INDEX

1. WAP to implement a Regular Expression
2. WAP to implement Mealy and Moore machines
3. WAP to implement the conversion of an NFA to a DFA
4. WAP to implement a Regular Grammar
5. WAP to implement a CFG grammar
6. WAP to to find out the First and Follow
7. Verify whether a given CFG is suitable for LL(1) parsing or not
8. To generate LL(1) parsing table for a given CFG and also perform LL(1) parsing using the same table
9. *WAP to implement a Regular expression. The program should read a RE through a file and should check whether a string given from the console is acceptable by the given RE or not.*

#include <iostream>

#include <fstream>

#include<string>

using namespace std;

int main()

{

ifstream infile;

infile.open("fAutomata.txt");

int initial,i=0,k=0,l=0,nf,nr,nc;

char ch;

//cout<<"helloworld";

int automata[100][100],final[100];

string rline;

infile.get(ch);

initial=(int)ch-48;

getline(infile,rline);

int j=0;

int len;

/\*

len=rline.size();

for(j=0;j<len;j++)

{

//final[j]=rline[i];

// cout<<rline[i];

if(rline[j]!=',')

{

cout<<"abdul";

final[k]=rline[j];

cout<<final[k];

k++;

}

}

cout<<"\n";\*/

//i++;

// cout<<rline;

cout<<initial<<"\n";

i=0;

while(getline(infile,rline)){

len=rline.size();

k=0;

for(j=0;j<len;j++)

{

if(l==0)

{

if(rline[j]!=',')

{

final[k]=rline[j]-'0';

// cout<<final[k];

k++;

}

i=-1;

nf=k;

}

else{

//final[j]=rline[i];

// cout<<rline[i];

if(rline[j]!=' ')

{

if(rline[j]=='-')

automata[i][k]=-1;

else

automata[i][k]=rline[j]-'0';

cout<<automata[i][k]<<" ";

k++;

}

}

}

l=1;

cout<<"\n";

i++;

}

//cout<<automata[0][0];

string inputVal;

int inputvalIndex;

while(1){

cout<<"Enter the string\n";

cin>>inputVal;

i=0;

j=initial;k=0;

int result;

while(inputVal[i]!='\0')

{

inputvalIndex=inputVal[i]-'a';

//cout<<inputvalIndex<<"\n";

result= automata[j][inputvalIndex];

// cout<<"\n"<<automata[j][inputvalIndex];

j=result;

if(j==-1)

break;

i++;

}

int flag=0;

//cout<<j;

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

{

if(j==-1)

break;

if(j==final[i])

flag=1;

}

if(flag==1)

{

cout<<"accept\n";

}

else

cout<<"reject\n";

}

return 0;

}



1. *Write a program to implement Mealy and Moore Machines.*

Mealy machine

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

using namespace std;

#define MAX\_ROWS 5

#define MAX\_COLS 5

/\*

INPUT FILE FORMAT:

1/0 0/1

1/1 0/0

\*/

#define INPUT\_FILE "input.txt"

int state\_matrix[MAX\_ROWS][MAX\_COLS];

int output\_matrix[MAX\_ROWS][MAX\_COLS];

////////////////////////////////////////////////////////////////////

void print\_arr(int arr[], int size) {

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

cout << arr[i] << " ";

}

cout << endl;

}

void print\_matrix(int mat[][MAX\_COLS], int rows, int cols) {

for (int i = 0; i < rows; ++i) {

print\_arr(mat[i], cols);

}

cout << endl;

}

////////////////////////////////////////////////////////////////////

// Read file and fill matrices

void read\_file() {

ifstream file(INPUT\_FILE);

if (!file.is\_open()) {

cerr << "Couldn't open input file: " << INPUT\_FILE;

}

string line;

int i = 0;

while (getline(file, line)) {

string cell;

istringstream line\_stream(line);

int j = 0;

while(getline(line\_stream, cell, ' ')) {

// Cell is of the form 'state/output'

// We store it by converting to int

state\_matrix[i][j] = cell[0] - 48;

output\_matrix[i][j] = cell[2] - 48;

j++;

}

i++;

}

}

int main(int argc, char const \*argv[])

{

read\_file();

// Initial State

int cur\_state = 0;

// Current Output

int cur\_output = 0;

// Input String

string input;

cout << "Enter input string: ";

cin >> input;

cout << endl;

// Header

cout << "Output: " << endl << endl;

cout << "S - O" << endl;

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

int length = input.length();

for (int i = 0; i < length; ++i)

{

// Convert char '1'/'0' to int 1/0

int cur\_input = input[i] - '0';

cur\_state = state\_matrix[cur\_state][cur\_input];

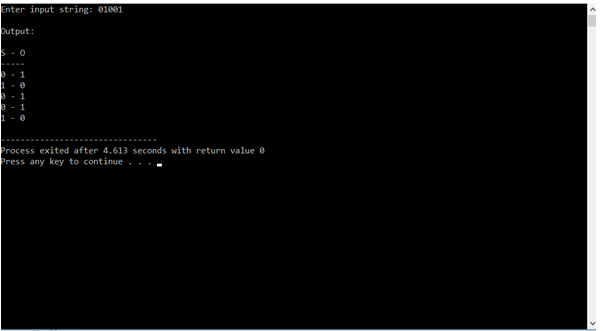
cur\_output = output\_matrix[cur\_state][cur\_input];

cout << cur\_state << " - " << cur\_output << endl;

}

return 0;

}



Moore machine

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

using namespace std;

#define MAX\_ROWS 5

#define MAX\_COLS 5

/\*

INPUT FILE FORMAT:

states, output

0 2, 1

1 1, 0

1 0, 1

\*/

#define INPUT\_FILE "input.txt"

int state\_matrix[MAX\_ROWS][MAX\_COLS];

int output[MAX\_ROWS];

////////////////////////////////////////////////////////////////////

void print\_arr(int arr[], int size) {

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

cout << arr[i] << " ";

}

cout << endl;

}

void print\_matrix(int mat[][MAX\_COLS], int rows, int cols) {

for (int i = 0; i < rows; ++i) {

print\_arr(mat[i], cols);

}

cout << endl;

}

////////////////////////////////////////////////////////////////////

// Read file and fill matrices

void read\_file() {

ifstream file(INPUT\_FILE);

if (!file.is\_open()) {

cerr << "Couldn't open input file: " << INPUT\_FILE;

}

string line;

int i = 0;

while (getline(file, line)) {

string cell;

// Remove the last three characters from the line; for eg: ', 2'

istringstream line\_stream(line.substr(0, line.length() - 3));

int j = 0;

while(getline(line\_stream, cell, ' ')) {

// Cell is of the form 'state'

// We store it by converting to int

state\_matrix[i][j] = cell[0] - 48;

j++;

}

// The last character contains the output of that state

output[i] = line[line.length() - 1] - 48;

i++;

}

}

int main(int argc, char const \*argv[])

{

read\_file();

// print\_matrix(state\_matrix, 3, 2);

// print\_arr(output, 3);

// Initial State

int cur\_state = 0;

// Current Output

int cur\_output = 0;

// Input String

string input;

cout << "Enter input string: ";

cin >> input;

cout << endl;

// Header

cout << "Output: " << endl << endl;

cout << "I - S - O" << endl;

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

// Print intial values

// cout << "-" << " - " << cur\_state << " - " << cur\_output << endl;

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

int length = input.length();

for (int i = 0; i < length; ++i)

{

// Convert char '1'/'0' to int 1/0

int cur\_input = int(input[i]) - 48;

cur\_state = state\_matrix[cur\_state][cur\_input];

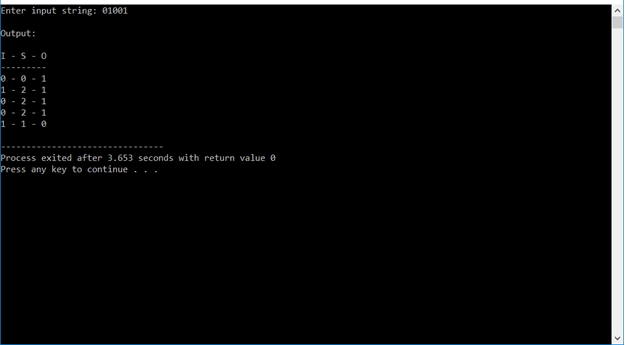
cur\_output = output[cur\_state];

cout << cur\_input << " - " << cur\_state << " - " << cur\_output << endl;

}

return 0;

}



1. *Conversion of NFA to DFA*

#include <stdio.h>

#include <string.h>

#define STATES 256

#define SYMBOLS 20

int N\_symbols;

int NFA\_states;

char \*NFAtab[STATES][SYMBOLS];

int DFA\_states; /\* number of DFA states \*/

int DFAtab[STATES][SYMBOLS];

/\*Print state-transition table.\*/

void put\_dfa\_table(

int tab[][SYMBOLS], /\* DFA table \*/

int nstates, /\* number of states \*/

int nsymbols) /\* number of input symbols \*/

{

int i, j;

puts("STATE TRANSITION TABLE");

/\* input symbols: '0', '1', ... \*/

printf(" | ");

for (i = 0; i < nsymbols; i++) printf(" %c ", '0'+i);

printf("\n-----+--");

for (i = 0; i < nsymbols; i++) printf("-----");

printf("\n");

for (i = 0; i < nstates; i++) {

printf(" %c | ", 'A'+i); /\* state \*/

for (j = 0; j < nsymbols; j++)

printf(" %c ", 'A'+tab[i][j]);

printf("\n");

}

}

/\*Initialize NFA table.\*/

void init\_NFA\_table()

{

/\*

NFA table for ex.21 at p.76

NFAtab[0][0] = "01";

NFAtab[0][1] = "0";

NFAtab[1][0] = "";

NFAtab[1][1] = "01";

NFA\_states = 2;

DFA\_states = 0;

N\_symbols = 2;

\*/

/\*

NFA table for ex.17 at p.72

\*/

NFAtab[0][0] = "12";

NFAtab[0][1] = "13";

NFAtab[1][0] = "12";

NFAtab[1][1] = "13";

NFAtab[2][0] = "4";

NFAtab[2][1] = "";

NFAtab[3][0] = "";

NFAtab[3][1] = "4";

NFAtab[4][0] = "4";

NFAtab[4][1] = "4";

NFA\_states = 5;

DFA\_states = 0;

N\_symbols = 2;

}

/\*String 't' is merged into 's' in an alphabetical order.\*/

void string\_merge(char \*s, char \*t)

{

char temp[STATES], \*r=temp, \*p=s;

while (\*p && \*t) {

if (\*p == \*t) {

\*r++ = \*p++; t++;

} else if (\*p < \*t) {

\*r++ = \*p++;

} else

\*r++ = \*t++;

}

\*r = '\0';

if (\*p) strcat(r, p);

else if (\*t) strcat(r, t);

strcpy(s, temp);

}

/\*Get next-state string for current-state string.\*/

void get\_next\_state(char \*nextstates, char \*cur\_states,

char \*nfa[STATES][SYMBOLS], int n\_nfa, int symbol)

{

int i;

char temp[STATES];

temp[0] = '\0';

for (i = 0; i < strlen(cur\_states); i++)

string\_merge(temp, nfa[cur\_states[i]-'0'][symbol]);

strcpy(nextstates, temp);

}

int state\_index(char \*state, char statename[][STATES], int \*pn)

{

int i;

if (!\*state) return -1; /\* no next state \*/

for (i = 0; i < \*pn; i++)

if (!strcmp(state, statename[i])) return i;

strcpy(statename[i], state); /\* new state-name \*/

return (\*pn)++;

}

/\*

Convert NFA table to DFA table.

Return value: number of DFA states.

\*/

int nfa\_to\_dfa(char \*nfa[STATES][SYMBOLS], int n\_nfa,

int n\_sym, int dfa[][SYMBOLS])

{

char statename[STATES][STATES];

int i = 0; /\* current index of DFA \*/

int n = 1; /\* number of DFA states \*/

char nextstate[STATES];

int j;

strcpy(statename[0], "0"); /\* start state \*/

for (i = 0; i < n; i++) { /\* for each DFA state \*/

for (j = 0; j < n\_sym; j++) { /\* for each input symbol \*/

get\_next\_state(nextstate, statename[i], nfa, n\_nfa, j);

dfa[i][j] = state\_index(nextstate, statename, &n);

}

}

return n; /\* number of DFA states \*/

}

void main()

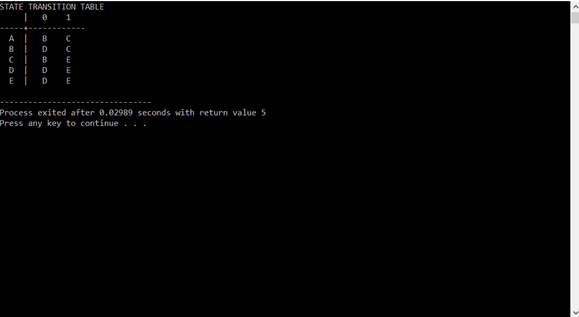
{

init\_NFA\_table();

DFA\_states = nfa\_to\_dfa(NFAtab, NFA\_states, N\_symbols, DFAtab);

put\_dfa\_table(DFAtab, DFA\_states, N\_symbols);

}



1. *Implement regular grammar*

/\* C program to check given grammar is Regular Grammar or not. \*/

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<string.h>

int i,j,k,l,m,n=0,o,p,nv,z=0,t,x=0;

char str[10],temp[10],temp2[10],temp3[10];

struct prod

{

char lhs[10],rhs[10][10];

int n;

}pro[10];

void findter()

{

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

{

if(temp[i]==pro[k].lhs[0])

{

for(t=0;t<pro[k].n;t++)

{

for(x=0;x<10;x++)

temp2[x]='\0';

for(l=i+1;l<strlen(temp);l++)

temp2[l-i-1]=temp[l];

temp[i]='\0';

for(l=0;l<strlen(pro[k].rhs[t]);l++)

temp[i+l]=pro[k].rhs[t][l];

strcat(temp,temp2);

if(str[i]==temp[i])

return;

}

}

}

}

int main()

{

FILE \*f;

for(i=0;i<10;i++)

pro[i].n=0;

f=fopen("tab3.txt","r");

while(!feof(f))

{

fscanf(f,"%s",pro[n].lhs);

if(n>0)

{

if( strcmp(pro[n].lhs,pro[n-1].lhs) == 0 )

{

pro[n].lhs[0]='\0';

fscanf(f,"%s",pro[n-1].rhs[pro[n-1].n]);

pro[n-1].n++;

continue;

}

}

fscanf(f,"%s",pro[n].rhs[pro[n].n]);

pro[n].n++;

n++;

}

n--;

printf("\n\nTHE GRAMMAR IS AS FOLLOWS\n\n");

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

for(j=0;j<pro[i].n;j++)

printf("%s -> %s\n",pro[i].lhs,pro[i].rhs[j]);

o=0;

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

{

for(j=0;j<pro[i].n;j++)

if( pro[i].rhs[j][0]>=65 && pro[i].rhs[j][0]<=90 )

{

o=1;

break;

}

if(o==1)

break;

}

if(i==n)

printf("\n\nTHE GRAMMAR is a REGULAR GRAMMAR !!!");

else

{

printf("\n\nTHE GRAMMAR is NOT a REGULAR GRAMMAR !!!");

exit(1);

}

while(1)

{

for(x=0;x<10;x++)

str[x]='\0';

printf("\n\nENTER ANY STRING ( 0 for EXIT ) : ");

scanf("%s",str);

if(str[0]=='0')

exit(1);

for(j=0;j<pro[0].n;j++)

{

for(x=0;x<10;x++)

temp[x]='\0';

strcpy(temp,pro[0].rhs[j]);

m=0;

for(i=0;i<strlen(str);i++)

{

if(str[i]==temp[i])

m++;

else if(str[i]!=temp[i] && temp[i]>=65 && temp[i]<=90)

{

findter();

if(str[i]==temp[i])

m++;

}

}

for(x=0;x<10;x++)

temp3[x]='\0';

strcpy(temp3,temp);

temp3[strlen(temp)-1]='\0';

//printf("%s",temp);

if(m==strlen(str) && strcmp(temp3,str)==0 && strlen(temp3)!=1)

{

printf("\n\nTHE STRING can be PARSED !!!");

break;

}

if(m==strlen(str) && strlen(str)==strlen(temp))

{

printf("\n\nTHE STRING can be PARSED !!!");

break;

}

}

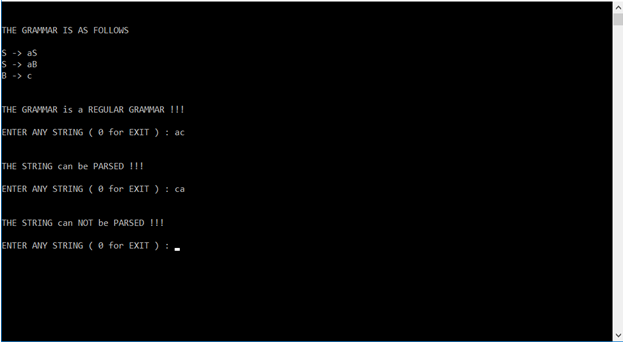
if(j==pro[0].n)

printf("\n\nTHE STRING can NOT be PARSED !!!");

}

printf("\n\n");

}



1. *Implement CFG*

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

int i,j,k,l,m,n=0,o,p,nv,z=0,t,x=0;

char str[10],temp[20],temp2[20],temp3[20];

struct prod

{

char lhs[10],rhs[10][10];

int n;

}pro[10];

void findter()

{

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

{

if(temp[i]==pro[k].lhs[0])

{

for(t=0;t<pro[k].n;t++)

{

for(l=0;l<20;l++)

temp2[l]='\0';

for(l=i+1;l<strlen(temp);l++)

temp2[l-i-1]=temp[l];

for(l=i;l<20;l++)

temp[l]='\0';

for(l=0;l<strlen(pro[k].rhs[t]);l++)

temp[i+l]=pro[k].rhs[t][l];

strcat(temp,temp2);

if(str[i]==temp[i])

return;

else if(str[i]!=temp[i] && temp[i]>=65 && temp[i]<=90)

break;

}

break;

}

}

if(temp[i]>=65 && temp[i]<=90)

findter();

}

int main()

{

FILE \*f;

for(i=0;i<10;i++)

pro[i].n=0;

f=fopen("input.txt","r");

while(!feof(f))

{

fscanf(f,"%s",pro[n].lhs);

if(n>0)

{

if( strcmp(pro[n].lhs,pro[n-1].lhs) == 0 )

{

pro[n].lhs[0]='\0';

fscanf(f,"%s",pro[n-1].rhs[pro[n-1].n]);

pro[n-1].n++;

continue;

}

}

fscanf(f,"%s",pro[n].rhs[pro[n].n]);

pro[n].n++;

n++;

}

n--;

printf("\nThe grammar detected is:\n");

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

for(j=0;j<pro[i].n;j++)

printf("%s -> %s\n",pro[i].lhs,pro[i].rhs[j]);

for(l=0;l<10;l++)

str[0]=NULL;

printf("\n\nEnter a string:\n");

scanf("%s",str);

for(j=0;j<pro[0].n;j++)

{

for(l=0;l<20;l++)

temp[l]=NULL;

strcpy(temp,pro[0].rhs[j]);

m=0;

for(i=0;i<strlen(str);i++)

{

if(str[i]==temp[i])

m++;

else if(str[i]!=temp[i] && temp[i]>=65 && temp[i]<=90)

{

findter();

if(str[i]==temp[i])

m++;

}

else if( str[i]!=temp[i] && (temp[i]<65 || temp[i]>90) )

break;

}

if(m==strlen(str) && strlen(str)==strlen(temp))

{

printf("\nIt satisfies the Grammar.");

break;

}

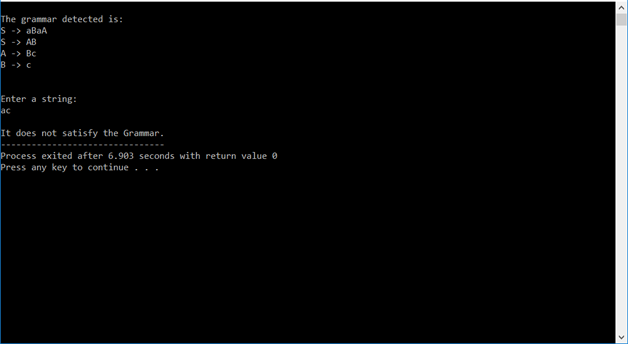
}

if(j==pro[0].n)

printf("\nIt does not satisfy the Grammar.");

return 0;

}

**

*6. First and Follow*

#include<stdio.h>

#include<string.h>

int i,j,l,m,n=0,o,p,nv,z=0,x=0;

char str[10],temp,temp2[10],temp3[20],\*ptr;

struct prod

{

char lhs[10],rhs[10][10],ft[10],fol[10];

int n;

}pro[10];

void findter()

{

int k,t;

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

{

if(temp==pro[k].lhs[0])

{

for(t=0;t<pro[k].n;t++)

{

if( pro[k].rhs[t][0]<65 || pro[k].rhs[t][0]>90 )

pro[i].ft[strlen(pro[i].ft)]=pro[k].rhs[t][0];

else if( pro[k].rhs[t][0]>=65 && pro[k].rhs[t][0]<=90 )

{

temp=pro[k].rhs[t][0];

if(temp=='S')

pro[i].ft[strlen(pro[i].ft)]='#';

findter();

}

}

break;

}

}

}

void findfol()

{

int k,t,p1,o1,chk;

char \*ptr1;

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

{

chk=0;

for(t=0;t<pro[k].n;t++)

{

ptr1=strchr(pro[k].rhs[t],temp);

if( ptr1 )

{

p1=ptr1-pro[k].rhs[t];

if(pro[k].rhs[t][p1+1]>=65 && pro[k].rhs[t][p1+1]<=90)

{

for(o1=0;o1<n;o1++)

if(pro[o1].lhs[0]==pro[k].rhs[t][p1+1])

{

strcat(pro[i].fol,pro[o1].ft);

chk++;

}

}

else if(pro[k].rhs[t][p1+1]=='\0')

{

temp=pro[k].lhs[0];

if(pro[l].rhs[j][p]==temp)

continue;

if(temp=='S')

strcat(pro[i].fol,"$");

findfol();

chk++;

}

else

{

pro[i].fol[strlen(pro[i].fol)]=pro[k].rhs[t][p1+1];

chk++;

}

}

}

if(chk>0)

break;

}

}

int main()

{

FILE \*f;

//clrscr();

for(i=0;i<10;i++)

pro[i].n=0;

f=fopen("firstFollowInput.txt","r");

while(!feof(f))

{

fscanf(f,"%s",pro[n].lhs);

if(n>0)

{

if( strcmp(pro[n].lhs,pro[n-1].lhs) == 0 )

{

pro[n].lhs[0]='\0';

fscanf(f,"%s",pro[n-1].rhs[pro[n-1].n]);

pro[n-1].n++;

continue;

}

}

fscanf(f,"%s",pro[n].rhs[pro[n].n]);

pro[n].n++;

n++;

}

printf("\n\nTHE GRAMMAR IS AS FOLLOWS\n\n");

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

for(j=0;j<pro[i].n;j++)

printf("%s -> %s\n",pro[i].lhs,pro[i].rhs[j]);

pro[0].ft[0]='#';

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

{

for(j=0;j<pro[i].n;j++)

{

if( pro[i].rhs[j][0]<65 || pro[i].rhs[j][0]>90 )

{

pro[i].ft[strlen(pro[i].ft)]=pro[i].rhs[j][0];

}

else if( pro[i].rhs[j][0]>=65 && pro[i].rhs[j][0]<=90 )

{

temp=pro[i].rhs[j][0];

if(temp=='S')

pro[i].ft[strlen(pro[i].ft)]='#';

findter();

}

}

}

printf("\n\nFIRST\n");

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

{

printf("\n%s -> ",pro[i].lhs);

for(j=0;j<strlen(pro[i].ft);j++)

{

for(l=j-1;l>=0;l--)

if(pro[i].ft[l]==pro[i].ft[j])

break;

if(l==-1)

printf("%c",pro[i].ft[j]);

}

}

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

temp2[i]=pro[i].lhs[0];

pro[0].fol[0]='$';

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

{

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

{

for(j=0;j<pro[i].n;j++)

{

ptr=strchr(pro[l].rhs[j],temp2[i]);

if( ptr )

{

p=ptr-pro[l].rhs[j];

if(pro[l].rhs[j][p+1]>=65 && pro[l].rhs[j][p+1]<=90)

{

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

if(pro[o].lhs[0]==pro[l].rhs[j][p+1])

strcat(pro[i].fol,pro[o].ft);

}

else if(pro[l].rhs[j][p+1]=='\0')

{

temp=pro[l].lhs[0];

if(pro[l].rhs[j][p]==temp)

continue;

if(temp=='S')

strcat(pro[i].fol,"$");

findfol();

}

else

pro[i].fol[strlen(pro[i].fol)]=pro[l].rhs[j][p+1];

}

}

}

}

printf("\n\nFOLLOW\n");

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

{

printf("\n%s -> ",pro[i].lhs);

for(j=0;j<strlen(pro[i].fol);j++)

{

for(l=j-1;l>=0;l--)

if(pro[i].fol[l]==pro[i].fol[j])

break;

if(l==-1)

printf("%c",pro[i].fol[j]);

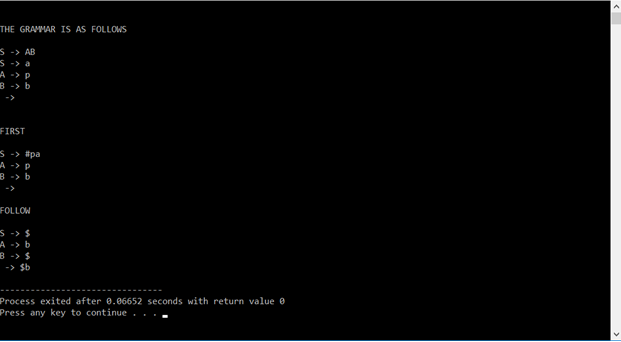
}

}

printf("\n");

//getch();

}



7. *WAP to verify whether a given CFG is suitable for LL(1) parsing or not.*

#include <iostream>

#include <vector>

#include <fstream>

#include <string>

#include<algorithm>

using namespace std;

int num\_of\_productions;

vector<string> productions;

vector<char> terminals;

vector<char> nonTerminals;

vector<vector <char> > firsts;

vector<vector <char> > follows;

vector<vector <string> > parsingTable;

vector<int> flag;

int globalCount=0; //to counter chain productions

char ttc;

char startSymbol;

int positionOfNonTerminal(char c){

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

if(nonTerminals[i]==c){

return i;

}

}

return -23;

}

int positionOfTerminal(char c){

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

if(terminals[i]==c){

return i;

}

}

return -23;

}

int isInTerminal(char c){//Check if a value is a valid Terminal

if(find(terminals.begin(),terminals.end(),c)!=terminals.end()){

return 1;

}

return 0;

}

int isInNonTerminal(char c){//Check if a value is a valid NonTerminal

if(find(nonTerminals.begin(),nonTerminals.end(),c)!=nonTerminals.end()){

return 1;

}

return 0;

}

int isTerminal(char tem){//Assume Capital Letters As non terminals and small letters as terminals

if(tem>='A' && tem<='Z'){

return 0;

}else{

return 1;

}

}

int prepareListOfCharacter(){//Prepare List Of Valid Characters

int f=0;

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

string tempString=productions[i];

int tempStringLen = tempString.length();

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

char t = tempString[j];

if(!(t==' ')){

if(isInNonTerminal(t)||isInTerminal(t)){

continue;

}else{

//Check if Terminal Or Non Terminal.

// Add in the respective Vector

if(isTerminal(t)){

if(t=='#'){

f=1;

continue;

}

terminals.push\_back(t);

}else{

nonTerminals.push\_back(t);

flag.push\_back(0);

vector<char> tV;

tV.push\_back('@');

firsts.push\_back(tV);

follows.push\_back(tV);

}

}

}

}

}

if(f==1){

terminals.push\_back('#');

}

return 1;

}

int ifExists(int i,char c){

for(int j=0;j<firsts[i].size();j++){

if(firsts[i][j]==c){

return 1;

}

}

return 0;

}

int readProductions(){

//Function to read the productions from the input file

//READING BEGINS

string fileName;

cout<<"Enter Filename : ";

getline(cin, fileName);

getline(cin, fileName);

ifstream input(fileName.c\_str());

string tempString;

input>>num\_of\_productions;

getline(input, tempString, '\n');

while (getline(input, tempString, '\n')) {

productions.push\_back(tempString);

}

cout<<"DATA FROM INPUT FILE "<<endl;

cout<<"Num Of Productions : "<<num\_of\_productions<<endl;

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

cout<<productions[i]<<endl;

}

//READING ENDS

prepareListOfCharacter();

cout<<"\nTerminals : ";

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

cout<<terminals[i]<<" ";

}

cout<<"\nNon Terminals : ";

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

cout<<nonTerminals[i]<<" ";

}

cout<<"...";

cout<<"\n";

return 1;

}

int isFlagTrue(int i){

return flag[i];

}

int firstCalcB(char c, int i){

//TODO

//Rectify for input.txt chain productions

if((ttc==c)&&(globalCount!=0)){

if(!ifExists(i,'#')){

firsts[i].push\_back('#');

}

return 1;

}

globalCount++;

//cout<<"\ninside firstcalcb for"<<c;

int posNonTerm = positionOfNonTerminal(c);

//cout<<"\tPosition : "<<posNonTerm;

if(isFlagTrue(posNonTerm)){

int tempFlag=0;

//Already Calculated the First

vector<char> tempFirsts;

tempFirsts = firsts[posNonTerm];

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

char tempC = tempFirsts[j];

if(tempC=='@'){

continue;

}else if(tempC == '#'){

tempFlag=1;

}else{

if(!ifExists(i,tempC)){

firsts[i].push\_back(tempC);

}

}

}

if(tempFlag==1){

return 22;

}else{

return 1;

}

}else{

//Need To Calculate Now

vector<string> tempProductions;

int tempFlag3 = 0;

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

if(productions[j][0]==c){

tempProductions.push\_back(productions[j]);

}

}

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

string tempProduction = tempProductions[j];

if(isTerminal(tempProduction[2])){

if(tempProduction[2]=='#'){

tempFlag3 = 1;

}else{

if(!ifExists(i,tempProduction[2])){

firsts[i].push\_back(tempProduction[2]);

}

}

}else{

int k=2;

int tempFlag2=22;

while(tempFlag2==22){

//cout<<"\nCalling for tempProduction["<<k<<"] = "<<tempProduction[k];

if(isTerminal(tempProduction[k])){

if(!ifExists(i,tempProduction[k])){

firsts[i].push\_back(tempProduction[k]);

}

tempFlag2=1;

}

tempFlag2=firstCalcB(tempProduction[k++],i);

//cout<<"\nTempFlag2 = "<<tempFlag2<<endl;

if(tempFlag2==22){

if(k==tempProduction.length()){

if(!ifExists(i,'#')){

firsts[i].push\_back('#');

}

}

}

}

}

}

if(tempFlag3==1){

return 22;

}else{

return 1;

}

}

return 1;

}

int printFirsts(){

cout<<"NonTerminal\tFirsts\n";

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

vector<char> tempFirsts = firsts[i];

cout<<nonTerminals[i]<<"\t\t";

for(int j=0;j<tempFirsts.size()-1;j++){

if(tempFirsts[j]=='@'){

continue;

}

cout<<tempFirsts[j]<<", ";

}

int fin=tempFirsts.size()-1;

if(fin>=0){

if(tempFirsts[fin]!='@'){

cout<<tempFirsts[fin]<<endl;

}

}

}

return 1;

}

int exists(char c, int i){

for(int j=0;j<follows[i].size();j++){

if(follows[i][j]==c){

return 1;

}

}

return 0;

}

int followCalcB(char c, int i){

int p = positionOfNonTerminal(c);

if(flag[p]==1){

for(int j=0;j<follows[p].size();j++){

if(!exists(follows[p][j],i)){

follows[i].push\_back(follows[p][j]);

}

}

}else{

//cout<<"\n\nFor Character : "<<c;

if(c==startSymbol){

if(!exists('$',i)){

//cout<<"\n\t\tAdding $ to follow";

follows[i].push\_back('$');

}

}

if((ttc==c)&&(globalCount!=0)){

return 1;

}

vector<string> tempProductions;

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

for(int k=2;k<productions[j].length();k++){

if(productions[j][k]==c){

tempProductions.push\_back(productions[j]);

}

}

}

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

string production=tempProductions[j];

int pos=0;

for(int k=2;k<production.length();k++){

if(production[k]==c){

pos=k;

//cout<<"\n\t"<<c<<" Found At Position : "<<pos<<" In String : "<<production;

while(true){

char tempChar = production[pos];

if(pos==production.length()-1){

//End Of String Reached

if(isTerminal(tempChar)){

if(!exists(tempChar,i)){

//cout<<"\n\t\tAdding "<<tempChar<<" to follow";

follows[i].push\_back(tempChar);

}

}else{

if(production[0]!=tempChar){

ttc=production[0];

globalCount++;

followCalcB(production[0],i);

}

}

break;

}else if(isTerminal(production[pos+1])){

if(!exists(production[pos+1],i)){

//cout<<"\n\t\tAdding "<<production[pos+1]<<" to follow";

follows[i].push\_back(production[pos+1]);

}

break;

}else{

int tempFlag=0;

vector<char> tempFirst = firsts[positionOfNonTerminal(production[pos+1])];

for(int l=0;l<tempFirst.size();l++){

char ch = tempFirst[l];

if(ch!='#'){

if(!exists(ch,i)){

//cout<<"\n\t\tAdding "<<ch<<" to follow";

follows[i].push\_back(ch);

}

}else{

tempFlag=1;

}

}

if(tempFlag==1){

pos++;

}else{

break;

}

}

}

}

}

}

}

return 1;

}

int printFollows(){

cout<<"\nNonTerminal\tFollows\n";

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

vector<char> tempFollows = follows[i];

cout<<nonTerminals[i]<<"\t\t";

for(int j=0;j<tempFollows.size()-1;j++){

if(tempFollows[j]=='@'){

continue;

}

cout<<tempFollows[j]<<", ";

}

int fin=tempFollows.size()-1;

if(fin>=0){

if(tempFollows[fin]!='@'){

cout<<tempFollows[fin]<<endl;

}

}

}

return 1;

}

int main(){

cout<<"# represents epsilon"<<endl;

cout<<"\n\nEnter The Start Symbol : ";

cin>>startSymbol;

//Read All The Productions

readProductions();

//Calculating Firsts BEGINS

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

//cout<<"\tFirst For Non Terminal["<<i<<"] : "<<nonTerminals[i]<<endl;

int retVal = 0;

ttc=nonTerminals[i];

retVal = firstCalcB(nonTerminals[i],i);

globalCount=0;

if(retVal==22){

firsts[i].push\_back('#');

}

flag[i]=1;

}

printFirsts();

//Calculating Firsts ENDS

//Reset Flag

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

flag[i]=0;

}

//Calculating Follows BEGINS

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

followCalcB(nonTerminals[i],i);

flag[i]=1;

globalCount=0;

}

printFollows();

//Calculating Follows ENDS

// First And Follow Have Been Computed.

//Now Preparing the Parser Table

//Blank Parsing Table

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

vector<string> tempRow;

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

tempRow.push\_back(" ");

}

parsingTable.push\_back(tempRow);

}

//Adding First Row Of Parsing Table

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

parsingTable[0][i+1]=terminals[i];

}

//Adding First Column Of Parsing Table

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

parsingTable[i+1][0]=nonTerminals[i];

}

//Filling the values in parsing table

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

string production = productions[i];

if(production[2]!='#'){

if(isTerminal(production[2])){

int col = positionOfTerminal(production[2]);

if(col==-23){

cout<<production[2];

cout<<"ERROR CODE 001\n";

return 0;

}

int row = positionOfNonTerminal(production[0]);

if(row==-23){

cout<<production[0];

cout<<"ERROR CODE 002\n";

return 0;

}

parsingTable[row+1][col+1]=production;

}else{

vector<char> tempFirst;

int row = positionOfNonTerminal(production[0]);

if(row==-23){

cout<<production[0];

cout<<"ERROR CODE 003\n";

return 0;

}

tempFirst = firsts[row];

for(int k=0;k<tempFirst.size();k++){

if(tempFirst[k]=='@'){

continue;

}

if(tempFirst[k]!='#'){

int col = positionOfTerminal(tempFirst[k]);

if(col==-23){

cout<<tempFirst[k];

cout<<"ERROR CODE 004\n";

return 0;

}

parsingTable[row+1][col+1]=production;

}

}

}

}else{

vector<char> tempFollow;

int row = positionOfNonTerminal(production[0]);

if(row==-23){

cout<<production[0];

cout<<"ERROR CODE 005\n";

return 0;

}

tempFollow = follows[row];

for(int k=0;k<tempFollow.size();k++){

if(tempFollow[k]=='@'){

continue;

}

int col;

if(tempFollow[k]=='$'){

col = terminals.size()-1;

}else{

col = positionOfTerminal(tempFollow[k]);

if(col==-23){

cout<<tempFollow[k];

cout<<"ERROR CODE 006\n";

return 0;

}

}

parsingTable[row+1][col+1]=production;

}

}

}

parsingTable[0][terminals.size()]="$";

//Printing the parsing table

cout<<"\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PARSING TABLE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";

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

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

cout<<parsingTable[i][j]<<"\t|";

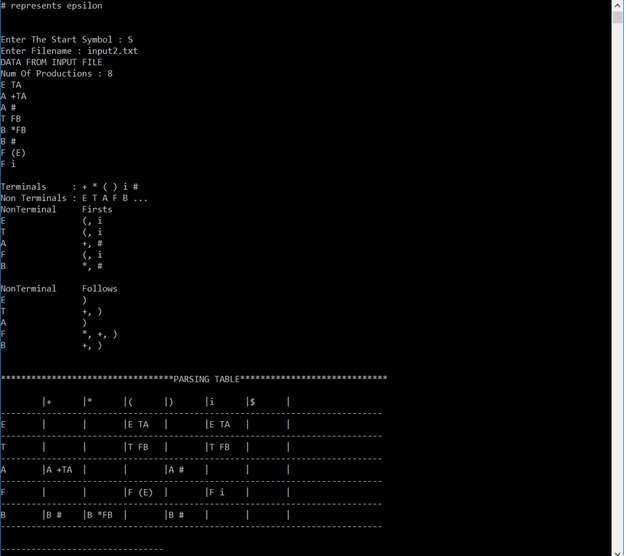
}

cout<<"\n---------------------------------------------------------------------------\n";

}

return 1;

}



8. *WAP to generate LL(1) parsing table for a given CFG*

#include <iostream>

#include <vector>

#include <fstream>

#include <string>

#include <algorithm>

using namespace std;

int num\_of\_productions;

vector<string> productions;

vector<char> terminals;

vector<char> nonTerminals;

vector<string> newProductions;

vector<string> deterministicProductions;

int isInTerminal(char c){

//Check if a value is a valid Terminal

if(find(terminals.begin(),terminals.end(),c)!=terminals.end()){

return 1;

}

return 0;

}

int isInNonTerminal(char c){//Check if a value is a valid NonTerminal

if(find(nonTerminals.begin(),nonTerminals.end(),c)!=nonTerminals.end()){

return 1;

}

return 0;

}

int isTerminal(char tem){//Assume Capital Letters As non terminals and small letters as terminals

if(tem>='A' && tem<='Z'){

return 0;

}else{

return 1;

}

}

int prepareListOfCharacter(){//Prepare List Of Valid Characters

for(int i=0;i<productions.size()-1;i++){

string tempString=productions[i];

int tempStringLen = tempString.length();

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

char t = tempString[j];

if(!(t==' ')){

if(isInNonTerminal(t)||isInTerminal(t)){

continue;

}else{

//Check if Terminal Or Non Terminal.

// Add in the respective Vector

if(isTerminal(t)){

terminals.push\_back(t);

}else{

nonTerminals.push\_back(t);

}

}

}

}

}

return 1;

}

int readProductions(){

//Function to read the productions from the input file

//READING BEGINS

string fileName ="";

cout<<"Enter File Name : ";

cin>>fileName;

cout<<endl;

ifstream input(fileName.c\_str());

string tempString;

input>>num\_of\_productions;

getline(input, tempString, '\n');

while (getline(input, tempString, '\n')) {

productions.push\_back(tempString);

}

cout<<"DATA FROM INPUT FILE BEGINS: "<<endl;

cout<<"Num Of Productions : "<<num\_of\_productions<<endl;

for(int i=0;i<productions.size()-1;i++){

cout<<productions[i]<<endl;

}

cout<<"DATA FROM INPUT FILE ENDS: "<<endl<<endl;

//READING ENDS

prepareListOfCharacter();

cout<<"\nAll The Terminals : ";

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

cout<<terminals[i]<<" ";

}

cout<<"\nAll The Non Terminals : ";

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

cout<<nonTerminals[i]<<" ";

}

cout<<"\n";

return 1;

}

char nonTermGen(){

//Generate A Non Terminal To Replace

char nonTerm = 'A';

while(true){

if(isInNonTerminal(nonTerm)){

//cout<<"\nNon Term Mai Hai";

nonTerm += 1;

}else{

//cout<<"\nNon Term mai Ghanta";

return nonTerm;

}

if(nonTerm==('Z'+1)){

return '$';

}

}

}

int isLeftRecursive(char c){

//cout<<"\nChar => "<<c;

//cout<<"\n Productions : \n";

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

string tempString = productions[j];

//cout<<"\nTempString["<<j<<"] => "<<tempString;

if(tempString[0]==c){

if(!isInNonTerminal(tempString[2])){

//First Character Is A Terminal

//cout<<"\n Not L Rec => "<<tempString;

newProductions.push\_back(tempString);

}else{

//First Character Is A Non Terminal

if(tempString[2]==c){

//Direct Recursion

return 1;

}else{

//Indirect Recursion IGNORED FOR NOW

newProductions.push\_back(tempString);

}

}

}else{

//cout<<"\n First Character not "<<c;

}

}

return 0;

}

int rectifyLeftRecursion(char c){

// A->Ap|B

// ||

// A->BA'

// A'->pA'|e

//Adding Beta Productions

char newNonTerm = nonTermGen();

if(newNonTerm=='$'){

cout<<"Ran Out Of Non Terminals To Assign. Too Many Left Recursive Entries.";

exit;

}

vector<string> beta;

for(int k=0;k<productions.size();k++){

if(productions[k][0]==c){

if(productions[k][2]!=c){

beta.push\_back(productions[k]);

}

}

}

for(int k=0;k<beta.size();k++){

//Add Beta Productions

string betaProduction = beta[k]+newNonTerm;

if(!isInNonTerminal(newNonTerm)){

nonTerminals.push\_back(newNonTerm);

}

newProductions.push\_back(betaProduction);

}

//Dealing With A' Productions

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

if(productions[j][0]==c){

string tempString = productions[j];

if(tempString[2]==c){

//Direct Left Recursion Present

string alpha = tempString.substr(3);

//Add A' Productions

string alphaProduction = "";

alphaProduction += newNonTerm;

alphaProduction += (" "+alpha);

alphaProduction += newNonTerm;

if(!isInNonTerminal(newNonTerm)){

nonTerminals.push\_back(newNonTerm);

}

newProductions.push\_back(alphaProduction);

}

}

}

string epsilonProduction = "";

epsilonProduction += newNonTerm;

epsilonProduction += " #";

newProductions.push\_back(epsilonProduction);

terminals.push\_back(newNonTerm);

return 1;

}

int printFinalGrammar(){

cout<<"\nFINAL LEFT RECURSION FREE GRAMMAR IS : "<<endl;

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

cout<<newProductions[i]<<endl;

}

return 1;

}

int clearProductions(char c){

vector<string> tempProds=productions;

int j=0;

while(true){

if(tempProds[j][0]==c){

tempProds.erase(tempProds.begin()+j);

}else{

j++;

}

if(j==tempProds.size()){

break;

}

}

productions = tempProds;

return 1;

}

int isNonDeterministic(char c){

//return 0;

vector<string> tempProductions;

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

string tempString = productions[j];

if(productions[j][0]==c){

tempProductions.push\_back(productions[j]);

}

}

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

string tempString = tempProductions[j];

//cout<<"\n\n\nTemp String => "<<tempString;

for(int k=0;k<tempProductions.size();k++){

//cout<<"\nTempProduction[k] => "<<k<<"=> "<<tempProductions[k];

if(tempString==tempProductions[k]){

//cout<<"\t 1";

continue;

}else{

if(tempString.length()>tempProductions[k].length()){

//cout<<"\t 2";

for(int l=tempProductions[k].length();l>1;l--){

size\_t found = tempString.find(tempProductions[k].substr(1,l));

if(found!=string::npos){

//Find the Element in tempString

if(found==1){

//Element At The Beginning This is the case We're interested in

//cout<<"\n ND 1 "<<tempProductions[k].substr(2,l);

return 1;

}

}

}

}else{

//cout<<"\t 3";

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

//cout<<"\n\tLooking For => "<<tempString.substr(1,l)<<"\t l = "<<l;

size\_t found = tempProductions[k].find(tempString.substr(1,l));

if(found!=string::npos){

//cout<<"\t=> Found => "<<tempString.substr(1,l)<<" At => "<<found;

//Find the Element in tempProduction

if(found==1){

//Element At The Beginning This is the case We're interested in

//cout<<"\n ND 2 "<<tempProductions[k].substr(1,l);

return 1;

}

}

}

}

}

}

}

return 0;

}

int exists(string s){

for(int m=0;m<deterministicProductions.size();m++){

if(s == deterministicProductions[m]){

return 1;

}

}

return 0;

}

int rectifyNonDeterminism(char c){

//return 0;

char newNonTerm = nonTermGen();

if(newNonTerm=='$'){

cout<<"Ran Out Of Non Terminals To Assign. Too Many Non Deterministic Entries.";

exit;

}

vector<string> tempProductions;

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

string tempString = productions[j];

if(productions[j][0]==c){

tempProductions.push\_back(productions[j]);

}

}

//cout<<"\nChar => "<<c;

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

string tempString = tempProductions[j];

//cout<<"\n\tTempString => "<<tempString;

int flag=0;

for(int k=0;k<tempProductions.size();k++){

//cout<<"\n\tTempProduction["<<k<<"] => "<<tempProductions[k];

if(tempString==tempProductions[k]){

//cout<<"\n Same";

continue;

}else{

//cout<<"\n\t\tDiffer";

if(tempString.length()>tempProductions[k].length()){

for(int l=tempProductions[k].length();l>1;l--){

//cout<<"\n\t\tLooking For => "<<tempProductions[k].substr(1,l);

size\_t found = tempString.find(tempProductions[k].substr(1,l));

if(found!=string::npos){

//Find the Element in tempString

if(found==1){

flag=1;

//Element At The Beginning This is the case We're interested in

//cout<<"\n\t ND 1 Found => "<<tempProductions[k].substr(1,l)<<"\t l = "<<l;

if(!isInNonTerminal(newNonTerm)){

nonTerminals.push\_back(newNonTerm);

}

string newProd1 = "";

newProd1+=c;

newProd1 += (" "+tempString.substr(1,l));

newProd1 += newNonTerm;

string newProd2 = "";

newProd2+=newNonTerm;

newProd2 += (" "+tempString.substr(l+1));

if(!exists(newProd1)){

//cout<<"\n\t\tnewProd1 => "<<newProd1;

deterministicProductions.push\_back(newProd1);

}

if(!exists(newProd2)){

//cout<<"\n\t\tnewProd2 => "<<newProd2;

deterministicProductions.push\_back(newProd2);

}

}

}

}

}else{

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

size\_t found = tempProductions[k].find(tempString.substr(1,l));

//cout<<"\n\t\tLooking For => "<<tempProductions[k].substr(1,l);

if(found!=string::npos){

//Find the Element in tempProduction

if(found==1){

flag=1;

//Element At The Beginning This is the case We're interested in

//cout<<"\n ND 2 Found => "<<tempString.substr(1,l)<<"\t l = "<<l;

if(!isInNonTerminal(newNonTerm)){

nonTerminals.push\_back(newNonTerm);

}

string newProd1 = "";

newProd1+=c;

newProd1 += tempString.substr(1,l);

newProd1 += newNonTerm;

//cout<<"\n\tnewProd1 => "<<newProd1;

string newProd2 = "";

newProd2+=newNonTerm;

newProd2 += (" "+tempString.substr(l+1));

//cout<<"\n\tnewProd2 => "<<newProd2;

if(!exists(newProd1)){

//cout<<"\n\t\tnewProd1 => "<<newProd1;

deterministicProductions.push\_back(newProd1);

}

if(!exists(newProd2)){

//cout<<"\n\t\tnewProd2 => "<<newProd2;

deterministicProductions.push\_back(newProd2);

}

}

}

}

}

}

}

if(flag==0){

if(!exists(tempString)){

deterministicProductions.push\_back(tempString);

}

}

}

return 1;

}

int printNDGrammar(){

cout<<"\nFINAL NON DETERMINISTIC FREE GRAMMAR IS : "<<endl;

//return 1;

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

cout<<productions[i]<<endl;

}

return 1;

}

int main(){

//Check For:

//Unambiguous -- undecidable

//Left Recursion

//Non Determinism

cout<<"# => epsilon"<<endl;

//Read All The Productions

readProductions();

int flag=0;

//NON DETERMINISM CHECK BEGINS

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

if(isNonDeterministic(nonTerminals[i])){

//cout<<"\nInside ND\n";

rectifyNonDeterminism(nonTerminals[i]);

clearProductions(nonTerminals[i]);

flag=1;

}else{

//cout<<"\nInside Dete\n ";

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

if(productions[j][0]==nonTerminals[i]){

deterministicProductions.push\_back(productions[j]);

}

}

}

}

if(flag==0){

cout<<"\nGrammar Is Free Of Non Determinism";

}else{

productions = deterministicProductions;

printNDGrammar();

}

//NON DETERMINISM CHECK ENDS

flag=0;

//LEFT RECURSION CHECK BEGINS

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

if(isLeftRecursive(nonTerminals[i])){

//Left Recursion Found

//cout<<"Grammar Has Left Recursion In Production Of : "<<nonTerminals[i]<<endl;

rectifyLeftRecursion(nonTerminals[i]);

clearProductions(nonTerminals[i]);

flag=1;

}

}

if(flag==0){

cout<<"\nGrammar Is Free Of Left Recursion"<<endl;

}else{

printFinalGrammar();

productions = newProductions;

}

//LEFT RECURSION CHECK FINISH

flag=0;

return 1;

}

