QUESTION 1

```
#include <stdio.h>
#define MAX_SIZE 100

typedef struct {
    int arr[MAX_SIZE];
    int top;
} Stack;

void initialize(Stack *s) {
    s->top = -1;
}

int isEmpty(Stack *s) {
    return (s->top == -1);
}

int isFull(Stack *s) {
    return (s->top == MAX_SIZE - 1);
}

void push(Stack *s, int item) {
    if (isFull(s)) {
        printf("Stack Overflow!\n");
        return;
    }
    s->top++;
    s->arr[s->top] = item;
```

```
}

int pop(Stack *s) {
    if (isEmpty(s)) {
        printf("Stack Underflow!\n");
        return -1;
    }
    int item = s->arr[s->top];
    s->top--;
    return item;
}

int peek(Stack *s) {
    if (isEmpty(s)) {
        printf("Stack is empty!\n");
        return -1;
    }
    return s->arr[s->top];
}

int main() {
    Stack s;
    initialize(&s);

    push(&s, 7);
    push(&s, 29);
    push(&s, 67);

    printf("Top element of the stack: %d\n", peek(&s));
    printf("Popped element: %d\n", pop(&s));
    printf("Top element of the stack: %d\n", peek(&s));
```

```
    printf("Popped element: %d\n", pop(&s));
    printf("Top element of the stack: %d\n", peek(&s));

    return 0;
}
```

main.c

Save Run

```
int peek(Stack *s) {
    if (isEmpty(s)) {
        printf("Stack is empty!\n");
        return -1;
    }
    return s->arr[s->top];
}

int main() {
    Stack s;
    initialize(&s);

    push(&s, 7);
    push(&s, 29);
    push(&s, 67);

    printf("Top element of the stack: %d\n", peek(&s));
    printf("Popped element: %d\n", pop(&s));
    printf("Top element of the stack: %d\n", peek(&s));
    printf("Popped element: %d\n", pop(&s));
    printf("Top element of the stack: %d\n", peek(&s));

    return 0;
}
```

Output

```
/tmp/Kd8S8wlC6r.o
Top element of the stack: 67
Popped element: 67
Top element of the stack: 29
Popped element: 29
Top element of the stack: 7
```

QUESTION 2

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

#define MAX_SIZE 100

struct Stack {
    int top;
    unsigned capacity;
    char *array;
};

struct Stack *createStack(unsigned capacity) {
    struct Stack *stack = (struct Stack *)malloc(sizeof(struct Stack));
    if (!stack) return NULL;
    stack->top = -1;
    stack->capacity = capacity;
    stack->array = (char *)malloc(stack->capacity * sizeof(char));
    if (!stack->array) return NULL;
    return stack;
}

int isEmpty(struct Stack *stack) {
    return stack->top == -1;
}

void push(struct Stack *stack, char item) {
    stack->array[++stack->top] = item;
```

```
}

char pop(struct Stack *stack) {
   if (!isEmpty(stack))
      return stack->array[stack->top--];
   return '$';
}

int precedence(char op) {
   if (op == '+' || op == '-')
      return 1;
   if (op == '*' || op == '/')
      return 2;
   return 0;
}

void infixToPostfix(char *infix, char *postfix) {
   struct Stack *stack = createStack(strlen(infix));
   int i, k;
   for (i = 0, k = -1; infix[i]; ++i) {

      if (isalnum(infix[i]))
         postfix[++k] = infix[i];

      else if (infix[i] == '(')
         push(stack, infix[i]);

      else if (infix[i] == ')') {
         while (!isEmpty(stack) && stack->array[stack->top] != '('
            postfix[++k] = pop(stack);
         if (!isEmpty(stack) && stack->array[stack->top] != '('
```

```
            return;
        else
            pop(stack);
    }

    else {
        while (!isEmpty(stack) && precedence(infix[i]) <= precedence(stack->array[stack->top]))
            postfix[++k] = pop(stack);
        push(stack, infix[i]);
    }
  }

  while (!isEmpty(stack))
    postfix[++k] = pop(stack);
  postfix[++k] = '\0';
}

int main() {
  char infix[MAX_SIZE];
  char postfix[MAX_SIZE];

  printf("Enter the infix expression: ");
  fgets(infix, MAX_SIZE, stdin);
  infix[strcspn(infix, "\n")] = 0;

  infixToPostfix(infix, postfix);
  printf("Postfix expression: %s\n", postfix);

  return 0;
}
```

main.c

Save Run

```
                precedence(stack->array[stack->top]))
67              postfix[++k] = pop(stack);
68          push(stack, infix[i]);
69      }
70  }
71
72  while (!isEmpty(stack))
73      postfix[++k] = pop(stack);
74  postfix[++k] = '\0';
75 }
76
77 int main() {
78      char infix[MAX_SIZE];
79      char postfix[MAX_SIZE];
80
81      printf("Enter the infix expression: ");
82      fgets(infix, MAX_SIZE, stdin);
83      infix[strcspn(infix, "\n")] = 0;
84
85      infixToPostfix(infix, postfix);
86      printf("Postfix expression: %s\n", postfix);
87
88      return 0;
89 }
```

Output

```
/tmp/Kd8S8wlC6r.o
Enter the infix expression: 5+7*9-21/1+6%5
Postfix expression: 579*+211/-6+5%
```

QUESTION 3

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

#define MAX_SIZE 100

typedef struct {
    int top;
    int items[MAX_SIZE];
} Stack;

void push(Stack *s, int value);
int pop(Stack *s);
int evaluatePostfix(char *exp);

int main() {
    char exp[MAX_SIZE];
    printf("Enter the postfix expression: ");
    scanf("%s", exp);
    int result = evaluatePostfix(exp);
    printf("Result: %d\n", result);
    return 0;
}

void push(Stack *s, int value) {
    if (s->top == MAX_SIZE - 1) {
        printf("Stack Overflow\n");
        exit(EXIT_FAILURE);
```

```
    }
    s->items[++(s->top)] = value;
}

int pop(Stack *s) {
    if (s->top == -1) {
        printf("Stack Underflow\n");
        exit(EXIT_FAILURE);
    }
    return s->items[(s->top)--];
}

int evaluatePostfix(char *exp) {
    Stack s;
    s.top = -1;
    int i, op1, op2, result;
    for (i = 0; exp[i] != '\0'; i++) {
        if (isdigit(exp[i])) {
            push(&s, exp[i] - '0');
        } else {
            op2 = pop(&s);
            op1 = pop(&s);
            switch (exp[i]) {
                case '+':
                    push(&s, op1 + op2);
                    break;
                case '-':
                    push(&s, op1 - op2);
                    break;
                case '*':
                    push(&s, op1 * op2);
```

```
            break;
        case '/':
            push(&s, op1 / op2);
            break;
        default:
            printf("Invalid operator\n");
            exit(EXIT_FAILURE);
      }
    }
  }
  result = pop(&s);
  return result;
}
```

```
15
16  int main() {
17      char exp[MAX_SIZE];
18      printf("Enter the postfix expression: ");
19      scanf("%s", exp);
20      int result = evaluatePostfix(exp);
21      printf("Result: %d\n", result);
22      return 0;
23  }
24
25  void push(Stack *s, int value) {
26      if (s->top == MAX_SIZE - 1) {
27          printf("Stack Overflow\n");
28          exit(EXIT_FAILURE);
29      }
30      s->items[++(s->top)] = value;
31  }
32
33  int pop(Stack *s) {
34      if (s->top == -1) {
35          printf("Stack Underflow\n");
36          exit(EXIT_FAILURE);
37      }
38      return s->items[(s->top)--];
```

```
/tmp/Kd8S8wlC6r.o
Enter the postfix expression: 57*
Result: 35
```

QUESTION 4

```
#include <stdio.h>

void move(int n, int source, int destination, int intermediate) {
    if (n == 1) {
        printf("Move disk 1 from rod %d to rod %d\n", source, destination);
        return;
    }
    move(n - 1, source, intermediate, destination);
    printf("Move disk %d from rod %d to rod %d\n", n, source, destination);
    move(n - 1, intermediate, destination, source);
}

int main() {
    int num_disks = 4;
    move(num_disks, 1, 3, 2);
    return 0;
}
```

```
#include <stdio.h>

void move(int n, int source, int destination, int
    intermediate) {
    if (n == 1) {
        printf("Move disk 1 from rod %d to rod %d\n", source,
            destination);
        return;
    }
    move(n - 1, source, intermediate, destination);
    printf("Move disk %d from rod %d to rod %d\n", n, source,
        destination);
    move(n - 1, intermediate, destination, source);
}

int main() {
    int num_disks = 4;
    move(num_disks, 1, 3, 2);
    return 0;
}
```

```
/tmp/Kd8S8wlC6r.o
Move disk 1 from rod 1 to rod 2
Move disk 2 from rod 1 to rod 3
Move disk 1 from rod 2 to rod 3
Move disk 3 from rod 1 to rod 2
Move disk 1 from rod 3 to rod 1
Move disk 2 from rod 3 to rod 2
Move disk 1 from rod 1 to rod 2
Move disk 4 from rod 1 to rod 3
Move disk 1 from rod 2 to rod 3
Move disk 2 from rod 2 to rod 1
Move disk 1 from rod 3 to rod 1
Move disk 3 from rod 2 to rod 3
Move disk 1 from rod 1 to rod 2
Move disk 2 from rod 1 to rod 3
Move disk 1 from rod 2 to rod 3
```