Indian Institute of Information Technology, Design & Manufacturing, Kurnool (IIITDMK)

Department of Computer Science and Engineering

Compiler Design Practice

Assignment 3

1. Write a program to implement
(a) Write a program to find FIRST
(b) Write a program to find FOLLOW
(iii) Implementation of LL(1) Parsing Table
Give the parsing process to show that it is neither using backtracking nor using other alternatives.
a. Sample Input and Output

```
Enter no of productions (epsilon (@)):8
Enter the production no 1:E->TP
Enter the production no 2:P->+TP
Enter the production no 3:P->@
Enter the production no 4:T->FQ
Enter the production no 5:Q->*FQ
Enter the production no 6:Q->@
Enter the production no 7:F->(E)
Enter the production no 8:F->a
The productions are:
E->TP
P->+TP
P->@
T->FQ
Q->*FQ
Q->@
F->(E)
F->a
--------------------------------
first(E) ( a
first(P) + @
first(T) ( a
first(Q) * @
first(F) ( a
```

2. Write a program for recursive descent parsing for the following grammar

E->TE'

E'->+TE/@

"@ represents null character"

T->FT'

T`->*FT'/@

F->(E)/ID

Sample Input and Output

Recursive descent parsing for the following grammar

E->TE'

E'->+TE'/@

T->FT'

T'->*FT'/@

F->(E)/ID

Enter the string to be checked:(a+b)*c

String is accepted

Recursive descent parsing for the following grammar

E->TE'
E'->+TE'/@
T->FT'
T'->*FT'/@
F->(E)/ID
Enter the string to be checked:a/c+d
String is not accepted

Grammar Rules for Question 3 & 4

1. E -> E + T
2. E -> T
3. T -> T * F
4. T -> F
5. F -> (E)
6. F -> id

- Non-terminal symbols: E, T, F
- Terminal symbols: +, *, (,), id

Problem 3: Implement a Simple Shift-Reduce Parser

You are required to implement a bottom-up parser for a given grammar. The grammar consists of a set of production rules, and the parser should use the shift-reduce technique to recognize whether a given input string is valid or not.

Your task is to write a C program that takes as input a grammar, an input string, and outputs whether the string is valid according to the grammar or not. Additionally, provide the sequence of actions taken by the parser (shift or reduce) to arrive at the final decision.

Input String: id + id * id

Parsing Actions:
Shift id
Reduce F -> id
Shift +
Shift id
Reduce F -> id
Shift *
Shift id
Reduce F -> id
Reduce T -> F
Shift +
Shift id
Reduce F -> id
Reduce T -> T * F
Reduce E -> T
Reduce E -> E + T
Valid Expression

Problem 4: Handle Operator Precedence in a Bottom-Up Parser

Extend the simple shift-reduce parser from Problem 1 to handle operator precedence. The grammar now includes operators with different precedence levels. Your parser should correctly parse expressions respecting the operator precedence and associativity rules.

Write a C program that takes a grammar, an input expression, and outputs whether the expression is syntactically valid. Include the sequence of actions taken by the parser, highlighting the use of operator precedence in the parsing process.

Parsing Actions:
Shift id
Shift +
Shift id
Shift *
Shift id
Reduce F -> id
Reduce T -> F
Reduce E -> T
Shift +
Shift id
Shift *
Shift id
Reduce F -> id
Reduce T -> F
Reduce E -> T

Reduce E -> E + T
Valid Expression