Prepared By

Omar Abdelaziz
200038794

Under Supervision

DR :Nehal abselsalam
ENG :Isalm

```
#include <iostream>      // مكتبة للإدخال والإخراج (cout, cin)
#include <cctype>        // مكتبة فيها دوال للتعامل مع الحروف
                         (زي isalpha, isdigit)
#include <string>        // مكتبة للتعامل مع النصوص
                         (std::string)

using namespace std;     // عشان نقدر نستخدم cout و string
                         من غير ما نكتب std::

/* المتغيرات العامة */
int charClass;           // بتحدد نوع الحرف الحالي (حرف، رقم،
                         غير معروف)
string lexeme;           // الكلمة اللي البرنامج بيجمعها دلوقتي (الـ
                         token)
char nextChar;           // الحرف الحالي اللي بنحلله
int token;               // الكود الخاص بالرمز الحالي (مش مستخدم
                         هنا بس ممكن في تطويرات)
int nextToken;           // نوع التوكن اللي تم تحليله
string input;            // السطر اللي هندخله ونعالجه
int inputIndex = 0;      // مؤشر للمكان الحالي في string input

/* تعريف أنواع الحروف */
#define LETTER 0         // تعريف للحرف الأبجدي
```

```
#define DIGIT 1          // تعريف للرقم
#define UNKNOWN 99       // تعريف للحرف الغير معروف

/* تعريف التوكنز */
#define INT_LIT 10       // رقم صحيح
#define IDENT 11         // اسم متغير
#define ASSIGN_OP 20     // علامة يساوي =
#define ADD_OP 21        // علامة جمع +
#define SUB_OP 22        // علامة طرح -
#define MULT_OP 23       // علامة ضرب *
#define DIV_OP 24        // علامة قسمة /
#define LEFT_PAREN 25    // قوس مفتوح (
#define RIGHT_PAREN 26   // قوس مغلق )
#define EOF_TOKEN -1     // نهاية الملف أو نهاية السطر
#define SEMICOLON 27     // فاصلة منقوطة ;

/* تعريف الدوال */
void addChar();          // دالة بتضيف الحرف الحالي لـ lexeme
void getChar();          // دالة بتجيب الحرف الجاي من input
void getNonBlank();      // دالة بتتجاهل الفراغات
int lex();               // دالة التحليل الرئيسي (lexical analyzer)
int lookup(char ch);     // دالة بتحدد نوع الرموز زي + و * و =

void addChar() {
  if (lexeme.length() <= 98) {    // لو lexeme لسه صغير
    lexeme += nextChar;           // ضيف الحرف الحالي ليه
  } else {
    cout << "Error - lexeme is too long" << endl;  // لو الكلمة طويلة جداً
  }
```

```
{

} ()void getChar
    } if (inputIndex < input.length())      // لو لسه في حروف في النص
        ;nextChar = input[inputIndex++]     // خذ الحرف التالي وزود المؤشر
        if (isalpha(nextChar))              // لو الحرف حرف
            ;charClass = LETTER
        else if (isdigit(nextChar))         // لو رقم
            ;charClass = DIGIT
        else
            ;charClass = UNKNOWN            // أي حاجة تانية
    } else {
        ;charClass = EOF_TOKEN              // لو وصلنا لنهاية النص
        ;'nextChar = '\0                    // نهاية النص
    {
{

} ()void getNonBlank
    while (isspace(nextChar))               // طالما الحرف فراغ
        ;()getChar                          // اتخطاه وخذ اللي بعده
{

} int lookup(char ch)
    } switch (ch)
case '(': addChar(); nextToken = LEFT_PAREN;
    ;break                                  // قوس مفتوح
```

```
case ')': addChar(); nextToken = RIGHT_PAREN;
;break    // قوس مغلق
;case '+': addChar(); nextToken = ADD_OP; break
+ //
;case '-': addChar(); nextToken = SUB_OP; break
- //
case '*': addChar(); nextToken = MULT_OP;
;break       * //
;case '/': addChar(); nextToken = DIV_OP; break
/ //
case ';': addChar(); nextToken = SEMICOLON;
;break      ; //
case '=': addChar(); nextToken = ASSIGN_OP;
;break      = //
default:  addChar(); nextToken = EOF_TOKEN;
;break      // أي رمز تاني غير معروف
{
;return nextToken  // رجع نوع التوكن
{

} ()int lex
;"" = lexeme            // نبدأ بكلمة فاضية
;()getNonBlank          // نتخطى أي فراغات

} switch (charClass)
:case LETTER            // لو الحرف اللي بنبدأ بيه
حرف
;()addChar              // ضيفه لـ lexeme
;()getChar              // خذ اللي بعده
```

```
while (charClass == LETTER || charClass ==
                                        } DIGIT)
addChar();                  // طالما الحروف أو أرقام،
ضيفهم
getChar();                  // وكمل
                                        {
nextToken = IDENT;              // عرفنا إن دي متغير
                                        break;

case DIGIT:                 // لو بدأنا برقم
                                        addChar();
                                        getChar();
while (charClass == DIGIT) { // استمر طالما في
أرقام
                                        addChar();
                                        getChar();
                                        {
nextToken = INT_LIT;            // رقم صحيح
                                        break;

case UNKNOWN:                   // رموز زي +، =،
*، ...
lookup(nextChar);               // نحدد نوعها
getChar();                  // نجيب الحرف اللي بعده
                                        break;

case EOF_TOKEN:                 // لو خلصنا كل حاجة
                                nextToken = EOF_TOKEN;
lexeme = "EOF";             // نكتب نهاية الملف
                                        break;
```

```
{

// نطبع نوع التوكن والكلمة اللي لقيناها
cout << "Next token is: " << nextToken << ", Next
;lexeme is " << lexeme << endl
;return nextToken // نرجع نوع التوكن
{

} ()int main
;";input = "sum = 9 + 2 * 5 // السطر اللي عايزين نحلله
;inputIndex = 0 // نبدأ من أول السطر

;()getChar // نقرأ أول حرف
} do
;()lex // نحلل التوكن اللي بعده
;(while (nextToken != EOF_TOKEN { // لحد ما نوصل للنهاية

;return 0 // نهاية البرنامج
{
```

1. Introduction

This program is a basic implementation of a Lexical Analyzer (or scanner), which is the first phase of a compiler. It reads the input string and breaks it into tokens such as identifiers, integers, and operators. This allows the next phase of the compiler to understand the structure of the program.

1.1. Phases of Compiler

The phases of a compiler include:

- Lexical Analysis: Breaks input into tokens.

- Syntax Analysis: Checks the structure using grammar rules.
- Semantic Analysis: Validates meaning (e.g., type checking).
- Intermediate Code Generation: Converts into intermediate representation.
- Code Optimization: Improves performance.
- Code Generation: Produces final machine code.
- Symbol Table Management: Tracks variable and function info.
- Error Handling: Reports and manages errors throughout compilation.

2. Lexical Analyzer

The lexical analyzer reads the source input (in this case, a string like 'sum = 9 + 2 * 5;') and splits it into tokens. Each token is categorized, such as:

- IDENT: Identifier like 'sum'
- INT_LIT: Integer literal like 9 or 5
- ADD_OP, MULT_OP: Operators like '+' or '*'

Whitespace is ignored, and characters are processed one at a time using functions like `getChar()` and `lookup()`.

3. Software Tools

For lexical analysis, developers can use tools like:

- Lex / Flex: For token generation.
- Custom C++/Java code: As shown in this example.
- IDEs and debuggers for testing code logic.

3.1. Computer Program

A computer program is a sequence of instructions written to perform a specific task. In this case, the

program reads an expression and categorizes parts of it into tokens. It's written in C++ and simulates lexical analysis behavior.

3.2. Programming Language

C++ is used to write this lexical analyzer. It's a high-level programming language that supports procedural and object-oriented paradigms. C++ allows for control over memory and efficient character processing, which is useful in compiler development.

4. Implementation of a Lexical Analyzer

```
<include <iostream#
<include <fstream#
<include <cctype#
<include <cstdlib#
<include <string#

;using namespace std

/* Global declarations */
/* Variables */
;int charClass
;string lexeme
;char nextChar
;int token
;int nextToken
;ifstream in_fp

/* Function declarations */
;()void addChar
;()void getChar
```

```
;()void getNonBlank
;()int lex

/* Character classes */
define LETTER 0#
define DIGIT 1#
define UNKNOWN 99#

/* Token codes */
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#
define EOF_TOKEN -1#
define SEMICOLON 27#

/* main driver */
} ()int main
/* Open the input data file and process its contents */
    Since we're in an online compiler, let's create a  //
                                    temporary input string
                                instead of reading from a file //
                            ;";string input = "sum = 9 + 2 * 5

                    Create a temporary file for input simulation //
```

```
                                    ;ofstream temp_file("front.in")
                                        ;temp_file << input
                                         ;()temp_file.close

                                      ;in_fp.open("front.in")

                                    } if (!in_fp.is_open())
;cout << "ERROR - cannot open front.in" << endl
                                          ;return 1
                                                {

                                         ;()getChar
                                              } do
                                           ;()lex
                        ;(while (nextToken != EOF_TOKEN {

                                       ;()in_fp.close
                                          ;return 0
                                                {

        lookup - a function to lookup operators and  */
                        /* parentheses and return the token
                                  } int lookup(char ch)
                                       } switch (ch)
                                           :')' case
                                     ;()addChar
                          ;nextToken = LEFT_PAREN
                                        ;break
                                         :'(' case
                                     ;()addChar
```

```
;nextToken = RIGHT_PAREN
                ;break
                 :'+' case
               ;()addChar
          ;nextToken = ADD_OP
                ;break
                 :'-' case
               ;()addChar
          ;nextToken = SUB_OP
                ;break
                 :'*' case
               ;()addChar
         ;nextToken = MULT_OP
                ;break
                 :'/' case
               ;()addChar
          ;nextToken = DIV_OP
                ;break
                 :';' case
               ;()addChar
     ;nextToken = SEMICOLON
                ;break
                 :'=' case
               ;()addChar
      ;nextToken = ASSIGN_OP
                ;break
                  :default
               ;()addChar
     ;nextToken = EOF_TOKEN
                ;break
```

```
                                                                {
                                                    ;return nextToken
                                                                {

    /* addChar - a function to add nextChar to lexeme */
                                                    } ()void addChar
                                            } if (lexeme.length() <= 98)
                                                ;lexeme += nextChar
                                                            } else {
        ;cout << "Error - lexeme is too long" << endl
                                                                {
                                                                  {

getChar - a function to get the next character of input  */
                                        /* and determine its character class
                                                    } ()void getChar
    if (in_fp.get(nextChar)) {  // Using ifstream's get()
                                                                  method
                                                if (isalpha(nextChar))
                                            ;charClass = LETTER
                                            else if (isdigit(nextChar))
                                                ;charClass = DIGIT
                                                                  else
                                            ;charClass = UNKNOWN
                                                            } else {
                                                ;charClass = EOF_TOKEN
     nextChar = EOF; // Set nextChar to EOF when
                                                    end of file is reached
                                                                {
                                                                  {
```

```
getNonBlank - a function to call getChar until it  */
                    /* returns a non-whitespace character
                                        } ()void getNonBlank
                                  while (isspace(nextChar))
                                              ;()getChar
                                                        {

lex - a simple lexical analyzer for arithmetic  */
                                                /* expressions
                                                    } ()int lex
                          lexeme = "";  // Clear previous lexeme
                                                ;()getNonBlank

                                                } switch (charClass)
                                             /* Parse identifiers */
                                                   :case LETTER
                                                  ;()addChar
                                                   ;()getChar
while (charClass == LETTER || charClass ==
                                                           } DIGIT)
                                                 ;()addChar
                                                  ;()getChar
                                                           {
                                          ;nextToken = IDENT
                                                       ;break

                                        /* Parse integer literals */
                                                    :case DIGIT
                                                   ;()addChar
```

```
                                    ;()getChar
                            } while (charClass == DIGIT)
                                        ;()addChar
                                         ;()getChar
                                                {
                                ;nextToken = INT_LIT
                                            ;break

                            /* Parentheses and operators */
                                        :case UNKNOWN
                                    ;lookup(nextChar)
                                        ;()getChar
                                            ;break

                                            /* EOF */
                                    :case EOF_TOKEN
                                ;nextToken = EOF_TOKEN
                                    ;"lexeme = "EOF
                                            ;break
                                                    {

cout << "Next token is: " << nextToken << ", Next
                                    ;lexeme is " << lexeme << endl
                                        ;return nextToken
                                                    {
```