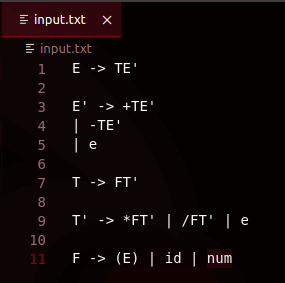
**Name: Mohammad Awais**

**Class: BSCS-8-A  
CMS: 242554**

# Compiler Construction Lab 8

#### < Input Grammmar >



## Task 1 ( Program to Calculate Firsts )

#### < Code >

#include <iostream>

using namespace std;

#include <fstream>

#include <vector>

#include<string>

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* INITIALIZATIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

vector<string> nt\_list = {};

vector<string> t\_list = {")","(","e","num","id","+","-","/","\*"};

vector<vector<string>> nt\_derivs = {};

vector<vector<string>> nt\_firsts = {};

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*/\* USER DEFINED FOR REPLACEMENT OF SUBTRING IN STRING \*/*

void replaceAll( string &s, const string &search, const string &replace ) {

for( size\_t pos = 0; ; pos += replace.length() ) {

*// Locate the substring to replace*

pos = s.find( search, pos );

if( pos == string::**npos** ) break;

*// Replace by erasing and inserting*

s.erase( pos, search.length() );

s.insert( pos, replace );

}

}

*/\* Find FIRST of Non-Terminal of input Index \*/*

vector<string> ntFirstCal(int i, char mode){

int e\_case = 0;

vector<string> local\_firsts = {};

if(mode=='t'){

local\_firsts.push\_back(t\_list[i]);

}

else{

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

string word="";

for(int k=0; k<nt\_derivs[i][j].length();k++){

char c = nt\_derivs[i][j][k];

word+=c;

if(c==']'){

if(word[1]=='t'){

if(t\_list[word[2]-'0']=="e"&&k!=nt\_derivs[i][j].length()-1){

word="";

}

else{

int x = word[2]- '0';

local\_firsts.push\_back(t\_list[x]);

break;

}

}

else if(word[1]=='n'){

int x = word[2]- '0';

vector<string> inherit\_firsts = {};

if(nt\_firsts[x].empty()){

inherit\_firsts = ntFirstCal(x,'n');

}

else{

inherit\_firsts = nt\_firsts[x];

}

int sec\_size = inherit\_firsts.size();

for(int p=0; p<sec\_size;p++){

string firsts = inherit\_firsts[p];

if(firsts == "e" && k!=nt\_derivs[i][j].length()-1){

x = nt\_derivs[i][j][k+3]-'0';

c = nt\_derivs[i][j][k+2];

k=k+4;

if(c=='n'){

vector<string> post\_inherit\_firsts = {};

if(nt\_firsts[x].empty())

post\_inherit\_firsts = ntFirstCal(x,c);

else

post\_inherit\_firsts = nt\_firsts[x];

for (string sfir : post\_inherit\_firsts){

inherit\_firsts.push\_back(sfir);

}

sec\_size = inherit\_firsts.size();

}

else{

local\_firsts.push\_back(t\_list[x]);

}

}

else{

int exist = 0;

for(string f : local\_firsts){

if(f.compare(firsts)==0){

exist = 1;

break;

}

}

if(exist==0){

local\_firsts.push\_back(firsts);

}

}

}

break;

}

}

}

}

nt\_firsts[i] = local\_firsts;

}

return local\_firsts;

}

*/\* Replaces Derivations with appropriate Tokens \*/*

void tokenizeDerivs(){

*// PRE PROCESSING*

int max\_size = 0;

for(int i = 0; i<nt\_list.size();i++){

int size = nt\_list[i].length();

if(size>max\_size){

max\_size=size;

}

}

*// SUBSTITUTES DERIVATIVES WITH IDs*

for(int j = 0; j<nt\_derivs.size();j++){

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

string s = nt\_derivs[j][k];

for(int sos=max\_size; sos>0;sos--)

{for(int i=0; i<nt\_list.size() ;i++){

if(sos == nt\_list[i].length()){

int spos = s.find(nt\_list[i]);

string x = nt\_list[i];

if (spos>=0){

replaceAll(nt\_derivs[j][k], nt\_list[i], "[n"+to\_string(i)+"]"); *// WORKING GREAT*

}}

}}

for(int i=0; i<t\_list.size();i++){

int spos = s.find(t\_list[i]);

if (spos>=0){

replaceAll(nt\_derivs[j][k], t\_list[i], "[t"+to\_string(i)+"]"); *// WORKING BETTER*

}

}

}

}

}

*/\* DIVIDES LHS and RHS INTO THEIR RESPECTIVE LISTS \*/*

void divider(string fileString){

int mode = 0;

string temp = "";

vector<string> tempList = {};

for(int i=0; i<fileString.length()-1;i++){

temp += fileString[i];

if(mode==0){

if(fileString[i+1]=='-'&&fileString[i+2]=='>'){

nt\_list.push\_back(temp);

i=i+2;

mode=1;

temp = "";

}

}

else if(mode==1){

if(fileString[i+1]=='\n' || fileString[i+1] == '|'){

if(fileString[i+1]=='\n'&&fileString[i+2]=='|'){

i=i+2;

}

else if(fileString[i+1]=='|'){

i=i+1;

}

else if(fileString[i+1]=='\n'&&fileString[i+2]!='|'){

mode = 0;

i=i+1;

}

tempList.push\_back(temp);

temp="";

if(mode==0){

nt\_derivs.push\_back(tempList);

nt\_firsts.push\_back({}); *// For FIRST LIST*

tempList = {};

}

}

}

}

}

*/\* READ FILE STRING AND STORE IN A STRING VARIABLE \*/*

string readNout(string filename){

string readLine;

*// Read from the text file*

ifstream grammerFile(filename);

string fileString = "";

while (getline (grammerFile, readLine)) {

*// Output the text from the file*

for(char c: readLine){

if(c!='\t' && c!=' '){

fileString=fileString+c;

}

}

if(!readLine.empty())

fileString=fileString+"\n";

}

grammerFile.close();

return fileString;

}

int main() {

cout << " \t < Program 1 >\n\n";

string filename = "input.txt";

string fileString = readNout(filename)+"\0"; *// Reads Text File Input of Grammar into String Variable*

divider(fileString); *// Divides LHS and RHS into Lists nt\_list & nt\_derivs*

tokenizeDerivs(); *// Tokenizes nt\_derivs*

*// Calculating Firsts and storing in nt\_firsts list.*

for(int i=0; i<nt\_derivs.size();i++){

if(nt\_firsts[i].empty()){

ntFirstCal(i,'n');

}

}

*// Displaying Firsts*

cout<<"FIRST of Non-Terminals:-\n";

for(int i=0; i<nt\_firsts.size();i++){

vector<string> vec = nt\_firsts[i];

cout<<"\n> First("<<nt\_list[i]<<") = { ";

for(string str: vec){

cout<<str<<" , ";

}

cout<<"}\n";

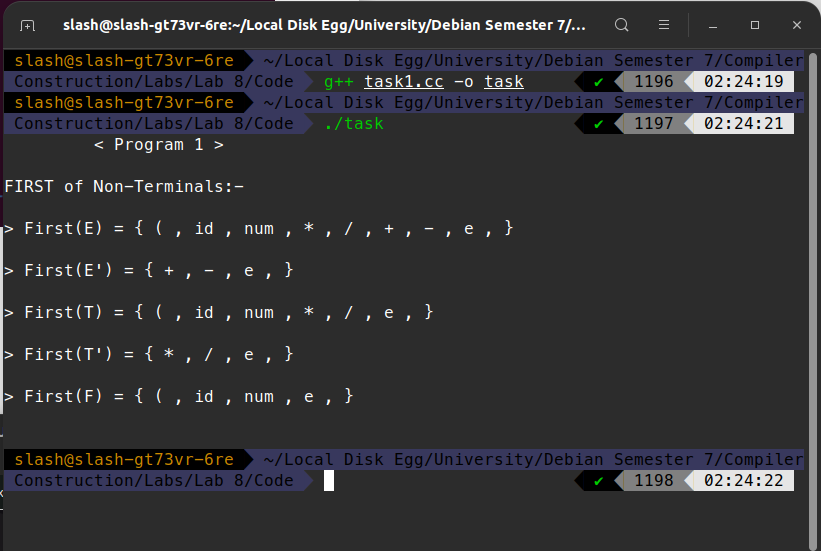
}

cout << "\n\n";

return 0;

}

#### < Output >



## Task 2 ( Program to Calculate Follows )

#### < Code >

#include <iostream>

using namespace std;

#include <fstream>

#include <vector>

#include<string>

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* INITIALIZATIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

vector<string> nt\_list = {};

vector<string> t\_list = {")","(","e","num","id","+","-","/","\*"};

vector<vector<string>> nt\_derivs = {};

vector<vector<string>> nt\_firsts = {};

vector<vector<string>> nt\_follows = {};

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*/\* USER DEFINED FOR REPLACEMENT OF SUBTRING IN STRING \*/*

void replaceAll( string &s, const string &search, const string &replace ) {

for( size\_t pos = 0; ; pos += replace.length() ) {

*// Locate the substring to replace*

pos = s.find( search, pos );

if( pos == string::**npos** ) break;

*// Replace by erasing and inserting*

s.erase( pos, search.length() );

s.insert( pos, replace );

}

}

*/\* Find FIRST of Non-Terminal of input Index \*/*

vector<string> ntFirstCal(int i, char mode){

int e\_case = 0;

vector<string> local\_firsts = {};

if(mode=='t'){

local\_firsts.push\_back(t\_list[i]);

}

else{

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

string word="";

for(int k=0; k<nt\_derivs[i][j].length();k++){

char c = nt\_derivs[i][j][k];

word+=c;

if(c==']'){

if(word[1]=='t'){

if(t\_list[word[2]-'0']=="e"&&k!=nt\_derivs[i][j].length()-1){

word="";

}

else{

int x = word[2]- '0';

local\_firsts.push\_back(t\_list[x]);

break;

}

}

else if(word[1]=='n'){

int x = word[2]- '0';

vector<string> inherit\_firsts = {};

if(nt\_firsts[x].empty()){

inherit\_firsts = ntFirstCal(x,'n');

}

else{

inherit\_firsts = nt\_firsts[x];

}

int sec\_size = inherit\_firsts.size();

for(int p=0; p<sec\_size;p++){

string firsts = inherit\_firsts[p];

if(firsts == "e" && k!=nt\_derivs[i][j].length()-1){

x = nt\_derivs[i][j][k+3]-'0';

c = nt\_derivs[i][j][k+2];

k=k+4;

if(c=='n'){

vector<string> post\_inherit\_firsts = {};

if(nt\_firsts[x].empty())

post\_inherit\_firsts = ntFirstCal(x,c);

else

post\_inherit\_firsts = nt\_firsts[x];

for (string sfir : post\_inherit\_firsts){

inherit\_firsts.push\_back(sfir);

}

sec\_size = inherit\_firsts.size();

}

else{

local\_firsts.push\_back(t\_list[x]);

}

}

else{

int exist = 0;

for(string f : local\_firsts){

if(f.compare(firsts)==0){

exist = 1;

break;

}

}

if(exist==0){

local\_firsts.push\_back(firsts);

}

}

}

break;

}

}

}

}

nt\_firsts[i] = local\_firsts;

}

return local\_firsts;

}

*/\* Find FOLLOW of Non-Terminal of input Index \*/*

vector<string> ntFollowCal(int i){

vector<string> local\_follows = {};

vector<vector<int>> local\_detects = {}; *// DELETE SAWA CHECK FROM HERE (UPDATE: PERHAPS NOT AS USED), MOST LIKELY ALL FOLLOWS NOT BEING CALC.*

for(int j=0; j<nt\_derivs.size();j++){

if(i!=j){

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

for(int l=0; l<nt\_derivs[j][k].length(); l++){

if(nt\_derivs[j][k][l]=='n' && nt\_derivs[j][k][l+1]-'0'==i){

local\_detects.push\_back({j,k,l+2});

}

}

}

}

}

for(vector<int> detect : local\_detects){

int z = detect[0];

int a = detect[1];

int b = detect[2];

int get\_out;

int x;

while(1){

get\_out =1;

if(b == nt\_derivs[z][a].length()-1){

vector<string> inherit\_follows;

if(nt\_follows[z].empty())

inherit\_follows = ntFollowCal(z);

else

inherit\_follows = nt\_follows[z];

for(string follows: inherit\_follows){

local\_follows.push\_back(follows);

}

}

else if(nt\_derivs[z][a][b+2]=='t'){

x = nt\_derivs[z][a][b+3]- '0';

if(t\_list[x]=="e"){

b=b+4;

get\_out = 0;

}

else{

local\_follows.push\_back(t\_list[x]);

}

}

else if(nt\_derivs[z][a][b+2] == 'n'){

x = nt\_derivs[z][a][b+3] - '0';

vector<string> inherit\_firsts;

if(nt\_firsts[x].empty())

inherit\_firsts = ntFirstCal(x,'n');

else

inherit\_firsts = nt\_firsts[x];

for(string first: inherit\_firsts){

if(first!="e")

local\_follows.push\_back(first);

else{

b=b+4;

get\_out = 0;

}

}

}

if(get\_out==1){

break;

}

}

}

if(i==0){

local\_follows.push\_back("$");

}

nt\_follows[i] = local\_follows;

vector<string> final\_local\_follows = {};

for(string f : local\_follows){

int detect\_f = 0;

for(string t : final\_local\_follows){

if(f == t){

detect\_f = 1;

break;

}

}

if(detect\_f == 0 ){

final\_local\_follows.push\_back(f);

}

}

nt\_follows[i] = final\_local\_follows;

return local\_follows;

}

*/\* Replaces Derivations with appropriate Tokens \*/*

void tokenizeDerivs(){

*// PRE PROCESSING*

int max\_size = 0;

for(int i = 0; i<nt\_list.size();i++){

int size = nt\_list[i].length();

if(size>max\_size){

max\_size=size;

}

}

*// SUBSTITUTES DERIVATIVES WITH IDs*

for(int j = 0; j<nt\_derivs.size();j++){

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

string s = nt\_derivs[j][k];

for(int sos=max\_size; sos>0;sos--)

{for(int i=0; i<nt\_list.size() ;i++){

if(sos == nt\_list[i].length()){

int spos = s.find(nt\_list[i]);

string x = nt\_list[i];

if (spos>=0){

replaceAll(nt\_derivs[j][k], nt\_list[i], "[n"+to\_string(i)+"]"); *// WORKING GREAT*

}}

}}

for(int i=0; i<t\_list.size();i++){

int spos = s.find(t\_list[i]);

if (spos>=0){

replaceAll(nt\_derivs[j][k], t\_list[i], "[t"+to\_string(i)+"]"); *// WORKING BETTER*

}

}

}

}

}

*/\* DIVIDES LHS and RHS INTO THEIR RESPECTIVE LISTS \*/*

void divider(string fileString){

int mode = 0;

string temp = "";

vector<string> tempList = {};

for(int i=0; i<fileString.length()-1;i++){

temp += fileString[i];

if(mode==0){

if(fileString[i+1]=='-'&&fileString[i+2]=='>'){

nt\_list.push\_back(temp);

i=i+2;

mode=1;

temp = "";

}

}

else if(mode==1){

if(fileString[i+1]=='\n' || fileString[i+1] == '|'){

if(fileString[i+1]=='\n'&&fileString[i+2]=='|'){

i=i+2;

}

else if(fileString[i+1]=='|'){

i=i+1;

}

else if(fileString[i+1]=='\n'&&fileString[i+2]!='|'){

mode = 0;

i=i+1;

}

tempList.push\_back(temp);

temp="";

if(mode==0){

nt\_derivs.push\_back(tempList);

nt\_firsts.push\_back({}); *// For FIRST LIST*

nt\_follows.push\_back({}); *// For FOLLOWS LIST*

tempList = {};

}

}

}

}

}

*/\* READ FILE STRING AND STORE IN A STRING VARIABLE \*/*

string readNout(string filename){

string readLine;

*// Read from the text file*

ifstream grammerFile(filename);

string fileString = "";

while (getline (grammerFile, readLine)) {

*// Output the text from the file*

for(char c: readLine){

if(c!='\t' && c!=' '){

fileString=fileString+c;

}

}

if(!readLine.empty())

fileString=fileString+"\n";

}

grammerFile.close();

return fileString;

}

int main() {

cout << " \t < Program 2 >\n\n";

string filename = "input.txt";

string fileString = readNout(filename)+"\0"; *// Reads Text File Input of Grammar into String Variable*

divider(fileString); *// Divides LHS and RHS into Lists nt\_list & nt\_derivs*

tokenizeDerivs(); *// Tokenizes nt\_derivs*

*// Calculating Firsts and storing in nt\_firsts list.*

for(int i=0; i<nt\_derivs.size();i++){

if(nt\_firsts[i].empty()){

ntFirstCal(i,'n');

}

}

*// Displaying Firsts*

cout<<"FIRST of Non-Terminals:-\n";

for(int i=0; i<nt\_firsts.size();i++){

vector<string> vec = nt\_firsts[i];

cout<<"\n> First("<<nt\_list[i]<<") = { ";

for(string str: vec){

cout<<str<<" , ";

}

cout<<"}\n";

}

*// Calculating Follows and storing in nt\_follows list.*

for(int i=0; i<nt\_derivs.size();i++){

if(nt\_follows[i].empty()){

ntFollowCal(i);

}

}

*// Displaying Follows*

cout<<"\n\nFOLLOWS of Non-Terminals:-\n";

for(int i=0; i<nt\_follows.size();i++){

vector<string> vec = nt\_follows[i];

cout<<"\n> Follow("<<nt\_list[i]<<") = { ";

for(string str: vec){

cout<<str<<" , ";

}

cout<<"}\n";

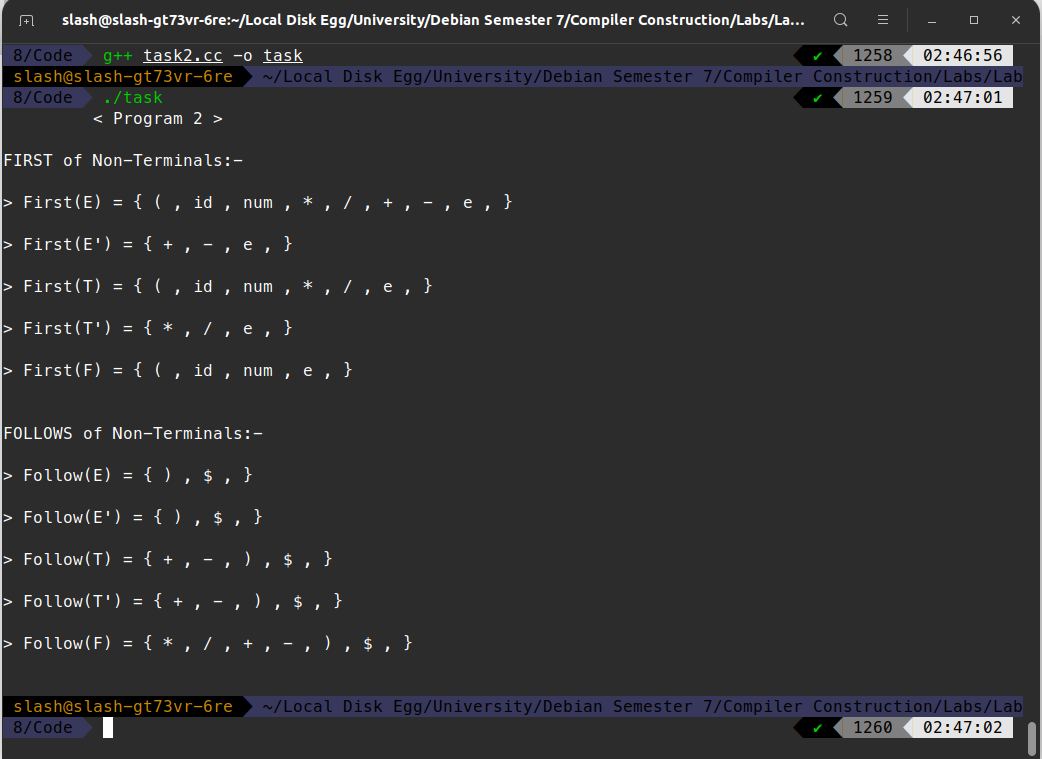
}

cout << "\n\n";

return 0;

}

#### < Output >



## Task 3 ( Parsing Table )

#### < Code >

#include <iostream>

using namespace std;

#include <fstream>

#include <vector>

#include<string>

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* INITIALIZATIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

vector<string> nt\_list = {};

vector<string> t\_list = {")","(","e","num","id","+","-","/","\*"};

vector<vector<string>> nt\_derivs = {};

vector<vector<string>> nt\_derivs\_o = {};

vector<vector<string>> nt\_firsts = {};

vector<vector<string>> nt\_parsing\_entries = {};

vector<vector<string>> nt\_follows = {};

vector<vector<string>> parsingTable = {};

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*/\* USER DEFINED FOR REPLACEMENT OF SUBTRING IN STRING \*/*

void replaceAll( string &s, const string &search, const string &replace ) {

for( size\_t pos = 0; ; pos += replace.length() ) {

*// Locate the substring to replace*

pos = s.find( search, pos );

if( pos == string::**npos** ) break;

*// Replace by erasing and inserting*

s.erase( pos, search.length() );

s.insert( pos, replace );

}

}

*/\* Find FIRST of Non-Terminal of input Index \*/*

vector<string> ntFirstCal(int i, char mode){

int e\_case = 0;

vector<string> local\_firsts = {};

vector<string> local\_pentries = {};

if(mode=='t'){

local\_firsts.push\_back(t\_list[i]);

}

else{

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

string word="";

for(int k=0; k<nt\_derivs[i][j].length();k++){

char c = nt\_derivs[i][j][k];

word+=c;

if(c==']'){

if(word[1]=='t'){

if(t\_list[word[2]-'0']=="e"&&k!=nt\_derivs[i][j].length()-1){

word="";

}

else{

int x = word[2]- '0';

local\_firsts.push\_back(t\_list[x]);

local\_pentries.push\_back(nt\_list[i]+"->"+nt\_derivs\_o[i][j]);

break;

}

}

else if(word[1]=='n'){

int x = word[2]- '0';

vector<string> inherit\_firsts = {};

if(nt\_firsts[x].empty()){

inherit\_firsts = ntFirstCal(x,'n');

}

else{

inherit\_firsts = nt\_firsts[x];

}

int sec\_size = inherit\_firsts.size();

for(int p=0; p<sec\_size;p++){

string firsts = inherit\_firsts[p];

if(firsts == "e" && k!=nt\_derivs[i][j].length()-1){

x = nt\_derivs[i][j][k+3]-'0';

c = nt\_derivs[i][j][k+2];

k=k+4;

if(c=='n'){

vector<string> post\_inherit\_firsts = {};

if(nt\_firsts[x].empty())

post\_inherit\_firsts = ntFirstCal(x,c);

else

post\_inherit\_firsts = nt\_firsts[x];

for (string sfir : post\_inherit\_firsts){

inherit\_firsts.push\_back(sfir);

}

sec\_size = inherit\_firsts.size();

}

else{

local\_firsts.push\_back(t\_list[x]);

local\_pentries.push\_back(nt\_list[i]+"->"+nt\_derivs\_o[i][j]);

}

}

else{

int exist = 0;

for(string f : local\_firsts){

if(f.compare(firsts)==0){

exist = 1;

break;

}

}

if(*/\*exist==0\*/*1){

local\_firsts.push\_back(firsts);

local\_pentries.push\_back( nt\_list[i]+"->"+nt\_derivs\_o[i][j]);

}

}

}

break;

}

}

}

}

nt\_firsts[i] = local\_firsts;

nt\_parsing\_entries[i] = local\_pentries;

}

return local\_firsts;

}

*/\* Find FOLLOW of Non-Terminal of input Index \*/*

vector<string> ntFollowCal(int i){

vector<string> local\_follows = {};

vector<vector<int>> local\_detects = {}; *// DELETE SAWA CHECK FROM HERE (UPDATE: PERHAPS NOT AS USED), MOST LIKELY ALL FOLLOWS NOT BEING CALC.*

for(int j=0; j<nt\_derivs.size();j++){

if(i!=j){

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

for(int l=0; l<nt\_derivs[j][k].length(); l++){

if(nt\_derivs[j][k][l]=='n' && nt\_derivs[j][k][l+1]-'0'==i){

local\_detects.push\_back({j,k,l+2});

}

}

}

}

}

for(vector<int> detect : local\_detects){

int z = detect[0];

int a = detect[1];

int b = detect[2];

int get\_out;

int x;

while(1){

get\_out =1;

if(b == nt\_derivs[z][a].length()-1){

vector<string> inherit\_follows;

if(nt\_follows[z].empty())

inherit\_follows = ntFollowCal(z);

else

inherit\_follows = nt\_follows[z];

for(string follows: inherit\_follows){

local\_follows.push\_back(follows);

}

}

else if(nt\_derivs[z][a][b+2]=='t'){

x = nt\_derivs[z][a][b+3]- '0';

if(t\_list[x]=="e"){

b=b+4;

get\_out = 0;

}

else{

local\_follows.push\_back(t\_list[x]);

}

}

else if(nt\_derivs[z][a][b+2] == 'n'){

x = nt\_derivs[z][a][b+3] - '0';

vector<string> inherit\_firsts;

if(nt\_firsts[x].empty())

inherit\_firsts = ntFirstCal(x,'n');

else

inherit\_firsts = nt\_firsts[x];

for(string first: inherit\_firsts){

if(first!="e")

local\_follows.push\_back(first);

else{

b=b+4;

get\_out = 0;

}

}

}

if(get\_out==1){

break;

}

}

}

if(i==0){

local\_follows.push\_back("$");

}

nt\_follows[i] = local\_follows;

vector<string> final\_local\_follows = {};

for(string f : local\_follows){

int detect\_f = 0;

for(string t : final\_local\_follows){

if(f == t){

detect\_f = 1;

break;

}

}

if(detect\_f == 0 ){

final\_local\_follows.push\_back(f);

}

}

nt\_follows[i] = final\_local\_follows;

return local\_follows;

}

*/\* Replaces Derivations with appropriate Tokens \*/*

void tokenizeDerivs(){

*// PRE PROCESSING*

int max\_size = 0;

for(int i = 0; i<nt\_list.size();i++){

int size = nt\_list[i].length();

if(size>max\_size){

max\_size=size;

}

}

nt\_derivs\_o = nt\_derivs;

*// SUBSTITUTES DERIVATIVES WITH IDs*

for(int j = 0; j<nt\_derivs.size();j++){

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

string s = nt\_derivs[j][k];

for(int sos=max\_size; sos>0;sos--)

{for(int i=0; i<nt\_list.size() ;i++){

if(sos == nt\_list[i].length()){

int spos = s.find(nt\_list[i]);

string x = nt\_list[i];

if (spos>=0){

replaceAll(nt\_derivs[j][k], nt\_list[i], "[n"+to\_string(i)+"]"); *// WORKING GREAT*

}}

}}

for(int i=0; i<t\_list.size();i++){

int spos = s.find(t\_list[i]);

if (spos>=0){

replaceAll(nt\_derivs[j][k], t\_list[i], "[t"+to\_string(i)+"]"); *// WORKING BETTER*

}

}

}

}

}

*/\* DIVIDES LHS and RHS INTO THEIR RESPECTIVE LISTS \*/*

void divider(string fileString){

int mode = 0;

string temp = "";

vector<string> tempList = {};

for(int i=0; i<fileString.length()-1;i++){

temp += fileString[i];

if(mode==0){

if(fileString[i+1]=='-'&&fileString[i+2]=='>'){

nt\_list.push\_back(temp);

i=i+2;

mode=1;

temp = "";

}

}

else if(mode==1){

if(fileString[i+1]=='\n' || fileString[i+1] == '|'){

if(fileString[i+1]=='\n'&&fileString[i+2]=='|'){

i=i+2;

}

else if(fileString[i+1]=='|'){

i=i+1;

}

else if(fileString[i+1]=='\n'&&fileString[i+2]!='|'){

mode = 0;

i=i+1;

}

tempList.push\_back(temp);

temp="";

if(mode==0){

nt\_derivs.push\_back(tempList);

nt\_firsts.push\_back({}); *// For FIRST LIST*

nt\_follows.push\_back({}); *// For FOLLOWS LIST*

nt\_parsing\_entries.push\_back({});

tempList = {};

}

}

}

}

}

*/\* READ FILE STRING AND STORE IN A STRING VARIABLE \*/*

string readNout(string filename){

string readLine;

*// Read from the text file*

ifstream grammerFile(filename);

string fileString = "";

while (getline (grammerFile, readLine)) {

*// Output the text from the file*

for(char c: readLine){

if(c!='\t' && c!=' '){

fileString=fileString+c;

}

}

if(!readLine.empty())

fileString=fileString+"\n";

}

grammerFile.close();

return fileString;

}

int main() {

cout << " \t < Program 3 >\n\n";

string filename = "input.txt";

string fileString = readNout(filename)+"\0"; *// Reads Text File Input of Grammar into String Variable*

divider(fileString); *// Divides LHS and RHS into Lists nt\_list & nt\_derivs*

tokenizeDerivs(); *// Tokenizes nt\_derivs*

*// Calculating Firsts and storing in nt\_firsts list.*

for(int i=0; i<nt\_derivs.size();i++){

if(nt\_firsts[i].empty()){

ntFirstCal(i,'n');

}

}

*// Displaying Firsts*

cout<<"FIRST of Non-Terminals:-\n";

for(int i=0; i<nt\_firsts.size();i++){

vector<string> vec = nt\_firsts[i];

cout<<"\n> First("<<nt\_list[i]<<") = { ";

for(string str: vec){

cout<<str<<" , ";

}

cout<<"}\n";

}

*// Calculating Follows and storing in nt\_follows list.*

for(int i=0; i<nt\_derivs.size();i++){

if(nt\_follows[i].empty()){

ntFollowCal(i);

}

}

*// Displaying Follows*

cout<<"\n\nFOLLOWS of Non-Terminals:-\n";

for(int i=0; i<nt\_follows.size();i++){

vector<string> vec = nt\_follows[i];

cout<<"\n> Follow("<<nt\_list[i]<<") = { ";

for(string str: vec){

cout<<str<<" , ";

}

cout<<"}\n";

}

*// Calculating Parsing Table*

t\_list.push\_back("$");

for(int i=0; i<nt\_list.size();i++){

vector<string> column\_data = {};

for(int j=0; j<t\_list.size();j++){

*//if(t\_list[j]!="e") // Note this one in final version*

column\_data.push\_back("");

}

parsingTable.push\_back(column\_data);

}

for(int i=0; i<nt\_list.size();i++){

int mode = 0;

for(int j=0; j<t\_list.size();j++){

*//if(t\_list[j]!='e') // Note this one in final version*

for(int x =0;x<nt\_firsts[i].size();x++){

if(nt\_firsts[i][x]!="e"){

if(nt\_firsts[i][x]==t\_list[j]){

parsingTable[i][j]+=nt\_parsing\_entries[i][x] + ", ";

}

}

else{

mode = 1;

}

}

}

if(mode==1){

for(int j=0; j<t\_list.size();j++){

for(int x =0;x<nt\_follows[i].size();x++){

if(nt\_follows[i][x]==t\_list[j]){

parsingTable[i][j]+=nt\_list[i]+"->e"+", ";

}

}

}

}

}

*// Displaying Parsing Table*

cout<<"\n\nParsing Table:-\n\n";

for(int i=0; i<t\_list.size(); i++){

if(t\_list[i]!="e")

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

}

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

for(int i=0; i<nt\_list.size();i++){

cout<<nt\_list[i];

for(int j=0;j<t\_list.size();j++){

*//if(t\_list[j]!='e') // Note this one in final version*

if(t\_list[j]!="e"){

if(parsingTable[i][j].empty()){

cout<<"\t{\t}";

}

else

cout<<"\t{ "<<parsingTable[i][j]<<" }";

}

}

cout<<"\n\n";

}

cout << "\n\n";

return 0;

}

#### < Output >

