Code:

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
int stack[100];
int size=100;
int top = -1;

void push(int a){
 if(top == size-1){
        return;
 }
 else{
 top = top+1;
 stack[top]=a;
 }
}

bool is_full(){
  if(top == size-1){
    return true;
 }
 else{
   return false;
 }
}

int pop(){
   if(top==-1){
       return NULL;
 }
else{
```

```
int ele = stack[top];
    top = top - 1;
    return ele;
  }
}
int peak(){
  if (top == -1){
    return NULL;
}
else{
  return stack[top];
  }
}
bool isempty(){
  if (top == -1){
  return true;
}
else{
  return false;
  }
}
void display(){
    for(int i=0; i<= top; i++){
        printf("%d ", stack[i]);
    }
}
int main(){
      printf("i am adding an ele \n");
       push(19);
       push(12);
       push(6);
       push(15);
```

display();
printf("\n");

printf("%d \n", pop());
display();
printf("\n");
printf("%d \n", is_empty());
printf("%d \n", is_full());
printf("%d \n", peak());

9.

Output

i am adding an ele.
14 12 15 13

13
14 12 15

0

0

15

7) WAP to convert given valid parenthesized infix arithmetic expression to postfix expression

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

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

char associativity (char c){
  if (c == '^')
     return 'R';
     return 'L';
}

void infix To Postfix (const char *s){
  int len = strlen(s);

char *result = (char *)malloc(len+1);
char * stack = (char *)malloc(len);
int resultIndex = 0;
int stackIndex = -1;
if (!result || !stack){
  printf("Memory allocation failed! \n");
  return;
```

```
}
for (int i = 0; i < len; i++) {
 char c = s[i];

if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'z')
|| (c >= '0' && c <= '9')) {
result [resultIndex++] = c;
}
else if (c == '(') {
 stack [++stackIndex] = c;
}
  else if (c == ')') {
  while (stackIndex >= 0 && stack [stackIndex] != '(') {
  result [resultIndex++] = stack [stackIndex--];
  }
  stackIndex--;
}
else {
 while (stackIndex >= 0 && (prec(c) < prec(stack [stack.
 IndexJ)) ||
(prec(c) == prec(stack [stackIndex]) && associativity(c)
 == 'L')) {
  result [resultIndex++] = stack [stackIndex--];
 }
 stack [++stackIndex] = c;
}
}
while (stackIndex >= 0) {
 result [resultIndex++] = stack [stackIndex--];
}
result [resultIndex] = '\0';
 printf ("%s\n", result);
```

```
free (result);
free (stack); }

int main() {
char exp[] = "a+b*(c^d-e)^(f+g*h)-i";
infix ToPostfix(exp);
return 0;
}
```

user input

output:

abcd^e-fgh*+^*+i-

Nameeta. M
7/10/2024

Implementation of circular queue

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

```
Define
# define size 5

int front = -1
int rear = -1        } pass by reference

Boolean is empty ( )
{
  return front == -1
}

Boolean if full()
{
  return front == (rear+1)% size ;
}

Void enqueue (int val, int arr[])
{
if (is full ())
{
printf (" overflow"); return;
}
if (is empty())
{
front = rear = 0;
arr [front] = val ;
```

```
}
else
{
rear = (rear + 1) % size;
arr[rear] = val;
}
}

int dequeue(int arr[])
{
if (isempty())
{
printf("underflow");
}
else
{
int val = arr[front];
front = (front + 1) % size;
return val;
}

void display(int arr[])
{

while (front != (rear + 1) % size
{
printf("%d ", arr[front]);

front = (front + 1) % size;
}
```

Execute
Namrata M
21/01/24

Output:-

Enter 1 to add
2 to remove
3 to display
4 to exit.

all cases

Create a link list and insert an element at first or last element.

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

struct Node {
    int data;
    struct Node *next;
};

struct Node* createNode(int data) {
    struct Node *newNode = (struct Node*) malloc(sizeof(
    struct Node));
    newNode -> data = data;
    newNode -> next = NULL;
    return newNode;
}

struct Node* createLinkedList(int data[], int size) {
    struct Node* head = NULL;
    struct Node* tail = NULL;

    for (int i=0; i<size; i++) {
        struct Node *newNode = createNode(data[i]);
        if (head == NULL) {
            head = newNode;
            tail = newNode;
        } else {
            tail -> next = newNode;
            tail = newNode;
        }
    }
```

```
return head;
}

struct Node* insertAtFirst(struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = head;
    return newNode;
}

void insertAtEnd(struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    if (head == NULL) {
        head = newNode;
        return;
    }

    struct Node* current = head;
    while (current->next != NULL)
        current = current->next;

    current->next = newNode;
}

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

```
int main() {
  int data[7] = {1,2,3};
  struct Node* linkedlist = create LinkedList (data,3);

  printf("Initial linked list : \n");
  display(linkedlist);

  LinkedList = insertAtFirst(LinkedList, 0);
  printf("After inserting 0 at the first position: \n");
  display(LinkedList);

  insertAtEnd(LinkedList, 4);
  printf("After inserting 4 at the end: \n");
  display(LinkedList);


struct Node* current = LinkedList;
struct Node * next;
while(current != NULL) {
  next = current -> Next;
  free(current);
  current = next;
}
  return 0;
}
```

execute
Namrata M
28/9/20

output:

Initial linked list:
1 -> 2 -> 3 -> NULL
enter your choice.
1 for addfirst
2 for addlast
3 to display
4 to exit.

1
enter the ele to add 48
enter your choice.
1 for addfirst
2 for addlast
3 for display:
4 to exit:

2
enter the ele to add 89
enter your choice
1 for add first
2 for add last
3 for display
4 to exit

3
48 -> 1 -> 2 -> 3 -> 89 -> NULL

o/p seen

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


struct Node {
  int data;
  struct Node * next;
};

struct Node* createNode(int data) {
  struct Node * newNode = (struct Node*) malloc
  (sizeof(struct Node));
   newNode->data = data;
  newNode->next = NULL;
  return newNode;
  }

void insertEnd(struct Node** head, int data) {
  struct Node *newNode = createNode(data);
   if (*head == NULL) {
      *head = newNode;
} else {
  struct Node *temp = *head;
  while (temp->next != NULL) {
     temp = temp->next;
  }
  temp->next = newNode;
}
}
```

```
void delete_first(struct Node **head) {
    if (*head == NULL) {
    printf("List is empty.\n");
    return;
}
struct node *temp = *head;
*head = (*head)->next;
free(temp);
}

void delete_element(struct Node **head, int key)
 {
   if (*head == NULL) {
   printf("List is empty.\n");
    return;
}

struct Node *temp = *head, *prev = NULL;

if (temp != NULL && temp->data == key) {
  *head = temp->next;
   free(temp);
   return;
}

while (temp != NULL && temp->data != key) {
    prev = temp;;
    temp = temp->next;
}

if (temp == NULL) {
printf("Element not found\n");
```

```
return;
}

prev->next = temp->next;
free(temp);
}

void deleteLast(struct node **head){
 if (*head==NULL){
  printf("List is empty.\n");
  return;
 }

 struct node *temp = *head;

 if (temp->next==NULL){
  *head = NULL;
  free(temp);
  return;
 }

 struct Node* secondLast = *head;
 while (secondLast->next->next!=NULL){
    secondLast = secondLast->next;
 }

 free(secondLast->next);
  secondLast->next=NULL;
}

void display(struct Node *head){
  if (head==NULL){
```

```
print ("List is empty.\n");
return;
}

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

int main() {
struct Node* head = NULL;
  int choice, data, key;

while (1) {
printf ("\n1. Insert at end\n2. Delete first\n3.
Delete specified element\n4. Delete last\n5.
Display\n6. Exit\n");
  printf ("\nEnter your choice:");
  scanf ("%d", &choice);

switch (choice) {
  case 1:
  printf ("\nEnter data to insert:");
  scanf ("%d", &data);
  insertEnd(&head, data);
  break;
case 2:
  deleteFirst(&head);
  break;
```

```
case 3:
printf("Enter element to delete:");
scanf("%d", &key);
deleteElement(&head, key);
break;
case 4:
deleteLast(&head);
break;
case 5:
display(head);
break;
case 6:
exit(0);
default:
printf("Invalid choice.\n");
}
}
return 0;
}
```

Output:

1. Insert at end
2. Delete first
3. Delete specified element
4. Delete last
5. Display
6. Exit

Enter your choice: 1
Enter data to insert: 20

1. Insert at end
2. Delete first
3. Delete specified element
4. Delete last
5. Display
6. Exit
Enter your choice: 1
Enter data to insert: 30

1. Insert at end
2. Delete first
3. Delete specified element
4. Delete last
5. Display
6. Exit
Enter your choice: 1
Enter data to insert: 40

1. Insert at end
2. Delete first
3. Delete specified element
4. Delete last
5. Display
6. Exit
Enter your choice: 5.
20 -> 30 -> 40 -> NULL

1. Insert at end
2. Delete first
3. Delete specified element
4. Delete last.
6. Exit.
5. Display.

Rafna Gold

Enter your choice: 2

1. Insert at end
2. Delete first
3. Delete specified element
4. Delete last
5. Display
6. Exit

Enter your choice: 3
Enter element to delete: 30

1. Insert at end
2. Delete first
3. Delete specified element
4. Delete last
5. Display
6. Exit

Enter your choice: 4

1. Insert at end
2. Delete first
3. Delete specified element
4. Delete last
5. Display
6. Exit

Enter your choice: 6

Navaneeta M
11/11/2021

WAP to implement singly linked list with following operation.
a) sort the linked list
b) reverse
c) concatenate.

→ #include <stdio.h>

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

Node * CreateNode (int data)
{
Node * newNode = (Node *) malloc (sizeof (node));
newNode -> data = data;
newNode -> next = NULL;
return newNode;
}
void add (int data, node **head)
{
Node * newNode = CreateNode (data);
if (*head == NULL)
{
*head = *newNode;
return *head;
}
node * temp = *head;
while (temp -> next != NULL)
{
temp = temp -> next;
}

```
temp->next = newnode;
}
}

void display(node *head)
{
if (head == NULL)
printf("empty");
return;
}
node *temp = head;
while (temp != NULL)
{
printf("%d", temp->data);
temp = temp->next;
}

void sort(node **head)
{
if (*head == NULL)
{
return;
}
node *temp1 = head;
node *temp2 = NULL;
int val;
while (temp1 != NULL){
temp2 = temp1->next;
while (temp2 != NULL){
if (temp1->data > temp2->data)
{
val = temp1->data;
temp1->data = temp2->data;
temp2->data = val;
```

```
}
temp2 = temp2 -> next;
}
temp2 = temp2 -> next;

void reverse (Node **head)
{
Node * prev = NULL;
Node * current = *head;
Node * next = NULL;

while (current != NULL)
{
next = current -> next;
current -> next = prev;
prev = current;
current = next;
}
*head = prev;
}

void concatenate (node ** head1, node **head2)
{
if (*head1 == NULL)
{
*head1 = *head2;
}
else
node * temp = *head1;
while (temp -> next != NULL)
temp = temp -> next;
}
temp -> next = *head2;
```

```
}
}

void main()
{
node *L1 = NULL;
node *L2 = NULL;
int choice, data;

while(1)
printf("1. Insert into L1\n 2. insert into L2\n
    3. Display L1\n 4. display L2 \n 5 sort list
    6. reverse list \n 7. Concatenate\n 8. Exit");

printf("\n Enter the choice:");
scanf("%d", &choice);

switch(choice)
{
case 1:
printf("Enter data to L1");
    scanf("%d", &data);
    add(&L1, data);
    break;

case 2:
printf("Enter data to insert to L2");
    scanf("%d", &data);
    add(&L2, data)
    break;

case 3: printf("L1");
        display(L1);
        break;
```

```
case 4: printf("L2");
        display(L2);
        break;

case 5: printf("Sorted list");
        sort(&L2);
        break;

case 6: printf("Reversed");
        reverse(&L2);
        break;

case 7: printf("concatenated");
        concatenate(&L2, &L2);
        break;

case 8: printf("Exit");
        exit(0);

default: printf("wrong choice");
}
return 0;
}
```

Program to implement stack and queue using LL:

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

node * create (int data)
{
node * newnode = (node *) malloc (sizeof (node));
    newnode -> data = data;
    newnode -> next = null;
    return newnode;
}

void add (int data, node **head);
{
node * newnode = create (data);
    if (*head == null)
    {
    *head = newnode;
    return;
    }
node * temp = *head;
    while (temp -> next != NULL)
    {
    temp = temp -> next;
    }
    temp -> next = newnode;
}
```

```
void push (Node ** top, int data)
{
Node * newNode = (Node *) malloc (sizeof(Node));
newNode -> next = *top;
*top = newNode;
printf ("push successful");
}

void pop (Node **top)
{
if (*top == NULL)
{
printf ("Empty stack");
return;
}
Node *temp = *top;
*top = (*top) -> next;
printf ("popped value %d", temp -> data);
}

void display (Node **top)
{
if (*top == NULL)
printf ("Empty");
return;
}
Node *temp = *top;
while (temp != NULL);
printf ("%d", temp -> data);
temp = temp -> next;
}
}

void enqueue (Node **front, Node **rear, int data)
{ node * newnode = create (data);
```

```
if (rear == NULL)
{
*rear = *front = newnode;
return;
}
*rear = *rear->next;
*rear = newnode;
printf("enqueued");
}

void dequeue(Node **front, Node **rear)
{
node *newNode = create(data);

if (*front == NULL);
{
printf("Empty");
return;
}
if (*front -> next = NULL)
{
*front = *rear = NULL;
return;

node *temp = *front;
*front = *front->next
printf("dequeue value is %d", temp->data);
}

void display(node *front, node *head);
{
if (*front == NULL){
```

```
printf("Empty");
return;
}
node * temp = *front;
while (temp != rear);
{
printf("%d", temp->data);
temp = temp -> next
}
```

Output.

Menu
→ 1
→ enter data : 45
Menu
→ 1
→ enter data : 50
Menu
→ 3
→ 45 → 50

Menu
→ 2
enter : 90
Menu
→ 2
enter : 99
Menu
→ 4
→ 90 → 99.

WAP to create a doubly linked list and platform

1) Insert at start
2) Insert at a specified position
3) Insert at rear
4) Display

```
#include <stdio.h>

typedef struct Node{
  int data;
  int prev;
  int next;
} node;

node *head = NULL;

void addfirst (int data)
{
node *nn = (node*) (malloc(sizeof(node)));
   nn->data = data;
   nn->prev = NULL;
   nn->next = NULL;
   if (head == NULL)
    {
   head = nn->next = head;
     head->prev = nn;
      head = nn;
   } return
```

```
void addmid(int data, int ps)
{ struct node *nn = (node*)malloc(sizeof(node));
    nn->data = data;
    nn->prev = NULL;
    nn->next = NULL;

    if (head == NULL)
    {
    head = nn;
    }

for(int i=0; i<pos-1; i++)
    {
    temp = temp->next;
    }

    nn->next = temp->next;
    nn->prev = temp;
    temp->next = nn;
    nn->next->prev = nn;
    }

void addleft(int data)
{
node *nn = (node*)malloc(sizeof(node));
    nn->data = data;
    nn->prev = NULL;
    nn->next = NULL;

if (head == NULL)
{
head = nn;
```

```
return;
}
node * temp = head;
while (temp->next != NULL)
{
temp = temp -> next;
}
temp -> next = nn;
nn -> prev = temp;
}

void display()
{
if (head == NULL)
{
printf("Empty LL");

node * temp = head;

while (temp != NULL);
{
printf("%d", temp -> data);
temp = temp -> next;
}
}
```

Name date: M 16/12/2021

Output =

Enter number of elements in

8) write a program
a) To construct a binary search tree
b) To traverse the tree using all the methods.

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

struct node{
 int value;
 struct node * left;
 struct Node * right;
};

Struct Node * newNode (int value){
Struct Node * node = (struct node*) malloc(sizeof struct)
 node → value = value;
 node → left = node → right = NULL;
  return node;
}

struct Node * insert (struct Node root, int value){
      if (root == NULL){
      return newnode(value);
  }
  if (value < root → value){
    root → left = insert (root → left, value); }
    else {
    root → right = insert (root → right, value); }
  return root;
}
```

```
}
}
```

Output:

```
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter item do insert: 10

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue contains 10

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 4
```

WAP to stimulate the working of a queue of integers using an array. Provide the following operations: insert, delete, display

```
#include <stdio.h>
#define size 10
int item, front=0, rear=-1, q[10];
void insert(int item){
if (rear==size-1){
printf("stack overflow");
return;
}
rear++;
q[rear]=item;
}
int delete(){
if (front>rear){
printf("Queue empty\n");
return -1;
}
return q[front++];
}

void display(){
int i;
if (front>rear){
printf("Queue empty\n");
}
return;
printf("Queue contents:\n");
for(i=front; i<=rear; i++){
printf("%d\n", q[i]);
```

```
int main() {
struct Graph g;
init Graph(&g, 6);

Output:

addEdge(&g, 0, 1);
addEdge(&g, 0, 2);
addEdge(&g, 1, 3);
addEdge(&g, 1, 4);
addEdge(&g, 2, 5);

bool visited[MAX] = {false};
printf("DFS Traversal");
dfs(&g, 0, visited);
printf("\n");
return 0;
}
```

Output:

DFS traversal: 0 1 3 4 2 5

5) Program to perform depth-first traversal.

```
#define MAX 100

struct graph {
  int vertices;
  int adj[MAX][MAX];
};

void initGraph(struct graph *g, int vertices) {
    g->vertices = vertices;
    for(int i=0; i<vertices; i++) {
    for(int j=0; j<vertices; i++) {
      g->adj[i][j]=0;
  }
  }
}

void addEdge(struct graph ~~struct~~, int u, int v) {
    g->adj[u][v]=1;
    g->adj[v][u]=1;
  }

void dfs(struct graph *g, int vertex, bool visited[]
    visited[vertex] = true;
    printf("%d", vertex);

for(int i=0; i<g->vertices; i++) {
  if(g->adj[vertex][i]==1 && !visited
      dfs(g, i, visited);
}
}
}
```

```
while (q.front != -1){
  int node = dequeue(&q);
printf("%d ", node);

for(int i=0; i<g->vertices; i++){
  if(g->adj[node][i] == 1 && !visited[i]){
     visited[i] = true;
     enqueue(&q, i);
  }
}
}
printf("\n");

int main(){
struct Graph* g
= createGraph(6);
addEdge(&g, 0, 1);
addEdge(&g, 0, 2);
addEdge(&g, 1, 3);
```

```
if (q->front > q->rear) {
    q->front = q->rear = -1;
}
return item;
}

struct graph {
    int vertices;
    int adj[MAX_VERTICES][MAX_VERTICES];
};

void initGraph(struct Graph* g, int vertices) {
    g->vertices = vertices;
    for (int i=0; i<vertices; i++) {
        for (int j=0; j<vertices; j++) {
            g->adj[i][j] = 0;
        }
    }
}

void addEdge(struct Graph* g, int u, int v) {
    g->adj[u][u] = v;
    g->adj[v][u] = 1;
}

void bfs(struct graph *g, int start) {
    bool visited[MAX] = {false};
    struct Queue q;
    initQueue(&q);

visited[start] = true;
enqueue(&q, start);

printf("BFS Traversal starting from node %d:
```

write a program to traverse a graph using BFS method.

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

# define max_VERTICES 100

struct Queue {
 int items [MAX_VERTICES];
  int front, rear;

void initQueue (struct Queue *q) {
   q -> front = q -> rear = -1;
}

void enqueue (struct Queue *q, int value) {
 if (q -> rear == MAX_VERTICES - 1) {
   printf ("Queue is full \n");
    return;
}
if (q -> front == -1) {
   q -> front = 0;
}
 q -> items [++q -> rear] = value;
}

int dequeue (struct Queue *q) {
   if (q -> front == -1) {
    printf ("Queue is empty");
    return -1;
}
int item = q -> items [q -> front++];
```

```
void inorder(struct Node *root){
  if (root != NULL){
    inorder(root->left);
    printf("%d", root->value);
    inorder(root->right);
  }
}

void postorder(struct Node *root){
  if (root != NULL){
    postorder(root->left);
    postorder(root->right);
    printf("%d", root->value);
  }
}

void preorder(struct Node *root){
  if (root != NULL){
    printf("%d", root->value);
    preorder(root->left);
    preorder(root->right);
  }
}
```