1)write a c programming code implementation infix prefix and postfix rotation for arithmetic expression using stack.

code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
typedef struct {
   int top;
   unsigned capacity;
   char* array;
} Stack;
Stack* createStack(unsigned capacity) {
   Stack* stack = (Stack*)malloc(sizeof(Stack));
   stack->capacity = capacity;
   stack->top = -1;
   stack->array = (char*)malloc(stack->capacity * sizeof(char));
   return stack;
}
int isFull(Stack* stack) {
   return stack->top == stack->capacity - 1;
}
int isEmpty(Stack* stack) {
   return stack->top == -1;
}
void push(Stack* stack, char item) {
   if (isFull(stack))
      return;
   stack->array[++stack->top] = item;
}
char pop(Stack* stack) {
   if (isEmpty(stack))
      return '$';
   return stack->array[stack->top--];
}
char peek(Stack* stack) {
   if (!isEmpty(stack))
      return stack->array[stack->top];
   return '$';
}
```

```
int precedence(char op) {
   switch (op) {
      case '+':
      case '-':
         return 1;
      case '*':
      case '/':
         return 2;
      case '^':
         return 3;
   }
   return 0;
}
void reverse(char* str) {
   int length = strlen(str);
   for (int i = 0; i < length / 2; i++) {
      char temp = str[i];
      str[i] = str[length - i - 1];
      str[length - i - 1] = temp;
   }
}
void infixToPostfix(char* expression, char* result) {
   Stack* stack = createStack(strlen(expression));
   if (!stack) {
      printf("Memory allocation failed\n");
      exit(EXIT_FAILURE);
   }
   int j = 0;
   for (int i = 0; expression[i]; i++) {
      if (expression[i] == ' ')
         continue;
      if (isalnum(expression[i])) {
         result[j++] = expression[i];
      } else if (expression[i] == '(') {
         push(stack, expression[i]);
      } else if (expression[i] == ')') {
         while (!isEmpty(stack) && peek(stack) != '(') {
            result[j++] = pop(stack);
         }
         pop(stack); // Remove '(' from stack
      } else { // An operator is encountered
         while (!isEmpty(stack) && precedence(peek(stack)) >=
precedence(expression[i])) {
            result[j++] = pop(stack);
            }
            push(stack, expression[i]);
        }
    }
    while (!isEmpty(stack)) {
        result[j++] = pop(stack);
    }
    result[j] = '\0';

    free(stack->array);
    free(stack);
}
void infixToPrefix(char* expression, char* result) {
    reverse(expression);
    for (int i = 0; i < strlen(expression); i++) {
        if (expression[i] == '(') {
            expression[i] = ')';
        } else if (expression[i] == ')') {
            expression[i] = '(';
        }
    }
    char postfix[100];
    infixToPostfix(expression, postfix);
    reverse(postfix);
    strcpy(result, postfix);
}

int main() {
    char infix[100];
    char postfix[100];
    char prefix[100];
    printf("Enter infix expression: ");
    fgets(infix, sizeof(infix), stdin);
    size_t length = strlen(infix);
    if (length > 0 && infix[length - 1] == '\n') {
        infix[length - 1] = '\0';
    }
    infixToPostfix(infix, postfix);
    printf("Postfix expression: %s\n", postfix);
    infixToPrefix(infix, prefix);
    printf("Prefix expression: %s\n", prefix);

    return 0;
}
```

Output:
```
Enter infix expression: 28
Postfix expression: 28
Prefix expression: 28
```

2)write a c programming implementation queue 1)its operations 2)array3)linked list(pointers)

code

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
typedef struct Node {
    int data;
    struct Node* next;
} Node;
typedef struct {
    Node* front;
    Node* rear;
} Queue;
void initializeQueue(Queue* q) {
    q->front = NULL;
    q->rear = NULL;
}
bool isEmpty(Queue* q) {
    return (q->front == NULL);
}
void enqueue(Queue* q, int item) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = item;
    newNode->next = NULL;
    if (isEmpty(q)) {
        q->front = newNode;
        q->rear = newNode;
    } else {
        q->rear->next = newNode;
        q->rear = newNode;
    }
}
int dequeue(Queue* q) {
    if (isEmpty(q)) {
```

```c
        printf("Queue is empty\n");
        return -1;
    }
    Node* temp = q->front;
    int item = temp->data;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }
    free(temp);
    return item;
}
int front(Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return -1; // Return -1 to indicate an error
    }
    return q->front->data;
}
void displayQueue(Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return;
    }
    Node* temp = q->front;
    printf("Queue elements: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    Queue q;
    initializeQueue(&q);

    enqueue(&q, 10);
    enqueue(&q, 20);
    enqueue(&q, 30);

    displayQueue(&q);

    printf("Dequeued: %d\n", dequeue(&q));
    displayQueue(&q);

    return 0;
}
```

Output:

```
Queue elements: 10 20 30
Dequeued: 10
Queue elements: 20 30
```

3)write a c programming code for enqueue and dequeue.

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX 100
typedef struct {
    int front, rear, size;
    int array[MAX];
} Queue;
void initializeQueue(Queue* q) {
    q->front = 0;
    q->rear = -1;
    q->size = 0;
}
bool isEmpty(Queue* q) {
    return (q->size == 0);
}
bool isFull(Queue* q) {
    return (q->size == MAX);
}
void enqueue(Queue* q, int item) {
    if (isFull(q)) {
        printf("Queue is full\n");
        return;
    }
    q->rear = (q->rear + 1) % MAX;
    q->array[q->rear] = item;
    q->size++;
}
int dequeue(Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return -1;
    }
    int item = q->array[q->front];
    q->front = (q->front + 1) % MAX;
    q->size--;
    return item;
}
void displayQueue(Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements: ");
    for (int i = 0; i < q->size; i++) {
        printf("%d ", q->array[(q->front + i) % MAX]);
    }
    printf("\n");
}

int main() {
    Queue q;
    initializeQueue(&q);

    enqueue(&q, 10);
    enqueue(&q, 20);
    enqueue(&q, 30);

    displayQueue(&q);

    printf("Dequeued: %d\n", dequeue(&q));
    displayQueue(&q);

    return 0;
}
```

Output:

```
Queue elements: 10 20 30
Dequeued: 10
Queue elements: 20 30
```