```c
#include<stdio.h>
#include<time.h>
#define size 5

void push(int st[], int *top){
    int x;

    if(*top ≥ size - 1){
        printf("ERROR: Stack overflow.\n");
        return ;
    }
    else{
    printf("Enter the value: ");
    scanf("%d", &x);
        (*top)++;
        st[*top] = x;
    }
}

void pop(int st[], int *top){
    if(*top < 0){
        printf("ERROR: Stack Underflow.\n");

    }
    else{
        (*top)--;
    }
}

void peep(int st[], int *top){
    printf("%d", st[*top]);
}

void change(int st[], int *top){

    int temp[100], changeIdx, i=0, val;

    label:
    printf("Enter the index at which you want to change the value: ");
    scanf("%d", &changeIdx);
    if(changeIdx > *top){
        printf("ERROR: Index out of bound.\n\n");
        goto label;
    }
    printf("Enter the new value: ");
    scanf("%d", &val);

    while(*top > changeIdx){
        temp[i++] = st[(*top)--];
    }
    st[changeIdx] = val;
    while(i > 0){
        st[++(*top)] = temp[--i];
    }
}

void display(int st[], int *top){
    for(int i = *top; i ≥ 0; i--){
        printf("%d\n", st[i]);
    }
}

void main(){

    int st[size], top = -1;

    printf("\n----------Push----------\n");
    push(st, &top);
    push(st, &top);
    push(st, &top);
    push(st, &top);
    push(st, &top);
    printf("----------Display----------\n");
    display(st, &top);
    printf("----------Pop----------\n");
    pop(st, &top);
    display(st, &top);
    printf("----------Peep----------\n");
    peep(st, &top);
    printf("\n----------Change----------\n");
    change(st, &top);
    printf("\nYour final stack is: \n");
    display(st, &top);
}
```

```c
#include<stdio.h>
#include<string.h>
#include<time.h>
#define size 50

void push(char arr[], int *top, char infix){
    if((*top) > size){
        printf("Stack Overflow\n");
        return;
    }
    else{
        arr[++(*top)] = infix;
    }
}

void pop(int *top){
    if((*top) < 0){
        printf("Stack Underflow\n");
        return;
    }
    else{
        (*top)--;
    }
}

char peep(char arr[], int *top){
    return arr[*top];
}

int precedence(char c){
    if (c == '^'){
        return 3;
    }
    else if(c == '*' || c == '/' || c == '%'){
        return 2;
    }
    else if(c == '+' || c == '-'){
        return 1;
    }
    else{
        return 0;
    }
}

int ifOperator(char c){
    if(c == '^' || c == '*' || c == '/' || c == '%' || c == '+' || c == '-' || c == '(' || c == ')'){
        return 1;
    }
    else{
        return 0;
    }
}

void main(){

    char infix[] = "((A+B)-C*(D/E))+F";
    strcat(infix, ")");
    char postfix[size], st[size];
    int top = -1, temp, i = 0, j = -1;
    push(st, &top, '(');

    for(i = 0; infix[i] ≠ '\0'; i++){
        if(ifOperator(infix[i]) == 0){
            push(postfix, &j, infix[i]);
        }
        else{
            push(st, &top, infix[i]);
            if(st[top] == ')'){
                pop(&top);
                while(st[top] ≠ '('){
                    push(postfix, &j, peep(st, &top));
                    pop(&top);
                }
                pop(&top);
            }
            else if(st[top] == '('){
                continue;
            }
            else if(precedence(st[top]) ≤ precedence(st[top - 1])){
                while(precedence(st[top]) ≤ precedence(st[top - 1])){
                    temp = peep(st, &top);
                    pop(&top);
                    push(postfix, &j, st[top]);
                    pop(&top);
                    push(st, &top, temp);
                }
            }
        }
    }

    printf("%s\n", postfix);
}
```

```c
#include <stdio.h>
#include <time.h>

void towerOfHanoi(int disks, char s, char m, char d) {
    if (disks == 1) {
        printf("Move disk 1 from %c to %c\n", s, d);
        return;
    }

    towerOfHanoi(disks - 1, s, d, m);
    printf("Move disk %d from %c to %c\n", disks, s, d);
    towerOfHanoi(disks - 1, m, s, d);
}

int main() {

    int disks;
    printf("Enter the number of disks: ");
    scanf("%d", &disks);

    towerOfHanoi(disks, 'A', 'B', 'C');

    return 0;
}
```

```c
#include<stdio.h>
#include<time.h>
#define n 10

void enqueue(int queue[], int *front, int *rear){
    int x;
    printf("Enter the number you want to add: ");
    scanf("%d", &x);

    if(*rear > n){
        printf("The queue is full.");
    }
    else if(*front = -1){
        (*rear)++;
        (*front)++;
        queue[*rear] = x;
    }
    else{
        (*rear)++;
        queue[*rear] = x;
    }

}

void dequeue(int queue[], int *front, int *rear){
    if(*rear < 0){
        printf("The queue is empty.");
    }
    else{
        (*front)++;
    }
}

void display(int queue[], int *front, int *rear){
    for(int i = (*front); i <= (*rear); i++){
        printf("%d ", queue[i]);
    }
}

void main(){

    int queue[n];
    int front = -1, rear = -1;

    printf("\n----------------Enqueue----------------\n");
    enqueue(queue, &front, &rear);
    enqueue(queue, &front, &rear);
    enqueue(queue, &front, &rear);
    enqueue(queue, &front, &rear);
    enqueue(queue, &front, &rear);
    display(queue, &front, &rear);

    printf("\n----------------Dequeue----------------\n");
    dequeue(queue, &front, &rear);
    display(queue, &front, &rear);
}
```

```c
#include<stdio.h>
#include<time.h>
#define n 5

void enqueue(int queue[], int *front, int *rear){
    int x;
    printf("Enter the number you want to add: ");
    scanf("%d", &x);

    if(((*rear)+1) % n == *front){
        printf("The queue is full.");
        return;
    }
    else if(*front = -1){
        (*rear)++;
        (*front)++;
        queue[*rear] = x;
    }
    else{
        (*rear) = ((*rear)+1) % n;
        queue[*rear] = x;
    }

}

void dequeue(int queue[], int *front, int *rear){
    if(*front==-1 && *rear==-1){
        printf("The queue is empty.");
    }
    else{
        (*front) = ((*front)+1) % n;
    }
}

void display(int queue[], int *front, int *rear){
    int i = (*front);
    while(i != (*rear)){
        printf("%d ", queue[i]);
        i++;
    }
    printf("%d ", queue[*rear]);
}

void main(){

    int queue[n];
    int front = -1, rear = -1;

    printf("\n----------------Enqueue----------------\n");
    enqueue(queue, &front, &rear);
    enqueue(queue, &front, &rear);
    enqueue(queue, &front, &rear);
    enqueue(queue, &front, &rear);
    enqueue(queue, &front, &rear);
    display(queue, &front, &rear);

    printf("\n----------------Dequeue----------------\n");
    dequeue(queue, &front, &rear);
    display(queue, &front, &rear);
}
```

```c
#include<stdio.h>
#include<time.h>

void main(){

    int arr[] = {43, 55, 114, 554, 6, 22, 98, 1, -42};
    int size = sizeof(arr) / sizeof(arr[0]);
    int isPresent = 0, value = 6, idx;

    for(idx = 0; idx < size; idx++){
        if(arr[idx] == value){
            isPresent = 1;
            break;
        }
    }

    if(isPresent == 1){
        printf("The value found at index: %d\n", idx);
    }
    else{
        printf("Value is found in the array.\n");
    }
}
```

```c
#include<stdio.h>
#include<time.h>

// Iterative Approach
int binarySearch(int arr[], int *lb, int *ub, int *value){
    int mid;
    while(*lb <= *ub){
        mid = (*lb + *ub) / 2;
        if(arr[mid] == *value){
            return mid;
        }
        else if(arr[mid] < *value){
            *lb = mid + 1;
        }
        else{
            *ub = mid - 1;
        }
    }
    return -1;
}

// Recursive Approach
// int binarySearch(int arr[], int lb, int ub, int value){
//     if(lb > ub){
//         return -1;
//     }
//     else{
//         int mid = (lb + ub) / 2;
//         if(arr[mid] == value){
//             return mid;
//         }
//         else if(arr[mid] < value){
//             return binarySearch(arr, mid + 1, ub, value);
//         }
//         else{
//             return binarySearch(arr, lb, mid - 1, value);
//         }
//     }
// }

void main(){

    int arr[] = {91,73,21,4,5,676,2,1,53,82,-1,-83};
    int value, lb = 0, ub = 7;

    printf("Enter the value you want to find: ");
    scanf("%d", &value);

    int found = binarySearch(arr, &lb, &ub, &value);  // Calling the function.

    if(found == -1){
        printf("Value not found.\n");
    }
    else{
        printf("The value %d found at index: %d\n", value, found);
    }
}
```

```c
#include<stdio.h>
#include<time.h>

void swap(int *a, int *b){      // swap function
    (*a) += (*b);
    (*b) = (*a) - (*b);
    (*a) -= (*b);
}

void bubbleSort(int arr[], int size){       // bubble sort function
    int ifSwapped, last = size - 1;

    for(int i = 0; i < size; i++){
        ifSwapped = 0;
        for(int j = 0; j < last; j++){
            if(arr[j] > arr[j + 1]){
                swap(&arr[j], &arr[j + 1]);
                ifSwapped++;
            }
        }
        if(ifSwapped == 0){
            break;
        }
        last--;
    }
}

void main(){

    int arr[] = {823, 54, 11, -53, 7, -23, 8, 0};
    int n = sizeof(arr)/sizeof(arr[0]);

    printf("Before sorting: ");
    for(int i = 0; i < n; i++){
        printf("%d ", arr[i]);
    }
    printf("\n");

    bubbleSort(arr, n);      // Calling the bubbleSort function

    printf("After sorting: ");
    for(int i = 0; i < n; i++){
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

```c
#include<stdio.h>

void swap(int *a, int *b){
    (*a) += (*b);
    (*b) = (*a) - (*b);
    (*a) -= (*b);
}

void insertionSort(int arr[], int size){
    int i, j;
    for(i = 0; i < size - 1; i++){
        if(arr[i] > arr[i + 1]){
            swap(&arr[i], &arr[i + 1]);
        }
        for(j = i; j > 0; j--){
            if(arr[j] < arr[j - 1]){
                swap(&arr[j], &arr[j - 1]);
            }
            else{
                break;
            }
        }
    }
}

void main(){

    int arr[] = {5,4,3,2,1,0};
    int n = sizeof(arr)/sizeof(arr[0]);

    insertionSort(arr, n);

    for(int i = 0; i < n; i++){
        printf("%d ", arr[i]);
    }
}
```

```c
#include<stdio.h>

void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}

void selectionSort(int arr[], int size){
    int i, j, min;
    for(i = 0; i < size; i++){
        min = i;
        for(j = i + 1; j < size; j++){
            if(arr[j] < arr[min]){
                min = j;
            }
        }
        if(arr[j] != arr[min]){
            swap(&arr[i], &arr[min]);
        }
    }
}

void main(){

    int arr[] = {5, 4, 3, 2, 1, 0};
    int n = sizeof(arr)/sizeof(arr[0]);

    selectionSort(arr, n);

    for(int i = 0; i < n; i++){
        printf("%d ", arr[i]);
    }
}
```

```c
#include<stdio.h>
#include<time.h>

void merge(int arr[], int left[], int right[], int size, int mid){
    int i = 0, j = 0, k = 0;
    int n = mid, m = size - mid;

    while(i < n && j < m){
        if(left[i] <= right[j]){
            arr[k++] = left[i++];
        }
        else{
            arr[k++] = right[j++];
        }
    }

    if(i < n && j == m){
        while(i < n){
            arr[k++] = left[i++];
        }
    }
    if(i == n && j < m){
        while(j < m){
            arr[k++] = right[j++];
        }
    }
}

void mergeSort(int arr[], int size){        // Creating merge sort function
    if(size <= 1){
        return;
    }
    int mid = size / 2;

    int left[mid], i;                   // Copying data into two sub arrays
    int right[size - mid], j;
    for(i = 0; i < mid; i++){
        left[i] = arr[i];
    }
    for(j = 0; j < (size - mid); j++){
        right[j] = arr[j + mid];
    }

    mergeSort(left, mid);                   // Dividing and sorting the left array
    mergeSort(right, size - mid);           // Dividing and sorting the right array
    merge(arr, left, right, size, mid);     // Merging all the arrays
}

void main(){

    int arr[] = {23, -4, 0, 32, 645, 311, -323, 76, 12, 1};
    int n = sizeof(arr)/sizeof(arr[0]);

    printf("Before sorting: ");
    for(int i = 0; i < n; i++){
        printf("%d ", arr[i]);
    }
    printf("\n");

    mergeSort(arr, n);      // Calling the mergeSort function

    printf("After sorting: ");
    for(int i = 0; i < n; i++){
        printf("%d ", arr[i]);
    }
    printf("\n");

}
```

```c
#include<stdio.h>
#include<time.h>

void swap(int *a, int *b){      // swap function
    int temp = *a;
    *a = *b;
    *b = temp;
}

void pivotPos(int arr[], int pivot, int size){      // funtion to put pivot element to its right place
    int count = 0;
    for(int i=0; i < size; i++){
        if(arr[i] < pivot){
            count++;
        }
    }
    swap(&arr[0], &arr[count]);
}

void quickSort(int arr[], int size, int l, int r){        // quick sort function
    int pivot = arr[0];
    pivotPos(arr, pivot, size);
    int left = 0, right = size - 1;

    if(left > right){
        return;
    }

    while(left < right){
        if(arr[left] < pivot){
            left++;
        }
        else if(arr[right] > pivot){
            right--;
        }
        else{
            swap(&arr[left], &arr[right]);
            left++;
            right--;
        }
    }
    quickSort(arr, left, 0, left);                        // sorting the left side of the pivot element
    quickSort(arr, size - left - 1, left, size - 1); // sorting the right side of the pivot element
}

void main(){

    int arr[] = {4, 5, 23, 6, -2, 0, 1212, -11111, -3};
    int n = sizeof(arr)/sizeof(arr[0]);

    printf("Before sorting: ");
    for(int i = 0; i < n; i++){
        printf("%d ", arr[i]);
    }
    printf("\n");

    quickSort(arr, n, 0, n-1);      // Calling the quickSort function

    printf("After sorting: ");
    for(int i = 0; i < n; i++){
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

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

typedef struct node{
    int data;
    struct node *next;
}node;

node* create(){
    int x;
    printf("Enter the value: ");
    scanf("%d", &x);

    node *new = (node *)malloc(sizeof(node));
    new->data = x;
    new->next = NULL;

    return new;
}

void insertAtBegin(node **head){
    node *new = create();

    new->next = *head;
    *head = new;
}

void insertAtEnd(node **head){
    node *temp, *new = create();

    if(*head == NULL){
        *head = new;
    }
    else{
        temp = *head;
        while(temp->next != NULL){
            temp = temp->next;
        }
        temp->next = new;
    }
}

void insertAtPos(node **head){
    int pos;
    printf("Enter the pos: ");
    scanf("%d", &pos);

    if(pos <= 0){
        printf("Enter the valid position");
    }
    else if(pos == 1){
        insertAtBegin(head);
    }
    else{
        node *temp = *head, *new = create();
        while(temp != NULL && (--pos) > 1){
            temp = temp->next;
        }
        if(temp == NULL || temp->next == NULL){
            printf("Position is greater than size.\n");
            return;
        }
        new->next = temp->next;
        temp->next = new;
    }
}

void insertInOrder(node **head){
    node *temp = *head, *new = create(), *prev = NULL;

    while((temp != NULL) && (new->data >= temp->data)){
        prev = temp;
        temp = temp->next;
    }
    if(prev == NULL){
        new->next = *head;
        *head = new;
    }
    else{
        prev->next = new;
        new->next = temp;
    }
}

void deleteHead(node **head){
    if(*head == NULL){
        printf("Linked list is already empty.");
    }
    *head = (*head)->next;
}

void deleteAtPos(node **head){
    int pos;
    printf("Enter the pos: ");
    scanf("%d", &pos);

    if(pos <= 0){
        printf("Enter the valid position");
    }
    else if(pos == 1){
        deleteHead(head);
    }
    else{
        node *temp = *head;
        while(temp != NULL && (--pos) > 1){
            temp = temp->next;
        }
        if(temp == NULL || temp->next == NULL){
            printf("Position is greater than size.\n");
            return;
        }
        temp->next = ((temp->next)->next);
    }
}

void deleteBefore(node **head){
    int pos;
    printf("Enter the pos: ");
    scanf("%d", &pos);

    if(pos <= 1){
        printf("Enter the valid position");
    }
    else if(pos == 2){
        deleteHead(head);
    }
    else{
        node *temp = *head, *before;
        while(temp != NULL && (--pos) > 1){
            before = temp;
            temp = temp->next;
        }
        if(temp == NULL || temp->next == NULL){
            printf("Position is greater than size.\n");
            return;
        }
        before->next = ((before->next)->next);
    }
}

void deleteAfter(node **head){
    int pos;
    printf("Enter the pos: ");
    scanf("%d", &pos);

    if(pos <= 0){
        printf("Enter the valid position");
    }
    else{
        node *temp = *head, *after;
        while(temp != NULL && (--pos) > 1){
            temp = temp->next;
        }
        if(temp == NULL || temp->next == NULL){
            printf("Position is greater than size.\n");
            return;
        }
        after = temp->next;
        after->next = ((after->next)->next);
    }
}

void display(node **head){
    node *curr = *head;
    while(curr != NULL){
        printf("%d->", curr->data);
        curr = curr->next;
    }
    printf("NULL\n");
}

void main(){

    node *head = NULL;

    insertAtEnd(&head);
    insertAtEnd(&head);
    insertAtEnd(&head);
    insertAtEnd(&head);
    insertAtEnd(&head);
    display(&head);

    deleteHead(&head);
    display(&head);
}
```

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

typedef struct node{
    int data;
    struct node *next;
}node;

node* create(){
    int x;
    printf("Enter the value: ");
    scanf("%d", &x);

    node *new = (node *)malloc(sizeof(node));
    new->data = x;
    new->next = NULL;

    return new;
}

void insertAtEnd(node **head, node **tail){
    node *new = create();

    if(*head == NULL){
        *head = *tail = new;
        new->next = *head;
    }
    else{
        new->next = *head;
        (*tail)->next = new;
        *tail = new;
    }
}

void insertBefore(node **head, node **tail){
    int pos;
    printf("Enter the pos: ");
    scanf("%d", &pos);
    node *new = create();

    if(pos == 1 || pos == 2){
        new->next = *head;
        (*tail)->next = new;
        *head = new;
    }
    else{
        node *temp = *head, *before;
        while((--pos) > 1){
            before = temp;
            temp = temp->next;
        }
        new->next = before->next;
        before->next = new;
    }
}

void deleteHead(node **head, node **tail){
    *head = (*head)->next;
    (*tail)->next = *head;
}

void deleteAfter(node **head, node **tail){
    int pos;
    printf("Enter the pos: ");
    scanf("%d", &pos);

    node *temp = *head, *after;
    while((--pos) > 1){
        temp = temp->next;
    }
    after = temp->next;
    after->next = ((after->next)->next);
}

void display(node **head){
    node *curr = *head;
    do{
        printf("%d->", curr->data);
        curr = curr->next;
    }
    while(curr != *head);
    printf("HEAD\n");
}

void main(){

    node *head = NULL;
    node *tail = head;

    insertAtEnd(&head, &tail);
    insertAtEnd(&head, &tail);
    insertAtEnd(&head, &tail);
    insertAtEnd(&head, &tail);
    insertBefore(&head, &tail);
    display(&head);
    deleteHead(&head, &tail);
    display(&head);
    deleteAfter(&head, &tail);
    display(&head);
}
```