Experiment 2

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define STACKSIZE 99

struct Stack {
   int top;
   int items[STACKSIZE];
};

int isoperand(char c) {
   if (c >= 'A' && c <= 'Z' ||
      c >= 'a' && c <= 'z' ||
      c >= '0' && c <= '9')
      return 1;
   else
      return 0;
}

int empty(struct Stack *ps) {
   if (ps->top == -1)
      return 1;
   else
      return 0;
}

char pop(struct Stack *ps) {
   return (ps->items[ps->top--]);
}

char stacktop(struct Stack *ps) {
   return (ps->items[ps->top]);
}

void push(struct Stack *ps, char x) {
```

```
    if (ps->top == STACKSIZE - 1) {
        printf("overflow");
        exit(1);
    }
    ps->items[(++ps->top)] = x;
}

int prcd(char st, char cs) {
    switch (st) {
        case '+':
        case '-':
            if (cs == '+' || cs == '-' || cs == ')') return 1;
            if (cs == '*' || cs == '/' || cs == '$' || cs == '(') return 0;

        case '*':
        case '/':
            if (cs == '$' || cs == '(') return 0;
            if (cs == '*' || cs == '/' || cs == '+' || cs == ')' || cs == '-') return 1;

        case '$':
            if (cs == '$' || cs == '(') return 0;
            if (cs == '*' || cs == '/' || cs == '+' || cs == ')' || cs == '-') return 1;

        case '(':
            return 0;
    }
    return -1;
}

void postfix(char ie[], char oe[]) {
    int i, j;
    char ts, cs;
    struct Stack s;
    s.top = -1;

    for (i = 0, j = 0; (cs = ie[i]) != '\0'; i++) {
        if (isoperand(cs))
            oe[j++] = cs;
```

```
        else {
            while (!empty(&s) && prcd(stacktop(&s), cs)) {
                ts = pop(&s);
                oe[j++] = ts;
            }

            if (empty(&s) || cs != ')')
                push(&s, cs);
            else
                ts = pop(&s);
        }
    }

    while (!empty(&s))
        oe[j++] = pop(&s);
    oe[j] = '\0';
}

int main() {
    char ie[50], oe[50];
    printf("Enter Infix Expression: \n");
    fgets(ie, sizeof(ie), stdin);
    ie[strcspn(ie, "\n")] = '\0';
    postfix(ie, oe);
    printf("Postfix expression: \n");
    puts(oe);
    return 0;
}
```

Output:

Enter Infix Expression:
A+B*C+D
Postfix expression:
ABC*+D+

Enter Infix Expression:
(A+B)*(C+D)
Postfix expression:
AB+CD+*

Enter Infix Expression:
A*B+C*D
Postfix expression:
AB*CD*+

Enter Infix Expression:
(a+b)*c/(e-f-g)*d$a
Postfix expression:
ab+c*ef-g-/da$*