Compiler Construction

CS-471

By Laeeq Khan Niazi

Dua

اللَّهُمَّ إِنِّي أَسْأَلُكَ عِلْمًا نَافِعًا وَرِزْقًا طَيِّبًا وَعَمَلاً مُتَقَبَّلاً

O Allah, I ask You for beneficial knowledge, goodly provision and acceptable deeds

CLOs

- Understand and Apply Mathematical Formalisms like Regular Expressions, Grammars, FAs & PDAs.
- Understand the working principles of various phases of a compiler and how they are integrated to produce a working.
- Compare Various Implementations Techniques of Phases of Compiler, in particular Lexical Analyser, Parser
- Be able to work in a team (preferably alone) to design, develop, test, and deliver analysis phases of a compiler

Books

- Compiler – Principal, Techniques and Tools by Aho, Sethi and Ullman
- Engineering a Compiler – Keith D. Cooper & Linda Torczon 2nd Edition

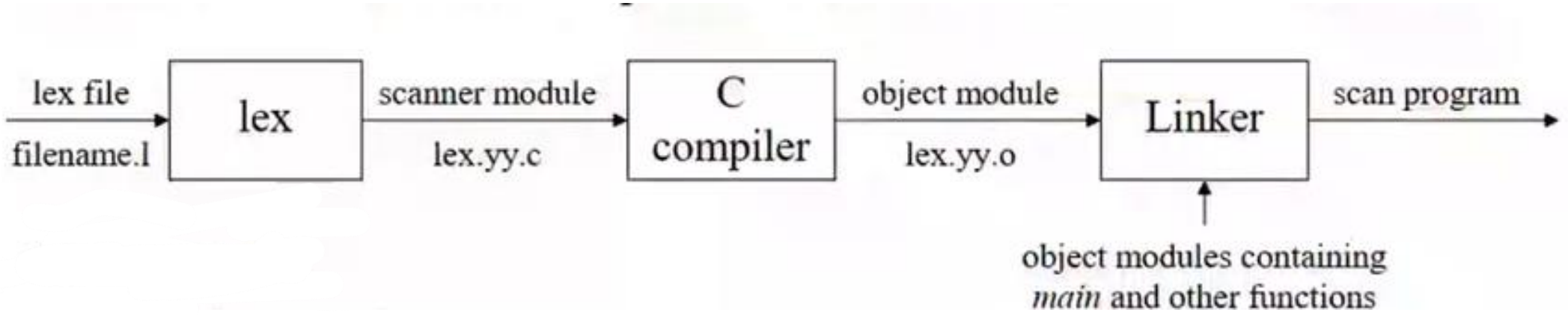
Agenda

- Building a Lexical Analyzer using C++ and Flex (Lex)

What will be the space complexity of the Tokenize function?

Using the FLex Scanner Generator

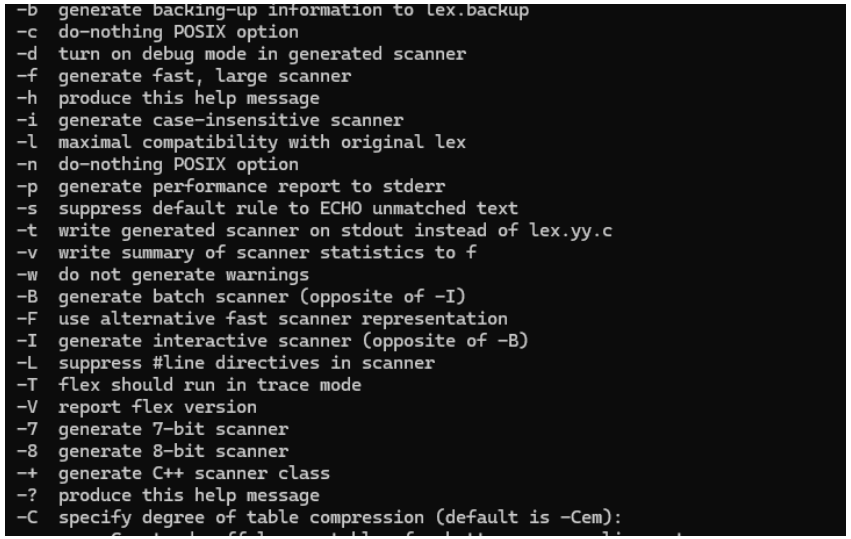
- Lex is popular scanner (Lexical Analyzer) generator
- Input to the FLex is called FLex Specification or Lex Program.
- Write your code and save the file with .l extention



Install Flex (Lex)

- https://gnuwin32-flex1.software.informer.com/download/

- Flex –help command the available features
- If not setup, configure the environment variable



Lex program to recognize and display keywords, numbers and words in a given statement.

Every lex program 3 sections

- Definition Section
- Rules Section
- User Subroutines (C Code) Section

first.l

```
%{
#include <stdio.h>
%}

%%
if|else|printf      { printf("%s is a keyword\n", yytext); }
[0-9]+              { printf("%s is a number\n", yytext); }
[a-zA-Z]+           { printf("%s is a word\n", yytext); }
[ \t\n]             ;  // Skip whitespace (spaces, tabs, newlines)
.                   { printf("Unknown character: %s\n", yytext); }
%%

int main() {
    printf("\nEnter the string here:\n");
    yylex();  // Call the lexical analyzer to start scanning input
    return 0;
}

int yywrap() {
    return 1;
}
```

Compile and run

```
D:\Teaching\Compiler Construction\Codes>flex first.l

D:\Teaching\Compiler Construction\Codes>g++ lex.yy.c -o lexer.exe

D:\Teaching\Compiler Construction\Codes>lexer.exe

Enter the string here:
int main()
int is a word
main is a word
Unknown character: (
Unknown character: )
```

Task

- Define more concrete rules in the file and generate the tokens for your any recursion C++ program.

Write a tokenizer using C++ code

Use the STL and collections available in C++

https://www.geeksforgeeks.org/the-c-standard-template-library-stl/

Algorithm

Initialize:

Create an empty list to store tokens.

Set an index to track the current position in the input string.

Process the String:

Loop through the string until you reach the end:

- **Skip Whitespace**: If the current character is a whitespace, move to the next character.
- **Identifiers/Keywords**: If the current character is a letter, collect the following alphanumeric characters to form a token. Check if this token is a keyword or an identifier.
- **Numbers**: If the current character is a digit, collect the following digits to form a number token.
- **Operators**: If the current character is an operator, create a token for it.
- **Punctuation**: If the current character is punctuation, create a token for it.
- **Unknown Characte**rs: For any other characters, create a token labeled as unknown.

Return:

Return the list of tokens.

Lab Tasks

By using best optimized data structures write the tokenizer for your language.

Include some helping libraries

lexical.cpp

```cpp
#include <iostream>
#include <cctype>
#include <string>
#include <vector>
#include <unordered_set>
using namespace std;

```

Create token type **enum** and template of token using **struct**

```
8  enum TokenType { //Define your enum or any other data structure of your choice
9      KEYWORD,
10     IDENTIFIER,
11     OPERATOR,
12     NUMBER,
13     PUNCTUATION,
14     UNKNOWN
15 };
16
17
18 struct Token { //Structure data type to store the template of token
19     string lexeme;
20     TokenType type;
21 };
22
```

Some lists to store keywords, operators and other symbols of the language

```
23
24  unordered_set<string> keywords = {"if", "else", "for", "while", "return", "int", "float", "char", "void", "double"};
25  unordered_set<char> operators = {'+', '-', '*', '/', '=', '>', '<', '&', '|', '!'};
26  unordered_set<char> punctuation = {'(', ')', '{', '}', ';', ','};
27
```

Breaking down your logic into functions

- Function to check string is keyword or not

```
28    bool isKeyword(const string& str) {
29        return keywords.find(str) != keywords.end();
30    }
31
```

Function to tokenize the input string

```
32  vector<Token> tokenize(const string& input) {
33      vector<Token> tokens;
34      int i = 0;
35      while (i < input.length()) {
36
37          if (isspace(input[i])) { // Skip whitespaces
38              i++;
39              continue;
40          }
41          if (isalpha(input[i])) {// Check if the token is an identifier or keyword
42              string lexeme;
43              while (isalnum(input[i])) {
44                  lexeme += input[i];
45                  i++;
46              }
47              if (isKeyword(lexeme)) {
48                  tokens.push_back({lexeme, KEYWORD});
49              } else {
50                  tokens.push_back({lexeme, IDENTIFIER});
51              }
52          }
```

Isspace function?

- **isspace** is a built-in function in C++ (as well as in C). It is part of the <cctype> (or <ctype.h> in C) standard library and is used to check whether a given character is a whitespace character. This includes spaces, tabs, newlines, and other space-related characters.

Isalpha function??

- **isalpha** is a built-in function in C++ (and C) that is used to check whether a given character is an alphabetic letter (either uppercase or lowercase). It is part of the <cctype> header in C++ (or <ctype.h> in C)

Isdigit function?

- isdigit is a built-in function in C++ provided by the C Standard Library. It is used to check if a given character is a digit (0-9)

Why Passing strings by reference?

- Strings in C++ can be large because they are dynamically allocated. When you pass a string by value (i.e., string str), C++ makes a copy of the entire string. This involves copying not only the string's metadata but also all the characters, which can be computationally expensive, especially if the string is large.
- By passing a string by reference (i.e., const string& str), no copy is made. Instead, a reference to the original string is passed, making the function call more efficient.
- In a compiler, lexical analysis (tokenization) can involve processing large amounts of text (the source code). If every token or string were passed by value, it would involve copying potentially thousands of characters repeatedly, which would slow down the compiler significantly.
- The **const** keyword ensures that the function cannot modify the string being passed.

```
53          else if (isdigit(input[i])) { // Check if the token is a number
54              string lexeme;
55              while (isdigit(input[i])) {
56                  lexeme += input[i];
57                  i++;
58              }
59              tokens.push_back({lexeme, NUMBER});
60          }
61
62          else if (operators.find(input[i]) != operators.end()) { // Check if the token is an operator
63              string lexeme(1, input[i]);
64              tokens.push_back({lexeme, OPERATOR});
65              i++;
66          }
67
68          else if (punctuation.find(input[i]) != punctuation.end()) {  // Check if the token is punctuation
69              string lexeme(1, input[i]);
70              tokens.push_back({lexeme, PUNCTUATION});
71              i++;
72          }
73          else { // Unknown character
74              string lexeme(1, input[i]);
75              tokens.push_back({lexeme, UNKNOWN});
76              i++;
77          }
78      }
79      return tokens;
80  }
```

Function to print the tokens

```
83  void printTokens(const vector<Token>& tokens) {
84      for (const auto& token : tokens) {
85          cout << "Token: " << token.lexeme << ", Type: ";
86          switch (token.type) {
87              case KEYWORD:
88                  cout << "Keyword";
89                  break;
90              case IDENTIFIER:
91                  cout << "Identifier";
92                  break;
93              case OPERATOR:
94                  cout << "Operator";
95                  break;
96              case NUMBER:
97                  cout << "Number";
98                  break;
99              case PUNCTUATION:
100                 cout << "Punctuation";
101                 break;
102             case UNKNOWN:
103                 cout << "Unknown";
104                 break;
105         }
106         cout << endl;
107     }
108 }
```

What will be the time complexity of this Tokenize function?

Week 4
Lab Assessment

Contents for Lab assessment

- Phasis of Compiler with short understanding
- C++ Fundamentals
- C++ OOP Concepts
- STL
- Template Programming
- Threading in C++