1) **WAP choice based program to perform following operations in Array. A) Insert a new element at beginning B) Insert a new element at last. C) Insert a new element at any position k. D) Delete an element from beginning E) Delete an element from last. F) Delete an element from any position k. G) Print Array.**

Code=>

```
#include <stdio.h>

#define MAX_SIZE 100

void insertAtBeginning(int arr[], int *size, int value) {
   if (*size < MAX_SIZE) {
      for (int i = *size; i > 0; i--) {
         arr[i] = arr[i - 1];
      }
      arr[0] = value;
      (*size)++;
   } else {
      printf("Array is full. Cannot insert at beginning.\n");
   }
}

void insertAtLast(int arr[], int *size, int value) {
   if (*size < MAX_SIZE) {
      arr[*size] = value;
      (*size)++;
   } else {
      printf("Array is full. Cannot insert at last.\n");
   }
}
```

```
void insertAtPosition(int arr[], int *size, int value, int position) {
    if (*size < MAX_SIZE && position >= 0 && position <= *size) {
        for (int i = *size; i > position; i--) {
            arr[i] = arr[i - 1];
        }
        arr[position] = value;
        (*size)++;
    } else {
        printf("Invalid position or array is full. Cannot insert at position %d.\n", position);
    }
}

void deleteFromBeginning(int arr[], int *size) {
    if (*size > 0) {
        for (int i = 0; i < *size - 1; i++) {
            arr[i] = arr[i + 1];
        }
        (*size)--;
    } else {
        printf("Array is empty. Cannot delete from beginning.\n");
    }
}

void deleteFromLast(int arr[], int *size) {
    if (*size > 0) {
        (*size)--;
    } else {
        printf("Array is empty. Cannot delete from last.\n");
    }
}
```

```
void deleteFromPosition(int arr[], int *size, int position) {
    if (*size > 0 && position >= 0 && position < *size) {
        for (int i = position; i < *size - 1; i++) {
            arr[i] = arr[i + 1];
        }
        (*size)--;
    } else {
        printf("Invalid position or array is empty. Cannot delete from position %d.\n", position);
    }
}

void printArray(int arr[], int size) {
    if (size == 0) {
        printf("Array is empty.\n");
    } else {
        printf("Array elements: ");
        for (int i = 0; i < size; i++) {
            printf("%d ", arr[i]);
        }
        printf("\n");
    }
}

int main() {
    int arr[MAX_SIZE];
    int size = 0;
    int choice, value, position;

    do {
        printf("\n1. Insert at beginning\n");
        printf("2. Insert at last\n");
        printf("3. Insert at any position\n");
        printf("4. Delete from beginning\n");
        printf("5. Delete from last\n");
        printf("6. Delete from any position\n");
        printf("7. Print array\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
              printf("Enter the value to insert at beginning: ");
              scanf("%d", &value);
             insertAtBeginning(arr, &size, value);
              break;
            case 2:
              printf("Enter the value to insert at last: ");
              scanf("%d", &value);
              insertAtLast(arr, &size, value);
              break;
            case 3:
              printf("Enter the value to insert: ");
              scanf("%d", &value);
              printf("Enter the position to insert at: ");
              scanf("%d", &position);
              insertAtPosition(arr, &size, value, position);
              break;
            case 4:
              deleteFromBeginning(arr, &size);
              break;
            case 5:
```

```
                deleteFromLast(arr, &size);
                break;
            case 6:
                printf("Enter the position to delete from: ");
                scanf("%d", &position);
                deleteFromPosition(arr, &size, position);
                break;
            case 7:
                printArray(arr, size);
                break;
            case 8:
            printf("Exit Successfully\n  ");
            printArray(arr,size);
             printf("Exit Successfully\n  ");
                break;
            default:
                printf("Invalid choice\n");
        }
    } while (choice != 8);

    return 0;
}
```

OUTPUT=>

```
1. Insert at beginning
2. Insert at last
3. Insert at any position
4. Delete from beginning
5. Delete from last
6. Delete from any position
7. Print array
8. Exit
```

```
Enter your choice: 1

Enter the value to insert at beginning: 5


1. Insert at beginning

2. Insert at last

3. Insert at any position

4. Delete from beginning

5. Delete from last

6. Delete from any position

7. Print array

8. Exit

Enter your choice: 2

Enter the value to insert at last: 10


1. Insert at beginning

2. Insert at last

3. Insert at any position

4. Delete from beginning

5. Delete from last

6. Delete from any position

7. Print array

8. Exit

Enter your choice: 1

Enter the value to insert at beginning: 21


1. Insert at beginning

2. Insert at last

3. Insert at any position

4. Delete from beginning

5. Delete from last

6. Delete from any position
```

```
7. Print array

8. Exit

Enter your choice: 4


1. Insert at beginning

2. Insert at last

3. Insert at any position

4. Delete from beginning

5. Delete from last

6. Delete from any position

7. Print array

8. Exit

Enter your choice: 8

Exit Successfully

  Array elements: 5 10

Exit Successfully
```

2) WAP choice based program to perform following operations in Single Linked List. A) Insert a new node at beginning B) Insert a new node at last. C) Insert a new node at any position k. D) Delete a node from beginning E) Delete a node from last. F) Delete a node from any position k. G) Print Linked List.

Code=>

```
#include <stdio.h>

#include <stdlib.h>

struct Node

{

   int data;

   struct Node *next;

};

void insertAtBeginning(struct Node **head, int data)

{

   struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));

   newNode->data = data;

   newNode->next = *head;

   *head = newNode;

}

void insertAtEnd(struct Node **head, int data)

{

   struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));

   newNode->data = data;

   newNode->next = NULL;

   if (*head == NULL)

   {

      *head = newNode;

      return;

   }

   struct Node *temp = *head;

   while (temp->next != NULL)
```

```
    {
        temp = temp->next;
    }
    temp->next = newNode;
}
void insertAtPosition(struct Node **head, int data, int position)
{
    if (position <= 0)
    {
        printf("Invalid position\n");
        return;
    }
    if (position == 1)
    {
        insertAtBeginning(head, data);
        return;
    }
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    struct Node *temp = *head;
    for (int i = 1; i < position - 1 && temp != NULL; i++)
    {
        temp = temp->next;
    }
    if (temp == NULL)
    {
        printf("Invalid position\n");
        free(newNode);
        return;
    }
    newNode->next = temp->next;
    temp->next = newNode;
}
void deleteFromBeginning(struct Node **head)
{
    if (*head == NULL)
    {
        printf("List is empty\n");
        return;
    }
    struct Node *temp = *head;
    *head = (*head)->next;
    free(temp);
}
void deleteFromEnd(struct Node **head)
{
    if (*head == NULL)
    {
        printf("List is empty\n");
        return;
    }
    if ((*head)->next == NULL)
    {
        free(*head);
        *head = NULL;
        return;
    }
    struct Node *temp = *head;
    while (temp->next->next != NULL)
    {
        temp = temp->next;
    }
    free(temp->next);
    temp->next = NULL;
}
void deleteFromPosition(struct Node **head, int position)
{
    if (*head == NULL)
    {
        printf("List is empty\n");
        return;
    }
    if (position <= 0)
    {
        printf("Invalid position\n");
        return;
    }
    if (position == 1)
    {
        deleteFromBeginning(head);
        return;
    }
    struct Node *temp = *head;
    for (int i = 1; i < position - 1 && temp != NULL; i++)
    {
        temp = temp->next;
    }
    if (temp == NULL || temp->next == NULL)
    {
        printf("Invalid position\n");
        return;
    }

    struct Node *nodeToDelete = temp->next;
    temp->next = temp->next->next;
    free(nodeToDelete);
}

void printList(struct Node *head)
{
    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, position;

    while (1)
    {
        printf("1. Insert at beginning\n");
        printf("2. Insert at last\n");
        printf("3. Insert at position\n");
        printf("4. Delete from beginning\n");
        printf("5. Delete from last\n");
        printf("6. Delete from position\n");
        printf("7. Print Linked List\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
           printf("Enter data: ");
           scanf("%d", &data);
           insertAtBeginning(&head, data);
           break;
        case 2:
           printf("Enter data: ");
           scanf("%d", &data);
           insertAtEnd(&head, data);
           break;
        case 3:
           printf("Enter data: ");
           scanf("%d", &data);
           printf("Enter position: ");
           scanf("%d", &position);
           insertAtPosition(&head, data, position);
           break;
        case 4:
           deleteFromBeginning(&head);
           break;
        case 5:
           deleteFromEnd(&head);
           break;
        case 6:
```

```
            printf("Enter position: ");
            scanf("%d", &position);
            deleteFromPosition(&head, position);
            break;
        case 7:
            printList(head);
            break;
        case 8:
            printList(head);
           printf("Exit Successfully\n  ");
            exit(0);
        default:
            printf("Invalid choice\n");
        }
    }

    return 0;
}
```

OUTPUT=>

```
1. Insert at beginning
2. Insert at last
3. Insert at position
4. Delete from beginning
5. Delete from last
6. Delete from position
7. Print Linked List
8. Exit
Enter your choice: 1
Enter data: 5
1. Insert at beginning
```

```
2. Insert at last

3. Insert at position

4. Delete from beginning

5. Delete from last

6. Delete from position

7. Print Linked List

8. Exit

Enter your choice: 1

Enter data: 10

1. Insert at beginning

2. Insert at last

3. Insert at position

4. Delete from beginning

5. Delete from last

6. Delete from position

7. Print Linked List

8. Exit

Enter your choice: 1

Enter data: 15

1. Insert at beginning

2. Insert at last

3. Insert at position

4. Delete from beginning

5. Delete from last

6. Delete from position

7. Print Linked List

8. Exit

Enter your choice: 2

Enter data: 65

1. Insert at beginning

2. Insert at last
```

```
3. Insert at position

4. Delete from beginning

5. Delete from last

6. Delete from position

7. Print Linked List

8. Exit

Enter your choice: 8

15 -> 10 -> 5 -> 65 -> NULL

Exit Successfully
```

3) WAP choice based program to perform following operations in Double Linked List. A) Insert a new node at beginning B) Insert a new node at last. C) Insert a new node at any position k. D) Delete a node from beginning E) Delete a node from last. F) Delete a node from any position k. G) Print Linked List.

Code=>

```
#include<stdio.h>

#include<stdlib.h>


struct Node {

   int data;

   struct Node* next;

   struct Node* prev;

};


void insertBegin(struct Node** head_ref) {

   int new_data;

   struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));


   printf("Enter the data of the new node: ");

   scanf("%d", &new_data);


   new_node->data = new_data;

   new_node->next = (*head_ref);

   new_node->prev = NULL;

   if ((*head_ref) != NULL)

      (*head_ref)->prev = new_node;

   (*head_ref) = new_node;

}


void insertEnd(struct Node** head_ref) {
   int new_data;
   struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
   struct Node *last = *head_ref;

   printf("Enter the data of the new node: ");
   scanf("%d", &new_data);

   new_node->data = new_data;
   new_node->next = NULL;
   new_node->prev = last;
   if (last != NULL)
      last->next = new_node;
   else
      *head_ref = new_node;
}

void insertAtPos(struct Node** head_ref, int pos) {
   int new_data;
   struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
   struct Node *temp = *head_ref;

   printf("Enter the data of the new node: ");
   scanf("%d", &new_data);

   if (pos == 1) {
      new_node->data = new_data;
      new_node->next = (*head_ref);
      new_node->prev = NULL;
      if ((*head_ref) != NULL)
          (*head_ref)->prev = new_node;
      (*head_ref) = new_node;
      return;
   }

   for (int i = 0; i < pos - 2 && temp != NULL; i++)
      temp = temp->next;

   if (temp == NULL) {
      printf("Invalid position.\n");
      return;
   }

   new_node->data = new_data;
   new_node->next = temp->next;
   new_node->prev = temp;
   if (temp->next != NULL)
      temp->next->prev = new_node;
   temp->next = new_node;
}

void deleteBegin(struct Node** head_ref) {
   if (*head_ref == NULL) {
      printf("List is empty.\n");
      return;
   }

   struct Node* temp = *head_ref;
   *head_ref = (*head_ref)->next;
   if (*head_ref != NULL)
      (*head_ref)->prev = NULL;
   free(temp);
}
```

```
void deleteEnd(struct Node** head_ref) {
    if (*head_ref == NULL) {
        printf("List is empty.\n");
        return;
    }

    struct Node* last = *head_ref;
    if (last->next == NULL) {
        free(last);
        *head_ref = NULL;
        return;
    }

    while (last->next != NULL)
        last = last->next;

    last->prev->next = NULL;
    free(last);
}

void deleteAtPos(struct Node** head_ref, int pos) {
    if (*head_ref == NULL) {
        printf("List is empty.\n");
        return;
    }

    struct Node *temp = *head_ref;

    if (pos == 1) {
        *head_ref = temp->next;
        if (*head_ref != NULL)
            (*head_ref)->prev = NULL;
        free(temp);
        return;
    }

    for (int i = 0; i < pos - 2 && temp != NULL; i++)
        temp = temp->next;

    if (temp == NULL || temp->next == NULL) {
        printf("Invalid position.\n");
        return;
    }

    struct Node* nodeToDelete = temp->next;
    temp->next = temp->next->next;
    if (temp->next != NULL)
        temp->next->prev = temp;
    free(nodeToDelete);
}

void printList(struct Node* node) {
    while (node != NULL) {
        printf("%d <-> ", node->data);
        node = node->next;
    }
    printf("NULL\n");
}

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


    while (1) {
        printf("\n1. Insert at beginning\n2. Insert at end\n3. Insert at position\n4. Delete from
beginning\n5. Delete from end\n6. Delete from position\n7. Print List\n8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);


        switch (choice) {
            case 1:
                insertBegin(&head);
                break;
            case 2:
                insertEnd(&head);
                break;
            case 3:
                printf("Enter the position: ");
                scanf("%d", &pos);
                insertAtPos(&head, pos);
                break;
            case 4:
                deleteBegin(&head);
                break;
            case 5:
                deleteEnd(&head);
                break;
            case 6:
                printf("Enter the position: ");
                scanf("%d", &pos);
                deleteAtPos(&head, pos);
                break;
            case 7:
                printList(head);
                break;
            case 8:
            printList(head);
            printf("Exit SuccessFully");
                exit(0);
            default:
                printf("Invalid choice.\n");
        }
    }

    return 0;
}
```

OUTPUT=>

```

1. Insert at beginning
2. Insert at end
3. Insert at position
4. Delete from beginning
5. Delete from end
6. Delete from position
7. Print List
8. Exit
Enter your choice: 1
Enter the data of the new node: 5

1. Insert at beginning
2. Insert at end
3. Insert at position
```

```
4. Delete from beginning
5. Delete from end
6. Delete from position
7. Print List
8. Exit
Enter your choice: 2
Enter the data of the new node: 10

1. Insert at beginning
2. Insert at end
3. Insert at position
4. Delete from beginning
5. Delete from end
6. Delete from position
7. Print List
8. Exit
Enter your choice: 3
Enter the position: 3
Enter the data of the new node: 15

1. Insert at beginning
2. Insert at end
3. Insert at position
4. Delete from beginning
5. Delete from end
6. Delete from position
7. Print List
8. Exit
Enter your choice: 8
5 <-> 10 <-> 15 <-> NULL
Exit SuccessFully
```

4) WAP choice based program to perform following operations in Circular Linked List. A) Insert a new node at beginning B) Insert a new node at last. C) Insert a new node at any position k. D) Delete a node from beginning E) Delete a node from last. F) Delete a node from any position k. G) Print Linked List.

Code=>

```
#include <stdio.h>

#include <stdlib.h>


struct Node {

   int data;

   struct Node* next;

};


void insertAtBeginning(struct Node** start_ref, int new_data) {

   struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));


   if (new_node == NULL) {

      printf("Memory out of space\n");

      return;

   }


   new_node->data = new_data;


   if (*start_ref == NULL) {

      new_node->next = new_node;

      *start_ref = new_node;

   } else {

      new_node->next = (*start_ref)->next;

      (*start_ref)->next = new_node;

      *start_ref = new_node;

   }
```

```
    printf("Node inserted\n");
}

void insertAtLast(struct Node** start_ref, int new_data) {
    struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));

    if (new_node == NULL) {
        printf("Memory out of space\n");
        return;
    }

    new_node->data = new_data;

    if (*start_ref == NULL) {
        new_node->next = new_node;
        *start_ref = new_node;
    } else {
        new_node->next = (*start_ref)->next;
        (*start_ref)->next = new_node;
        *start_ref = new_node;

        struct Node* temp = *start_ref;
        while (temp->next != *start_ref) {
            temp = temp->next;
        }
        temp->next = *start_ref;
    }

    printf("Node inserted\n");
}
```

```
void insertAtPosition(struct Node** start_ref, int new_data, int position) {
    struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
    struct Node* temp = *start_ref;

    if (new_node == NULL) {
        printf("Memory out of space\n");
        return;
    }

    new_node->data = new_data;

    if (position == 1) {
        if (*start_ref == NULL) {
            new_node->next = new_node;
            *start_ref = new_node;
        } else {
            new_node->next = temp->next;
            temp->next = new_node;
            *start_ref = new_node;
        }
    } else {
        for (int i = 0; i < position - 2; i++) {
            if (temp != NULL) {
                temp = temp->next;
            }
        }

        if (temp == NULL) {
            printf("Position out of range\n");
            return;
        }

        new_node->next = temp->next;
        temp->next = new_node;
    }

    printf("Node inserted\n");
}

void deleteFromBeginning(struct Node** start_ref) {
    struct Node* temp = *start_ref;

    if (temp == NULL) {
        printf("List is empty\n");
        return;
    }

    if (*start_ref == (*start_ref)->next) {
        *start_ref = NULL;
    } else {
        *start_ref = temp->next;
        temp->next = NULL;
    }

    free(temp);

    printf("Node deleted\n");
}

void deleteFromLast(struct Node** start_ref) {
    struct Node* temp = *start_ref;
    struct Node* prev;

    if (temp == NULL) {
        printf("List is empty\n");
        return;
    }

    if (*start_ref == (*start_ref)->next) {
        *start_ref = NULL;
        free(temp);
    } else {
        while (temp->next != *start_ref) {
            prev = temp;
            temp = temp->next;
        }

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

    printf("Node deleted\n");
}

void deleteFromPosition(struct Node** start_ref, int position) {
    struct Node* temp = *start_ref;
    struct Node* prev;

    if (temp == NULL) {
        printf("List is empty\n");
        return;
    }

    if (position == 1) {
        if (*start_ref == (*start_ref)->next) {
            *start_ref = NULL;
            free(temp);
        } else {
            *start_ref = temp->next;
            temp->next = NULL;
            free(temp);
        }
    } else {
        for (int i = 0; i < position - 2; i++) {
            if (temp != NULL) {
                temp = temp->next;
            }
        }

        if (temp == NULL) {
            printf("Position out of range\n");
            return;
        }

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

    printf("Node deleted\n");
}
```

```
void printList(struct Node* node) {
    struct Node* temp = node;

    if (node == NULL) {
        printf("List is empty\n");
        return;
    }

    printf("Nodes of the Circular Linked List are: ");

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

    printf("\n");
}

int main() {
    struct Node* start = NULL;
    int choice, data, position;

    while (1) {
        printf("1. Insert in beginning\t");
        printf("2. Insert at last\t");
        printf("3. Insert at any position\t");
        printf("\n4. Delete from Beginning\t");
        printf("5. Delete from last\t");
        printf("6. Delete from any position\t");
        printf("\n7. Print Linked List\n");
printf("8. Exit\n");
printf("Enter your choice ");
scanf("%d", &choice);

switch (choice) {
  case 1:
    printf("Enter the node data? ");
    scanf("%d", &data);
    insertAtBeginning(&start, data);
    break;
  case 2:
    printf("Enter Data? ");
    scanf("%d", &data);
    insertAtLast(&start, data);
    break;
  case 3:
    printf("Enter Data? ");
    scanf("%d", &data);
    printf("Enter position? ");
    scanf("%d", &position);
    insertAtPosition(&start, data, position);
    break;
  case 4:
    deleteFromBeginning(&start);
    break;
  case 5:
    deleteFromLast(&start);
    break;
  case 6:
    printf("Enter position? ");
    scanf("%d", &position);
```

```
            deleteFromPosition(&start, position);
            break;
          case 7:
            printList(start);
            break;
          case 8:
          printList(start);
          printf("ExitSuccessFully");
            exit(0);
          default:
            printf("Please enter valid choice\n");
        }
    }

    return 0;
}
```

OUTPUT=>

```
1. Insert in beginning  2. Insert at last      3. Insert at any position
4. Delete from Beginning       5. Delete from last    6. Delete from any position
7. Print Linked List
8. Exit
Enter your choice 1
Enter the node data? 5
Node inserted
1. Insert in beginning  2. Insert at last      3. Insert at any position
4. Delete from Beginning       5. Delete from last    6. Delete from any position
7. Print Linked List
8. Exit
Enter your choice 2
Enter Data? 10
Node inserted
```

```
1. Insert in beginning  2. Insert at last      3. Insert at any position

4. Delete from Beginning       5. Delete from last    6. Delete from any position

7. Print Linked List

8. Exit

Enter your choice 1

Enter the node data? 15

Node inserted

1. Insert in beginning  2. Insert at last      3. Insert at any position

4. Delete from Beginning       5. Delete from last    6. Delete from any position

7. Print Linked List

8. Exit

Enter your choice 2

Enter Data? 155

Node inserted

1. Insert in beginning  2. Insert at last      3. Insert at any position

4. Delete from Beginning       5. Delete from last    6. Delete from any position

7. Print Linked List

8. Exit

Enter your choice 8

Nodes of the Circular Linked List are: 155 5 10 15

Exit SuccessFully
```

5) WAP choice based program to perform following operations in Doubly Circular Linked List. A) Insert a new node at beginning B) Insert a new node at last. C) Insert a new node at any position k. D) Delete a node from beginning E) Delete a node from last. F) Delete a node from any position k. G) Print Linked List.

Code=>

```
#include <stdio.h>

#include <stdlib.h>

struct Node

{

   int data;

   struct Node *prev;

   struct Node *next;

};

struct Node *head = NULL;

void insertAtBeginning(int value)

{

   struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));

   newNode->data = value;

   if (head == NULL)

   {

      newNode->prev = newNode;

      newNode->next = newNode;

      head = newNode;

   }

   else

   {

      struct Node *last = head->prev;

      newNode->prev = last;

      newNode->next = head;

      head->prev = newNode;

      last->next = newNode;
```

```
        head = newNode;
    }
}
void insertAtLast(int value)
{
    if (head == NULL)
    {
        insertAtBeginning(value);
    }
    else
    {
        struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
        newNode->data = value;
        struct Node *last = head->prev;
        newNode->prev = last;
        newNode->next = head;
        head->prev = newNode;
        last->next = newNode;
    }
}
void insertAtPosition(int value, int position)
{
    if (position <= 0)
    {
        printf("Invalid position\n");
        return;
    }
    if (position == 1)
    {
        insertAtBeginning(value);
    }
    else
    {
        struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
        newNode->data = value;
        struct Node *current = head;
        for (int i = 1; i < position - 1; i++)
        {
            if (current->next == head)
            {
                printf("Invalid position\n");
                return;
            }
            current = current->next;
        }
        newNode->prev = current;
        newNode->next = current->next;
        current->next->prev = newNode;
        current->next = newNode;
    }
}
void deleteFromBeginning()
{
    if (head == NULL)
    {
        printf("List is empty. Cannot delete from beginning.\n");
    }
    else
    {
        struct Node *last = head->prev;
        if (head == last)
        {
            free(head);
            head = NULL;
        }
        else
        {
            struct Node *temp = head;
            last->next = head->next;
            head = head->next;
            head->prev = last;
            free(temp);
        }
    }
}

void deleteFromLast()
{
    if (head == NULL)
    {
        printf("List is empty. Cannot delete from last.\n");
    }
    else
    {
        struct Node *last = head->prev;
        if (head == last)
        {
            free(head);
            head = NULL;
        }
        else
        {
            struct Node *temp = last;
            last->prev->next = head;
            head->prev = last->prev;
            free(temp);
        }
    }
}

void deleteFromPosition(int position)
{
    if (position <= 0 || head == NULL)
    {
        printf("Invalid position or list is empty\n");
    }
    else if (position == 1)
    {
        deleteFromBeginning();
    }
    else
    {
        struct Node *current = head;
        for (int i = 1; i < position; i++)
        {
            if (current->next == head)
            {
                printf("Invalid position\n");
                return;
            }
            current = current->next;
        }

        current->prev->next = current->next;
        current->next->prev = current->prev;
        free(current);
    }
}
void printList()
{
    if (head == NULL)
    {
        printf("List is empty\n");
    }
    else
    {
        struct Node *current = head;
        do
        {
            printf("%d ", current->data);
            current = current->next;
        } while (current != head);
        printf("\n");
    }
}
int main()
{
    int choice, value, position;
    do
    {
        printf("\n1. Insert at beginning\t");
        printf("2. Insert at last\t");
        printf("3. Insert at any position\t");
        printf("\n4. Delete from beginning\t");
        printf("5. Delete from last\t");
        printf("6. Delete from any position\t");
        printf("\n7. Print linked list\n");


        printf("\n8. Exit\n");
        printf("Enter your choice:\t ");
        scanf("%d", &choice);


        switch (choice)
        {
        case 1:
            printf("Enter the value to insert at beginning: ");
            scanf("%d", &value);
            insertAtBeginning(value);
            break;
        case 2:
            printf("Enter the value to insert at last: ");
            scanf("%d", &value);
            insertAtLast(value);
            break;
        case 3:
            printf("Enter the value to insert: ");
            scanf("%d", &value);
            printf("Enter the position to insert at: ");
            scanf("%d", &position);
            insertAtPosition(value, position);
            break;
        case 4:
            deleteFromBeginning();
            break;
        case 5:
            deleteFromLast();
```

```
            break;
        case 6:
            printf("Enter the position to delete from: ");
            scanf("%d", &position);
            deleteFromPosition(position);
            break;
        case 7:
            printList();
            break;
        case 8:
        printList();
        printf("\nExit SuccessFully");
            break;
        default:
            printf("Invalid choice\n");
        }
    } while (choice != 8);

    return 0;
}
```

OUTPUT=>

```
1. Insert at beginning  2. Insert at last       3. Insert at any position
4. Delete from beginning        5. Delete from last     6. Delete from any position
7. Print linked list

8. Exit
Enter your choice:      1
Enter the value to insert at beginning: 5

1. Insert at beginning  2. Insert at last       3. Insert at any position
4. Delete from beginning        5. Delete from last     6. Delete from any position
```

```
7. Print linked list

8. Exit
Enter your choice:      2
Enter the value to insert at last: 10

1. Insert at beginning  2. Insert at last      3. Insert at any position
4. Delete from beginning       5. Delete from last     6. Delete from any position
7. Print linked list

8. Exit
Enter your choice:      2
Enter the value to insert at last: 15

1. Insert at beginning  2. Insert at last      3. Insert at any position
4. Delete from beginning       5. Delete from last     6. Delete from any position
7. Print linked list

8. Exit
Enter your choice:      2
Enter the value to insert at last: 20

1. Insert at beginning  2. Insert at last      3. Insert at any position
4. Delete from beginning       5. Delete from last     6. Delete from any position
7. Print linked list

8. Exit
Enter your choice:      8
5<-> 10<-> 15<-> 20<->5
Exit SuccessFully
```

6) WAP choice based program to perform following operations on STACK a) PUSH B) POP c) Display

Code=>

```
//write a c program to implement stack

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

#define MAX 10

int stack[MAX];
int top = -1;

void push(int value) {
  if (top >= MAX - 1) {
    printf("Stack Overflow\n");
    return;
  }
  stack[++top] = value;
  printf("Pushed %d to the stack\n", value);
}

int pop() {
  if (top < 0) {
    printf("Stack Underflow\n");
    return -1;
  }
  return stack[top--];
}

void display() {
    if (top < 0) {
        printf("Stack is empty\n");
        return;
    }
    for (int i = top; i >= 0; i--) {
        printf("%d ", stack[i]);
    }
    printf("\n");
}

int main() {
    int choice, value;

    while (1) {
        printf("1.Push\n2.Pop\n3.Display\n4.Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(value);
                break;
            case 2:
                value = pop();
                if (value != -1) {
                    printf("Popped %d from the stack\n", value);
                }
                break;
            case 3:
```

```
            display();
            break;
          case 4:
           display();
           printf("Exit SucessFully");
            exit(0);
          default:
            printf("Invalid choice, please try again\n");
        }
    }

    return 0;
}
```

OUTPUT=>

```
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 1
Enter value to push: 5
Pushed 5 to the stack
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 1
Enter value to push: 10
Pushed 10 to the stack
1.Push
2.Pop
3.Display
```

```
4.Exit

Enter your choice: 1

Enter value to push: 15

Pushed 15 to the stack

1.Push

2.Pop

3.Display

4.Exit

Enter your choice: 2

Popped 15 from the stack

1.Push

2.Pop

3.Display

4.Exit

Enter your choice: 4

10 5

Exit SuccessFully
```

7) WAP choice based program to perform following operations on QUEUE a) Insert b) Delete

Code=>

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

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

void insert(QNode **front, QNode **rear, int data)
{
   QNode *newNode = (QNode *)malloc(sizeof(QNode));
   newNode->data = data;
   newNode->next = NULL;

   if (*rear == NULL)
   {
      *front = *rear = newNode;
      return;
   }

   (*rear)->next = newNode;
   *rear = newNode;
}

void delete(QNode **front, QNode **rear)
{
   if (*front == NULL)
```

```
    {
        printf("Queue is empty.\n");
        return;
    }

    QNode *temp = *front;
    *front = (*front)->next;

    if (*front == NULL)
        *rear = NULL;

    free(temp);
}

void display(QNode *front)
{
    QNode *temp = front;
    while (temp != NULL)
    {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main()
{
    QNode *front = NULL, *rear = NULL;
    int choice, data;

    while (1)
    {
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            printf("Enter the value to be inserted: ");
            scanf("%d", &data);
            insert(&front, &rear, data);
            break;
        case 2:
            delete (&front, &rear);
            break;
        case 3:
            display(front);
            break;
        case 4:
            display(front);
            printf("Exit SuccessFully");
            exit(0);
        default:
            printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}
```

OUTPUT=>

```
1. Insert

2. Delete

3. Display

4. Exit

Enter your choice: 1

Enter the value to be inserted: 5

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice: 1

Enter the value to be inserted: 10

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice: 1

Enter the value to be inserted: 15

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice: 2

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice: 3

10 15

1. Insert
```

```
2. Delete

3. Display

4. Exit

Enter your choice: 4

10 15

Exit SuccessFully
```

8) WAP choice based program to perform following operations on CIRCULAR QUEUE a) Insert b) Delete

Code=>

```
//write a c program to implement circular queue

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

#define SIZE 5

int Q[SIZE];
int front = -1;
int rear = -1;

int isFull() {
    if((rear + 1) % SIZE == front) {
        return 1;
    }
    return 0;
}

int isEmpty() {
    if(front == -1) {
        return 1;
    }
    return 0;
}

void insert(int val) {
    if(isFull()) {
        printf("Queue is Full\n");
    } else {
        if(front == -1) {
            front = 0;
        }
        rear = (rear + 1) % SIZE;
        Q[rear] = val;
        printf("Inserted %d\n", val);
    }
}

void delete() {
    if(isEmpty()) {
        printf("Queue is Empty\n");
    } else {
        if(front == rear) {
            front = -1;
            rear = -1;
        } else {
            front = (front + 1) % SIZE;
        }
        printf("Deleted element\n");
    }
}
void display() {
    if(isEmpty()) {
        printf("Queue is Empty\n");
    } else {
        int i = front;
        do {
            printf("%d ", Q[i]);
            i = (i + 1) % SIZE;
        } while(i != (rear + 1) % SIZE);
        printf("\n");
    }
}
int main() {
    int choice, value;
    while(1) {
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Display\n");
        printf("4. Quit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch(choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insert(value);
                break;
            case 2:
                delete();
                break;
            case 3:
                display();
                break;
            case 4:
             display();
            printf("Exit SuccessFully");
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

Output=>

```
1. Insert
2. Delete
3. Display
4. Quit
Enter your choice: 1
Enter value to insert: 5
Inserted 5
1. Insert
2. Delete
3. Display
4. Quit
Enter your choice: 1
Enter value to insert: 10
Inserted 10
1. Insert
2. Delete
3. Display
4. Quit
Enter your choice: 1
Enter value to insert: 15
Inserted 15
1. Insert
2. Delete
3. Display
4. Quit
```

```
Enter your choice:

2

Deleted element

1. Insert

2. Delete

3. Display

4. Quit

Enter your choice: 4

10 15

Exit SuccessFully
```

9) WAP to implement linear search.

Code=>

```
#include <stdio.h>
int linear_search(int *a, int key,int size)
{
   int index = 0;
   for(int i=0;i<size;i++)
     {
      if (a[i] == key)
         return index;
   }
   return -1;
}
int main()
{
   int n, key, index;
   int a[100];
   printf("Enter the SIZE : ");
   scanf("%d", &n);
   printf("Enter the element : ");
   for (int i = 0; i < n; i++)
   {
      scanf("%d", &a[i]);
   }
   printf("Enter the key to search for: ");
   scanf("%d", &key);
   index = linear_search(a, key,n);
   if (index != -1)
   {
      printf("Element found at index %d\n", index);
```

```
    }
    else
    {
        printf("Element not found in the  Arrayt\n");
    }
    return 0;
}
```

OUTPUT=>

```
Enter the SIZE : 10
Enter the element : 1
2
3
4
6
-99
78
102
45
333
Enter the key to search for: -99
Element found at index
```

10) WAP to implement binary search.

Code=>

```
#include <stdio.h>

#include <stdlib.h>

int binarySearch(int *a, int target, int left, int right)

{

    while (left <= right)

    {

        int mid = left + (right - left) / 2;

        if (a[mid] == target)

        {

            return 1;

        }

        if (a[mid] < target)

        {


            left = mid + 1;

        }

        else

        {

            right = mid - 1;

        }

    }

    return 0;

}


int main()

{

    struct Node *head = NULL;

    int array[] = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19};
    int n = sizeof(array) / sizeof(array[0]);
    printf("Given Array [");
    for (int i = 0; i < n; i++)
    {
      printf("%d ",array[i]);
    }
    printf("]\n");
    int target;
    printf("Enter the target value: ");
    scanf("%d", &target);
    if (binarySearch(array, target, 0, n - 1))
    {
        printf("Element %d is present in the list.\n", target);
    }
    else
    {
        printf("Element %d is not present in the list.\n", target);
    }
    return 0;
}
```

OUTPUT=>

```
Given Array [1 3 5 7 9 11 13 15 17 19 ]
Enter the target value: 21
Element 21 is not present in the list.
Given Array [1 3 5 7 9 11 13 15 17 19 ]
Enter the target value: 3
Element 3 is present in the list.
```

11) WAP to implement quick sort.

Code=>

```
#include <stdio.h>
void swap(int* a, int* b) {
    int t = *a;
    *a = *b;
    *b = t;
}
int partition (int arr[], int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high- 1; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
int main() {
    int n, i;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter %d integers: ", n);
    for(i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    quickSort(arr, 0, n-1);
    printf("\nSorted array: \n");
    for(i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

OUTPUT=>

```
Enter the number of elements: 8
Enter 8 integers:
1
5
6
4
8
77
-99
0
```

Sorted array:

-99 0 1 4 5 6 8 77

12) WAP to implement insertion sort.

```
#include <stdio.h>
void insertionSort(int arr[], int n)
{
    int i, j, key;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
void printArray(int arr[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
int main()
{
    int n, i;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter %d integers: ", n);
    for (i = 0; i < n; i++)
        scanf("%d", &arr[i]);
    insertionSort(arr, n);
    printArray(arr, n);
    return 0;
}
```

OUTPUT=>

```
Enter the number of elements: 6
Enter 6 integers:
-2
-99
-877
-987
1
5
SORTED ARRAY:
-987 -877 -99 -2 1 5
```

13) WAP to implement merge sort.

Code=>

```
#include <stdio.h>
void merge(int array[], int start, int mid, int end)
{
    int i, j, k;
    int n1 = mid - start + 1;
    int n2 = end - mid;
    int L[n1], R[n2];
    for (i = 0; i < n1; i++)
        L[i] = array[start + i];
    for (j = 0; j < n2; j++)
        R[j] = array[mid + 1 + j];
    i = 0;
    j = 0;
    k = start;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            array[k] = L[i];
            i++;
        }
        else
        {
            array[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1)
    {
        array[k] = L[i];
        i++;
        k++;
    }
    while (j < n2)
    {
        array[k] = R[j];
        j++;
        k++;
    }
}
void mergeSort(int array[], int start, int end)
{
    if (start < end)
    {
        int mid = (start + end) / 2;
        mergeSort(array, start, mid);
        mergeSort(array, mid + 1, end);

        merge(array, start, mid, end);
    }
}
int main()
{
    int array[50], n, i;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    printf("Enter the elements: ");
    for (i = 0; i < n; i++)
```

```
        scanf("%d", &array[i]);
    mergeSort(array, 0, n - 1);
    printf("\nSorted array: \n");
    for (i = 0; i < n; i++)
        printf("%d ", array[i]);
    return 0;
}
```

OUTPUT=>

```
Enter the number of elements: 6
Enter the elements:
-888
-999
-1000
-1
-20
55
Sorted array:
-1000 -999 -888 -20 -1 55
```

14) WAP to create a binary tree and then traverse using in order, pre order post order traversing techniques.

Code=>

```
#include <stdio.h>

#include <stdlib.h>

struct node {

   int data;

   struct node* left;

   struct node* right;

};

struct node* createNode(int data) {

   struct node* newNode = (struct node*)malloc(sizeof(struct node));

   newNode->data = data;

   newNode->left = NULL;

   newNode->right = NULL;

   return newNode;

}

void inorderTraversal(struct node* root) {

   if (root == NULL)

      return;

   inorderTraversal(root->left);

   printf("%d ", root->data);

   inorderTraversal(root->right);

}

void preorderTraversal(struct node* root) {

   if (root == NULL)

      return;

   printf("%d ", root->data);

   preorderTraversal(root->left);

   preorderTraversal(root->right);

}
```

```
void postorderTraversal(struct node* root) {
    if (root == NULL)
        return;
    postorderTraversal(root->left);
    postorderTraversal(root->right);
    printf("%d ", root->data);
}
int main() {
    int data, n, i;
    struct node *root, *temp;
    printf("Enter the number of nodes in the binary tree: ");
    scanf("%d", &n);
    printf("Enter the values for the binary tree:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &data);
        if (i == 0) {
            root = createNode(data);
        } else {
            temp = root;
            while (temp != NULL) {
                if (data < temp->data) {
                    if (temp->left == NULL) {
                        temp->left = createNode(data);
                        break;
                    } else {
                        temp = temp->left;
                    }
                } else {
                    if (temp->right == NULL) {
                        temp->right = createNode(data);
                        break;
                    } else {
                        temp = temp->right;
                    }
                }
            }
        }
    }
    printf("In-order traversal: ");
    inorderTraversal(root);
    printf("\n");
    printf("Pre-order traversal: ");
    preorderTraversal(root);
    printf("\n");
    printf("Post-order traversal: ");
    postorderTraversal(root);
    printf("\n");
    return 0;
}
```

OUTPUT=>

```
Enter the values for the binary tree:
1
3
5
7
9
11
13
15
```

In-order traversal: 1 3 5 7 9 11 13 15

Pre-order traversal: 1 3 5 7 9 11 13 15

Post-order traversal: 15 13 11 9 7 5 3 1

15) WAP to create BST and print all elements using in order traversing technique. Create user defined function for each operation and call then in main function

Code=>

```
#include <stdio.h>

#include <stdlib.h>

struct node

{

   int data;

   struct node *left;

   struct node *right;

};

struct node *newNode(int data)

{

   struct node *node = (struct node *)malloc(sizeof(struct node));

   node->data = data;

   node->left = node->right = NULL;

   return node;

}


struct node *insert(struct node *root, int data)

{

   if (root == NULL)

   {

      root = newNode(data);

      return root;

   }

   if (data <= root->data)

      root->left = insert(root->left, data);

   else

      root->right = insert(root->right, data);

   return root;
```

```
}
void inOrder(struct node *temp)
{
  if (temp == NULL)
    return;
  inOrder(temp->left);
  printf("%d ", temp->data);
  inOrder(temp->right);
}
int main()
{
  struct node *root = NULL;
  int choice, data;
  while (1)
  {
    printf("\n1. Insert\n2. In-order traversal\n3. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    switch (choice)
    {
    case 1:
      printf("Enter the value to insert: ");
      scanf("%d", &data);
      root = insert(root, data);
      break;
    case 2:
      printf("In-order traversal: ");
      inOrder(root);
      printf("\n");
      break;
    case 3:
    printf("Exit SuccessFully");
      exit(0);
    default:
      printf("Invalid choice!\n");
    }
  }
  return 0;
}
```

OUTPUT=>

```
1. Insert
2. In-order traversal
3. Exit
Enter your choice: 1
Enter the value to insert: 5
1. Insert
2. In-order traversal
3. Exit
Enter your choice: 1
Enter the value to insert: 10
1. Insert
2. In-order traversal
3. Exit
Enter your choice: 1
Enter the value to insert: 15
1. Insert
2. In-order traversal
3. Exit
Enter your choice: 2
In-order traversal: 5 10 15
1. Insert
2. In-order traversal
```

```
3. Exit

Enter your choice: 3

Exit SuccessFully
```