Jashore University of Science and Technology

Department of Computer Science and Engineering

Course Code: CSE-3204

Course Title: Compiler Design Labratory

A Lab Report On

Lexical analysis

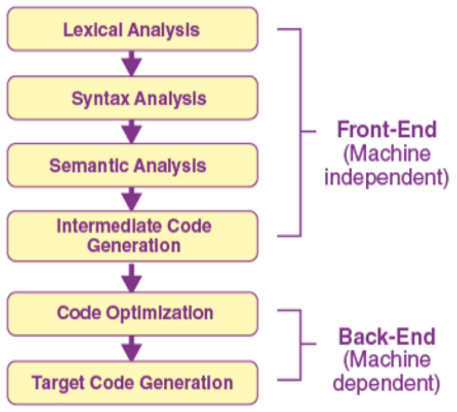
Submitted to	Submitted by
Mustain Billah Lecturer, Department of Computer Science and Engineering Jashore University of Science and Technology	**Shongkor Talukdar** Roll: 180129 3rd Year 2nd Semester Session: 2018-2019 Dept. of Computer Science and Engineering Jashore University of Science and Technology.

Submission Date: 25.07.2022

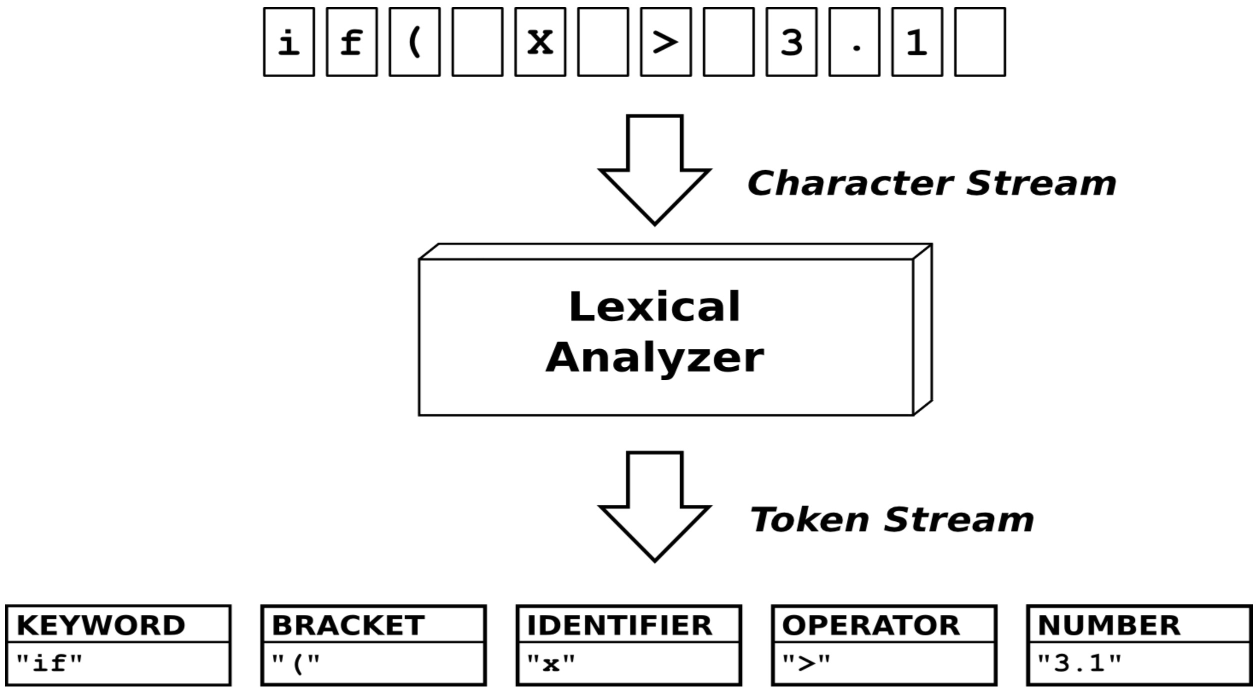
Experiment Name: Lexical analysis

Objective: Finding various kind of tokens and a token is either a keyword, an identifier, a constant, a string literal, or a symbol.

Description: The compilation procedure is nothing but a series of different phases. Each stage acquires input from its previous phase.


Lexical Analysis
Syntax Analysis
Semantic Analysis
Intermediate Code Generation
Code Optimization
Target Code Generation
Front-End (Machine independent)
Back-End (Machine dependent)

Lexical analysis is the first phase of compiler also known as scanner. It converts the input program into a sequence of Tokens. A C program consists of various tokens and a token is either a keyword, an identifier, a constant, a string literal, or a symbol.



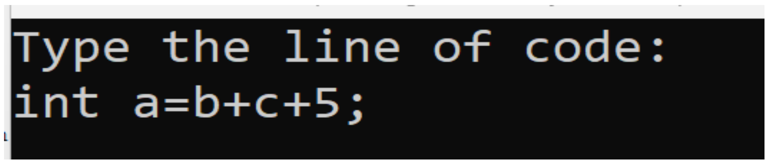
Below is a C program to print all the keywords, literals, valid identifiers, invalid identifiers, integer number, real number in a given C program:

Source Code :

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<stdbool.h>

bool isDelimeter(char ch)

{

   if(ch == ' ' || ch == ';' || ch == '=' || ch == '+' || ch == '-')

      return true;

   return false;

}

bool isOperator(char ch)

{

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

      return true;

   return false;

}

bool isKeyword(char* str)

{

   if(!strcmp(str, "int") || !strcmp(str, "if"))

      return true;

   return false;
```

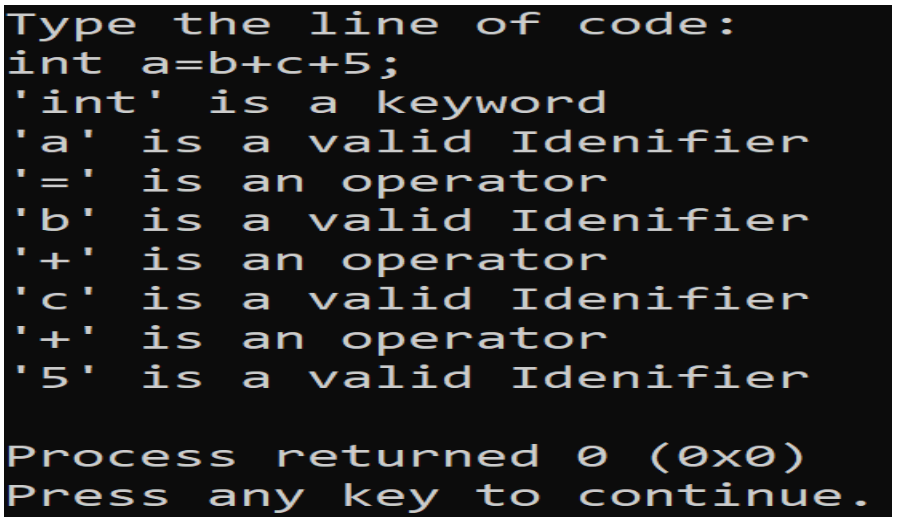
```
}
bool validIdenifier(char* str)
{
    if(str[0] == '0' || str[0] == '1' || isDelimeter(str[0]) == true)
        return false;
    return true;
}
bool isInteger(char* str)
{
    int i;
    int len = strlen(str);
    if(len == 0)
        return false;
    for(i = 0; i<len; i++)
    {
        if(str[i] != '0' && str[i] != '1')
            return false;
    }
    return true;
}
char* subStringGenerator(char* str, int left, int right)
{
    char* subString = (char*)malloc(sizeof(char) * (right-left +2));
    int i;
    for(i=left; i<=right; i++)
        subString[i-left] = str[i];
    subString[right-left+1] = '\0';
    return subString;
}
void parserFunction(char* str)
{
    int left = 0, right = 0;
    int len = strlen(str);
    while(right <= len && left <= right)
    {
        if(isDelimeter(str[right]) == false)
            right++;
        if(isDelimeter(str[right]) == true && left == right)
        {
            if(isOperator(str[right]) == true)
                printf("'%c' is an operator\n", str[right]);
            right++;
            left = right;
        }
        else if(isDelimeter(str[right]) == true && left != right || (left !=right && right == len))
        { char* subString = subStringGenerator(str, left, right-1);
            if(isKeyword(subString) == true)
                printf("'%s' is a keyword\n", subString);
            else if(isInteger(subString) == true)
                printf("'%s' is a Integer\n", subString);
            else if(validIdenifier(subString) == true)
                printf("'%s' is a valid Idenifier\n", subString);
            left = right;
        }
    }
}
int main()
{
    char str[100];
    printf("Type the line of code: \n");
    scanf("%[^\n]",str);
    parserFunction(str);
    return 0;
}
```

Input :



```
Type the line of code:
int a=b+c+5;
```

Output:



```
Type the line of code:
int a=b+c+5;
'int' is a keyword
'a' is a valid Idenifier
'=' is an operator
'b' is a valid Idenifier
'+' is an operator
'c' is a valid Idenifier
'+' is an operator
'5' is a valid Idenifier

Process returned 0 (0x0)
Press any key to continue.
```