LP - LAB (TASK3)

P. Prasuna
411863
III - CSE

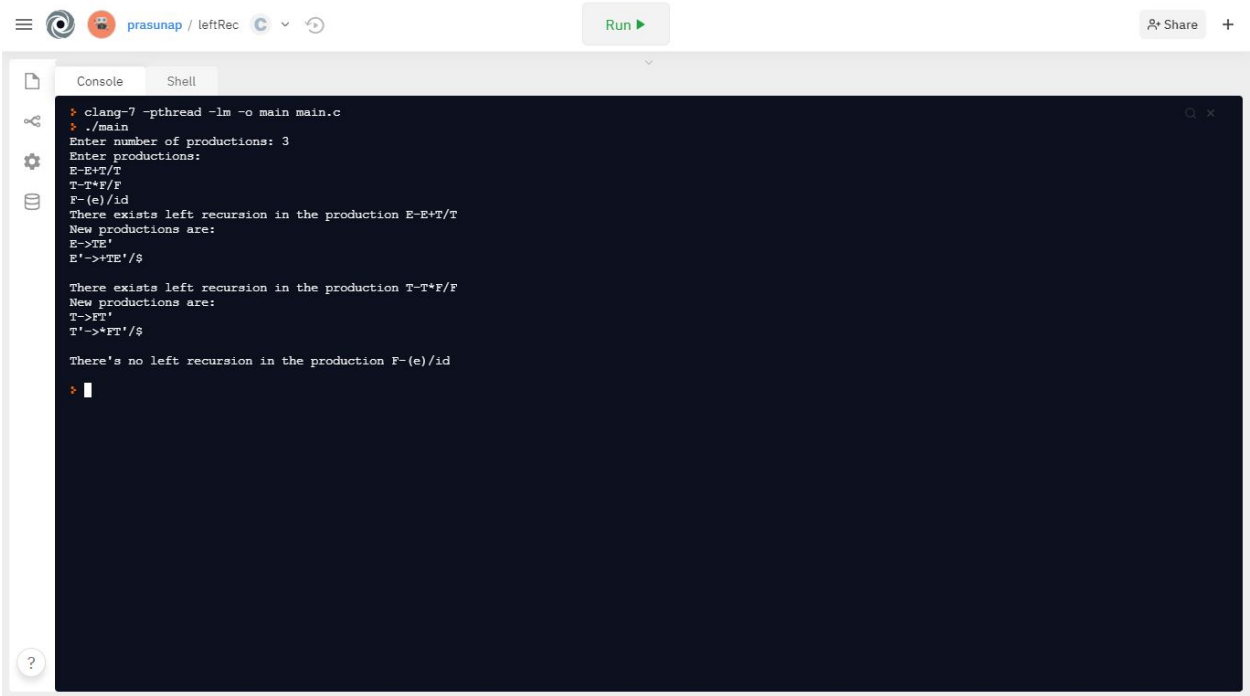
LeftRecursion.c

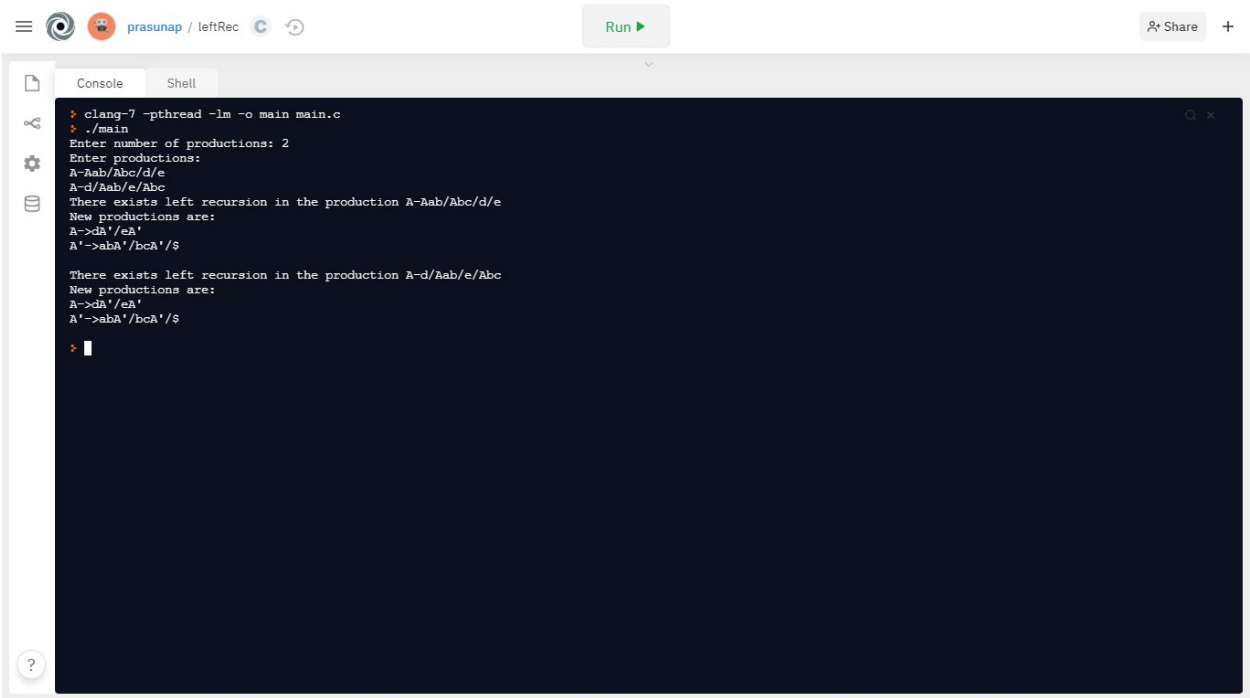
```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(void) {
 int productions;
 printf("Enter number of productions: ");
 scanf("%d", &productions);
 printf("Enter productions:\n");
 char prod[productions][100];
 for (int i = 0; i < productions; i++) {
   scanf("%s", prod[i]);
 }
 for (int i = 0; i < productions; i++) {
     int p1 = 0;
     char alpha[100][100];
     int l = 0, j = 2;
     while (prod[i][j] != '\0') {
       if (prod[i][j] != prod[i][0]) {
         while (prod[i][j] != '/' && prod[i][j] != '\0') j++;
         if (prod[i][j] != '\0') j++;
         continue;
       }
       while (prod[i][j] != '\0' && prod[i][j] == prod[i][0]) {
         j++;
         char a1[100] = { (char)NULL };
         while (prod[i][j] != '\0' && prod[i][j] != '/') {
           char s[2]; s[0] = prod[i][j]; s[1] = '\0';
           strcat(a1, s);
           j++;
         }
         strcpy(alpha[p1], a1);
         p1++;
         if (prod[i][j] == '/') j++;
       }
     }
```

```
    if(p1 == 0) {
      printf("There's no left recursion in the production %s\n", prod[i]);
      continue;
    }
    printf("There exists left recursion in the production %s\n", prod[i]);
    char beta[100][100];
    int p2 = 0;
    j = 2;
    while (prod[i][j] != '\0') {
      if (prod[i][j] == prod[i][0]) {
        while (prod[i][j] != '/' && prod[i][j] != '\0') j++;
        if (prod[i][j] != '\0') j++;
        continue;
      }
      char a2[100] = { (char)NULL };
      while (prod[i][j] != '\0' && prod[i][j] != '/') {
        char s[2]; s[0] = prod[i][j]; s[1] = '\0';
        strcat(a2, s);
        j++;
      }
      strcpy(beta[p2], a2);
      p2++;
      if (prod[i][j] == '/') j++;
    }
    printf("New productions are:\n%c->", prod[i][0]);
    for (int q1 = 0; q1 < p2; q1++) {
      if (q1 == p2 - 1)
        printf("%s%c'", beta[q1], prod[i][0]);
      else printf("%s%c'/", beta[q1], prod[i][0]);
    }
    printf("\n%c'->", prod[i][0]);
    for (int q1 = 0; q1 < p1; q1++) {
      printf("%s%c'/", alpha[q1], prod[i][0]);
    }
    printf("$\n\n");
  }
 return 0;
}
```

Works even if there are multiple alpha and betas.

Screenshot of output:





prasunap / leftRec

Run

Share

Console Shell

```
$ clang-7 -pthread -lm -o main main.c
$ ./main
Enter number of productions: 3
Enter productions:
E-T/Eg
E-Eg/T
E-TEg/Etg
There exists left recursion in the production E-T/Eg
New productions are:
E->TE'
E'->gE'/$

There exists left recursion in the production E-Eg/T
New productions are:
E->TE'
E'->gE'/$

There exists left recursion in the production E-TEg/Etg
New productions are:
E->TEgE'
E'->tgE'/$

$
```

LeftFactoring.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int min(int a, int b) {
  if (a < b) return a;
  return b;
}

char* commonPrefixes(char prods[100][100], int n, char prev[100][100], int
m) {
  char *res;
  res = (char*)malloc(100);
  int max = 0;
  for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++) {
      int k = 0, count = 0;
      k = min(strlen(prods[i]), strlen(prods[j]));
      for (int p1 = 0; p1 < k; p1++) {
        if (prods[i][p1] == prods[j][p1]) count++;
        else break;
      }
      char a1[100] = { (char)NULL };
      for (int p1 = 0; p1 < count; p1++) {
        a1[p1] = prods[i][p1];
      }
      a1[count] = '\0';
      int found = 0;
      for (int p1 = 0; p1 < m; p1++) {
        if (strcmp(a1, prev[p1]) == 0) {
          found = 1;
          break;
        }
      }
      if (max < count && found == 0) {
        max = count;
        for (int p1 = 0; p1 < count; p1++) {
          res[p1] = prods[i][p1];
        }
        res[count] = '\0';
      }
```

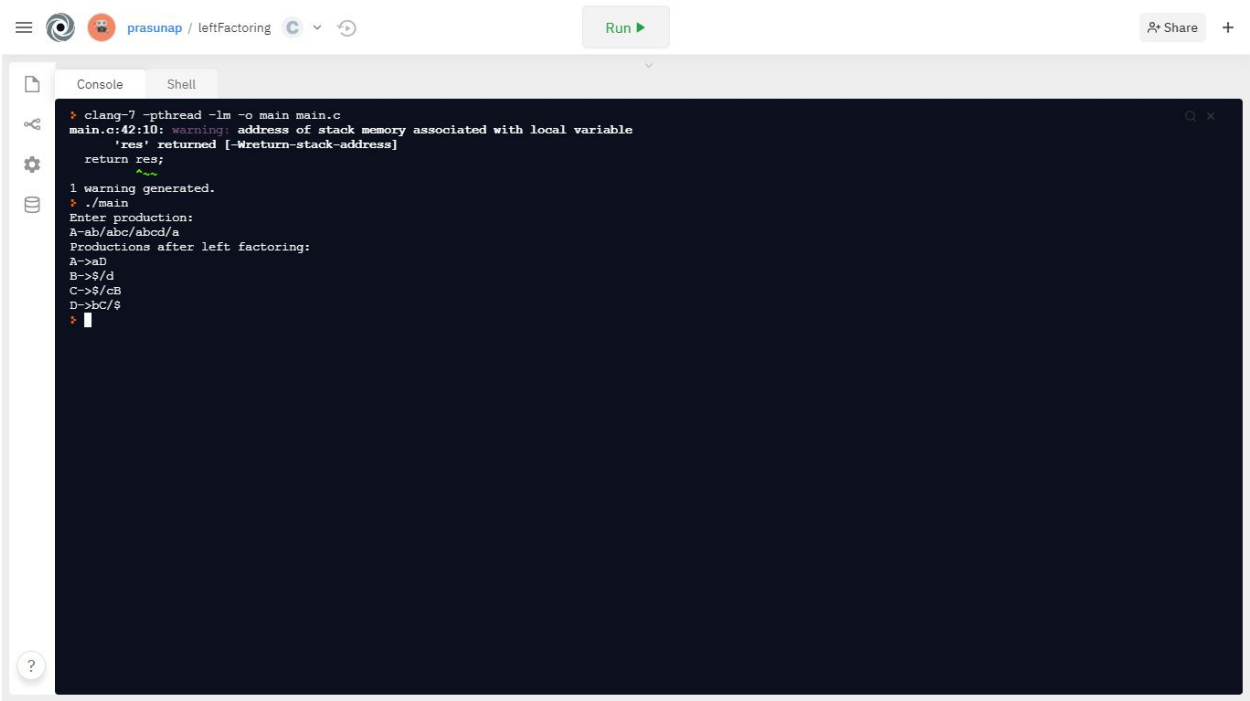
```
    }
  }
  return (char*)res;
}

int main() {
  char input[100];
  printf("Enter production:\n");
  scanf("%s", input);
  int z = 2, sz = 1, i = 0, res1 = 0, p1 = 0;
  char start = input[0];
  char prods[100][100];
  while (input[z] != '\0') {
    if (input[z] == '/') {
      sz++;
      i++; p1 = 0;
    } else {
      prods[i][p1] = input[z];
      p1++;
    }
    z++;
  }
  char ress[100][100];
  char prev[100][100];
  i = 0;
  char extra = 'A';
  while (1) {
    if (extra == start) extra++;
    char* res = commonPrefixes(prods, sz, prev, i);
    strcpy(prev[i], res);
    if (res[0] == '\0' || (i > 0 && strcmp(prev[i - 1], prev[i]) == 0)) {
      break;
    }
    char beta[100][100];
    int b = 0, kept = 0;
    for (int j = 0; j < sz; j++) {
      if (strlen(prods[j]) < strlen(res)) continue;
      char a1[100];
      for (int k = 0; k < strlen(res); k++) {
        a1[k] = prods[j][k];
      }
      a1[strlen(res)] = '\0';
      if (strcmp(a1, res) == 0) {
```

```
        int q1 = 0;
        for (int k = strlen(res); k < strlen(prods[j]); k++) {
          beta[b][q1] = prods[j][k];
          q1++;
        }
        beta[b][q1] = '\0';
        if (kept == 0) {
          // printf("yes");
          for (int k = 0; k < strlen(res); k++) {
            prods[j][k] = res[k];
          }
          prods[j][strlen(res)] = extra;
          prods[j][strlen(res) + 1] = '\0';
          kept = 1;
        } else {
          prods[j][0] = '\0';
        }
      }
      b++;
    }
    int p1 = 0;
    ress[res1][p1] = extra;
    ress[res1][++p1] = '-';
    ress[res1][++p1] = '>';
    for (int k = 0; k < b; k++) {
      if (beta[k][0] == '\0') {
        ress[res1][++p1] = '$';
      } else {
        for (int k1 = 0; k1 < strlen(beta[k]); k1++) {
          ress[res1][++p1] = beta[k][k1];
        }
      }
      if (k != b - 1) ress[res1][++p1] = '/';
    }
    res1++;
    i++;
    extra++;
  }
  printf("Productions after left factoring:\n");
  for (int i = 0; i < sz; i++) {
    if (strlen(prods[i]) > 0) {
      printf("%c->", start);
      printf("%s\n", prods[i]);
```

```
    }
  }
  for (int i = 0; i < res1; i++) {
    printf("%s\n", ress[i]);
  }
  return 0;
}
```

Works even if there are multiple common prefixes.

Screenshot of output:



prasunap / leftFactoring

Run

Share

Console Shell

```
> clang-7 -pthread -lm -o main main.c
main.c:42:10: warning: address of stack memory associated with local variable
      'res' returned [-Wreturn-stack-address]
  return res;
         ^~~
1 warning generated.
> ./main
Enter production:
A-ab/abc/abd
Productions after left factoring:
A->abB
B->$/c/d
>
```