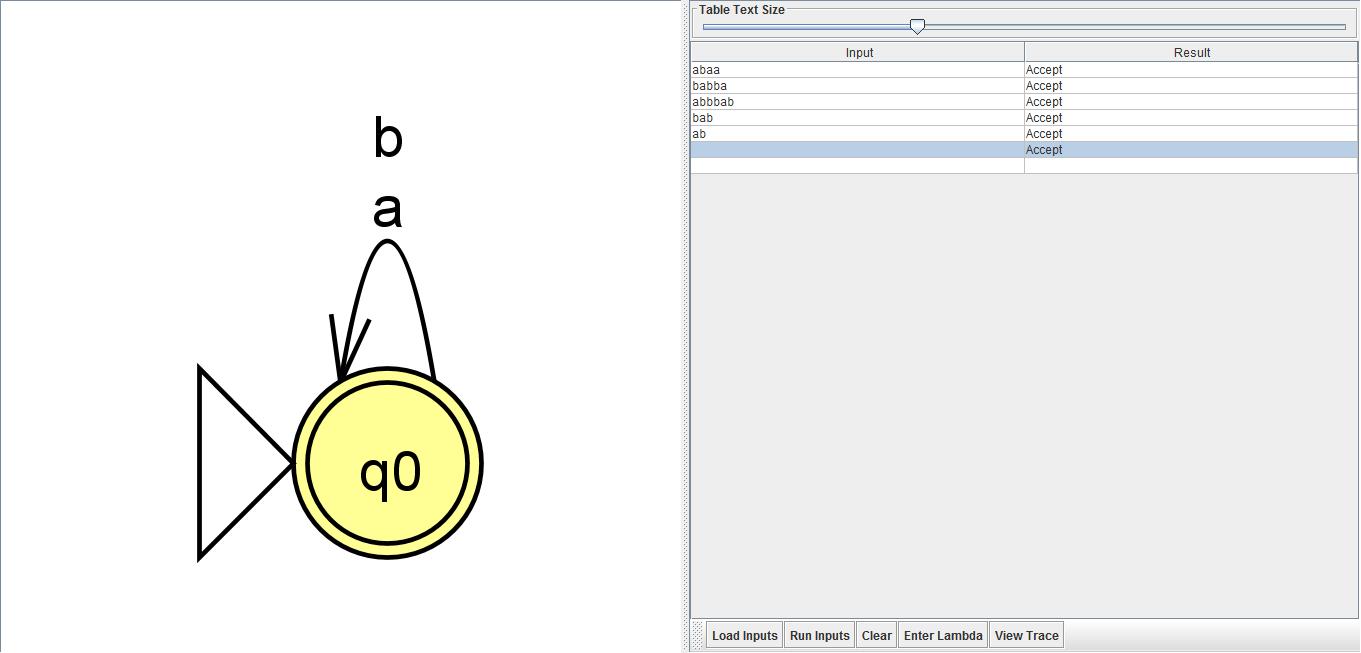
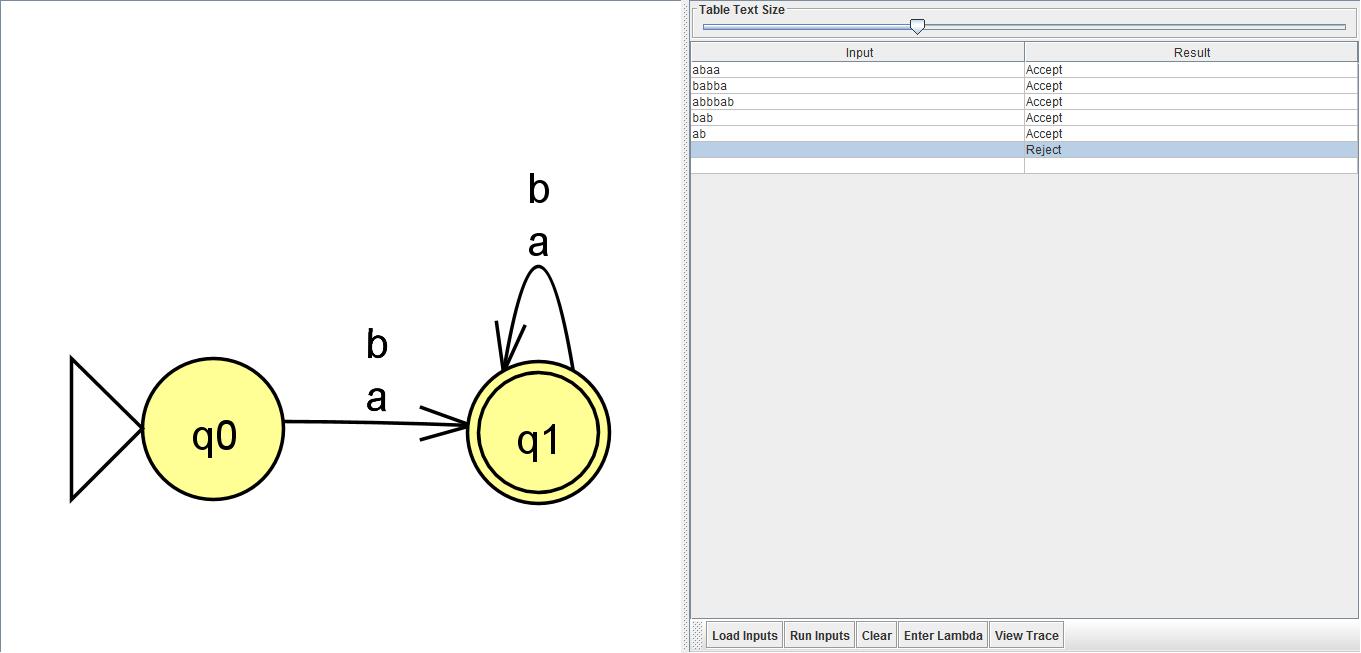
EXPERIMENT 2:

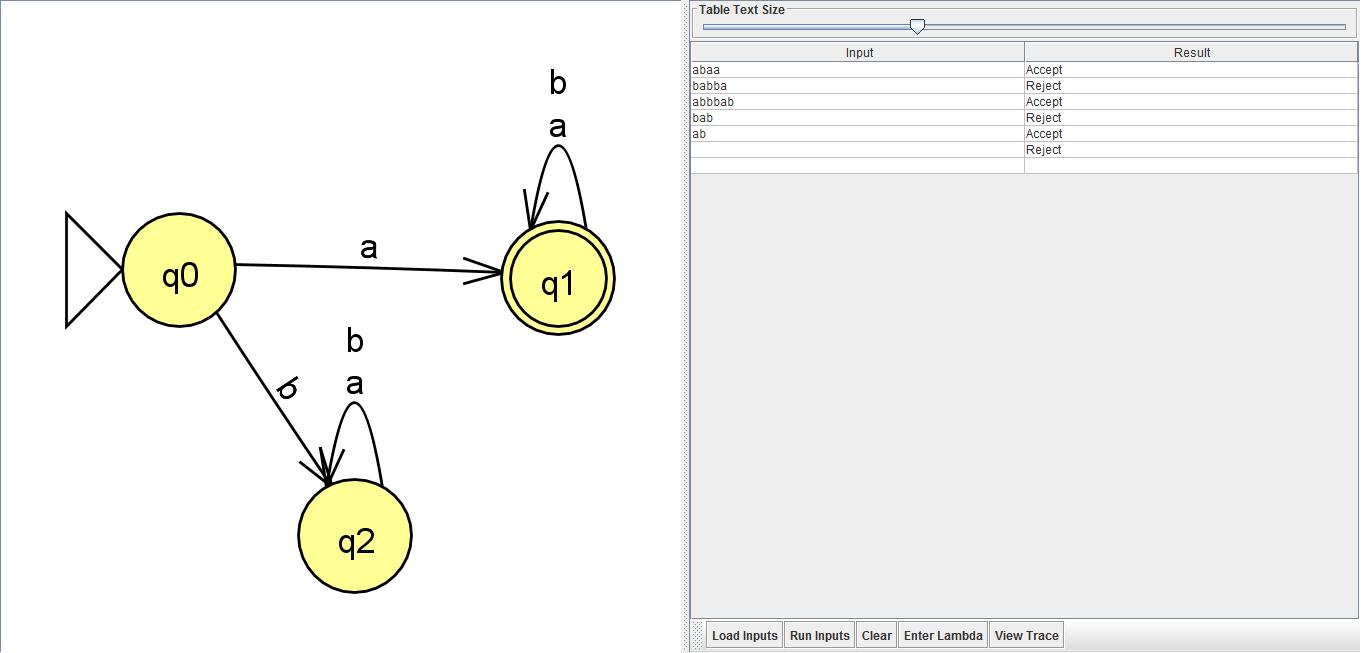
1. Construct a minimal DFA which accepts all strings of a, b including lambda.



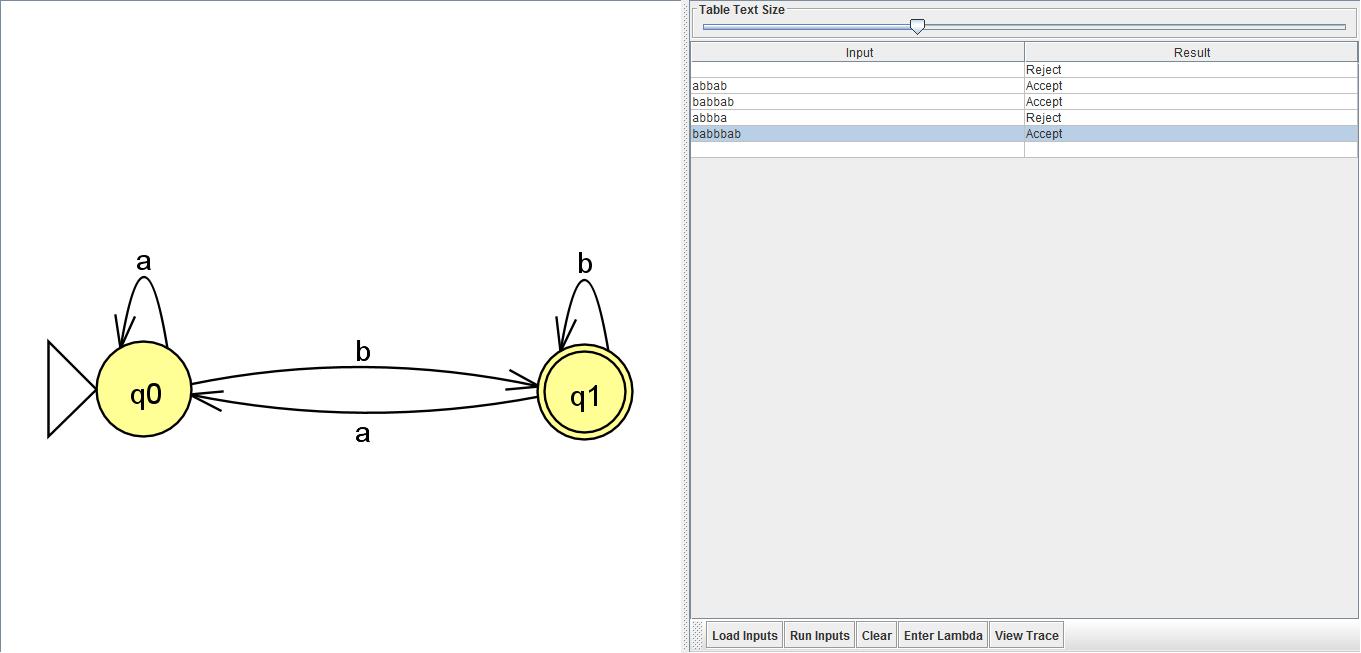
1. Construct a minimal DFA which accepts all strings of a, b excluding lambda.



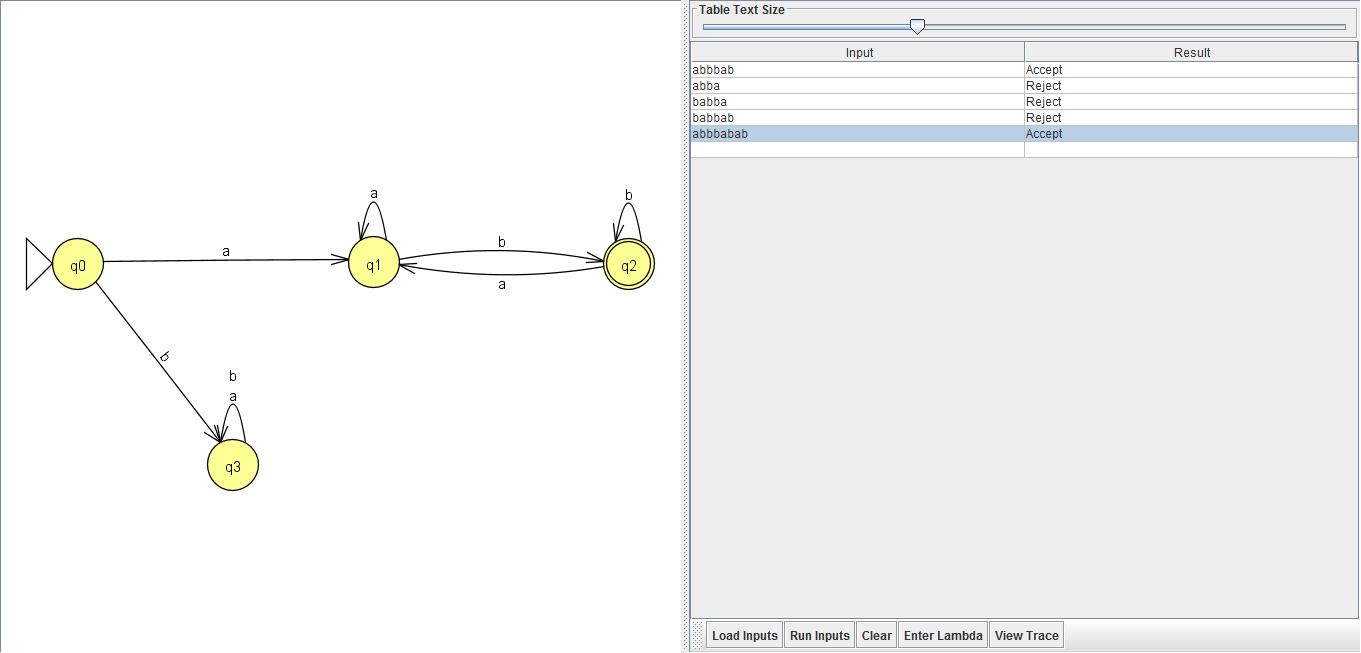
1. Construct a minimal DFA that accepts all string starting with ‘a’.



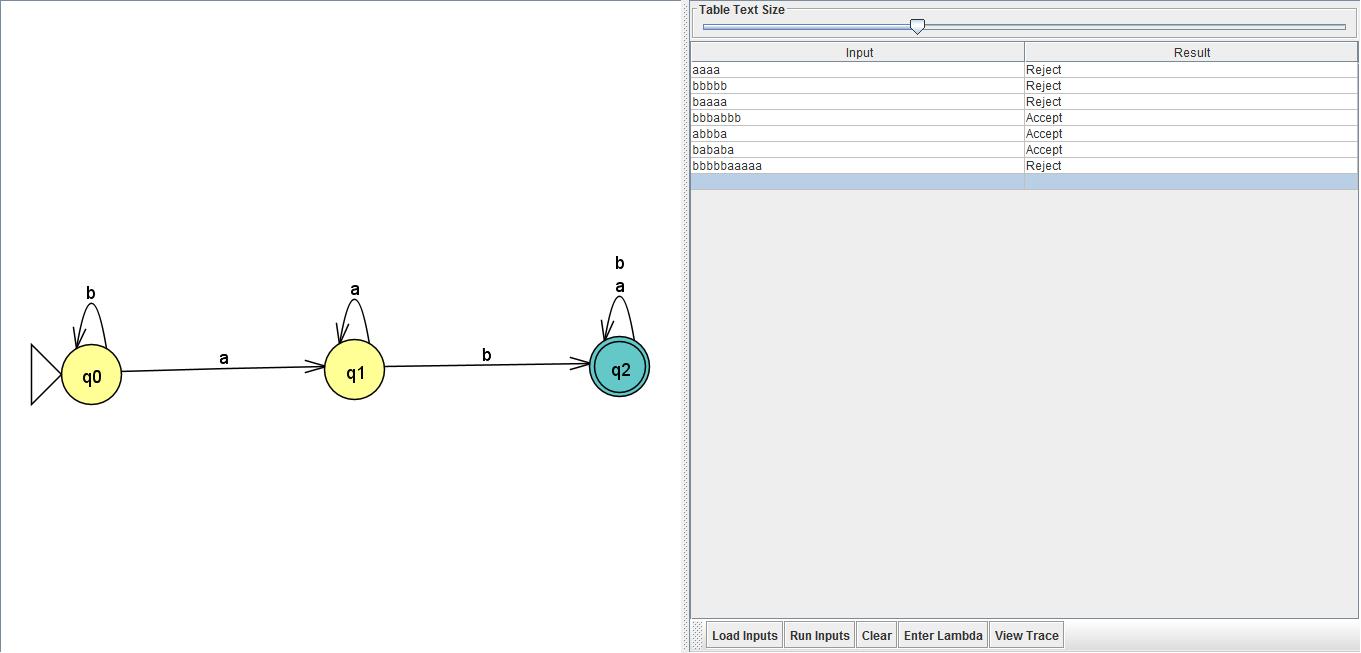
1. Construct a minimal DFA that accepts all strings that ends with ‘b’.



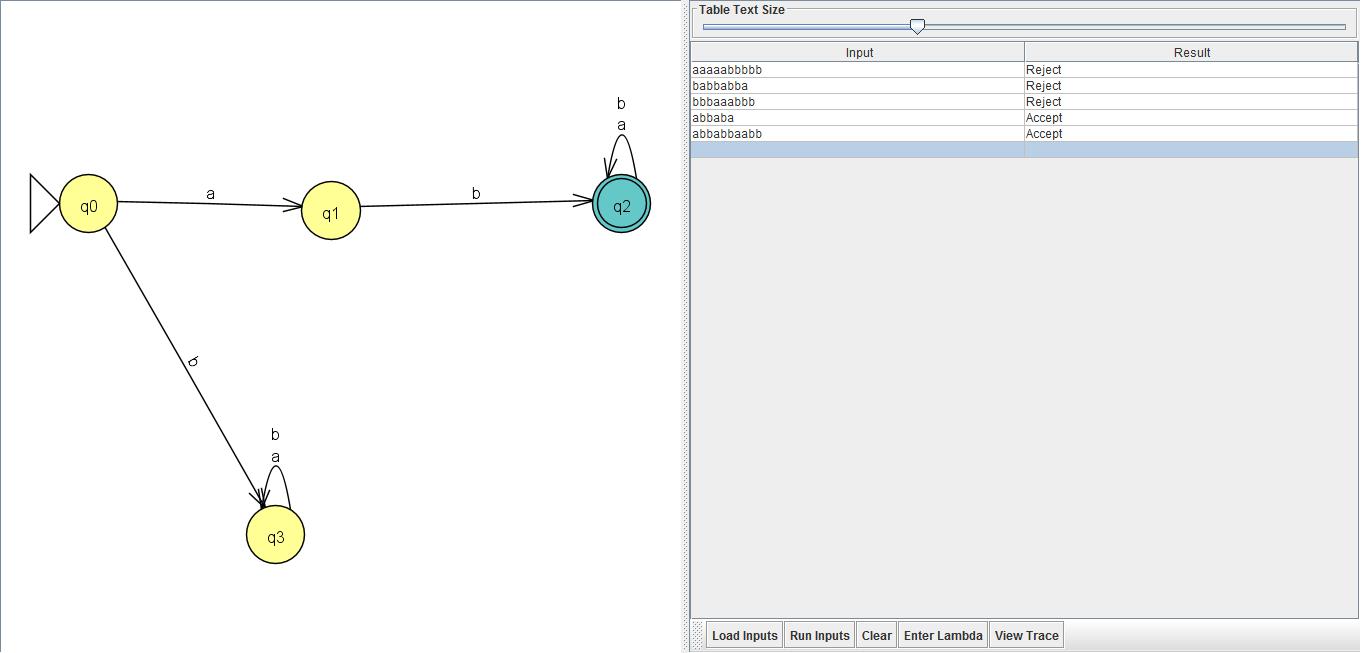
1. Construct a minimal DFA that accepts all strings starting with ‘a’ and ending with ‘b’.



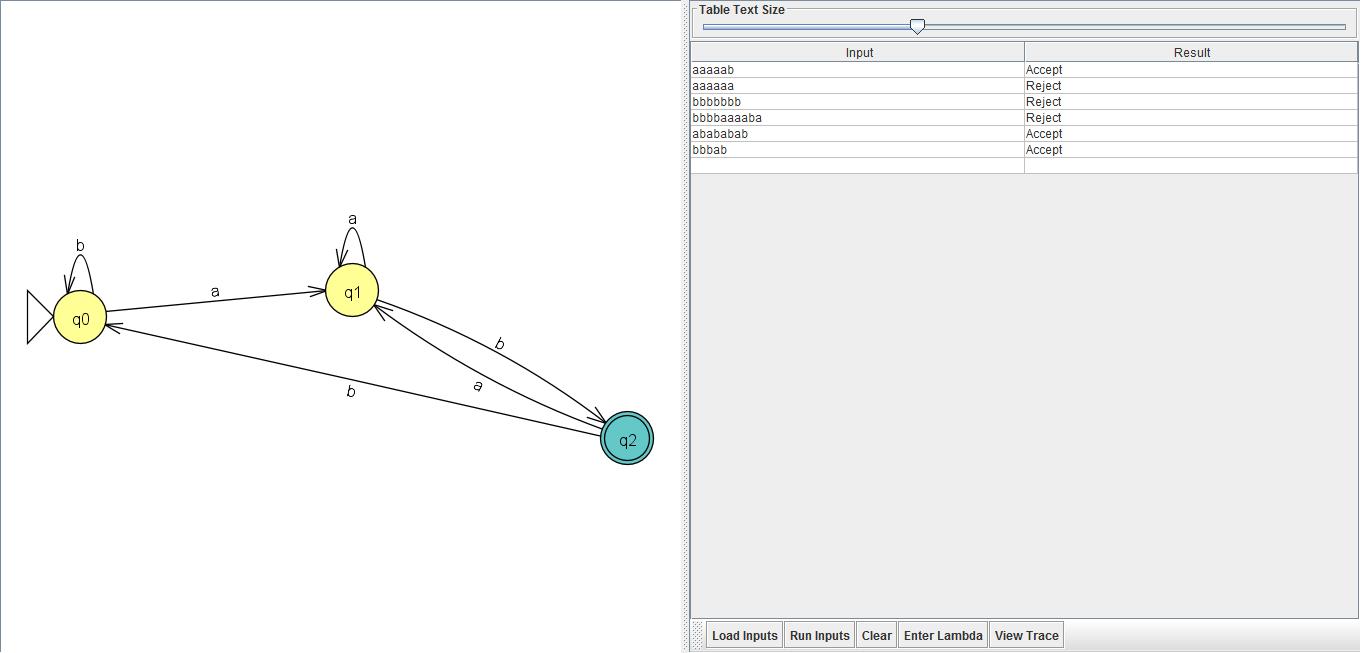
1. Construct a minimal DFA that accepts strings with substring ‘ab’.



1. Construct a minimal DFA that accepts all strings which starts with ‘ab’.



1. Construct a minimal DFA that accepts all strings which ends with ‘ab’.



EXPERIMENT 1:

1.Write a C program to find all the prefix of a String.

#include<iostream>

#include<string.h>

using namespace std;

int main(){

string s;

cout<<"enter a string:\n";

cin>>s;

int l=1;

int length=s.length();

while(l<=length)

{

for(int j=0;j<l;j++){

cout<<s[j];

}

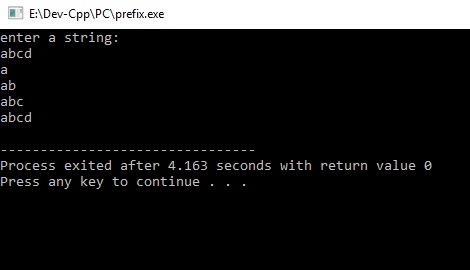
l++;

cout<<"\n";

}

return 0;

}



2.Write a C program to find suffix of a string.

#include<iostream>

#include<string.h>

using namespace std;

int main(){

string s;

cout<<"enter a string:\n";

cin>>s;

int length=s.length();

int l=length-1;

while(l>=0)

{ for(int j=length-1;j>=l;j--){

cout<<s[j];

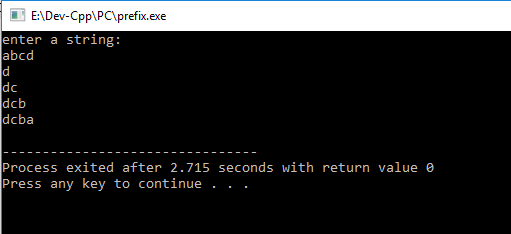
} l--;

cout<<"\n";

}

return 0;

}



EXPERIMENT 3:

1.Write a C program to implement CYK algorithm.

#include<iostream>

#include<cstring>

#include<algorithm>

#include<string>

#include<cassert>

#include<iomanip>

using namespace std;

#define MAX 100

#define for(i,a,b) for(i=a;i<b; i++)

string gram[MAX][MAX]; //to store entered grammar

string dpr[MAX];

int p,np; //np-> number of productions

inline string concat( string a, string b) //concatenates unique non-terminals

{

int i;

string r=a;

for(i,0,b.length())

if(r.find(b[i]) > r.length())

r+=b[i];

return (r);

}

inline void break\_gram(string a) //seperates right hand side of entered grammar

{

int i;

p=0;

while(a.length())

{

i=a.find("|");

if(i>a.length())

{

dpr[p++] = a;

a="";

}

else

{

dpr[p++] = a.substr(0,i);

a=a.substr(i+1,a.length());

}

}

}

inline int lchomsky(string a) //checks if LHS of entered grammar is in CNF

{

if(a.length()==1 && a[0]>='A' && a[0]<='Z')

return 1;

return 0;

}

inline int rchomsky(string a) //checks if RHS of grammar is in CNF

{

if (a.length() == 1 && a[0]>='a' && a[0] <='z')

return 1;

if (a.length()==2 && a[0]>='A' && a[0]<='Z' && a[1]>='A' && a[1]<='Z' )

return 1;

return 0;

}

inline string search\_prod(string p) //returns a concatenated string of variables which can produce string p

{

int j,k;

string r="";

for(j,0,np)

{

k=1;

while(gram[j][k] != "")

{

if(gram[j][k] == p)

{

r=concat(r,gram[j][0]);

}

k++;

}

}

return r;

}

inline string gen\_comb(string a, string b) //creates every combination of variables from a and b . For eg: BA \* AB = {BA, BB, AA, BB}

{

int i,j;

string pri=a,re="";

for(i,0,a.length())

for(j,0,b.length())

{

pri="";

pri=pri+a[i]+b[j];

re=re+search\_prod(pri); //searches if the generated productions can be created or not

}

return re;

}

int main()

{

int i,pt,j,l,k;

string a,str,r,pr,start;

cout<<"\nEnter the start Variable ";

cin >> start;

cout<<"\nNumber of productions ";

cin >> np;

for(i,0,np)

{

cin >> a;

pt=a.find("->");

gram[i][0] = a.substr(0,pt);

if (lchomsky(gram[i][0]) == 0)

{

cout<<"\nGrammar not in Chomsky Form";

abort();

}

a = a.substr(pt+2, a.length());

break\_gram(a);

for(j,0,p)

{

gram[i][j+1]=dpr[j];

if (rchomsky(dpr[j]) == 0)

{

cout<<"\nGrammar not in Chomsky Form";

abort();

}

}

}

string matrix[MAX][MAX],st;

cout<<"\nEnter string to be checked : ";

cin >> str;

for(i,0,str.length()) //Assigns values to principal diagonal of matrix

{

r="";

st = "";

st+=str[i];

for(j,0,np)

{

k=1;

while(gram[j][k] != "")

{

if(gram[j][k] == st)

{

r=concat(r,gram[j][0]);

}

k++;

}

}

matrix[i][i]=r;

}

int ii,kk;

for(k,1,str.length()) //Assigns values to upper half of the matrix

{

for(j,k,str.length())

{

r="";

for(l,j-k,j)

{

pr = gen\_comb(matrix[j-k][l],matrix[l+1][j]);

r = concat(r,pr);

}

matrix[j-k][j] = r;

}

}

for(i,0,str.length()) //Prints the matrix

{

k=0;

l=str.length()-i-1;

for(j,l,str.length())

{

cout<<setw(5)<<matrix[k++][j]<<" ";

}

cout<<endl;

}

int f=0;

for(i,0,start.length())

if(matrix[0][str.length()-1].find(start[i]) <= matrix[0][str.length()-1].length()) //Checks if last element of first row contains a Start variable

{

cout<<"String can be generated\n";

return 0;

}

cout<<"String cannot be generated\n";

return 0;

}

