COMPILER DESIGN LAB

LAB ASSESSMENT-6

NAME: KARAN SEHGAL
REGISTRATION NUMBER: 22BCE3939

Q1. YACC program to convert Infix expression to Postfix expression

Code:

Yacc file:

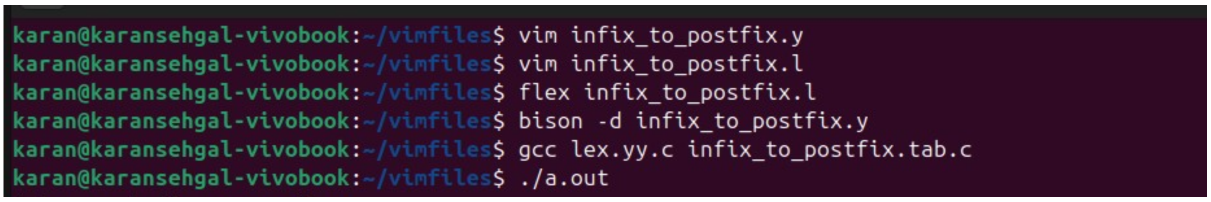
```
%{
#include <stdio.h>
#include <stdlib.h>

void yyerror(const char *s);
int yylex();
%}

%token NUMBER PLUS MINUS MULT DIV LPAREN RPAREN
%left PLUS MINUS
%left MULT DIV
%right UMINUS

%%

expression:
    expression PLUS expression    { printf("+ "); }
  | expression MINUS expression   { printf("- "); }
  | expression MULT expression    { printf("* "); }
  | expression DIV expression     { printf("/ "); }
  | LPAREN expression RPAREN      { /* do nothing */ }
  | MINUS expression %prec UMINUS { printf("-%d ", $2); }
  | NUMBER                        { printf("%d ", $1); }
  ;

%%

void yyerror(const char *s) {
    fprintf(stderr, "Error: %s\n", s);
}

int main() {
    printf("Enter an infix expression:\n");
    yyparse();
    printf("\n");
    return 0;
}
```

Lex file:

```
%{
#include "infix_to_postfix.tab.h"
%}

%%

"+"      return PLUS;
"-"      return MINUS;
"*"      return MULT;
"/"      return DIV;
"("      return LPAREN;
")"      return RPAREN;
[0-9]+   { yylval = atoi(yytext); return NUMBER; }
[ \t\n]  ;  // ignore whitespace
.        return yytext[0];

%%

int yywrap() {
   return 1;
}
```

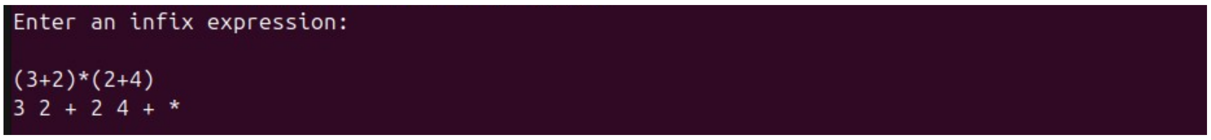
RUN COMMANDS:-

INPUT1



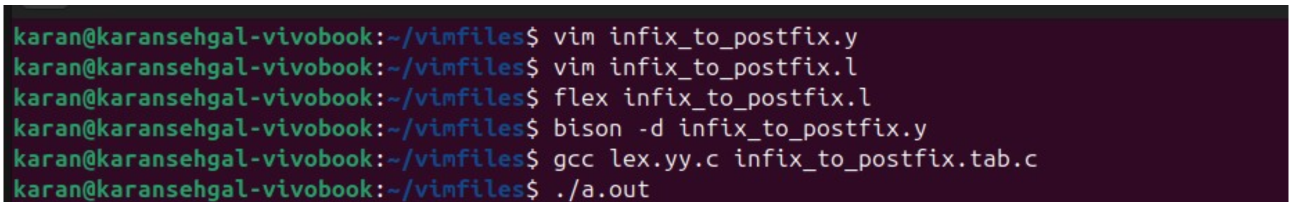
```
karan@karansehgal-vivobook:~/vimfiles$ vim infix_to_postfix.y
karan@karansehgal-vivobook:~/vimfiles$ vim infix_to_postfix.l
karan@karansehgal-vivobook:~/vimfiles$ flex infix_to_postfix.l
karan@karansehgal-vivobook:~/vimfiles$ bison -d infix_to_postfix.y
karan@karansehgal-vivobook:~/vimfiles$ gcc lex.yy.c infix_to_postfix.tab.c
karan@karansehgal-vivobook:~/vimfiles$ ./a.out
```

OUTPUT 1



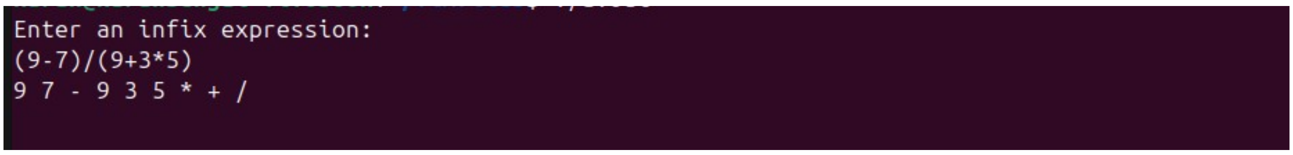
```
Enter an infix expression:

(3+2)*(2+4)
3 2 + 2 4 + *
```

INPUT 2



```
karan@karansehgal-vivobook:~/vimfiles$ vim infix_to_postfix.y
karan@karansehgal-vivobook:~/vimfiles$ vim infix_to_postfix.l
karan@karansehgal-vivobook:~/vimfiles$ flex infix_to_postfix.l
karan@karansehgal-vivobook:~/vimfiles$ bison -d infix_to_postfix.y
karan@karansehgal-vivobook:~/vimfiles$ gcc lex.yy.c infix_to_postfix.tab.c
karan@karansehgal-vivobook:~/vimfiles$ ./a.out
```

OUTPUT 2



```
Enter an infix expression:
(9-7)/(9+3*5)
9 7 - 9 3 5 * + /
```

Q2. YACC program to generate 3-Address code for a given expression:

Code:

YACC File:

```
%{
#include <stdio.h>
#include <stdlib.h>

int temp_count = 0;

void yyerror(const char *s);
int yylex();

char *new_temp() {
   char *temp = (char *)malloc(8);
   sprintf(temp, "t%d", temp_count++);
   return temp;
}
%}

%token <val> NUMBER
%token PLUS MINUS MULT DIV LPAREN RPAREN
%left PLUS MINUS
%left MULT DIV
%right UMINUS

%union {
   int val;
   char *tac;
}

%type <tac> expression

%%

expression:
   expression PLUS expression   {
      char *temp = new_temp();
      printf("%s = %s + %s\n", temp, $1, $3);
      $$ = temp;
   }
  | expression MINUS expression  {
      char *temp = new_temp();
      printf("%s = %s - %s\n", temp, $1, $3);
      $$ = temp;
   }
```

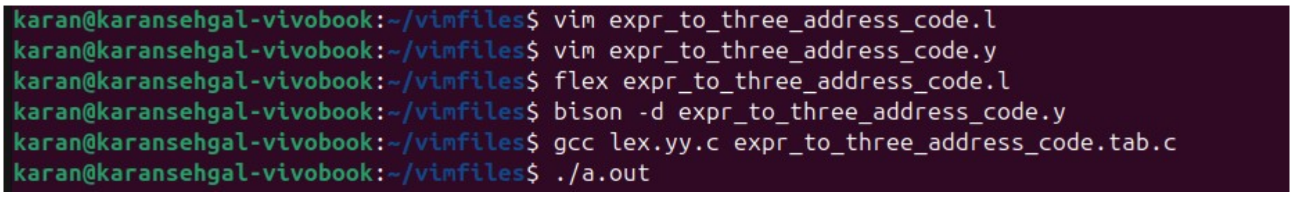
```
 | expression MULT expression   {
      char *temp = new_temp();
      printf("%s = %s * %s\n", temp, $1, $3);
      $$ = temp;
   }
 | expression DIV expression    {
      char *temp = new_temp();
      printf("%s = %s / %s\n", temp, $1, $3);
      $$ = temp;
   }
 | LPAREN expression RPAREN     { $$ = $2; }
 | MINUS expression %prec UMINUS {
      char *temp = new_temp();
      printf("%s = -%s\n", temp, $2);
      $$ = temp;
   }
 | NUMBER                       {
      char *temp = new_temp();
      printf("%s = %d\n", temp, $1);
      $$ = temp;
   }
 ;

%%

void yyerror(const char *s) {
   fprintf(stderr, "Error: %s\n", s);
}

int main() {
   printf("Enter an arithmetic expression:\n");
   yyparse();
   return 0;
}
```

Lex file:

```
%{
#include "expr_to_tac.tab.h"
%}

%%

"+"      return PLUS;
"-"      return MINUS;
"*"      return MULT;
"/"      return DIV;
"("      return LPAREN;
")"      return RPAREN;
[0-9]+   { yylval.val = atoi(yytext); return NUMBER; }
[ \t\n]  ;
.        return yytext[0];

%%

int yywrap() {
   return 1;
}
```

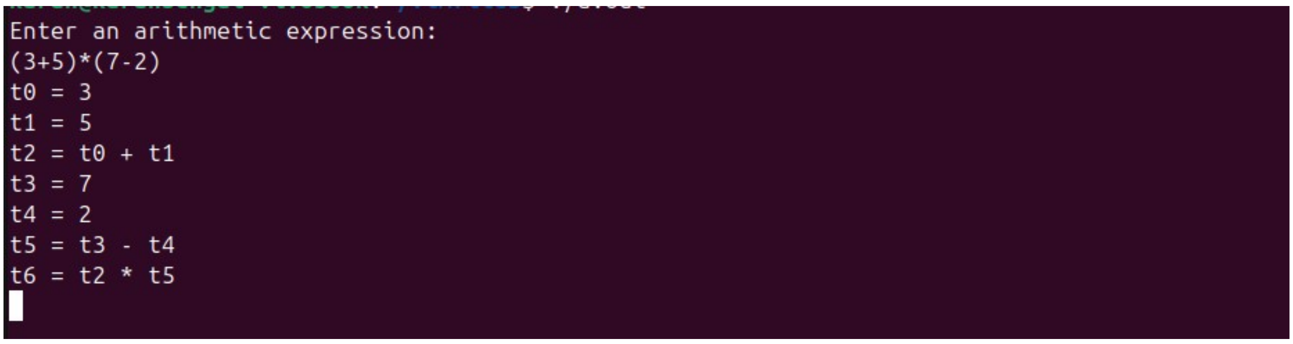
RUN COMMANDS:-

INPUT 1



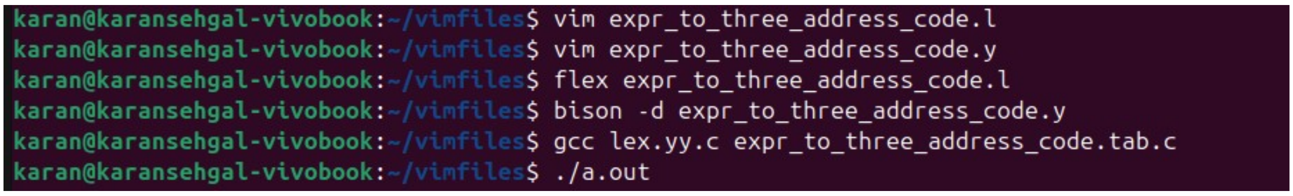
```
karan@karansehgal-vivobook:~/vimfiles$ vim expr_to_three_address_code.l
karan@karansehgal-vivobook:~/vimfiles$ vim expr_to_three_address_code.y
karan@karansehgal-vivobook:~/vimfiles$ flex expr_to_three_address_code.l
karan@karansehgal-vivobook:~/vimfiles$ bison -d expr_to_three_address_code.y
karan@karansehgal-vivobook:~/vimfiles$ gcc lex.yy.c expr_to_three_address_code.tab.c
karan@karansehgal-vivobook:~/vimfiles$ ./a.out
```

OUTPUT 1



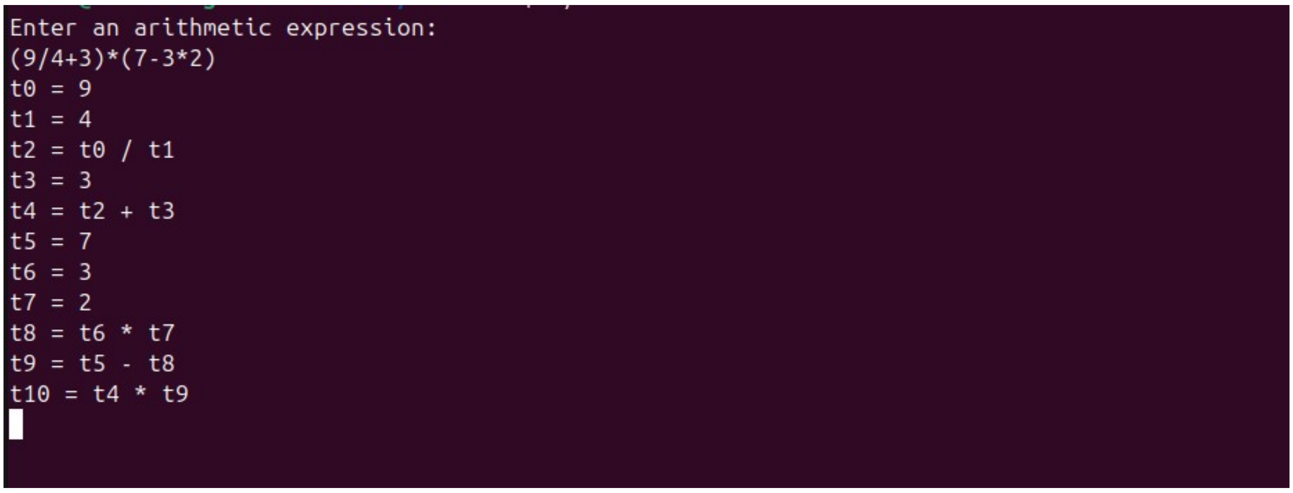
```
Enter an arithmetic expression:
(3+5)*(7-2)
t0 = 3
t1 = 5
t2 = t0 + t1
t3 = 7
t4 = 2
t5 = t3 - t4
t6 = t2 * t5
```

INPUT 2



```
karan@karansehgal-vivobook:~/vimfiles$ vim expr_to_three_address_code.l
karan@karansehgal-vivobook:~/vimfiles$ vim expr_to_three_address_code.y
karan@karansehgal-vivobook:~/vimfiles$ flex expr_to_three_address_code.l
karan@karansehgal-vivobook:~/vimfiles$ bison -d expr_to_three_address_code.y
karan@karansehgal-vivobook:~/vimfiles$ gcc lex.yy.c expr_to_three_address_code.tab.c
karan@karansehgal-vivobook:~/vimfiles$ ./a.out
```

OUTPUT 2



```
Enter an arithmetic expression:
(9/4+3)*(7-3*2)
t0 = 9
t1 = 4
t2 = t0 / t1
t3 = 3
t4 = t2 + t3
t5 = 7
t6 = 3
t7 = 2
t8 = t6 * t7
t9 = t5 - t8
t10 = t4 * t9
```

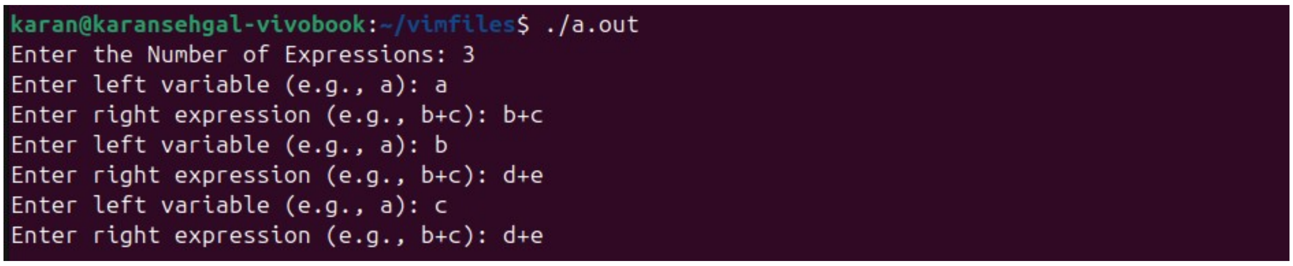
Q3. C Program for implementation of Code Optimization Technique

Code:

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

struct op {
   char l;
   char r[10];
} op[10], pr[10];

// Function to check if the left side of an expression is used anywhere
int isUsed(char var, struct op op[], int n) {
   for (int i = 0; i < n; i++) {
      if (strchr(op[i].r, var)) {
         return 1;  // If found, return 1 (used)
      }
   }
   return 0;  // Not used
}

// Function to eliminate dead code
void deadCodeElimination(struct op op[], int *n) {
   int used[10] = {0};  // Array to mark used expressions

   // Mark expressions that are used
   for (int i = 0; i < *n; i++) {
      if (isUsed(op[i].l, op, *n)) {
         used[i] = 1;
      }
   }
   // Eliminate dead code
   int newIdx = 0;
   for (int i = 0; i < *n; i++) {
      if (used[i]) {
         op[newIdx++] = op[i];
      }
   }
 *n = newIdx;  // Update number of expressions
}

// Function to eliminate common subexpressions
void commonExpressionElimination(struct op op[], int *n) {
   for (int i = 0; i < *n; i++) {
      for (int j = i + 1; j < *n; j++) {
         // Check for common subexpression
```

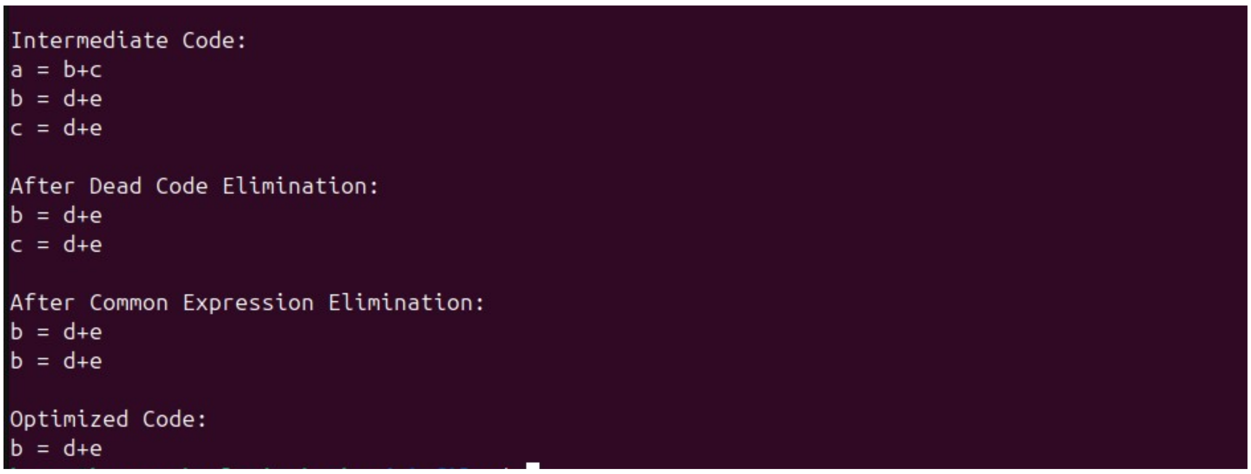
```
            if (strcmp(op[i].r, op[j].r) == 0) {
                char common = op[j].l;
                op[j].l = op[i].l;  // Replace with the same variable
                for (int k = 0; k < *n; k++) {
                    // Replace common expression in other places
                    char *p = strchr(op[k].r, common);
                    if (p) {
                        *p = op[i].l;  // Replace with the optimized variable
                    }
                }
            }
        }
    }
}

// Function to print the expressions
void printCode(struct op op[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%c = %s\n", op[i].l, op[i].r);
    }
}

// Function to print the optimized code (removes duplicates)
void printOptimizedCode(struct op op[], int n) {
    int printed[10] = {0};  // Array to track printed expressions

    for (int i = 0; i < n; i++) {
        // Check if this expression has already been printed
        int isDuplicate = 0;
        for (int j = 0; j < i; j++) {
            if (op[i].l == op[j].l && strcmp(op[i].r, op[j].r) == 0) {
                isDuplicate = 1;
                break;
            }
        }

        // Print the expression only if it's not a duplicate
        if (!isDuplicate) {
            printf("%c = %s\n", op[i].l, op[i].r);
        }
    }
}

int main() {
    int n;

    printf("Enter the Number of Expressions: ");
    scanf("%d", &n);

    // Input the expressions
    for (int i = 0; i < n; i++) {
        printf("Enter left variable (e.g., a): ");
        scanf(" %c", &op[i].l);
        printf("Enter right expression (e.g., b+c): ");
        scanf(" %s", op[i].r);
    }
    printf("\nIntermediate Code:\n");
    printCode(op, n);
    // Apply Dead Code Elimination
    deadCodeElimination(op, &n);
    printf("\nAfter Dead Code Elimination:\n");
    printCode(op, n);

    // Apply Common Expression Elimination
    commonExpressionElimination(op, &n);
    printf("\nAfter Common Expression Elimination:\n");
    printCode(op, n);

    // Print the Optimized Code (removes duplicates)
    printf("\nOptimized Code:\n");
    printOptimizedCode(op, n);

    return 0;
}
```
RUN COMMANDS:-

INPUT 1



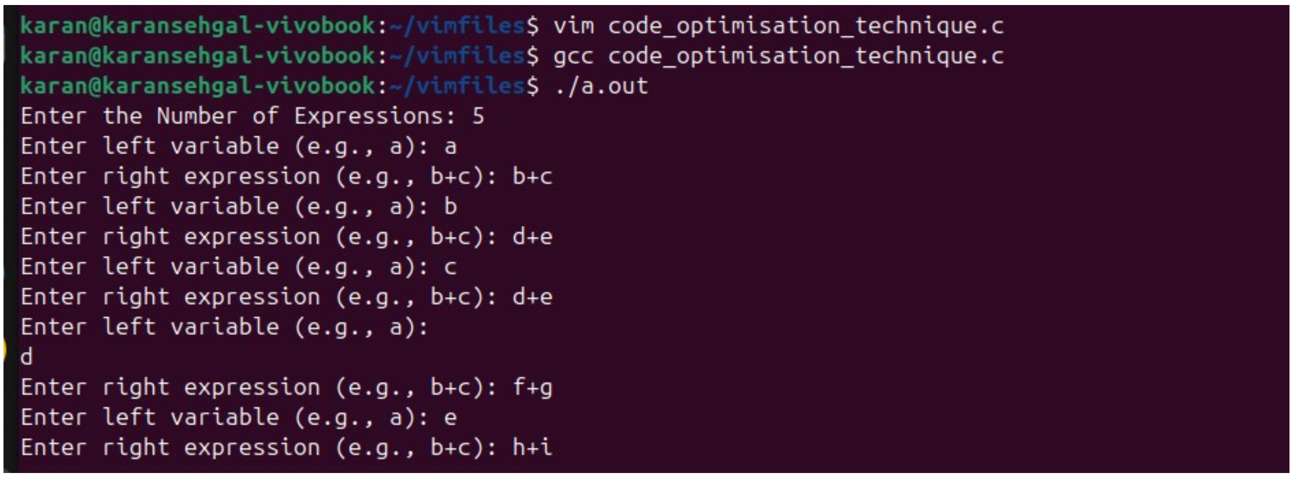
```
karan@karansehgal-vivobook:~/vimfiles$ ./a.out
Enter the Number of Expressions: 3
Enter left variable (e.g., a): a
Enter right expression (e.g., b+c): b+c
Enter left variable (e.g., a): b
Enter right expression (e.g., b+c): d+e
Enter left variable (e.g., a): c
Enter right expression (e.g., b+c): d+e
```

OUTPUT 1



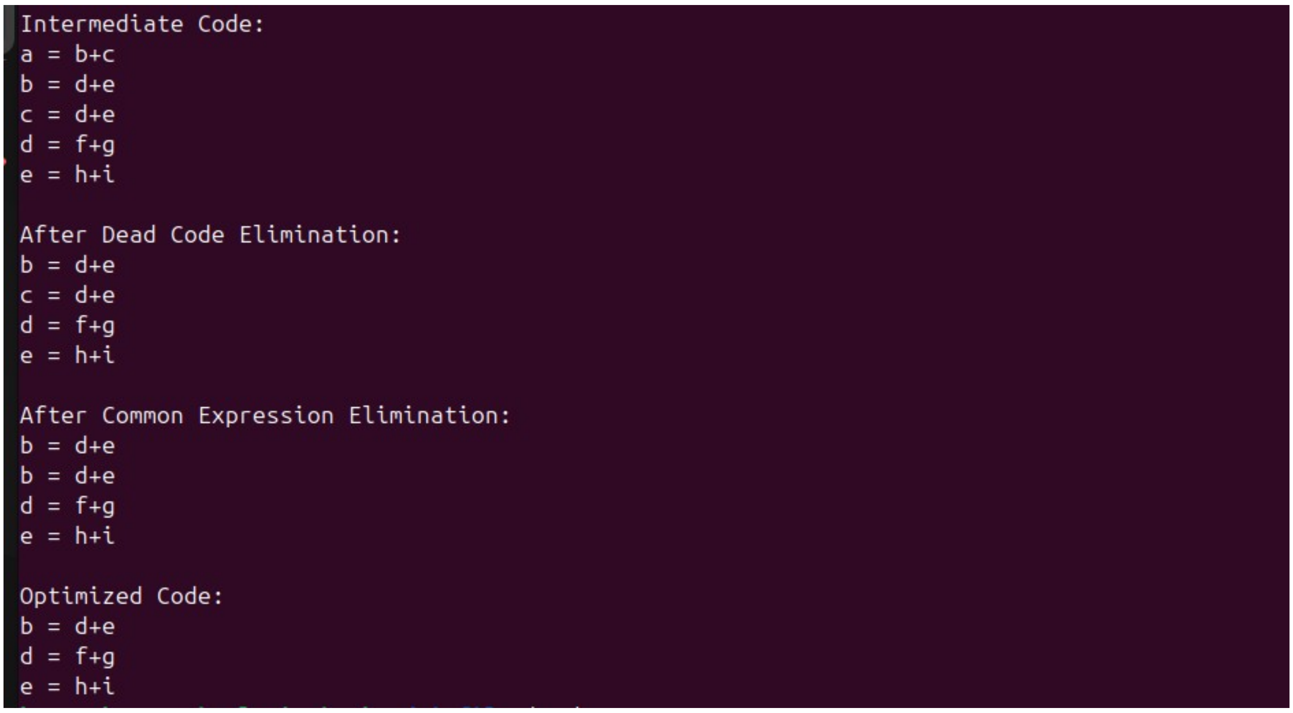
```
Intermediate Code:
a = b+c
b = d+e
c = d+e

After Dead Code Elimination:
b = d+e
c = d+e

After Common Expression Elimination:
b = d+e
b = d+e

Optimized Code:
b = d+e
```

INPUT 2



```
karan@karansehgal-vivobook:~/vimfiles$ vim code_optimisation_technique.c
karan@karansehgal-vivobook:~/vimfiles$ gcc code_optimisation_technique.c
karan@karansehgal-vivobook:~/vimfiles$ ./a.out
Enter the Number of Expressions: 5
Enter left variable (e.g., a): a
Enter right expression (e.g., b+c): b+c
Enter left variable (e.g., a): b
Enter right expression (e.g., b+c): d+e
Enter left variable (e.g., a): c
Enter right expression (e.g., b+c): d+e
Enter left variable (e.g., a):
d
Enter right expression (e.g., b+c): f+g
Enter left variable (e.g., a): e
Enter right expression (e.g., b+c): h+i
```

OUTPUT 2



```
Intermediate Code:
a = b+c
b = d+e
c = d+e
d = f+g
e = h+i

After Dead Code Elimination:
b = d+e
c = d+e
d = f+g
e = h+i

After Common Expression Elimination:
b = d+e
b = d+e
d = f+g
e = h+i

Optimized Code:
b = d+e
d = f+g
e = h+i
```