1. **The lexical analyzer should ignore redundant spaces, tabs and new lines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value. Write a LEX specification file to take input C program from a .c file and count t he number of characters, number of lines & number of words.**

**Input Source Program: (sample.c)**

**#include <stdio.h>**

**int main()**

**{**

**int number1, number2, sum;**

**printf("Enter two integers: ");**

**scanf("%d %d", &number1, &number2);**

**sum = number1 + number2;**

**printf("%d + %d = %d", number1, number2, sum);**

**return 0;**

**}**

%{

int nline,nchar,nword;

%}

%%

[\n] { nline++; nchar+=yyleng;}

[^\t\n ]+ { nword++; nchar+=yyleng;}

.|\n { nchar++; }

%%

int yywrap(void)

{

return 1;

}

int main(int argc,char \*argv[])

{

yyin=fopen(argv[1],"r");

yylex();

printf("number of line =%d\n",nline);

printf("number of words =%d\n",nword);

printf("number of characters =%d\n",nchar);

fclose(yyin);

}

1. **Write a LEX program to count the number of comment lines in a given C program and eliminate them and write into another file.**

# **Input Source File: (input.c)**

**#include<stdio.h>**

**int main()**

**{**

**int a,b,c; /\*varible declaration\*/ printf(“enter two numbers”); scanf(“%d %d”,&a,&b); c=a+b;//adding two numbers printf(“sum is %d”,c);**

**return 0;**

**}**

%{

int com = 0;

%}

%%

"/\*"[^\n]+"\*/" {com++; fprintf(yyout," ");}

\/\/.\* {; com++; fprintf(yyout," ");}

%%

void main(int argc, char \*argv[])

{

yyin=fopen(argv[1],"r");

yyout=fopen(argv[2],"w");

yylex();

printf("\n number of comments are = %d\n",com);

}

int yywrap()

{

return 1;

}

1. **Write a LEX Program to convert the substring abc to ABC from the given input string.**

%{

%}

%%

[a-z] {printf("%c",yytext[0]-32);}

. {}

%%

int yywrap(void){}

int main()

{

printf("\nenter the string : ");

yylex();

}

1. **In a class, an English teacher was teaching the vowels and consonants to the students. She says “Vowel sounds allow the air to flow freely, causing the chin to drop noticeably, whilst consonant sounds are produced by restricting the airflow”. As a class activity, the students are asked to identify the vowels and consonants in the given word/sentence and count the number of elements in each. Write an algorithm to help the student to count the number of consonants in the given sentence.**

%{

#include<stdio.h>

int v=0;

int c=0;

%}

%%

[ \t\n]+;

[aeiouAEIOU] {v++;}

[^aeiouAEIOU] {c++;}

%%

int main( )

{

printf ("Enter the input String :\n");

yylex();

printf("no of vowels are %d\n",v);

printf("No of consonants are %d \n",c);

}

int yywrap( )

{

return 1;

}

1. **Keywords are predefined, reserved words used in programming that have special meanings to the compiler. Keywords are part of the syntax and they cannot be used as an identifier. In general, there are 32 keywords. The prime function of Lexical Analyser is token Generation. Among the 6 types of tokens, differentiating Keywords and identifiers is a challenging issue. Thus write a LEX program to separate keywords and identifiers.**

%{

#include<stdio.h>

%}

%%

if|else|while|do {printf("%s is a keyword", yytext);}

[0-9]+ {printf("%s is a digit", yytext);}

[a-z A-Z]+ {printf("%s is an alphabet", yytext);}

. ;

%%

int main()

{

printf("\n Enter the input string:");

yylex();

}

int yywrap()

{

return 1;

}

1. **Write a LEX program to identify and count positive and negative numbers.**

%{

int positive\_no = 0, negative\_no = 0;

%}

%%

^[-][0-9]+ {negative\_no++;

printf("negative number = %s\n",

yytext);}

[0-9]+ {positive\_no++;

printf("positive number = %s\n",

yytext);}

%%

int yywrap(){}

int main()

{

yylex();

printf ("number of positive numbers = %d,"

"number of negative numbers = %d\n",

positive\_no, negative\_no);

return 0;

}

1. **A School student was asked to do basic mathematical operations. Implement a LEX program to implement the same.**

%{

#include<stdio.h>

int op=0;

int i;

float a,b;

int digit();

%}

digit [0-9]+|([0-9]\*)"."([0-9]+)

add "+"

sub "-"

mul "\*"

div "/"

pow "^"

ln \n

%%

{digit} {digit();}

{add} {op=1;}

{sub} {op=2;}

{mul} {op=3;}

{div} {op=4;}

{pow} {op=5;}

{ln} {printf("\n The answer :%f\n\n",a);}

%%

int digit()

{

if(op==0)

a=atof(yytext);

else

{

b=atof(yytext);

switch(op)

{

case 1:a=a+b;

break;

case 2:a=a-b;

break;

case 3:a=a\*b;

break;

case 4:a=a/b;

break;

case 5:for(i=a;b>1;b--)

a=a\*i;

break;

}

op=0;

}

}

int main()

{

printf("enter operation along with values");

yylex();

}

int yywrap()

{ }

1. **Write a Lex program to count the frequency of the given word in a given sentence.**

%{

#include<stdio.h>

#include<string.h>

char word [] = "best";

int count = 0;

%}

%%

[a-zA-Z]+ { if(strcmp(yytext, word)==0)

count++; }

. ;

%%

int yywrap()

{

return 1;

}

int main()

{

extern FILE \*yyin, \*yyout;

yyin=fopen("a.txt", "r");

yylex();

printf("%d", count);

}

1. **Write a Lex program to find the length of the longest word.**

%{

#include<stdio.h>

int k=0;

%}

%%

[a-zA-Z]+ {

if(yyleng>k)

{ k= yyleng;

}

}

%%

int yywrap(void)

{

return 1;

}

int main(int argc[],char \*\*argv[])

{

yyin=fopen("a.txt","r");

yylex();

printf("largest: %d",k);

printf("\n");

return 0;

}

1. **Write a Lex code to replace a word with another word in a file.**

%{

#include<stdio.h>

#include<string.h>

char replace\_with [] = "Best";

char replace [] ="A";

%}

%%

[a-zA-Z]+ { if(strcmp(yytext, replace)==0)

fprintf(yyout, "%s", replace\_with);

else

fprintf(yyout, "%s", yytext);}

. fprintf(yyout, "%s", yytext);

%%

int yywrap()

{

return 1;

}

int main()

{

extern FILE \*yyin, \*yyout;

yyin=fopen("a.txt", "r");

yyout=fopen("b.txt", "w");

yylex();

}

1. **Extend the lexical Analyzer to Check comments, defined as follows in C:**

**a) A comment begins with // and includes all characters until the end of that line.**

**b) A comment begins with /\* and includes all characters through the next occurrence of the character sequence \*/Develop a lexical Analyzer to identify whether a given line is a comment or not.**

%{

#include<stdio.h>

%}

%%

[/]{1}[/]{1}[a-zA-Z0-9]\* {printf("Single Line Comment", yytext);}

[/]{1}[\*]{1}[a-zA-Z0-9]\*[\*]{1}[/]{1} {printf("MultiLine Comment", yytext);}

%%

int main()

{

printf("\n Enter the input string:");

yylex();

}

int yywrap()

{

return 1;

}

1. **Design a lexical Analyzer to find the number of whitespaces and newline characters.**

%{

#include<stdio.h>

int sc=0,wc=0,lc=0,cc=0;

%}

%%

[\n] { lc++; cc+=yyleng;}

[ \t] { sc++; cc+=yyleng;}

[^\t\n ]+ { wc++; cc+=yyleng;}

%%

int main(int argc ,char\* argv[])

{

if(argc==2)

{

yyin=fopen(argv[1],"r");

}

else

{

printf("Enter the input\n");

yyin=stdin;

}

yylex();

printf("The number of new lines=%d\n",lc);

printf("The number of spaces=%d\n",sc);

}

int yywrap( )

{

return 1;

}

1. **Develop a lexical Analyzer to test whether a given identifier is valid or not.**

%{

#include<stdio.h>

%}

%%

([a-zA-Z][0-9])+|[a-zA-Z]\* {printf("Identifier\n");}

^[0-9]+ {printf("Not a Identifier\n");}

.|\n;

%%

int yywrap()

{

return 1;

}

int main(void)

{

yylex();

return 0;

}