Program 1 : Count number of lines, tabs, spaces, words, characters

```
#include <iostream>
#include <fstream>
#include <string>
#include <cctype>

int main() {
    std::ifstream file("example.txt"); // Replace with your file name
    if (!file.is_open()) {
        std::cerr << "Could not open the file." << std::endl;
        return 1;
    }

    int lines = 0, tabs = 0, spaces = 0, words = 0, characters = 0;
    std::string word;
    char ch;

    while (file.get(ch)) {
        characters++; // Count every character

        if (ch == '\n') {
            lines++; // Count new lines
        } else if (ch == '\t') {
            tabs++; // Count tabs
        } else if (ch == ' ') {
            spaces++; // Count spaces
        }

        if (std::isspace(ch) || ch == '\n' || ch == '\t') {
            file >> std::ws; // Skip whitespace before reading next word
            if (file.peek() != EOF) {
                words++; // Count words
            }
        }
    }

    // Account for the last word in the file if it doesn't end with a newline
    if (!std::isspace(ch)) {
        words++;
    }

    // Account for the last line in the file if it doesn't end with a newline
    if (ch != '\n') {
        lines++;
    }

    file.close();

    std::cout << "Lines: " << lines << std::endl;
    std::cout << "Tabs: " << tabs << std::endl;
    std::cout << "Spaces: " << spaces << std::endl;
    std::cout << "Words: " << words << std::endl;
    std::cout << "Characters: " << characters << std::endl;
```

```
    return 0;
}
```

Program 2 : redundant files

```
#include <iostream>
#include <vector>
#include <string>
#include <regex>
#include <sstream>

enum TokenType {
    KEYWORD, IDENTIFIER, INTEGER, OPERATOR, DELIMITER, COMMENT, WHITESPACE,
UNKNOWN
};

struct Token {
    TokenType type;
    std::string value;
    int line;
    int column;

    Token(TokenType type, const std::string& value, int line, int column)
        : type(type), value(value), line(line), column(column) {}
};

class Lexer {
public:
    Lexer(const std::string& input) : input(input), line(1), column(1) {}

    std::vector<Token> tokenize() {
        std::vector<Token> tokens;
        std::regex tokenRegex(tokenPatterns);
        auto words_begin = std::sregex_iterator(input.begin(), input.end(), tokenRegex);
        auto words_end = std::sregex_iterator();

        for (std::sregex_iterator i = words_begin; i != words_end; ++i) {
            std::smatch match = *i;
            std::string matchStr = match.str();
            TokenType type = getTokenType(match);

            if (type != WHITESPACE && type != COMMENT) {
                tokens.emplace_back(type, matchStr, line, column);
            }

            column += matchStr.length();
            if (matchStr.find('\n') != std::string::npos) {
                line++;
                column = 1;
            }
        }

        return tokens;
    }
```

```
private:
    std::string input;
    int line;
    int column;

    std::string tokenPatterns =
        "(\\b(if|else|while|return)\\b)|"      // Keywords
        "(\\b[A-Za-z_][A-Za-z0-9_]*\\b)|"      // Identifiers
        "(\\b\\d+\\b)|"                        // Integer literals
        "([+\\-*/=<>!]+)|"                     // Operators
        "([(){};,])|"                         // Delimiters
        "(//.*|/\\*[\\s\\S]*?\\*/)|"           // Comments
        "(\\s+)|"                              // Whitespace
        "(.)";                                 // Any other character

    TokenType getTokenType(const std::smatch& match) {
        if (match[1].matched) return KEYWORD;
        if (match[2].matched) return IDENTIFIER;
        if (match[3].matched) return INTEGER;
        if (match[4].matched) return OPERATOR;
        if (match[5].matched) return DELIMITER;
        if (match[6].matched) return COMMENT;
        if (match[7].matched) return WHITESPACE;
        return UNKNOWN;
    }
};

int main() {
    std::string code = R"(
    if (x == 10) {
        // This is a comment
        return y + 20;
    }
    )";

    Lexer lexer(code);
    std::vector<Token> tokens = lexer.tokenize();

    for (const auto& token : tokens) {
        std::cout << "Token(" << token.value << ", " << token.line << ":" << token.column << ")\n";
    }

    return 0;
}
```

Program 3 : LEX COUNT

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
```

```
#include <iomanip>

void processFile(const std::string& inputFileName, const std::string& outputFileName) {
    std::ifstream inputFile(inputFileName);
    if (!inputFile) {
        std::cerr << "Unable to open input file: " << inputFileName << std::endl;
        return;
    }

    std::ofstream outputFile(outputFileName);
    if (!outputFile) {
        std::cerr << "Unable to open output file: " << outputFileName << std::endl;
        return;
    }

    std::string line;
    int lineCount = 0;
    int wordCount = 0;
    int charCount = 0;

    while (std::getline(inputFile, line)) {
        lineCount++;
        charCount += line.length() + 1; // Including newline character

        std::istringstream lineStream(line);
        std::string word;
        while (lineStream >> word) {
            wordCount++;
        }

        // Write the line with line number to the output file
        outputFile << std::setw(4) << lineCount << ": " << line << std::endl;
    }

    // Output the counts
    std::cout << "Characters: " << charCount << std::endl;
    std::cout << "Words: " << wordCount << std::endl;
    std::cout << "Lines: " << lineCount << std::endl;

    inputFile.close();
    outputFile.close();
}

int main() {
    std::string inputFileName = "input.txt";
    std::string outputFileName = "output.txt";

    processFile(inputFileName, outputFileName);

    return 0;
}
```

PROGRAM 4 : word occurences

```
%{
#include <iostream>
#include <fstream>
#include <sstream>
#include <cctype>
#include <string>

using namespace std;

int vowel_word_count = 0;
int total_word_count = 0;
string current_vowel_word;

bool starts_with_vowel(const string &word) {
    char first_char = tolower(word[0]);
    return first_char == 'a' || first_char == 'e' || first_char == 'i' || first_char == 'o' || first_char == 'u';
}

void process_word(const string &word) {
    total_word_count++;
    if (starts_with_vowel(word)) {
        vowel_word_count++;
        current_vowel_word = word + to_string(vowel_word_count);
    } else {
        current_vowel_word = word;
    }
}
%}

%%
[a-zA-Z]+ {
    string word(yytext);
    process_word(word);
    cout << current_vowel_word << " ";
}

[ \t\n]+  { /* Ignore whitespace */ }

.       { /* Ignore other characters */ }
%%

int main(int argc, char** argv) {
    if (argc != 2) {
        cerr << "Usage: " << argv[0] << " <input_file>" << endl;
        return 1;
    }

    ifstream input_file(argv[1]);
    if (!input_file.is_open()) {
        cerr << "Error: Could not open file " << argv[1] << endl;
        return 1;
    }
```

```
    stringstream buffer;
    buffer << input_file.rdbuf();
    string input = buffer.str();
    input_file.close();

    YY_BUFFER_STATE buffer_state = yy_scan_string(input.c_str());
    yylex();
    yy_delete_buffer(buffer_state);

    cout << "\nTotal words: " << total_word_count << endl;
    cout << "Words starting with a vowel: " << vowel_word_count << endl;

    return 0;
}
```

PROGRAM 5 : #

```
%{
#include <iostream>
#include <fstream>
#include <string>
#include <cctype>

using namespace std;

int char_count = 0;
int word_count = 0;
int line_count = 0;
bool skip_line = false;

void count_word(const char* yytext) {
    if (!skip_line) {
        word_count++;
    }
}
%}

%%

"abc"          {
                  if (!skip_line) {
                      char_count += 3;
                      cout << "ABC";
                  }
               }

[a-zA-Z]+      {
                  count_word(yytext);
                  if (!skip_line) {
                      char_count += yyleng;
```

```
                cout << yytext;
            }
        }

[ \t]+      {
            if (!skip_line) {
                char_count += yyleng;
                cout << yytext;
            }
        }

\n          {
            if (!skip_line) {
                char_count++;
                cout << yytext;
            }
            line_count++;
            skip_line = false;
        }

"#".*       { skip_line = true; }

.           {
            if (!skip_line) {
                char_count++;
                cout << yytext;
            }
        }

%%

int main(int argc, char** argv) {
    if (argc != 2) {
        cerr << "Usage: " << argv[0] << " <input_file>" << endl;
        return 1;
    }

    ifstream input_file(argv[1]);
    if (!input_file.is_open()) {
        cerr << "Error: Could not open file " << argv[1] << endl;
        return 1;
    }

    string content((istreambuf_iterator<char>(input_file)), istreambuf_iterator<char>());
    input_file.close();

    YY_BUFFER_STATE buffer_state = yy_scan_string(content.c_str());
    yylex();
    yy_delete_buffer(buffer_state);

    cout << "\n\nNumber of characters: " << char_count << endl;
    cout << "Number of words: " << word_count << endl;
    cout << "Number of lines: " << line_count << endl;

    return 0;
```

```
}
```

PROGRAM NO 6 : CODE OPTIMIZER

```
#include<stdio.h>
#include<string.h>
struct op
{
char l;
char r[20];
}
op[10],pr[10];
void main()
{
int a,i,k,j,n,z=0,m,q;
char *p,*l;
char temp,t;
char *tem;
printf("Enter the Number of Values:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("left: ");
scanf(" %c",&op[i].l);
printf("right: ");
scanf(" %s",&op[i].r);
}
printf("Intermediate Code\n") ;
for(i=0;i<n;i++)
{
printf("%c=",op[i].l);
printf("%s\n",op[i].r);
}
for(i=0;i<n-1;i++)
{
temp=op[i].l;
for(j=0;j<n;j++)
{
p=strchr(op[j].r,temp);
if(p)
{
pr[z].l=op[i].l;
strcpy(pr[z].r,op[i].
r);
z++;
}
}
}
pr[z].l=op[n-1].l;
strcpy(pr[z].r,op[n-1].r);
z++;
printf("\nAfter Dead Code Elimination\n");
for(k=0;k<z;k++)
```

```
{
printf("%c\t=",pr[k].l);
printf("%s\n",pr[k].r);
}
for(m=0;m<z;m++)
{
tem=pr[m].r;
for(j=m+1;j<z;j++)
{
p=strstr(tem,pr[j].r);
if(p)
{
t=pr[j].l;
pr[j].l=pr[m].l;
for(i=0;i<z;i++)
{
l=strchr(pr[i].r,t) ;
if(l)
{
a=l-pr[i].r;
printf("pos: %d\n",a);
pr[i].r[a]=pr[m].l;
}}}}}
printf("Eliminate Common Expression\n");
for(i=0;i<z;i++)
{
printf("%c\t=",pr[i].l);
printf("%s\n",pr[i].r);
}
for(i=0;i<z;i++)
{
for(j=i+1;j<z;j++)
{
q=strcmp(pr[i].r,pr[j].r);
if((pr[i].l==pr[j].l)&&!q)
{
pr[i].l='\0';
}
}
}
printf("Optimized Code\n");
for(i=0;i<z;i++)
{
if(pr[i].l!='\0')
{
printf("%c=",pr[i].l);
printf("%s\n",pr[i].r);
}
}
}
```

PROGRAM 7 : CODE GENERATION

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char op[2],arg1[5],arg2[5],result[5];
void main()
{
  FILE *fp1,*fp2;
  fp1=fopen("input.txt","r");
  fp2=fopen("output.txt","w");
  while(!feof(fp1))
  {

    fscanf(fp1,"%s%s%s%s",op,arg1,arg2,result);
    if(strcmp(op,"+")==0)
    {
      fprintf(fp2,"\nMOV R0,%s",arg1);
      fprintf(fp2,"\nADD R0,%s",arg2);
      fprintf(fp2,"\nMOV %s,R0",result);
    }
     if(strcmp(op,"*")==0)
    {
      fprintf(fp2,"\nMOV R0,%s",arg1);
      fprintf(fp2,"\nMUL R0,%s",arg2);
      fprintf(fp2,"\nMOV %s,R0",result);
    }
    if(strcmp(op,"-")==0)
    {
      fprintf(fp2,"\nMOV R0,%s",arg1);
      fprintf(fp2,"\nSUB R0,%s",arg2);
      fprintf(fp2,"\nMOV %s,R0",result);
    }
       if(strcmp(op,"/")==0)
    {
      fprintf(fp2,"\nMOV R0,%s",arg1);
      fprintf(fp2,"\nDIV R0,%s",arg2);
      fprintf(fp2,"\nMOV %s,R0",result);
    }
if(strcmp(op,"=")==0)
    {
      fprintf(fp2,"\nMOV R0,%s",arg1);
      fprintf(fp2,"\nMOV %s,R0",result);
    }
    }
    fclose(fp1);
    fclose(fp2);
    getch();
  }
}
```

input.txt

```
+ a b t1
* c d t2
- t1 t2 t
= t ? x
```

```
flex word_counter.l
g++ lex.yy.c -o word_counter -lfl
./word_counter input.txt
```