A black background with grey leaves

AI-generated content may be incorrect.

Lexical Analyzer

Build Scanner

A grey logo on a black background

AI-generated content may be incorrect.

**Prepared By**

Student Name :

**Mustafa hany gomaa**

Student ID:

**200045179**

**Under Supervision**

Name of Doctor:

**Nehal Abdelsalam**

Name of T. A.:

**Fares**

**Introduction**

**The Lexical Analyzer is the first step in a compiler. It takes the source code and splits it into small units called tokens, which are then used by later compiler phases. In this project, we built a simple program to analyze arithmetic expressions.**

* 1. **Phases of Compiler**

**A compiler has multiple phases:**

1. **Lexical Analysis: Converts code into tokens (our focus).**
2. **Syntax Analysis: Checks if the tokens follow language rules.**
3. **Semantic Analysis: Ensures the code makes sense.**
4. **Code Generation: Produces machine code. This project focuses on the Lexical Analysis phase.**
5. **Lexical Analyzer**

**Our lexical analyzer reads from a file named front.in and processes arithmetic expressions. For example, with the input Mustafa = ( 2005 + 20 ) \* 10, it identifies identifiers (like Mustafa), numbers (like 2005), operators (like +, \*), and parentheses, printing each token with its code and lexeme.**

1. **Software Tools**

**We used simple tools to develop and test this program.**

* 1. **Computer Program**

**We used GCC to compile the C code into an executable (front.exe) and visual studio 2022 to run and test the program.**

* 1. **Programming Language**

**We chose C because it’s fast, simple, and provides good control over file reading and character processing.**

1. **Implementation of a Lexical Analyzer**

**The program opens front.in and reads it character by character. It classifies each part as an identifier, integer, operator, or parenthesis, and assigns token codes. For the input Mustafa = ( 2005 + 20 ) \* 10, it produces the output shown in Table 1.**

**Simple Explanation of the Code**

**c**

**#define \_CRT\_SECURE\_NO\_WARNINGS**

**- Stops warnings about some functions.**

**---------------------------**

**#include <stdio.h>**

**..**

**- Adds input/output functions like `printf`.**

**---------------------------**

**#include <ctype.h>**

**..**

**- Adds functions to check if a character is a letter or digit.**

**---------------------------**

**int charClass;**

**..**

**- Stores the type of the current character.**

**---------------------------**

**char lexeme[100];**

**..**

**- Holds the current token string.**

**---------------------------**

**char nextChar;**

**..**

**- The next character from the file.**

**---------------------------**

**int lexLen;**

**..**

**- Length of the token string.**

**---------------------------**

**int token;**

**..**

**- Current token code.**

**---------------------------**

**int nextToken;**

**..**

**- Next token code to process.**

**---------------------------**

**FILE\* in\_fp;**

**..**

**- Pointer to the input file.**

**---------------------------**

**void addChar();**

**void getChar();**

**void getNonBlank();**

**int lex();**

**..**

**- Declares functions to use later.**

**---------------------------**

**#define LETTER 0**

**#define DIGIT 1**

**#define UNKNOWN 99**

**..**

**- Sets codes for letters, digits, and other characters.**

**---------------------------**

**#define INT\_LIT 10**

**#define IDENT 11**

**#define ASSIGN\_OP 20**

**#define ADD\_OP 21**

**#define SUB\_OP 22**

**#define MULT\_OP 23**

**#define DIV\_OP 24**

**#define LEFT\_PAREN 25**

**#define RIGHT\_PAREN 26**

**..**

**- Sets codes for tokens like numbers, variables, and operators.**

**---------------------------**

**int main() {**

**..**

**- Starts the program.**

**---------------------------**

**if ((in\_fp = fopen("front.in", "r")) == NULL)**

**..**

**- Opens the file `front.in` to read.**

**---------------------------**

**printf("ERROR - cannot open front.in \n");**

**..**

**- Shows an error if the file doesn’t open.**

**---------------------------**

**else {**

**..**

**- If the file opens, continue.**

**---------------------------**

**getChar();**

**..**

**- Reads the first character.**

**---------------------------**

**do {**

**..**

**- Starts a loop.**

**---------------------------**

**lex();**

**..**

**- Analyzes the next token.**

**---------------------------**

**} while (nextToken != EOF);**

**..**

**- Loops until the file ends.**

**---------------------------**

**}**

**return 0;**

**..**

**- Ends the program.**

**---------------------------**

**int lookup(char ch) {**

**..**

**- Checks if a character is an operator or parenthesis.**

**---------------------------**

**switch (ch) {**

**..**

**- Looks at the character.**

**---------------------------**

**case '(':**

**addChar();**

**nextToken = LEFT\_PAREN;**

**break;**

**..**

**- Sets `(` as token 25.**

**---------------------------**

**case ')':**

**addChar();**

**nextToken = RIGHT\_PAREN;**

**break;**

**..**

**- Sets `)` as token 26.**

**---------------------------**

**case '+':**

**addChar();**

**nextToken = ADD\_OP;**

**break;**

**..**

**- Sets `+` as token 21.**

**---------------------------**

**case '-':**

**addChar();**

**nextToken = SUB\_OP;**

**break;**

**..**

**- Sets `-` as token 22.**

**---------------------------**

**case '\*':**

**addChar();**

**nextToken = MULT\_OP;**

**break;**

**..**

**- Sets `\*` as token 23.**

**---------------------------**

**case '/':**

**addChar();**

**nextToken = DIV\_OP;**

**break;**

**..**

**- Sets `/` as token 24.**

**---------------------------**

**case '=':**

**addChar();**

**nextToken = ASSIGN\_OP;**

**break;**

**..**

**- Sets `=` as token 20.**

**---------------------------**

**default:**

**addChar();**

**nextToken = EOF;**

**break;**

**..**

**- Sets anything else as end of file.**

**---------------------------**

**}**

**return nextToken;**

**..**

**- Returns the token code.**

**---------------------------**

**void addChar() {**

**..**

**- Adds a character to the token string.**

**---------------------------**

**if (lexLen <= 98) {**

**..**

**- Checks if the string has space.**

**---------------------------**

**lexeme[lexLen++] = nextChar;**

**..**

**- Adds the character.**

**---------------------------**

**lexeme[lexLen] = 0;**

**..**

**- Ends the string with 0.**

**---------------------------**

**}**

**else**

**printf("Error - lexeme is too long \n");**

**..**

**- Shows an error if the string is too long.**

**---------------------------**

**void getChar() {**

**..**

**- Reads the next character.**

**---------------------------**

**if ((nextChar = getc(in\_fp)) != EOF) {**

**..**

**- Gets a character from the file.**

**---------------------------**

**if (isalpha(nextChar))**

**charClass = LETTER;**

**..**

**- Sets letters as type 0.**

**---------------------------**

**else if (isdigit(nextChar))**

**charClass = DIGIT;**

**..**

**- Sets digits as type 1.**

**---------------------------**

**else**

**charClass = UNKNOWN;**

**..**

**- Sets other characters as type 99.**

**---------------------------**

**}**

**else**

**charClass = EOF;**

**..**

**- Sets end of file as `EOF`.**

**---------------------------**

**void getNonBlank() {**

**..**

**- Skips spaces.**

**---------------------------**

**while (isspace(nextChar))**

**getChar();**

**..**

**- Reads until a non-space character.**

**---------------------------**

**int lex() {**

**..**

**- Analyzes tokens.**

**---------------------------**

**lexLen = 0;**

**..**

**- Resets the token string length.**

**---------------------------**

**getNonBlank();**

**..**

**- Skips spaces.**

**---------------------------**

**switch (charClass) {**

**..**

**- Checks the character type.**

**---------------------------**

**case LETTER:**

**addChar();**

**getChar();**

**..**

**- Adds a letter and reads the next.**

**---------------------------**

**while (charClass == LETTER || charClass == DIGIT) {**

**addChar();**

**getChar();**

**..**

**- Adds more letters or digits.**

**---------------------------**

**}**

**nextToken = IDENT;**

**break;**

**..**

**- Sets the token as a variable (11).**

**---------------------------**

**case DIGIT:**

**addChar();**

**getChar();**

**..**

**- Adds a digit and reads the next.**

**---------------------------**

**while (charClass == DIGIT) {**

**addChar();**

**getChar();**

**..**

**- Adds more digits.**

**---------------------------**

**}**

**nextToken = INT\_LIT;**

**break;**

**..**

**- Sets the token as a number (10).**

**---------------------------**

**case UNKNOWN:**

**lookup(nextChar);**

**getChar();**

**break;**

**..**

**- Checks operators or parentheses.**

**---------------------------**

**case EOF:**

**nextToken = EOF;**

**..**

**- Sets the token as end of file.**

**---------------------------**

**lexeme[0] = 'E';**

**lexeme[1] = 'O';**

**lexeme[2] = 'F';**

**lexeme[3] = 0;**

**..**

**- Writes "EOF" in the token string.**

**---------------------------**

**break;**

**}**

**printf("Next token is: %d, Next lexeme is %s\n", nextToken, lexeme);**

**..**

**- Prints the token code and string.**

**---------------------------**

**return nextToken;**

**..**

**- Returns the token code.**

**Lexeme -------------------------- token**

**Mustafa Identifier**

**=** **Assignment.** **Operator**

**( Left Parenthesis**

**2005 Integer Literal**

**+ plus. Operator**

**20 Integer Literal**

**) Right Parenthesis**

**\* mult.** **Operator**

**10 Integer Literal**

1. **References**

* **"Compilers: Principles, Techniques, and Tools" by Aho, Sethi, and Ullman.**
* **Online C tutorials from W3Schools.**

**Important Note: -**

Technical reports include a mixture of text, tables, and figures. Consider how you can present the information best for your reader. Would a table or figure help to convey your ideas more effectively than a paragraph describing the same data?

Figures and tables should: -

* Be numbered
* Be referred to in-text, e.g. *In Table 1*…, and
* Include a simple descriptive label - above a table and below a figure.