**Compiler Construction Lab**

***CSL 323***

***FINAL TERM***

**Final Paper**



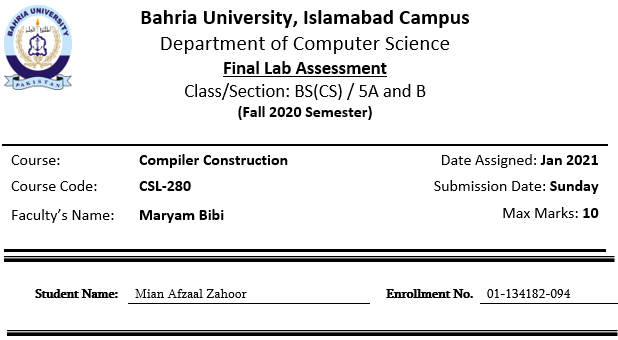
Mian Afzaal Zahoor

01-134182-094

BS (CS) – 5A

**Department of Computer Sciences**

**BAHRIA UNIVERSITY, ISLAMABAD**



**Journal 5**

**Lexical Analyzer**

**Introduction**

The compiler design is divided into six stages. Each stage is assigned with the task and is interlinked with each other, like the current phase output will be used as the input in the next phase. Following are the phases of compiler design.

* Lexical analyzer

Lexical Analyzer is a process of converting a sequence of characters into a sequence of tokens.

* Parser (syntax analyzer)

Parser is the process of analyzing a string of symbol confirming to the rules of a formal grammar.

* Semantic analyzer

Semantic Analyzer is a process in compiler construction after parsing to gather information from the source code.

* Intermediate code generator

It receives input from the semantic analyzer in the form of annotated syntax tree.

* Code optimization

It is the process of modifying a system software to make some aspects of its work more efficient or use of less resources.

* Target code generator

It finally generates the final code after all going through the all processes.

The very first step in creating a compiler is Lexical Analysis. A lexical analyzer takes the source code written in regular coding syntax and changes it into the form of a token sequence. It would erase any additional space or comments written in the source code.

A lexical analyzer includes a scanner or tokenizer. If the lexical analyzer detects the token to be invalid, an error is produced. It reads character streams from the source code, checks for legal tokens, and, when necessary, passes the information to the syntax analyzer. A lexical analyzer is also know as Lexer or Tokenizer Some of the relevant terms used in Lexical analyzer

* **Lexeme:**

A lexeme is a set of characters used according to a token's matching pattern in the source software. It is nothing but a token case.

* **Token:**

The token is a sequence of characters that represents a unit of information in the **source program.** For example Numbers, Identifiers, Operators, Brackets, Keywords, etc.

* **Pattern:**

A pattern is a description that the token uses. In the case of a keyword used as a token, the pattern is a character set.

**Working:**

The Lexical Analyzer checks the program's entire source code. One by one, it recognizes each token. Scanners are typically only introduced as needed by a parser to generate tokens. So firstly, it will get the next token by providing a 'Get next token' command to the lexical analyzer from the parser. Then the lexical analyzer searches the input until it detects the next token upon obtaining this instruction. And, then it returns the Parser token.

When generating these tokens, the Lexical Analyzer skips white spaces and notes. If there is an error, the Lexical Analyzer compares the error to the source file and line number. Lexical Analyzer work in such a way as mentioned in the below steps.

* Tokenization.
* Give error messages.
* Eliminate comments and white spaces.

**Video Link**

[CC LAB FINAL VIVA RECORDED](https://drive.google.com/file/d/1Yommw1gWAZOvz4J8DGrhAkRP4QXHHWuW/view?usp=sharing)

**Code**

#include<iostream>

#include<fstream>

#include<string>

#include<conio.h>

using namespace std;

int index = 0;

string \*ptr = new string[100];

ifstream file;

typedef enum

{

PLUS, MINUS, MULTIPLY, DIVIDE, EQUAL, LSB, RSB, LCB, RCB, LRB, RRB, SPACE, SEMICOLON, COLON, COMMA, LESSTH, GRTH, INC, DEC, NOT, EQUALEQUAL, PLUSEQUAL, ALPHA,

DIGIT, MINUSEQUAL, DIVEQUAL, MULEQUAL, NOTEQUAL, GREATEREQUAL, LESSEQUAL, OUT, IN, IF, ELSE, THEN, ELSEIF, DO, WHILE, FOR

}

TokenType;

TokenType TokenRec(char ch)

{

string num = "";

if (ch == '/')

{

file.get(ch);

if (ch == '/') //FOR COMMENTS

{

while (ch != '\n')

{

file.get(ch);

}

}

else if (ch == '=') //FOR DIVIDE EQUAL TO

{

cout << "/= ";

\*(ptr + index) = "/=";

return DIVEQUAL;

}

else //FOR DIVIDE

{

file.putback(ch);

}

cout << "/ ";

\*(ptr + index) = "/";

return DIVIDE;

}

else if (ch == '(') //FOR LEFT ROUND BRACKET

{

cout << "( ";

\*(ptr + index) = "(";

return LRB;

}

else if (ch == ')') //FOR RIGHT ROUND BRACKET

{

cout << ") ";

\*(ptr + index) = ")";

return RRB;

}

else if (ch == '{') //FOR LEFT CURLEY BRACKET

{

cout << "{ ";

\*(ptr + index) = "{";

return LCB;

}

else if (ch == '}') //FOR RIGHT CURLEY BRACKET

{

cout << "} ";

\*(ptr + index) = "}";

return RCB;

}

else if (ch == '[') //FOR LEFT SQUARE BRACKET

{

cout << "[ ";

\*(ptr + index) = "[";

return LSB;

}

else if (ch == ']') //FOR RIGHT SQUARE BRACKET

{

cout << "] ";

\*(ptr + index) = "]";

return RSB;

}

else if (ch == ',') //FOR COMMA

{

cout << ", ";

\*(ptr + index) = ",";

return COMMA;

}

else if (ch == ';') //FOR SEMICOLON

{

cout << "; ";

\*(ptr + index) = ";";

return SEMICOLON;

}

else if (ch == ':') //FOR COLON

{

cout << ": ";

\*(ptr + index) = ":";

return COLON;

}

else if (ch == '>')

{

file.get(ch);

if (ch == '>') //FOR CONSOLE INPUT

{

cout << ">> ";

\*(ptr + index) = ">>";

return IN;

}

else if (ch == '=') //FOR GREATER THEN EQUAL TO

{

cout << ">= ";

\*(ptr + index) = ">=";

return GREATEREQUAL;

}

else //FOR GREATER THEN

{

file.putback(ch);

}

cout << "> ";

\*(ptr + index) = ">";

return GRTH;

}

else if (ch == '<')

{

file.get(ch);

if (ch == '<') //FOR CONSOLE OUTPUT

{

cout << "<< ";

\*(ptr + index) = "<<";

return OUT;

}

if (ch == '=') //FOR LESSER THEN EQUAL TO

{

cout << "<= ";

\*(ptr + index) = "<=";

return LESSEQUAL;

}

else //FOR LESSER THEN

{

file.putback(ch);

}

cout << "< ";

\*(ptr + index) = "<";

return LESSTH;

}

else if (ch == '!')

{

file.get(ch);

if (ch == '=') //FOR NOT EQUAL TO

{

cout << "!= ";

\*(ptr + index) = "!=";

return NOTEQUAL;

}

else //FOR NOT

{

file.putback(ch);

}

cout << "! ";

\*(ptr + index) = "!";

return NOT;

}

else if (ch == '\*')

{

file.get(ch);

if (ch == '=') //FOR MULTIPLY EQUAL TO

{

cout << "\*= ";

\*(ptr + index) = "=";

return MULEQUAL;

}

else //FOR MULTIPLY

{

file.putback(ch);

}

cout << "\* ";

\*(ptr + index) = "\*";

return MULTIPLY;

}

else if (ch == '=')

{

file.get(ch);

if (ch == '=') //FOR COMPARISON EQUAL-EQUAL

{

cout << "== ";

\*(ptr + index) = "==";

return EQUALEQUAL;

}

else //FOR ASSIGNING (EQUAL)

{

file.putback(ch);

}

cout << "= ";

\*(ptr + index) = "=";

return EQUAL;

}

else if (ch == '+')

{

file.get(ch);

if (ch == '+') //FOR INCREMENT

{

cout << "++ ";

\*(ptr + index) = "++";

return INC;

}

else if (ch == '=') //FOR PLUS EQUAL TO

{

cout << "+= ";

\*(ptr + index) = "+=";

return PLUSEQUAL;

}

else //FOR PLUS

{

file.putback(ch);

}

cout << "+ ";

\*(ptr + index) = "+";

return PLUS;

}

else if (ch == '-')

{

file.get(ch);

if (ch == '-') //FOR DECREMENT

{

cout << "-- ";

\*(ptr + index) = "--";

return DEC;

}

else if (ch == '=') //FOR MINUS EQUAL TO

{

cout << "-= ";

\*(ptr + index) = "-=";

return MINUSEQUAL;

}

else //FOR MINUS

{

file.putback(ch);

}

cout << "- ";

\*(ptr + index) = "-";

return MINUS;

}

else if (ch == '0' || ch == '1' || ch == '2' || ch == '3' || ch == '4' || ch == '5' || ch == '6' || ch == '7' || ch == '8' || ch == '9')

{

num += ch;

file.get(ch);

if (ch == '0' || ch == '1' || ch == '2' || ch == '3' || ch == '4' || ch == '5' || ch == '6' || ch == '7' || ch == '8' || ch == '9') //FOR NUMBERS

{

num += ch;

while (ch != '0' || ch != '1' || ch != '2' || ch != '3' || ch != '4' || ch != '5' || ch != '6' || ch != '7' || ch != '8' || ch != '9') //UNTIL NUMBERS ARE DONE

{

file.get(ch);

num += ch;

}

}

else

{

file.putback(ch);

}

cout << "Digit ";

\*(ptr + index) = num;

return DIGIT;

}

else if (isalpha(ch)) //FOR STATEMENTS

{

string str;

do

{

str += ch;

file.get(ch);

}

while (isalpha(ch) || isdigit(ch));

file.putback(ch);

if (str == "if") //FOR IF STATEMENT

{

cout << "IF ";

\*(ptr + index) = str;

return IF;

}

else if (str == "else") //FOR ELSE-IF STATEMENT

{

file.get(ch);

if (str == " ")

{

if (str == "if")

{

cout << "ELSE IF ";

\*(ptr + index) = str;

return ELSEIF;

}

else

{

file.putback(ch);

}

}

file.putback(ch);

cout << "ELSE ";

\*(ptr + index) = str;

return ELSE;

}

else if (str == "then") //FOR THEN STATEMENT

{

cout << "THEN ";

\*(ptr + index) = str;

return THEN;

}

else if (str == "do") //FOR DO STATEMENT

{

cout << "DO ";

\*(ptr + index) = str;

return DO;

}

else if (str == "while") //FOR WHILE STATEMENT

{

cout << "WHILE ";

\*(ptr + index) = str;

return WHILE;

}

else if (str == "for") //FOR FOR STATEMENT

{

cout << "FOR ";

\*(ptr + index) = str;

return FOR;

}

else //FOR VARIABLES

{

cout << "Variable ";

\*(ptr + index) = str;

return ALPHA;

}

}

return SPACE; //FOR SPACE

}

int main()

{

TokenType T;

char ch;

file.open("Text1.txt");

if (!file)

{

cout << "File Error";

exit(1);

}

else

{

while (!file.eof())

{

while (file.get(ch))

{

T = TokenRec(ch);

if (T != 11)

{

cout << T << endl;

}

index++;

}

}

file.close();

}

\_getch();

return 0;

}

**OUTPUT**

