GHARDA FOUNDATION'S
GHARDA INSTITUTE OF TECHNOLOGY, LAVEL
COMPUTER ENGINEERING DEPARTMENT
A/P: Lavel, Tal.Khed Dist. Ratnagiri

Evaluation Sheet

Class: TE-Computer

Semester: VI

Subject: **System Programming & Compiler Construction**

Experiment No: 3

Title of Experiment: Program to Compute first() and follow() set of the grammar.

Sr. No.	Evaluation Criteria	Max Marks	Marks Obtained
1	Practical Performance	10	
2	Oral	5	
	Total	15	

Signature of Subject Teacher

Theory:

First and Follow are two important concepts in the field of system programming and compiler construction. They are used in the process of designing and constructing parsers for programming languages.

First Set:

The First Set of a grammar symbol is the set of all possible terminal symbols that can appear as the first symbol in any string derived from that grammar symbol. In other words, it is the set of all symbols that can occur in the first position of any valid sentence generated by the grammar symbol.

For example, consider the following grammar rule:

$A \rightarrow BCd \mid \varepsilon$

The First Set of A is {B, ε} because the first symbol of any sentence derived from A can be either B or ε.

Follow Set:

The Follow Set of a grammar symbol is the set of all possible terminal symbols that can appear immediately after that grammar symbol in any string derived from the grammar. In other words, it is the set of all symbols that can occur in the position immediately following the grammar symbol in any valid sentence generated by the grammar.

For example, consider the following grammar rules:

$S \rightarrow AaBb \mid Cc$

$A \rightarrow d$

$B \rightarrow e$

$C \rightarrow f$

The Follow Set of A is {a} because the symbol 'a' can only appear immediately after A in any sentence derived from the grammar. The Follow Set of B is {b} because 'b' can only appear immediately after B in any sentence derived from the grammar.

How do First and Follow sets help in constructing efficient parsers?

The First and Follow sets play a crucial role in constructing efficient parsers for programming languages. They help in reducing the amount of backtracking and lookahead required during the parsing process, which can significantly improve the parser's performance.

Here are some ways in which the First and Follow sets help in constructing efficient parsers:

1. Predictive Parsing: The First and Follow sets are used in constructing predictive parsers, which can predict the next production rule to apply without any backtracking. Predictive parsing is more efficient than other parsing techniques because it eliminates the need for backtracking, which can be time-consuming and resource-intensive.
2. Error Recovery: The Follow set can also be used to recover from parsing errors. When the parser encounters an error, it can use the Follow set to skip over the erroneous input and resume parsing at the next valid position. This technique is known as error recovery, and it can help the parser to continue parsing even when errors occur.
3. Ambiguity Resolution: The First and Follow sets can also be used to resolve grammar ambiguities. When a grammar has multiple valid parse trees for a given input, the parser can use the First and Follow sets to choose the most appropriate parse tree. This can help to ensure that the parser produces the correct output and avoids any unintended behavior.

In summary, the First and Follow sets provide essential information that can be used to construct efficient parsers for programming languages. By reducing backtracking, enabling error recovery, and resolving grammar ambiguities, these sets help to ensure that the parser produces accurate and reliable output while minimizing the time and resources required for the parsing process.

Program Code –

```
import sys
sys.setrecursionlimit(100)


def first(string):
    first_ = set()
    if string in non_terminals:
        alternatives = productions_dict[string]


        for alternative in alternatives:
            first_2 = first(alternative)
            first_ = first_ |first_2


    elif string in terminals:
        first_ = {string}


    elif string=='' or string=='@':
        first_ = {'@'}


    else:
        first_2 = first(string[0])
        if '@' in first_2:
            i = 1
            while '@' in first_2:
                first_ = first_ | (first_2 - {'@'})
                if string[i:] in terminals:
                    first_ = first_ | {string[i:]}
                    break
                elif string[i:] == '':
                    first_ = first_ | {'@'}
```

```
                    break
                first_2 = first(string[i:])
                first_ = first_ | first_2 - {'@'}
                i += 1
        else:
            first_ = first_ | first_2

    return  first_

def follow(nT):
    follow_ = set()
    prods = productions_dict.items()
    if nT==starting_symbol:
        follow_ = follow_ | {'$'}
    for nt,rhs in prods:
        for alt in rhs:
            for char in alt:
                if char==nT:
                    following_str = alt[alt.index(char) + 1:]
                    if following_str=='':
                        if nt==nT:
                            continue
                        else:
                            follow_ = follow_ | follow(nt)
                    else:
                        follow_2 = first(following_str)
                        if '@' in follow_2:
                            follow_ = follow_ | follow_2-{'@'}
                            follow_ = follow_ | follow(nt)
                        else:
```

```
                            follow_ = follow_ | follow_2
    return follow_


no_of_terminals=int(input("Enter no. of terminals: "))
terminals = []
print("Enter the terminals :")
for _ in range(no_of_terminals):
    terminals.append(input())

no_of_non_terminals=int(input("Enter no. of non terminals: "))
non_terminals = []
print("Enter the non terminals :")
for _ in range(no_of_non_terminals):
    non_terminals.append(input())

starting_symbol = input("Enter the starting symbol: ")
no_of_productions = int(input("Enter no of productions: "))
productions = []
print("Enter the productions:")
for _ in range(no_of_productions):
    productions.append(input())

productions_dict = {}
for nT in non_terminals:
    productions_dict[nT] = []

for production in productions:
    nonterm_to_prod = production.split("->")
    alternatives = nonterm_to_prod[1].split("/")
    for alternative in alternatives:
```

```
        productions_dict[nonterm_to_prod[0]].append(alternative)


FIRST = {}
FOLLOW = {}


for non_terminal in non_terminals:
    FIRST[non_terminal] = set()


for non_terminal in non_terminals:
    FOLLOW[non_terminal] = set()


for non_terminal in non_terminals:
    FIRST[non_terminal] = FIRST[non_terminal] | first(non_terminal)


FOLLOW[starting_symbol] = FOLLOW[starting_symbol] | {'$'}
for non_terminal in non_terminals:
    FOLLOW[non_terminal] = FOLLOW[non_terminal] | follow(non_terminal)


print("{: ^20}{: ^20}{: ^20}".format('Non Terminals','First','Follow'))
for non_terminal in non_terminals:

print("{:^20}{:^20}{:^20}".format(non_terminal,str(FIRST[non_terminal]),
str(FOLLOW[non_terminal])))
```

Output –

```
Enter no. of terminals: 4
Enter the terminals :
a
b
c
d
Enter no. of non terminals: 3
Enter the non terminals :
S
A
B
Enter the starting symbol: S
Enter no of productions: 3
Enter the productions:
S->aABb
A->c/@
B->d/@
   Non Terminals          First             Follow
         S               {'a'}              {'$'}
         A             {'@', 'c'}         {'b', 'd'}
         B             {'@', 'd'}          {'b'}
```

Conclusion –

Thus we implemented the program to compute first() and follow() set of the grammar.