**FINITE AUTOMATA: -**

**PROGRAM: -**

#include<iostream>

#include<string>

using namespace std;

int main()

{ int flg=0;

string ST;

cout<<"RE=(a+b)\*bab\n";

cout<<"\nEnter a pattern:- ";

cin>>ST;

int len=ST.length();

for(int i=0;i<len-3;i++)

{ cout<<"Input- "<<ST[i]<<"--> State 1 (start)"<<endl;

if(ST[i]!='a'&& ST[i]!='b')

{ cout<<"NOT IN LANGUAGE (Reason- invalid character)\n";

flg=1;

break;

}

}

if(flg!=1)

{

if(ST[len-3]=='b'&&ST[len-2]=='a'&&ST[len-1]=='b')

{ cout<<"Input- b--> State 2"<<endl;

cout<<"Input- a--> State 3"<<endl;

cout<<"Input- b--> State 4 (accepting)"<<endl;

cout<<endl<<"String accepted by language";

}

else

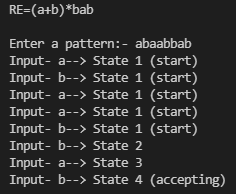
cout<<"NOT IN LANGUAGE (Reason- doesn't end with bab)";

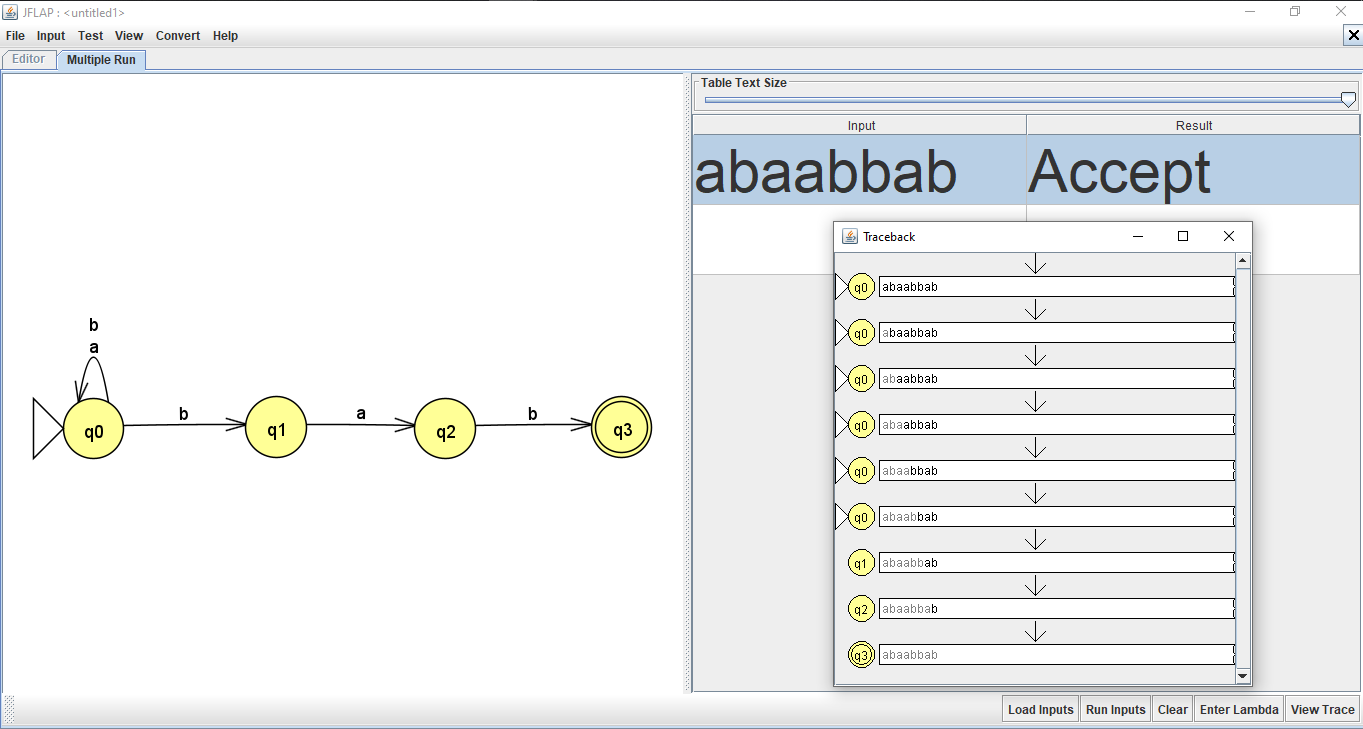
}

return 0;

}

**OUTPUT: -**





**MEALY MACHINE: -**

**PROGRAM: -**

#include<bits/stdc++.h>

using namespace std;

int mealy(string str) {

int count= 0;

int state = 1;

string output = " ";

string st\_output = "q1";

for(int i=0; i<str.size(); i++) {

if(state == 1) {

if(str[i] == 'a') {

state = 2;

output+="0";

}

else {

output+="0";

state=1;

}

}

else if(state == 2) {

if(str[i] == 'b') {

output+="0";

state = 3;

}

else {

output+="0";

state = 2;

}

}

else if(state == 3) {

if(str[i] == 'b') {

output+="1";

state = 4;

count++;

}

else {

output+="0";

state =2;

}

}

else {

if(str[i] == 'a') {

state=2;

output+="0";

}

else {

state=1;

output+="0";

}

}

output += " ";

st\_output += "--";

st\_output += str[i];

st\_output += "--q" + (state);

}

cout<<"The input states are: "<<st\_output<<endl;

cout<<"\nThe output string is: "<<output<<endl;

return count;

}

int main() {

string str;

cout<<"Enter the string to count occurence of 'abb': ";

cin>> str;

int count = mealy(str);

cout<<"Number of occurences of 'abb' is: "<<count<<endl;

if(count == 0)

cout<<"String rejected"<<endl;

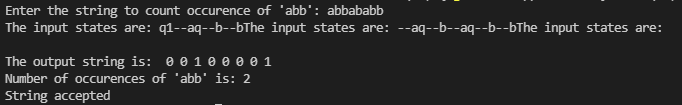
else

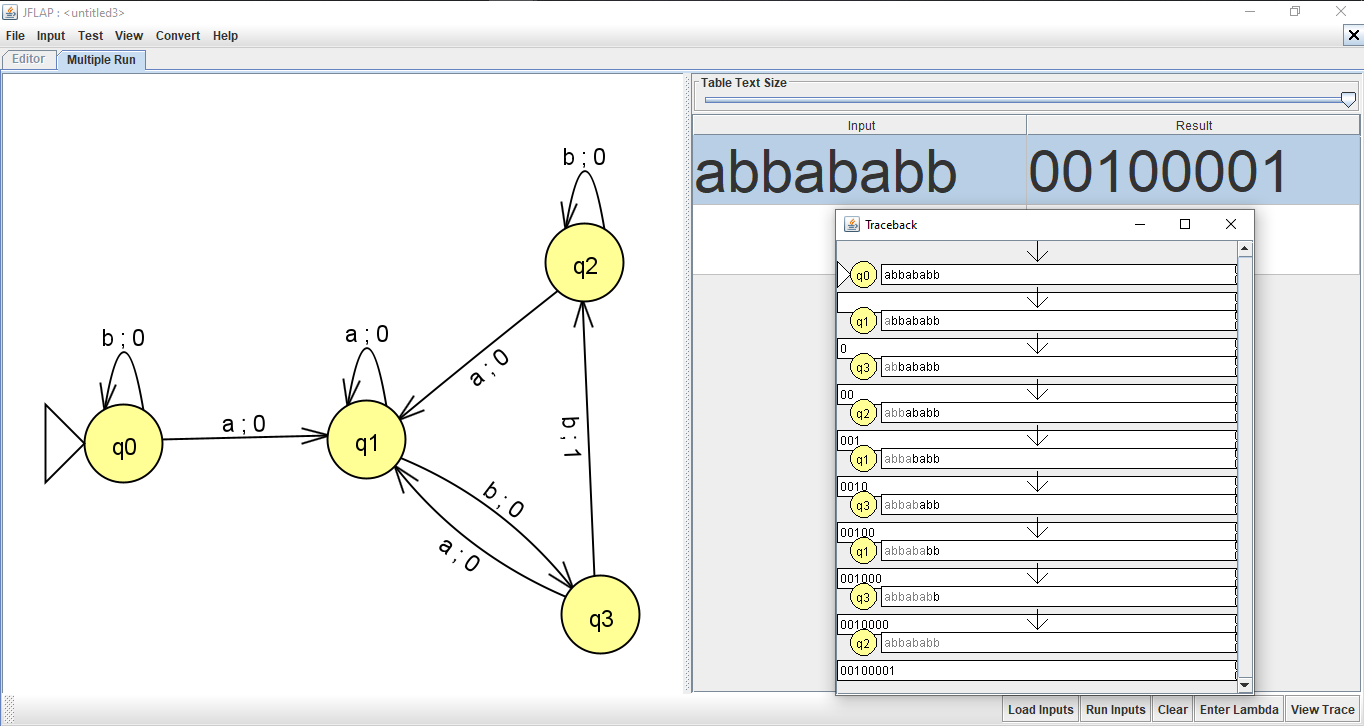
cout<<"String accepted";

return 0;

}

**OUTPUT: -**

****

****

**Conclusion:** Mealy machine for a given string and string tested to see if accepted or not and Mealy machine created in JFLAP and checked for input if accepted or not successfully completed.

**MOORE MACHINE: -**

**PROGRAM: -**

#include<bits/stdc++.h>

using namespace std;

int moore(string str) {

int count= 0;

int state = 1;

string output = "0 ";

string st\_output = "q1";

for(int i=0; i<str.size(); i++) {

if(state == 1) {

if(str[i] == 'a') {

state = 2;

output+="0";

}

else {

output+="0";

state=1;

}

}

else if(state == 2) {

if(str[i] == 'b') {

output+="0";

state = 3;

}

else {

output+="0";

state = 2;

}

}

else if(state == 3) {

if(str[i] == 'b') {

output+="1";

state = 4;

count++;

}

else {

output+="0";

state =2;

}

}

else {

if(str[i] == 'a') {

state=2;

output+="0";

}

else {

state=1;

output+="0";

}

}

output += " ";

st\_output += "--";

st\_output += str[i];

st\_output += "--q" + to\_string(state);

}

cout<<"The input states are: "<<st\_output<<endl;

cout<<"\nThe output string is: "<<output<<endl;

return count;

}

int main() {

string str;

cout<<"Enter the string to count occurence of 'abb': ";

cin>> str;

int count = moore(str);

cout<<"Number of occurences of 'abb' is: "<<count<<endl;

if(count == 0)

cout<<"String rejected"<<endl;

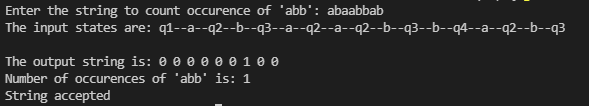
else

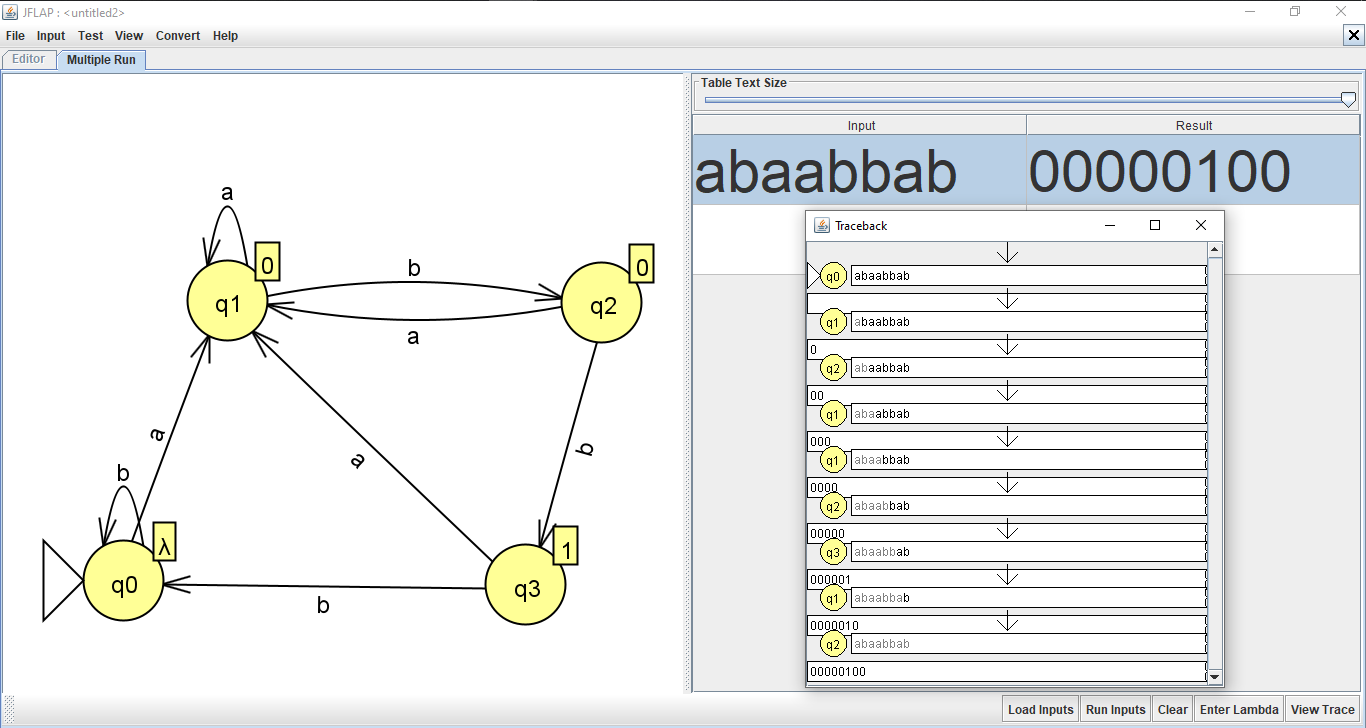
cout<<"String accepted";

return 0;

}

**OUTPUT: -**





**Conclusion:** Moore machine for a given string and string tested to see if accepted or not and Moore machine created in JFLAP and checked for input if accepted or not successfully completed.

**Null Production:**

**PROGRAM:**

#include <iostream>

using namespace std;

string S="ABAC";

string A="aA/^";

string B="bB/^";

string C="c";

string n="";

int nullable(string b)

{

int i;

int r=0;

for (i=0;i<b.length();i++)

{

if (b[i]=='^')

{

r++;

}

}

if(r==0)

return 0;

else

return 1;

}

void remove\_from\_S()

{

int i=0,j=0,k=0;

cout<<"S->"<<S<<"/ ";

int h[S.length()];

for (i=0;i<S.length();i++)

{

h[i]=0;

}

for(j=0;j<n.length();j++)

{

for (i=0;i<S.length();i++)

{

if(n[j]==S[i])

{

h[i]=1;

for (k=0;k<S.length();k++)

{

if(k==i)

{

continue;

}

else

{

cout<<S[k];

}

}

cout<<"/ ";}

}

}

for(i=0;i<S.length();i++)

{

if(h[i]==1)

{

continue;

}

else

{

if(S[i]=='/' or S[i]=='^')

{

continue;

}

else

{

cout<<S[i];

}

}

}

cout<<"\n";

}

void remove\_from\_A()

{

int i=0,j=0,k=0;

cout<<"A->";

for (i=0;i<A.length();i++)

{

if(A[i]=='/' or A[i]=='^')

{

continue;

}

else

{

cout<<A[i];

}

}

cout<<"/ ";

for(j=0;j<n.length();j++)

{

for (i=0;i<A.length();i++)

{

if(n[j]==A[i])

{

for (k=0;k<A.length();k++)

{

if(k==i)

{

continue;

}

else

{

if(A[k]=='/' or A[k]=='^')

{

continue;

}

else

{

cout<<A[k];

}

}

}

}

}

}

cout<<"\n";

}

void remove\_from\_B()

{

int i=0,j=0,k=0;

cout<<"B->";

for (i=0;i<B.length();i++)

{

if(B[i]=='/' or B[i]=='^')

{

continue;

}

else

{

cout<<B[i];

}

}

cout<<"/ ";

for(j=0;j<n.length();j++)

{

for (i=0;i<B.length();i++)

{

if(n[j]==B[i])

{

for (k=0;k<B.length();k++)

{

if(k==i)

{

continue;

}

else{if(B[k]=='/' or B[k]=='^'){ continue;}

else{ cout<<B[k];

}

}

}

}

}

}

cout<<"\n";\

}

void remove\_from\_C()

{

int i=0,j=0,k=0;

cout<<"B->";

for (i=0;i<C.length();i++)

{

if(C[i]=='/' or C[i]=='^'){ continue;}

else{ cout<<C[i]; }

}

for(j=0;j<n.length();j++)

{

for (i=0;i<C.length();i++)

{

if(n[j]==C[i]){

for (k=0;k<C.length();k++)

{

if(k==i){ continue;}

else{if(C[k]=='/' or C[k]=='^'){continue;}

else{ cout<<C[k];

}

}

}

}

}

}

cout<<"\n";

}

int main(){

cout<<"removal of null production\n";

cout<<"S->"<<S<<"\n";

cout<<"A->"<<A<<"\n";

cout<<"B->"<<B<<"\n";

cout<<"C->"<<C<<"\n";

cout<<"the nullable products are \n";

if(nullable(S))

{

cout<<"S\t";

n=n+"S";

}

if (nullable(A))

{ cout<<"A\t";

n=n+"A";

}

if (nullable(B))

{ cout<<"B\t";

n=n+"B";

}

if (nullable(C))

{

cout<<"C\t";

n=n+"C";

}

cout<<"\n";

cout<<"productions after removing the null production\n";

remove\_from\_S();

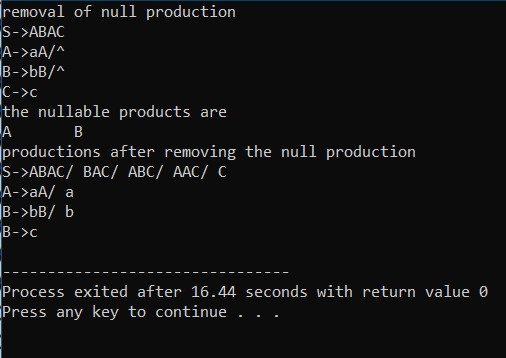
remove\_from\_A();

remove\_from\_B();

remove\_from\_C();

}

**OUTPUT:**



**Unit Production:**

**PROGRAM:**

#include <iostream>

using namespace std;

string S="Aa/A";

string A="B/b";

string B="a";

int main()

{

cout<<"removal of unit production\n";

cout<<"S->"<<S<<"\n";

cout<<"A->"<<A<<"\n";

cout<<"B->"<<B<<"\n";

int i,j; string s="";

string a="";

string b="";

S="/"+S+"/";

A="/"+A+"/";

B="/"+B+"/";

for (i=0;i<B.length();i++)

{

if(B[i]=='A' or B[i]=='B' or B[i]=='S'){

if(B[i-1]=='/' and B[i+1]=='/'){

if(B[i]=='S'){B=B+S; }

if(B[i]=='A'){B=B+A;}

if(B[i]=='B'){ B=B+B;}

}

}

else{ continue; }

}

for(i=1;i<B.length()-1;i++)

{

if(B[i]=='A' or B[i]=='B' or B[i]=='S')

{

if(B[i-1]=='/' and B[i+1]=='/')

{ continue; }

}

if(B[i]=='/' and B[i+1]=='/')

{

i++;

}

b=b+B[i];

}

for (i=0;i<A.length();i++)

{

if(A[i]=='A' or A[i]=='B' or A[i]=='S')

{

if(A[i-1]=='/' and A[i+1]=='/')

{

if(A[i]=='S'){ A=A+S; }

if(A[i]=='A'){ A=A+a;}

if(A[i]=='B'){ A=A+b;}

}

}

else

{

continue;

}

}

for(i=1;i<A.length();i++)

{

if(A[i]=='A' or A[i]=='B' or A[i]=='S'){

if(A[i-1]=='/' and A[i+1]=='/') { i++; continue; }}

if(A[i]=='/' and A[i+1]=='/'){ i++;}

a=a+A[i];

}

for (i=0;i<S.length();i++)

{

if(S[i]=='A' or S[i]=='B' or S[i]=='S'){

if(S[i-1]=='/' and S[i+1]=='/')

{ if(S[i]=='S')

{

S=S+S;

}

if(S[i]=='A'){ S=S+a;}

if(S[i]=='B'){ S=S+b;} }

}

else{continue;}

}

for(i=1;i<S.length();i++)

{

if(S[i]=='A' or S[i]=='B' or S[i]=='S')

{

if(S[i-1]=='/' and S[i+1]=='/')

{

i+=1;

continue;

}

}

if(S[i]=='/' and S[i+1]=='/')

{

i++;

}

s=s+S[i];

}

cout<<"\n";

cout<<"productions after removing the unit production\n";

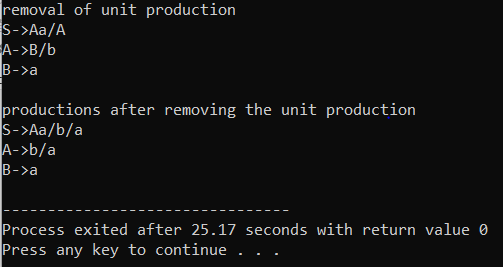
cout<<"S->"<<s<<"\n";

cout<<"A->"<<a<<"\n";

cout<<"B->"<<b<<"\n";

}

**OUTPUT:**



**Pushdown Automata:**

**PROGRAM:**

#include <stdio.h>

#include <iostream>

#include <conio.h>

#include<string.h>

using namespace std;

#define MAX 20

class Stack

{

public:

int top;

char a[MAX];

Stack()

{

top = -1;

}

void push(char x)

{

a[++top] = x;

}

void pop()

{

--top;

}

char peek()

{

char x = a[top];

return x;

}

bool isEmpty()

{

return (top < 0);

}

void display()

{

for (int s = top; s >= 0; s--)

cout << a[s];

}

};

void displaytransition(string state, Stack stack, char str[], int t, int l)

{

cout << " " << state << " | ";

for (int k = t; k < l; k++)

cout << str[k];

for (int k = 0; k < (10 - l + t); k++)

cout << " ";

cout << " | ";

stack.display();

cout << endl;

}

int main()

{

enum state { q0, q1, q2 };

int len;

state current\_state = q0;

char str[10];

class Stack stack;

//clrscr();

cout << "Enter the string: ";

cin >> str;

len = strlen(str);

cout << "Using Push Down Automata: " << endl;

cout << "State | Input | Stack" << endl;

cout << "--------------------------" << endl;

displaytransition("q0", stack, str, 0, len);

for (int i = 0; i <= len;)

{

switch (current\_state)

{

case q0:

if (i < len)

if (str[i] == 'a' && stack.isEmpty())

{

current\_state = q0;

stack.push('a');

displaytransition("q0", stack, str, i+1, len);

}

else if (str[i] == 'a' && stack.peek() == 'a')

{

current\_state = q0;

stack.push('a');

displaytransition("q0", stack, str, i+1, len);

}

else if (str[i] == 'b' && stack.isEmpty())

{

current\_state = q0;

stack.push('b');

displaytransition("q0", stack, str, i+1, len);

}

else if (str[i] == 'b' && stack.peek() == 'a')

{

current\_state = q1;

stack.pop();

displaytransition("q1", stack, str, i+1, len);

}

else

{

cout << "--------------------------" << endl;

cout << "String Rejected !!";

goto end;

}

i++;

break;

case q1:

if (i < len && str[i] == 'b' && stack.peek() == 'a')

{

current\_state = q1;

stack.pop();

displaytransition("q1", stack, str, i+1, len);

}

else if (i == len && stack.isEmpty())

{

current\_state = q2;

displaytransition("q2", stack, str, i, len);

cout << "--------------------------" << endl;

cout << "String Accepted !!";

goto end;

}

else

{

cout << "--------------------------" << endl;

cout << "String Rejected !!";

goto end;

}

i++;

}

}

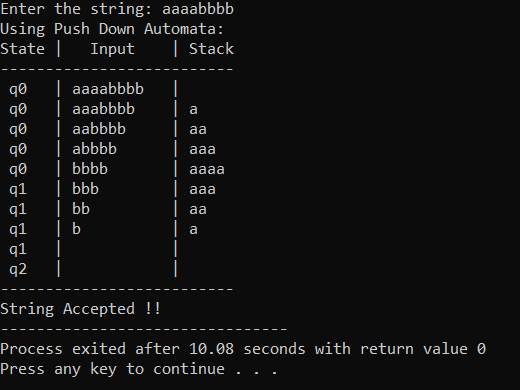
end:

\_getch();

return 0;

}

**OUTPUT:**



**Null Production and Unit Production:**

**PROGRAM:**

#include<bits/stdc++.h>

using namespace std;

void hello() {

    cout<<"Hello";

}

bool isPresent(string s, string sub) {

    int found = 0;

    while(s.find(sub, found) != string::npos) {

        found = s.find(sub, found);

        if( (found == 0 || s[found-1] == '/') && (found + sub.size() == s.size() || s[found + sub.size()] == '/') )

            return true;

        found++;

    }

    return false;

}

bool allNull(string s, vector<char> variables) {

    bool present;

    for(char c: s) {

        present = false;

        for(char a: variables) {

            if(a==c) {

                present = true;

                break;

            }

        }

        if(!present)

            return false;

    }

    return true;

}

int takeProductions(string str[]) {

    cout<<"Enter the productions:\n  Use \"->\" for production \n  Use \"/\" for OR Example A->A\*B/a \n  Use \'^\' for null \n  And press 0 to exit"<<endl;

    int i=0;

    // do {

    //     getline(cin,str[i]);

    //     i++;

    // }while(str[i-1] != "0");

    // i -= 2;

            //hardcoded values

            str[0] = "S->ABA";

            str[1] = "A->aA/^";

            str[2] = "B->bB/^";

            i=2;

    return i;

}

void EliminateNUll(string str[], int i) {

    cout<<"\nEliminating null production..."<<endl;

    //find nullable variables

    vector<char> nullable\_variables;

    int temp =i;

    nullable\_variables.push\_back('^');

    bool changed = true;

    while(changed) {

        changed = false;

        for(int j=i; j>=0; j--) {

            string production = str[j];

            for(int k=3; k<production.size(); ) {

                int right = production.find('/', k);

                if(right == -1)

                    right = production.size();

                if(allNull(production.substr(k, right-k), nullable\_variables)) {

                    string s(1, production[0]);

                    if(!allNull(s, nullable\_variables)) {

                        nullable\_variables.push\_back(production[0]);

                        if(production[k] == '^') {

                            if(k == 3)

                                production.erase(k,2);

                            else

                                production.erase(k-1,2);

                        }

                        changed =true;

                    }

                    k=production.size();

                }

                else

                    k = right+1;

            }

            str[j] = production;

        }

    }

    nullable\_variables.erase(nullable\_variables.begin());

    cout<< "\tNullable variables are: " ;

    for(char c: nullable\_variables)

        cout << c << " ";

    for(i = temp; i >= 0; i-- ) {

        string production = str[i];

        for(int j=0; j<nullable\_variables.size(); j++) {

            for(int i = 3; i<production.size(); i++) {

                if( production[i] == nullable\_variables[j] ) {

                    int pos = i;

                    int left = pos-1, right = pos+1;

                    while(production[left] != '>' && production[left] != '/')

                        left--;

                    while(right != production.size() && production[right] != '/')

                        right++;

                    if ( right - left > 2) {

                        string sub = production.substr(left+1, right-left-1);

                        sub.erase(sub.begin() + (pos-left-1));

                        // Check here whether the duplicate is present

                        if(!isPresent(production.substr(3), sub)) {

                            sub = "/" + sub;

                            production.insert(right, sub);

                            j = 0;

                        }

                    }

                }

            }

        }

        str[i] = production;

    }

    cout<<"\n\nAfter removing null productions"<<endl;

    for(int j=0; j<=temp; j++)

        cout<<str[j]<<endl;

}

void EliminateUnit(string str[], int i) {

    cout<<"\n\nEliminating unit production..."<<endl;

    int temp = i;

    for(; i>=0;i--) {

        string production = str[i];

        for(int j=0; j<production.size(); j++ ) {

            if(production[j] == '>' || production[j] == '/') {

                if(j+2 == production.size() || production[j+2] == '/') {

                    if(isupper(production[j+1])) { //use is Upper

                        if (production[0] == production[j+1]) { //if recursion then ignore

                            if(j+2 == production.size() )

                                production.erase(j);

                            else if( j == 3)

                                production.erase(j+1, 2);

                            else

                                production.erase(j, 3);

                        }

                        else {

                            int k= temp;

                            while(str[k][0] != production[j+1])

                                k--;

                            production.replace(j+1,1,str[k],3); //replace unit with its production

                        }

                    }

                }

            }

        }

        str[i] = production;

    }

    cout<<"\nAfter removing unit productions: "<<endl;

    for(int j=0; j<=temp; j++)

        cout<<str[j]<<endl;

}

int main() {

    string str[10];

    int i = takeProductions(str);

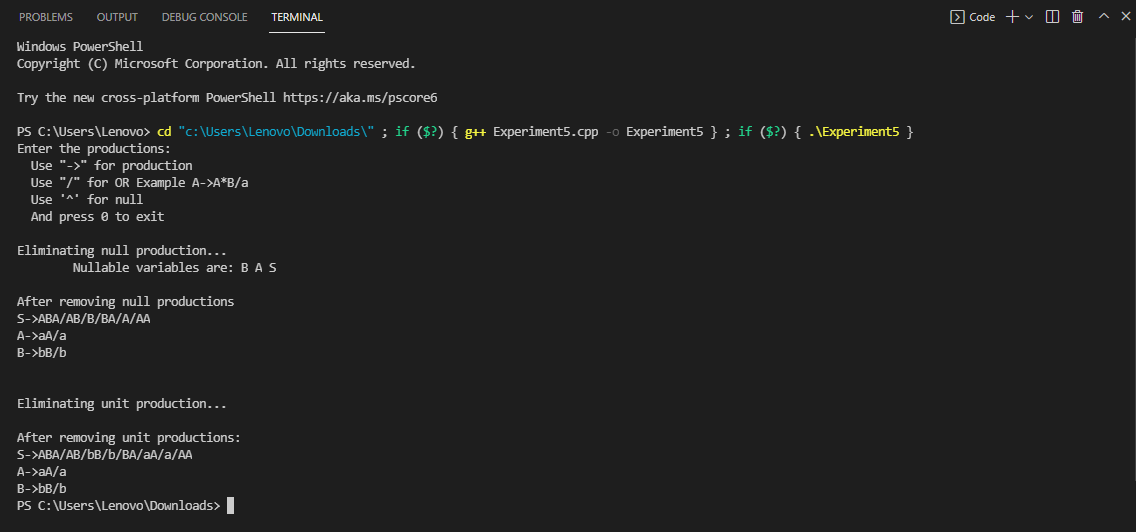
    EliminateNUll(str, i);

    EliminateUnit(str, i);

    return 0;

}

OUTPUT:



**Conclusion:** Program of Null and Unit Production were successfully implemented.

**Context Free Grammar to Chomsky Normal Form:**

**PROGRAM:**

#include<bits/stdc++.h>

using namespace std;

int main() {

    string str[10];

    cout<<"Enter unit and null free productions only!!!"<<endl;

    cout<<"Enter the productions:\n  Use \"->\" for production \n  Use \"/\" for OR Example A->A\*B/a \n  Use \'^\' for null \n  And press 0 to exit"<<endl;

    int i=0;

    // do {

    //     getline(cin,str[i]);

    //     i++;

    // }while(str[i-1] != "0");

    // i -= 2;

            //hardcoded values

            str[0] = "S->AB";

            str[1] = "A->BS";

            str[2] = "B->SA/a";

            i=2;

            cout<<"The productions are :" << endl;

            for(int j=0;j<=i;j++)

                cout << str[j] << endl;

    //removing terminal symbols

    map<char, string> newProductions;

    for(int j=0; j<=i; j++) {

        string production = str[j];

        for(int k=3; k<production.size(); k++ ) {

            if(islower(production[k])) {

                if( (production[k-1] != '/' && production[k-1] != '>') || (k+1 != production.size() && production[k+1] != '/') ) {

                    if(newProductions.find(production[k]) == newProductions.end()) {

                        string newPro;

                        string s(1, ('X' + newProductions.size()));

                        newPro += s + "->" + production[k];

                        newProductions.insert({production[k], newPro});

                        production[k] = newPro[0];

                        auto itr = newProductions.find(production[k]);

                        str[++i] = newPro;

                    }

                    else {

                        production[k] = newProductions[production[k]][0];

                    }

                }

            }

        }

        str[j] = production;

    }

    // cout<<"\nAfter removing terminal productions: "<<endl;

    // for(int j=0; j<=i; j++)

    //     cout<<str[j]<<endl;

    //convert to CNF

    map<string, string> newP;

    for(int j=0; j<=i; j++) {

        string production = str[j];

        int right = 4;

        for(int left=2; left<production.size(); right=left+2) {

            while(right != production.size() && production[right] != '/')

                right++;

            if (right - left > 3) {

                string key = production.substr(left+2, right-left-2);

                if (newP.find(key) == newP.end()) {

                    string s(1, ('L' + newP.size()));

                    string newPro = s + "->" + key;

                    newP.insert({key, newPro});

                    production.replace(left+2, right-left-2, newPro,0,1);

                    i++;

                    str[i] = newPro;

                }

                else {

                    production.replace(left+2, right-left-2, newP[key],0,1);

                }

            }

            if(islower(production[left+1]))

                left--;

            left = left+3;

        }

        str[j] = production;

    }

    cout<<"\nAfter converting to CNF: "<<endl;

    for(int j=0; j<=i; j++)

        cout<<str[j]<<endl;

    //indexing Non terminals

    map<char, int> indexing;

    int index = 1;

    for(int j=0; j<=i; j++) {

        for(int k=0; k<str[j].size(); k++) {

            if(isupper(str[j][k])) {

                if (indexing.find(str[j][k]) == indexing.end())

                    indexing.insert({str[j][k], index++});

            }

        }

    }

    //Eliminating > and left recursions

    int temp = i;

    for(int j=0; j<=temp; j++) {

        string production = str[j];

        int left = indexing[production[0]];

        int ii=0;

        int k=3;

        do {

            if(!islower(production[k])) {

                while(left > indexing[production[k]]) {

                    int temp = 0;

                    while(str[temp][0] != production[k])

                        temp++;

                    string substitute = str[temp].substr(3);

                    int last = k+1;

                    while(last != production.size() && production[last] != '/')

                        last++;

                    string s = production.substr(k+1, last-k-1);

                    int a = substitute.find('/');

                    while(a != -1) {

                        substitute.insert(a,s);

                        a = substitute.find('/', a+s.size()+1);

                    }

                    substitute.append(s);

                    production.replace(k, last-k, substitute);

                    //k += substitute.size() -2;

                    if(islower(production[k]))

                        break;

                }

                if(!islower(production[k]) && left == indexing[production[k]]) {

                    int last = k+1;

                    while(last != production.size() && production[last] != '/')

                        last++;

                    string sub = production.substr(k, last-k);

                    production.erase(k, last-k+1);

                    string s(1, 'E'+ii);

                    ii++;

                    string mainSub = production.substr(3);

                    int a = mainSub.find('/');

                    while(a != -1) {

                        mainSub.insert(a,s);

                        a = mainSub.find('/', a+s.size()+1);

                    }

                    mainSub.append(s);

                    production.push\_back('/');

                    production.append(mainSub);

                    s += "->" + sub + s + "/" + sub;

                    i++;

                    str[i] = s;

                }

            }

            k = production.find('/', k) + 1;

        }while(k != 0);

        str[j] = production;

    }

    cout<<"\nAfter removing left recursions: "<<endl;

    for(int j=0; j<=i; j++)

        cout<<str[j]<<endl;

    //Convert to GNF

    for(int j=0; j<=i; j++) {

        string production = str[j];

        int k=3;

        do {

            while(!islower(production[k])) {

                int temp = 0;

                while(str[temp][0] != production[k])

                    temp++;

                string substitute = str[temp].substr(3);

                int last = k+1;

                while(last != production.size() && production[last] != '/')

                    last++;

                string s = production.substr(k+1, last-k-1);

                int a = substitute.find('/');

                while(a != -1) {

                    substitute.insert(a,s);

                    a = substitute.find('/', a+s.size()+1);

                }

                substitute.append(s);

                production.replace(k, last-k, substitute);

            }

            k = production.find('/', k) + 1;

        }while(k != 0);

        str[j] = production;

    }

    cout<<"\nGNF Form: "<<endl;

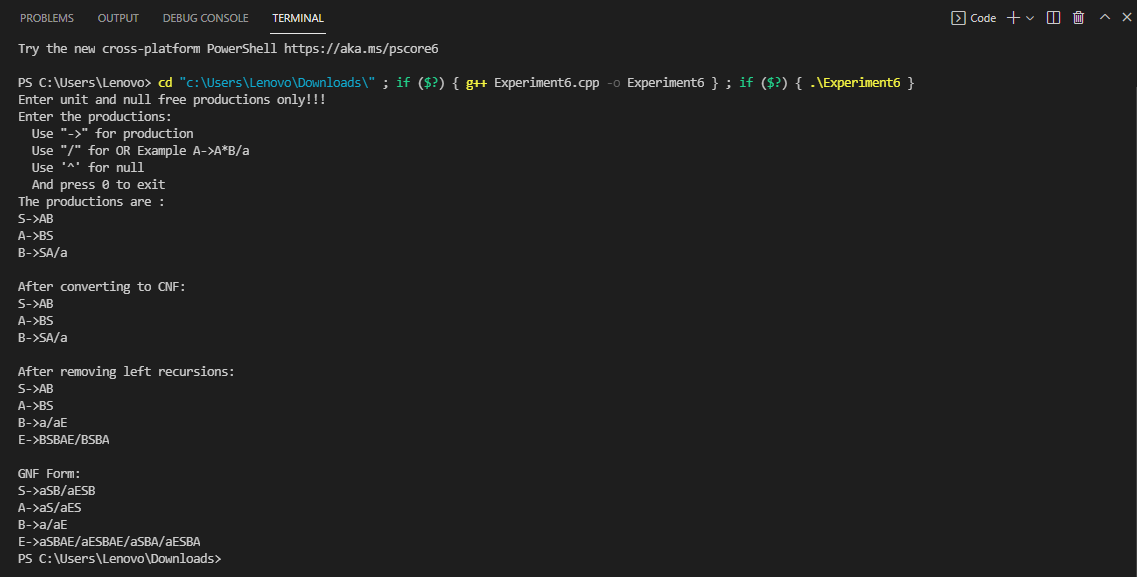
    for(int j=0; j<=i; j++)

        cout<<str[j]<<endl;

    return 0;

}

**OUTPUT:**



**Conclusion:** Program of CFG to CNF were successfully implemented.

**Pushdown Automata:**

**PROGRAM:**

#include<iostream>

#include<vector>

using namespace std;

int main() {

    cout << "Pushdown for accepting a string with more number of a's than b" << endl;

    string input;

    cout << "Enter the string containing 'a' and 'b' only: ";

    cin >> input;

    vector<char> stack;

    stack.push\_back('z');

    int top =0;

    while(!input.empty()) {

        cout << "Stack contents(down to top) : ";

        for(char c: stack)

            cout << c << " ";

        if(stack[top] == 'z') {

            stack.push\_back(input.front());

            top++;

        }

        else if(stack[top] == 'a') {

            if(input.front() == 'a') {

                stack.push\_back(input.front());

                top++;

            }

            else {

                stack.pop\_back();

                top--;

            }

        }

        else if(stack[top] == 'b') {

            if(input.front() == 'b') {

                stack.push\_back(input.front());

                top++;

            }

            else {

                stack.pop\_back();

                top--;

            }

        }

        input.erase(input.begin());

        cout << "\n\nString : " << input << endl;

    }

    if(stack[top] == 'a')

        cout << "\nString is accpeted\n";

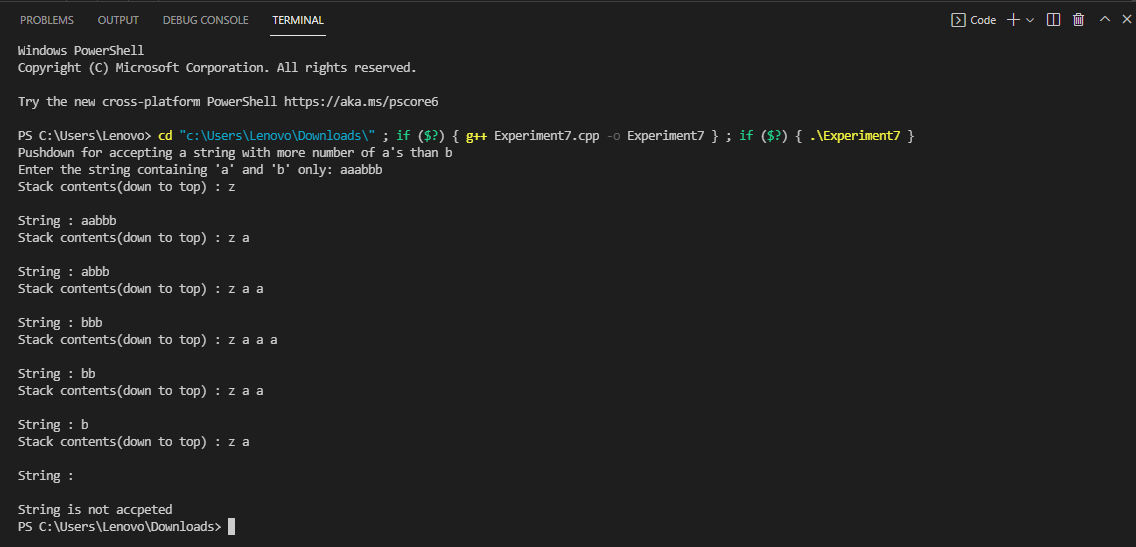
    else

        cout << "\nString is not accpeted\n";

    return 0;

}

**OUTPUT:**



**Conclusion:** Program of Pushdown Automata were successfully implemented.

**Turing Machine:**

**PROGRAM:**

#include<iostream>

#include<vector>

using namespace std;

int main() {

    cout<<"Turing machine that accepts palindrome\n\tstring should contain a and b only \n\tnull is denoted by 0"<<endl;

    string tape;

    cout<<"Enter the string: ";

    cin>>tape;

    tape = "0" + tape + "0";

    int head =0;

    int state = 0;

    cout << "Tape: " << tape <<endl;

    cout << "state = " << state << " head = " << head << endl << endl;

    while(state != -1 && state != 7) {

        if(state == 0) {

            head++;

            state = 1;

        }

        else if(state == 1) {

            if(tape[head] == 'a') {

                tape[head] = '0';

                head++;

                state = 2;

            }

            else if(tape[head] == 'b') {

                tape[head] = '0';

                head++;

                state = 4;

            }

            else {

                state = 7;

            }

        }

        else if(state == 2) {

            while(tape[head] != '0')

                head++;

            state = 3;

            head--;

        }

        else if(state == 4) {

            while(tape[head] != '0')

                head++;

            state = 5;

            head--;

        }

        else if(state == 3) {

            if(tape[head] == 'a') {

                tape[head] = '0';

                head--;

                state = 6;

            }

            else if(tape[head] == 'b') {

                state = -1;

            }

            else {

                state = 7;

            }

        }

        else if(state == 5) {

            if(tape[head] == 'b') {

                tape[head] = '0';

                head--;

                state = 6;

            }

            else if(tape[head] == 'a') {

                state = -1;

            }

            else {

                state = 7;

            }

        }

        else {

            while(tape[head] != '0')

                head--;

            state = 1;

            head++;

        }

        cout << "Tape: " << tape <<endl;

        cout << "state = " << state << " head = " << head << endl << endl;

    }

    if(state == 7)

        cout << "The string is palindrome";

    else

        cout << "The string is not palindrome";

    return 0;

}

**OUTPUT:**



**Conclusion:** Program of Turning Machine were successfully implemented.