BHARATIYA VIDYA BHAVAN'S

SARDAR PATEL INSTITUTE OF TECHNOLOGY

(Empowered Autonomous Institute Affiliated to University of Mumbai)

[Knowledge is Nectar]

Department of Computer Engineering

Course - System Programming and Compiler Construction (SPCC)

UID	2021300126
Name	Pranay Singhvi
Class and Batch	TE Computer Engineering Class B - Batch B3
Date	17-01-2024
Lab #	1
Aim	Design a Lexical analyser for different programming languages and implement using lex tool.
Objective	Design and implement a Lexical Analyzer using the Lex tool to facilitate the efficient and accurate tokenization of various programming languages. The analyzer should be capable of recognizing and categorizing lexical elements such as keywords, identifiers, operators, and literals, providing a foundational component for subsequent phases in the compiler construction process.
Theory	**Understanding Tokens and Lexical Analysis in C** Token: A token is the smallest meaningful unit of code in a programming language. It can be a keyword (like `if` or `while`), an identifier (variable name like `age`), a literal (number like `10` or string like `"Hello"`), a symbol (operator like `+` or `*`), or a punctuation mark (semicolon `;`). Tokens are the building blocks of programs, and lexical analysis is the process of identifying and classifying them. Different Tokens in C: • Keywords: Reserved words with specific meanings in the language (`if`, `while`, `for`, etc.). • Identifiers: User-defined names for variables, functions, and other entities. • Literals: Values like numbers (`10`, `3.14`), strings (`"Hello"`, `"World"`), and characters (`'a'`).

- Symbols: Operators for arithmetic (`+`, `-`, `*`), comparison (`<`, `>`, `==`), logical (`&&`, `||`), and other operations.
- Punctuation: Delimiters like semicolons (;), commas (,), braces ({}), parentheses (), brackets [], and others.

Lexical Analysis:

Lexical analysis is the first stage of compilation, responsible for breaking down the source code into a sequence of tokens. It involves:

- Scanning: Reading the characters of the code one by one and identifying potential tokens.
- Tokenization: Grouping characters into meaningful tokens based on the language's grammar and rules.
- Attribute attachment: Adding additional information to each token, like its type (keyword, identifier, etc.) and value (number, string, etc.).

Lexical Analyzer:

A lexical analyzer, also known as a lexer, is a program that performs lexical analysis. It uses a set of rules and algorithms to scan the code, identify tokens, and attach attributes. Common lexer implementations include finite state automata and regular expressions.

Lex Tools:

Lex tools are software tools like Flex in Unix and Lexlib in Windows that help develop lexers. They provide a framework for defining rules and actions for tokenization, making it easier to build custom lexers for different programming languages.

Running a Lex Program:

1. Write the Lex code: Define the rules for tokenization in a `.l` file.

2. Generate the C code: Use the `flex` command to generate a C file from the `.l` file.
3. Compile the C code: Compile the generated C file with a compiler like `gcc` to create an executable file.
4. Run the program: Pass the source code to be analyzed as an argument to the executable.

Good Theory Explanations:

- Focus on the "why" behind each concept: Explain the purpose of tokens, lexical analysis, and lexers in the context of program compilation.
- Use clear and concise language: Avoid technical jargon unless necessary and explain complex concepts in simple terms.
- Provide examples: Illustrate each concept with practical examples from C code to make it easier to understand.
- Connect to practical applications: Explain how lexical analysis is used in real-world compiler development and software tools.

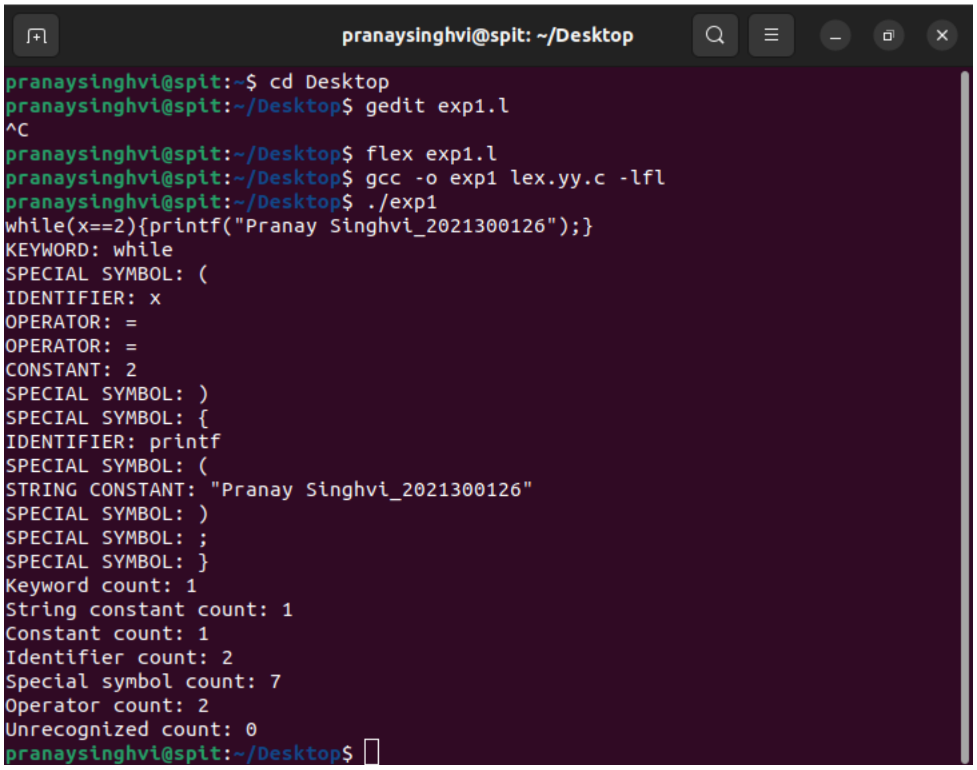
By following these tips, you can provide a good theoretical understanding of tokens and lexical analysis in C programming. Remember, the key is to make the explanation clear, engaging, and relevant to your audience.

Implementation / Code

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

int keywordCount = 0;
int stringCount = 0;
int constantCount = 0;
int identifierCount = 0;
int specialSymbolCount = 0;
int operatorCount = 0;
int unrecognizedCount = 0;
%}

%%
"if"|"else"|"while"|"do"|"break"|"continue"|"int"|"double"|"float"|"return"|"char"|"case"|"sizeof"
|"long"|"short"|"typedef"|"switch"|"unsigned"|"void"|"static"|"struct"|"goto" {
printf("KEYWORD: %s\n", yytext); keywordCount++; }
```

```
\"[^\n\"]*\"    { printf("STRING CONSTANT: %s\n", yytext); stringCount++; }
[0-9]+          { printf("CONSTANT: %s\n", yytext); constantCount++; }
[a-zA-Z_][a-zA-Z0-9_]* { printf("IDENTIFIER: %s\n", yytext); identifierCount++; }
[ \t\n]         /* Ignore whitespace */
[{}()[\],;.]    { printf("SPECIAL SYMBOL: %s\n", yytext); specialSymbolCount++; }
[-+*/%=&|^<>!~]=?  { printf("OPERATOR: %s\n", yytext); operatorCount++; }
[@.]             { printf("UNRECOGNIZED: %s\n", yytext); unrecognizedCount++; }

%%

int main() {
    yylex();
    printf("Keyword count: %d\n", keywordCount);
    printf("String constant count: %d\n", stringCount);
    printf("Constant count: %d\n", constantCount);
    printf("Identifier count: %d\n", identifierCount);
    printf("Special symbol count: %d\n", specialSymbolCount);
    printf("Operator count: %d\n", operatorCount);
    printf("Unrecognized count: %d\n", unrecognizedCount);
    return 0;
}
```

Output



```
pranaysinghvi@spit:~$ cd Desktop
pranaysinghvi@spit:~/Desktop$ gedit exp1.l
^C
pranaysinghvi@spit:~/Desktop$ flex exp1.l
pranaysinghvi@spit:~/Desktop$ gcc -o exp1 lex.yy.c -lfl
pranaysinghvi@spit:~/Desktop$ ./exp1
while(x==2){printf("Pranay Singhvi_2021300126");}
KEYWORD: while
SPECIAL SYMBOL: (
IDENTIFIER: x
OPERATOR: =
OPERATOR: =
CONSTANT: 2
SPECIAL SYMBOL: )
SPECIAL SYMBOL: {
IDENTIFIER: printf
SPECIAL SYMBOL: (
STRING CONSTANT: "Pranay Singhvi_2021300126"
SPECIAL SYMBOL: )
SPECIAL SYMBOL: ;
SPECIAL SYMBOL: }
Keyword count: 1
String constant count: 1
Constant count: 1
Identifier count: 2
Special symbol count: 7
Operator count: 2
Unrecognized count: 0
pranaysinghvi@spit:~/Desktop$
```

Conclusion	From this experiment I learnt about tokenization and lexical analysis of C program.
References	[1] Google Bard- Understanding Tokens and Lexical Analysis in C: https://g.co/bard/share/1783d741ab73