Name: Kush Munot
Roll No. 47

Practical No. 3

AIM - Write a program to find FIRST for any grammar. All the following rules of FIRST must be implemented.

For a generalized grammar: A □ αXY

FIRST (A) = FIRST (αXY)

= α if α is the terminal symbol (Rule-1)

= FIRST (α) if α is non-terminal and FIRST (α) does not contain ε (Rule-2)

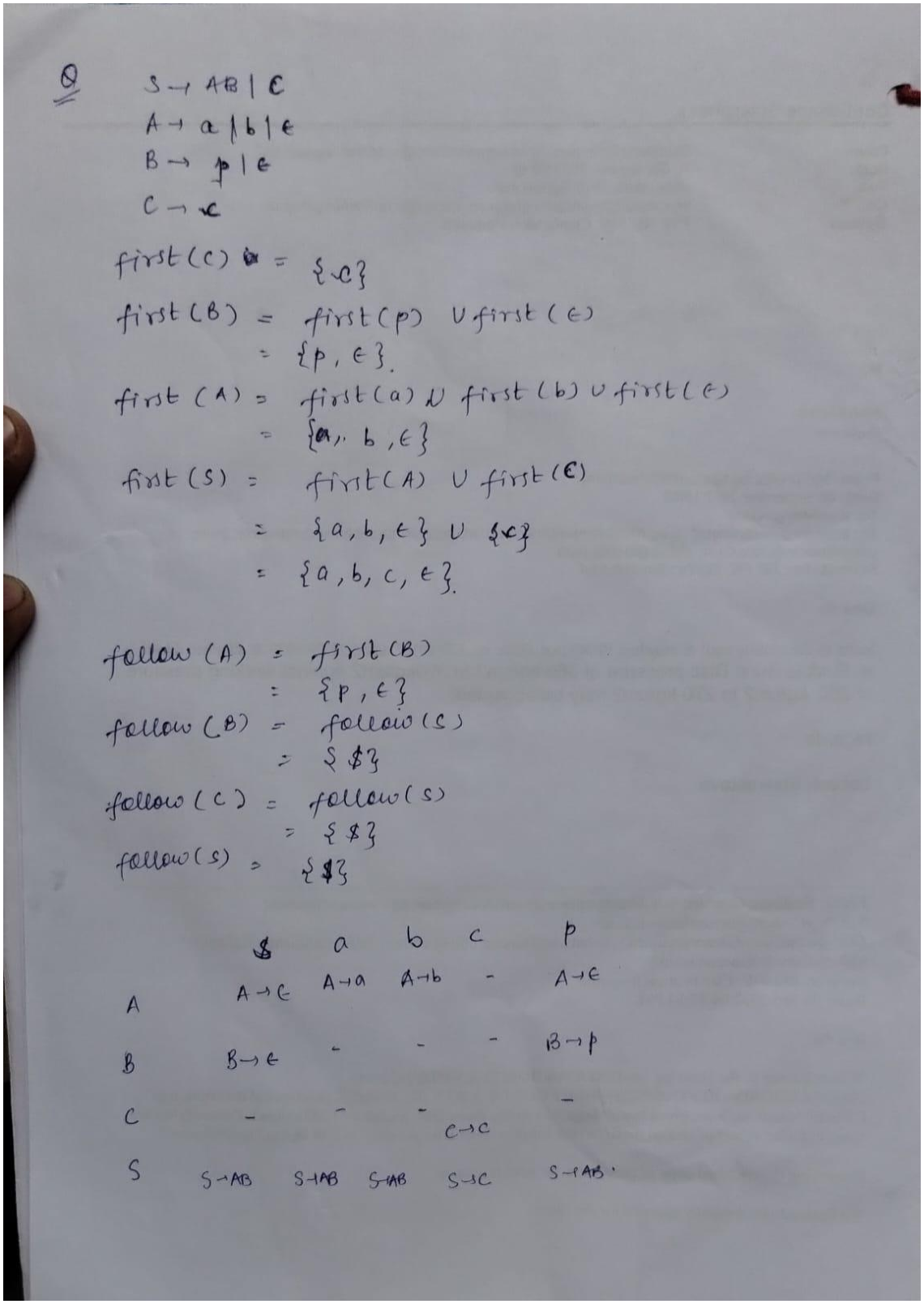
= FIRST (α) - ε U FIRST (XY) if a is a non-terminal and FIRST (α) contains ε (Rule-3)

Batch A3:

S AB | C
A a | b | □
B p | □
C c

Calculate Follow for the given grammar and Construct the LL (1) parsing table using the FIRST and FOLLOW.

Solved Problem



Q

$S \rightarrow AB \mid c$

$A \rightarrow a \mid b \mid \epsilon$

$B \rightarrow p \mid \epsilon$

$C \rightarrow c$

first(C) = {c}

first(B) = first(p) ∪ first(ε)
= {p, ε}.

first(A) = first(a) ∪ first(b) ∪ first(ε)
= {a, b, ε}

first(S) = first(A) ∪ first(c)
= {a, b, ε} ∪ {c}
= {a, b, c, ε}.

follow(A) = first(B)
= {p, ε}

follow(B) = follow(S)
= {$}

follow(C) = follow(S)
= {$}

follow(S) = {$}

	$	a	b	c	p
A	A→ε	A→a	A→b	-	A→ε
B	B→ε	-	-	-	B→p
C	-	-	-	C→c	-
S	S→AB	S→AB	S→AB	S→c	S→AB

Code

```
#include <iostream>
#include <fstream>
#include <vector>
#include <set>
#include <map>
#include <stack>
#include <bits/stdc++.h>

using namespace std;

void find_first(vector< pair<char, string> > gram,
      map< char, set<char> > &firsts,
      char non_term);

void find_follow(vector< pair<char, string> > gram,
      map< char, set<char> > &follows,
      map< char, set<char> > firsts,
      char non_term);


int main(int argc, char const *argv[])
{
      if(argc != 3) {
            cout<<"Arguments should be <grammar file> <input string>\n";
            return 1;
      }
      fstream grammar_file;
      grammar_file.open(argv[1], ios::in);
      if(grammar_file.fail()) {
            cout<<"Error in opening grammar file\n";
            return 2;
      }

      cout<<"Grammar parsed from grammar file: \n";
      vector< pair<char, string> > gram;
      int count = 0;
      while(!grammar_file.eof()) {
            char buffer[20];
            grammar_file.getline(buffer, 19);

            char lhs = buffer[0];
            string rhs = buffer+3;
            pair <char, string> prod (lhs, rhs);
            gram.push_back(prod);
            cout<<count++<<".                  "<<gram.back().first<<"        ->
"<<gram.back().second<<"\n";
      }
      cout<<"\n";

      set<char> non_terms;
      for(auto i = gram.begin(); i != gram.end(); ++i) {
            non_terms.insert(i->first);
      }
      cout<<"The non terminals in the grammar are: ";
```

```
	for(auto i = non_terms.begin(); i != non_terms.end(); ++i) {
		cout<<*i<<" ";
	}
	cout<<"\n";

	set<char> terms;
	for(auto i = gram.begin(); i != gram.end(); ++i) {
		for(auto ch = i->second.begin(); ch != i->second.end(); ++ch) {
			if(!isupper(*ch)) {
				terms.insert(*ch);
			}
		}
	}

	terms.erase('e');
	terms.insert('$');
	cout<<"The terminals in the grammar are: ";
	for(auto i = terms.begin(); i != terms.end(); ++i) {
		cout<<*i<<" ";
	}
	cout<<"\n\n";

	char start_sym = gram.begin()->first;


	map< char, set<char> > firsts;
	for(auto non_term = non_terms.begin(); non_term != non_terms.end();
++non_term) {
		if(firsts[*non_term].empty()){
			find_first(gram, firsts, *non_term);
		}
	}

	cout<<"Firsts list: \n";
	for(auto it = firsts.begin(); it != firsts.end(); ++it) {
		cout<<it->first<<" : ";
		for(auto firsts_it = it->second.begin(); firsts_it !=
it->second.end(); ++firsts_it) {
			cout<<*firsts_it<<" ";
		}
		cout<<"\n";
	}
	cout<<"\n";


	map< char, set<char> > follows;

	char start_var = gram.begin()->first;
	follows[start_var].insert('$');
	find_follow(gram, follows, firsts, start_var);

	for(auto it = non_terms.begin(); it != non_terms.end(); ++it) {
		if(follows[*it].empty()) {
			find_follow(gram, follows, firsts, *it);
		}
	}
```

```
	cout<<"Follows list: \n";
	for(auto it = follows.begin(); it != follows.end(); ++it) {
		cout<<it->first<<" : ";
		for(auto follows_it = it->second.begin(); follows_it != 
it->second.end(); ++follows_it) {
			cout<<*follows_it<<" ";
		}
		cout<<"\n";
	}
	cout<<"\n";
	return 0;
}

void find_first(vector< pair<char, string> > gram,
	map< char, set<char> > &firsts,
	char non_term) {

	for(auto it = gram.begin(); it != gram.end(); ++it) {

		if(it->first != non_term) {
			continue;
		}
		string rhs = it->second;

		for(auto ch = rhs.begin(); ch != rhs.end(); ++ch) {

			if(!isupper(*ch)) {
				firsts[non_term].insert(*ch);
				break;
			}
			else {

				if(firsts[*ch].empty()) {
					find_first(gram, firsts, *ch);
				}

				if(firsts[*ch].find('e') == firsts[*ch].end()) {
					firsts[non_term].insert(firsts[*ch].begin(),
firsts[*ch].end());
					break;
				}

				set<char>		firsts_copy(firsts[*ch].begin(),
firsts[*ch].end());


				if(ch + 1 != rhs.end()) {
					firsts_copy.erase('e');
				}


				firsts[non_term].insert(firsts_copy.begin(),
firsts_copy.end());
			}
		}

	}
```

```
}

void find_follow(vector< pair<char, string> > gram,
      map< char, set<char> > &follows,
      map< char, set<char> > firsts,
      char non_term) {

      for(auto it = gram.begin(); it != gram.end(); ++it) {

            bool finished = true;
            auto ch = it->second.begin();

            for(;ch != it->second.end() ; ++ch) {
                  if(*ch == non_term) {
                        finished = false;
                        break;
                  }
            }
            ++ch;

            for(;ch != it->second.end() && !finished; ++ch) {
                  if(!isupper(*ch)) {
                        follows[non_term].insert(*ch);
                        finished = true;
                        break;
                  }

                  set<char> firsts_copy(firsts[*ch]);

                  if(firsts_copy.find('e') == firsts_copy.end()) {
                        follows[non_term].insert(firsts_copy.begin(),
firsts_copy.end());
                        finished = true;
                        break;
                  }
                  firsts_copy.erase('e');
                  follows[non_term].insert(firsts_copy.begin(),
firsts_copy.end());

            }
            if(ch == it->second.end() && !finished) {
                  if(follows[it->first].empty()) {
                        find_follow(gram, follows, firsts, it->first);
                  }
                  follows[non_term].insert(follows[it->first].begin(),
follows[it->first].end());
            }

      }

}
```

Execution Output

```
~/cd-a-43-prac-3$ ./parser grammar2.txt bp
Grammar parsed from grammar file:
0.  S -> AB
1.  S -> C
2.  A -> a
3.  A -> b
4.  A -> e
5.  B -> p
6.  B -> e
7.  C -> c

The non-terminals in the grammar are: A B C S
The terminals in the grammar are: $ a b c p

Firsts list:
A : a b e
B : e p
C : c
S : a b c e p

Follows list:
A : $ p
B : $
C : $
S : $

Parsing Table:
   $ a b c p
A  4 2 3 - 4
B  6 - - - 5
C  - - - 7 -
S  0 0 0 1 0

Input string is accepted
```

```
~/cd-a-43-prac-3$ ./parser grammar2.txt bp
Grammar parsed from grammar file:
0.  S -> AB
1.  S -> C
2.  A -> a
3.  A -> b
4.  A -> e
5.  B -> p
6.  B -> e
7.  C -> c

The non terminals in the grammar are: A B C S
The terminals in the grammar are: $ a b c p

Firsts list:
A : a b e
B : e p
C : c
S : a b c e p

Follows list:
A : $ p
B : $
C : $
S : $

Parsing Table:
   $ a b c p
A  4 2 3 - 4
B  6 - - - 5
C  - - - 7 -
S  0 0 0 1 0

Input string is accepted
```