

Experiment Number 1

Aim: Implementation of Lexical Analyzer.

Algorithm:

Step 1: Start

Step 2: Read code into buffer.

Step 3: Analyse character by character.

Step 4: Identify tokens.

Step 5: Print respective output.

Step 6: Stop.

Code:

```
#include <bits/stdc++.h>
using namespace std;
bool isPunctuator(char ch) //check if the given character is a punctuator or not
{
    if (ch == ' ' || ch == '+' || ch == '-' || ch == '*' ||
        ch == '/' || ch == ',' || ch == ';' || ch == '>' ||
        ch == '<' || ch == '=' || ch == '(' || ch == ')' ||
        ch == '[' || ch == ']' || ch == '{' || ch == '}' ||
        ch == '&' || ch == '|')
    {
        return true;
    }
    return false;
}
bool validIdentifier(char* str) //check if the given identifier is valid or not
{
    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' || isPunctuator(str[0]))
    {
        return false;
    } //if first character of string is a digit or a special character, identifier is not valid
    int i, len = strlen(str);
    if (len == 1)
    {
        return true;
    } //if length is one, validation is already completed, hence return true
    else
    {
        for (i = 1 ; i < len ; i++) //identifier cannot contain special characters
        {
```

```

        if (isPunctuator(str[i]))
        {
            return false;
        }
    }
    return true;
}

bool isOperator(char ch) //check if the given character is an operator or not
{
    if (ch == '+' || ch == '-' || ch == '*' ||
        ch == '/' || ch == '>' || ch == '<' ||
        ch == '=' || ch == '|' || ch == '&')
    {
        return true;
    }
    return false;
}

bool isKeyword(char *str) //check if the given substring is a keyword or not
{
    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, "long") || !strcmp(str, "short")
        || !strcmp(str, "typedef") || !strcmp(str, "switch")
        || !strcmp(str, "unsigned") || !strcmp(str, "void")
        || !strcmp(str, "static") || !strcmp(str, "struct")
        || !strcmp(str, "sizeof") || !strcmp(str, "long")
        || !strcmp(str, "volatile") || !strcmp(str, "typedef")
        || !strcmp(str, "enum") || !strcmp(str, "const")
        || !strcmp(str, "union") || !strcmp(str, "extern")
        || !strcmp(str, "bool"))
    {
        return true;
    }
    else
    {
        return false;
    }
}

bool isNumber(char* str) //check if the given substring is a number or not
{
    int i, len = strlen(str), numOfDecimal = 0;
    if (len == 0)
    {
        return false;
    }
}

```

```

for (i = 0 ; i < len ; i++)
{
    if (numOfDecimal > 1 && str[i] == '.')
    {
        return false;
    } else if (numOfDecimal <= 1)
    {
        numOfDecimal++;
    }
    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;
}
char* subString(char* realStr, int l, int r) //extract the required substring from the main string
{
    int i;

    char* str = (char*) malloc(sizeof(char) * (r - l + 2));

    for (i = l; i <= r; i++)
    {
        str[i - l] = realStr[i];
        str[r - l + 1] = '\0';
    }
    return str;
}
void parse(char* str) //parse the expression
{
    int left = 0, right = 0;
    int len = strlen(str);
    while (right <= len && left <= right) {
        if (isPunctuator(str[right]) == false) //if character is a digit or an alphabet
        {
            right++;
        }

        if (isPunctuator(str[right]) && left == right) //if character is a punctuator
        {
            if(isOperator(str[right])&& isOperator(str[right+1])){
                std::cout<< str[right] <<str[right+1]<<" IS AN OPERATOR\n";
                right++;
            }
            else if (isOperator(str[right]))
            {

```

```

        std::cout<< str[right] <<" IS AN OPERATOR\n";
    }
    right++;
    left = right;
    } else if (isPunctuator(str[right]) && left != right
        || (right == len && left != right)) //check if parsed substring is a keyword or
identifier or number
    {
        char* sub = subString(str, left, right - 1); //extract substring

        if (isKeyword(sub))
        {
            cout<< sub <<" IS A KEYWORD\n";
        }
        else if (isNumber(sub))
        {
            cout<< sub <<" IS A NUMBER\n";
        }
        else if (validIdentifier(sub)
            && isPunctuator(str[right - 1]) == false)
        {
            cout<< sub <<" IS A VALID IDENTIFIER\n";
        }
        else if (validIdentifier(sub) == false
            && isPunctuator(str[right - 1]) == false)
        {
            cout<< sub <<" IS NOT A VALID IDENTIFIER\n";
        }
        left = right;
    }
}
}
int main()
{
    char c[100] = "double a=2;";
    parse(c);
    return 0;
}

```

Output:

```

double IS A KEYWORD
a IS A VALID IDENTIFIER
= IS AN OPERATOR
2 IS A NUMBER

```

Result: Thus, Lexical Analyzer implemented successfully.