Compiler Design

Lab – 2

Dr. Hazha Saeed yahia

Email: hazha.yahia@knu.edu.iq

Implementing lexical analysis using C

Introduction:

In this lab we will learn how to write a program in C for finding the lexemes fir any given codes.

In the lexical analysis phase, input is the source program that is to be read from left to right and the output we get is a sequence of tokens that will be analyzed by the next Syntax Analysis phase. During scanning the source code, white space characters, comments, carriage return characters, preprocessor directives, macros, line feed characters, blank spaces, tabs, etc. are removed.

The steps of writing the C codes:

Step 1: Start the program.

Step 2: Include necessary header files, (<stdbool.h>,<stdio.h>, <string.h>, <stdlib.h>).

Step 3: Define the main Tokens (delimiter, operator, identifier, keyword, integer, real number).

Step 4: Return “true” values for the tokens using Boolean.

Step 5: Extract the substring.

Step 6: Detect the tokens.

Step 7: Drive the main function.

The code:

```
#include <stdbool.h> // a header file for using bool
```

```
#include <stdio.h> // a header file for importing different variables, macros, and functions to perform input and output operators

#include <string.h> // a header file contains functions for manipulating strings

#include <stdlib.h> // a header file that declare various utility functions

// Returns 'true' if the character is a DELIMITER.

bool isValidDelimiter(char ch) {

  if (ch == ' ' || ch == '+' || ch == '-' || ch == '*' ||

  ch == '/' || ch == ',' || ch == ';' || ch == '>' ||

  ch == '<' || ch == '=' || ch == '(' || ch == ')' ||

  ch == '[' || ch == ']' || ch == '{' || ch == '}')

  return (true);

  return (false);

}

// Returns 'true' if the character is an OPERATOR

bool isValidOperator(char ch){

  if (ch == '+' || ch == '-' || ch == '*' ||

  ch == '/' || ch == '>' || ch == '<' ||

  ch == '=')

  return (true);

  return (false);

}

// Returns 'true' if the string is a VALID IDENTIFIER.

bool isvalidIdentifier(char* str){

  if (str[0] == '0' || str[0] == '1' || str[0] == '2' ||
```

```
  str[0] == '3' || str[0] == '4' || str[0] == '5' ||
  str[0] == '6' || str[0] == '7' || str[0] == '8' ||
  str[0] == '9' || isValidDelimiter(str[0]) == true)
  return (false);
  return (true);
}
// Returns 'true' if the string is a KEYWORD
bool isValidKeyword(char* str) {
  if (!strcmp(str, "if") || !strcmp(str, "else") || !strcmp(str, "while") || !strcmp(str, "do") ||
!strcmp(str, "break") || !strcmp(str, "continue") || !strcmp(str, "int")
  || !strcmp(str, "double") || !strcmp(str, "float") || !strcmp(str, "return") || !strcmp(str,
"char") || !strcmp(str, "case") || !strcmp(str, "char")
  || !strcmp(str, "sizeof") || !strcmp(str, "long") || !strcmp(str, "short") || !strcmp(str,
"typedef") || !strcmp(str, "switch") || !strcmp(str, "unsigned")
  || !strcmp(str, "void") || !strcmp(str, "static") || !strcmp(str, "struct") || !strcmp(str, "goto"))
  return (true);
  return (false);
}
// Returns 'true' if the string is an INTEGER
bool isValidInteger(char* str) {
  int i, len = strlen(str);
  if (len == 0)
  return (false);
  for (i = 0; i < len; i++) {
    if (str[i] != '0' && str[i] != '1' && str[i] != '2'&& str[i] != '3' && str[i] != '4' && str[i] != '5'
```

```
    && str[i] != '6' && str[i] != '7' && str[i] != '8' && str[i] != '9' || (str[i] == '-' && i > 0))
    return (false);
  }
  return (true);
}
// Returns 'true' if the string is a REAL NUMBER
bool isRealNumber(char* str) {
  int i, len = strlen(str);
  bool hasDecimal = false;
  if (len == 0)
  return (false);
  for (i = 0; i < len; i++) {
    if (str[i] != '0' && str[i] != '1' && str[i] != '2' && str[i] != '3' && str[i] != '4' && str[i]      != '5'
&& str[i] != '6' && str[i] != '7' && str[i] != '8'
    && str[i] != '9' && str[i] != '.' || (str[i] == '-' && i > 0))
    return (false);
      if (str[i] == '.')
    hasDecimal = true;
  }
  return (hasDecimal);
}
// Extracts the SUBSTRING
char* subString(char* str, int left, int right) {
  int i;
```

```
  char* subStr = (char*)malloc( sizeof(char) * (right - left + 2));
  for (i = left; i <= right; i++)
    subStr[i - left] = str[i];
  subStr[right - left + 1] = '\0';
  return (subStr);
}

// Detect the Tokens
void detectTokens(char* str) {
  int left = 0, right = 0;
  int length = strlen(str);
  while (right <= length && left <= right) {
    if (isValidDelimiter(str[right]) == false)
    right++;
    if (isValidDelimiter(str[right]) == true && left == right) {
      if (isValidOperator(str[right]) == true)
      printf("Valid operator : '%c' \n", str[right]);
      right++;
      left = right;
    } else if (isValidDelimiter(str[right]) == true && left != right || (right == length && left != right)) {
      char* subStr = subString(str, left, right - 1);
      if (isValidKeyword(subStr) == true)
        printf("Valid keyword : '%s'\n", subStr);
```

```
      else if (isValidInteger(subStr) == true)
        printf("Valid Integer : '%s'\n", subStr);
      else if (isRealNumber(subStr) == true)
        printf("Real Number : '%s'\n", subStr);
      else if (isvalidIdentifier(subStr) == true
        && isValidDelimiter(str[right - 1]) == false)
      printf("Valid Identifier : '%s'\n", subStr);
      else if (isvalidIdentifier(subStr) == false
        && isValidDelimiter(str[right - 1]) == false)
      printf("Invalid Identifier : '%s'\n", subStr);
      left = right;
    }
  }
  return;
}
// DRIVER FUNCTION
int main(){
  char str[100] = "float x = 3 + 1b; \n";
  printf("The Program is : '%s'\n", str);
  printf("All Tokens are :\n");
  detectTokens(str);
  return (0);
}
```

Another code for asking the used to insert the codes for recognizing the tokens:

```
// DRIVER FUNCTION

int main()

{

   char str[100];

   printf("Enter the String: \n");

   scanf("%[^\n]s", &str);

   detectTokens(str); // calling the parse function

   return (0);

}
```

Explanation:

1. isDelimiter:

First, we need to understand what Delimiter is. A Delimiter is a character that separates text strings by one or more characters. Commas (,), semicolon (;), quotes (“, ‘), braces (), pipes (|), and slashes (/) are all common delimiters. When software saves sequential or tabular data, it uses a preset character to delimit each item of data.

== means “Equal to”. That means when “ch” is equal to any of the given delimiters, then it will return True, otherwise False.

2. isOperator

It is similar to the Delimiter. When the “ch” value is equal to any of the given operators, then it will return true, otherwise false.

3. isValidIdentifier

It is checking if the string is a Valid Identifier or not. To be a valid identifier, the first letter of an identifier can’t be a number and special character (Delimiter). It is checking whether the first letter or character of the identifier is a Number or Special Character (Delimiter).

If the first letter of a string is a number or special character then this function will return false, otherwise, return true.

4. isKeyword

The iskeyword() method accepts a word/string as a parameter and returns a Boolean value True or False based on the input argument. It returns true for all the reserved words and for the rest it will give the result as False.

5. isIntegr

In this section, first, we check if the string length is 0. If the string length is 0, then we don’t need to check it and it returns the value False.

But if the length is not 0, then it will go to the for loop and then if condition statement. In the if conditional statement we are seeing if the string does not contain any of the Digit or contain – (minus) and if the array is not 0.

Many of you can ask why we are seeing if it contains minus (-) we already checked it in isOperator functions. It is just a precautionary measure. The code will run smoothly without

|| (str[i] == '-' && i > 0) this section. You can try it. Don't forget to tell us what occurs.

So, in simple words, in this section, we are checking for digits. If the string contains any character other than a digit then it will return false, otherwise true.

6. isRealNumber

It is similar to the isInteger() function. But we are checking if it has a decimal (.) value. If it has a decimal value it will return true, otherwise false.

7. Extract the Substrings

A substring is a contiguous sequence of characters within a string. The function **subString** builds a dynamically allocated array that will contain a substring of the passed string with the range [left, right]. So, the length of the substring is calculated like (right-left+1). Also, one more byte shall be reserved for the terminating zero character '/0' of the substring. Therefore, in whole there is a required to allocate memory of the size right-left+2.

*** malloc () methods:**

The "malloc" or "memory allocation" method in C is used to dynamically allocate a single large block of memory with the specified size. It returns a pointer of type

void which can be cast into a pointer of aby form. It doesn’t initialize memory at execution time so that it has initialized each block with the default garbage value initially.

* [right-left+2]

right - left + 1 to calculate the number of characters in the substring (because both indices are included in the substring) + another 1 for the null terminator '\0'.

The **right - left + 2** calculate the correct values in practice and there is required to allocate memory of the size right – left +2.

Compiler Design

Lab – 4

Dr. Hazha Saeed Yahia

Email: hazha.yahia@knu.edu.iq

Writing programs in Lex and Yacc

Introduction:

In this lab we will learn how to write simple programs for both lexical and syntax phases of computer compiler.

Example_1:

Write a simple program for recognizing and display keywords, numbers, and words in a statement. For this purpose we first write the codes in DevC++ and save it in any name with extension of (.l). Let is save it as (new.l).

Note:

1. (l) is a small letter of (L).
2. The green lines in the codes are explanations do not write them in your codes.

The steps of for writing the codes in DevC++ with explanation:

The full code:

```
*Lex code for recognizing keywords, numbers, and words */

%{
#include <stdio.h>
%}

// rule section
%%
if |
else |

// recognition rules
```

```
printf {printf("%s is a keyword", yytext);}
[0-9]+ {printf ("%s is a number", yytext);}
[a-zA-Z]+ {printf ("%s is a word", yytext);}
.|\n {ECHO;}
%%

// asking the user to enter a string
int main()
{
	printf("\n Enter the string:");
	yylex();
}

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

Example_2: A program for recognizing tokens and counting the total number of tokens:

```
*Lex code to recognize tokens and count total number of tokens */

%{
int n = 0;
%}

// rule section
%%

//count number of keywords
"while"|"if"|"else" {n++;printf("\t keywords : %s", yytext);}

// count number of keywords
"int"|"float" {n++;printf("\t keywords : %s", yytext);}

// count number of identifiers
[a-zA-Z_][a-zA-Z0-9_]* {n++;printf("\t identifier : %s", yytext);}

// count number of operators
"<="|"=="|"="|"++"|"-"|"*"|"+" {n++;printf("\t operator : %s", yytext);}

// count number of separators
[(){}|, ;]    {n++;printf("\t separator : %s", yytext);}
```

```
// count number of floats
[0-9]*"."[0-9]+ {n++;printf("\t float : %s", yytext);}

// count number of integers
[0-9]+ {n++;printf("\t integer : %s", yytext);}

.    ;
%%

int main()
{
    yylex();
    printf("\n total no. of token = %d\n", n);
}
int yywrap()
{
      return 1;
}
```