**PRACTICAL – 6**

|  |  |
| --- | --- |
|  | **Problem statement**  **Write A Program To Left factor the given grammar** |

**SOLUTION:**

|  |
| --- |
| **CODE:**  #include <stdio.h>  #include <string.h>  #define MAX 100  void leftFactor(char \*input, char \*output) {  char nonTerminal;  char productions[MAX][MAX];  int prodCount = 0;  int i, j, k, l;  char commonPrefix[MAX];  char newNonTerminal;  char newProductions[MAX][MAX];  int newProdCount = 0;  // Parse the input grammar  sscanf(input, "%c->%s", &nonTerminal, productions[prodCount]);  prodCount++;  // Find common prefix  for (i = 0; i < strlen(productions[0]); i++) {  commonPrefix[i] = productions[0][i];  for (j = 1; j < prodCount; j++) {  if (productions[j][i] != commonPrefix[i]) {  commonPrefix[i] = '\0';  break;  }  }  if (commonPrefix[i] == '\0') {  break;  }  }  // Create new non-terminal  newNonTerminal = 'E';  // Create new productions  for (i = 0; i < prodCount; i++) {  if (strncmp(productions[i], commonPrefix, strlen(commonPrefix)) == 0) {  strcpy(newProductions[newProdCount], productions[i] + strlen(commonPrefix));  newProdCount++;  }  }  // Output the transformed grammar  sprintf(output, "%c->%s%c\n%c->", nonTerminal, commonPrefix, newNonTerminal, newNonTerminal);  for (i = 0; i < newProdCount; i++) {  strcat(output, newProductions[i]);  if (i < newProdCount - 1) {  strcat(output, "|");  }  }  }  int main()  {  char input[MAX];  char output[MAX] = "";  printf("Enter the grammar: ");  fgets(input, MAX, stdin);  input[strcspn(input, "\n")] = '\0'; // Remove trailing newline  leftFactor(input, output);  printf("Transformed Grammar:\n%s\n", output);  return 0;  }  **OUTPUT:** |