Experiment - 3
CD G2 Lab A3 Batch

SUBMITTED BY - Kartik Verma
2K18-CO-169

Aim : To Implement the Lexical Analyser.

Theory :

Lexical analysis is the first phase of a compiler. It takes the modified source code from language preprocessors that are written in the form of sentences. The lexical analyzer breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code.

If the lexical analyzer finds a token invalid, it generates an error. The lexical analyzer works closely with the syntax analyzer. It reads character streams from the source code, checks for legal tokens, and passes the data to the syntax analyzer when it demands.

For Eg.
1) Operators:
Examples- '+', '++', '-' etc.
2) Separators:
Examples- ', ' ';' etc

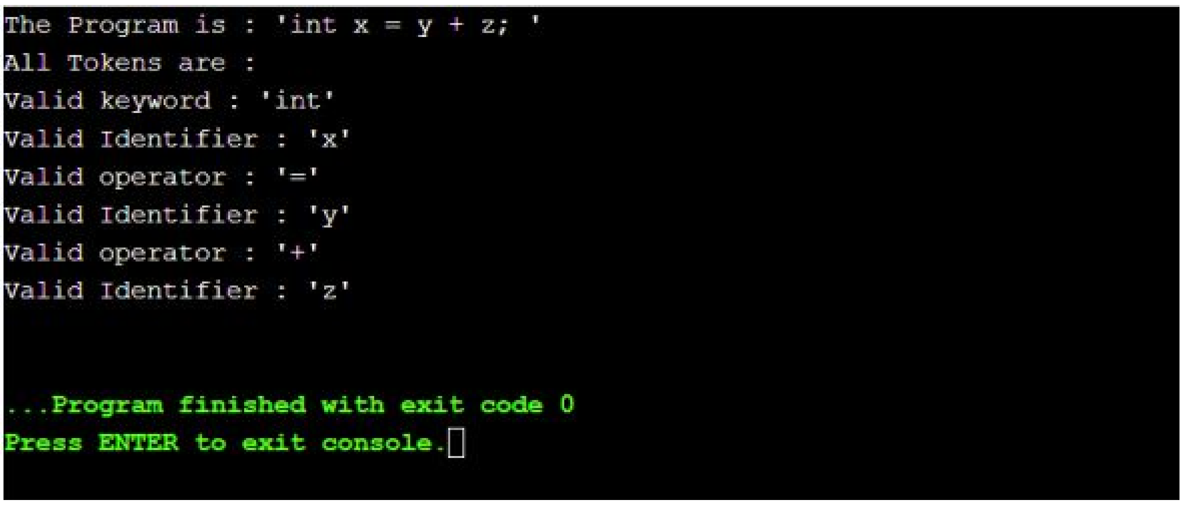
Code :

```
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

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);
}
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);
}
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, "t
        ypedef ") || !strcmp(str, "
        switch ") || !strcmp(str, "
        unsigned ") ||
        !strcmp(str, "void") || !strcmp(str, "static") || !strcmp(str,
"struct") || !strcmp(str, "
          goto "))
          return (true);
          return (false);
        }
        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' && st r[i] != '5' &&
              str[i] != '6' && str[i] != '7' && str[i] != '8' && str[i]
!= '9' || (str[i] == '-' && i > 0))
              return (false);
          }
          return (true);
        }
        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' && s tr[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);
        }
        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);
        }
        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 && le ft != 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;
        }
int main() {
      char str[100] = "float x = y + z; ";
      printf("The Program is : '%s' \n", str);
      printf("All Tokens are : \n");
      detectTokens(str);
      return (0);
 }
```

Output :



```
The Program is : 'int x = y + z; '
All Tokens are :
Valid keyword : 'int'
Valid Identifier : 'x'
Valid operator : '='
Valid Identifier : 'y'
Valid operator : '+'
Valid Identifier : 'z'


...Program finished with exit code 0
Press ENTER to exit console.
```