Lesson 2
The Lexer (Tokenizer)

Covered in this lesson:

- *What tokens are*
- *Token types*
- *Building a simple lexer in C++*

What is a TOKEN?

Before the interpreter understands the code, it **breaks text into pieces** called ***tokens***. This process is called **lexical analysis** or **tokenizing**.

Input:

```
LET x = 5 + 3
```

Your interpreter sees:

```
[LET] [x] [=] [5] [+] [3]
```

This things are called TOKENS

What is a Token Type?

Token Types are what each of your token is classified as:

Token	Type
LET	KEYWORD
X	IDENTIFIER
=	EQUAL
5	NUMBER
+	PLUS
3	NUMBER

Building a Lexer

These are the libraries need for building this lexer. Keep in mind that libraries will be added as the lesson progress.

```
#include <string>
#include <iostream>
#include <vector>
#include <cctype>
#include <unordered_set>
```

Step 1:

Think of the token types you want to include in your lexer.

```
//define the token types that exists
enum class TokenType{
    NUMBER,
    IDENTIFIER,
    KEYWORD,
    PLUS,
    MINUS,
    STAR,
    SLASH,
    EQUAL,
    LPAREN,
    RPAREN,
    SEMICOLON,
    END
};
```

Step 2

Define the Token Structure
Each token has a **type** and **value**:

```
struct Token{
    TokenType type;
    std::string value;
};
```

Step 3

The Lexer Class -The lexer walks through your source code.

The class will have more functions as the interpreter is being built.

```
class Lexer{
    std::string text;
    int pos = 0;
    char current;

    public:
    Lexer(std::string src): text(src){
        current = text[pos];
    }

    void advance();
    void skipWhiteSpace();
    Token number();
    Token identifier();
    Token getNextToken();
    char peek();

};
```

Functions Explanations

`advance() – is called everytime the current position is to be moved.`

```
void Lexer::advance(){
    pos++;
    //if the end of the string is reached, make the current = to NULL
    if(pos >= text.size()) current = '\0';
    else current = text[pos]; //if not, move to the next character
}
```

`skipWhiteSpace() – tells the lexer to not read spaces`

```
void Lexer::skipWhiteSpace(){
    while(current == ' ') advance();
}
```

number() – is of Token type function which returns the type of the token and the value of it. This considers both integer and floating numbers.

```
Token Lexer::number(){
    std::string result;
    bool dotUsed = false;

    while(isdigit(current) || current == '.' && !dotUsed){
        if(current == '.'){
            dotUsed = true;
        }
        result += current;
        advance();
    }

    return {TokenType::NUMBER, result};
}
```

Identifier() – returns a Token type. The function is used everytime the lexer begins to read a letter. This does not read whitespaces but reads both letters and numbers.

Ex: Who (W->h->o), so now the result = “Who”

Ex: x1 (x->1), result = “x1”

I know there is a number in the x1. Why is 1 not classified as a number?

Like I said, the function is called everytime it encounters a letter and ends when it sees a whitespace. So x1 is different from x 1

```
Token Lexer::identifier(){
    std::string result;

    while(isalnum(current)){
        result += current;
        advance();
    }

    static std::unordered_set<std::string> keywords = {
        "PRINT", "LET", "IF", "ELSE", "INPUT", "THEN", "FOR", "TO", "NEXT",
"END"
    };

    if(keywords.count(result)) return {TokenType::KEYWORD, result};
```

```
    return {TokenType::IDENTIFIER, result};
}
```

peek() – returns the next character without moving the position

```
char Lexer::peek(){
    if(pos + 1 >= text.size()) return '/0';
    return text[pos + 1];
}
```

getNextToken() – this is where the lexer begins to walk through your source code. This is cliché but, this is the heart of your Lexer Class. This is where your code is read and where the rest of the functions are called.

If you would look at it, you would notice that we begin with a while loop. This while loop will continue running as long as it has not reach the end of the string.

How does it know if it has reach the end? If you would back read at the advance() function you would see that the pos variable keeps incrementing everytime the function is called. and in the advance() function, there is a condition that if the pos >= the length or size of the text (in this case, the source code that you want to interpret), it would make the current = \0. And when it does that the while loop will then stop.

```
Token Lexer::getNextToken(){

    while(current != '\0'){
        if(current == ' '){
            skipWhiteSpace();
            continue;
        }

        if(isdigit(current)) return number();
        if(isalpha(current)) return identifier();

        if(current == '+') {
            advance();
            return {TokenType::PLUS, "+"};
```

```
        }
        if(current == '-') {
            advance();
            return {TokenType::MINUS, "-"};
        }
        if(current == '*') {
            advance();
            return {TokenType::STAR, "*"};
        }
        if(current == '/') {
            advance();
            return {TokenType::SLASH, "/"};
        }
        if(current == '=') {
            advance();
            return {TokenType::EQUAL, "="};
        }
        if(current == '(') {
            advance();
            return {TokenType::LPAREN, "("};
        }
        if(current == ')') {
            advance();
            return {TokenType::RPAREN, ")"};
        }
        if(current == ';') {
            advance();
            return {TokenType::SEMICOLON, ";"};
        }


        throw std::runtime_error("Invalid Character: " + current);
    }

    return {TokenType::END, ""};

}
```

(optional)

You can add this to get your token type, if you want. This is also helpful if you are gonna debug. This is a function separate from the Lexer class.

```
//token name getter
std::string tokenNames(TokenType t){
    switch (t)
    {
        case TokenType::NUMBER: return "NUMBER";
```

```
        case TokenType::IDENTIFIER: return "IDENTIFIER";
        case TokenType::KEYWORD: return "KEYWORD";
        case TokenType::PLUS: return "PLUS";
        case TokenType::MINUS: return "NUMBER";
        case TokenType::STAR: return "IDENTIFIER";
        case TokenType::SLASH: return "KEYWORD";
        case TokenType::EQUAL: return "PLUS";
        case TokenType::LPAREN: return "KEYWORD";
        case TokenType::RPAREN: return "PLUS";
        case TokenType::SEMICOLON: return "SEMICOLON";
        case TokenType::END: return "END";
        default:
            return "UNKNOWN";
    }
}
```

Step 4 – Running the Lexer

```
int main(){
    Lexer lexer("PRINT x x = 20.5 + 25");

    Token t;
    do{
        t = lexer.getNextToken();
        std::cout << "Token " << (int)t.type << " : " << tokenNames(t.type) <<
"->" << t.value << "\n";
    }while(t.type != TokenType::END);
}
```

Full Program

```
#include <string>
#include <iostream>
#include <vector>
#include <cctype>
#include <unordered_set>

//define the token types that exists
enum class TokenType{
    NUMBER,
    IDENTIFIER,
    KEYWORD,
    PLUS,
    MINUS,
```

```
    STAR,
    SLASH,
    EQUAL,
    LPAREN,
    RPAREN,
    SEMICOLON,
    END
};

struct Token{
    TokenType type;
    std::string value;
};


class Lexer{
    std::string text;
    int pos = 0;
    char current;

    public:
    Lexer(std::string src): text(src){
        current = text[pos];
    }

    void advance();
    void skipWhiteSpace();
    Token number();
    Token identifier();
    Token getNextToken();
    char peek();

};

void Lexer::advance(){
    pos++;
    //if the end of the string is reached, make the current = to NULL
    if(pos >= text.size()) current = '\0';
    else current = text[pos]; //if not, move to the next character
}

void Lexer::skipWhiteSpace(){
    while(current == ' ') advance();
}

Token Lexer::number(){
    std::string result;
    bool dotUsed = false;
```

```
    while(isdigit(current) || current == '.' && !dotUsed){
        if(current == '.'){
            dotUsed = true;
        }
        result += current;
        advance();
    }

    return {TokenType::NUMBER, result};
}

Token Lexer::identifier(){
    std::string result;

    while(isalnum(current)){
        result += current;
        advance();
    }

    static std::unordered_set<std::string> keywords = {
        "PRINT", "LET", "IF", "ELSE", "INPUT", "THEN", "FOR", "TO", "NEXT",
"END"
    };

    if(keywords.count(result)) return {TokenType::KEYWORD, result};

    return {TokenType::IDENTIFIER, result};
}

char Lexer::peek(){
    if(pos + 1 >= text.size()) return '/0';
    return text[pos + 1];
}

Token Lexer::getNextToken(){

    while(current != '\0'){
        if(current == ' '){
            skipWhiteSpace();
            continue;
        }

        if(isdigit(current)) return number();
        if(isalpha(current)) return identifier();

        if(current == '+') {
            advance();
            return {TokenType::PLUS, "+"};
```

```
        }
        if(current == '-') {
            advance();
            return {TokenType::MINUS, "-"};
        }
        if(current == '*') {
            advance();
            return {TokenType::STAR, "*"};
        }
        if(current == '/') {
            advance();
            return {TokenType::SLASH, "/"};
        }
        if(current == '=') {
            advance();
            return {TokenType::EQUAL, "="};
        }
        if(current == '(') {
            advance();
            return {TokenType::LPAREN, "("};
        }
        if(current == ')') {
            advance();
            return {TokenType::RPAREN, ")"};
        }
        if(current == ';') {
            advance();
            return {TokenType::SEMICOLON, ";"};
        }


        throw std::runtime_error("Invalid Character: " + current);
    }

    return {TokenType::END, ""};

}

//token name getter
std::string tokenNames(TokenType t){
    switch (t)
    {
        case TokenType::NUMBER: return "NUMBER";
        case TokenType::IDENTIFIER: return "IDENTIFIER";
        case TokenType::KEYWORD: return "KEYWORD";
        case TokenType::PLUS: return "PLUS";
        case TokenType::MINUS: return "NUMBER";
        case TokenType::STAR: return "IDENTIFIER";
        case TokenType::SLASH: return "KEYWORD";
```

```
        case TokenType::EQUAL: return "PLUS";
        case TokenType::LPAREN: return "KEYWORD";
        case TokenType::RPAREN: return "PLUS";
        case TokenType::SEMICOLON: return "SEMICOLON";
        case TokenType::END: return "END";
        default:
            return "UNKNOWN";
    }
}

int main(){
    Lexer lexer("PRINT x x = 20.5 + 25");

    Token t;
    do{
        t = lexer.getNextToken();
        std::cout << "Token " << (int)t.type << " : " << tokenNames(t.type) <<
"->" << t.value << "\n";
    }while(t.type != TokenType::END);
}
```