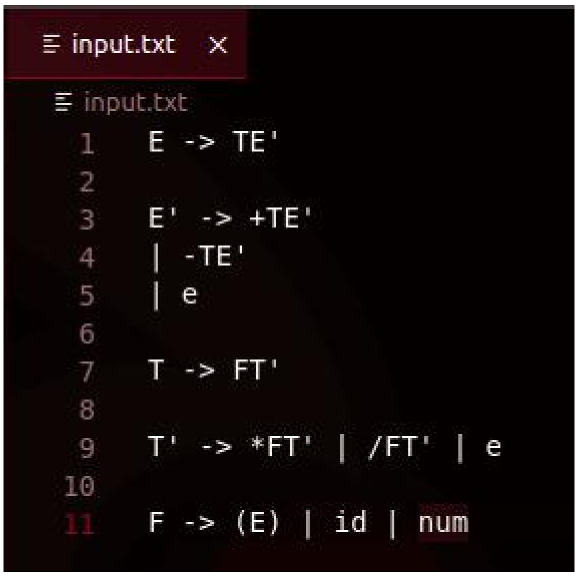
Name: Mohammad Awais
Class: BSCS-8-A
CMS: 242554

Compiler Construction Lab 8

< Input Grammmar >



```
input.txt
E -> TE'

E' -> +TE'
| -TE'
| e

T -> FT'

T' -> *FT' | /FT' | e

F -> (E) | id | num
```

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
```

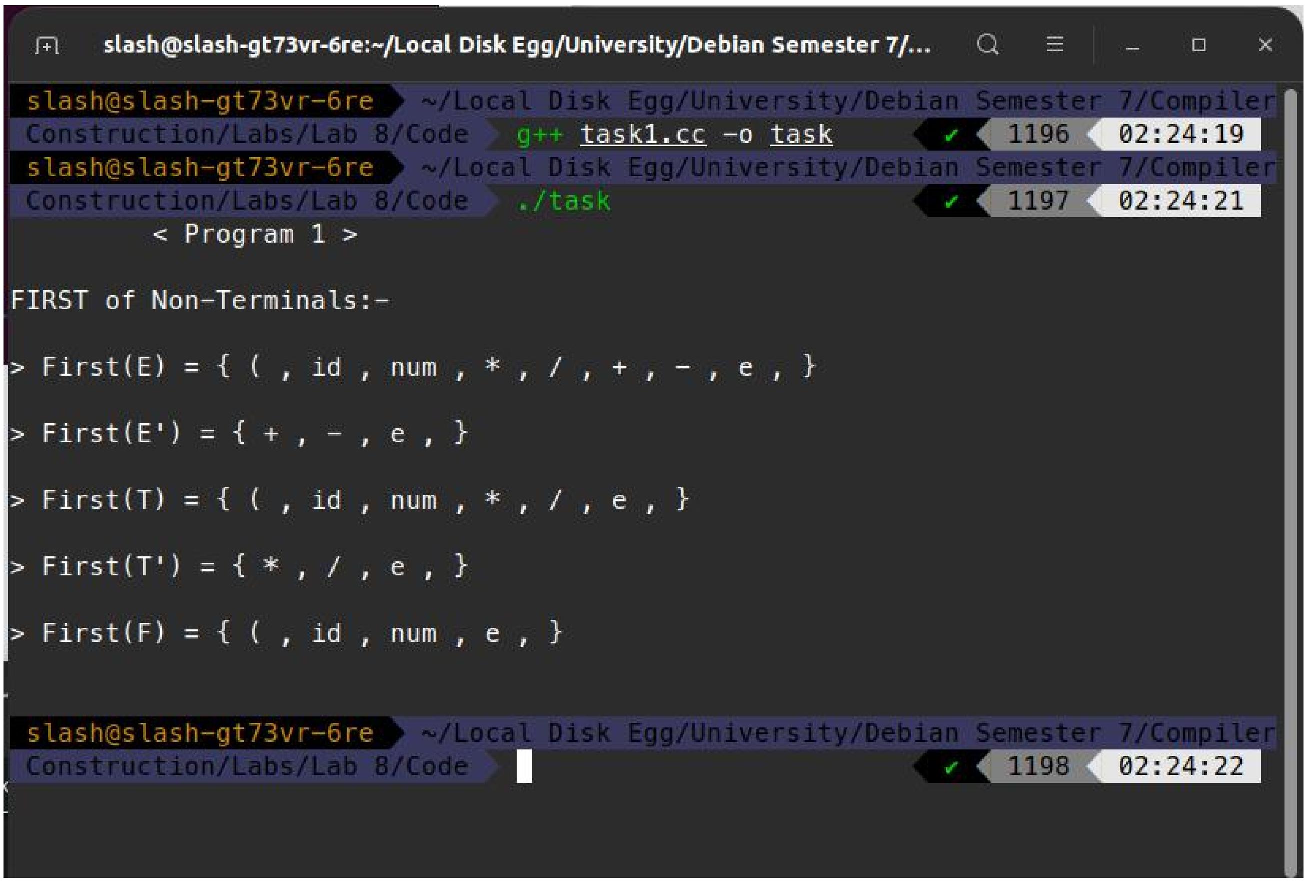
```
// Calculating Firsts and storing in nt_firsts list.
for(int i=0; i<nt_derivs.size();i++){
if(nt_firsts[i].empty()){
ntFirstCal(i,'n');
}
}
```

```
// 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 >



slash@slash-gt73vr-6re:~/Local Disk Egg/University/Debian Semester 7/...

```
slash@slash-gt73vr-6re  ~/Local Disk Egg/University/Debian Semester 7/Compiler
Construction/Labs/Lab 8/Code  g++ task1.cc -o task          ✔  1196  02:24:19
slash@slash-gt73vr-6re  ~/Local Disk Egg/University/Debian Semester 7/Compiler
Construction/Labs/Lab 8/Code  ./task                        ✔  1197  02:24:21
        < Program 1 >

FIRST of Non-Terminals:-

> First(E) = { ( , id , num , * , / , + , - , e , }

> First(E') = { + , - , e , }

> First(T) = { ( , id , num , * , / , e , }

> First(T') = { * , / , e , }

> First(F) = { ( , id , num , e , }


slash@slash-gt73vr-6re  ~/Local Disk Egg/University/Debian Semester 7/Compiler
Construction/Labs/Lab 8/Code                                 ✔  1198  02:24:22
```

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";
}
```

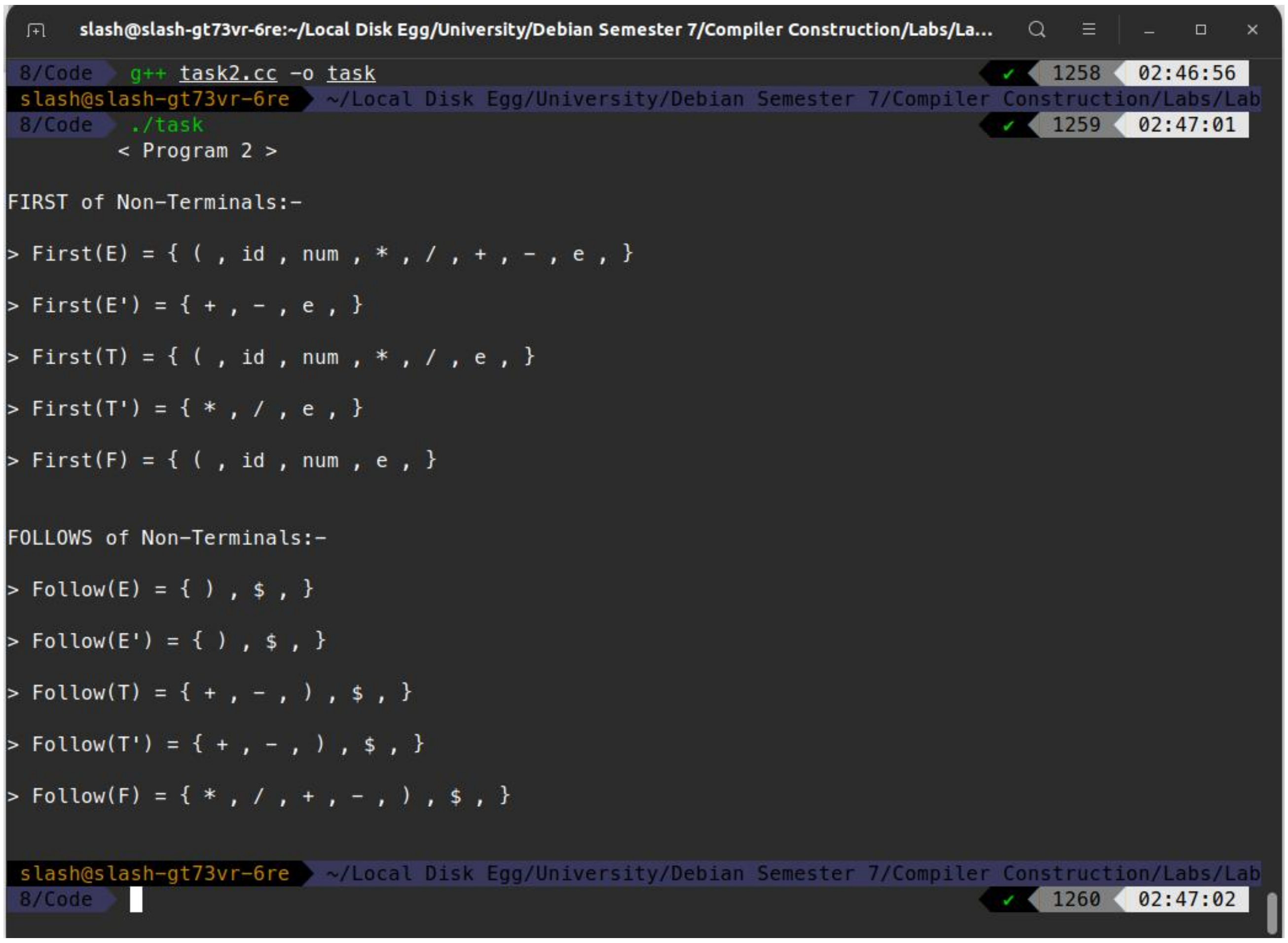
```
// Calculating Follows and storing in nt_follows list.
```

```
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 >



```
slash@slash-gt73vr-6re:~/Local Disk Egg/University/Debian Semester 7/Compiler Construction/Labs/La...
8/Code  g++ task2.cc -o task                                                  ✔  1258  02:46:56
slash@slash-gt73vr-6re  ~/Local Disk Egg/University/Debian Semester 7/Compiler Construction/Labs/Lab
8/Code  ./task                                                                ✔  1259  02:47:01
         < Program 2 >

FIRST of Non-Terminals:-

> First(E) = { ( , id , num , * , / , + , - , e , }

> First(E') = { + , - , e , }

> First(T) = { ( , id , num , * , / , e , }

> First(T') = { * , / , e , }

> First(F) = { ( , id , num , e , }


FOLLOWS of Non-Terminals:-

> Follow(E) = { ) , $ , }

> Follow(E') = { ) , $ , }

> Follow(T) = { + , - , ) , $ , }

> Follow(T') = { + , - , ) , $ , }

> Follow(F) = { * , / , + , - , ) , $ , }


slash@slash-gt73vr-6re  ~/Local Disk Egg/University/Debian Semester 7/Compiler Construction/Labs/Lab
8/Code                                                                        ✔  1260  02:47:02
```

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;
}
}
}
```

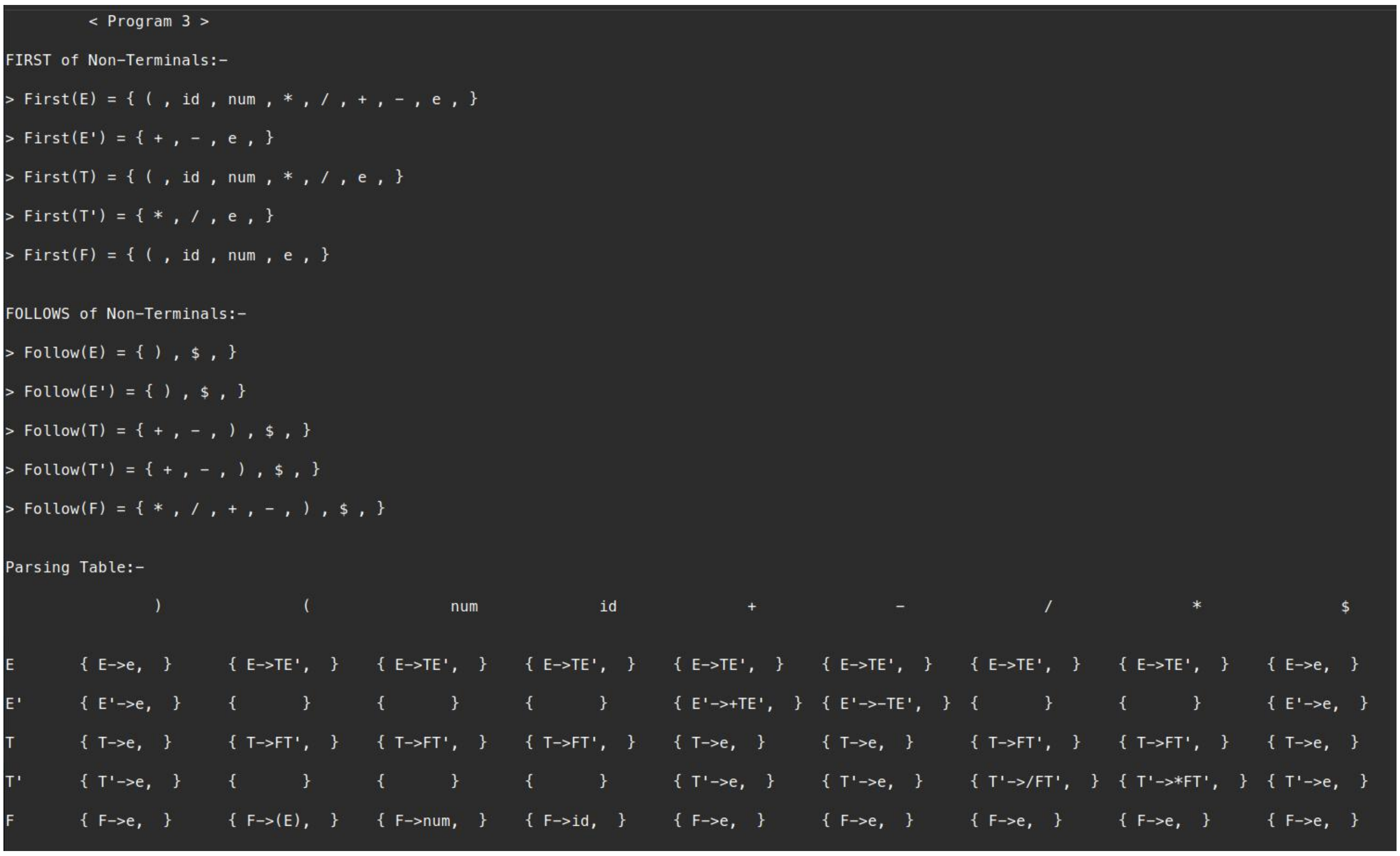
```
if(mode==1){
for(int j=0; j<t_list.size();j++){
for(int x =0;x<nt_follows[i].size();x++){
if(nt_follows[i][x]==t_list[j]){
parsingTable[i][j]+=nt_list[i]+"->e"+", ";
}
}
```

```
}
}
}
```

```
// 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 >



```
         < Program 3 >

FIRST of Non-Terminals:-

> First(E) = { ( , id , num , * , / , + , - , e , }

> First(E') = { + , - , e , }

> First(T) = { ( , id , num , * , / , e , }

> First(T') = { * , / , e , }

> First(F) = { ( , id , num , e , }


FOLLOWS of Non-Terminals:-

> Follow(E) = { ) , $ , }

> Follow(E') = { ) , $ , }

> Follow(T) = { + , - , ) , $ , }

> Follow(T') = { + , - , ) , $ , }

> Follow(F) = { * , / , + , - , ) , $ , }


Parsing Table:-

                )               (               num             id              +               -               /               *               $

E       { E->e,  }      { E->TE',  }    { E->TE',  }    { E->TE',  }    { E->TE',  }    { E->TE',  }    { E->TE',  }    { E->TE',  }    { E->e,  }
E'      { E'->e,  }     {       }       {       }       {       }       { E'->+TE',  }  { E'->-TE',  }  {       }       {       }       { E'->e,  }
T       { T->e,  }      { T->FT',  }    { T->FT',  }    { T->FT',  }    { T->e,  }      { T->e,  }      { T->FT',  }    { T->FT',  }    { T->e,  }
T'      { T'->e,  }     {       }       {       }       {       }       { T'->e,  }     { T'->e,  }     { T'->/FT',  }  { T'->*FT',  }  { T'->e,  }
F       { F->e,  }      { F->(E),  }    { F->num,  }    { F->id,  }     { F->e,  }      { F->e,  }      { F->e,  }      { F->e,  }      { F->e,  }
```