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>
int charCount = 0;
int wordCount = 0;
int lineCount = 0;
%}

%%

\n      { charCount++; wordCount++; lineCount++; fprintf(yyout, "%d: %s",
lineCount, yytext); }
[^\n]+  { charCount += yyleng; wordCount++; fprintf(yyout, "%d: %s",
lineCount, yytext); }

%%

int main(int argc, char *argv[]) {
    FILE *inputFile, *outputFile;

    if (argc != 3) {
        printf("Usage: %s input_file output_file\n", argv[0]);
        return 1;
    }

    inputFile = fopen(argv[1], "r");
    if (!inputFile) {
        printf("Error opening input file.\n");
        return 1;
    }

    outputFile = fopen(argv[2], "w");
    if (!outputFile) {
        printf("Error creating output file.\n");
        fclose(inputFile);
        return 1;
    }

    yyin = inputFile;
    yyout = outputFile;

    yylex();

    printf("Number of characters: %d\n", charCount);
    printf("Number of words: %d\n", wordCount);
    printf("Number of lines: %d\n", lineCount);

    fclose(inputFile);
    fclose(outputFile);

    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>
int totalWords = 0;
int vowelWords = 0;
%}
```

```
%%

[ \t\n]+    ; // Skip whitespace
[aAeEiIoOuU][a-zA-Z]*   { printf("%s_%d ", yytext, ++vowelWords);
totalWords++; }
[a-zA-Z]+   { totalWords++; }

%%

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

    FILE *file = fopen(argv[1], "r");
    if (!file) {
        printf("Error opening input file.\n");
        return 1;
    }

    yyin = file;

    yylex();

    printf("\nTotal words: %d\n", totalWords);
    printf("Words starting with a vowel: %d\n", vowelWords);

    fclose(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 “#”.

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

%%

[ \t\n]+    ; // Skip whitespace
[aAeEiIoOuU][a-zA-Z]*   { printf("%s_%d ", yytext, ++vowelWords);
totalWords++; }
[a-zA-Z]+   { totalWords++; }

%%

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

    FILE *file = fopen(argv[1], "r");
    if (!file) {
        printf("Error opening input file.\n");
        return 1;
    }

    yyin = file;

    yylex();

    printf("\nTotal words: %d\n", totalWords);
    printf("Words starting with a vowel: %d\n", vowelWords);

    fclose(file);

    return 0;
}
```

5. Write a program for code optimization.

Isme koi bhi code likh ke uska optimized version likh aao jaise for example
Before : Linear Search
After : Binary Search
Aise bas yahi hai code optimization

5. Write a program for code generation.

```cpp
#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;
}
```