult = %d\n", pop()); }
;

E:
     E E '+' { int a = pop(), b = pop(); push(b + a); }
   | E E '-' { int a = pop(), b = pop(); push(b - a); }
   | E E '*' { int a = pop(), b = pop(); push(b * a); }
   | E E '/' {
         int a = pop(), b = pop();
         if (a == 0) {
            fprintf(stderr, "Error: Division by zero\n");
            push(0);
         } else {
            push(b / a);
         }
     }
   | oprnd { push($1); }
;
%%

void push(int val) {
   stack[++top] = val;
}

int pop() {
   return stack[top--];
}

int main() {
   printf("Enter the postfix expression: ");
   yyparse();
   return 0;
}

int yyerror(const char *s) {
```

```
    fprintf(stderr, "Error: %s\n", s);
    return 0;
}
```

```
student@Ubuntu:~$ cd Downloads
student@Ubuntu:~/Downloads$ lex a8.l
student@Ubuntu:~/Downloads$ yacc a8.y
a8.y:15 parser name defined to default :"parse"
student@Ubuntu:~/Downloads$ yacc a8.y -d
a8.y:15 parser name defined to default :"parse"
student@Ubuntu:~/Downloads$ gcc lex.yy.c y.tab.c
student@Ubuntu:~/Downloads$ ./a.out
Enter the postfix expression: 1 2 + 3 * 0 /
Error: Division by zero
```

2. YACC program for desk calculators to evaluate arithmetic expressions involving parenthesis.

Lex file

```
%{
#include "y.tab.h"
%}
%%
[0-9]+ [ \t] ;
[\n]        { return '\n'; }
.           { return yytext[0]; }
%%
{ yylval = atoi(yytext); return NUMBER; }
int yywrap() { return 1; }
```

Yacc file

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

void yyerror(const char *s);
int yylex(void);
%}

%token NUMBER
%left '+' '-'
%left '*' '/'

%%
calculation:
     /* empty */
   | calculation expression '\n' { printf("= %d\n", $2); }
;
```

```
expression:
     expression '+' expression { $$ = $1 + $3; }
   | expression '-' expression { $$ = $1 - $3; }
   | expression '*' expression { $$ = $1 * $3; }
   | expression '/' expression {
        if ($3 == 0) {
           yyerror("division by zero");
           $$ = 0;
        } else {
           $$ = $1 / $3;
        }
     }
   | '(' expression ')' { $$ = $2; }
   | NUMBER { $$ = $1; }
;
%%

void yyerror(const char *s) {
   fprintf(stderr, "Error: %s\n", s);
}

int main() {
   printf("Enter expressions (Ctrl+D to exit):\n");
   yyparse();
   return 0;
}
```

```
student@Ubuntu:~/Downloads$ lex a8.l
student@Ubuntu:~/Downloads$ yacc a8.y -d
a8.y:35 parser name defined to default :"parse"
student@Ubuntu:~/Downloads$ gcc lex.yy.c y.tab.c
student@Ubuntu:~/Downloads$ ./a.out
Enter expressions:
2 + ( 3 * 4 )
= 14
```

Assignment 9

1. YACC program for parser for “FOR” loop statements.

Lex file

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

```
"for" {return FOR;}
"(" {return LP;}
")" {return RP;}
";" {return SC;}
[a-zA-Z_][a-zA-Z0-9_]* {return ID;}
"+"|"-"|"*"|"/"|"="|"=="|"<"|"<="|">"|">="|"&&"|"||" {return BOP;}"++"|"--" {return UOP;}
[0-9]+ {return NUM;}
. {}
%%
int yywrap()
{
return 1;
}
```

Yacc file

```
%{
#include<stdio.h>
extern int yylex();
int yyerror(const char *);
%}
%token FOR LP RP SC ID NUM UOP BOP
%%
P:S
;
S: FOR LP E SC E SC E RP
{
printf("valid for loop detected\n");
E: ID | NUM | ID BOP ID | ID BOP NUM | ID UOP | UOP ID
}
;
;
%%
int main()
{
yyparse();
return 0;
}
int yyerror(const char *s){
fprintf(stderr, "Errror:%s\n",s);
return 0;
}
```

2. YACC program for checking syntax for a While loop.

Lex File:

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

```
%%
"while" { return WHILE; }
"(" { return LP; }
")" { return RP; }
[a-zA-Z_][a-zA-Z0-9_]* { return ID; }
[0-9]+ { return NUM; }
"=="|"!="|"<"|"<="|">"|">="|"&&"|"||" { return BOP; }
[ \t\n]+ { /* skip whitespace */ }
. { /* ignore other characters */ }
%%
int yywrap() {
return 1;
}
```

Yacc file

```
%{
#include <stdio.h>
#include <stdlib.h>
int yylex();
int yyerror(const char *s);
%}
%token WHILE LP RP ID NUM BOP
%%
P : S
;
S : WHILE LP E RP
{
}
printf("Valid while loop detected\n");
;
E : ID
| NUM
| ID BOP ID
| ID BOP NUM
;%%
int main() {
return yyparse();
}
int yyerror(const char *s) {
fprintf(stderr, "Error: %s\n", s);
return 0;
}
```

3. YACC program for checking syntax for a Switch case.

Lex File

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

```
%}
%%
"switch" { return SWITCH; }
"case" { return CASE; }
"default" { return DEFAULT; }
"break" { return BREAK; }
":" { return COLON; }
";" { return SC; }
"{" { return LB; }
"}" { return RB; }
"(" { return LP; }
")" { return RP; }
[a-zA-Z_][a-zA-Z0-9_]* { return ID; }
[0-9]+ { return NUM; }
[ \t\n]+ { } // skip whitespace
. { } // ignore unknown characters
%%
int yywrap() { return 1; }
```

Yacc file

```
%{
#include <stdio.h>
int yylex();
int yyerror(const char *s);%}
%token SWITCH CASE DEFAULT BREAK COLON SC LB RB LP
RP ID NUM
%%
P : S ;
S : SWITCH LP ID RP LB CASES DEFAULT_BLOCK RB {
printf("Valid switch-case statement\n");
};
CASES : CASE NUM COLON ID SC BREAK SC CASES
| /* empty */ ;
DEFAULT_BLOCK : DEFAULT COLON ID SC BREAK SC
| /* empty */ ;
%%
int main() {
return yyparse();
}
int yyerror(const char *s) {
fprintf(stderr, "Error: %s\n", s);
return 0;
}
```

```
student@Ubuntu:~/Downloads$ lex a9.l
student@Ubuntu:~/Downloads$ yacc a9.y -d
a9.y:25 parser name defined to default :"parse"
student@Ubuntu:~/Downloads$ gcc lex.yy.c y.tab.c
student@Ubuntu:~/Downloads$ ./a.out
while(a!=0)
Valid while loop detected
while(a=0;b=5)
Error: parse error
```

```
student@Ubuntu:~/Downloads$ lex a9.l
student@Ubuntu:~/Downloads$ yacc a9.y -d
a9.y:21 parser name defined to default :"parse"
student@Ubuntu:~/Downloads$ gcc lex.yy.c y.tab.c
student@Ubuntu:~/Downloads$ ./a.out
switch(x){
case 1: y;
break;
case 2: z;
break;
default: a;
break;
}
Valid switch-case statement
^C
student@Ubuntu:~/Downloads$ ./a.out
switch(x){
case 1 y;
break;
}
Error: parse error
```

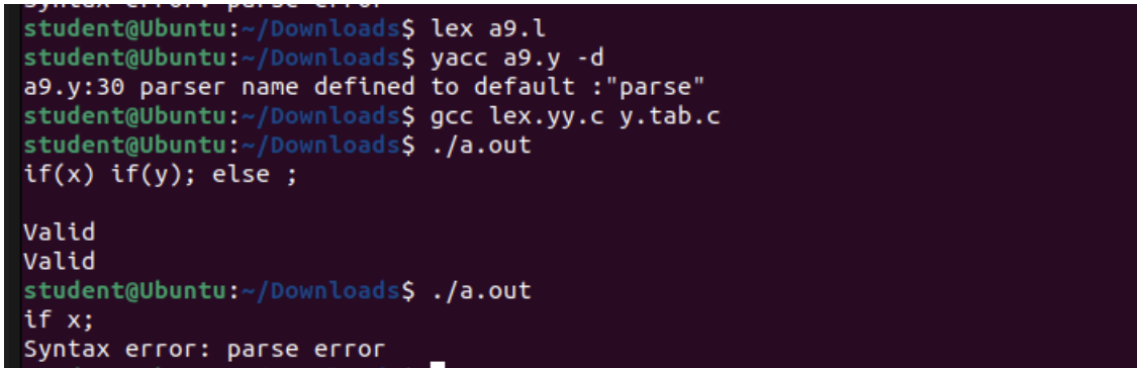
4. YACC program for checking syntax for an If statement (with and without else).

Lex File

```
%{
#include "y.tab.h"
%}
%%
"if" { return IF; }
"else" { return ELSE; }
"(" { return LP; }
")" { return RP; }
";" { return SC; }
[a-zA-Z
_][a-zA-Z0-9
_]* { return ID; }
[ \t\n]+ { } // ignore whitespace
. { } // ignore unknown characters
%%
int yywrap() { return 1; }
```

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

int yylex();
```

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

%token IF ELSE LP RP SC ID

%%
S:
     MIF { printf("Valid\n"); }
   | UMIF { printf("Valid\n"); }
;

MIF:
     IF LP E RP MIF ELSE MIF
   | other_statement
;

UMIF:
     IF LP E RP S
   | IF LP E RP MIF ELSE UMIF
;

other_statement:
     SC
   | ID SC
;

E:
     ID
;
%%

int main() {
   printf("Enter IF-ELSE statement:\n");
   return yyparse();
}

int yyerror(const char *s) {
   fprintf(stderr, "Syntax error: %s\n", s);
   return 0;
}
```



Assignment 10

1.YACC program for Intermediate code (IC) generator for arithmetic expression.

Lex file

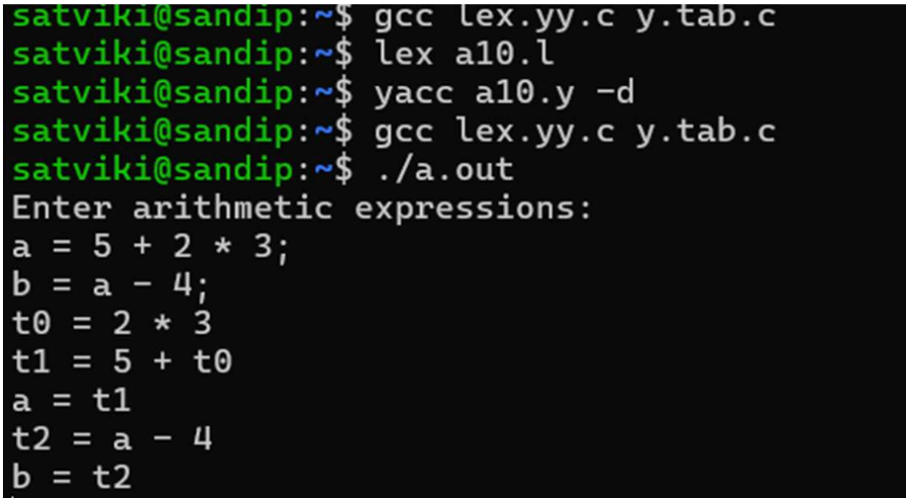
```
%{
#include "y.tab.h"
#include <stdlib.h>
#include <string.h>
%}
digit [0-9]+
id [a-zA-Z_][a-zA-Z0-9_]*
%%
{digit}+ { yylval.ival = atoi(yytext); return NUM; }
{id} { yylval.sval = strdup(yytext); return ID; }
[\t\n ]+ ; // ignore whitespace
"=" { return '='; }"+" { return '+'; }
"-" { return '-'; }
"*" { return '*'; }
"/" { return '/'; }
";" { return ';'; }
"(" { return '('; }
")" { return ')'; }
. { return yytext[0]; }
%%
int yywrap() { return 1; }
```

Yacc file

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

int temp_count = 0;

// Generate a new temporary variable
char* newTemp() {
```

```
    char* temp = (char*)malloc(10);
    sprintf(temp, "t%d", temp_count++);
    return temp;
}

// Generate intermediate code
void gen(char* op, char* arg1, char* arg2, char* res) {
    printf("%s = %s %s %s\n", res, arg1, op, arg2);
}

void yyerror(const char *s);
int yylex(void);
%}

%union {
    int ival;
    char* sval;
}

%token <ival> NUM
%token <sval> ID
%left '+' '-'
%left '*' '/'
%type <sval> expr term factor stmt

%%
prog:
      stmt_list
;

stmt_list:
      stmt_list stmt
    | stmt
;

stmt:
      ID '=' expr ';' {
            printf("%s = %s\n", $1, $3);
            free($1);
            free($3);
      }
    | expr ';' { free($1); }
;

expr:
      expr '+' term {
            char* t = newTemp();
            gen("+", $1, $3, t);
            free($1); free($3);
```

```
            $$ = t;
        }
    | expr '-' term {
            char* t = newTemp();
            gen("-", $1, $3, t);
            free($1); free($3);
            $$ = t;
        }
    | term { $$ = $1; }
;

term:
      term '*' factor {
            char* t = newTemp();
            gen("*", $1, $3, t);
            free($1); free($3);
            $$ = t;
        }
    | term '/' factor {
            char* t = newTemp();
            gen("/", $1, $3, t);
            free($1); free($3);
            $$ = t;
        }
    | factor { $$ = $1; }
;

factor:
      '(' expr ')' { $$ = $2; }
    | NUM {
            char* t = (char*)malloc(20);
            sprintf(t, "%d", $1);
            $$ = t;
        }
    | ID { $$ = strdup($1); }
;
%%

int main() {
    printf("Enter arithmetic expressions:\n");
    yyparse();
    return 0;
}

void yyerror(const char *s) {
    printf("Error: %s\n", s);
}
```



2. YACC program for IC generation for the expression involving parenthesis.

Lex file

```
%{
#include "y.tab.h"
#include <stdlib.h>
#include <string.h>
%}
digit [0-9]+
id [a-zA-Z_][a-zA-Z0-9_]*
%%
{digit}+ { yylval.ival = atoi(yytext); return NUM; }
{id} { yylval.sval = strdup(yytext); return ID; }
[\t\n ]+ ; // ignore whitespace
"=" { return '='; }
"+" { return '+'; }
"-" { return '-'; }
"*" { return '*'; }
"/" { return '/'; }
";" { return ';'; }
"(" { return '('; }
")" { return ')'; }
. { return yytext[0]; }
%%
int yywrap() { return 1; }
```

Yacc file

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

int temp_count = 0;
```

```
// Create a new temporary variable
char* newTemp() {
    char* temp = (char*)malloc(10);
    sprintf(temp, "t%d", temp_count++);
    return temp;
}

// Generate intermediate code
void gen(char* op, char* arg1, char* arg2, char* res) {
    printf("%s = %s %s %s\n", res, arg1, op, arg2);
}

void yyerror(const char *s);
int yylex(void);
%}

%union {
    int ival;
    char* sval;
}

%token <ival> NUM
%token <sval> ID
%left '+' '-'
%left '*' '/'
%type <sval> expr term factor stmt

%%
prog:
      stmt_list
;

stmt_list:
      stmt_list stmt
    | stmt
;

stmt:
      ID '=' expr ';' {
            printf("%s = %s\n", $1, $3);
            free($1);
            free($3);
      }
    | expr ';' { free($1); }
;

expr:
      expr '+' term {
            char* t = newTemp();
```

```
            gen("+", $1, $3, t);
            free($1); free($3);
            $$ = t;
        }
    | expr '-' term {
            char* t = newTemp();
            gen("-", $1, $3, t);
            free($1); free($3);
            $$ = t;
        }
    | term { $$ = $1; }
;

term:
      term '*' factor {
            char* t = newTemp();
            gen("*", $1, $3, t);
            free($1); free($3);
            $$ = t;
        }
    | term '/' factor {
            char* t = newTemp();
            gen("/", $1, $3, t);
            free($1); free($3);
            $$ = t;
        }
    | factor { $$ = $1; }
;

factor:
      '(' expr ')' { $$ = $2; }
    | NUM {
            char* t = (char*)malloc(10);
            sprintf(t, "%d", $1);
            $$ = t;
        }
    | ID {
            $$ = strdup($1);
        }
;
%%

int main() {
    printf("Enter arithmetic expressions:\n");
    yyparse();
    return 0;
}

void yyerror(const char *s) {
```

```
    printf("Error: %s\n", s);
}
```

```
satviki@sandip:~$ lex a10_2.l
satviki@sandip:~$ yacc a10_2.y
satviki@sandip:~$ yacc a10_2.y -d
satviki@sandip:~$ gcc lex.yy.c y.tab.c
satviki@sandip:~$ ./a.out
Enter arithmetic expressions:
x = (2 + 3) * (4 - 1);
t0 = 2 + 3
t1 = 4 - 1
t2 = t0 * t1
x = t2
```

1. **YACC program for IC generation for the expression involving programming constructs.**

```
%{
#include "y.tab.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
%}

digit   [0-9]+
id      [a-zA-Z_][a-zA-Z0-9_]*

%%
{digit}        { yylval.ival = atoi(yytext); return NUM; }
{id}           { yylval.sval = strdup(yytext); return ID; }
"("            { return '('; }
")"            { return ')'; }
"+"            { return '+'; }
"-"            { return '-'; }
"*"            { return '*'; }
"/"            { return '/'; }
"="            { return '='; }
[\t ]+         ;    /* ignore whitespace */
\n+            { return '\n'; }  /* treat newline as end of statement */
.              { return yytext[0]; }
%%

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

```
satviki@sandip:~$ lex i.l
satviki@sandip:~$ yacc a.y -d
satviki@sandip:~$ gcc lex.yy.c y.tab.c
satviki@sandip:~$ ./a.out
Enter program statements (press Enter after each statement, C
trl+D to finish):
a = 5 + 2
b = a * (3 + 4)
c = b t0 = 5 + 2
a = t0
t1 = 3 + 4
t2 = a * t1

b = t2
```

Assignment 11

1.LEX program to implement a simple calculator.

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

int num1 = 0, num2 = 0;
char op = '\0';
%}

%%
[0-9]+ {
   if (num1 == 0)
      num1 = atoi(yytext);
   else
      num2 = atoi(yytext);
}

[+\-*/]  { op = yytext[0]; }

\n {
   switch(op) {
      case '+': printf("Result = %d\n", num1 + num2); break;
      case '-': printf("Result = %d\n", num1 - num2); break;
      case '*': printf("Result = %d\n", num1 * num2); break;
      case '/':
         if (num2 != 0)
            printf("Result = %d\n", num1 / num2);
         else
            printf("Error: Division by zero\n");
         break;
      default:
```

```
        printf("Invalid operation\n");
    }
    num1 = num2 = 0;
    op = '\0';
}

.   ;   /* Ignore other characters */
%%

int yywrap() { return 1; }

int main() {
    printf("Enter expression (e.g., 12+4) and press Enter:\n");
    yylex();
    return 0;
}
```

```
satviki@sandip:~$ lex a11_1.l
satviki@sandip:~$ gcc lex.yy.c
satviki@sandip:~$ ./a.out
Enter expression (e.g., 12+4) and press Enter:
12+4
Result = 16
```

2. LEX program for only checking digits and operators.

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

%%
[0-9]+       { printf("DIGIT: %s\n", yytext); }
[\+\-\*/]    { printf("OPERATOR: %s\n", yytext); }
[ \t\n]+     ;    /* Ignore whitespace */
.            { printf("UNKNOWN: %s\n", yytext); }
%%

int main() {
    printf("Enter expression: ");
    yylex();  /* Start scanning */
    return 0;
}

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

3. LEX based calculator for three operands and two operators based expression evaluation.

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

int n1 = 0, n2 = 0, n3 = 0;
char op1 = 0, op2 = 0;
%}

%%
[0-9]+ {
   if (n1 == 0)
      n1 = atoi(yytext);
   else if (n2 == 0)
      n2 = atoi(yytext);
   else
      n3 = atoi(yytext);
}

[+\-*/] {
   if (op1 == 0)
      op1 = yytext[0];
   else
      op2 = yytext[0];
}

[ \t]+    ;   /* ignore spaces */

\n {
   int temp, result;

   // Handle simple precedence: * and / before + and -
   if ((op1 == '*' || op1 == '/') && (op2 == '+' || op2 == '-')) {
      temp = (op1 == '*') ? n1 * n2 : n1 / n2;
      result = (op2 == '+') ? temp + n3 : temp - n3;
   }
```

```
    else if ((op2 == '*' || op2 == '/') && (op1 == '+' || op1 == '-')) {
        temp = (op2 == '*') ? n2 * n3 : n2 / n3;
        result = (op1 == '+') ? n1 + temp : n1 - temp;
    }
    else {
        // If same precedence → evaluate left to right
        if (op1 == '+') temp = n1 + n2;
        else if (op1 == '-') temp = n1 - n2;
        else if (op1 == '*') temp = n1 * n2;
        else temp = n1 / n2;

        if (op2 == '+') result = temp + n3;
        else if (op2 == '-') result = temp - n3;
        else if (op2 == '*') result = temp * n3;
        else result = temp / n3;
    }

    printf("Result = %d\n", result);

    // Reset values for next expression
    n1 = n2 = n3 = 0;
    op1 = op2 = 0;
}

.   ;   /* ignore any other characters */
%%

int yywrap() { return 1; }

int main() {
    printf("Enter expression with 3 operands and 2 operators (e.g., 2+3*4):\n");
    yylex();
    return 0;
}
```

```
satviki@sandip:~$ lex a.l
satviki@sandip:~$ gcc lex.yy.c
satviki@sandip:~$ ./a.out
Enter expression with 3 operands and 2 operators:
2 + 3 * 4
Result = 14
```

Assignment 12

1. LEX program to check a valid email address.

```
%{
```

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

%%
[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,} {
   printf("Valid Email: %s\n", yytext);
}

\n      ;   /* Ignore newlines */
.       { printf("Invalid Input: %s\n", yytext); }
%%

int main() {
   printf("Enter email addresses (Ctrl+D to end):\n");
   yylex();
   return 0;
}

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

```
satviki@sandip:~$ lex a12.l
satviki@sandip:~$ gcc lex.yy.c
satviki@sandip:~$ ./a.ouy
-bash: ./a.ouy: No such file or directory
satviki@sandip:~$ ./a.out
Enter email addresses (Ctrl+D to end):
abc@g.com
Valid Email: abc@g.com
^C
satviki@sandip:~$ ./a.out
Enter email addresses (Ctrl+D to end):
abc
Invalid Input: a
Invalid Input: b
Invalid Input: c
```

2. LEX program for checking all domain types email addresses.

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

%%
[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}(\.[a-zA-Z]{2,})? {
   printf("Valid Email: %s\n", yytext);
```

```
}

\n      ;   /* Ignore newlines */
.       { printf("Invalid Input: %s\n", yytext); }
%%

int main() {
   printf("Enter email addresses (Ctrl+D to end):\n");
   yylex();
   return 0;
}

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

```
satviki@sandip:~$ lex b.l
satviki@sandip:~$ gcc lex.yy.c
satviki@sandip:~$ ./a.out
Enter email addresses (Ctrl+D to end):
satviki@sandip.com
user@domain.co.in
test@university.ac.uk
invalValid Email: satviki@sandip.com
Valid Email: user@domain.co.in
Valid Email: test@university.ac.uk
```

3. LEX program for checking all domain types email addresses through file handling concept.

```
%{
#include <stdio.h>
#include <stdlib.h>
FILE *inputFile;
%}

%%
[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}(\.[a-zA-Z]{2,})* {
   fprintf(stdout, "Valid Email: %s\n", yytext);
}
\n      ;
.       { fprintf(stdout, "Invalid Input: %s\n", yytext); }
%%

int main(int argc, char *argv[]) {
   if (argc < 2) {
      printf("Usage: %s <input_file>\n", argv[0]);
```

```
        return 1;
    }

    inputFile = fopen(argv[1], "r");
    if (!inputFile) {
        perror("Error opening file");
        return 1;
    }

    yyin = inputFile;
    yylex();
    fclose(inputFile);
    return 0;
}

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

Input.txt

```
john@example.com
contact@mail.co.in
invalid@@domain
```