MARWADI UNIVERSITY
DEPARTMENT OF COMPUTER
ENGINEERING CLASS: TC3 BATCH: A

COMPILER DESIGN

(01CE0714)

2025-2026

STUDENT LAB MANUAL

Samath Chavda 92200103165

INDEX

Sr. No.	Title	Date	Grade	Sign
1	Write a C Program to remove Left Recursion from grammar			
2	Write a C Program to remove Left Factoring from grammar			
3	Write a C program to implement finite automata and string validation.			
4	Prepare report for Lex and install Lex on Linux/Windows			
5	(a) WALEx Program to count words, characters, lines, Vowels and consonants from given input (b) WALEx Program to generate string which is ending with zeros.			
6	(a) WALex Program to generate Histogram of words (b) WALex Program to remove single or multi line comments from C program.			
7	WALex Program to check whether given statement is compound or simple.			
8	WALex Program to extract HTML tags from .html file.			
9	Write a C Program to compute FIRST Set of the given grammar			
10	Write a C Program to compute FOLLOW Set of the given grammar			
11	Write a C Program to implement Operator precedence parser			
12	Write a C Program for constructing LL (1) parsing			
13	Write a C program to implement SLR parsing			
14	Prepare a report on YACC and generate Calculator Program using YACC.			

Practical 1

Title: Write a C Program to remove Left Recursion from the grammar

Hint : The program reads a grammar production, checks for left recursion, extracts `α` and `β`, and then constructs and prints a new grammar without left recursion using the transformations \(A \rightarrow βA' \) and \(A' \rightarrow αA' | \epsilon \).

Program :

```
#include <stdio.h>
#define SIZE 10

void main() {
   char non_terminal;
   char beta, alpha[6];
   char production[SIZE];
   int index = 3;
   int i = 0, j = 0;

   printf("Name: Samath Chavda\n");
   printf("Enrollment No: 92200103165\n\n");
   printf("Enter the grammar (e.g., A->Aa|b):\n");
   scanf("%s", production);
   non_terminal = production[0];
   if (non_terminal == production[index]) {
      // Extract alpha (recursive part)
      for (i = index + 1; production[i] != '|' && production[i] != '\0'; i++) { alpha[j]
         = production[i];
         j++;
      }
      alpha[j] = '\0';

      printf("Grammar is left recursive.\n");

      while (production[index] != '\0' && production[index] != '|')
         index++;
      if (production[index] != '\0') {
```

```
        beta = production[index + 1];
        printf("Grammar without left recursion:\n");
        printf("%c -> %c%c'\n", non_terminal, beta, non_terminal);
        printf("%c' -> %s%c' | ε\n", non_terminal, alpha, non_terminal);
      } else {
        printf("Grammar can't be reduced.\n");
      }
    } else {
      printf("Grammar is not left recursive.\n");
    }
}
```

Output:

```
sql

Name: Samarth Chavda
Enrollment No: 92200103165

Enter the grammar (e.g., A->Aa|b):
Grammar is left recursive.
Grammar without left recursion:
A -> bA'
A' -> aA' | ε
```

Practical 2

Title: Write a C Program to remove Left Factoring from the grammar

Hint : This program reads a production of the form A->part1|part2, finds the common prefix in part1 and part2, and then restructures the grammar to eliminate left factoring.

Program:

```
#include <stdio.h>
#include <string.h>
int main() {
   char gram[20], part1[20], part2[20], modifiedGram[20], newGram[20];
   int i, j = 0, k = 0, pos = 0;
   printf("Name: Samath Chavda\n");
   printf("Enrollment No: 92200103165\n");
   printf("\nEnter Production : A->");
   scanf("%s", gram);
   for (i = 0; gram[i] != '|' && gram[i] != '\0'; i++, j++)
      { part1[j] = gram[i];
   }
   part1[j] = '\0';
   i++; // Skip '|'
   for (j = 0; gram[i] != '\0'; i++, j++) { part2[j]
      = gram[i];
   }
   part2[j] = '\0';
   for (i = 0; part1[i] != '\0' && part2[i] != '\0'; i++)
      { if (part1[i] == part2[i]) {
         modifiedGram[k++] = part1[i];
         pos = i + 1;
      } else {
         break;
      }
   }
   if (k == 0) {
      printf("\nNo left factoring required.\n");
      return 0;
   }
   modifiedGram[k++] = 'X';
   modifiedGram[k] = '\0';

   j = 0;
```

```
    for (i = pos; part1[i] != '\0'; i++)
        { newGram[j++] = part1[i];
    }
    newGram[j++] = '|';
    for (i = pos; part2[i] != '\0'; i++)
        { newGram[j++] = part2[i];
    }
    newGram[j] = '\0';
    printf("\nGrammar Without Left Factoring:\n");
    printf("A -> %s\n", modifiedGram);
    printf("X -> %s\n", newGram);

    return 0;
}
```

Output:

```
rust

Name: Samarth Chavda
Enrollment No: 92200103165

Enter Production (e.g., A->abX|abY):
Grammar Without Left Factoring:
A -> abX
X -> X | Y
```