**EXPERIMENT** 1

Implementation of Lexical Analyzer

**Aim**: Write a program in C/C++ to implement a lexical analyzer.

**Algorithm**:

1. Start

2. Get the input expression from the user.

3. Store the keywords and operators.

4. Perform analysis of the tokens based on the ASCII values.

5.

ASCII Range TOKEN TYPE

97-122 Keyword else identifier

48-57 Constant else operator

Greater than 12 Symbol

6. Print the token types.

7. Stop

**Program** (lexi.c):

#include <stdio.h>

#include <conio.h>

#include <ctype.h>

#include <string.h>

using namespace std;

int main()

{

char key[11][10] =

{"for", "while", "do", "then", "else", "break", "switch", "case", "if",

"continue"};

char oper[13] =

{'+', '-', '\*', '/', '%', '&', '<', '>', '=', ';', ':', '!'};

char a[20], b[20], c[20];

int i, j, l, m, k, flag;

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

gets(a);

i = 0;

while (a[i])

{

flag = 0;

j = 0;

l = 0;

b[0] = '\0';

if ((toascii(a[i] >= 97)) && (toascii(a[i] <= 122)))

{

if ((toascii(a[i + 1] >= 97)) && (toascii(a[i + 1] <= 122)))

{

while ((toascii(a[i] >= 97)) && (toascii(a[i] <= 122)))

{

b[j] = a[i];

j++;

i++;

}

b[j] = '\0';

}

else

{

b[j] = a[i];

i++;

b[j + 1] = '\0';

}

for (k = 0; k <= 9; k++)

{

if (strcmp(b, key[k]) == 0)

{

flag = 1;

break;

}

}

if (flag == 1)

printf("\n %s is the keyword", b);

else

printf("\n %s is the identifier", b);

}

else if ((toascii(a[i] >= 48)) && (toascii(a[i] <= 57)))

{

if ((toascii(a[i + 1] >= 48)) && (toascii(a[i + 1] <= 57)))

{

while ((toascii(a[i] >= 48)) && (toascii(a[i] <= 57)))

{

c[l] = a[i];

l++;

i++;

}

}

else

{

c[l] = a[i];

i++;

l++;

}

c[l] = '\0';

printf("\n %s is the constant", c);

} // second ifelse

else

{

for (m = 0; m < 13; m++)

{

if (a[i] == oper[m])

{

printf("\n %c is the operator", a[i]);

break;

}

}

if (m >= 13)

printf("\n %c is the symbol", a[i]);

i++;

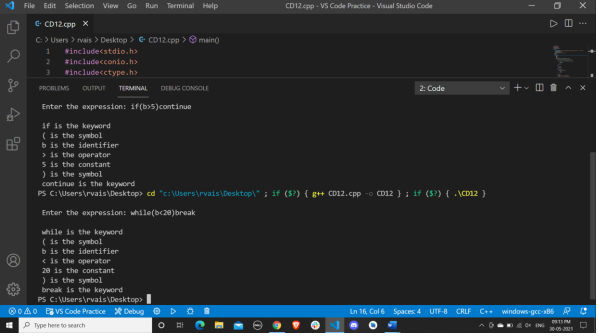
} // last else

} // while

return 0;

}

**OUTPUT**:

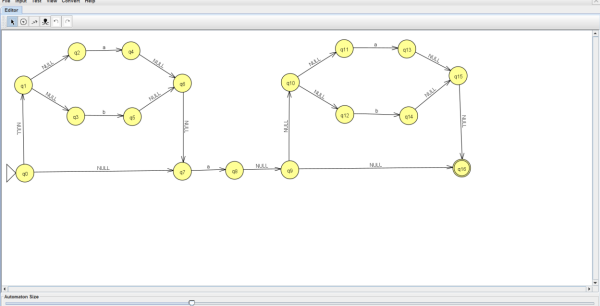


**Result**: The Program Executed successfully

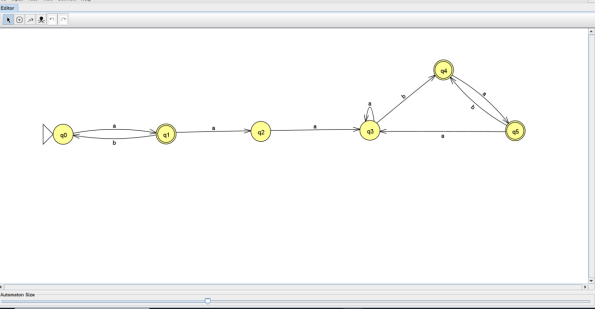
**EXPERIMENT 3**

Regular Expression to NFA to DFA

**Aim:** To convert the given Regular expression to DFA by using JFLAP. Ques: (a+b) \*a(a+b) \* NFA for the given expression is:



**OUTPUT**: - DFA for the given expression is:



**Result**: We converted the given Regular expression to DFA.

**EXPERIMENT 4**

Computation of FIRST in a grammar

**Aim**: Write a program in C/C++ to find the FIRST set for a given set of production rule of a grammar.

**Algorithm**: Procedure First

1. Input the number of production N.

2. Input all the production rule PArray

3. Repeat steps a, b, c until process all input production rule i.e. PArray[N] a. If Xi ≠ Xi+1 then i. Print Result array of Xi which contain FIRST(Xi) b. If first element of Xi of PArray is Terminal or ε Then i. Add Result = Result U first element c. If first element of Xi of PArray is Non-Terminal Then i. searchFirst(i, PArray, N)

4. End Loop

5. If N (last production) then a. Print Result array of Xi which contain FIRST(Xi)

6. End

Procedure searchFirst(i, PArray, N)

1. Repeat steps Loop j=i+1 to N a. If first element of Xj of PArray is Non-Terminal Then i. searchFirst(j, of PArray, N) b. If first element of Xj of PArray is Terminal or ε Then i. Add Result = Result U first element ii. Flag=0

2. End Loop

3. If Flag = 0 Then a. Print Result array of Xj which contain FIRST(Xj)

4. End

**Program**:

#include <iostream>

#include <conio.h>

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

using namespace std;

void searchFirst(int n, int i, char pl[], char r[], char result[], int k)

{

int j, flag;

for (j = i + 1; j < n; j++)

{

if (r[i] == pl[j])

{

if (isupper(r[j]))

{

searchFirst(n, j, pl, r, result, k);

}

if (islower(r[j]) || r[j] == '+' || r[j] == '\*' || r[j] == ')' || r[j] == '(')

{

result[k++] = r[j];

result[k++] = ',';

flag = 0;

}

}

}

if (flag == 0)

{

for (j = 0; j < k - 1; j++)

cout << result[j];

}

}

int main()

{

char pr[10][10], pl[10], r[10], prev, result[10];

int i, n, k, j;

cout << "\nHow many production rule : ";

cin >> n;

if (n == 0)

exit(0);

for (i = 0; i < n; i++)

{

cout << "\nInput left part of production rules : ";

cin >> pl[i];

cout << "\nInput right part of production rules : ";

cin >> pr[i];

r[i] = pr[i][0];

}

cout << "\nProduction Rules are : \n";

for (i = 0; i < n; i++)

{

cout << pl[i] << "->" << pr[i] << "\n"; //<<";"<<r[i]<<"\n";

}

cout << "\n----O U T P U T---\n\n";

prev = pl[0];

k = 0;

for (i = 0; i < n; i++)

{

if (prev != pl[i])

{

cout << "\nFIRST(" << prev << ")={";

for (j = 0; j < k - 1; j++)

cout << result[j];

cout << "}";

k = 0;

prev = pl[i];

// cout<<"\n3";

}

if (prev == pl[i])

{

if (islower(r[i]) || r[i] == '+' || r[i] == '\*' || r[i] == ')' || r[i] == '(')

{

result[k++] = r[i];

result[k++] = ',';

}

if (isupper(r[i]))

{

cout << "\nFIRST(" << prev << ")={";

searchFirst(n, i, pl, r, result, k);

cout << "}";

k = 0;

prev = pl[i + 1];

}

}

}

if (i == n)

{

cout << "\nFIRST(" << prev << ")={";

for (j = 0; j < k - 1; j++)

cout << result[j];

cout << "}";

k = 0;

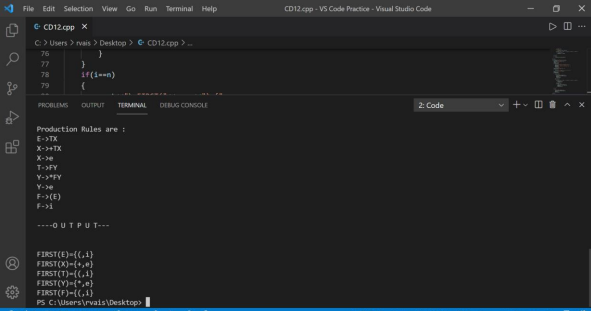
prev = pl[i];

}

return 0;

}

**OUtput**

****

**Result: The Program Executed successfully.**