Name	Shubham Golwal
UID no.	2020300015
Experiment No.	04

AIM:	Program to find first and follow sets for the given grammar.
THEORY:	FIRST and FOLLOW are two functions associated with grammar that help us fill in the entries of an M-table. FIRST ()− It is a function that gives the set of terminals that begin the strings derived from the production rule. A symbol c is in FIRST (α) if and only if $\alpha \Rightarrow c\beta$ for some sequence β of grammar symbols. A terminal symbol a is in FOLLOW (N) if and only if there is a derivation from the start symbol S of the grammar such that $S \Rightarrow \alpha N\alpha\beta$, where α and β are a (possible empty) sequence of grammar symbols. In other words, a terminal c is in FOLLOW (N) if c can follow N at some point in a derivation. **Benefit of FIRST ( ) and FOLLOW ( )** • It can be used to prove the LL (K) characteristic of grammar. • It can be used to promote in the construction of predictive parsing tables. • It provides selection information for recursive descent parsers.

Computation of FIRST

FIRST (α) is defined as the collection of terminal symbols which are the first letters of strings derived from α.

FIRST (α) = {α |α →∗ αβ for some string β }

If X is Grammar Symbol, then First (X) will be −

If X is a terminal symbol, then FIRST(X) = {X}
If X → ε, then FIRST(X) = {ε}
If X is non-terminal & X → a α, then FIRST (X) = {a}
If X → Y1, Y2, Y3, then FIRST (X) will be
(a) If Y is terminal, then

FIRST (X) = FIRST (Y1, Y2, Y3) = {Y1}

(b) If Y1 is non-terminal and

If Y1 does not derive to an empty string i.e., If FIRST (Y1) does not contain ε then, FIRST (X) = FIRST (Y1, Y2, Y3) = FIRST(Y1)

(c) If FIRST (Y1) contains ε, then.

FIRST (X) = FIRST (Y1, Y2, Y3) = FIRST(Y1) − {ε} ∪ FIRST(Y2, Y3)

Similarly, FIRST (Y2, Y3) = {Y2}, If Y2 is terminal otherwise if Y2 is Non-terminal then

FIRST (Y2, Y3) = FIRST (Y2), if FIRST (Y2) does not contain ε.

If FIRST (Y2) contain ε, then

FIRST (Y2, Y3) = FIRST (Y2) − {ε} ∪ FIRST (Y3)

Similarly, this method will be repeated for further Grammar symbols, i.e., for Y4, Y5, Y6 … . YK.

Computation of FOLLOW

Follow (A) is defined as the collection of terminal symbols that occur directly to the right of A.

FOLLOW(A) = {a|S ⇒* αAaβ where α, β can be any strings}

Rules to find FOLLOW

If S is the start symbol, FOLLOW (S) ={$}

If production is of form A → α B β, β ≠ ε.

(a) If FIRST (β) does not contain ε then, FOLLOW (B) = {FIRST (β)}

Or

(b) If FIRST (β) contains ε (i. e. , β ⇒* ε), then

FOLLOW (B) = FIRST (β) − {ε} ∪ FOLLOW (A)

∵ when β derives ε, then terminal after A will follow B.

If production is of form A → αB, then Follow (B) ={FOLLOW (A)}.

EXAMPLE

Consider the expression grammar (4.11), repeated below:

E -> T E'
E'-> + T E' | e
T -> F T'
T'-> * F T' | e
F -> (E) | id

Then:

FIRST(E) = FIRST(T) = FIRST(F) = {(, id}
FIRST(E') = {+, e}
FIRST(T') = {*, e}
FOLLOW(E) = FOLLOW(E') = {) , $}
FOLLOW(T) = FOLLOW(T') = {+,), $}
FOLLOW(F) = {+, *,), $}

PROGRAM:

```
#include <iostream>
#include <unordered_map>
#include <vector>
#include <unordered_set>
using namespace std;

int n;
vector<char> nTer;
vector<int> nProd;
unordered_map<char, int> indices;
vector<vector<string>> prod;
vector<unordered_set<char>> *first, *follow;

void printer(vector<unordered_set<char>> *first);
unordered_set<char> calFirst(int s);
unordered_set<char> calFollow(int s);

int main()
{
    int temp;
    char ch;
    string str;

    cout << "First & Follow\n"
         << endl;

    // Examples:
    // ========================================

    n = 4;
    nTer.push_back('S');
    nTer.push_back('A');
    nTer.push_back('B');
    nTer.push_back('C');

    indices['S'] = 0;
    indices['A'] = 1;
    indices['B'] = 2;
    indices['C'] = 3;
```

```
nProd.push_back(3);
nProd.push_back(2);
nProd.push_back(2);
nProd.push_back(2);

prod.push_back({"ACB", "CbB", "Ba"});
prod.push_back({"da", "BC"});
prod.push_back({"g", "#"});
prod.push_back({"h", "#"});

// ========================================

// n = 3;
// nTer.push_back('S');
// nTer.push_back('A');
// nTer.push_back('B');

// indices['S'] = 0;
// indices['A'] = 1;
// indices['B'] = 2;

// nProd.push_back(2);
// nProd.push_back(1);
// nProd.push_back(1);

// prod.push_back({"AaAb", "BbBa"});
// prod.push_back({"#"});
// prod.push_back({"#"});

// ========================================

// n = 6;
// nTer.push_back('S');
// nTer.push_back('B');
// nTer.push_back('C');
// nTer.push_back('D');
// nTer.push_back('E');
// nTer.push_back('F');

// indices['S'] = 0;
```

```
// indices['B'] = 1;
// indices['C'] = 2;
// indices['D'] = 3;
// indices['E'] = 4;
// indices['F'] = 5;

// nProd.push_back(1);
// nProd.push_back(1);
// nProd.push_back(2);
// nProd.push_back(1);
// nProd.push_back(2);
// nProd.push_back(2);

// prod.push_back({"aBDh"});
// prod.push_back({"cC"});
// prod.push_back({"bC", "#"});
// prod.push_back({"EF"});
// prod.push_back({"g", "#"});
// prod.push_back({"f", "#"});

// ========================================

// Custom inputs:
// ========================================

// cout << "No. of non-terminals -> ";
// cin >> n;
// cout << endl;

// for (int i = 0; i < n; i++)
// {
//     cout << "Enter non-terminal -> ";
//     cin >> ch;
//     cout << "No. of productions for " << ch << " -> ";
//     cin >> temp;
//     indices[ch] = i;
//     nTer.push_back(ch);
//     nProd.push_back(temp);

//     vector<string> vec;
```

```
    //     for (int j = 0; j < nProd[i]; j++)
    //     {
    //         cout << "Enter productions for " << nTer[i] << " -> ";
    //         cin >> str;
    //         vec.push_back(str);
    //     }
    //     prod.push_back(vec);
    //     cout << endl;
    // }

    // ========================================

    first = new vector<unordered_set<char>>(n, unordered_set<char>());
    follow = new vector<unordered_set<char>>(n, unordered_set<char>());

    cout << "Productions:\n";

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

    for (int i = 0; i < 2; i++)
        for (int j = 0; j < n; j++)
            calFollow(j);

    cout << "\nFirst set:\n";
    printer(first);

    cout << "\nFollow set:\n";
    printer(follow);

    return 0;
}

void printer(vector<unordered_set<char>> *vec)
{
    int size = 0, current = 0;
    for (int i = 0; i < n; i++)
    {
        cout << nTer[i] << " = {";
        current = 0;
```

```
        size = vec->at(i).size();
        for (unordered_set<char>::iterator it = vec->at(i).begin(); it != vec-
>at(i).end(); it++)
        {
            cout << *it;
            current++;
            if (current != size)
                cout << ", ";
        }
        cout << "}" << endl;
    }
}

unordered_set<char> calFirst(int s)
{
    if (!first->at(s).empty())
        return first->at(s);

    unordered_set<char> distinct;

    cout << nTer[s] << " -> ";
    for (int i = 0; i < prod[s].size(); i++)
    {
        cout << prod[s][i];
        if (i != prod[s].size() - 1)
            cout << ", ";
    }
    cout << endl;

    for (auto &&str : prod[s])
    {
        for (int j = 0; j < str.size(); j++)
        {
            if (str[j] < 'A' || str[j] > 'Z')
            {
                distinct.insert(str[j]);
                break;
            }
            else
            {
```

```
            unordered_set<char> rec = calFirst(indices[str[j]]);

            bool found = false;
            for (unordered_set<char>::iterator it = rec.begin(); it != rec.end();
it++)
            {
                distinct.insert(*it);
                if (*it == '#')
                    found = true;
            }
            if (!found)
                break;
        }
    }
}

first->at(s) = distinct;

return distinct;
}

unordered_set<char> calFollow(int s)
{
    if (s == 0)
        follow->at(0).insert('$');

    for (auto &&str : prod[s])
    {
        for (int i = 0; i < str.size(); i++)
        {
            if (str[i] >= 'A' && str[i] <= 'Z')
            {
                unordered_set<char> distinct;
                if (i + 1 < str.size() && str[i + 1] != '#')
                {
                    for (int j = i + 1; j < str.size(); j++)
                    {
                        if (str[j] >= 'A' && str[j] <= 'Z')
                        {
                            bool found = false;
```

```
                    for (auto &&ch : first->at(indices[str[j]]))
                    {
                        if (ch == '#')
                        {
                            if (j == str.size() - 1)
                            {
                                unordered_set<char> parentFollow = follow->at(s);
                                distinct.insert(parentFollow.begin(),
parentFollow.end());
                            }
                            found = true;
                        }
                        else
                            distinct.insert(ch);
                    }
                    if (!found)
                        break;
                }
                else
                    distinct.insert(str[j]);
            }
        }
        else
        {
            unordered_set<char> parentFollow = follow->at(s);
            distinct.insert(parentFollow.begin(), parentFollow.end());
        }
        for (auto &&ch : distinct)
            follow->at(indices[str[i]]).insert(ch);
    }
  }

  return follow->at(s);
}
```

RESULT:

Input:

```
n = 4;
nTer.push_back('S');
```

```
nTer.push_back('A');
nTer.push_back('B');
nTer.push_back('C');

indices['S'] = 0;
indices['A'] = 1;
indices['B'] = 2;
indices['C'] = 3;

nProd.push_back(3);
nProd.push_back(2);
nProd.push_back(2);
nProd.push_back(2);

prod.push_back({"ACB", "CbB", "Ba"});
prod.push_back({"da", "BC"});
prod.push_back({"g", "#"});
prod.push_back({"h", "#"});
```

Output:

```
Productions:
S -> ACB, CbB, Ba
A -> da, BC
B -> g, #
C -> h, #

First set:
S = {b, #, d, a, h, g}
A = {h, g, d, #}
B = {#, g}
C = {#, h}

Follow set:
S = {$}
A = {$, g, h}
B = {g, $, h, a}
C = {h, b, g, $}
```

Input:

```
n = 3;
nTer.push_back('S');
nTer.push_back('A');
nTer.push_back('B');

indices['S'] = 0;
indices['A'] = 1;
indices['B'] = 2;
```

```
nProd.push_back(2);
nProd.push_back(1);
nProd.push_back(1);

prod.push_back({"AaAb", "BbBa"});
prod.push_back({"#"});
prod.push_back({"#"});
```

Output:

```
First & Follow

Productions:
S -> AaAb, BbBa
A -> #
B -> #

First set:
S = {b, #, a}
A = {#}
B = {#}

Follow set:
S = {$}
A = {a, b}
B = {b, a}
```

CONCLUSION:	• Learned how to find the first and follow sets for any given grammar. • Used this knowledge to make a bare minimum code to find the first and follow of any input production rules. • First and follow sets are calculated as a first step in making of LR parser.
REFERENCES:	• https://www.tutorialspoint.com/what-are-first-and-follow-and-how-they-are-computed#:~:text=FIRST%20and%20FOLLOW%20are%20two,sequence%20%CE%B2%20of%20grammar%20symbols. • https://www.gatevidyalay.com/first-and-follow-compiler-design/ • https://www.cs.uaf.edu/~cs331/notes/FirstFollow.pdf