Name: Hammad Shabbir, Iqrash Qureshi

ID: 22I1140, 22I-1174

CS-F

Assignment Report

Original CFG:

Contents

Left Factoring

```
Performing Left Factoring...
Factored E with prefix 'a'
Created new non-terminal: A

Context-Free Grammar:
E -> aA
G -> G+E | b
L -> d
A -> c | x

Select an action:
1. Display CFG
2. Perform Left Factoring
3. Perform Left Recursion Removal
4. Compute First Sets (Non-terminals only)
5. Compute Follow Sets
6. Generate LL(1) Parsing Table
7. Exit
Enter your choice: 3

Performing Left Recursion Removal...
```

Left Recursion:

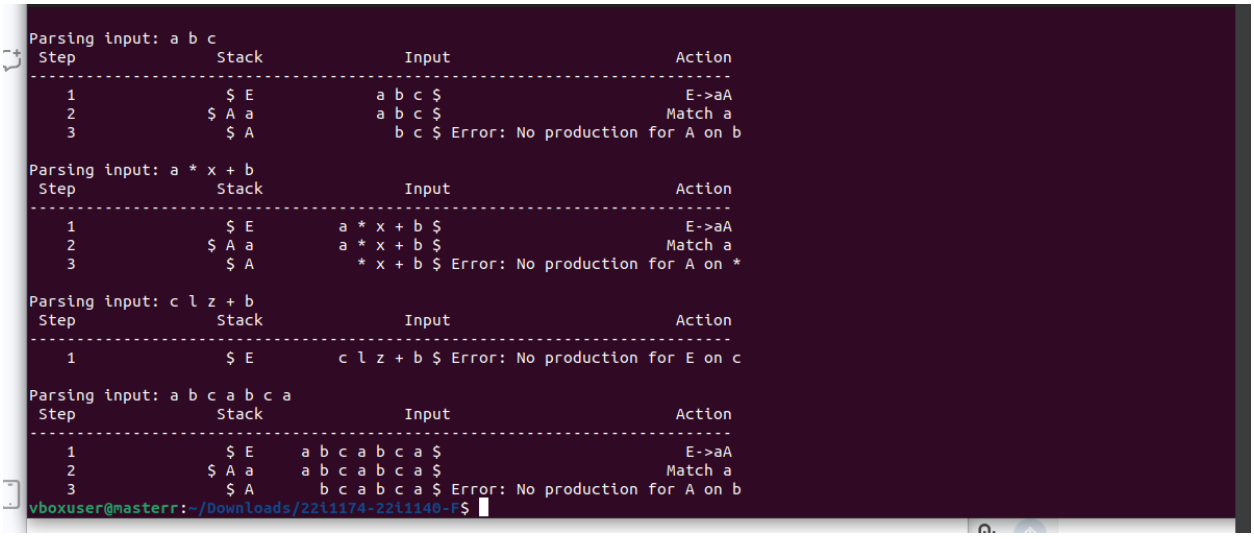
```
Enter your choice: 3

Performing Left Recursion Removal...
Left recursion removed for: G

Context-Free Grammar:
E -> aA
G -> bC
L -> d
A -> c | x
C -> +EC | #

Select an action:
1. Display CFG
2. Perform Left Factoring
3. Perform Left Recursion Removal
4. Compute First Sets (Non-terminals only)
5. Compute Follow Sets
6. Generate LL(1) Parsing Table
7. Exit
```

First & Follow

```
Computing FIRST sets...

FIRST sets:
FIRST(A) = { c x }
FIRST(C) = { # + }
FIRST(E) = { a }
FIRST(G) = { b }
FIRST(L) = { d }

Computing FOLLOW sets...

FOLLOW sets:
FOLLOW(A) = { $ + }
FOLLOW(C) = { }
FOLLOW(E) = { $ + }
FOLLOW(G) = { }
FOLLOW(L) = { }

Generating LL(1) Parsing Table...
```

Parsing Table:

Parsing Stack with Error



Successfully Parse

```
Parsing input: a b c
 Step              Stack              Input                      Action
-------------------------------------------------------------------------
    1                $ E            a b c $                     E->abA
    2            $ A b a            a b c $                    Match a
    3              $ A b              b c $                    Match b
    4                $ A                c $                      A->c
    5                $ c                c $                    Match c
    6                  $                  $                    Match $

Parsing completed successfully!
```

Another Example:

E -> abc | abx

H -> xyz | xyp

G -> G + E | ab

J -> J * H | ef

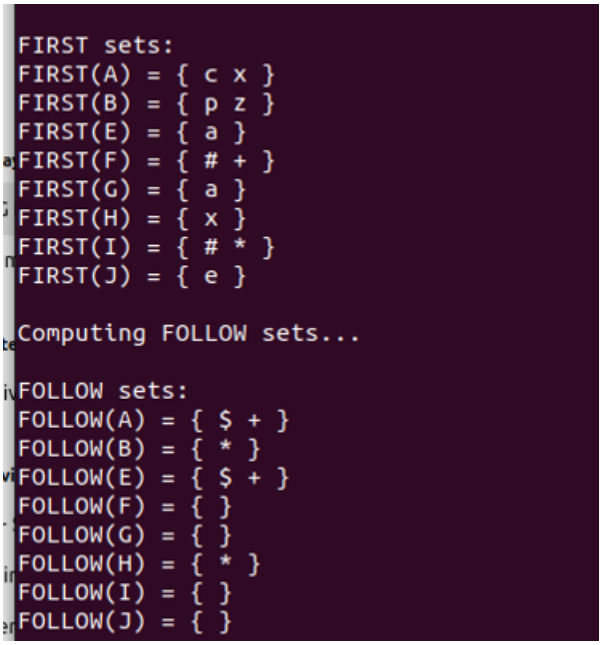
After Factoring:

```
Context-Free Grammar:
E -> abA
H -> xyB
G -> G+E | ab
J -> J*H | ef
A -> c | x
B -> z | p
```

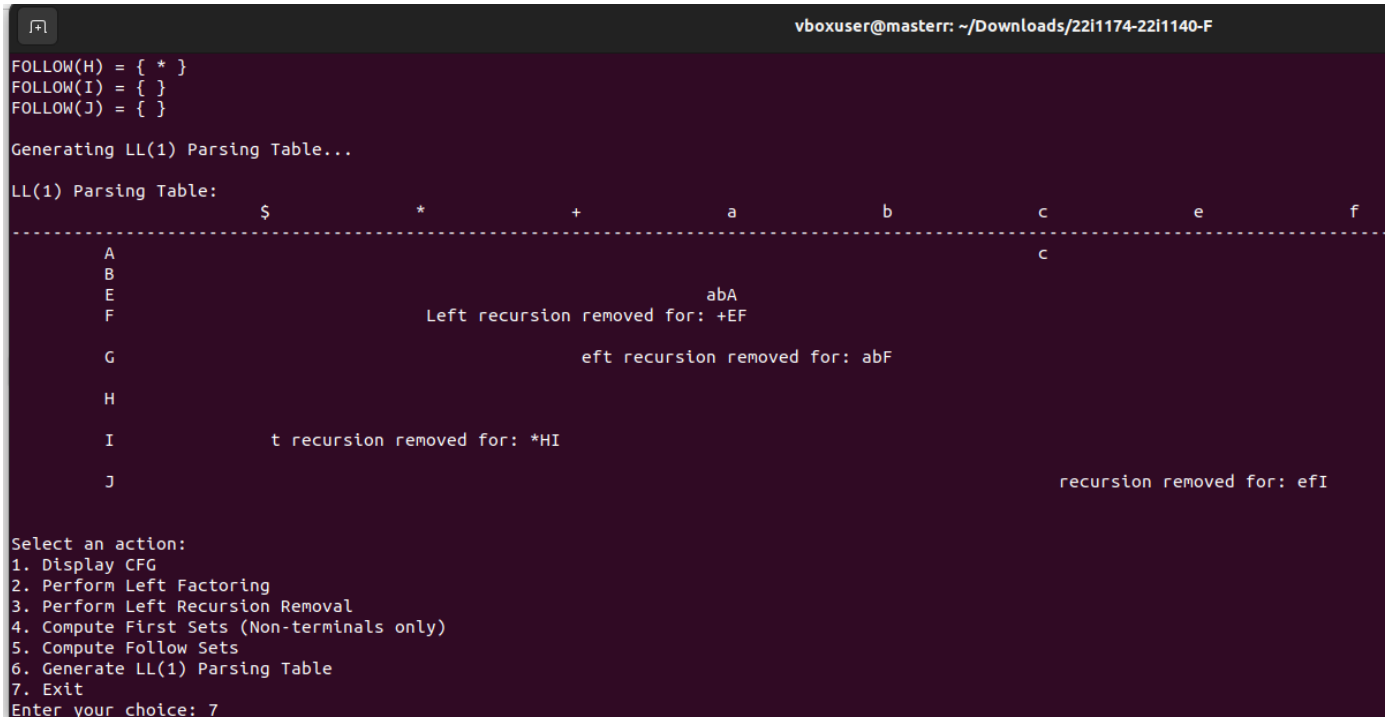
After Recursion:

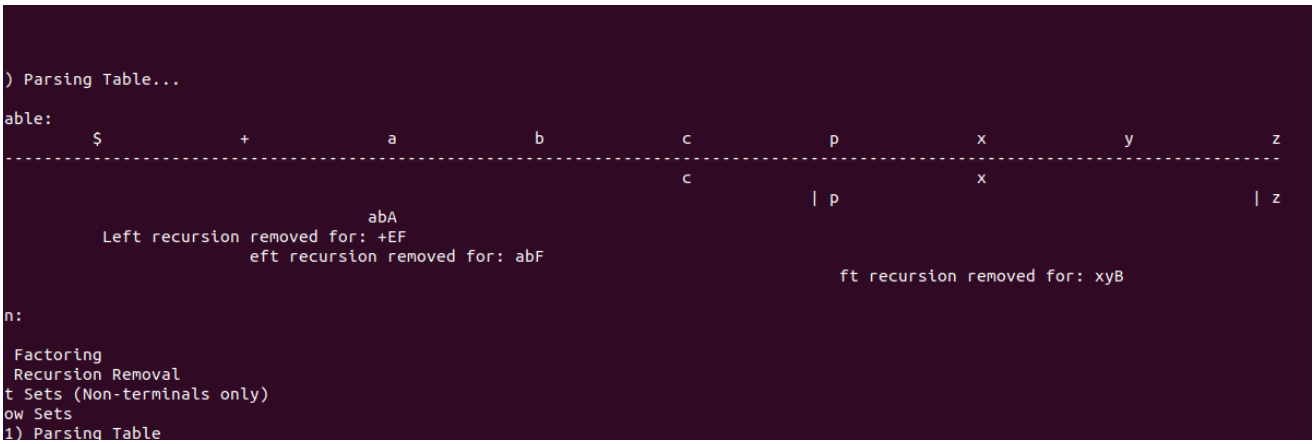
```
Context-Free Grammar:
E -> abA
H -> xyB
G -> abF
J -> efI
A -> c | x
B -> z | p
F -> +EF | #
I -> *HI | #
```

First & Follow



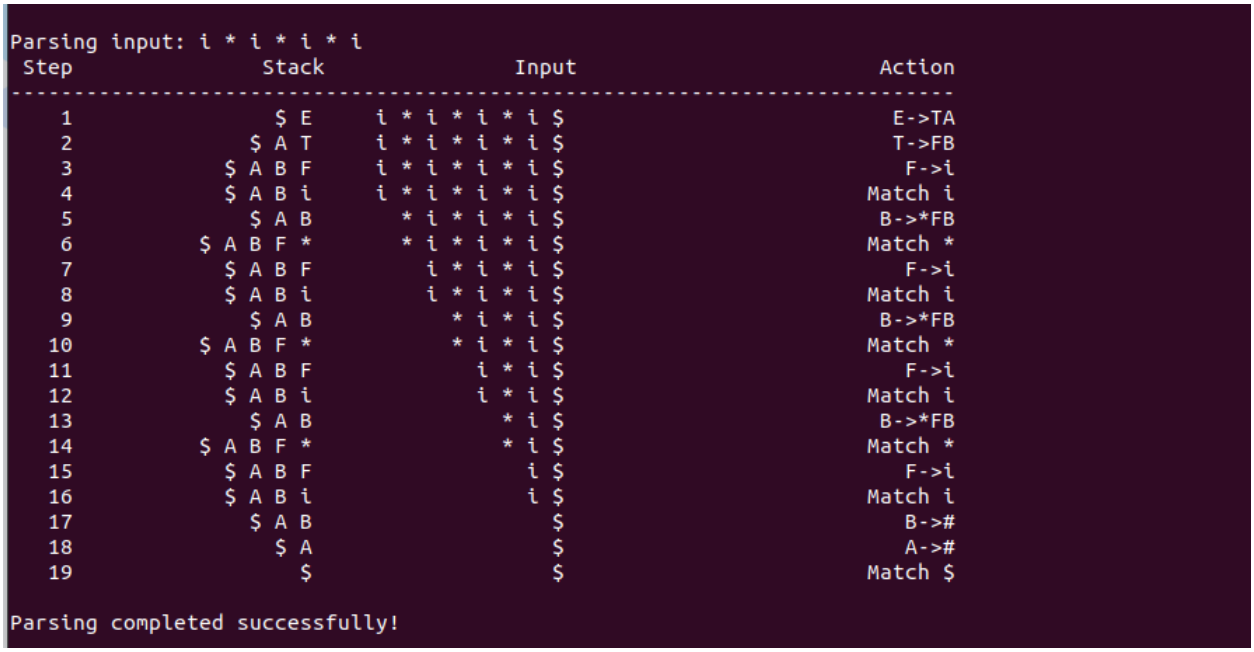
Parsing Table:





3rd Example:

```
1 E -> E + T | T
2 T -> T * F | F
3 F -> ( E ) | i
```



```
Parsing input: i * i * i * i
 Step              Stack                Input                          Action
-------------------------------------------------------------------------------
    1                $ E      i * i * i * i $                          E->TA
    2              $ A T      i * i * i * i $                          T->FB
    3            $ A B F      i * i * i * i $                            F->i
    4            $ A B i      i * i * i * i $                         Match i
    5              $ A B        * i * i * i $                          B->*FB
    6          $ A B F *        * i * i * i $                         Match *
    7            $ A B F          i * i * i $                            F->i
    8            $ A B i          i * i * i $                         Match i
    9              $ A B            * i * i $                          B->*FB
   10          $ A B F *            * i * i $                         Match *
   11            $ A B F              i * i $                            F->i
   12            $ A B i              i * i $                         Match i
   13              $ A B                * i $                          B->*FB
   14          $ A B F *                * i $                         Match *
   15            $ A B F                  i $                            F->i
   16            $ A B i                  i $                         Match i
   17              $ A B                    $                            B->#
   18                $ A                    $                            A->#
   19                  $                    $                         Match $

Parsing completed successfully!
```

Report: CFG Processing - Left Recursion Removal, Left Factoring, FIRST & FOLLOW Sets

1. Approach

The main goal of the project was to read a context-free grammar (CFG) from a file and transform it to make it suitable for LL(1) parsing by:

- Removing left recursion (both immediate and indirect).
- Performing left factoring to eliminate ambiguity.
- Computing FIRST and FOLLOW sets for all non-terminals.

1.1 Reading the CFG

- Designed a function readCFGFromFile() to read grammar rules from a text file.
- Each line represents one non-terminal's productions, e.g., S -> Sa | b.
- Skips whitespace and comments (after a # symbol).
- Non-terminals and terminals are stored separately using set<char>.

1.2 Removing Left Recursion

- Immediate left recursion (e.g., A -> Aα | β) handled by introducing a new non-terminal A':
    - New productions:
    - A -> βA'
      A' -> αA' | ε

- Indirect left recursion was handled by substituting previously defined rules into current productions before solving immediate recursion.
- Functions:
    - performLeftRecursionRemoval()
    - solveImmediateLeftRecursion(int index)

1.3 Left Factoring

- When two productions share a common prefix, grammar was factored:
    - A new non-terminal was created for the differing parts.
    - Example:
    - A -> abcd | abef
    - After factoring:
    - A -> abA'
      A' -> cd | ef

- Function:
    - performLeftFactoring()
- Loop used to repeatedly factor until no common prefixes remain.

1.4 Computing FIRST and FOLLOW sets

- FIRST Set: Terminals that can appear at the beginning of a string derived from a non-terminal.
- FOLLOW Set: Terminals that can appear immediately after a non-terminal in some derivation.
- Recursive definitions implemented carefully with iterative updates until no changes occurred.

Functions:

- computeFirstSets()
- computeFollowSets()

2. Challenges Faced

Challenge	How It Was Overcome
Dealing with indirect left recursion	Carefully replaced indirect recursion first, then handled direct recursion.

Choosing unique non-terminal symbols for new productions	Implemented getUniqueNonTerminal() cycling from 'A' to 'Z'.
Managing empty productions (epsilon #) during computations	Explicitly handled epsilon in FIRST and FOLLOW set rules.
Avoiding infinite loops during FIRST/FOLLOW computation	Used a changed flag and looped until convergence.
Updating grammar rules properly during factoring	Used temporary data structures to manage productions safely.

3. Verifying Correctness

3.1 Testing with Example Inputs

- Tested grammars included immediate recursion, indirect recursion, and left factoring needs.
- Sample grammar:
- S -> Sa | b
  A -> Ac | Sd | ε

3.2 Manual Dry Runs

- Manually computed expected results.
- Compared against program outputs.

3.3 Output Verification

- Printed modified CFG after each step.
- Printed FIRST and FOLLOW sets.
- Ensured no direct left recursion or ambiguity remained.
- Confirmed FIRST and FOLLOW sets matched theoretical calculations.

4. Conclusion

- Successfully implemented CFG reading, left recursion removal, left factoring, and FIRST/FOLLOW set computation.
- Solved challenges related to recursion, unique symbol generation, and epsilon handling.
- Verified correctness through theoretical comparison and manual dry runs.