# Experiment 4

## Aim:

WAP to design a Lexical analyzer for identifying different types of token used in C language.

## Code:

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

int isDelimiter(char ch)

{

if (ch == ' ' || ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == ',' || ch == ';' || ch == '>' || ch == '<' || ch == '=' || ch == '(' || ch == ')' || ch == '[' || ch == ']' || ch == '{' || ch == '}') return 1;

return 0;

}

int isOperator(char ch)

{

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

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

ch == '=' || ch == '{' || ch == '}' || ch == '(' || ch == ')') return 1;

return 0;

}

int validIdentifier(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' || isDelimiter(str[0]) == 1) return 0;

return 1;

}

int isKeyword(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 1;

return 0;

}

int isInteger(char \*str)

{

int i, len = strlen(str);

if (len == 0) return 0;

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 0;

}

return 1;

}

int isRealNumber(char \*str)

{

int i, len = strlen(str); int hasDecimal = 0;

if (len == 0) return 0;

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 0;

if (str[i] == '.') hasDecimal = 1;

}

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 parse(char \*str)

{

int left = 0, right = 0; int len = strlen(str);

while (right <= len && left <= right)

{

if (isDelimiter(str[right]) == 0) right++;

if (isDelimiter(str[right]) == 1 && left == right)

{

if (isOperator(str[right]) == 1)

{

printf("'%c' is an OPERATOR\n", str[right]);

}

right++; left = right;

}

else if (isDelimiter(str[right]) == 1 && left != right || (right == len && left != right))

{

char \*subStr = subString(str, left, right - 1); if (isKeyword(subStr) == 1)

{

printf("'%s' is a KEYWORD\n", subStr);

}

else if (isInteger(subStr) == 1)

{

printf("'%s' is a LITERAL (Integer)\n", subStr);

}

else if (isRealNumber(subStr) == 1)

{

printf("'%s' is a LITERAL (Real Number)\n", subStr);

}

else if (validIdentifier(subStr) == 1 && isDelimiter(str[right - 1]) == 0)

{

printf("'%s' is a valid IDENTIFIER\n", subStr);

}

else if (validIdentifier(subStr) == 0 && isDelimiter(str[right - 1]) == 0)

{

printf("'%s' is NOT a valid IDENTIFIER\n", subStr);

}

left = right;

}

}

return;

}

int main()

{

char str[50];

printf("Enter a statement: ");

gets(str);

printf("\n");

parse(str);

return (0);

}

**Output:**

Text

Description automatically generated

# Experiment 5

## Aim:

WAP to count the number of tokens in the given code.

## Code:

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

int isDelimiter(char ch)

{

if (ch == ' ' || ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == ',' || ch == ';' || ch == '>' || ch == '<' || ch == '=' || ch == '(' || ch == ')' || ch == '[' || ch == ']' || ch == '{' || ch == '}') return 1;

return 0;

}

int isOperator(char ch)

{

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

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

ch == '=' || ch == '{' || ch == '}' || ch == '(' || ch == ')') return 1;

return 0;

}

int validIdentifier(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' || isDelimiter(str[0]) == 1) return 0;

return 1;

}

int isKeyword(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 1;

return 0;

}

int isInteger(char \*str)

{

int i, len = strlen(str);

if (len == 0) return 0;

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 0;

}

return 1;

}

int isRealNumber(char \*str)

{

int i, len = strlen(str); int hasDecimal = 0;

if (len == 0) return 0;

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 0;

if (str[i] == '.') hasDecimal = 1;

}

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 parse(char \*str)

{

int left = 0, right = 0; int len = strlen(str); int total\_count = 0;

while (right <= len && left <= right)

{

if (isDelimiter(str[right]) == 0) right++;

if (isDelimiter(str[right]) == 1 && left == right)

{

if (isOperator(str[right]) == 1)

{

total\_count += 1;

}

right++; left = right;

}

else if (isDelimiter(str[right]) == 1 && left != right || (right == len && left != right))

{

char \*subStr = subString(str, left, right - 1); if (isKeyword(subStr) == 1)

{

total\_count += 1;

}

else if (isInteger(subStr) == 1)

{

total\_count += 1;

}

else if (isRealNumber(subStr) == 1)

{

total\_count += 1;

}

else if (validIdentifier(subStr) == 1 && isDelimiter(str[right - 1]) == 0)

{

total\_count += 1;

}

else if (validIdentifier(subStr) == 0 && isDelimiter(str[right - 1]) == 0)

{

total\_count += 1;

}

left = right;

}

}

printf("\nTotal number of tokens: %d", total\_count); return;

}

int main()

{

char str[50];

printf("Enter a statement: "); gets(str);

printf("\n"); parse(str); return (0);

}

**Output:**

Text

Description automatically generated

# Experiment 6

## Aim:

WAP to identify whether a given line is a comment or not.

## Code:

import re

def main():

statement = input('Enter statement:')

regex1 = re.search("//(.\*)", statement)

regex2 = re.search("/\\\*(.\*?)\\\*/", statement)

if regex1 or regex2:

print("It is a comment!")

else:

print("It is not a comment!")

if \_\_name\_\_== '\_\_main\_\_':

main()

**Output:**

Text

Description automatically generated with medium confidence

# Experiment 7

## Aim:

WAP to create a symbol table for following input strings.

1. int i , j;
2. int k = int i + int j;

## Code:

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

int isDelimiter(char ch)

{

if (ch == ' ' || ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == ',' || ch == ';' || ch == '>' || ch == '<' || ch == '=' || ch == '(' || ch == ')' || ch == '[' || ch == ']' || ch == '{' || ch == '}') return 1;

return 0;

}

int validIdentifier(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' || isDelimiter(str[0]) == 1) return 0;

return 1;

}

int isKeyword(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 1;

return 0;

}

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 parse(char \*str)

{

int left = 0, right = 0; int len = strlen(str); int count = 0;

while (right <= len && left <= right)

{

if (isDelimiter(str[right]) == 0) right++;

if (isDelimiter(str[right]) == 1 && left == right)

{

right++; left = right;

}

else if (isDelimiter(str[right]) == 1 && left != right || (right == len && left != right))

{

char \*subStr = subString(str, left, right - 1);

if (isKeyword(subStr) == 0 && validIdentifier(subStr) == 1 && isDelimiter(str[right - 1]) ==

0)

{

count++;

printf("%s id%d\n", subStr,count);

}

left = right;

}

}

return;

}

int main()

{

char str[50];

printf("Enter a statement: "); gets(str);

printf("\n"); printf("Symbol Table\n"); parse(str);

return (0);

}

**Output:**

Text

Description automatically generated with medium confidenceText

Description automatically generated

# Experiment 8

## Aim:

WAP to create lexeme and token table for:

(a) print(“Hello”);

## Code:

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

int isDelimiter(char ch)

{

if (ch == ' ' || ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == ',' || ch == ';' || ch == '>' || ch == '<' || ch == '=' || ch == '(' || ch == ')' || ch == '[' || ch == ']' || ch == '{' || ch == '}') return 1;

return 0;

}

int isOperator(char ch)

{

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

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

ch == '=' || ch == '{' || ch == '}' || ch == '(' || ch == ')') return 1;

return 0;

}

int validIdentifier(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' || isDelimiter(str[0]) == 1) return 0;

return 1;

}

int isKeyword(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 1;

return 0;

}

int isInteger(char \*str)

{

int i, len = strlen(str);

if (len == 0) return 0;

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 0;

}

return 1;

}

int isRealNumber(char \*str)

{

int i, len = strlen(str); int hasDecimal = 0;

if (len == 0) return 0;

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 0;

if (str[i] == '.') hasDecimal = 1;

}

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 parse(char \*str)

{

int left = 0, right = 0; int len = strlen(str);

while (right <= len && left <= right)

{

if (isDelimiter(str[right]) == 0) right++;

if (isDelimiter(str[right]) == 1 && left == right)

{

if (isOperator(str[right]) == 1)

{

printf("%c OPERATOR\n", str[right]);

}

right++; left = right;

}

else if (isDelimiter(str[right]) == 1 && left != right || (right == len && left != right))

{

char \*subStr = subString(str, left, right - 1); if (isKeyword(subStr) == 1)

{

printf("%s KEYWORD\n", subStr);

}

else if (isInteger(subStr) == 1)

{

printf("%s LITERAL (Integer)\n", subStr);

}

else if (isRealNumber(subStr) == 1)

{

printf("%s LITERAL (Real Number)\n", subStr);

}

else if (validIdentifier(subStr) == 1 && isDelimiter(str[right - 1]) == 0)

{

printf("%s IDENTIFIER\n", subStr);

}

left = right;

}

}

return;

}

int main()

{

char str[50];

printf("Enter a statement: ");

gets(str);

printf("\n");

printf("Lexeme Token\n");

parse(str);

return (0);

}

**Output:**

Text

Description automatically generated with low confidence

# Experiment 20

## Aim:

WAP to check whether given grammar is operator grammar or not.

## Code:

def main():

n = int(input('Enter number of productions: '))

grammar = []

operator\_grammar = True

for i in range(n):

production = input(f'Enter production {i+1}: ')

grammar.append(production)

for production in grammar:

if '#' in production[3:]:

operator\_grammar = False

for symbol in range(len(production[3:])-1):

if production[3:][symbol].isupper() and production[3:][symbol+1].isupper():

operator\_grammar = False

if operator\_grammar:

print('Given grammar is operator grammar.')

else:

print('Given grammar is not operator grammar.')

if \_\_name\_\_== '\_\_main\_\_':

main()

**Output:**

Text

Description automatically generated