1. Write a program to count the number of lines, tabs, spaces, words,

characters from a given text title in c++

#include <bits/stdc++.h>

using namespace std;

int main() {

string filename;

cout << "File ka naam to batao tab manu ";

cin >> filename;

ifstream file(filename);

int lineCount = 0;

int tabCount = 0;

int spaceCount = 0;

int wordCount = 0;

int charCount = 0;

char c;

while (file.get(c)) {

if (c == '\n')

{

lineCount++;

wordCount++;

}

else if (c == '\t')

{

tabCount++;

wordCount++;

}

else if (c == ' ')

{

spaceCount++;

wordCount++;

}

else

charCount++;

}

file.close();

cout << "Number of lines: " << lineCount << endl;

cout << "Number of tabs: " << tabCount << endl;

cout << "Number of spaces: " << spaceCount << endl;

cout << "Number of words: " << wordCount << endl;

cout << "Number of characters: " << charCount << endl;

return 0;

}

2. Implement the Lexical analyzer for the given language. 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.

#include <bits/stdc++.h>

using namespace std;

int main() {

string filename;

cout << "File ka naam to bata bhai ";

cin >> filename;

ifstream file(filename);

string line;

string s;

while (getline(file, line)) {

s += line + '\n';

}

file.close();

bool flag = false;

string token;

for (char c : s)

{

if (!flag && c == '/' && token.empty())

{

token += c;

continue;

}

if (flag && c == '\n')

{

flag = false;

continue;

}

if (flag)

continue;

if (c == ' ' || c == '\t')

{

if (!token.empty())

{

cout << "Ye lo bhai token " << token << endl;

token.clear();

}

continue;

}

if (!token.empty() && token.back() == '/' && c == '/')

{

token.clear();

flag = true;

continue;

}

token += c;

}

if (!token.empty())

cout << "Ye lo bhai token " << token << endl;

return 0;

}

3. Write a lex program to count number of characters, words and lines in a given input text file. Create an output text file that consists of the content of the input file as well as line numbers.  
  
%{

#include <stdio.h>

#include <stdlib.h>

int char\_count = 0;

int word\_count = 0;

int line\_count = 0;

FILE \*output\_file;

void print\_line\_number();

%}

%%

[^ \t\n]+ { word\_count++; char\_count += yyleng; fprintf(output\_file, "%s", yytext); }

\n { line\_count++; char\_count++; fprintf(output\_file, "%s", yytext); print\_line\_number(); }

. { char\_count++; fprintf(output\_file, "%s", yytext); }

%%

int yywrap() {

return 1;

}

void print\_line\_number() {

fprintf(output\_file, "%d ", line\_count + 1);

}

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

if (argc != 3) {

fprintf(stderr, "Usage: %s <input file> <output file>\n", argv[0]);

exit(1);

}

FILE \*input\_file = fopen(argv[1], "r");

if (!input\_file) {

perror("Unable to open input file");

exit(1);

}

output\_file = fopen(argv[2], "w");

if (!output\_file) {

perror("Unable to open output file");

fclose(input\_file);

exit(1);

}

// Redirect lex input to the input file

yyin = input\_file;

// Initialize line number for output

fprintf(output\_file, "1 ");

// Start the lexer

yylex();

// Print the final counts

printf("Characters: %d\n", char\_count);

printf("Words: %d\n", word\_count);

printf("Lines: %d\n", line\_count);

fclose(input\_file);

fclose(output\_file);

return 0;

}

4. Write LEX specifications and necessary C code that reads English words from a text file and calculates the count of words that starts with a vowel. The program appends the current value of the counter to every occurrences of such word. The program should also compute total numbers of words read.

%{

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

int vowel\_count = 0;

int total\_word\_count = 0;

FILE \*output\_file;

void append\_vowel\_count(const char\* word);

%}

%%

[AEIOUaeiou][a-zA-Z]\* {

vowel\_count++;

total\_word\_count++;

append\_vowel\_count(yytext);

}

[a-zA-Z]+ {

total\_word\_count++;

fprintf(output\_file, "%s", yytext);

}

[^a-zA-Z]+ { fprintf(output\_file, "%s", yytext); }

%%

int yywrap() {

return 1;

}

void append\_vowel\_count(const char\* word) {

fprintf(output\_file, "%s%d", word, vowel\_count);

}

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

if (argc != 3) {

fprintf(stderr, "Usage: %s <input file> <output file>\n", argv[0]);

exit(1);

}

FILE \*input\_file = fopen(argv[1], "r");

if (!input\_file) {

perror("Unable to open input file");

exit(1);

}

output\_file = fopen(argv[2], "w");

if (!output\_file) {

perror("Unable to open output file");

fclose(input\_file);

exit(1);

}

// Redirect lex input to the input file

yyin = input\_file;

// Start the lexer

yylex();

// Print the final counts

printf("Total words: %d\n", total\_word\_count);

printf("Words starting with a vowel: %d\n", vowel\_count);

fclose(input\_file);

fclose(output\_file);

return 0;

}

5.Write LEX specifications and necessary C code that reads English words from a text file and replaces every occurrence of the sub string ‘abc’ with ‘ABC’. The program should also compute number of characters, words and lines read. It should not consider and count any line(s) that begin with a symbol “#”.Write LEX specifications and necessary C code that reads English words from a text file and replaces every occurrence of the sub string ‘abc’ with ‘ABC’. The program should also compute number of characters, words and lines read. It should not consider and count any line(s) that begin with a symbol “#”.

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int char\_count = 0;

int word\_count = 0;

int line\_count = 0;

int skip\_line = 0;

FILE \*output\_file;

void increment\_counts(const char \*str);

%}

%%

#.\*\n { skip\_line = 1; } // Skip lines starting with #

\n { if (!skip\_line) { line\_count++; char\_count++; } skip\_line = 0; fprintf(output\_file, "%s", yytext); }

abc { if (!skip\_line) { char\_count += 3; fprintf(output\_file, "ABC"); } }

[a-zA-Z]+ { if (!skip\_line) { increment\_counts(yytext); } fprintf(output\_file, "%s", yytext); }

. { if (!skip\_line) { char\_count++; } fprintf(output\_file, "%s", yytext); }

%%

int yywrap() {

return 1;

}

void increment\_counts(const char \*str) {

word\_count++;

char\_count += strlen(str);

}

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

if (argc != 3) {

fprintf(stderr, "Usage: %s <input file> <output file>\n", argv[0]);

exit(1);

}

FILE \*input\_file = fopen(argv[1], "r");

if (!input\_file) {

perror("Unable to open input file");

exit(1);

}

output\_file = fopen(argv[2], "w");

if (!output\_file) {

perror("Unable to open output file");

fclose(input\_file);

exit(1);

}

// Redirect lex input to the input file

yyin = input\_file;

// Start the lexer

yylex();

// Print the final counts

printf("Characters: %d\n", char\_count);

printf("Words: %d\n", word\_count);

printf("Lines: %d\n", line\_count);

fclose(input\_file);

fclose(output\_file);

return 0;

}

6. Write a program for code generation.

#include <bits/stdc++.h>

#include <cstdint>

enum class Opcode : uint8\_t {

ADD = 0x01,

SUB = 0x02,

MUL = 0x03,

DIV = 0x04,

};

std::vector<uint8\_t> generateMachineCode(Opcode op, int operand1, int operand2) {

std::vector<uint8\_t> code;

code.push\_back(static\_cast<uint8\_t>(op));

code.push\_back(static\_cast<uint8\_t>(operand1));

code.push\_back(static\_cast<uint8\_t>(operand2));

return code;

}

int main() {

Opcode op = Opcode::ADD;

int operand1 = 10;

int operand2 = 20;

std::vector<uint8\_t> machineCode = generateMachineCode(op, operand1, operand2);

std::cout << "Generated machine code:";

for (uint8\_t byte : machineCode) {

std::cout << " " << std::hex << static\_cast<int>(byte);

}

std::cout << std::endl;

return 0;

}