# CSPC62
# Compiler Design Lab Report

Topic:

# Design and Implementation of C++ Compiler

By

- Sai Dorababu(106121065)
- Chetan Reddy(106121139)
- Harshith Babu(106121029)

## Introduction:

The development of programming languages has been pivotal in shaping the landscape of modern computing. Among these languages, C++ stands out as a powerful and versatile tool for software development, renowned for its efficiency, performance, and object-oriented features. However, the process of transforming human-readable C++ code into machine-executable instructions requires a critical intermediary: the compiler.

This project report delves into the intricate world of compiler design, focusing specifically on the creation of a C++ compiler. Our aim is to provide a comprehensive understanding of the underlying principles, methodologies, and challenges involved in building such a fundamental tool.

Throughout this report, we will explore the various phases of compiler construction, starting from lexical analysis and syntax parsing, to semantic analysis, intermediate code generation, optimization, and ultimately, code generation. Each phase plays a crucial role in translating high-level C++ code into efficient machine code that can be executed by a computer.

## Components of the Development Language:

### 1.Keywords:

Keywords, also known as reserved words, are predefined tokens with special meanings in the programming language. These words are reserved by the language and cannot be used for any other purpose such as naming variables or functions. Examples of keywords in C++ include int, double, if, else, for, while, class, struct, return, namespace, public, private, and virtual, among others. Keywords typically represent language constructs, control flow statements, data types, access specifiers, and other fundamental elements of the language.

### 2.Identifiers:

Identifiers are user-defined names used to represent various program elements such as variables, functions, classes, and namespaces. An identifier can consist of letters, digits, and underscores, with the first character being a letter or an underscore.

Identifiers must follow certain rules:

- They cannot be a keyword or contain a keyword.
- They cannot contain spaces or special characters (except underscores).
- They are case-sensitive (e.g., myVariable, MyVariable, and myvariable are considered different identifiers).
- Examples of identifiers include variable names (int age, double balance), function names (void calculateInterest()), class names (class Car, struct Point), and namespace names (namespace MathUtils), among others.

C++ supports various data types such as integers (int, short, long), floating-point numbers (float, double), characters (char), Boolean (bool), and user-defined types (structures, classes, enumerations, unions).

### 3. Control Structures:

Control structures in C++ facilitate decision-making and looping in programs. These include if-else statements, switch-case statements, while loops, do-while loops, and for loops.

## 4.Functions:

Functions are blocks of code that perform specific tasks. In C++, functions can be defined to accept parameters and return values. C++ also supports function overloading, allowing multiple functions with the same name but different parameter lists.

## 5.Classes and Objects:

C++ is an object-oriented programming (OOP) language, and classes are the building blocks of object-oriented design. A class defines a blueprint for creating objects that encapsulate data and behavior. Objects are instances of classes and can interact with each other through member functions and variables.

## 6.Headers:

In C++, headers are files that contain declarations and definitions used by other files in a program. They typically have a .h extension, although it's not a strict requirement. Here's some key information about headers in C++:

Include Directives: Headers are included in C++ source files using the #include directive. For example:

```
#include <iostream>  // Includes the iostream header
```

Contents of Headers: Headers may contain function prototypes, class declarations, constants, macros, template definitions, and inline function implementations.

Standard Library Headers: C++ provides a standard library, often referred to as the Standard Template Library (STL), which contains many pre-defined headers for common functionalities. Examples include:

- iostream: Input/output operations like cout and cin.
- vector: Dynamic array implementation.
- string: String manipulation functions.
- algorithm: Algorithms like sorting and searching.
- cmath: Mathematical functions like sqrt and sin.
- bits/stdc++.h: to include all the libraries

User-Defined Headers: Programmers can create their own headers to organize and modularize their code. These headers typically contain declarations for classes, functions, and other entities defined in corresponding source files.

## 7.Relational Operators:

Relational operators in C++ are symbols used to compare the relationship between two operands. They evaluate to a boolean value (true or false) based on whether the specified relationship holds true. Here are the relational operators in C++:

- Equality (==): Compares if two operands are equal.
- Inequality (!=): Compares if two operands are not equal.

- Greater Than (>): Checks if the left operand is greater than the right operand.
- Less Than (<): Checks if the left operand is less than the right operand.
- Greater Than or Equal To (>=): Checks if the left operand is greater than or equal to the right operand.
- Less Than or Equal To (<=): Checks if the left operand is less than or equal to the right operand.

## 8.Punctuation:

Braces, brackets, and semicolons are punctuation symbols used in C++ to define structure, delineate blocks of code, and terminate statements. Here's a brief explanation of each:

- **Braces {}:**
  Braces are used to define the beginning and end of code blocks in C++. They are commonly used in functions, control flow statements (such as if-else statements, loops), and to define the body of classes, structs, and namespaces.

  Example:

  ```
  if (condition) {
     // Code block
  }
  ```

- **Brackets []:**
  Brackets have multiple uses like Array indexing:- Used to access elements of an array, in Array declaration:- Used to declare arrays, Used in lambda expressions for capture lists.

  Example:

  ```
  int arr[5]; // Declaration of an integer array with 5 elements
  int x = arr[2]; // Accessing the third element of the array
  ```

- **Semicolons ;:**

  Semicolons are used to terminate statements in C++. They are placed at the end of each statement to indicate the end of the statement.

  Example:

  ```
  int x = 5; // Statement assigning the value 5 to variable x
  ```

- **Parentheses ():**

  Parentheses have various uses in C++:

  Function calls:- Used to enclose the arguments passed to a function.

  Expression grouping:- Used to specify the order of evaluation in expressions.

  Control flow statements:- Used to enclose conditions in if, while, for statements.

  Example:

int result = add(3, 5); // Function call with arguments 3 and 5

These punctuation symbols are fundamental in C++ syntax and play a crucial role in defining the structure and behavior of C++ programs.

- **Commas:**

Commas (,) are punctuation symbols used in C++ for various purposes. Here's a brief explanation of their uses:

Separating Items in Lists:- Commas are commonly used to separate items in lists, such as function arguments, variable declarations, and elements in initializer lists.

Example:

int a, b, c; // Declaring multiple variables

void foo(int x, int y) { // Function body}

int arr[] = {1, 2, 3, 4, 5}; // Initializing an array

In Function Calls:-

Commas are used to separate arguments in function calls.

Example:

int sum = add(3, 5); // Function call with two arguments

In Initialization:-

Commas are used to separate elements in initialization lists for arrays, structs, and classes.

Example:

```
int arr[] = {1, 2, 3, 4, 5}; // Initializing an array

struct Point {

   int x; int y;

};

Point p = {10, 20}; // Initializing a struct
```

In For Loops:-

Commas are used in the initialization, condition, and iteration expressions of for loops.

Example: for (int i = 0, j = 10; i < 5; ++i, --j) {// Loop body}

In Enumerations:-

Commas are used to separate enumerators in enumerations.

Example: enum Color { RED,GREEN,BLUE};

# LEXICAL ANALYZER

## Introduction:

The lexical analyzer, also known as a lexer or scanner, is the initial phase of a compiler or interpreter responsible for breaking the source code into meaningful tokens. Here's some information about the lexical analyzer:

1. **Tokenization:** The primary task of the lexical analyzer is to tokenize the input source code. It scans the characters of the source code sequentially and groups them into tokens based on predefined rules. Tokens represent the smallest meaningful units of the source code, such as keywords, identifiers, literals, operators, and punctuation symbols.

2. **Character Recognition:** The lexical analyzer recognizes individual characters and combines them into tokens based on their syntactic significance. It handles whitespace, comments, and other non-essential characters by ignoring or skipping them.

3. **Lexical Rules:** The lexical analyzer operates according to a set of lexical rules defined by the programming language. These rules specify how to recognize and categorize different types of tokens. For example, in C++, identifiers typically start with a letter or underscore, followed by letters, digits, or underscores.

4. **Token Types:** The lexical analyzer identifies various types of tokens, such as:

    - Keywords: Reserved words with special meanings in the language.
    - Identifiers: Names defined by the programmer for variables, functions, classes, etc.
    - Constants: Fixed values like integers, floating-point numbers, strings, and character literals.
    - Operators: Symbols representing mathematical or logical operations.
    - Punctuation: Symbols like braces, parentheses, commas, and semicolons.
    - Comments: Text ignored by the compiler, used for documentation or annotations.

5. **Output:** The lexical analyzer typically produces a stream of tokens, which serve as input for subsequent phases of the compiler or interpreter, such as the parser. This token stream contains the necessary information to analyze and process the source code further.

6. **Error Handling:** The lexical analyzer may detect lexical errors, such as unrecognized characters or invalid token sequences. It may report these errors and provide diagnostic messages to assist the programmer in identifying and correcting issues in the source code.

## Regular Expressions and Definitions:

```
delim [ \t]

ws {delim}+

letter [a-zA-Z]

digit [0-9]

id {letter}({letter}|{digit})*

relop (<=|>=|==|!=|<|>)

leftshift (<<)

rightshift (>>)

plus [+]

minus [-]

mult [*]

div [/]

num ({digit}+)

float ({num}\.{num})

arithmeticop ({plus}|{minus}|{mult}|{div})

increment {plus}{plus}

decrement {minus}{minus}

assignop =

string (\"(\\.|[^\\"])*\"|\'(\\.|[^\\'])*\')

keyword (if|else|const|while|for|int|float|return|void|main|char|"long
long"|double|short|long|unsigned|signed|define|struct|enum|typedef|sizeof|static|register|auto
|break|case|continue|default|do|goto|switch|cout|cin|endl|bool|using|namespace|std|include|
iostream|vector|map|set|queue|stack|push_back|pop_back|pop|push|top|front|priority_queue)

inval (({digit}+{id}))
```

## Code Snippet:

```
%{
   #include <stdio.h>
   extern int yylex();
   extern char *yytext;
   char *prevtoken;
   char *tokentobeadded;
   int idCount = 0;
   int lineno = 0;
%}
/* regular definitions */
delim [ \t]
ws {delim}+
letter [a-zA-Z]
digit [0-9]
id {letter}({letter}|{digit})*
relop (<=|>=|==|!=|<|>)
leftshift (<<)
rightshift (>>)
plus [+]
minus [-]
mult [*]
div [/]
num ({digit}+)
float ({num}\.{num})
arithmeticop ({plus}|{minus}|{mult}|{div})
increment {plus}{plus}
decrement {minus}{minus}
assignop =
string (\"(\\.|[^\\"])*\"|\'(\\.|[^\\'])*\')
```

```
keyword (if|else|const|while|for|int|float|return|void|main|char|"long
long"|double|short|long|unsigned|signed|define|struct|enum|typedef|sizeof|static|register|auto
|break|case|continue|default|do|goto|switch|cout|cin|endl|bool|using|namespace|std|include|
iostream|vector|map|set|queue|stack|push_back|pop_back|pop|push|top|front|priority_queue)

inval (({digit}+{id}))


%%

"bits/stdc++.h" {printf("HEADER: %s\n", yytext);}

"\n" {lineno++;printf("\nNEWLINE: %s", yytext);}

"//".* { /* ignore comments */ }

"/*".*"*/" { /* ignore comments */ }

{keyword} {

    printf("KEYWORD: %s   line: %d\n", yytext, lineno);

}

{id} {

    prevtoken = "id";

    printf("ID: %s, line: %d\n", yytext, lineno);

}

{ws} { /* ignore whitespace */ }

{leftshift} {

  printf("LeftShift: %s, line: %d\n", yytext, lineno);

}

{rightshift} {

  printf("RightShift: %s, line: %d\n", yytext, lineno);

}

{relop} {

  printf("RELOP: %s, line: %d\n", yytext, lineno);

}

{inval} {printf("Invalid token: %s\n", yytext);}

{increment} {printf("INCREMENT: %s\n", yytext);}

{decrement} {printf("DECREMENT: %s\n", yytext);}

{float} {printf("FLOAT: %s, line: %d\n", yytext, lineno);}
```

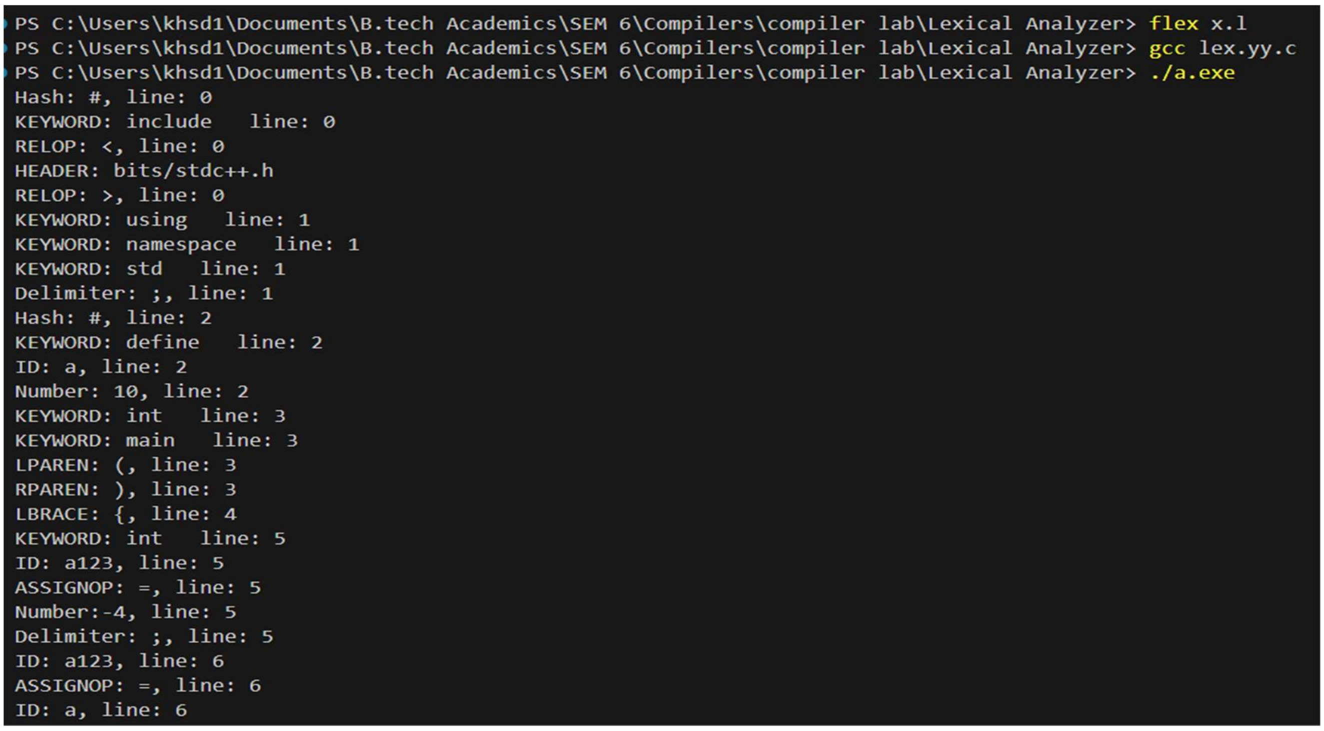
```
{num} {
  if(tokentobeadded && strcmp(tokentobeadded, "-") == 0) {
    printf("Number:-%s, line: %d\n", yytext, lineno);
    prevtoken = NULL;
    tokentobeadded = NULL;
  }else{
    printf("Number: %s, line: %d\n", yytext, lineno);
  }
}
{arithmeticop} {
  if(strcmp(prevtoken, "assign") == 0 && strcmp(yytext,"-") == 0) {
    tokentobeadded = "-";
  }else{
    printf("ARITHMETICOP: %s, line: %d\n", yytext, lineno);
  }
}
"." {printf("DOT: %s, line: %d\n", yytext, lineno);}
"#" {printf("Hash: %s, line: %d\n", yytext, lineno);}
"(" {printf("LPAREN: %s, line: %d\n", yytext, lineno);}
")" {printf("RPAREN: %s, line: %d\n", yytext, lineno);}
"[" {printf("LBRACKET: %s, line: %d\n", yytext, lineno);}
"]" {printf("RBRACKET: %s, line: %d\n", yytext, lineno);}
"{" {printf("LBRACE: %s, line: %d\n", yytext, lineno);}
"}" {printf("RBRACE: %s, line: %d\n", yytext, lineno);}
"=" {prevtoken = "assign";printf("ASSIGNOP: %s, line: %d\n", yytext, lineno);}
";" {printf("Delimiter: %s, line: %d\n", yytext, lineno);}
"," {printf("COMMA: %s, line: %d\n", yytext, lineno);}
{string} {printf("string: %s, line: %d\n", yytext, lineno);}
. {printf("Invalid token: %s, line: %d\n", yytext, lineno);}
%%
int main() {
  FILE *fp = fopen("test.cpp", "r");
  yyin = fp;
  yylex();
  return 0;
}
int yywrap(){ return 1; }
```

## Sample Program:

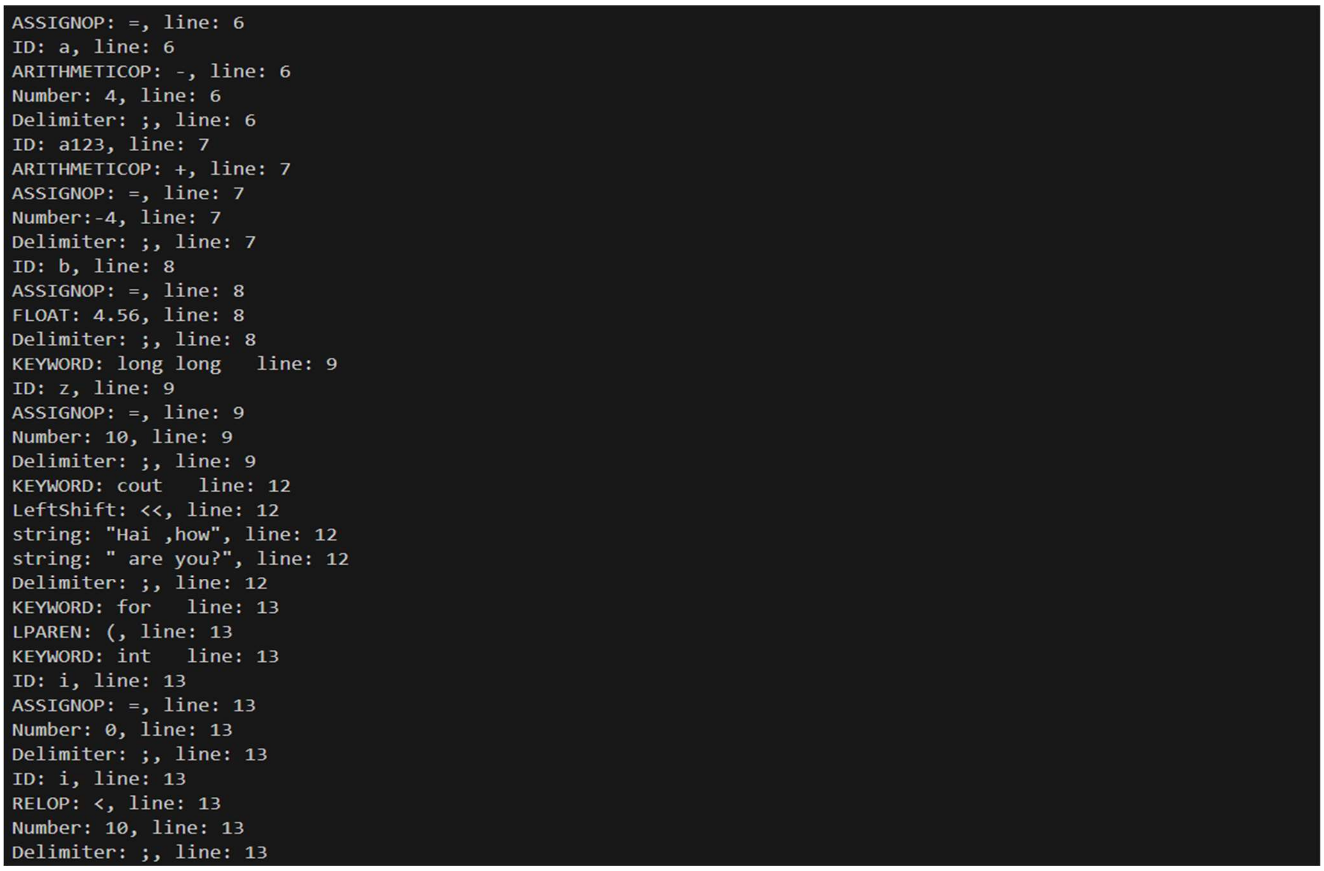
```
#include<bits/stdc++.h>
using namespace std;
#define a 10
int main()
{
    int a123 = -4;
    a123=a-4;
    a123+=-4;
    b = 4.56;
    long long z = 10;
    //cout<<"Hello World";
    /* cout<<"Jai Baalayya"*/
    cout<<"Hai ,how"" are you?";
    for(int i=0;i<10;i++)
    {
        cout<<i<<endl;
    }
    vector<int> v;
    v.push_back(1);
    v.pop_back();
    while(1)
    {
        cout<<"Hello World"<<'c';
```

```
  }
 Int 123ab = 5;
  return 0;
}
```

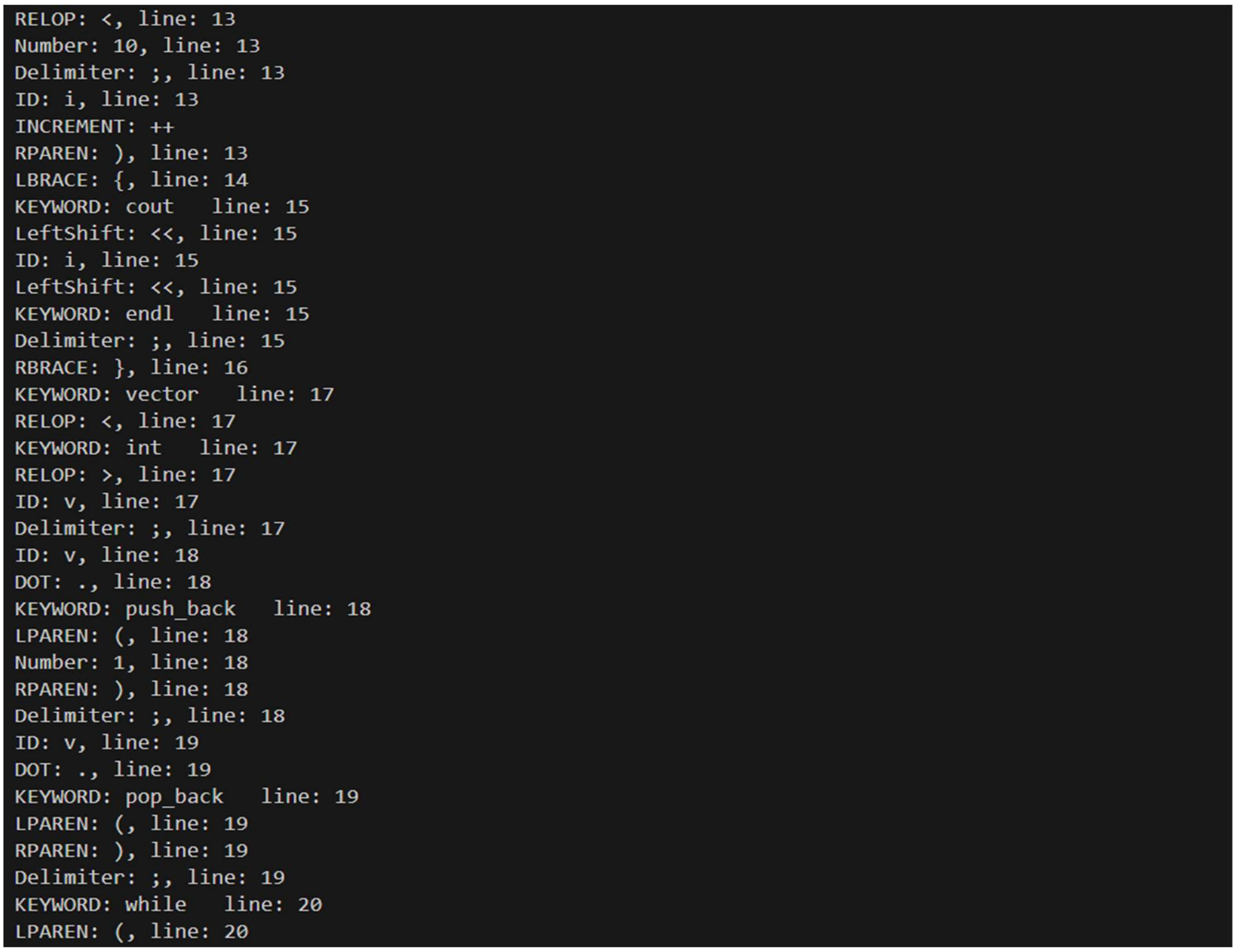
## Generated Tokens Output:



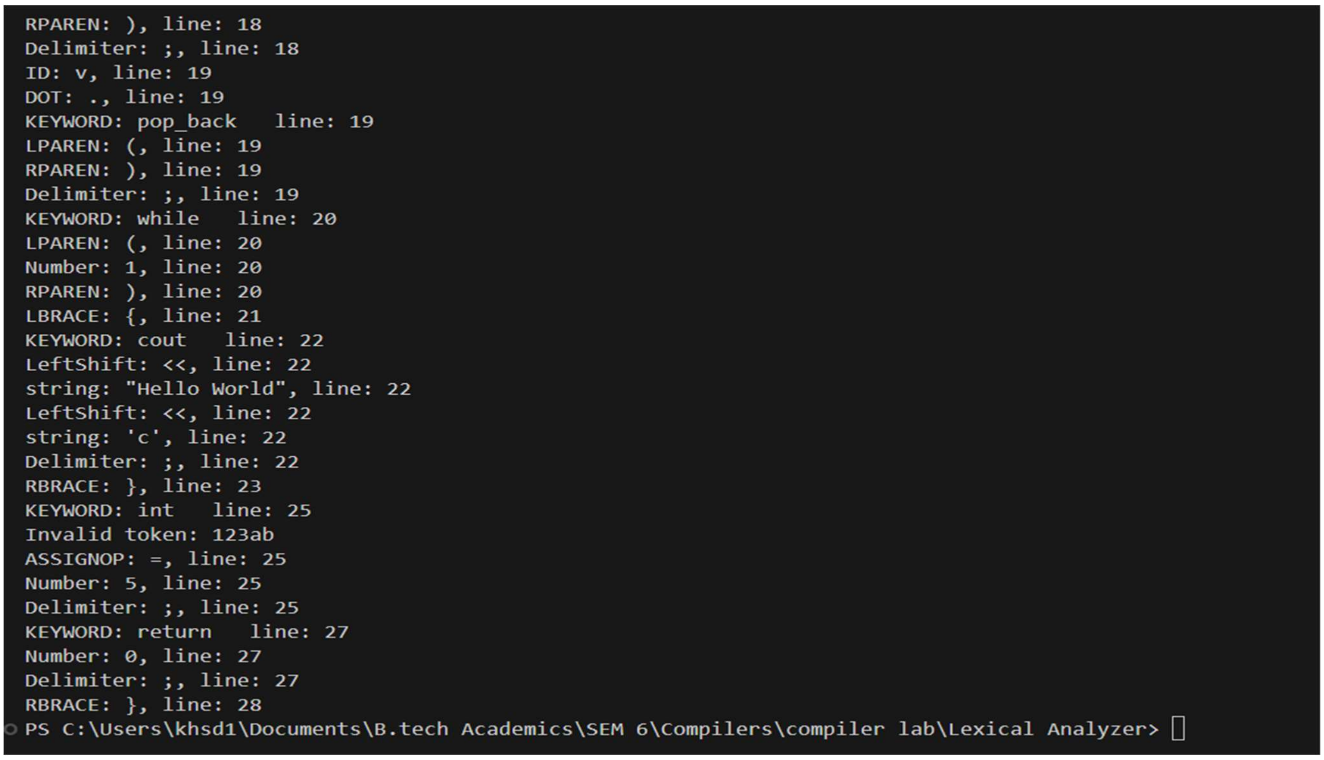
```
PS C:\Users\khsd1\Documents\B.tech Academics\SEM 6\Compilers\compiler lab\Lexical Analyzer> flex x.l
PS C:\Users\khsd1\Documents\B.tech Academics\SEM 6\Compilers\compiler lab\Lexical Analyzer> gcc lex.yy.c
PS C:\Users\khsd1\Documents\B.tech Academics\SEM 6\Compilers\compiler lab\Lexical Analyzer> ./a.exe
Hash: #, line: 0
KEYWORD: include   line: 0
RELOP: <, line: 0
HEADER: bits/stdc++.h
RELOP: >, line: 0
KEYWORD: using   line: 1
KEYWORD: namespace   line: 1
KEYWORD: std   line: 1
Delimiter: ;, line: 1
Hash: #, line: 2
KEYWORD: define   line: 2
ID: a, line: 2
Number: 10, line: 2
KEYWORD: int   line: 3
KEYWORD: main   line: 3
LPAREN: (, line: 3
RPAREN: ), line: 3
LBRACE: {, line: 4
KEYWORD: int   line: 5
ID: a123, line: 5
ASSIGNOP: =, line: 5
Number:-4, line: 5
Delimiter: ;, line: 5
ID: a123, line: 6
ASSIGNOP: =, line: 6
ID: a, line: 6
```



```
ASSIGNOP: =, line: 6
ID: a, line: 6
ARITHMETICOP: -, line: 6
Number: 4, line: 6
Delimiter: ;, line: 6
ID: a123, line: 7
ARITHMETICOP: +, line: 7
ASSIGNOP: =, line: 7
Number:-4, line: 7
Delimiter: ;, line: 7
ID: b, line: 8
ASSIGNOP: =, line: 8
FLOAT: 4.56, line: 8
Delimiter: ;, line: 8
KEYWORD: long long   line: 9
ID: z, line: 9
ASSIGNOP: =, line: 9
Number: 10, line: 9
Delimiter: ;, line: 9
KEYWORD: cout   line: 12
LeftShift: <<, line: 12
string: "Hai ,how", line: 12
string: " are you?", line: 12
Delimiter: ;, line: 12
KEYWORD: for   line: 13
LPAREN: (, line: 13
KEYWORD: int   line: 13
ID: i, line: 13
ASSIGNOP: =, line: 13
Number: 0, line: 13
Delimiter: ;, line: 13
ID: i, line: 13
RELOP: <, line: 13
Number: 10, line: 13
Delimiter: ;, line: 13
```



```
RELOP: <, line: 13
Number: 10, line: 13
Delimiter: ;, line: 13
ID: i, line: 13
INCREMENT: ++
RPAREN: ), line: 13
LBRACE: {, line: 14
KEYWORD: cout   line: 15
LeftShift: <<, line: 15
ID: i, line: 15
LeftShift: <<, line: 15
KEYWORD: endl   line: 15
Delimiter: ;, line: 15
RBRACE: }, line: 16
KEYWORD: vector   line: 17
RELOP: <, line: 17
KEYWORD: int   line: 17
RELOP: >, line: 17
ID: v, line: 17
Delimiter: ;, line: 17
ID: v, line: 18
DOT: ., line: 18
KEYWORD: push_back   line: 18
LPAREN: (, line: 18
Number: 1, line: 18
RPAREN: ), line: 18
Delimiter: ;, line: 18
ID: v, line: 19
DOT: ., line: 19
KEYWORD: pop_back   line: 19
LPAREN: (, line: 19
RPAREN: ), line: 19
Delimiter: ;, line: 19
KEYWORD: while   line: 20
LPAREN: (, line: 20
```



```
RPAREN: ), line: 18
Delimiter: ;, line: 18
ID: v, line: 19
DOT: ., line: 19
KEYWORD: pop_back   line: 19
LPAREN: (, line: 19
RPAREN: ), line: 19
Delimiter: ;, line: 19
KEYWORD: while   line: 20
LPAREN: (, line: 20
Number: 1, line: 20
RPAREN: ), line: 20
LBRACE: {, line: 21
KEYWORD: cout   line: 22
LeftShift: <<, line: 22
string: "Hello World", line: 22
LeftShift: <<, line: 22
string: 'c', line: 22
Delimiter: ;, line: 22
RBRACE: }, line: 23
KEYWORD: int   line: 25
Invalid token: 123ab
ASSIGNOP: =, line: 25
Number: 5, line: 25
Delimiter: ;, line: 25
KEYWORD: return   line: 27
Number: 0, line: 27
Delimiter: ;, line: 27
RBRACE: }, line: 28
PS C:\Users\khsd1\Documents\B.tech Academics\SEM 6\Compilers\compiler lab\Lexical Analyzer>
```