Institute of Computer Technology, Ganpat University

Computer Science & Engineering (CBA/CS/BDA)

Data Structures

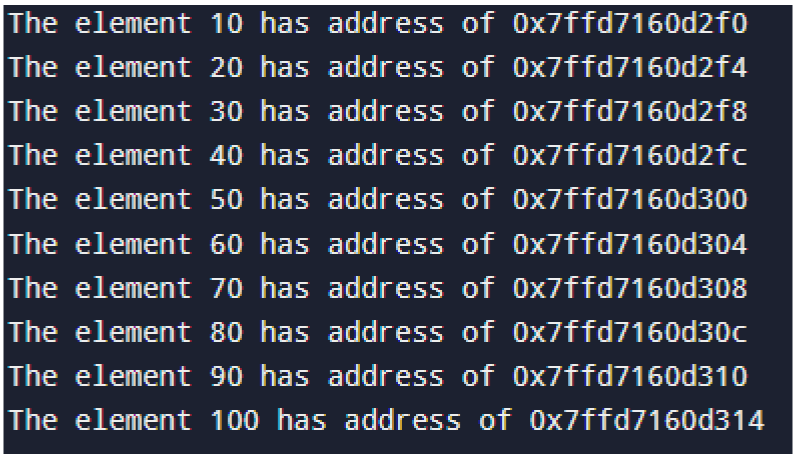
Practical List

1. Write a Program to display elements of single dimensional array with their memory addresses.

Code:-

```
//Write a Program to display elements of single dimensional array with their memory addresses.
#include<stdio.h>
int main(){
    int arr[10]={10,20,30,40,50,60,70,80,90,100};
    int *ptr=arr;
    for (int i=0;i<10;i++)
    {
        printf("The element %d has address of %p\n",arr[i],&arr[i]);
    }
    return 0;
}
```

Output:-



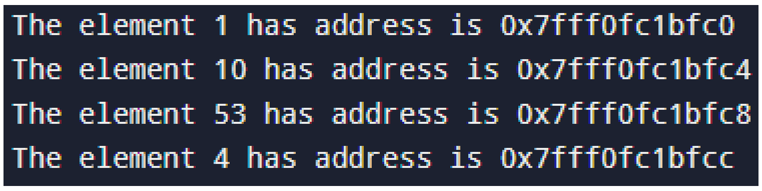
```
The element 10 has address of 0x7ffd7160d2f0
The element 20 has address of 0x7ffd7160d2f4
The element 30 has address of 0x7ffd7160d2f8
The element 40 has address of 0x7ffd7160d2fc
The element 50 has address of 0x7ffd7160d300
The element 60 has address of 0x7ffd7160d304
The element 70 has address of 0x7ffd7160d308
The element 80 has address of 0x7ffd7160d30c
The element 90 has address of 0x7ffd7160d310
The element 100 has address of 0x7ffd7160d314
```

2. Write a Program to display elements of Two-dimensional array with their memory addresses.

Code:-

```
// Write a Program to display elements of Two-dimensional array
with their memory addresses.
#include<stdio.h>
int main(){
   int arr[2][2]={{1,10},{53,4}};
   int *ptr=arr[0];
   for (int i=0;i<2;i++)
   {
      for (int j=0;j<2;j++)
      {
         printf("The element %d has address is
%p\n",arr[i][j],&arr[i][j]);
      }
   }
   return 0;
}
```

Output:-



```
The element 1 has address is 0x7fff0fc1bfc0
The element 10 has address is 0x7fff0fc1bfc4
The element 53 has address is 0x7fff0fc1bfc8
The element 4 has address is 0x7fff0fc1bfcc
```

3. Write a program to implement the concept of Stack and perform following operations on Stack.

- Push
- Pop
- Peep
- Change
- Display

Code:-

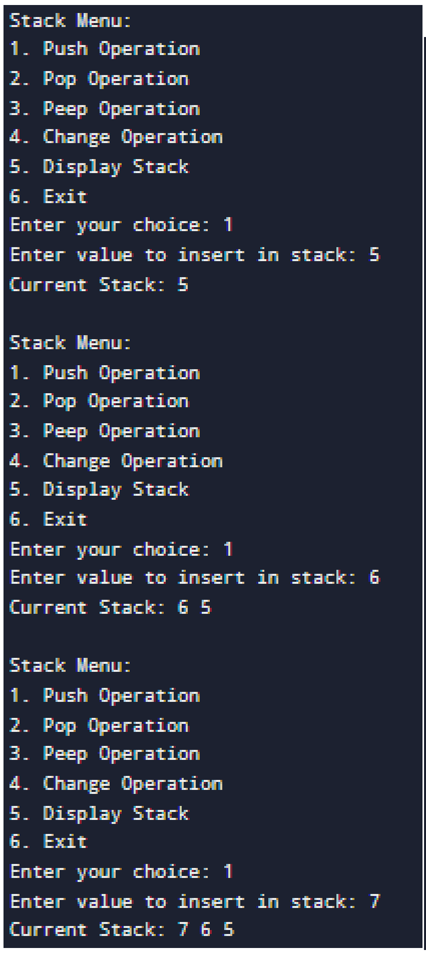
```
#include <stdio.h>
#define SIZE 4

int top = -1;
int Stack[SIZE];
int value, position;

void push() {
   if (top == SIZE - 1) {
      printf("Stack Overflow\n");
   } else {
      top++;
      printf("Enter value to insert in stack: ");
      scanf("%d", &value);
      Stack[top] = value;
   }
}

void pop() {
   if (top == -1) {
      printf("Stack Underflow\n");
   } else {
      value = Stack[top];
      top--;
      printf("The Deleted Value is %d\n", value);
```

```
    }
}

void peep() {
    if (top == -1) {
        printf("Stack Underflow\n");
    } else {
        printf("Enter the position to retrieve the element: ");
        scanf("%d", &position);
        if (position > 0 && position <= top + 1) {
            printf("The element at position %d is %d\n", position, Stack[top -
position + 1]);
        } else {
            printf("Invalid position!\n");
        }
    }
}

void change() {
    if (top == -1) {
        printf("Stack Underflow\n");
    } else {
        printf("Enter the position for the value to change: ");
        scanf("%d", &position);
        if (position > 0 && position <= top + 1) {
            printf("Enter the new value: ");
            scanf("%d", &value);
            Stack[top - position + 1] = value;
            printf("Value at position %d changed to %d\n", position, value);
        } else {
            printf("Invalid position!\n");
        }
    }
}
```

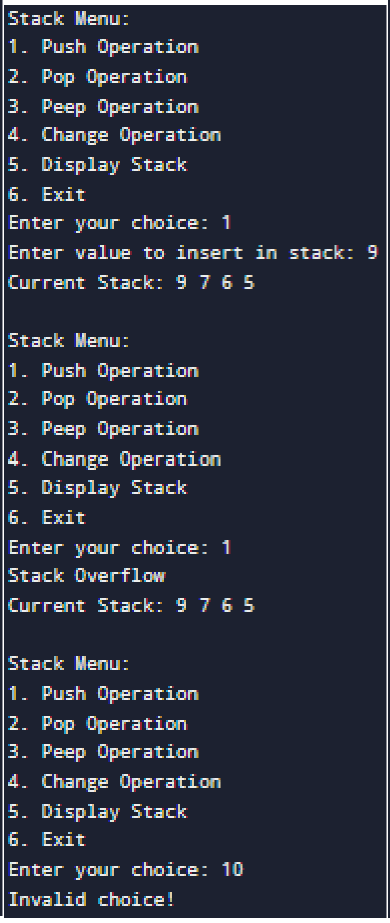
```
void display() {
    if (top == -1) {
        printf("Current Stack is Empty\n");
    } else {
        printf("Current Stack: ");
        for (int i = top; i >= 0; i--) {
            printf("%d ", Stack[i]);
        }
        printf("\n");
    }
}

int main() {
    int choice;
    do {
        printf("\nStack Menu:\n");
        printf("1. Push Operation\n");
        printf("2. Pop Operation\n");
        printf("3. Peep Operation\n");
        printf("4. Change Operation\n");
        printf("5. Display Stack\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                push();
                display();
                break;
            case 2:
                pop();
                display();
                break;
            case 3:
```

```
            peep();
            display();
            break;
        case 4:
            change();
            display();
            break;
        case 5:
            display();
            break;
        case 6:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice!\n");
        }
    } while (choice != 6);

    return 0;
}
```

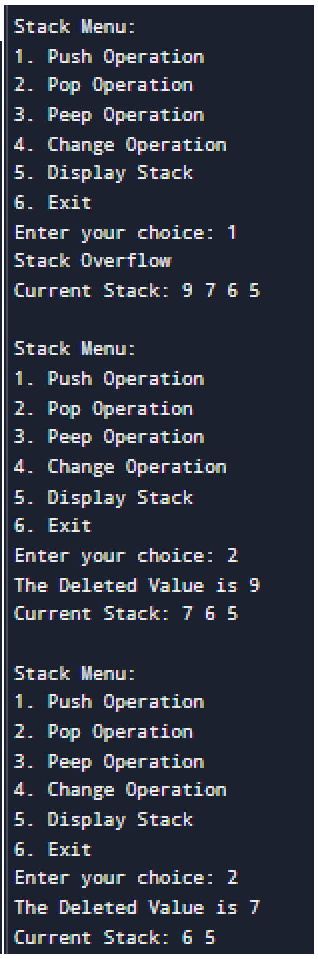
Output:-



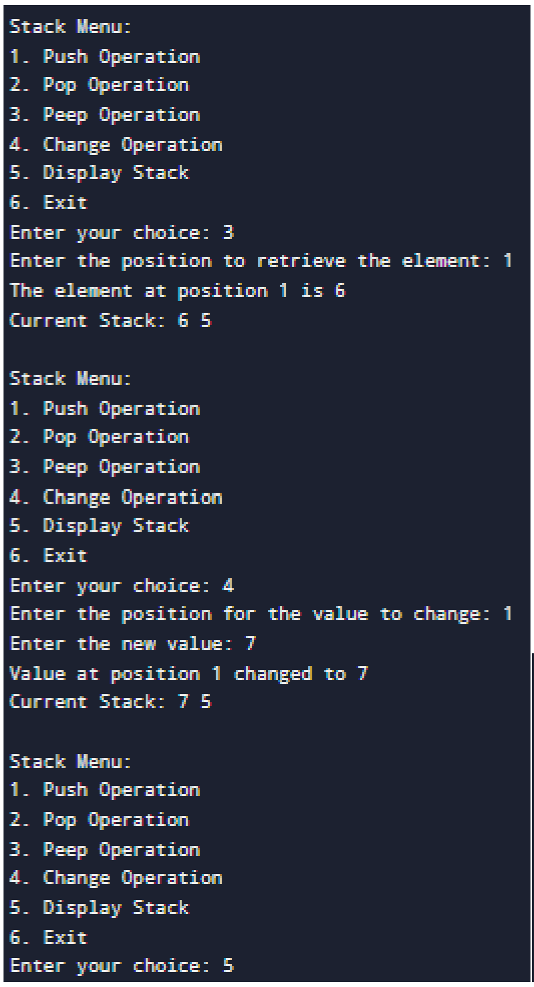
```
Stack Menu:
1. Push Operation
2. Pop Operation
3. Peep Operation
4. Change Operation
5. Display Stack
6. Exit
Enter your choice: 1
Enter value to insert in stack: 5
Current Stack: 5

Stack Menu:
1. Push Operation
2. Pop Operation
3. Peep Operation
4. Change Operation
5. Display Stack
6. Exit
Enter your choice: 1
Enter value to insert in stack: 6
Current Stack: 6 5

Stack Menu:
1. Push Operation
2. Pop Operation
3. Peep Operation
4. Change Operation
5. Display Stack
6. Exit
Enter your choice: 1
Enter value to insert in stack: 7
Current Stack: 7 6 5
```



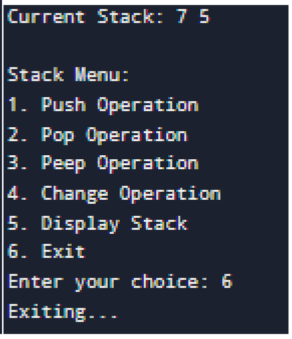
```
Stack Menu:
1. Push Operation
2. Pop Operation
3. Peep Operation
4. Change Operation
5. Display Stack
6. Exit
Enter your choice: 1
Enter value to insert in stack: 9
Current Stack: 9 7 6 5

Stack Menu:
1. Push Operation
2. Pop Operation
3. Peep Operation
4. Change Operation
5. Display Stack
6. Exit
Enter your choice: 1
Stack Overflow
Current Stack: 9 7 6 5

Stack Menu:
1. Push Operation
2. Pop Operation
3. Peep Operation
4. Change Operation
5. Display Stack
6. Exit
Enter your choice: 10
Invalid choice!
```



```
Stack Menu:
1. Push Operation
2. Pop Operation
3. Peep Operation
4. Change Operation
5. Display Stack
6. Exit
Enter your choice: 1
Stack Overflow
Current Stack: 9 7 6 5

Stack Menu:
1. Push Operation
2. Pop Operation
3. Peep Operation
4. Change Operation
5. Display Stack
6. Exit
Enter your choice: 2
The Deleted Value is 9
Current Stack: 7 6 5

Stack Menu:
1. Push Operation
2. Pop Operation
3. Peep Operation
4. Change Operation
5. Display Stack
6. Exit
Enter your choice: 2
The Deleted Value is 7
Current Stack: 6 5
```



```
Stack Menu:
1. Push Operation
2. Pop Operation
3. Peep Operation
4. Change Operation
5. Display Stack
6. Exit
Enter your choice: 3
Enter the position to retrieve the element: 1
The element at position 1 is 6
Current Stack: 6 5

Stack Menu:
1. Push Operation
2. Pop Operation
3. Peep Operation
4. Change Operation
5. Display Stack
6. Exit
Enter your choice: 4
Enter the position for the value to change: 1
Enter the new value: 7
Value at position 1 changed to 7
Current Stack: 7 5

Stack Menu:
1. Push Operation
2. Pop Operation
3. Peep Operation
4. Change Operation
5. Display Stack
6. Exit
Enter your choice: 5
```



```
Current Stack: 7 5

Stack Menu:
1. Push Operation
2. Pop Operation
3. Peep Operation
4. Change Operation
5. Display Stack
6. Exit
Enter your choice: 6
Exiting...
```

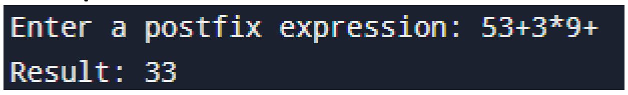
4. Write a program to evaluate a postfix expression using stack.

Code:-

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

#define MAX_SIZE 100

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

void push(int x) {
   if (top >= MAX_SIZE - 1) {
      printf("Stack Overflow\n");
      return;
   }
   top++;
   stack[top] = x;
}

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

```
int isOperand(char c) {
    return (c >= '0' && c <= '9');
}

int evaluatePostfix(char *postfix) {
    int i, op1, op2, result;

    for (i = 0; postfix[i] != '\0'; i++) {
        if (isOperand(postfix[i])) {
            push(postfix[i] - '0');
        } else {
            op2 = pop();
            op1 = pop();
            switch (postfix[i]) {
                case '+':
                    result = op1 + op2;
                    break;
                case '-':
                    result = op1 - op2;
                    break;
                case '*':
                    result = op1 * op2;
                    break;
                case '/':
                    result = op1 / op2;
                    break;
                default:
                    printf("Invalid operator\n");
                    return -1;
            }
            push(result);
        }
    }

    return pop();
}

int main() {
    char postfix[MAX_SIZE];

    printf("Enter a postfix expression: ");
    scanf("%s", postfix);

    int result = evaluatePostfix(postfix);
    printf("Result: %d\n", result);

    return 0;
}
```

Output:-



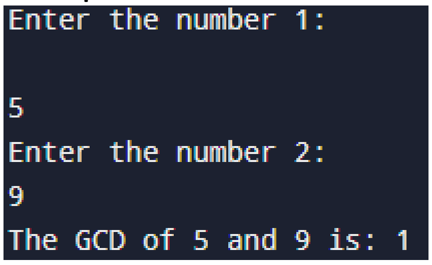
```
Enter a postfix expression: 53+3*9+
Result: 33
```

5. Write a program to find GCD of two numbers using recursion.

Code:-

```
//5. Write a program to find GCD of two numbers using recursion.
#include<stdio.h>
int gcd(int a,int b){
    if (b==0)
    {
        return a;
    }
    return gcd(b, a % b);
}
int main(){
    int num1;
    int num2;
    printf("Enter the number 1:\n");
    scanf("%d",&num1);
    printf("Enter the number 2:\n");
    scanf("%d",&num2);
    int result=gcd(num1,num2);
    printf("The GCD of %d and %d is: %d\n", num1, num2, result);
    return 0;
}
```

Output:-



```
Enter the number 1:

5
Enter the number 2:
9
The GCD of 5 and 9 is: 1
```

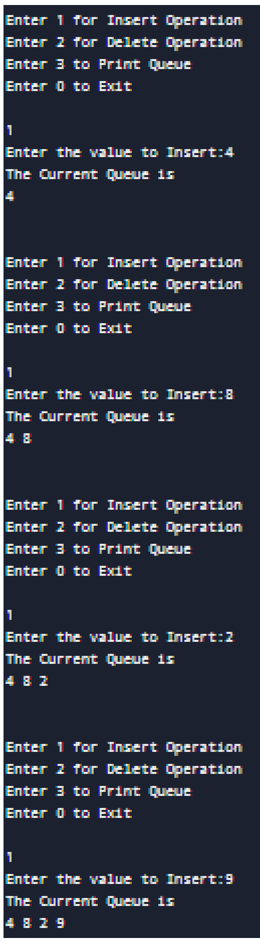
6. Write a program to implement the concept of Simple Queue and perform insert and delete operations on simple queue.

Code:-
```
#include<stdio.h>
#define SIZE 4
int Queue[SIZE];
int front=-1;
int rear=-1;
int value;

void insert(){
    if (rear>=SIZE-1)
    {
        printf("Stack is Full");
    }
    else{
        printf("Enter the value to Insert:");
        scanf("%d",&value);
        if (front==-1)
        {
            front=0;
        }
        rear=rear+1;
        Queue[rear]=value;
    }
}

void delete(){
   if (front==-1)
   {
      printf("Queue is empty");
   }
   else{
      value=Queue[front];
      if (front==rear)
      {
         front=rear=-1;
      }
      else{
         front=front+1;
      }
      printf("The deleted value is %d\n",value);
   }
}

void printQueue(){
   if (front==-1)
   {
      printf("Queue is empty there is no element in Queue");
   }
   else{
      printf("The Current Queue is\n");
      for (int i = front; i <= rear; i++) {
         printf("%d ", Queue[i]);
      }
      printf("\n");
   }
}
```

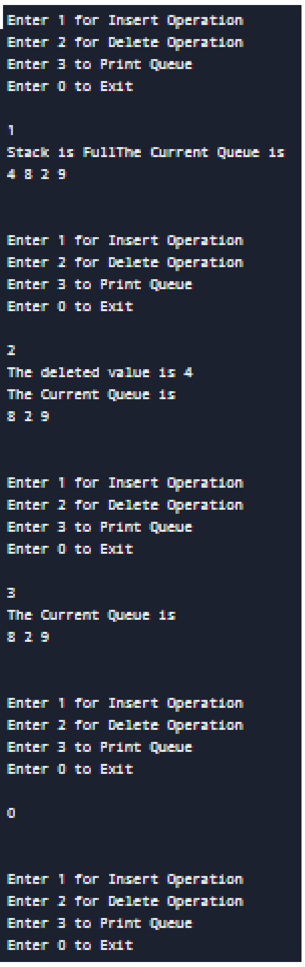
```
int main(){
   int choice;
   while (1){
      printf("\n-------------------\n");
      printf("Enter 1 for Insert Operation\n");
      printf("Enter 2 for Delete Operation\n");
      printf("Enter 3 to Print Queue\n");
      printf("Enter 0 to Exit\n");
      printf("-------------------\n");
      scanf("%d", &choice);
   {
   switch (choice)
   {
   case 1:
      insert();
      printQueue();
      break;

   case 2:
      delete();
      printQueue();
      break;

   case 3:
      printQueue();
      break;

   default:
      break;
   }
```

```
    }
    }
    return 0;
}
```

Output:-



```
Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 3 to Print Queue
Enter 0 to Exit

1
Enter the value to Insert:4
The Current Queue is
4

Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 3 to Print Queue
Enter 0 to Exit

1
Enter the value to Insert:8
The Current Queue is
4 8

Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 3 to Print Queue
Enter 0 to Exit

1
Enter the value to Insert:2
The Current Queue is
4 8 2

Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 3 to Print Queue
Enter 0 to Exit

1
Enter the value to Insert:9
The Current Queue is
4 8 2 9
```



```
Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 3 to Print Queue
Enter 0 to Exit

1
Stack is FullThe Current Queue is
4 8 2 9

Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 3 to Print Queue
Enter 0 to Exit

2
The deleted value is 4
The Current Queue is
8 2 9

Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 3 to Print Queue
Enter 0 to Exit

3
The Current Queue is
8 2 9

Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 3 to Print Queue
Enter 0 to Exit

0

Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 3 to Print Queue
Enter 0 to Exit
```

6.Write a program to implement the concept of Circular Queue and perform insert and delete operations on circular queue.

Code:-

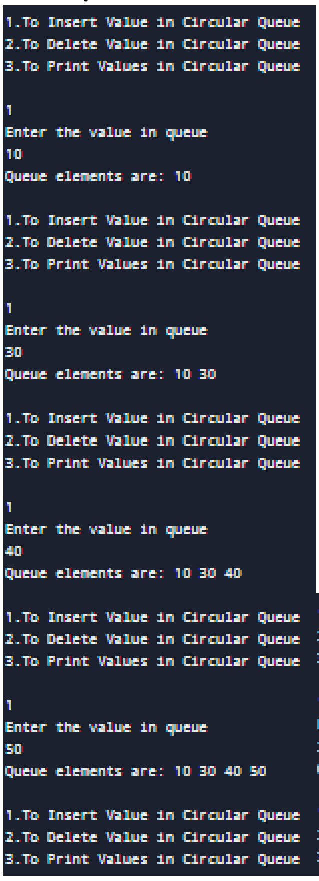
```
#include<stdio.h>
#define SIZE 4
int CircularQueue[SIZE];
int front=-1,rear=-1;
```

```
void insert(){
    int value;
    if ((front==0 && rear==SIZE-1) || (rear == (front - 1) % (SIZE - 1)))
    {
        printf("Queue is full there is no space in queue\n");
    }
    else{
        printf("Enter the value in queue\n");
        scanf("%d",&value);
        if (front==-1)
        {
            front=rear=0;
        }
        else if (rear==SIZE-1 && front!=0)
        {
            rear=0;
        }
        else{
            rear=rear+1;
        }
        CircularQueue[rear]=value;
    }
}
void delete(){
    if (front==-1)
    {
        printf("Queue is empty");
    }else {
        printf("Deleted element: %d\n", CircularQueue[front]);
        if (front == rear) {
            front = rear = -1;
        } else if (front == SIZE - 1) {
            front = 0;
        } else {
            front++;
        }
    }
}
void printqueue(){
    if (front == -1) {
        printf("Queue is Empty\n");
    } else {
        printf("Queue elements are: ");
        if (rear >= front) {
            for (int i = front; i <= rear; i++) {
                printf("%d ", CircularQueue[i]);
            }
        } else {
            for (int i = front; i < SIZE; i++) {
                printf("%d ", CircularQueue[i]);
            }
            for (int i = 0; i <= rear; i++) {
                printf("%d ", CircularQueue[i]);
            }
        }
        printf("\n");
    }
}
```

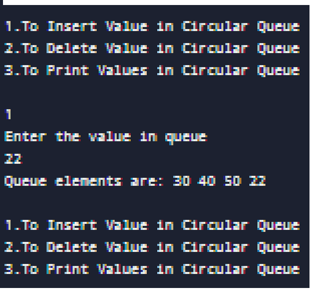
```
int main(){
   int choice;
   while (1)
   {
      printf("---------------\n");
      printf("1.To Insert Value in Circular Queue\n");
      printf("2.To Delete Value in Circular Queue\n");
      printf("3.To Print Values in Circular Queue\n");
      printf("---------------\n");
      scanf("%d",&choice);

      switch (choice)
      {
      case 1:
         insert();
         printqueue();
         break;

      case 2:
         delete();
         printqueue();
         break;

      case 3:
         printqueue();
         break;
      default:
         break;
      }
   }
```

```
    return 0;
}
```

Output:-



```
1.To Insert Value in Circular Queue
2.To Delete Value in Circular Queue
3.To Print Values in Circular Queue

1
Enter the value in queue
10
Queue elements are: 10

1.To Insert Value in Circular Queue
2.To Delete Value in Circular Queue
3.To Print Values in Circular Queue

1
Enter the value in queue
30
Queue elements are: 10 30

1.To Insert Value in Circular Queue
2.To Delete Value in Circular Queue
3.To Print Values in Circular Queue

1
Enter the value in queue
40
Queue elements are: 10 30 40

1.To Insert Value in Circular Queue
2.To Delete Value in Circular Queue
3.To Print Values in Circular Queue

1
Enter the value in queue
50
Queue elements are: 10 30 40 50

1.To Insert Value in Circular Queue
2.To Delete Value in Circular Queue
3.To Print Values in Circular Queue
```



```
1.To Insert Value in Circular Queue
2.To Delete Value in Circular Queue
3.To Print Values in Circular Queue

1
Enter the value in queue
22
Queue elements are: 30 40 50 22

1.To Insert Value in Circular Queue
2.To Delete Value in Circular Queue
3.To Print Values in Circular Queue
```

8. Write a program to implement the concept of Deque and perform insert and delete operations on deque.

Code:-

```
#include <stdio.h>
#define SIZE 4

int rear = -1;
int front = -1;
int Queue[SIZE];
int value;

void insertRear(){
   if ((rear + 1) % SIZE == front)
   {
      printf("Queue is full you can't insert the value\n");
   }
   else{
      if (front==-1)
      {
         front=0;
         rear=0;
      }
      else{
         rear=(rear+1)%SIZE;
      }
      printf("Enter the value to insert in queue\n");
      scanf("%d",&value);
      Queue[rear]=value;
   }
}

void insertFront(){
   if ((front == 0 && rear == SIZE - 1) || (front == rear + 1)) {
      printf("Queue Overflow\n");
      return;
   } else {
      if (front == -1) {
         front = 0;
         rear = 0;
      } else if (front == 0) {
         front = SIZE - 1;
      } else {
         front = front - 1;
      }
      printf("Enter the value to insert in queue:\n");
      scanf("%d", &value);
      Queue[front] = value;
   }
}

void deleteFront(){
   if (front==-1)
   {
      printf("Queue is Empty\n");
   }
   else{
      value=Queue[front];
      printf("Deleted Value is %d\n",value);
      if (front==rear)
      {
         front=rear=-1;
      }
      else{
         front = (front + 1) % SIZE;
      }
   }
}
```

```
void printQueue() {
   if (front == -1) {
      printf("Queue is empty.\n");
   } else {
      printf("Queue: ");
      int i = front;
      while (i != rear) {
         printf("%d ", Queue[i]);
         i = (i + 1) % SIZE;
      }
      printf("%d\n", Queue[rear]);
   }
}

int main(){
   int choice;
   while (1)
   {
      printf("------------\n");
      printf("1.Insert the value at rear pointer\n");
      printf("2.Insert the value at front pointer\n");
      printf("3.Delete the value at front pointer\n");
      printf("4.Print the Queue\n");
      printf("-------------\n");
      scanf("%d",&choice);

      switch (choice){
      case 1:
         insertRear();
         printQueue();
         break;

      case 2:
         insertFront();
         printQueue();
         break;

      case 3:
         deleteFront();
         printQueue();
         break;

      case 4:
         printQueue();
         break;

      default:
         break;
      }
   }
   return 0;
}
```

Output:-

```
1.Insert the value at rear pointer
2.Insert the value at front pointer
3.Delete the value at front pointer
4.Print the Queue

1
Enter the value to insert in queue
20
Queue: 20

1.Insert the value at rear pointer
2.Insert the value at front pointer
3.Delete the value at front pointer
4.Print the Queue

1
Enter the value to insert in queue
30
Queue: 20 30

1.Insert the value at rear pointer
2.Insert the value at front pointer
3.Delete the value at front pointer
4.Print the Queue

1
Enter the value to insert in queue
11
Queue: 20 30 11

1.Insert the value at rear pointer
2.Insert the value at front pointer
3.Delete the value at front pointer
4.Print the Queue

1
Enter the value to insert in queue
22
Queue: 20 30 11 22

1.Insert the value at rear pointer
2.Insert the value at front pointer
3.Delete the value at front pointer
4.Print the Queue

1
Queue is full you can't insert the value
```

```
Queue: 20 30 11 22

1.Insert the value at rear pointer
2.Insert the value at front pointer
3.Delete the value at front pointer
4.Print the Queue

2
Queue Overflow
Queue: 20 30 11 22

1.Insert the value at rear pointer
2.Insert the value at front pointer
3.Delete the value at front pointer
4.Print the Queue

4
Queue: 20 30 11 22

1.Insert the value at rear pointer
2.Insert the value at front pointer
3.Delete the value at front pointer
4.Print the Queue

3
Deleted Value is 20
Queue: 30 11 22

1.Insert the value at rear pointer
2.Insert the value at front pointer
3.Delete the value at front pointer
4.Print the Queue

3
Deleted Value is 30
Queue: 11 22

1.Insert the value at rear pointer
2.Insert the value at front pointer
3.Delete the value at front pointer
4.Print the Queue

2
Enter the value to insert in queue:
11
Queue: 11 11 22

1.Insert the value at rear pointer
2.Insert the value at front pointer
3.Delete the value at front pointer
```

```
4.Print the Queue

1
Enter the value to insert in queue
44
Queue: 11 11 22 44

1.Insert the value at rear pointer
2.Insert the value at front pointer
3.Delete the value at front pointer
4.Print the Queue

4
Queue: 11 11 22 44

1.Insert the value at rear pointer
2.Insert the value at front pointer
3.Delete the value at front pointer
4.Print the Queue
```

9. Write a program to implement the concept of Singly Linked List. Perform following operations:

- Insert First
- Insert Last
- Insert by Position
- Delete First
- Delete Last
- Delete by Position
- Display the List

Code:-

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

struct node {
   int info;
```

```
    struct node *link;
} *first = NULL;

void leftInsert(int value) {
    struct node *newnode = (struct node *)malloc(sizeof(struct node));
    if (newnode == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    newnode->info = value;
    newnode->link = first;
    first = newnode;

    printf("Node inserted at the front with value %d\n", value);
}

void RightInsert(int value) {
    struct node *newnode = (struct node *)malloc(sizeof(struct node));
    if (newnode == NULL) {
        printf("Memory allocation Failed!\n");
        return;
    }

    newnode->info = value;
    newnode->link = NULL;

    if (first == NULL) {
        first = newnode;
    } else {
        struct node *temp = first;
        while (temp->link != NULL) {
            temp = temp->link;
        }
        temp->link = newnode;
    }
    printf("Node inserted at the end with value %d\n", value);
}

void DeleteLeft() {
    if (first == NULL) {
        printf("List is empty, nothing to delete\n");
        return;
    }

    struct node *temp = first;
    first = first->link;
    printf("Node with value %d deleted from the front\n", temp->info);
    free(temp);
}

void DeleteRight() {
    if (first == NULL) {
        printf("List is empty, nothing to delete\n");
        return;
    }

    if (first->link == NULL) {
        printf("Node with value %d deleted from the end\n", first->info);
        free(first);
        first = NULL;
        return;
    }

    struct node *temp = first;
    while (temp->link->link != NULL) {  // Find the second last node
        temp = temp->link;
    }

    struct node *lastNode = temp->link;
    printf("Node with value %d deleted from the end\n", lastNode-
>info);
   free(lastNode);
   temp->link = NULL;
}

void InsertAtPosition(int value, int position) {
   struct node *newnode = (struct node *)malloc(sizeof(struct node));
   if (newnode == NULL) {
      printf("Memory allocation failed\n");
      return;
   }
   newnode->info = value;

   if (position == 1) {
      newnode->link = first;
      first = newnode;
      printf("Node inserted at position %d with value %d\n", position,
value);
      return;
   }

   struct node *temp = first;
   for (int i = 1; i < position - 1 && temp != NULL; i++) {
      temp = temp->link;
   }

   if (temp == NULL) {
      printf("Position %d is out of bounds\n", position);
      free(newnode);
   } else {
      newnode->link = temp->link;
      temp->link = newnode;
      printf("Node inserted at position %d with value %d\n", position,
value);
   }
```

```
}

void DeleteAtPosition(int position) {
    if (first == NULL) {
        printf("List is empty, nothing to delete\n");
        return;
    }

    if (position == 1) {
        struct node *temp = first;
        first = first->link;
        printf("Node with value %d deleted from position %d\n", temp-
>info, position);
        free(temp);
        return;
    }

    struct node *temp = first;
    for (int i = 1; i < position - 1 && temp->link != NULL; i++) {
        temp = temp->link;
    }

    if (temp->link == NULL) {
        printf("Position %d is out of bounds\n", position);
    } else {
        struct node *nodeToDelete = temp->link;
        temp->link = nodeToDelete->link;
        printf("Node with value %d deleted from position %d\n",
nodeToDelete->info, position);
        free(nodeToDelete);
    }
}

void displayList() {
    struct node *temp = first;
    if (first == NULL) {
        printf("List is empty\n");
        return;
    }

    printf("The linked list is: ");
    while (temp != NULL) {
        printf("%d -> ", temp->info);
        temp = temp->link;
    }
    printf("NULL\n");
}

int main() {
    int choice;
    do {
        printf("\nMenu:\n");
        printf("1. Insert Left \n");
        printf("2. Insert Right \n");
        printf("3. Delete Left Node\n");
        printf("4. Delete Right Node\n");
        printf("5. Insert at Position\n");
        printf("6. Delete at Position\n");
        printf("7. Display List\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        int value, position;
        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &value);
                leftInsert(value);
                break;
```

```
case 2:
   printf("Enter the value to insert: ");
   scanf("%d", &value);
   RightInsert(value);
   break;

case 3:
   DeleteLeft();
   break;

case 4:
   DeleteRight();
   break;

case 5:
   printf("Enter the value to insert: ");
   scanf("%d", &value);
   printf("Enter the position: ");
   scanf("%d", &position);
   InsertAtPosition(value, position);
   break;

case 6:
   printf("Enter the position: ");
   scanf("%d", &position);
   DeleteAtPosition(position);
   break;

case 7:
   displayList();
   break;

case 8:
   printf("Exiting program...\n");
```

```
            break;

        default:
            printf("Invalid choice! Please enter again.\n");
        }
    } while (choice != 8);

    return 0;
}
```

Output:-

```
Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Insert at Position
6. Delete at Position
7. Display List
8. Exit
Enter your choice: 1
Enter the value to insert: 11
Node inserted at the front with value 11

Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Insert at Position
6. Delete at Position
7. Display List
8. Exit
Enter your choice: 2
Enter the value to insert: 22
Node inserted at the end with value 22

Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Insert at Position
6. Delete at Position
7. Display List
8. Exit
```

```
Enter your choice: 1
Enter the value to insert: 33
Node inserted at the front with value 33

Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Insert at Position
6. Delete at Position
7. Display List
8. Exit
Enter your choice: 2
Enter the value to insert: 44
Node inserted at the end with value 44

Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Insert at Position
6. Delete at Position
7. Display List
8. Exit
Enter your choice: 3
Node with value 33 deleted from the front

Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Insert at Position
6. Delete at Position
7. Display List
8. Exit
Enter your choice: 4
Node with value 44 deleted from the end

Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Insert at Position
6. Delete at Position
7. Display List
8. Exit
Enter your choice: 5
Enter the value to insert: 2
Enter the position: 2
Node inserted at position 2 with value 2

Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Insert at Position
6. Delete at Position
7. Display List
8. Exit
Enter your choice: 6
Enter the position: 2
Node with value 2 deleted from position 2

Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Insert at Position
6. Delete at Position
7. Display List
8. Exit
Enter your choice: 7
The linked list is: 11 -> 22 -> NULL

Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Insert at Position
6. Delete at Position
7. Display List
8. Exit
Enter your choice: 8
```

10. Write a program to implement the concept of Singly Circular Linked List. Perform following operations:

- Insert First
- Insert Last

- Delete First
- Delete Last
- Display the List

Code:-

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

// Define the structure for a node
struct node {
   int info;
   struct node *link;
} *first = NULL;

void leftInsert(int value) {
   struct node *newnode = (struct node *)malloc(sizeof(struct node));
   if (newnode == NULL) {
      printf("Memory allocation failed\n");
      return;
   }

   newnode->info = value;

   if (first == NULL) {
      first = newnode;
      newnode->link = first; // Point to itself in a circular manner
   } else {
      struct node *temp = first;
      while (temp->link != first) {
         temp = temp->link;
      }
      newnode->link = first; // New node points to the first node
      temp->link = newnode;  // Last node points to the new node
      first = newnode;       // Update first to the new node
   }
```

```
    printf("Node inserted at the front with value %d\n", value);
}

void RightInsert(int value) {
    struct node *newnode = (struct node *)malloc(sizeof(struct node));
    if (newnode == NULL) {
        printf("Memory allocation failed\n");
        return;
    }

    newnode->info = value;

    if (first == NULL) {
        first = newnode;
        newnode->link = first; // Point to itself in a circular manner
    } else {
        struct node *temp = first;
        while (temp->link != first) {
            temp = temp->link;
        }
        temp->link = newnode;  // Last node points to the new node
        newnode->link = first; // New node points back to the first node
    }

    printf("Node inserted at the end with value %d\n", value);
}

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

    struct node *temp = first;
    if (first->link == first) { // Only one node in the list
        first = NULL;
    } else {
        struct node *last = first;
        while (last->link != first) {
            last = last->link;
        }
        first = first->link; // Update first to the second node
        last->link = first;  // Update last node to point to the new first
    }

    printf("Node with value %d deleted from the front\n", temp->info);
    free(temp);
}

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

    struct node *temp = first;

    if (first->link == first) { // Only one node in the list
        first = NULL;
    } else {
        struct node *prev = NULL;
        while (temp->link != first) {
            prev = temp;
            temp = temp->link;
        }
        prev->link = first; // Update the second last node to point to first
    }
    printf("Node with value %d deleted from the end\n", temp->info);
    free(temp);
}

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

    struct node *temp = first;
    printf("The circular linked list is: ");
    do {
        printf("%d -> ", temp->info);
        temp = temp->link;
    } while (temp != first);
    printf("...\n");
}

int main() {
    int choice, value;

    do {
        printf("\nMenu:\n");
        printf("1. Circular Insert Left\n");
        printf("2. Circular Insert Right\n");
        printf("3. Delete Left Node\n");
        printf("4. Delete Right Node\n");
        printf("5. Display List\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &value);
                leftInsert(value);
                break;

            case 2:
                printf("Enter the value to insert: ");
                scanf("%d", &value);
                RightInsert(value);
                break;

            case 3:
                DeleteLeft();
                break;

            case 4:
                DeleteRight();
                break;

            case 5:
                displayList();
                break;

            case 6:
                printf("Exiting program...\n");
                break;

            default:
                printf("Invalid choice! Please enter again.\n");
        }
    } while (choice != 6);

    return 0;
}
```

Output:-

```
Menu:
1. Circular Insert Left
2. Circular Insert Right
3. Delete Left Node
4. Delete Right Node
5. Display List
6. Exit
Enter your choice: 1
Enter the value to insert: 1
Node inserted at the front with value 1

Menu:
1. Circular Insert Left
2. Circular Insert Right
3. Delete Left Node
4. Delete Right Node
5. Display List
6. Exit
Enter your choice: 2
Enter the value to insert: 2
Node inserted at the end with value 2

Menu:
1. Circular Insert Left
2. Circular Insert Right
3. Delete Left Node
4. Delete Right Node
5. Display List
6. Exit
Enter your choice: 1
Enter the value to insert: 4
Node inserted at the front with value 4

Menu:
1. Circular Insert Left
2. Circular Insert Right
3. Delete Left Node
4. Delete Right Node
5. Display List
6. Exit
Enter your choice: 2
Enter the value to insert: 5
Node inserted at the end with value 5

Menu:
1. Circular Insert Left
2. Circular Insert Right
3. Delete Left Node
4. Delete Right Node
```

```
5. Display List
6. Exit
Enter your choice: 1
Enter the value to insert: 33
Node inserted at the front with value 33

Menu:
1. Circular Insert Left
2. Circular Insert Right
3. Delete Left Node
4. Delete Right Node
5. Display List
6. Exit
Enter your choice: 1
Enter the value to insert: 55
Node inserted at the front with value 55

Menu:
1. Circular Insert Left
2. Circular Insert Right
3. Delete Left Node
4. Delete Right Node
5. Display List
6. Exit
Enter your choice: 3
Node with value 55 deleted from the front

Menu:
1. Circular Insert Left
2. Circular Insert Right
3. Delete Left Node
4. Delete Right Node
5. Display List
6. Exit
Enter your choice: 4
Node with value 5 deleted from the end

Menu:
1. Circular Insert Left
2. Circular Insert Right
3. Delete Left Node
4. Delete Right Node
5. Display List
6. Exit
Enter your choice: 5
The circular linked list is: 33 -> 4 -> 1 -> 2 -> ...

Menu:
1. Circular Insert Left
2. Circular Insert Right
3. Delete Left Node
```

```
4. Delete Right Node
5. Display List
6. Exit
Enter your choice: 6
Exiting program...
```

11. Write a program to implement the concept of Doubly Linked List. Perform following operations:
• Insert First
• Insert Last
• Delete First
• Delete Last
• Display the List

Code:-
```
#include <stdio.h>
#include <stdlib.h>

struct node {
   int info;
   struct node *next;
   struct node *prevs;
} *first = NULL;

void leftInsert(int value) {
   struct node *newnode = (struct node *)malloc(sizeof(struct node));
   if (newnode == NULL) {
      printf("Memory allocation failed\n");
      return;
   }
   newnode->info = value;
   newnode->next = first;
   newnode->prevs = NULL;

   if (first != NULL) {
      first->prevs = newnode;
```

```
    }
    first = newnode;

    printf("Node inserted at the front with value %d\n", value);
}

void RightInsert(int value) {
    struct node *newnode = (struct node *)malloc(sizeof(struct node));
    if (newnode == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    newnode->info = value;
    newnode->next = NULL;

    if (first == NULL) {
        newnode->prevs = NULL;
        first = newnode;
    } else {
        struct node *temp = first;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newnode;
        newnode->prevs = temp;
    }

    printf("Node inserted at the end with value %d\n", value);
}

void DeleteLeft() {
    if (first == NULL) {
        printf("List is empty, nothing to delete\n");
        return;
    }
    struct node *temp = first;
    first = first->next;

    if (first != NULL) {
        first->prevs = NULL;
    }

    printf("Node with value %d deleted from the front\n", temp->info);
    free(temp);
}

void DeleteRight() {
    if (first == NULL) {
        printf("List is empty, nothing to delete\n");
        return;
    }

    struct node *temp = first;

    if (first->next == NULL) { // Only one node
        first = NULL;
    } else {
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->prevs->next = NULL;
    }

    printf("Node with value %d deleted from the end\n", temp->info);
    free(temp);
}

void displayList() {
    struct node *temp = first;
    if (first == NULL) {
        printf("List is empty\n");
        return;
    }

    printf("The doubly linked list is: ");
    while (temp != NULL) {
        printf("%d -> ", temp->info);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    int choice, value;

    do {
        printf("\nMenu:\n");
        printf("1. Insert Left\n");
        printf("2. Insert Right\n");
        printf("3. Delete Left Node\n");
        printf("4. Delete Right Node\n");
        printf("5. Display List\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &value);
                leftInsert(value);
                break;
```

```
         case 2:
            printf("Enter the value to insert: ");
            scanf("%d", &value);
            RightInsert(value);
            break;

         case 3:
            DeleteLeft();
            break;

         case 4:
            DeleteRight();
            break;

         case 5:
            displayList();
            break;

         case 6:
            printf("Exiting program...\n");
            break;

         default:
            printf("Invalid choice! Please enter again.\n");
      }
   } while (choice != 6);

   return 0;
}
```

Output:-

```
Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Display List
6. Exit
Enter your choice: 1
Enter the value to insert: 22
Node inserted at the front with value 22

Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Display List
6. Exit
Enter your choice: 2
Enter the value to insert: 11
Node inserted at the end with value 11

Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Display List
6. Exit
Enter your choice: 2
Enter the value to insert: 11
Node inserted at the end with value 11

Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Display List
6. Exit
Enter your choice: 1
Enter the value to insert: 44
```

```
Node inserted at the front with value 44

Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Display List
6. Exit
Enter your choice: 3
Node with value 44 deleted from the front

Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Display List
6. Exit
Enter your choice: 4
Node with value 11 deleted from the end

Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Display List
6. Exit
Enter your choice: 5
The doubly linked list is: 22 -> 11 -> NULL

Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Display List
6. Exit
Enter your choice: 6
Exiting program...
```

12. Write a program to implement the concept of Doubly Circular Linked List. Perform following operations:

- Insert First
- Insert Last
- Delete First

- Delete Last
- Display the List

Code:-

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

struct node {
   int info;
   struct node *next;
   struct node *prevs;
} *first = NULL;

void leftInsert(int value) {
   struct node *newnode = (struct node *)malloc(sizeof(struct node));
   if (newnode == NULL) {
      printf("Memory allocation failed\n");
      return;
   }
   newnode->info = value;

   if (first == NULL) { // First node in the list
      newnode->next = newnode;
      newnode->prevs = newnode;
      first = newnode;
   } else { // Insert before the first node
      struct node *last = first->prevs;
      newnode->next = first;
      newnode->prevs = last;
      last->next = newnode;
      first->prevs = newnode;
      first = newnode;
   }
```

```
    printf("Node inserted at the front with value %d\n", value);
}

void RightInsert(int value) {
    struct node *newnode = (struct node *)malloc(sizeof(struct node));
    if (newnode == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    newnode->info = value;

    if (first == NULL) { // First node in the list
        newnode->next = newnode;
        newnode->prevs = newnode;
        first = newnode;
    } else { // Insert after the last node
        struct node *last = first->prevs;
        newnode->next = first;
        newnode->prevs = last;
        last->next = newnode;
        first->prevs = newnode;
    }

    printf("Node inserted at the end with value %d\n", value);
}

void DeleteLeft() {
    if (first == NULL) {
        printf("List is empty, nothing to delete\n");
        return;
    }

    struct node *temp = first;
    if (first->next == first) { // Only one node
        first = NULL;
    } else {
        struct node *last = first->prevs;
        first = first->next;
        first->prevs = last;
        last->next = first;
    }

    printf("Node with value %d deleted from the front\n", temp->info);
    free(temp);
}

void DeleteRight() {
    if (first == NULL) {
        printf("List is empty, nothing to delete\n");
        return;
    }

    struct node *temp = first->prevs;

    if (first->next == first) { // Only one node
        first = NULL;
    } else {
        struct node *secondLast = temp->prevs;
        secondLast->next = first;
        first->prevs = secondLast;
    }

    printf("Node with value %d deleted from the end\n", temp->info);
    free(temp);
}

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

    struct node *temp = first;
    printf("The circular linked list is: ");
    do {
        printf("%d -> ", temp->info);
        temp = temp->next;
    } while (temp != first);
    printf("(back to start)\n");
}

int main() {
    int choice, value;

    do {
        printf("\nMenu:\n");
        printf("1. Insert Left\n");
        printf("2. Insert Right\n");
        printf("3. Delete Left Node\n");
        printf("4. Delete Right Node\n");
        printf("5. Display List\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &value);
                leftInsert(value);
                displayList();
                break;
            case 2:
                printf("Enter the value to insert: ");
                scanf("%d", &value);
                RightInsert(value);
                displayList();
                break;

            case 3:
                DeleteLeft();
                displayList();
                break;

            case 4:
                DeleteRight();
                displayList();
                break;

            case 5:
                displayList();
                break;

            case 6:
                printf("Exiting program...\n");
                break;

            default:
                printf("Invalid choice! Please enter again.\n");
        }
    } while (choice != 6);

    return 0;
}
```

Output:-

```
Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Display List
6. Exit
Enter your choice: 1
Enter the value to insert: 22
Node inserted at the front with value 22
The circular linked list is: 22 -> (back to start)

Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Display List
6. Exit
Enter your choice: 2
Enter the value to insert: 33
Node inserted at the end with value 33
The circular linked list is: 22 -> 33 -> (back to start)

Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Display List
6. Exit
Enter your choice: 1
Enter the value to insert: 44
Node inserted at the front with value 44
The circular linked list is: 44 -> 22 -> 33 -> (back to start)

Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Display List
```

```
6. Exit
Enter your choice: 2
Enter the value to insert: 01
Node inserted at the end with value 1
The circular linked list is: 44 -> 22 -> 33 -> 1 -> (back to start)

Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Display List
6. Exit
Enter your choice: 3
Node with value 44 deleted from the front
The circular linked list is: 22 -> 33 -> 1 -> (back to start)

Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Display List
6. Exit
Enter your choice: 4
Node with value 1 deleted from the end
The circular linked list is: 22 -> 33 -> (back to start)

Menu:
1. Insert Left
2. Insert Right
3. Delete Left Node
4. Delete Right Node
5. Display List
6. Exit
Enter your choice: 5
The circular linked list is: 22 -> 33 -> (back to start)
```

13. Write a program to implement the concept of Binary Search Tree. Perform following operations on it:

- Insert Elements
- Inorder Travelling
- Preorder Travelling
- Postorder Travelling

Code:-

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

```
#include <stdlib.h>

// Define the structure for a tree node
struct Node {
   int data;
   struct Node *left;
   struct Node *right;
};

// Function to create a new node
struct Node* createNode(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   if (newNode == NULL) {
      printf("Memory allocation failed\n");
      return NULL;
   }
   newNode->data = value;
   newNode->left = NULL;
   newNode->right = NULL;
   return newNode;
}

// Function to insert an element into the BST
struct Node* insertElement(struct Node* root, int value) {
   if (root == NULL) {
      return createNode(value);
   }
   if (value < root->data) {
      root->left = insertElement(root->left, value);
   } else if (value > root->data) {
      root->right = insertElement(root->right, value);
   } else {
      printf("Duplicate values are not allowed in BST\n");
   }
   return root;
}

// Inorder traversal
void inorderTraversal(struct Node* root) {
   if (root == NULL) return;
```

```
  inorderTraversal(root->left);
  printf("%d ", root->data);
  inorderTraversal(root->right);
}

// Preorder traversal
void preorderTraversal(struct Node* root) {
  if (root == NULL) return;
  printf("%d ", root->data);
  preorderTraversal(root->left);
  preorderTraversal(root->right);
}

// Postorder traversal
void postorderTraversal(struct Node* root) {
  if (root == NULL) return;
  postorderTraversal(root->left);
  postorderTraversal(root->right);
  printf("%d ", root->data);
}

// Main function
int main() {
  struct Node* root = NULL;
  int choice, value;

  do {
    printf("\nMenu:\n");
    printf("1. Insert Element\n");
    printf("2. Inorder Traversal\n");
    printf("3. Preorder Traversal\n");
    printf("4. Postorder Traversal\n");
    printf("5. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
      case 1:
        printf("Enter the value to insert: ");
        scanf("%d", &value);
```

```
            root = insertElement(root, value);
            printf("Element %d inserted successfully.\n", value);
            break;

          case 2:
            printf("Inorder Traversal: ");
            inorderTraversal(root);
            printf("\n");
            break;

          case 3:
            printf("Preorder Traversal: ");
            preorderTraversal(root);
            printf("\n");
            break;

          case 4:
            printf("Postorder Traversal: ");
            postorderTraversal(root);
            printf("\n");
            break;

          case 5:
            printf("Exiting program...\n");
            break;

          default:
            printf("Invalid choice! Please enter again.\n");
        }
    } while (choice != 5);

    return 0;
}
```

Output:-

```
Menu:
1. Insert Element
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 4
Element 4 inserted successfully.

Menu:
1. Insert Element
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 7
Element 7 inserted successfully.

Menu:
1. Insert Element
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Exit
Enter your choice: 2
Inorder Traversal: 4 7

Menu:
1. Insert Element
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Exit
Enter your choice: 3
Preorder Traversal: 4 7

Menu:
1. Insert Element
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Exit
Enter your choice: 4
Postorder Traversal: 7 4

Menu:
1. Insert Element
```

```
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Exit
Enter your choice: 5
Exiting program...
```

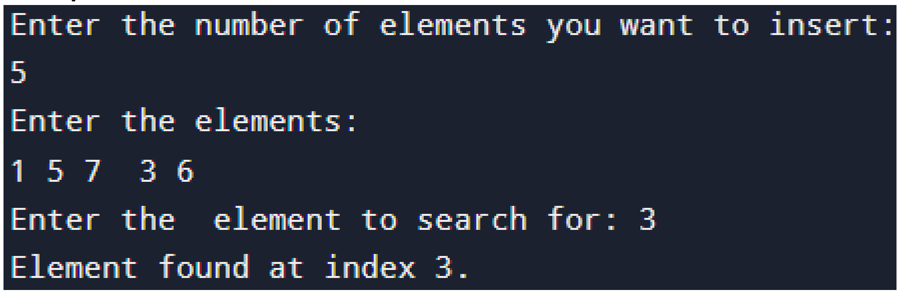
14. Write a program to search element from array or list using sequential search. On successful search print the position of the element along with proper message.

Code:-

```
#include<stdio.h>

int sequentialsearch(int arr[], int size, int valuetofind){
   for (int i = 0; i < size; i++) {
      if (arr[i] == valuetofind) {
         return i;
      }
   }
   return -1;
}
int main(){
   int size;
   int valuetofind;

   printf("Enter the number of elements you want to insert:\n");
   scanf("%d",&size);

   int arr[size];
   printf("Enter the elements:\n");
   for (int i = 0; i < size; i++) {
      scanf("%d", &arr[i]);
   }

   printf("Enter the  element to search for: ");
   scanf("%d", &valuetofind);
```

```
    int result=sequentialsearch(arr,size,valuetofind);

    if (result != -1) {
        printf("Element found at index %d.\n", result);
    } else {
        printf("Element not found in the array.\n");
    }
}
```

Output:-



```
Enter the number of elements you want to insert:
5
Enter the elements:
1 5 7  3 6
Enter the  element to search for: 3
Element found at index 3.
```

15. "Write a program to perform the following operations on an array or list of elements:

Search for an element using Binary Search.

If the element is found, display its position along with a proper message.
Sort the elements of the array or list in ascending order using one of the following sorting algorithms:

- Bubble Sort
- Selection Sort
- Insertion Sort
- Quick Sort
- Merge Sort
- Heap Sort

Allow the user to choose the desired sorting algorithm. Once sorted, the array should be displayed. After sorting, implement binary search to allow searching within the sorted array."

Code:-
```
#include <stdio.h>
#include <stdlib.h>

// Function prototypes
void bubbleSort(int array[], int size);
void insertionSort(int array[], int size);
void selectionSort(int array[], int size);
void mergeSort(int array[], int low, int high);
void quickSort(int array[], int low, int high);
void heapSort(int array[], int size);
int binarySearch(int array[], int start, int end, int target);
void printArray(int array[], int size);
void merge(int array[], int low, int mid, int high);
void heapify(int array[], int size, int root);

int main() {
   int arraySize;

   printf("Enter size: ");
   if (scanf("%d", &arraySize) != 1 || arraySize <= 0) {
      printf("Invalid input. Array size must be a positive integer.\n");
      return 1;
   }

   int array[arraySize];

   printf("Enter %d elements: ", arraySize);
   for (int i = 0; i < arraySize; i++) {
      if (scanf("%d", &array[i]) != 1) {
         printf("Error reading input.\n");
         return 1;
      }
   }

   int sortChoice;
   printf("\nChoose a sorting method:\n");
   printf("1. Bubble Sort\n");
   printf("2. Insertion Sort\n");
printf("3. Selection Sort\n");
printf("4. Merge Sort\n");
printf("5. Quick Sort\n");
printf("6. Heap Sort\n");
scanf("%d", &sortChoice);

switch (sortChoice) {
  case 1:
    bubbleSort(array, arraySize);
    printf("Array after Bubble Sort: ");
    printArray(array, arraySize);
    break;
  case 2:
    insertionSort(array, arraySize);
    printf("Array after Insertion Sort: ");
    printArray(array, arraySize);
    break;
  case 3:
    selectionSort(array, arraySize);
    printf("Array after Selection Sort: ");
    printArray(array, arraySize);
    break;
  case 4:
    mergeSort(array, 0, arraySize - 1);
    printf("Array after Merge Sort: ");
    printArray(array, arraySize);
    break;
  case 5:
    quickSort(array, 0, arraySize - 1);
    printf("Array after Quick Sort: ");
    printArray(array, arraySize);
    break;
  case 6:
    heapSort(array, arraySize);
    printf("Array after Heap Sort: ");
    printArray(array, arraySize);
    break;
  default:
    printf("Invalid choice. Exiting program.\n");
    return 1;
}

int choice;
while (1) {
  printf("\nEnter your choice:\n");
  printf("1. Search for an element\n");
  printf("2. Print the array\n");
  printf("3. Exit\n");
  scanf("%d", &choice);

  switch (choice) {
    case 1: {
       int key;
       printf("Enter key to search: ");
       if (scanf("%d", &key) != 1) {
          printf("Error reading input.\n");
          return 1;
       }

       int index = binarySearch(array, 0, arraySize - 1, key);
       if (index == -1) {
          printf("Key not found in the array.\n");
       } else {
          printf("Key %d found at index %d.\n", array[index], index);
       }
       break;
    }
    case 2:
       printf("Array: ");
       printArray(array, arraySize);
       break;
    case 3:
       printf("Exiting...\n");
       return 0;
    default:
       printf("Invalid choice. Please try again.\n");
       break;
  }
}
```

```
    return 0;
}

// Bubble Sort
void bubbleSort(int array[], int size) {
    for (int pass = 0; pass < size - 1; pass++) {
        int swapped = 0;
        for (int i = 0; i < size - 1 - pass; i++) {
            if (array[i] > array[i + 1]) {
                int temp = array[i];
                array[i] = array[i + 1];
                array[i + 1] = temp;
                swapped = 1;
            }
        }
        if (!swapped) break;
    }
}

// Insertion Sort
void insertionSort(int array[], int size) {
    for (int i = 1; i < size; i++) {
        int temp = array[i];
        int j = i - 1;
        while (j >= 0 && array[j] > temp) {
            array[j + 1] = array[j];
            j--;
        }
        array[j + 1] = temp;
    }
}

// Selection Sort
void selectionSort(int array[], int size) {
    for (int i = 0; i < size - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < size; j++) {
            if (array[j] < array[minIndex]) {
                minIndex = j;
            }
        }
        if (minIndex != i) {
            int temp = array[i];
            array[i] = array[minIndex];
            array[minIndex] = temp;
        }
    }
}

// Merge Sort
void mergeSort(int array[], int low, int high) {
    if (low < high) {
        int mid = low + (high - low) / 2;
        mergeSort(array, low, mid);
        mergeSort(array, mid + 1, high);
        merge(array, low, mid, high);
    }
}

void merge(int array[], int low, int mid, int high) {
    int n1 = mid - low + 1;
    int n2 = high - mid;

    int left[n1], right[n2];
    for (int i = 0; i < n1; i++) left[i] = array[low + i];
    for (int i = 0; i < n2; i++) right[i] = array[mid + 1 + i];

    int i = 0, j = 0, k = low;
    while (i < n1 && j < n2) {
        if (left[i] <= right[j]) {
            array[k++] = left[i++];
        } else {
            array[k++] = right[j++];
        }
    }
    while (i < n1) array[k++] = left[i++];
    while (j < n2) array[k++] = right[j++];
}

// Quick Sort
```

```
void quickSort(int array[], int low, int high) {
   if (low < high) {
      int pivot = array[high];
      int i = low - 1;

      for (int j = low; j < high; j++) {
         if (array[j] < pivot) {
            i++;
            int temp = array[i];
            array[i] = array[j];
            array[j] = temp;
         }
      }

      int temp = array[i + 1];
      array[i + 1] = array[high];
      array[high] = temp;

      quickSort(array, low, i);
      quickSort(array, i + 2, high);
   }
}

// Heap Sort
void heapSort(int array[], int size) {
   for (int i = size / 2 - 1; i >= 0; i--) {
      heapify(array, size, i);
   }
   for (int i = size - 1; i > 0; i--) {
      int temp = array[0];
      array[0] = array[i];
      array[i] = temp;

      heapify(array, i, 0);
   }
}

void heapify(int array[], int size, int root) {
   int largest = root;
   int left = 2 * root + 1;
```

```
    int right = 2 * root + 2;

    if (left < size && array[left] > array[largest]) {
        largest = left;
    }
    if (right < size && array[right] > array[largest]) {
        largest = right;
    }
    if (largest != root) {
        int temp = array[root];
        array[root] = array[largest];
        array[largest] = temp;

        heapify(array, size, largest);
    }
}

// Binary Search
int binarySearch(int array[], int start, int end, int target) {
    if (start > end) {
        return -1; // not found
    }

    int middle = start + (end - start) / 2;

    if (array[middle] == target) {
        return middle;
    }
    if (array[middle] < target) {
        return binarySearch(array, middle + 1, end, target);
    } else {
        return binarySearch(array, start, middle - 1, target);
    }
}

// Print Array
void printArray(int array[], int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", array[i]);
    }
```

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

Output:-

```
Enter size: 5
Enter 5 elements: 2 5 9 1 9

Choose a sorting method:
1. Bubble Sort
2. Insertion Sort
3. Selection Sort
4. Merge Sort
5. Quick Sort
6. Heap Sort
1
Array after Bubble Sort: 1 2 5 9 9

Enter your choice:
1. Search for an element
2. Print the array
3. Exit
1
Enter key to search: 2
Key 2 found at index 1.

Enter your choice:
1. Search for an element
2. Print the array
3. Exit
3
Exiting...
```