# 

**Department of Information Technology**

**KRISHNA ENGINEERING COLLEGE**

95, Loni Road, Mohan Nagar, Ghaziabad (Uttar Pradesh), Pin-2010

# **COMPILER DESIGN LAB**

**(RCS-652)**

**ROHIT KUMAR**

**(1816110170)**

**3C**

**Semester - V**

**Program 1**

**AIM: WAP to check whether the entered string is accepted or not for a given grammar.**

**PROGRAM:**

#include <iostream>

#include <string.h>

#include <stdlib.h>

using namespace std;

char a[100];

int n, i;

int main()

{

    cout << "\nenter string: ";

    cin >> a;

    n = strlen(a);

    if (a[0] == 'a' && (a[n - 1] == 'a' || a[n - 1] == 'b') && a[n - 2] == 'c')

    {

        for (i = 1; i < n - 2; i++)

        {

            if (a[i] != 'b')

            {

                cout << "string is not accepted";

                exit(0);

            }

        }

        cout << "string is accepted";

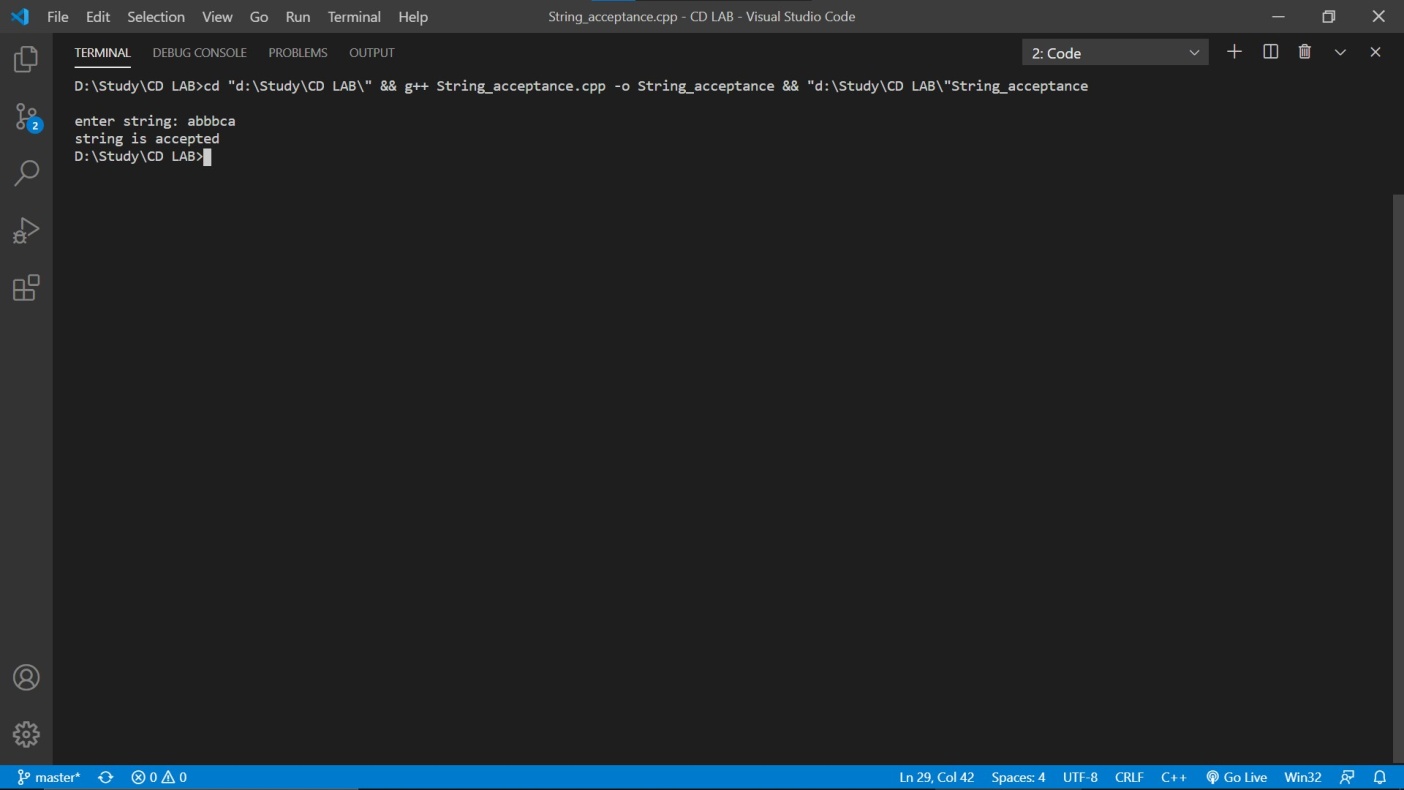
    }

    else

        cout << "string is not accepted";

}

**Output:**



**Program 2**

**AIM: WAP to convert infix expression to postfix expression.**

**PROGRAM:**

*//Program to convert infix expression to postfix expression*

#include<iostream>

#include<stack>

using namespace std;

*//FUNCTION TO CHECK AND RETURN THE PRECEDENCE OF THE OPERATOR*

int precedence(char *c*)

{

    if(*c* == '^')

    {

        return 3;

    }

    else if(*c* == '\*' || *c* == '/')

    {

        return 2;

    }

    else if(*c* == '+' || *c* == '-')

    {

        return 1;

    }

    else

    {

        return -1;

    }

}

void infixToPost(string *s*)

{

    stack<char> stack;

    stack.push('{');

    int len = *s*.length();

    string str2;

    for(int i = 0; i < len; i++)

    {

        if((*s*[i] >= 'a' and *s*[i] <= 'z')or(*s*[i] >= 'A' and *s*[i] <= 'Z'))

        {

            str2+=*s*[i];

        }

        else if(*s*[i] == '(')

        {

        stack.push('(');

        }

*//THIS DEALS WITH THE ENCOUNTER OF BRACKETS IN THE EXPRESSION*

        else if(*s*[i] == ')')

        {

            while(stack.top() != '{' and stack.top() != '(')

            {

                char c = stack.top();

                stack.pop();

                str2 += c;

            }

            if(stack.top() == '(')

            {

                char c = stack.top();

                stack.pop();

            }

        }

*//IF THE PRECEDENCE OF THE NEW OPERATOR SCANNED IS LESS THAN THAT OF PRESENT IN THE STACK*

*//THEN THIS CONDITION INVOKES*

        else{

            while(stack.top() != '{' and precedence(*s*[i]) <= precedence(stack.top()))

            {

                char oper = stack.top();

                stack.pop();

                str2 += oper;

            }

            stack.push(*s*[i]);

        }

    }

*//DEALING WITH REMAINING PART OF THE OPERATORS IN STACK*

    while(stack.top() != '{')

    {

        char c = stack.top();

        stack.pop();

        str2 += c;

    }

    cout<<"The postfix expression is: "<<str2 << endl;

}

int main()

{

    string expression;

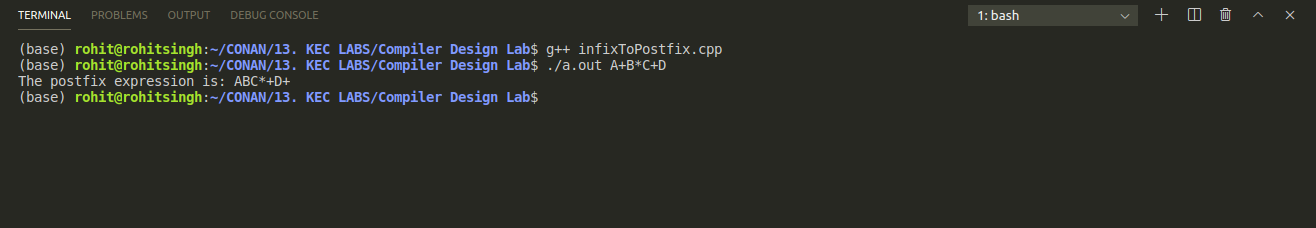
    cin>>expression;

    infixToPost(expression);

    return 0;

}

OUTPUT:



**Program 3**

**AIM: WAP to convert infix expression to prefix expression.**

**PROGRAM:**

*//Program to convert infix expression to prefix expression*

#include<iostream>

#include<algorithm>

#include<stack>

using namespace std;

*//FUNCTION TO CHECK AND RETURN THE PRECEDENCE OF THE OPERATOR*

int precedence(char *c*)

{

    if(*c* == '^')

    {

        return 3;

    }

    else if(*c* == '\*' || *c* == '/')

    {

        return 2;

    }

    else if(*c* == '+' || *c* == '-')

    {

        return 1;

    }

    else

    {

        return -1;

    }

}

void infixToPre(string *s*)

{

    stack<char> stack;

    stack.push('{');

    int len = *s*.length();

    string str2;

    for(int i = 0; i < len; i++)

    {

        if((*s*[i] >= 'a' and *s*[i] <= 'z')or(*s*[i] >= 'A' and *s*[i] <= 'Z'))

        {

            str2+=*s*[i];

        }

        else if(*s*[i] == '(')

        {

        stack.push('(');

        }

*//THIS DEALS WITH THE ENCOUNTER OF BRACKETS IN THE EXPRESSION*

        else if(*s*[i] == ')')

        {

            while(stack.top() != '{' and stack.top() != '(')

            {

                char c = stack.top();

                stack.pop();

                str2 += c;

            }

            if(stack.top() == '(')

            {

                char c = stack.top();

                stack.pop();

            }

        }

*//IF THE PRECEDENCE OF THE NEW OPERATOR SCANNED IS LESS THAN THAT OF PRESENT IN THE STACK*

*//THEN THIS CONDITION INVOKES*

        else{

            while(stack.top() != '{' and precedence(*s*[i]) <= precedence(stack.top()))

            {

                char oper = stack.top();

                stack.pop();

                str2 += oper;

            }

            stack.push(*s*[i]);

        }

    }

*//DEALING WITH REMAINING PART OF THE OPERATORS IN STACK*

    while(stack.top() != '{')

    {

        char c = stack.top();

        stack.pop();

        if(c=='(' or c==')') continue;

        else str2 += c;

    }

    reverse(str2.begin(),str2.end());

    cout<<"The prefix expression is: "<<str2 << endl;

}

int main()

{

    string expression,exp;

    cin>>expression;

    for(int i=expression.length();i>0;i--)

    {

        if(expression[i]==')')

        {

            exp+='(';

        }

        else if (expression[i]=='(')

        {

            exp+=')';

        }

        else

        {

            exp+=expression[i];

        }

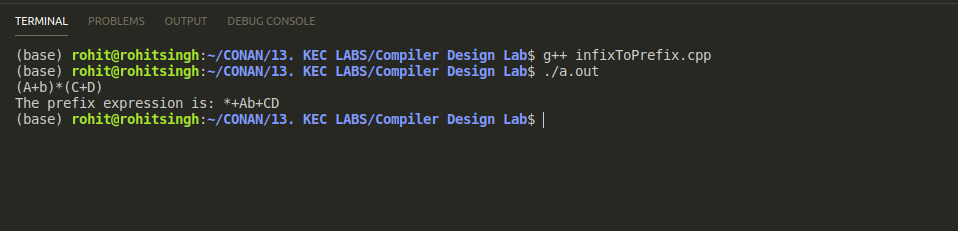
    }

    infixToPre(exp);

    return 0;

}

Output:



**Program 4**

**AIM: WAP to find the no. of tokens and list them according to their category in an expression (given/entered)**

**PROGRAM:**

#include<iostream>

#include<algorithm>

#include<unordered\_set>

using namespace std;

int main(){

    string s;

    getline(cin, s);

    int n=s.size();

    unordered\_set<int> st;

*// For +*

    st.insert(43);

*//For -*

    st.insert(45);

*//For \**

    st.insert(42);

*//For /*

    st.insert(47);

*//For =*

    st.insert(61);*//=*

*//Initialisation*

    int opr=0, cons=0, var=0, i=0;

    while(i<n){

        if (int(s[i])>=48 && int(s[i])<=57){

            cons++;

            while(i<n && int(s[i])>=48 && int(s[i])<=57)

                i++;

        }

        else if ((int(s[i])>=65 && int(s[i])<=90) || (int(s[i])>=97 && int(s[i])<=122)){

            var++;

            while(i<n && ((int(s[i])>=65 && int(s[i])<=90) || (int(s[i])>=97 && int(s[i])<=122) || (int(s[i])>=48 && int(s[i])<=57))){

                i++;

            }

        }

        else if(st.find(int(s[i]))!=st.end()){

            opr++;

            i++;

        }

        else if (int(s[i])==32){

            i++;

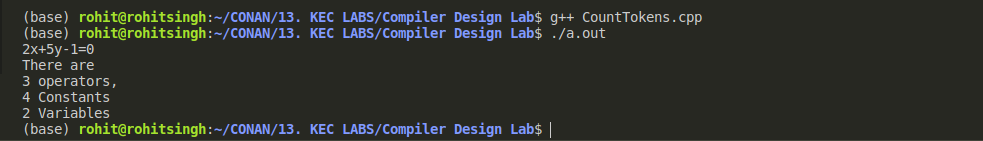
        }

    }

    cout<<"There are "<<endl<<opr<<" operators, "<<endl<<cons<<" Constants"<<endl<<var<<" Variables"<<endl;

}

Output:



**Program 5**

**AIM: WAP to construct an NFA from a regular expression (given) and display the transition table of NFA constructed.**

**PROGRAM:**

#include<bits/stdc++.h>

using namespace std;

int main()

{

  char s[10];

  int n,init=0,fin=1;

  cout<<"enter R.E\n";

  cin>>s;

  n=strlen(s);

  for(int i=0;i<n;i++)

  {

    if(s[i]=='\*')

    fin+=2;

    if(s[i]=='.')

    fin+=1;

    if(s[i]=='/')

    fin+=4;

   }

  char c=238;

  int i=0;

  int ch;

  if(s[0]>=97&&s[0]<=122)

  ch=1;

  if(s[0]=='('&&s[4]==')')

  ch=2;

  switch(ch)

  {

   case 1:

   if(s[i+1]=='/')

    {

     if(s[i+2]>=97 && s[i+2]<=122)

      {

        cout<<"\n"<<init+2<<"--"<<s[i]<<"-->"<<init+3;

        cout<<"\n"<<init+4<<"--"<<s[i+2]<<"-->"<<init+5;

        goto pt1;

      }

     }

   case 2:

   if(s[i+1]>=97 && s[i+1]<=122)

   if(s[i+2]=='/')

  {

    if(s[i+3]>=97 && s[i+3]<=122)

    {

       cout<<"\n"<<init+2<<"--"<<s[i+1]<<"-->"<<init+3;

       cout<<"\n"<<init+4<<"--"<<s[i+3]<<"-->"<<init+5;

       if(s[i+5]=='\*')

        {

          goto pt;

        }

     else

     goto pt1;

     }

  }

 }

 pt:

 cout<<"\n"<<init<<"--"<<c<<"-->"<<init+1;

 cout<<"\n"<<init<<"--"<<c<<"-->"<<fin;

 pt1:

 cout<<"\n"<<init+1<<"--"<<c<<"-->"<<init+2;

 cout<<"\n"<<init+1<<"--"<<c<<"-->"<<init+4;

 cout<<"\n"<<init+3<<"--"<<c<<"-->"<<init+6;

 cout<<"\n"<<init+5<<"--"<<c<<"-->"<<init+6;

 cout<<"\n"<<init+6<<"--"<<c<<"-->"<<init+1;

 cout<<"\n"<<init+6<<"--"<<c<<"-->"<<fin;

 return 0;

}

Output:



**Program 6**

**AIM: WAP to compute LEADING and TRAILING sets of a grammar(given).**

LEADING:

#include <iostream>

using namespace std;

bool Termin(char *c*)

{

  if (!isupper(*c*) && *c* != '\0' && *c* != '|')

    return true;

  else

    return false;

}

int Prod(char *NT*, char \*\**strings*, int *currPos*, int *n*)

{

  for (int i = 0; i < *n*; ++i)

  {

    if (i == *currPos*)

      continue;

    if (*strings*[i][0] == *NT*)

      return i;

  }

}

void StringChk(char \**string*, char \*\**strings*, int *currPos*, int *n*)

{

  int i = 2;

  for (; i < 50; ++i)

  {

    if (Termin(*string*[i]))

    {

      cout << *string*[i] << " ";

      while (*string*[i] != '|' && *string*[i] != '\0')

        i++;

    }

    else if (*string*[i] == '|' || *string*[i] == '\0')

    {

      int productionNo = Prod(*string*[i - 1], *strings*, *currPos*, *n*);

      StringChk(*strings*[productionNo], *strings*, productionNo, *n*);

    }

    if (*string*[i] == '\0')

      break;

  }

}

int main()

{

  cout << "Enter the number of non-terminals: ";

  int n = 0;

  cin >> n;

  cout << "Enter the productions: " << endl;

  char \*\*strings = new char \*[n];

  for (int i = 0; i < n; ++i)

  {

    strings[i] = new char[50];

    cin >> strings[i];

  }

  cout << endl

       << "Leadings are: " << endl;

  for (int i = 0; i < n; ++i)

  {

    cout << "{ ";

    StringChk(strings[i], strings, i, n);

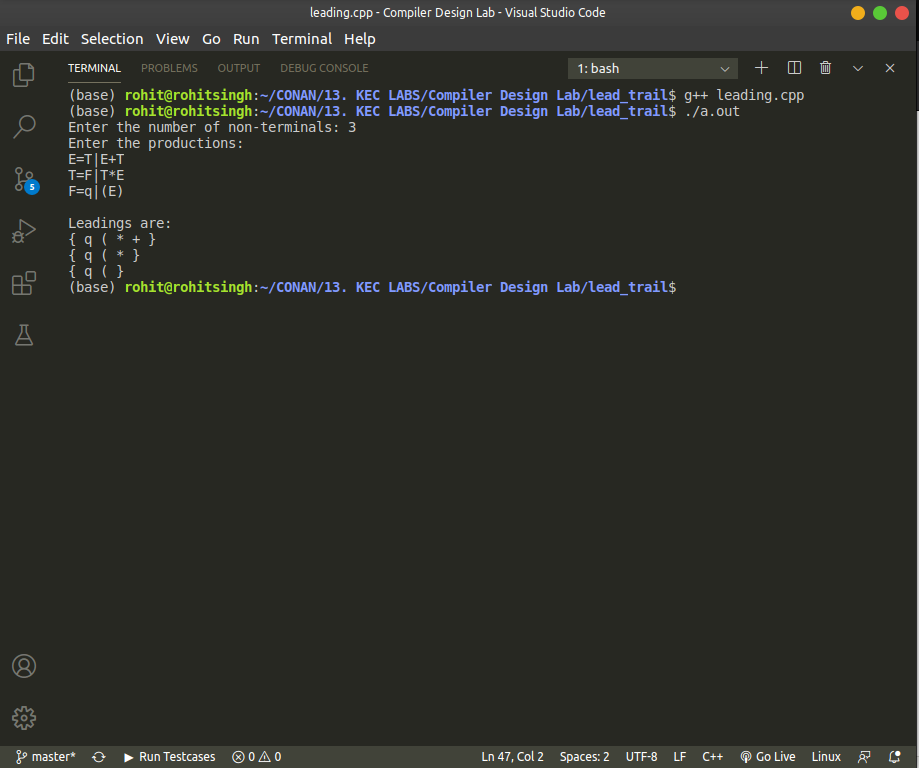
    cout << "}" << endl;

  }

  return 0;

}

OUTPUT:



TRAILING

#include <iostream>

using namespace std;

bool Termin(char *c*)

{

    if (!isupper(*c*) && *c* != '|' && *c* != '=')

        return true;

    else

        return false;

}

int Prod(char *NT*, char \*\**strings*, int *currPos*, int *n*)

{

    for (int i = 0; i < *n*; ++i)

    {

        if (i == *currPos*)

            continue;

        if (*strings*[i][0] == *NT*)

            return i;

    }

}

void StringChk(char \**string*, char \*\**strings*, int *currPos*, int *n*)

{

    int i = 49;

    while (*string*[i] != '\0')

        i--;

    i = i - 1;

    for (; i >= 1; --i)

    {

        if (Termin(*string*[i]))

        {

            cout << *string*[i] << " ";

            while (*string*[i] != '|' && *string*[i] != '=')

                i--;

        }

        else if (*string*[i] == '|' || *string*[i] == '=')

        {

            int prodNo = Prod(*string*[i + 1], *strings*, *currPos*, *n*);

            StringChk(*strings*[prodNo], *strings*, prodNo, *n*);

        }

        if (*string*[i] == '=')

            break;

    }

}

int main()

{

    cout << "Enter the number of non-terminals: ";

    int n = 0;

    cin >> n;

    cout << "Enter the productions: " << endl;

    char \*\*strings = new char \*[n];

    for (int i = 0; i < n; ++i)

    {

        strings[i] = new char[50];

        cin >> strings[i];

    }

    cout << endl

         << "Trailings are: " << endl;

    for (int i = 0; i < n; ++i)

    {

        cout << "{ ";

        StringChk(strings[i], strings, i, n);

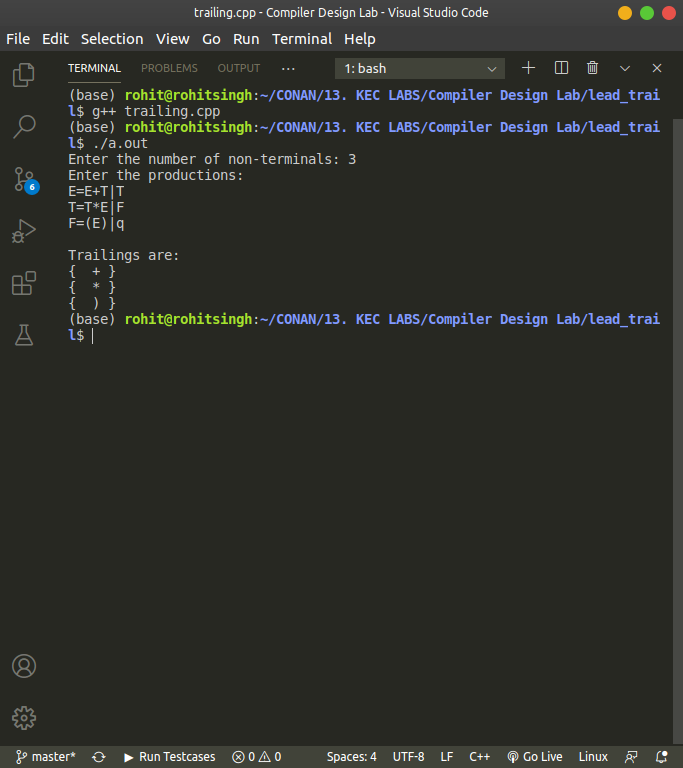
        cout << "}" << endl;

    }

    return 0;

}

OUTPUT:



**Program 8**

**AIM: WAP to calculate FIRST and FOLLOW.**

**PROGRAM:**

#include<stdio.h>

#include<math.h>

#include<string.h>

#include<ctype.h>

#include<stdlib.h>

int n,m=0,p,i=0,j=0;

char a[10][10],f[10];

void follow(char *c*);

void first(char *c*);

int main()

    {

    int i,z;

    char c,ch;

    printf("Enter the no of prooductions:\n");

    scanf("%d",&n);

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

    for(i=0;i<n;i++)

    scanf("%s%c",a[i],&ch);

    do

    {

        m=0;

        printf("Enter the elemets whose fisrt & follow is to be found:");

        scanf("%c",&c);

        first(c);

        printf("First(%c)={",c);

        for(i=0;i<m;i++)

        printf("%c",f[i]);

        printf("}\n");

        strcpy(f," ");

        m=0;

        follow(c);

        printf("Follow(%c)={",c);

        for(i=0;i<m;i++)

        printf("%c",f[i]);

        printf("}\n");

        printf("Continue(0/1)?");

        scanf("%d%c",&z,&ch);

    }while(z==1);

    return(0);

    }

    void first(char *c*)

    {

        int k;

        if(!isupper(*c*))

        f[m++]=*c*;

        for(k=0;k<n;k++)

        {

            if(a[k][0]==*c*)

            {

                if(a[k][2]=='$')

                follow(a[k][0]);

                else if(islower(a[k][2]))

                f[m++]=a[k][2];

                else first(a[k][2]);

            }

        }

    }

    void follow(char *c*)

        {

        if(a[0][0]==*c*)

        f[m++]='$';

        for(i=0;i<n;i++)

            {

            for(j=2;j<strlen(a[i]);j++)

                {

                if(a[i][j]==*c*)

                {

                    if(a[i][j+1]!='\0')

                    first(a[i][j+1]);

                    if(a[i][j+1]=='\0' && *c*!=a[i][0])

                    follow(a[i][0]);

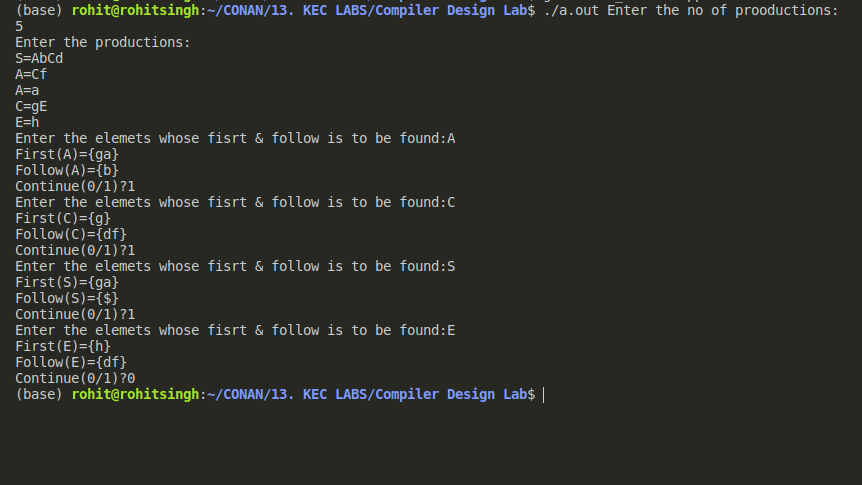
                }

                }

            }

        }

OUTPUT:



**Program 9**

**AIM: WAP in C to check whether the Grammar is Left-recursive and remove left recursion.**

**PROGRAM:**

#include<iostream>

#include<stdio.h>

#include<conio.h>

#include<string.h>

using namespace std;

struct production

{

            char l;

            char r[10];

            int rear;

};

struct production prod[20],pr\_new[20];

int p=0,b=0,d,f,q,n,flag=0;

char terminal[20],nonterm[20],alpha[10];

char x,epsilon='^';

int main()

{

       cout<<"Enter the number of terminals: ";

    cin>>d;

    cout<<"Enter the terminal symbols for your production: ";

    for(int k=0;k<d;k++)

    {

      cin>>terminal[k];

    }

    cout<<"\nEnter the number of non-terminals: ";

    cin>>f;

    cout<<"Enter the non-terminal symbols for your production: ";

    for(int k=0;k<f;k++)

    {

      cin>>nonterm[k];

    }

    cout<<"\nEnter the number of Special characters(except non-terminals): ";

    cin>>q;

    cout<<"Enter the special characters for your production: ";

    for(int k=0;k<q;k++)

    {

      cin>>alpha[k];

    }

    cout<<"\nEnter the number of productions: ";

    cin>>n;

    for(int k=0;k<=n-1;k++)

    {

      cout<<"Enter the "<< k+1<<" production: ";

      cin>>prod[k].l;

      cout<<"->";

      cin>>prod[k].r;

      prod[k].rear=strlen(prod[k].r);

    }

    for(int m=0;m<f;m++)

    {

      x=nonterm[m];

      for(int j=0;j<n;j++)

      {

               if((prod[j].l==x)&&(prod[j].r[0]==prod[j].l))

                  flag=1;

      }

      for(int i=0;i<n;i++)

      {

               if((prod[i].l==x)&&(prod[i].r[0]!=x)&&(flag==1))

               {

                 int c;

                           pr\_new[b].l=x;

                           for(c=0;c<prod[i].rear;c++)

                           pr\_new[b].r[c]=prod[i].r[c];

                           pr\_new[b++].r[c]=alpha[p];

               }

               else if((prod[i].l==x)&&(prod[i].r[0]==x)&&(flag==1))

               {

                 int a;

                          pr\_new[b].l=alpha[p];

                          for(a=0;a<=prod[i].rear-2;a++)

                           pr\_new[b].r[a]=prod[i].r[a+1];

                          pr\_new[b++].r[a]=alpha[p];

                          pr\_new[b].l=alpha[p];

                          pr\_new[b++].r[0]=epsilon;

                }

                else if((prod[i].l==x)&&(prod[i].r[0]!=x)&&(flag==0))

                {

                           pr\_new[b].l=prod[i].l;

                           strcpy(pr\_new[b].r,prod[i].r);

                           b++;

                }

            }

            flag=0;

            p++;

  }

    cout<<"\n\n\*\*\*\*\*\*\* AFTER REMOVING LEFT RECURSION \*\*\*\*\*\*\*"<<endl;

    for(int s=0;s<=b-1;s++)

            {

                        cout<<"Production "<<s+1<<" is: ";

                        cout<<pr\_new[s].l;

                        cout<<"->";

                        cout<<pr\_new[s].r;

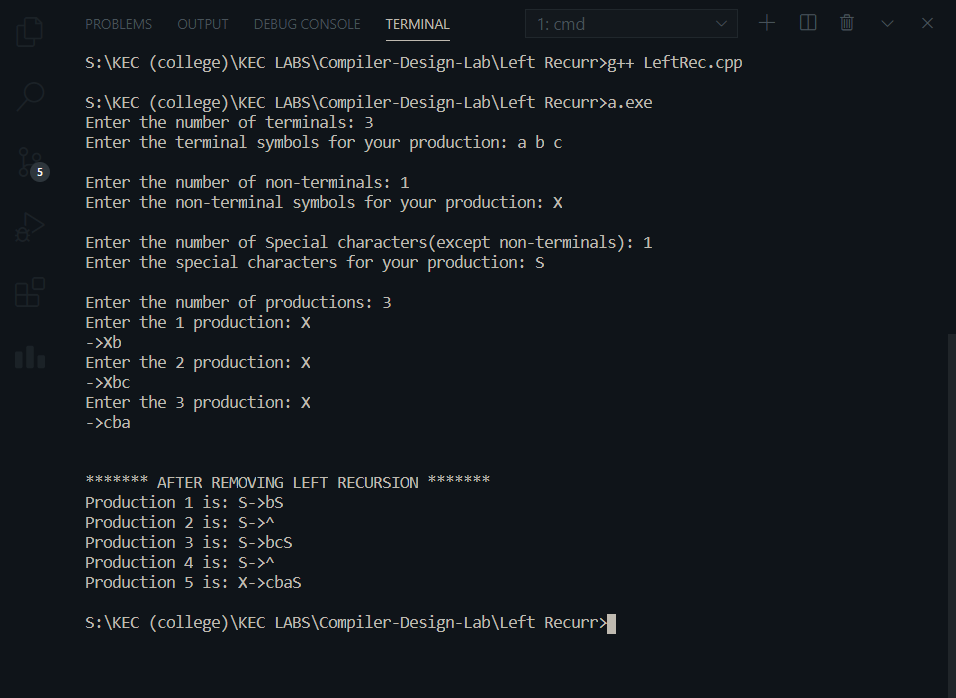
                        cout<<endl;

            }

    return 0;

}

Output:



**Prgramm-10**

**AIM: WAP in C to draw a SLR parsing table for a given grammar**

**PROGRAM:**

#include <iostream>

#include <string.h>

#include <stdlib.h>

#include <stdio.h>

using namespace std;

char terminals[100] = {};

int no\_t;

char non\_terminals[100] = {};

int no\_nt;

char goto\_table[100][100];

char reduce[20][20];

char follow[20][20];

char fo\_co[20][20];

char first[20][20];

struct state

{

    int prod\_count;

    char prod[100][100] = {{}};

};

void add\_dots(struct state \*I)

{

    for (int i = 0; i < I->prod\_count; i++)

    {

        for (int j = 99; j > 3; j--)

            I->prod[i][j] = I->prod[i][j - 1];

        I->prod[i][3] = '.';

    }

}

void augument(struct state \*S, struct state \*I)

{

    if (I->prod[0][0] == 'S')

        strcpy(S->prod[0], "Z->.S");

    else

    {

        strcpy(S->prod[0], "S->.");

        S->prod[0][4] = I->prod[0][0];

    }

    S->prod\_count++;

}

void get\_prods(struct state \*I)

{

    cout << "Enter the number of productions:\n";

    cin >> I->prod\_count;

    cout << "Enter the number of non terminals:" << endl;

    cin >> no\_nt;

    cout << "Enter the non terminals one by one:" << endl;

    for (int i = 0; i < no\_nt; i++)

        cin >> non\_terminals[i];

    cout << "Enter the number of terminals:" << endl;

    cin >> no\_t;

    cout << "Enter the terminals (single lettered) one by one:" << endl;

    for (int i = 0; i < no\_t; i++)

        cin >> terminals[i];

    cout << "Enter the productions one by one in form (S->ABc):\n";

    for (int i = 0; i < I->prod\_count; i++)

    {

        cin >> I->prod[i];

    }

}

bool is\_non\_terminal(char a)

{

    if (a >= 'A' && a <= 'Z')

        return true;

    else

        return false;

}

bool in\_state(struct state \*I, char \*a)

{

    for (int i = 0; i < I->prod\_count; i++)

    {

        if (!strcmp(I->prod[i], a))

            return true;

    }

    return false;

}

char char\_after\_dot(char a[100])

{

    char b;

    for (int i = 0; i < strlen(a); i++)

        if (a[i] == '.')

        {

            b = a[i + 1];

            return b;

        }

}

char \*move\_dot(char b[100], int len)

{

    char a[100] = {};

    strcpy(a, b);

    for (int i = 0; i < len; i++)

    {

        if (a[i] == '.')

        {

            swap(a[i], a[i + 1]);

            break;

        }

    }

    return &a[0];

}

bool same\_state(struct state \*I0, struct state \*I)

{

    if (I0->prod\_count != I->prod\_count)

        return false;

    for (int i = 0; i < I0->prod\_count; i++)

    {

        int flag = 0;

        for (int j = 0; j < I->prod\_count; j++)

            if (strcmp(I0->prod[i], I->prod[j]) == 0)

                flag = 1;

        if (flag == 0)

            return false;

    }

    return true;

}

void closure(struct state \*I, struct state \*I0)

{

    char a = {};

    for (int i = 0; i < I0->prod\_count; i++)

    {

        a = char\_after\_dot(I0->prod[i]);

        if (is\_non\_terminal(a))

        {

            for (int j = 0; j < I->prod\_count; j++)

            {

                if (I->prod[j][0] == a)

                {

                    if (!in\_state(I0, I->prod[j]))

                    {

                        strcpy(I0->prod[I0->prod\_count], I->prod[j]);

                        I0->prod\_count++;

                    }

                }

            }

        }

    }

}

void goto\_state(struct state \*I, struct state \*S, char a)

{

    int time = 1;

    for (int i = 0; i < I->prod\_count; i++)

    {

        if (char\_after\_dot(I->prod[i]) == a)

        {

            if (time == 1)

            {

                time++;

            }

            strcpy(S->prod[S->prod\_count], move\_dot(I->prod[i], strlen(I->prod[i])));

            S->prod\_count++;

        }

    }

}

void print\_prods(struct state \*I)

{

    for (int i = 0; i < I->prod\_count; i++)

        printf("%s\n", I->prod[i]);

    cout << endl;

}

bool in\_array(char a[20], char b)

{

    for (int i = 0; i < strlen(a); i++)

        if (a[i] == b)

            return true;

    return false;

}

char \*chars\_after\_dots(struct state \*I)

{

    char a[20] = {};

    for (int i = 0; i < I->prod\_count; i++)

    {

        if (!in\_array(a, char\_after\_dot(I->prod[i])))

        {

            a[strlen(a)] = char\_after\_dot(I->prod[i]);

        }

    }

    return &a[0];

}

void cleanup\_prods(struct state \*I)

{

    char a[100] = {};

    for (int i = 0; i < I->prod\_count; i++)

        strcpy(I->prod[i], a);

    I->prod\_count = 0;

}

int return\_index(char a)

{

    for (int i = 0; i < no\_t; i++)

        if (terminals[i] == a)

            return i;

    for (int i = 0; i < no\_nt; i++)

        if (non\_terminals[i] == a)

            return no\_t + i;

}

void print\_shift\_table(int state\_count)

{

    cout << endl

         << "\*\*\*\*\*\*\*\*Shift Actions\*\*\*\*\*\*\*\*\*" << endl

         << endl;

    cout << "\t";

    for (int i = 0; i < no\_t; i++)

        cout << terminals[i] << "\t";

    for (int i = 0; i < no\_nt; i++)

        cout << non\_terminals[i] << "\t";

    cout << endl;

    for (int i = 0; i < state\_count; i++)

    {

        int arr[no\_nt + no\_t] = {-1};

        for (int j = 0; j < state\_count; j++)

        {

            if (goto\_table[i][j] != '~')

            {

                arr[return\_index(goto\_table[i][j])] = j;

            }

        }

        cout << "I" << i << "\t";

        for (int j = 0; j < no\_nt + no\_t; j++)

        {

            if (i == 1 && j == no\_t - 1)

                cout << "ACC"

                     << "\t";

            if (arr[j] == -1 || arr[j] == 0)

                cout << "\t";

            else

            {

                if (j < no\_t)

                    cout << "S" << arr[j] << "\t";

                else

                    cout << arr[j] << "\t";

            }

        }

        cout << "\n";

    }

}

int get\_index(char c, char \*a)

{

    for (int i = 0; i < strlen(a); i++)

        if (a[i] == c)

            return i;

}

void add\_dot\_at\_end(struct state \*I)

{

    for (int i = 0; i < I->prod\_count; i++)

    {

        strcat(I->prod[i], ".");

    }

}

void add\_to\_first(int n, char b)

{

    for (int i = 0; i < strlen(first[n]); i++)

        if (first[n][i] == b)

            return;

    first[n][strlen(first[n])] = b;

}

void add\_to\_first(int m, int n)

{

    for (int i = 0; i < strlen(first[n]); i++)

    {

        int flag = 0;

        for (int j = 0; j < strlen(first[m]); j++)

        {

            if (first[n][i] == first[m][j])

                flag = 1;

        }

        if (flag == 0)

            add\_to\_first(m, first[n][i]);

    }

}

void add\_to\_follow(int n, char b)

{

    for (int i = 0; i < strlen(follow[n]); i++)

        if (follow[n][i] == b)

            return;

    follow[n][strlen(follow[n])] = b;

}

void add\_to\_follow(int m, int n)

{

    for (int i = 0; i < strlen(follow[n]); i++)

    {

        int flag = 0;

        for (int j = 0; j < strlen(follow[m]); j++)

        {

            if (follow[n][i] == follow[m][j])

                flag = 1;

        }

        if (flag == 0)

            add\_to\_follow(m, follow[n][i]);

    }

}

void add\_to\_follow\_first(int m, int n)

{

    for (int i = 0; i < strlen(first[n]); i++)

    {

        int flag = 0;

        for (int j = 0; j < strlen(follow[m]); j++)

        {

            if (first[n][i] == follow[m][j])

                flag = 1;

        }

        if (flag == 0)

            add\_to\_follow(m, first[n][i]);

    }

}

void find\_first(struct state \*I)

{

    for (int i = 0; i < no\_nt; i++)

    {

        for (int j = 0; j < I->prod\_count; j++)

        {

            if (I->prod[j][0] == non\_terminals[i])

            {

                if (!is\_non\_terminal(I->prod[j][3]))

                {

                    add\_to\_first(i, I->prod[j][3]);

                }

            }

        }

    }

}

void find\_follow(struct state \*I)

{

    for (int i = 0; i < no\_nt; i++)

    {

        for (int j = 0; j < I->prod\_count; j++)

        {

            for (int k = 3; k < strlen(I->prod[j]); k++)

            {

                if (I->prod[j][k] == non\_terminals[i])

                {

                    if (I->prod[j][k + 1] != '\0')

                    {

                        if (!is\_non\_terminal(I->prod[j][k + 1]))

                        {

                            add\_to\_follow(i, I->prod[j][k + 1]);

                        }

                    }

                }

            }

        }

    }

}

int get\_index(int \*arr, int n)

{

    for (int i = 0; i < no\_t; i++)

    {

        if (arr[i] == n)

            return i;

    }

    return -1;

}

void print\_reduce\_table(int state\_count, int \*no\_re, struct state \*temp1)

{

    cout << "\*\*\*\*\*\*\*\*\*\*Reduce actions\*\*\*\*\*\*\*\*\*\*" << endl

         << endl;

    cout << "\t";

    int arr[temp1->prod\_count][no\_t] = {-1};

    for (int i = 0; i < no\_t; i++)

    {

        cout << terminals[i] << "\t";

    }

    cout << endl;

    for (int i = 0; i < temp1->prod\_count; i++)

    {

        int n = no\_re[i];

        for (int j = 0; j < strlen(follow[return\_index(temp1->prod[i][0]) - no\_t]); j++)

        {

            for (int k = 0; k < no\_t; k++)

            {

                if (follow[return\_index(temp1->prod[i][0]) - no\_t][j] == terminals[k])

                    arr[i][k] = i + 1;

            }

        }

        cout << "I" << n << "\t";

        for (int j = 0; j < no\_t; j++)

        {

            if (arr[i][j] != -1 && arr[i][j] != 0 && arr[i][j] < state\_count)

                cout << "R" << arr[i][j] << "\t";

            else

                cout << "\t";

        }

        cout << endl;

    }

}

int main()

{

    struct state init;

    struct state temp;

    struct state temp1;

    int state\_count = 1;

    get\_prods(&init);

    temp = init;

    temp1 = temp;

    add\_dots(&init);

    for (int i = 0; i < 100; i++)

        for (int j = 0; j < 100; j++)

            goto\_table[i][j] = '~';

    struct state I[50];

    augument(&I[0], &init);

    closure(&init, &I[0]);

    cout << "\nI0:\n";

    print\_prods(&I[0]);

    char characters[20] = {};

    for (int i = 0; i < state\_count; i++)

    {

        char characters[20] = {};

        for (int z = 0; z < I[i].prod\_count; z++)

            if (!in\_array(characters, char\_after\_dot(I[i].prod[z])))

                characters[strlen(characters)] = char\_after\_dot(I[i].prod[z]);

        for (int j = 0; j < strlen(characters); j++)

        {

            goto\_state(&I[i], &I[state\_count], characters[j]);

            closure(&init, &I[state\_count]);

            int flag = 0;

            for (int k = 0; k < state\_count - 1; k++)

            {

                if (same\_state(&I[k], &I[state\_count]))

                {

                    cleanup\_prods(&I[state\_count]);

                    flag = 1;

                    cout << "I" << i << " on reading the symbol " << characters[j] << " goes to I" << k << ".\n";

                    goto\_table[i][k] = characters[j];

                    ;

                    break;

                }

            }

            if (flag == 0)

            {

                state\_count++;

                cout << "I" << i << " on reading the symbol " << characters[j] << " goes to I" << state\_count - 1 << ":\n";

                goto\_table[i][state\_count - 1] = characters[j];

                print\_prods(&I[state\_count - 1]);

            }

        }

    }

    int no\_re[temp.prod\_count] = {-1};

    terminals[no\_t] = '$';

    no\_t++;

    add\_dot\_at\_end(&temp1);

    for (int i = 0; i < state\_count; i++)

    {

        for (int j = 0; j < I[i].prod\_count; j++)

            for (int k = 0; k < temp1.prod\_count; k++)

                if (in\_state(&I[i], temp1.prod[k]))

                    no\_re[k] = i;

    }

    find\_first(&temp);

    for (int l = 0; l < no\_nt; l++)

    {

        for (int i = 0; i < temp.prod\_count; i++)

        {

            if (is\_non\_terminal(temp.prod[i][3]))

            {

                add\_to\_first(return\_index(temp.prod[i][0]) - no\_t, return\_index(temp.prod[i][3]) - no\_t);

            }

        }

    }

    find\_follow(&temp);

    add\_to\_follow(0, '$');

    for (int l = 0; l < no\_nt; l++)

    {

        for (int i = 0; i < temp.prod\_count; i++)

        {

            for (int k = 3; k < strlen(temp.prod[i]); k++)

            {

                if (temp.prod[i][k] == non\_terminals[l])

                {

                    if (is\_non\_terminal(temp.prod[i][k + 1]))

                    {

                        add\_to\_follow\_first(l, return\_index(temp.prod[i][k + 1]) - no\_t);

                    }

                    if (temp.prod[i][k + 1] == '\0')

                        add\_to\_follow(l, return\_index(temp.prod[i][0]) - no\_t);

                }

            }

        }

    }

    print\_shift\_table(state\_count);

    cout << endl

         << endl;

    print\_reduce\_table(state\_count, &no\_re[0], &temp1);

}

Output:





**Program 11**

**AIM: WAP in C to draw an operator precedence parsing table for the given grammar**

**PROGRAM:**

#include <iostream>

#include <string>

#include <deque>

using namespace std;

int n, n1, n2;

int getPosition(string *arr*[], string *q*, int *size*)

{

    for (int i = 0; i < *size*; i++)

    {

        if (*q* == *arr*[i])

            return i;

    }

    return -1;

}

int main()

{

    string prods[10], leads[10], trails[10], nonterms[10], terms[10];

    char op\_table[20][20] = {};

    cout << "Enter the number of productions : ";

    cin >> n;

    cin.ignore();

    cout << "Enter the productions" << endl;

    for (int i = 0; i < n; i++)

    {

        getline(cin, prods[i]);

    }

    cout << "Enter the number of Terminals : ";

    cin >> n2;

    cin.ignore();

    cout << "Enter the Terminals" << endl;

    for (int i = 0; i < n2; i++)

    {

        cin >> terms[i];

    }

    terms[n2] = "$";

    n2++;

    cout << "Enter the number of Non-Terminals : ";

    cin >> n1;

    cin.ignore();

    for (int i = 0; i < n1; i++)

    {

        cout << "Enter Non-Terminal : ";

        getline(cin, nonterms[i]);

        cout << "Enter Leads of " << nonterms[i] << " : ";

        getline(cin, leads[i]);

        cout << "Enter Trails of " << nonterms[i] << " : ";

        getline(cin, trails[i]);

    }

    cout << "Enter the Rules (exit to stop)" << endl;

    string rule = "";

    while (rule != "exit")

    {

        getline(cin, rule);

        if (rule[0] == '1')

        {

            int row = getPosition(terms, rule.substr(2, 1), n2);

            int column = getPosition(terms, rule.substr(4, 1), n2);

            op\_table[row][column] = '=';

        }

        if (rule[0] == '2')

        {

            int ntp = getPosition(nonterms, rule.substr(4, 1), n1);

            int row = getPosition(terms, rule.substr(2, 1), n2);

            for (int j = 0; j < leads[ntp].size(); j++)

            {

                int col = getPosition(terms, leads[ntp].substr(j, 1), n2);

                op\_table[row][col] = '<';

            }

        }

        if (rule[0] == '3')

        {

            int col = getPosition(terms, rule.substr(4, 1), n2);

            int ntp = getPosition(nonterms, rule.substr(2, 1), n1);

            for (int j = 0; j < trails[ntp].size(); j++)

            {

                int row = getPosition(terms, trails[ntp].substr(j, 1), n2);

                op\_table[row][col] = '>';

            }

        }

    }

    for (int j = 0; j < leads[0].size(); j++)

    {

        int col = getPosition(terms, leads[0].substr(j, 1), n2);

        op\_table[n2 - 1][col] = '<';

    }

    for (int j = 0; j < trails[0].size(); j++)

    {

        int row = getPosition(terms, trails[0].substr(j, 1), n2);

        op\_table[row][n2 - 1] = '>';

    }

    cout << endl;

    cout << "Grammar" << endl;

    for (int i = 0; i < n; i++)

    {

        cout << prods[i] << endl;

    }

*//Display Table*

    for (int j = 0; j < n2; j++)

        cout << "\t" << terms[j];

    cout << endl;

    for (int i = 0; i < n2; i++)

    {

        cout << terms[i] << "\t";

        for (int j = 0; j < n2; j++)

        {

            cout << op\_table[i][j] << "\t";

        }

        cout << endl;

    }

*//Parsing String*

    char c;

    do

    {

        string ip;

        deque<string> op\_stack;

        op\_stack.push\_back("$");

        cout << "Enter the string to be parsed : ";

        getline(cin, ip);

        ip.push\_back('$');

        cout << "Stack\ti/p Buffer\tRelation\tAction" << endl;

        while (true)

        {

            for (int i = 0; i < op\_stack.size(); i++)

                cout << op\_stack[i];

            cout << "\t";

            cout << ip << "\t";

            int row = getPosition(terms, op\_stack.back(), n2);

            int column = getPosition(terms, ip.substr(0, 1), n2);

            if (op\_table[row][column] == '<')

            {

                op\_stack.push\_back("<");

                op\_stack.push\_back(ip.substr(0, 1));

                ip = ip.substr(1);

                cout << "\t"

                     << "<\t\tPush";

            }

            else if (op\_table[row][column] == '=')

            {

                op\_stack.push\_back("=");

                op\_stack.push\_back(ip.substr(0, 1));

                ip = ip.substr(1);

                cout << "\t"

                     << "=\t\tPush";

            }

            else if (op\_table[row][column] == '>')

            {

                string last;

                do

                {

                    op\_stack.pop\_back();

                    last = op\_stack.back();

                    op\_stack.pop\_back();

                } while (last != "<");

                cout << "\t"

                     << ">\t\tPop";

            }

            else

            {

                if (ip[0] == '$' && op\_stack.back() == "$")

                {

                    cout << "\t\t\tAccept\nString Parsed." << endl;

                    break;

                }

                else

                {

                    cout << endl

                         << "String cannot be Parsed." << endl;

                    break;

                }

            }

            cout << endl;

        }

        cout << "Continue?(Y/N) ";

        cin >> c;

        cin.ignore();

    } while (c == 'y' || c == 'Y');

    return 0;

}

**OUTPUT:**

