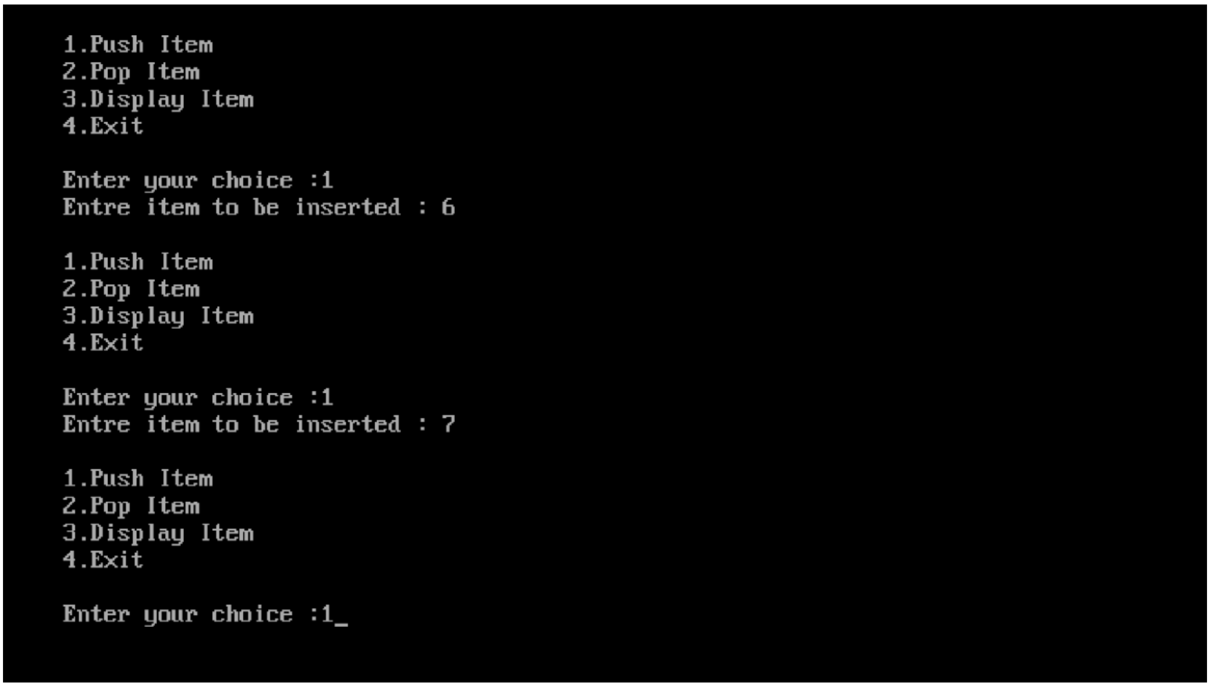
Write a program to implement a Stack using two Queues

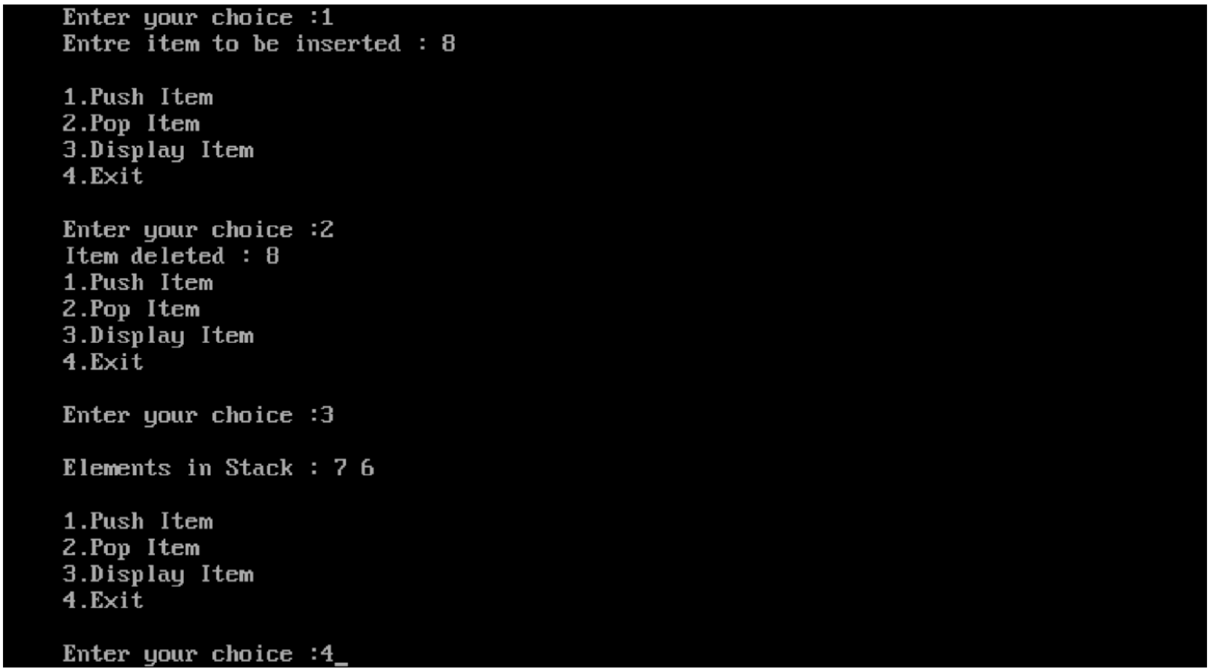
```
#include <stdio.h>
#include<conio.h>
#define N 20
//declaring two queues and there fron and rear variables
int queue1[N],queue2[N];
int f1= -1, r1= -1;
int f2= -1, r2= -1;
int count=0;
//declaring two queues operations
void enqueue1(int x);
int dequeue1();
void enqueue2(int x);
int dequeue2();
//declaring stack operations
void push(int x);
int pop();
void display();
//main function
void main()
{ //declaring local variable
        int ch, num;
        clrscr();
        while (ch != 4)
        { //input user choice
                printf("\n1.Push Item\n2.Pop Item\n3.Display Item\n4.Exit\n");
                 printf("\nEnter your choice :");
                scanf("%d", &ch);
                switch (ch)
                {
                        case 1: printf("Entre item to be inserted : ");
                        scanf("%d", &num);
                        push(num); //insert function call
                        break;
                        case 2: printf("Item deleted : %d",pop()); //delete function call
                        break;
                        case 3: display(); //printing stack elements
                        break;
                        case 4: exit(0);
                        break; //exit
                        default:printf("\nInvalide Choice !!!\n"); //invalid input
                }
        }
}
//enqueue operation for queue 1
void enqueue1(int x)
{
        if(r1==N-1)
        {
                printf("Overflow");
        }
        else
        {
                if(f1== -1)
                {
                        f1=0;
                }
                r1=r1+1;
                queue1[r1]=x;
        }
}
//dequeue operation for queue 1
int dequeue1()
{
        int temp;
        if(f1== -1 || f1 > r1)
        {
                printf("underflow");
        }
        else
        {
                temp = queue1[f1];
                f1++;
        }
        return(temp);
}
 //enqueue operation for queue 2
void enqueue2(int x)
{
        if(r2==N-1)
        {
                printf("Overflow");
        }
        else
        {
                if(f2== -1)
                {
                        f2=0;
                }
```

```
            r2=r2+1;
            queue2[r2]=x;
        }
}
//dequeue operation for queue 2
int dequeue2()
{
        int temp;
        if(f2== -1 || f2 > r2)
        {
            printf("Underflow");
        }
        else
        {
            temp = queue2[f2];
            f2++;
        }
        return(temp);
}
// push functon to insert data into stack of two Queues

void push(int x)
{
        int i;
        enqueue1(x);
        for (i = 0; i < count ; i++)
        {
            enqueue1(dequeue2());
        }
        count++;
```

```
     for(i=0; i<count;i++)
    {
        enqueue2(dequeue1());
    }
}
// pop function to delete data from stack of two Queues
int pop()
{
    count--;
     return dequeue2();
}
// displaying the data of stack of two Queue
void display()
{
    int i;
    printf("\nElements in Stack : ");
    for (i = f2; i <=r2 ; i++)
    {
        printf("%d ", queue2[i]);
    }
    printf("\n");
}
```



```
1.Push Item
2.Pop Item
3.Display Item
4.Exit

Enter your choice :1
Entre item to be inserted : 6

1.Push Item
2.Pop Item
3.Display Item
4.Exit

Enter your choice :1
Entre item to be inserted : 7

1.Push Item
2.Pop Item
3.Display Item
4.Exit

Enter your choice :1_
```

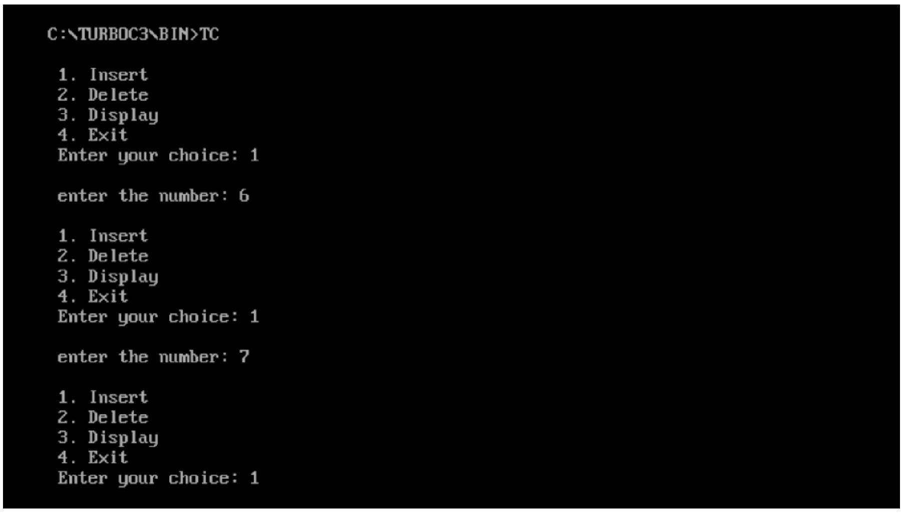


```
Enter your choice :1
Entre item to be inserted : 8

1.Push Item
2.Pop Item
3.Display Item
4.Exit

Enter your choice :2
 Item deleted : 8
1.Push Item
2.Pop Item
3.Display Item
4.Exit

Enter your choice :3

Elements in Stack : 7 6

1.Push Item
2.Pop Item
3.Display Item
4.Exit

Enter your choice :4_
```

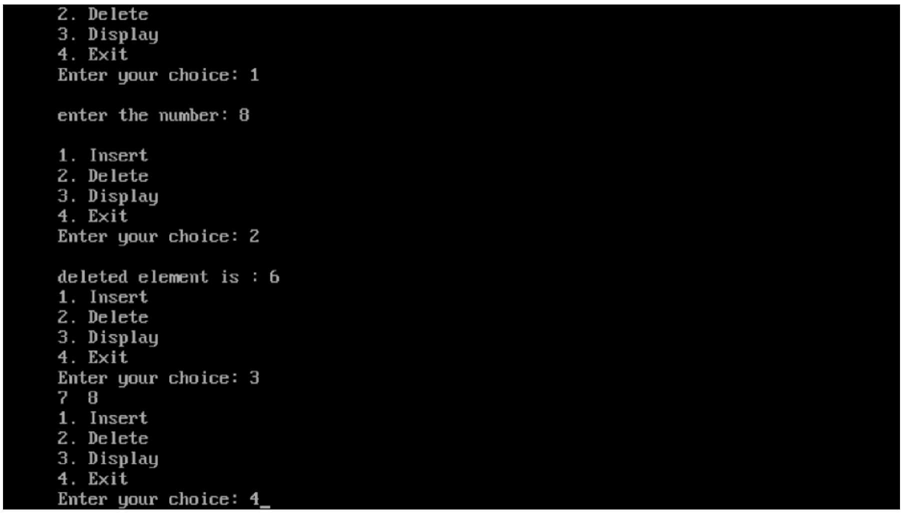
Write a program to implement a Queue using two stacks.

```
#include<stdio.h>
#include<stdlib.h>
#define N 10

int s1[N],s2[N];
int top1=-1;
int top2=-1;
int count=0;

void enqueue(int x);
void deque();
void push1(int x);
void push2(int x);
int pop1();
int pop2();
void display();

void main()
{
        int ch,x;
        while(1)
        {
                printf("\n 1. Insert");
                printf("\n 2. Delete");
                printf("\n 3. Display");
                printf("\n 4. Exit");
                printf("\n Enter your choice: ");
                scanf("%d", &ch);
                switch(ch)
                {
                        case 1: printf("\n enter the number: ");
                scanf("%d", &x);
                        enqueue(x);
                break;
                        case 2: deque();
                break;
                        case 3: display();
                break;
                        case 4: exit(0);
                        default: printf("wrong choice");
                }
        }
}

void enqueue(int x)
{
        push1(x);
        count++;
}

void push1(int x)
{
        if(top1 == N-1)
        {
                printf("\n stack is full");
        }
        else
        {
                top1++;
                s1[top1]=x;
        }
}

void push2(int x)
{
        if(top2 == N-1)
        {
                printf("\n stack is full");
        }
        else
        {
                top2++;
                s2[top2]=x;
        }
}

int pop1()
{
        return(s1[top1--]);
}

int pop2()
{
        return(s2[top2--]);
}
```

```
void deque()
{
        int i,a,b;
        if(top1 ==  -1 && top2 == -1)
        {
                printf("\n stack is empty");
        }
        else
        {
                for(i=0;i<count;i++)
                {
                        a=pop1();
                        push2(a);
                }
                b=pop2();
                printf("\n deleted element is : %d", b);

                count--;
                for(i=0;i<count;i++)
                {
                         a=pop2();
                         push1(a);
                }
        }
}

void display()
{
        int i;
        for(i=0;i<=top1;i++)
        {
                printf(" %d ",s1[i]);
        }
}
```



```
C:\TURBOC3\BIN>TC

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

 enter the number: 6

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

 enter the number: 7

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

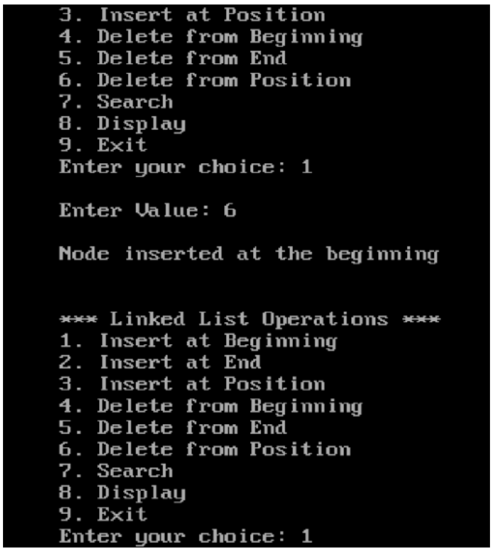


```
 2. Delete
 3. Display
 4. Exit
 Enter your choice: 1

 enter the number: 8

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

 deleted element is : 6
 1. Insert
 2. Delete
 3. Display
 4. Exit
 Enter your choice: 3
 7  8
 1. Insert
 2. Delete
 3. Display
 4. Exit
 Enter your choice: 4_
```

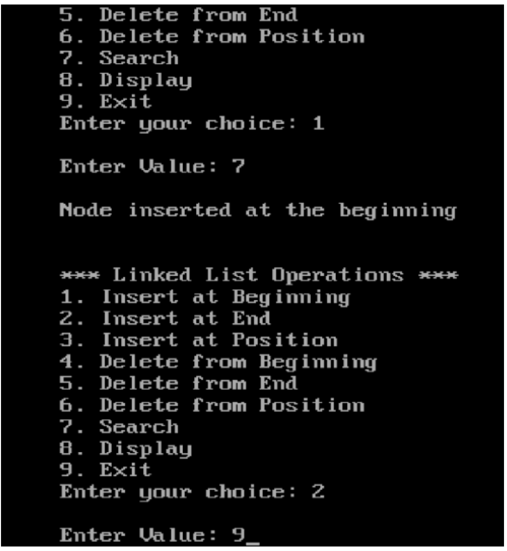
Write programs to implement the following data structures: (a) Single Linked list

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
	int data;
	struct Node* next;
};
struct Node* head = NULL;
struct Node* ptr, *temp;
void insert_begin();
void insert_last();
void insert_position();
void delete_begin();
void delete_last();
void delete_position();
void search();
void display();
void main()
{
	int choice;
	while (1)
	{
		printf("\n\n*** Linked List Operations ***\n");
		printf("1. Insert at Beginning\n");
		printf("2. Insert at End\n");
		printf("3. Insert at Position\n");
		printf("4. Delete from Beginning\n");
		printf("5. Delete from End\n");
		printf("6. Delete from Position\n");
		printf("7. Search\n");
		printf("8. Display\n");
		printf("9. Exit\n");
		printf("Enter your choice: ");
		scanf("%d", &choice);

		switch (choice)
		{
			case 1: insert_begin(); break;
			case 2: insert_last(); break;
			case 3: insert_position(); break;
			case 4: delete_begin(); break;
			case 5: delete_last(); break;
			case 6: delete_position(); break;
```

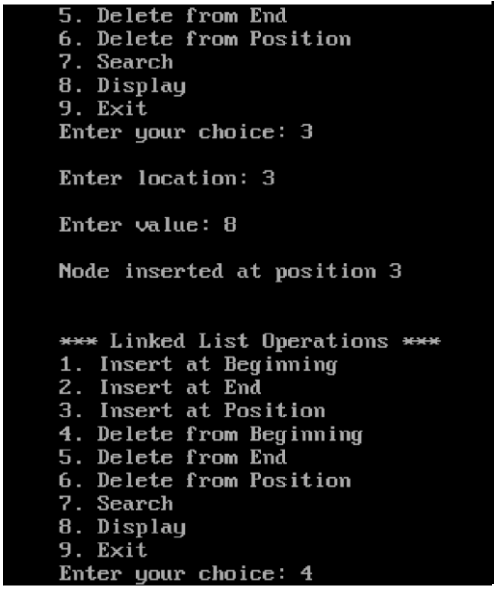
```
            case 7: search(); break;
            case 8: display(); break;
            case 9: exit(0); break;
            default: printf("Invalid choice, please try again\n");
        }
    }
}
void insert_begin()
{
    int val;
    ptr = (struct Node*) malloc(sizeof(struct Node));
    if (ptr == NULL)
    {
        printf("Memory is not allocated\n");
    }
     else
    {
        printf("\nEnter Value: ");
        scanf("%d", &val);
        ptr->data = val;
        ptr->next = head;
        head = ptr;
        printf("\nNode inserted at the beginning\n");
    }
}
void insert_last()
{
    int val;
     ptr = (struct Node*) malloc(sizeof(struct Node));
    if (ptr == NULL)
    {
        printf("Memory is not allocated\n");
    }
    else
    {
        printf("\nEnter Value: ");
        scanf("%d", &val);
        ptr->data = val;
        ptr->next = NULL;
        if (head == NULL)
        {
            head = ptr;
        }
        else
        {
            temp = head;
            while (temp->next != NULL)
            {
                 temp = temp->next;
            }
            temp->next = ptr;
        }
        printf("\nNode inserted at the end\n");
    }
}
void insert_position()
{
    int i, pos, val;
    printf("\nEnter location: ");
    scanf("%d", &pos);
    ptr = (struct Node*) malloc(sizeof(struct Node));
    if (ptr == NULL)
    {
        printf("\nMemory is not Allocated\n");
    }
    else
    {
        printf("\nEnter value: ");
        scanf("%d", &val);
        ptr->data = val;
        if (pos == 1)
        {
            ptr->next = head;
            head = ptr;
        }
        else
        {
            temp = head;
            for (i = 1; i < pos - 1; i++)
            {
                temp = temp->next;
            }
            if (temp == NULL)
            {
                printf("\nCan't Insert");
            }
            else
            {
                ptr->next = temp->next;
                temp->next = ptr;
                printf("\nNode inserted at position %d\n", pos);
            }
        }
    }
}
void delete_begin()
{
    if (head == NULL)
    {
        printf("Linked List is empty\n");
    }
    else
    {
        temp = head;
        printf("\n%d is deleted\n", head->data);
        head = head->next;
        free(temp);
    }
}
void delete_last()
{
    if (head == NULL)
    {
        printf("Linked List is empty\n");
    }
    else if (head->next == NULL)
    {
        printf("\n%d is deleted\n", head->data);
        free(head);
        head = NULL;
     }
    else
    {
        struct Node* temp1 = head;
        while (temp1->next->next != NULL)
        {
            temp1 = temp1->next;
        }
        printf("\n%d is deleted\n", temp1->next->data);
        free(temp1->next);
        temp1->next = NULL;
    }
}
void delete_position()
{
    int pos, i;
    if (head == NULL)
    {
        printf("Linked List is empty\n");
    }
    else
    {
        printf("\nEnter Position: ");
        scanf("%d", &pos);
        temp = head;
        if (pos == 1)
        {
            printf("\n%d is deleted\n", head->data);
            head = head->next;
            free(temp);
        }
        else
        {
            struct Node* temp1;
            for (i = 1; i < pos; i++)
            {
                temp = temp->next;
            }
            if (temp == NULL)
            {
                printf("\n Cant delete");
            }
            else
            {
                temp1 = temp->next;
                printf("\n%d is deleted\n", temp1->data);
                temp->next = temp1->next;
                free(temp1);
            }
        }
    }
}
void search()
{
    int val, i = 0, flag = 0;
    if (head == NULL)
    {
        printf("Linked List is empty\n");
    }
    else
    {
        printf("\nEnter value to search: ");
        scanf("%d", &val);
        temp = head;
        while (temp != NULL)
        {
            if (temp->data == val)
            {
                printf("\n%d found at location %d\n", val, i + 1);
                flag = 1;
                break;
            }
            i++;
            temp = temp->next;
        }
        if (!flag)
        {
            printf("\nValue %d not found\n", val);
        }
    }
}
void display()
{
    if (head == NULL)
    {
        printf("Linked List is empty\n");
    }
    else
    {
        temp = head;
        printf("\nLinked List: ");
        while (temp != NULL)
        {
            printf("%d -> ", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}
```



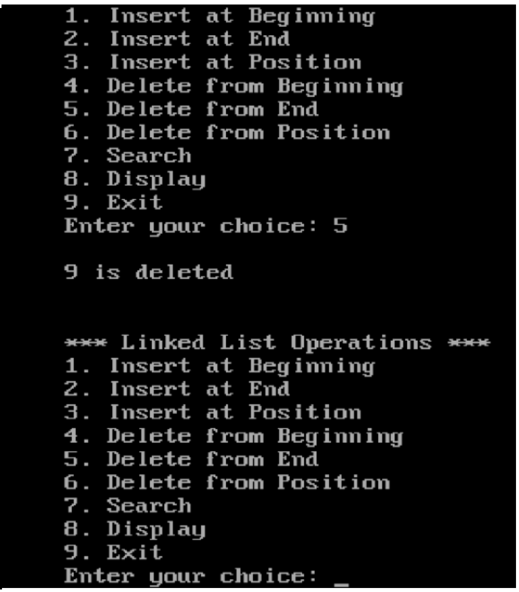
```
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 1

Enter Value: 6

Node inserted at the beginning


*** Linked List Operations ***
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. Search
8. Display
9. Exit
Enter your choice: 1
```



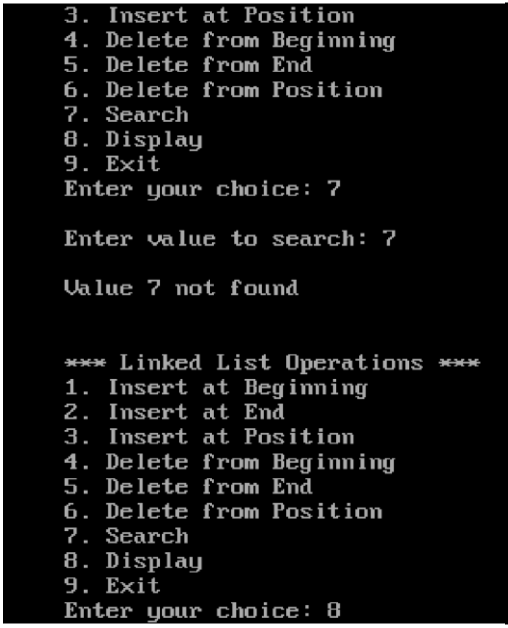
```
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 1

Enter Value: 7

Node inserted at the beginning


*** Linked List Operations ***
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. Search
8. Display
9. Exit
Enter your choice: 2

Enter Value: 9_
```



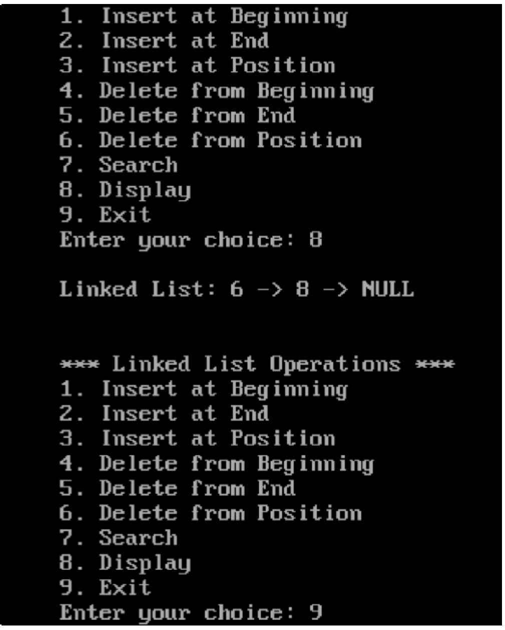
```
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 3

Enter location: 3

Enter value: 8

Node inserted at position 3


*** Linked List Operations ***
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. Search
8. Display
9. Exit
Enter your choice: 4
```



```
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. Search
8. Display
9. Exit
Enter your choice: 5

9 is deleted


*** Linked List Operations ***
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. Search
8. Display
9. Exit
Enter your choice: _
```



```
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 7

Enter value to search: 7

Value 7 not found


*** Linked List Operations ***
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. Search
8. Display
9. Exit
Enter your choice: 8
```



```
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. Search
8. Display
9. Exit
Enter your choice: 8

Linked List: 6 -> 8 -> NULL


*** Linked List Operations ***
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. Search
8. Display
9. Exit
Enter your choice: 9
```

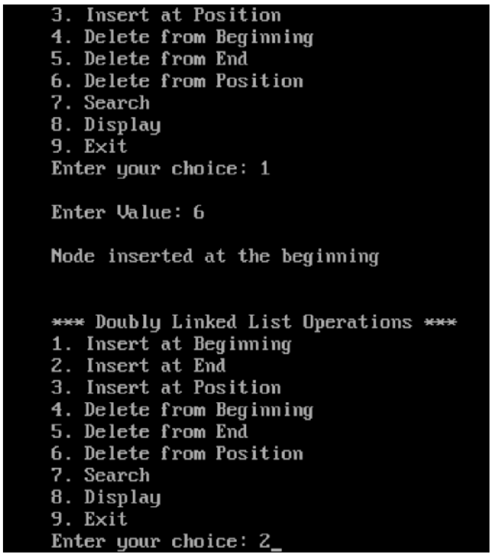
Write programs to implement the following data structures: (b) Double Linked list.

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

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

struct Node* head = NULL;
struct Node* ptr;
struct Node*temp, *temp1;

int val, flag=0, i=0, loc, pos;

void insert_begin();
void insert_last();
void insert_position();
void delete_begin();
void delete_last();
void delete_position();
void search();
void display();

void main()
{
        int choice;
        while (1)
        {
      printf("\n\n*** Doubly Linked List Operations ***\n");
      printf("1. Insert at Beginning\n");
      printf("2. Insert at End\n");
      printf("3. Insert at Position\n");
      printf("4. Delete from Beginning\n");
      printf("5. Delete from End\n");
      printf("6. Delete from Position\n");
      printf("7. Search\n");
      printf("8. Display\n");
      printf("9. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice)
      {
        case 1: insert_begin(); break;
        case 2: insert_last(); break;
```

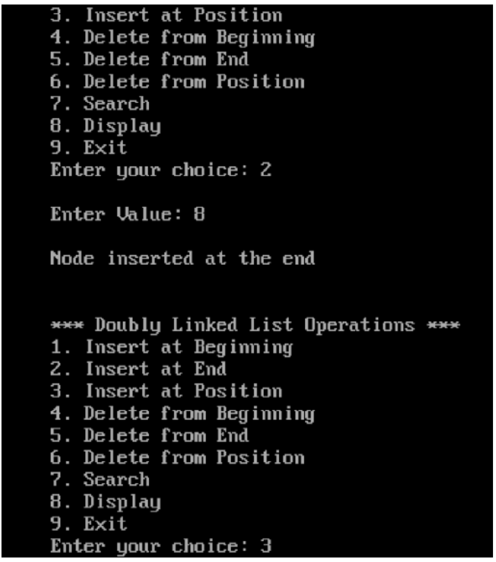
```
            case 3: insert_position(); break;
            case 4: delete_begin(); break;
            case 5: delete_last(); break;
            case 6: delete_position(); break;
            case 7: search(); break;
            case 8: display(); break;
            case 9: exit(0); break;
            default: printf("Invalid choice, please try again\n");
        }
    }
}

void insert_begin()
{
            ptr = (struct Node*) malloc(sizeof(struct Node));
            if (ptr == NULL)
            {
            printf("Memory is not allocated\n");
     }
            else
            {
        printf("\nEnter Value: ");
        scanf("%d", &val);
                        if (head == NULL)
                        {
              ptr->data = val;
              ptr->prev = NULL;
                                    ptr->next = NULL;
                                    head = ptr;
          }
                        else
                        {
                                    ptr->data = val;
                                    ptr->prev = NULL;
                        ptr->next = head;
                        head->prev = ptr;
                        head = ptr;
                        }
            }
            printf("\nNode inserted at the beginning\n");
}

void insert_last()
{
            ptr = (struct Node*) malloc(sizeof(struct Node));
            if (ptr == NULL)
            {
            printf("Memory is not allocated\n");
     }
            else
        {
        printf("\nEnter Value: ");
        scanf("%d", &val);
                if (head == NULL)
                {
         ptr->data = val;
         ptr->next = NULL;
         ptr->prev = NULL;
         head = ptr;
       }
                else
                {
                temp = head;
                while (temp->next != NULL)
                        {
                        temp = temp->next;
                }
                        ptr->data = val;
                ptr->next = NULL;
                temp->next = ptr;
                ptr->prev = temp;
        }
        }
   printf("\nNode inserted at the end\n");
}

void insert_position()
{
        ptr = (struct Node*) malloc(sizeof(struct Node));
        if (ptr == NULL)
        {
        printf("\nMemory is not Allocated\n");
   }
        else
        {
        printf("\nEnter location: ");
        scanf("%d", &pos);
        temp = head;
        for (i = 0; i < pos ; i++)
                {
                if (temp == NULL)
                        {
                printf("\nCan't Insert");
                }
                }
                printf("\nEnter Value");
                scanf("%d",&val);
                ptr-> data = val;
       ptr->prev = temp;
                ptr->next = temp->next;
        temp->next = ptr;
          }
    printf("\nNode inserted at position %d\n", pos);
}

void delete_begin()
{
          if (head == NULL)
          {
          printf("Doubly Linked List is empty\n");
    }
          else
          {
          temp = head;
          printf("\n%d is deleted\n", head->data);
          head = head->next;
          head->prev = NULL;
          temp->next = NULL;
          free(temp);
    }
}

void delete_last()
{
          if (head == NULL)
          {
          printf("Doubly Linked List is empty\n");
    }
          else
          {
          temp = head;
          while (temp->next != NULL)
                    {
                    temp = temp->next;
          }
          printf("\n%d is deleted\n", temp->data);
          temp->prev->next = NULL;
          temp->prev=NULL;
                    free(temp);
          }
}

void delete_position()
{
          if (head == NULL)
          {
          printf("Doubly Linked List is empty\n");
    }
          else
          {
```

```
        temp=head;
        printf("\nEnter Position: ");
        scanf("%d", &pos);
        for (i = 0; i < pos; i++)
                {
        temp = temp->next;
        if (temp == NULL)
                        {
                printf("\n Can't Delete");
                }
                }
                printf("\n%d is deleted\n", temp->data);
                temp->prev->next = temp->next;
                temp->next->prev = temp->prev;
                temp->next =NULL;
                temp->prev =NULL;
                free(temp);
        }
}

void search()
{
        if (head == NULL)
        {
        printf("Doubly Linked List is empty\n");
        }
        else
        {
        printf("\nEnter value to search: ");
        scanf("%d", &val);
        temp = head;
        while (temp != NULL)
                {
        if (temp->data == val)
                        {
             printf("\n%d found at location %d\n", val, i);
             flag = 1;
             break;
                }
                i++;
                temp = temp->next;
        }
        if (!flag)
        {
                printf("\nValue %d not found\n", val);
        }
        }
}

void display()
```

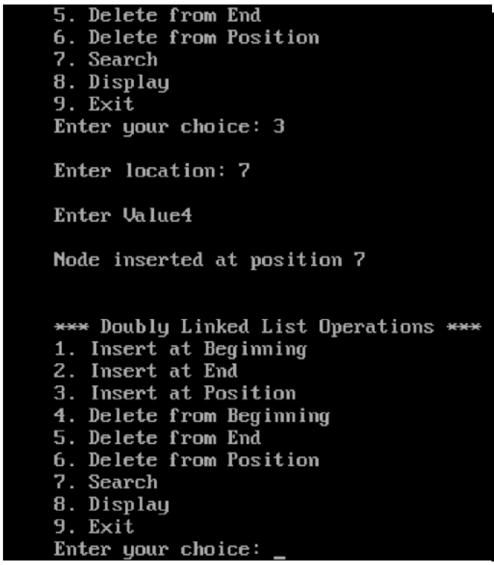
```
{
        if (head == NULL)
        {
        printf("Doubly Linked List is empty\n");
    }
        else
        {
        temp = head;
        printf("\nDoubly Linked List: ");
        while (temp != NULL)
                {
                printf("%d <-> ", temp->data);
                temp = temp->next;
        }
        printf("NULL\n");
        }
}
```



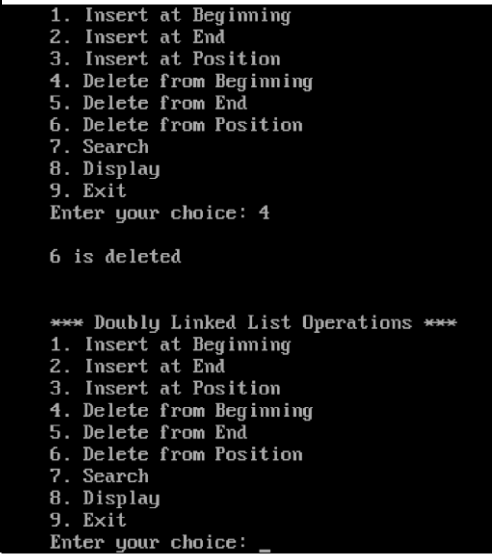
```
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 1

Enter Value: 6

Node inserted at the beginning


*** Doubly Linked List Operations ***
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. Search
8. Display
9. Exit
Enter your choice: 2_
```



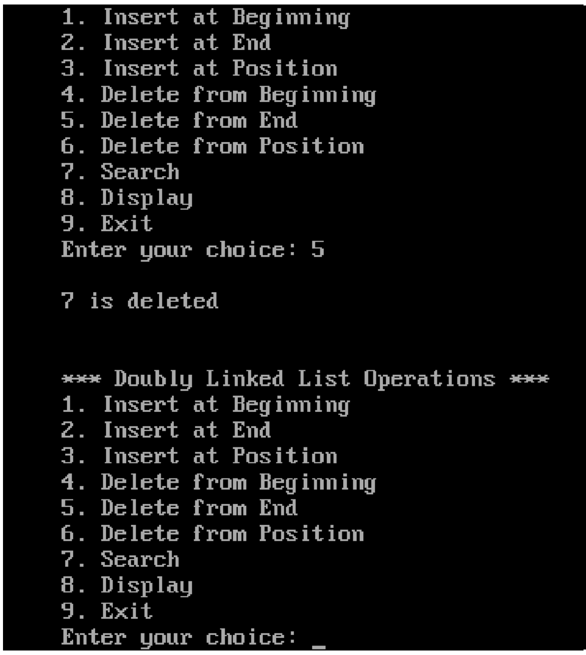
```
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 2

Enter Value: 8

Node inserted at the end


*** Doubly Linked List Operations ***
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. Search
8. Display
9. Exit
Enter your choice: 3
```



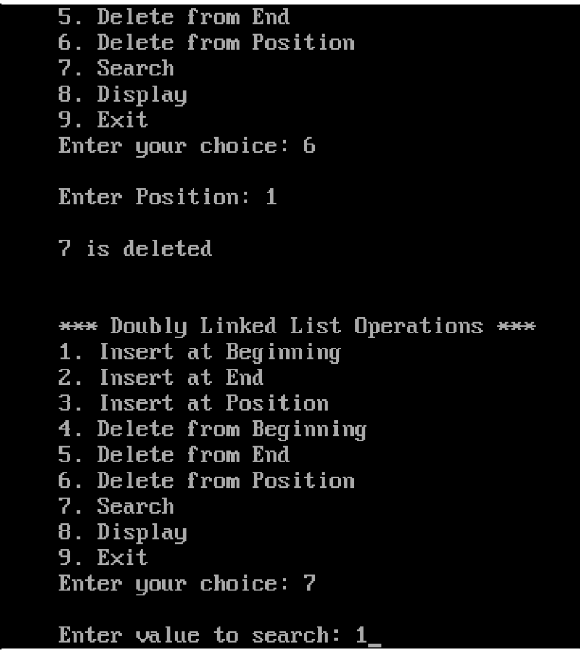
```
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 3

Enter location: 7

Enter Value4

Node inserted at position 7


*** Doubly Linked List Operations ***
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. Search
8. Display
9. Exit
Enter your choice: _
```



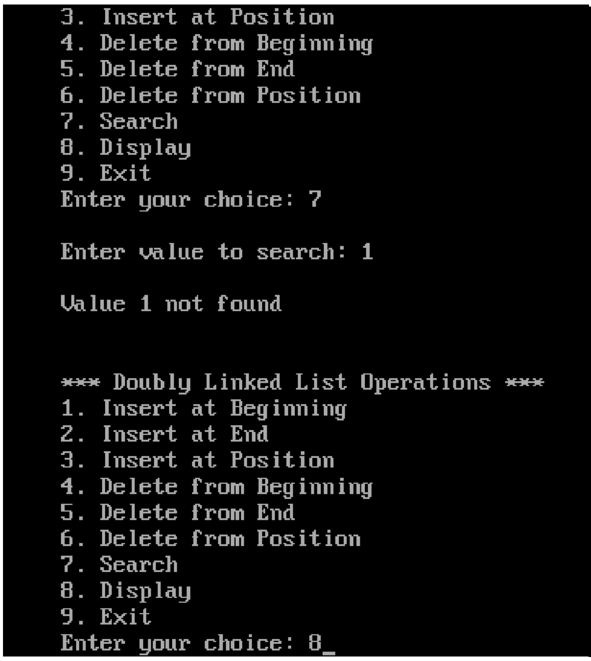
```
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 4

6 is deleted


*** Doubly Linked List Operations ***
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. Search
8. Display
9. Exit
Enter your choice: _
```



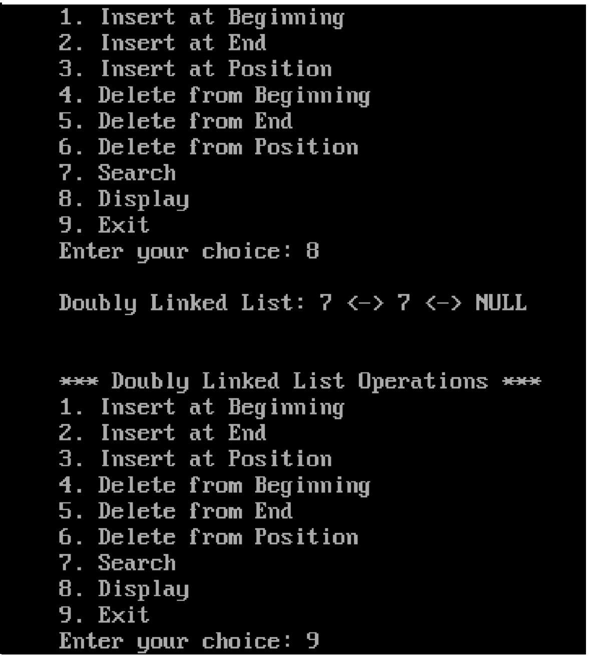
```
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 5

7 is deleted


*** Doubly Linked List Operations ***
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. Search
8. Display
9. Exit
Enter your choice: _
```



```
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 6

Enter Position: 1

7 is deleted


*** Doubly Linked List Operations ***
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. Search
8. Display
9. Exit
Enter your choice: 7

Enter value to search: 1_
```



```
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 7

Enter value to search: 1

Value 1 not found


*** Doubly Linked List Operations ***
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. Search
8. Display
9. Exit
Enter your choice: 8_
```



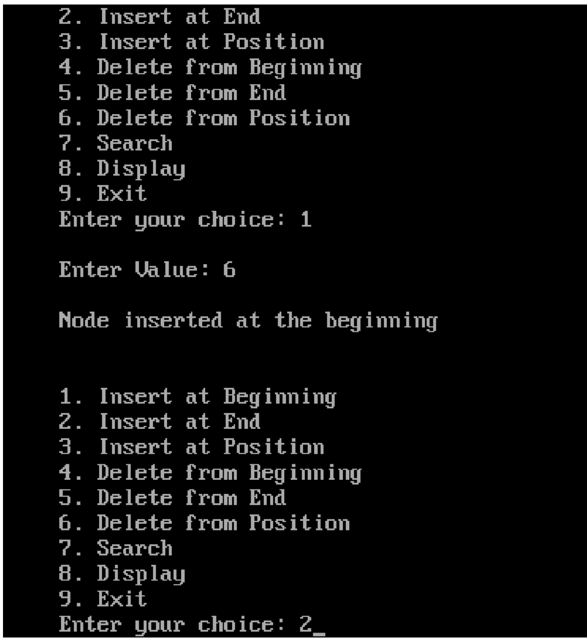
```
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 8

Doubly Linked List: 7 <-> 7 <-> NULL


*** Doubly Linked List Operations ***
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. Search
8. Display
9. Exit
Enter your choice: 9
```

Write a program to implement the Circular Singly linked list

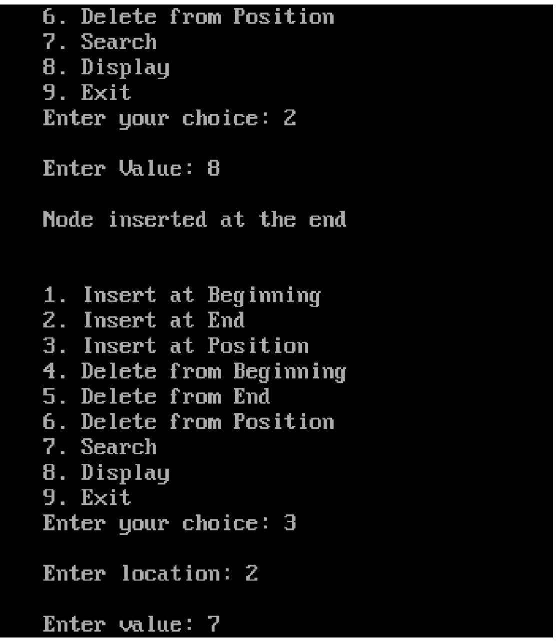
```
#include <stdio.h>
#include <stdlib.h>

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

struct Node* head = NULL;
struct Node* ptr, *temp, *temp1;
int val,i,pos,flag=0;

void insert_begin();
void insert_last();
void insert_position();
void delete_begin();
void delete_last();
void delete_position();
void search();
void display();

void main()
{
        int choice;
        while (1)
        {
                 printf("\n\n1. Insert at Beginning\n");
                printf("2. Insert at End\n");
                printf("3. Insert at Position\n");
                printf("4. Delete from Beginning\n");
                printf("5. Delete from End\n");
                 printf("6. Delete from Position\n");
                printf("7. Search\n");
                printf("8. Display\n");
                printf("9. Exit\n");
                printf("Enter your choice: ");
                scanf("%d", &choice);
                switch (choice)
                {
                        case 1: insert_begin(); break;
                        case 2: insert_last(); break;
                        case 3: insert_position(); break;
                        case 4: delete_begin(); break;
                        case 5: delete_last(); break;
                        case 6: delete_position(); break;
                        case 7: search(); break;
                        case 8: display(); break;
                        case 9: exit(0); break;
                        default: printf("Invalid choice, please try again\n");
                }
        }
}
void insert_begin()
{
        ptr = (struct Node*) malloc(sizeof(struct Node));
        if (ptr == NULL)
        {
                 printf("Memory is not allocated\n");
        }
        else
        {
                printf("\nEnter Value: ");
                scanf("%d", &val);
                 if (head == NULL)
                {
                        ptr->data = val;
                        head = ptr;
                        ptr->next = head;
                }
                else
                {
                        temp = head;
                        while (temp->next != head)
                        {
                                temp = temp->next;
                        }
                        ptr->data = val;
                        ptr->next = head;
                        head = ptr;
                        temp->next = ptr;
                }
                printf("\nNode inserted at the beginning\n");
        }
}

void insert_last()
{
        ptr = (struct Node*) malloc(sizeof(struct Node));
        if (ptr == NULL)
        {
                printf("Memory is not allocated\n");
        }
        else
        {
                 printf("\nEnter Value: ");
                 scanf("%d", &val);
        if (head == NULL)
        {
            ptr->data = val;
            head = ptr;
            ptr->next = ptr;
        }
        else
        {
            temp = head;
            while (temp->next != head)
            {
                temp = temp->next;
            }
             ptr->data = val;
            temp->next = ptr;
            ptr->next = head;
        }
         printf("\nNode inserted at the end\n");
    }
}

void insert_position()
{
    int i, pos, val;
    printf("\nEnter location: ");
    scanf("%d", &pos);
    ptr = (struct Node*) malloc(sizeof(struct Node));
    if (ptr == NULL)
    {
         printf("\nMemory is not Allocated\n");
    }
    else
    {
        printf("\nEnter value: ");
        scanf("%d", &val);
        ptr->data = val;
        if (pos == 1)
        {
            ptr->next = head;
            head = ptr;
        }
        else
        {
            temp = head;
            for (i = 1; i < pos - 1; i++)
            {
                temp = temp->next;
            }
            if (temp == NULL)
            {
                printf("\nCan't Insert");
            }
            else
            {
            ptr->next = temp->next;
            temp->next = ptr;
            printf("\nNode inserted at position %d\n", pos);
            }
        }
    }
}

void delete_begin()
{
    if (head == NULL)
    {
        printf("Circular Linked List is empty\n");
    }
    else
    {
        temp = head;
        while (temp->next != head)
        {
            temp = temp->next;
        }
        printf("\n%d is deleted\n", head->data);
        temp1 = head;
        head = head->next;
        temp->next = head;
        temp1->next = NULL;
        free(temp1);
    }
}

void delete_last()
{
    if (head == NULL)
    {
        printf("Circular Linked List is empty\n");
    }
     else
    {
        temp = head;
        while (temp->next!= head)
        {
            temp1 = temp;
            temp = temp->next;
        }
        printf("\n%d is deleted\n", temp->data);
        temp1->next = head;
```

```
		free(temp);
	}
}

void delete_position()
{
	int pos, i;
	if (head == NULL)
	{
		printf("Linked List is empty\n");
	}
	else
	{
		printf("\nEnter Position: ");
		scanf("%d", &pos);
		temp = head;
		if (pos == 1)
		{
			printf("\n%d is deleted\n", head->data);
			head = head->next;
			free(temp);
		}
		else
		{
			struct Node* temp1;
			for (i = 1; i < pos-1; i++)
			{
				temp = temp->next;
			}
			if (temp == NULL)
			{
				printf("\n Cant delete");
			}
			else
			{
				temp1 = temp->next;
				printf("\n%d is deleted\n", temp1->data);
				temp->next = temp1->next;
				free(temp1);
			}
		}
	}
}

void search()
{
	int val, i = 0, flag = 0;
	if (head == NULL)
	{
		printf("Linked List is empty\n");
```

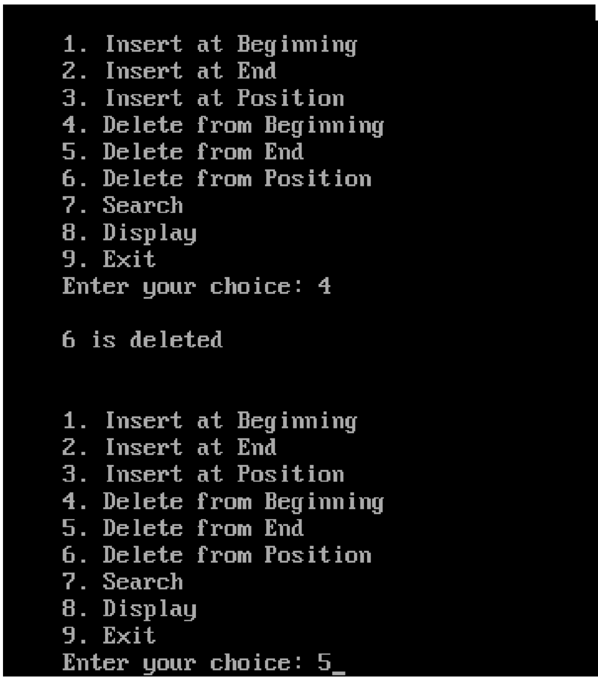
```
    }
    else
    {
        printf("\nEnter value to search: ");
        scanf("%d", &val);
        temp = head;
        while (temp != NULL)
        {
            if (temp->data == val)
            {
                printf("\n%d found at location %d\n", val, i + 1);
                flag = 1;
                break;
            }
            i++;
            temp = temp->next;
        }
        if (!flag)
        {
            printf("\nValue %d not found\n", val);
        }
    }
}

void display()
{
    if (head == NULL)
    {
         printf("Circular Linked List is empty\n");
    }
    else
    {
        temp = head;
        printf("\nCircular Linked List: ");
        do
        {
            printf("%d -> ", temp->data);
            temp = temp->next;
        }
        while (temp != head);
    }
}
```



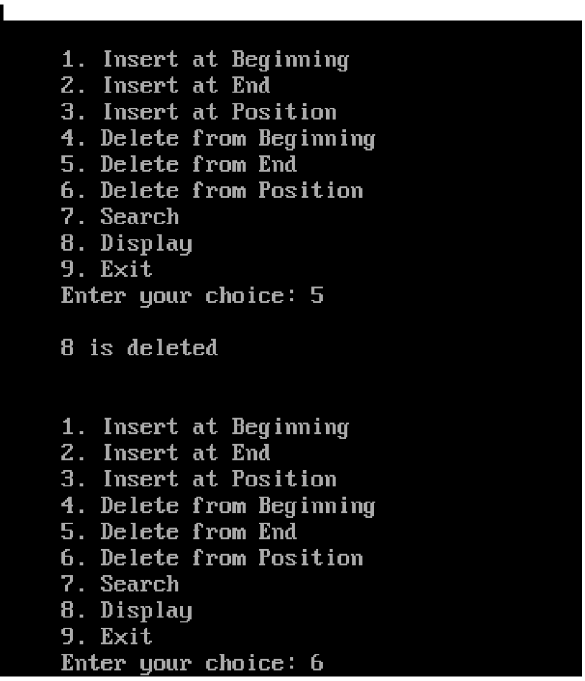
```
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 1

Enter Value: 6

Node inserted at the beginning


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. Search
8. Display
9. Exit
Enter your choice: 2_
```



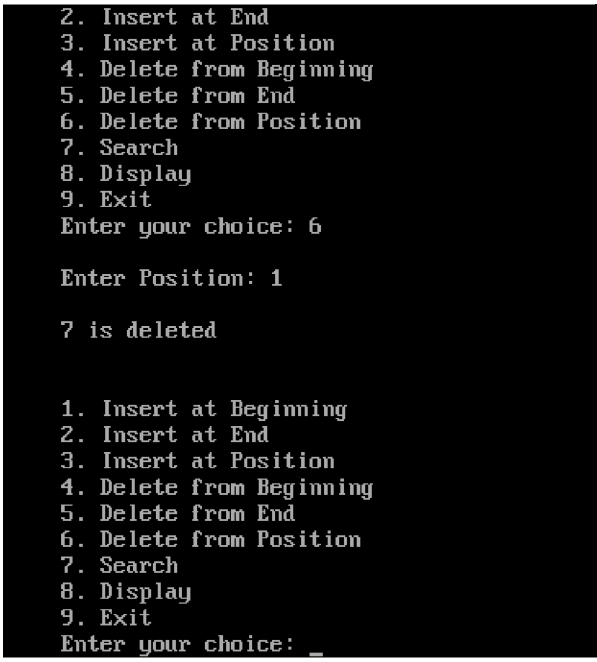
```
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 2

Enter Value: 8

Node inserted at the end


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. Search
8. Display
9. Exit
Enter your choice: 3

Enter location: 2

Enter value: 7
```



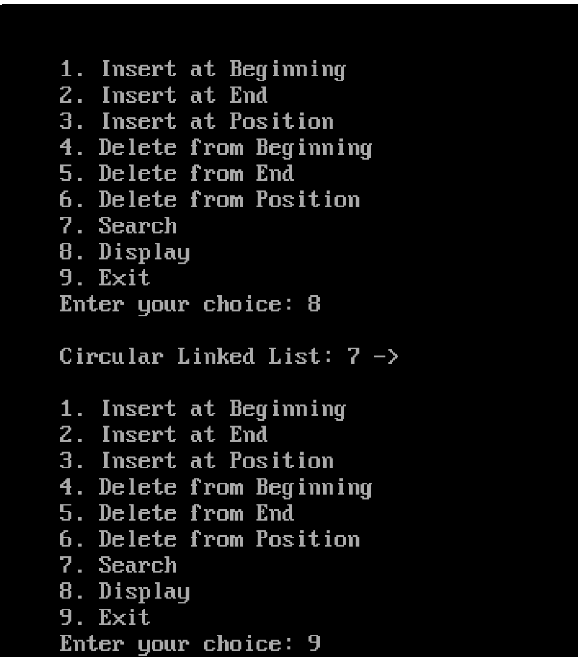
```
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 4

6 is deleted


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. Search
8. Display
9. Exit
Enter your choice: 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. Search
8. Display
9. Exit
Enter your choice: 5

8 is deleted


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. Search
8. Display
9. Exit
Enter your choice: 6
```



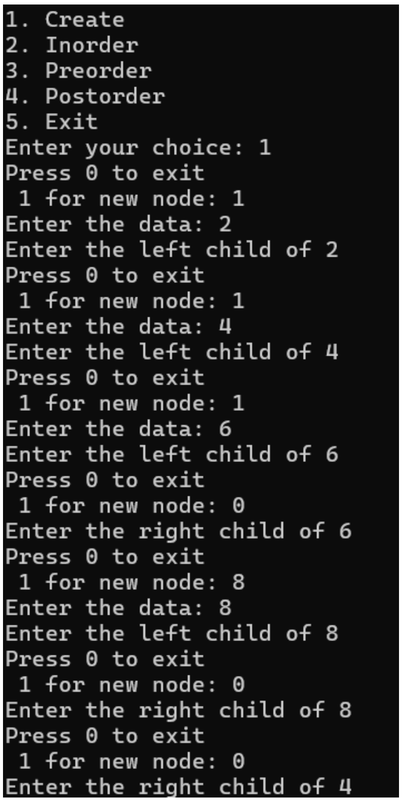
```
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
9. Exit
Enter your choice: 6

Enter Position: 1

7 is deleted


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. Search
8. Display
9. Exit
Enter your choice: _
```



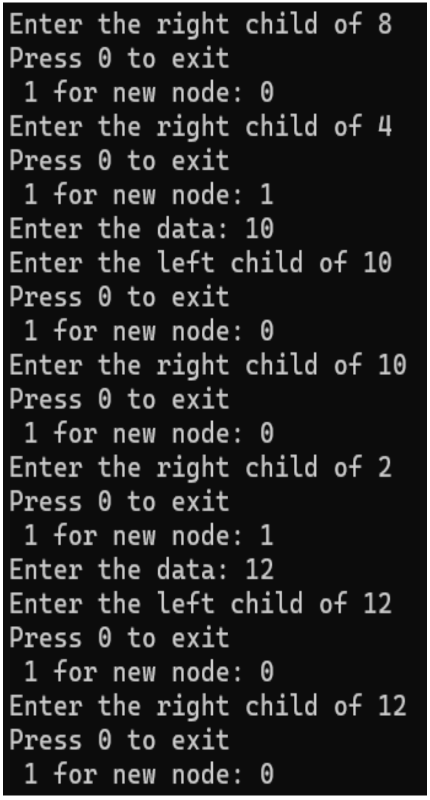
```

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. Search
8. Display
9. Exit
Enter your choice: 8

Circular Linked List: 7 ->

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. Search
8. Display
9. Exit
Enter your choice: 9
```

Write a program to implement Binary Tree.

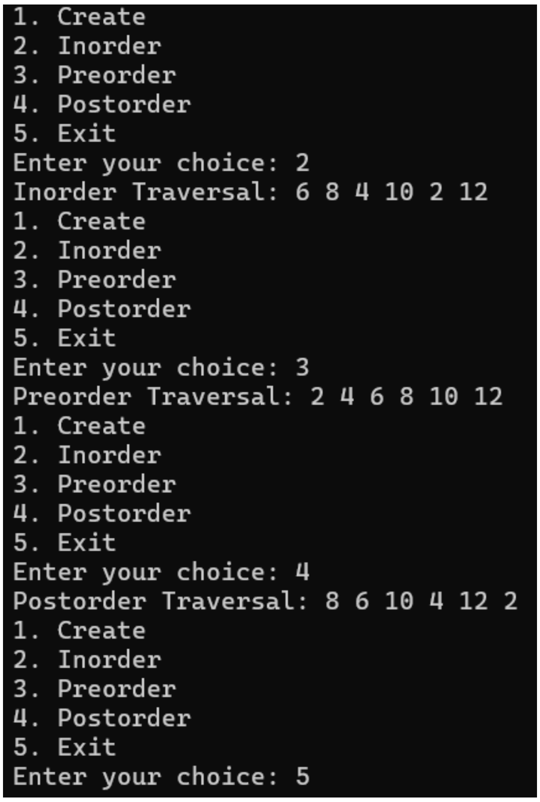
```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
        int data;
        struct node *left;
        struct node *right;
};
struct node *root= NULL;
void main()
 {
        int ch;
        while(1)
        {
                printf(“1. Create\n 2. Inorder\n 3. Preorder\n 4. Postorder\n 5. Exit”);
                printf(“\n Enter your choice: “);
                scanf(“%d”,&ch);
                switch(ch)
                {
                        case 1: root = create();
                        break;
                        case 2: inorder(root);
                        break;
                        case 3: preorder(root);
                        break;
                        case 4: postorder(root);
                        break;
                        case 5: exit(0);
                        break;
                        default: printf(“\n Wrong Choice:”);
                }
        }
}

struct node *create()
{
        struct node *temp;
        int data;
        temp = (struct node *)malloc(sizeof(struct node));
        printf("Press 0 to exit");
        printf("\n Press 1 for new node");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        if(choice==0)
        {
                return 0;
        }
        else
        {
                printf("Enter the data:");
                scanf("%d", &data);
                temp->data = data;
                printf("Enter the left child of %d", data);
                temp->left = create();
                printf("Enter the right child of %d", data);
                temp->right = create();
                return temp;
        }
}
void pre_order_traversal(struct node* root)
{
        if(root != NULL)
        {
                printf("%d ",root->data);
                pre_order_traversal(root->leftChild);
                pre_order_traversal(root->rightChild);
        }
}
void inorder_traversal(struct node* root)
{
        if(root != NULL)
        {
                inorder_traversal(root->leftChild);
                printf("%d ",root->data);
                inorder_traversal(root->rightChild);
        }
}
void post_order_traversal(struct node* root)
 {
        if(root != NULL)
        {
                post_order_traversal(root->leftChild);
                post_order_traversal(root->rightChild);
                printf("%d ", root->data);
        }
}
```



```
1. Create
2. Inorder
3. Preorder
4. Postorder
5. Exit
Enter your choice: 1
Press 0 to exit
 1 for new node: 1
Enter the data: 2
Enter the left child of 2
Press 0 to exit
 1 for new node: 1
Enter the data: 4
Enter the left child of 4
Press 0 to exit
 1 for new node: 1
Enter the data: 6
Enter the left child of 6
Press 0 to exit
 1 for new node: 0
Enter the right child of 6
Press 0 to exit
 1 for new node: 8
Enter the data: 8
Enter the left child of 8
Press 0 to exit
 1 for new node: 0
Enter the right child of 8
Press 0 to exit
 1 for new node: 0
Enter the right child of 4
```



```
Enter the right child of 8
Press 0 to exit
 1 for new node: 0
Enter the right child of 4
Press 0 to exit
 1 for new node: 1
Enter the data: 10
Enter the left child of 10
Press 0 to exit
 1 for new node: 0
Enter the right child of 10
Press 0 to exit
 1 for new node: 0
Enter the right child of 2
Press 0 to exit
 1 for new node: 1
Enter the data: 12
Enter the left child of 12
Press 0 to exit
 1 for new node: 0
Enter the right child of 12
Press 0 to exit
 1 for new node: 0
```



```
1. Create
2. Inorder
3. Preorder
4. Postorder
5. Exit
Enter your choice: 2
Inorder Traversal: 6 8 4 10 2 12
1. Create
2. Inorder
3. Preorder
4. Postorder
5. Exit
Enter your choice: 3
Preorder Traversal: 2 4 6 8 10 12
1. Create
2. Inorder
3. Preorder
4. Postorder
5. Exit
Enter your choice: 4
Postorder Traversal: 8 6 10 4 12 2
1. Create
2. Inorder
3. Preorder
4. Postorder
5. Exit
Enter your choice: 5
```

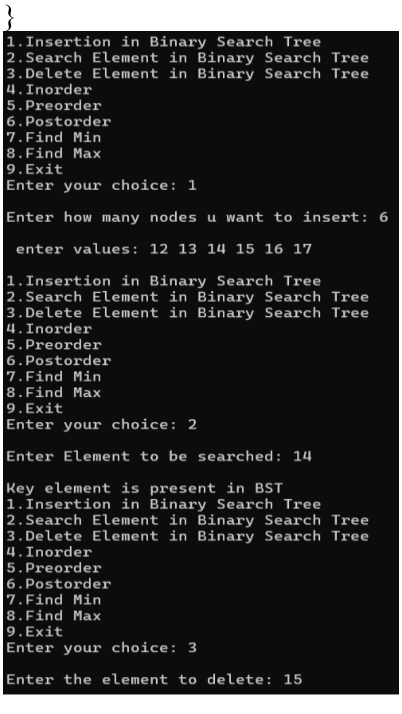
Write a program to create a binary search tree (BST)

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
        int data;
        struct node *left, *right;
};
struct node *createtree(struct node *root, int data);
void search(struct node *root);
void findmax(struct node *root);
struct node *delet(struct node *root, int data);
struct node *findmin(struct node *root);
void preorder(struct node *root);
void inorder(struct node *root);
void postorder(struct node *root);
struct node *root = NULL;
void main()
{
        struct node *temp;
        int data, ch, i, n;
        while(1)
        {
                printf("\n1.Insertion in Binary Search Tree");
                printf("\n2.Search Element in Binary Search Tree");
                printf("\n3.Delete Element in Binary Search Tree");
                printf("\n4.Inorder\n5.Preorder\n6.Postorder\n7.Find Min\n8.Find Max\n9.Exit");
                printf("\nEnter your choice: ");
                scanf("%d",&ch);
                switch (ch)
                {
                case 1: printf("\nEnter how many nodes u want to insert: " );
                        scanf("%d", &n);
                        printf("\n enter values: ");
                        for(i=0; i<n; i++)
                        {
                        scanf("%d", &data);
                        root=createtree(root, data);
                        }
                        break;
                case 2: search(root);
                        break;
                case 3: printf("\nEnter the element to delete: ");
                        scanf("%d", &data);
                        root=delet(root, data);
                        break;
                case 4: printf("\nInorder Traversal: \n");
                        inorder(root);
                        break;
                case 5: printf("\nPreorder Traversal: \n");
                preorder(root);
                        break;
                case 6: printf("\nPostorder Traversal: \n");
```

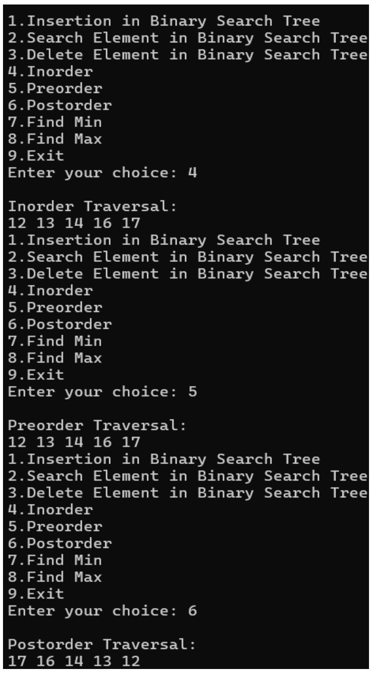
```
                        postorder(root);
                        break;
                case 7: temp=findmin(root);
                        printf("\n %d is minimum no in BST",temp->data);
                        break;
                case 8: findmax(root);
                break;
                case 9: exit(0);
                default: printf("WRONG CHOICE");
                        break;
                }
        }
}
struct node *createtree(struct node *root, int data)
{
        if (root == NULL)
        {
                struct node *temp;
                temp= (struct node*)malloc(sizeof(struct node));
                temp->data = data;
                temp->left = NULL;
                temp->right = NULL;
                return(temp);
        }
        if (data < (root->data))
        {
                root->left = createtree(root->left, data);
        }
        else if (data > root->data)
        {
                root->right = createtree(root->right, data);
        }
        return root;
}
void preorder(struct node *root)
{
        if(root != NULL)
        {
                printf("%d ",root->data);
                preorder(root->left);
                preorder(root->right);
        }
}
void inorder(struct node *root)
{
        if(root != NULL)
        {
                inorder(root->left);
                printf("%d ",root->data);
                inorder(root->right);
        }
}
void postorder(struct node *root)
{
        if(root != NULL)
```

```
		{
			postorder(root->left);
			postorder(root->right);
			printf("%d ", root->data);
		}
}
struct node *delet(struct node *root, int data)
{
		struct node *temp;
		if(root == NULL)
		{
			printf("\nElement not found");
		}
		else if(data < root->data)
		{
			root->left = delet(root->left, data);
		}
		else if(data > root->data)
		{
			root->right = delet(root->right, data);
		}
		else
		{
			if(root->right && root->left)
			{
			temp = findmin(root->right);
			root->data = temp->data;
			root->right = delet(root->right,temp->data);
			}
			else
			{
			temp = root;
			if(root->left == NULL)
				root = root->right;
			else if(root->right == NULL)
				root = root->left;
				free(temp); /* temp is longer required */
			}
		}
		return root;
}
 struct node *findmin(struct node *root)
{
		struct node *temp;
		temp = root;
		if(temp==NULL)
		{
			return NULL;
		}
		if(temp->left)
			return findmin(temp->left);
		else
			return temp;
}
void findmax(struct node *root)
{
		if(root==NULL)
		{
```

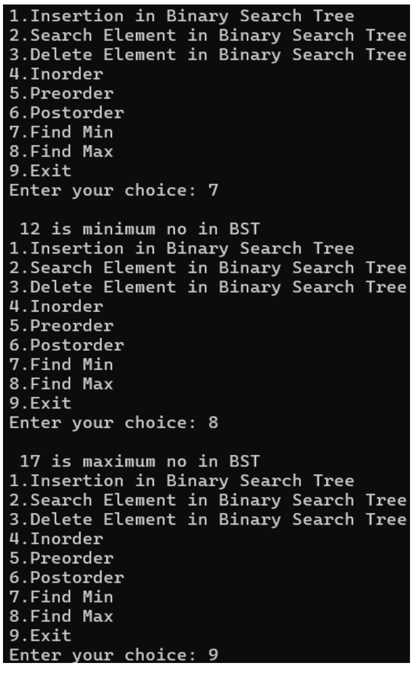
```
                return;
        }
        if(root->right)
                findmax(root->right);
        else
                printf("\n %d is maximum no in BST",root->data);
}
void search(struct node *root)
{
        int data;
        if(root == NULL)
        {
                printf("\nBST is empty.");
                return;
        }
        printf("\nEnter Element to be searched: ");
        scanf("%d", &data);
        while(root != NULL)
        {
                if (root->data == data)
                {
                printf("\nKey element is present in BST");
                return;
                }
                if (data < root->data)
                        root = root->left;
                else
                        root = root->right;
        }
        printf("\nKey element is not found in the BST");
}
```



```
1.Insertion in Binary Search Tree
2.Search Element in Binary Search Tree
3.Delete Element in Binary Search Tree
4.Inorder
5.Preorder
6.Postorder
7.Find Min
8.Find Max
9.Exit
Enter your choice: 1

Enter how many nodes u want to insert: 6

 enter values: 12 13 14 15 16 17

1.Insertion in Binary Search Tree
2.Search Element in Binary Search Tree
3.Delete Element in Binary Search Tree
4.Inorder
5.Preorder
6.Postorder
7.Find Min
8.Find Max
9.Exit
Enter your choice: 2

Enter Element to be searched: 14

Key element is present in BST
1.Insertion in Binary Search Tree
2.Search Element in Binary Search Tree
3.Delete Element in Binary Search Tree
4.Inorder
5.Preorder
6.Postorder
7.Find Min
8.Find Max
9.Exit
Enter your choice: 3

Enter the element to delete: 15
```



```
1.Insertion in Binary Search Tree
2.Search Element in Binary Search Tree
3.Delete Element in Binary Search Tree
4.Inorder
5.Preorder
6.Postorder
7.Find Min
8.Find Max
9.Exit
Enter your choice: 4

Inorder Traversal:
12 13 14 16 17
1.Insertion in Binary Search Tree
2.Search Element in Binary Search Tree
3.Delete Element in Binary Search Tree
4.Inorder
5.Preorder
6.Postorder
7.Find Min
8.Find Max
9.Exit
Enter your choice: 5

Preorder Traversal:
12 13 14 16 17
1.Insertion in Binary Search Tree
2.Search Element in Binary Search Tree
3.Delete Element in Binary Search Tree
4.Inorder
5.Preorder
6.Postorder
7.Find Min
8.Find Max
9.Exit
Enter your choice: 6

Postorder Traversal:
17 16 14 13 12
```



```
1.Insertion in Binary Search Tree
2.Search Element in Binary Search Tree
3.Delete Element in Binary Search Tree
4.Inorder
5.Preorder
6.Postorder
7.Find Min
8.Find Max
9.Exit
Enter your choice: 7

 12 is minimum no in BST
1.Insertion in Binary Search Tree
2.Search Element in Binary Search Tree
3.Delete Element in Binary Search Tree
4.Inorder
5.Preorder
6.Postorder
7.Find Min
8.Find Max
9.Exit
Enter your choice: 8

 17 is maximum no in BST
1.Insertion in Binary Search Tree
2.Search Element in Binary Search Tree
3.Delete Element in Binary Search Tree
4.Inorder
5.Preorder
6.Postorder
7.Find Min
8.Find Max
9.Exit
Enter your choice: 9
```

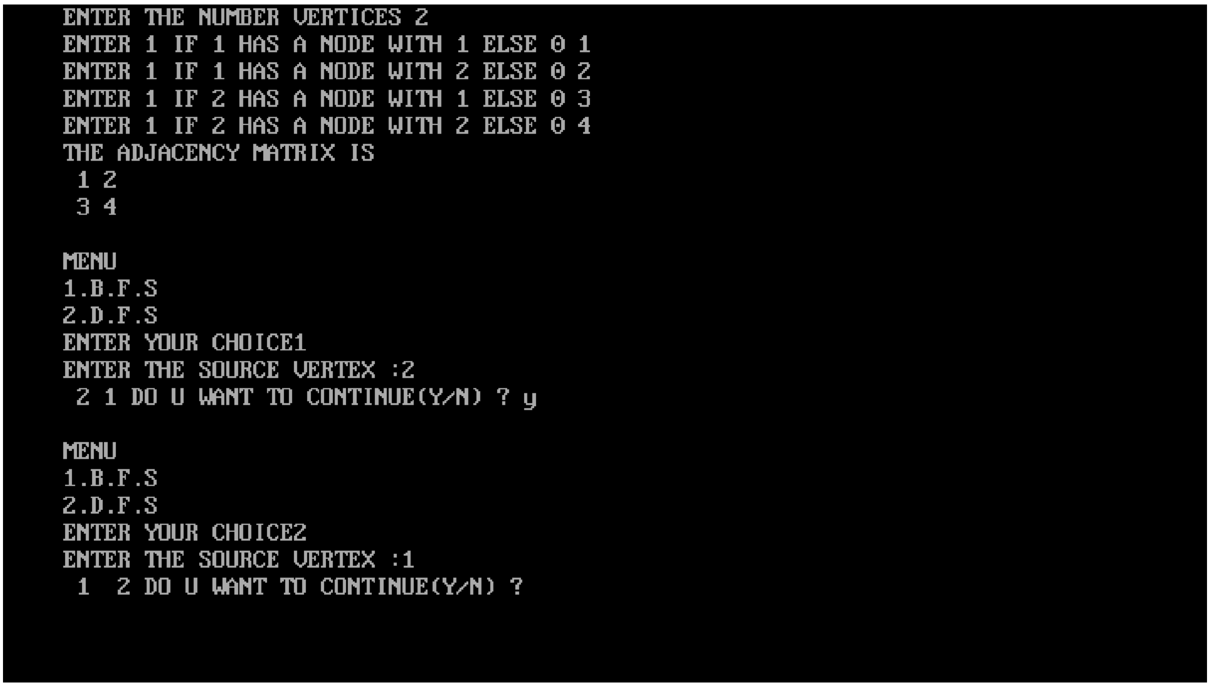
Write programs for implementation of graph traversals by applying:

(a) BFS (b) DFS

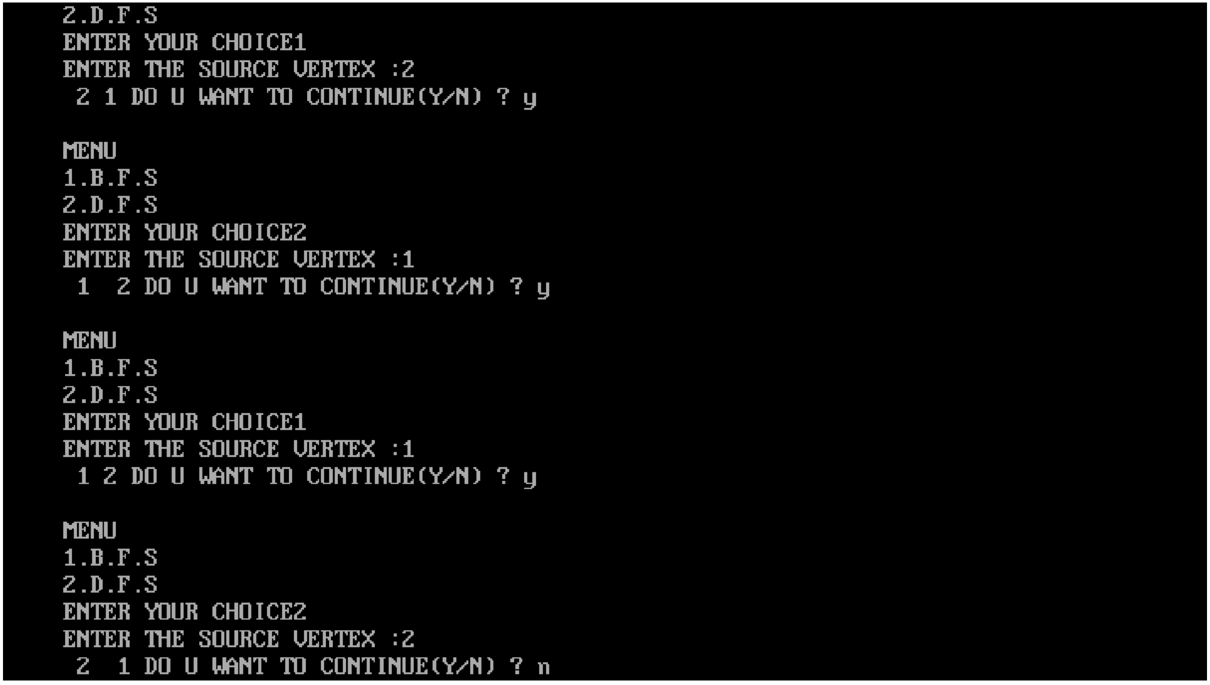
```
#include<stdio.h>
int q[20],top=-1,front=-1,rear=-1,a[20][20],vis[20],stack[20];
int delete();
void add(int item);
void bfs(int s,int n);
void dfs(int s,int n);
void push(int item);
int pop();

void main()
{
        int n,i,s,ch,j;
        char c,dummy;
        printf("ENTER THE NUMBER VERTICES ");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        printf("ENTER 1 IF %d HAS A NODE WITH %d ELSE 0 ",i,j);
                        scanf("%d",&a[i][j]);
                }
        }
        printf("THE ADJACENCY MATRIX IS\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
```

```
            printf(" %d",a[i][j]);
        }
        printf("\n");
    }
    do
    {
        for(i=1;i<=n;i++)
            vis[i]=0;
        printf("\nMENU");
        printf("\n1.B.F.S");
        printf("\n2.D.F.S");
        printf("\nENTER YOUR CHOICE");
        scanf("%d",&ch);
        printf("ENTER THE SOURCE VERTEX :");
        scanf("%d",&s);
        switch(ch)
        {
            case 1:bfs(s,n);
                  break;
            case 2:dfs(s,n);
                  break;
        }
        printf("DO U WANT TO CONTINUE(Y/N) ? ");
        scanf("%c",&dummy);
        scanf(" %c",&c);  // Added space before %c
    }while((c=='y')||(c=='Y'));
}

void bfs(int s,int n)
{
    int p,i;
    add(s);
    vis[s]=1;
    p=delete();
    if(p!=0)
        printf(" %d",p);
    while(p!=0)
    {
        for(i=1;i<=n;i++)
            if((a[p][i]!=0)&&(vis[i]==0))
            {
                    add(i);
                    vis[i]=1;
            }
            p=delete();
            if(p!=0)
                    printf(" %d ",p);
    }
    for(i=1;i<=n;i++)
        if(vis[i]==0)
            bfs(i,n);
}

void add(int item)
{
    if(rear==19)
        printf("QUEUE FULL");
    else
    {
        if(rear==-1)
        {
            q[++rear]=item;
            front++;
        }
        else
            q[++rear]=item;
    }
}

int delete()
{
    int k;
    if((front>rear)||(front==-1))
        return(0);
    else
    {
         k=q[front++];
         return(k);
    }
}

void dfs(int s,int n)
{
    int i,k;
    push(s);
    vis[s]=1;
    k=pop();
    if(k!=0)
        printf(" %d ",k);
    while(k!=0)
    {
        for(i=1;i<=n;i++)
            if((a[k][i]!=0)&&(vis[i]==0))
            {
                push(i);
                vis[i]=1;
            }
            k=pop();
            if(k!=0)
                printf(" %d ",k);
    }
    for(i=1;i<=n;i++)
        if(vis[i]==0)
            dfs(i,n);
}

void push(int item)
{
    if(top==19)
        printf("Stack overflow ");
    else
        stack[++top]=item;
}
int pop()
{
    int k;
    if(top==-1)
        return(0);
    else
    {
```

```
                k=stack[top--];

                return(k);

        }

}
```



```
    ENTER THE NUMBER VERTICES 2
    ENTER 1 IF 1 HAS A NODE WITH 1 ELSE 0 1
    ENTER 1 IF 1 HAS A NODE WITH 2 ELSE 0 2
    ENTER 1 IF 2 HAS A NODE WITH 1 ELSE 0 3
    ENTER 1 IF 2 HAS A NODE WITH 2 ELSE 0 4
    THE ADJACENCY MATRIX IS
     1 2
     3 4

    MENU
    1.B.F.S
    2.D.F.S
    ENTER YOUR CHOICE1
    ENTER THE SOURCE VERTEX :2
     2 1 DO U WANT TO CONTINUE(Y/N) ? y

    MENU
    1.B.F.S
    2.D.F.S
    ENTER YOUR CHOICE2
    ENTER THE SOURCE VERTEX :1
     1  2 DO U WANT TO CONTINUE(Y/N) ?
```



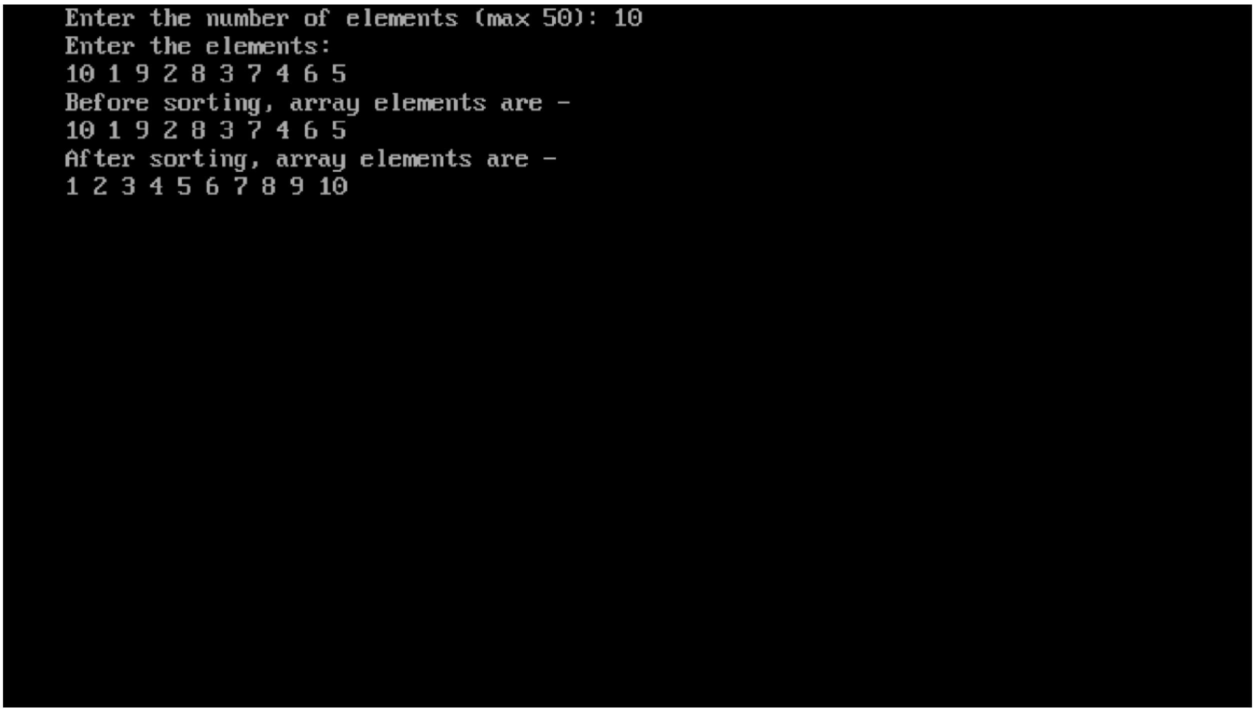
```
    2.D.F.S
    ENTER YOUR CHOICE1
    ENTER THE SOURCE VERTEX :2
     2 1 DO U WANT TO CONTINUE(Y/N) ? y

    MENU
    1.B.F.S
    2.D.F.S
    ENTER YOUR CHOICE2
    ENTER THE SOURCE VERTEX :1
     1  2 DO U WANT TO CONTINUE(Y/N) ? y

    MENU
    1.B.F.S
    2.D.F.S
    ENTER YOUR CHOICE1
    ENTER THE SOURCE VERTEX :1
     1 2 DO U WANT TO CONTINUE(Y/N) ? y

    MENU
    1.B.F.S
    2.D.F.S
    ENTER YOUR CHOICE2
    ENTER THE SOURCE VERTEX :2
     2  1 DO U WANT TO CONTINUE(Y/N) ? n
```

Implement the following sorting algorithms:

(a) Insertion sort

```
#include <stdio.h>
#include <conio.h>  // Required for clrscr() and getch()
void insert(int a[], int n) /* function to sort an array with insertion sort */
{
        int i, j, temp;
        for (i = 1; i < n; i++)
        {
                temp = a[i];
                j = i - 1;

                while (j >= 0 && temp <= a[j])
                {
                        a[j + 1] = a[j];
                        j = j - 1;
                }
                a[j + 1] = temp;
        }
}

void printArr(int a[], int n) /* function to print the array */
{
        int i;
        for (i = 0; i < n; i++)
                printf("%d ", a[i]);
}

void main()
```

```
{
    int a[50];  // Fixed size array for Turbo C compatibility
    int n, i;
    clrscr();  // Clear screen at start
    printf("Enter the number of elements (max 50): ");
    scanf("%d", &n);

    if (n > 50)
    {
        printf("Number of elements should not exceed 50.");
        getch();  // Wait for key press before exit
        return;
    }

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

    printf("Before sorting, array elements are - \n");
    printArr(a, n);
    insert(a, n);
    printf("\nAfter sorting, array elements are - \n");
    printArr(a, n);

    getch();  // Wait for key press before closing
}
```



```
Enter the number of elements (max 50): 10
Enter the elements:
10 1 9 2 8 3 7 4 6 5
Before sorting, array elements are -
10 1 9 2 8 3 7 4 6 5
After sorting, array elements are -
1 2 3 4 5 6 7 8 9 10
```

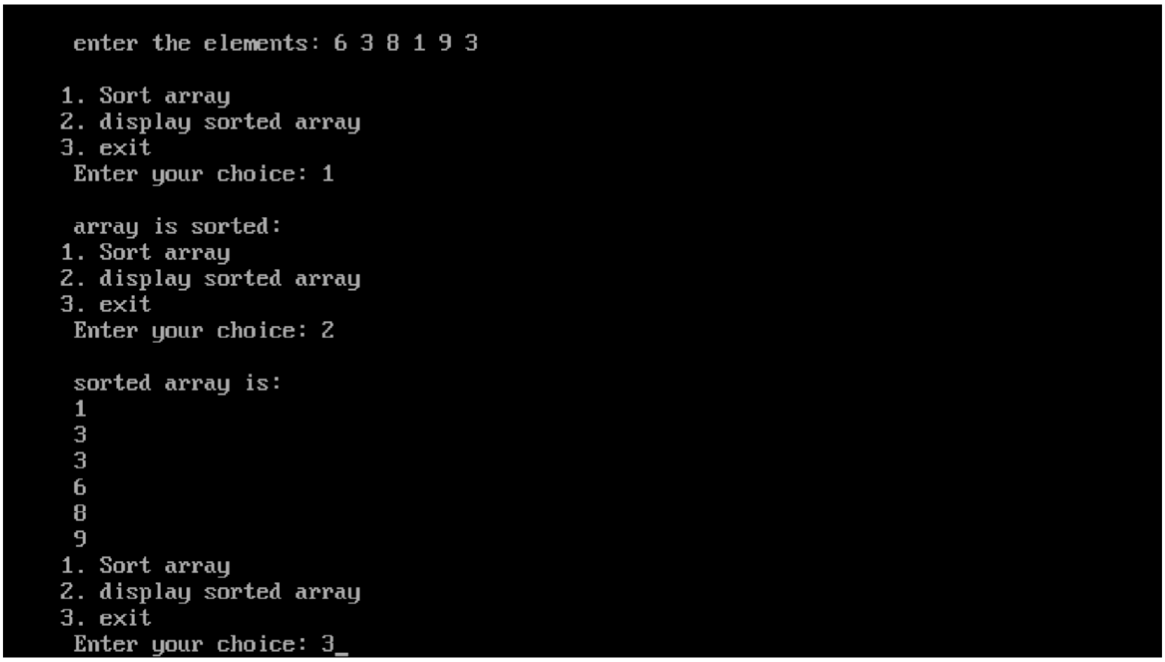
Implement the following sorting algorithms:

(b) Selection Sort

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void selection_sort(int a[],int n);
void display(int a[], int n);
int a[10], n;
void main()
{
	int i,ch;
	clrscr();
	printf("\n enter how many elements u want to insert: ");
	scanf("%d",&n);
	printf("\n enter the elements: ");
	for(i=0;i<n;i++)
	{
		scanf("%d",&a[i]);
	}
	while(1)
	{
		printf("\n1. Sort array");
		printf("\n2. display sorted array");
		printf("\n3. exit");
		printf("\n Enter your choice: ");
		scanf("%d", &ch);
		switch(ch)
		{
			case 1: selection_sort(a,n);
```

```
            printf("\n array is sorted: ");
            break;
            case 2: display(a,n);
            break;
            case 3: exit(0);
            default: printf("\n wrong choice: ");
        }
    }
}
void selection_sort(int a[], int n)
{
    int i,j,small, temp;
    for(i=0;i<n-1;i++)
    {
        small=i;
        for(j=i+1;j<n;j++)
        {
            if(a[j]<a[small])
            {
                small=j;
            }
        }
        temp=a[small];
        a[small]=a[i];
        a[i]=temp;
    }
}
void display(int a[], int n)
{
    int i;
```

```
        printf("\n sorted array is: ");
        for(i=0;i<n;i++)
        {
                printf("\n %d",a[i]);
        }
}
```



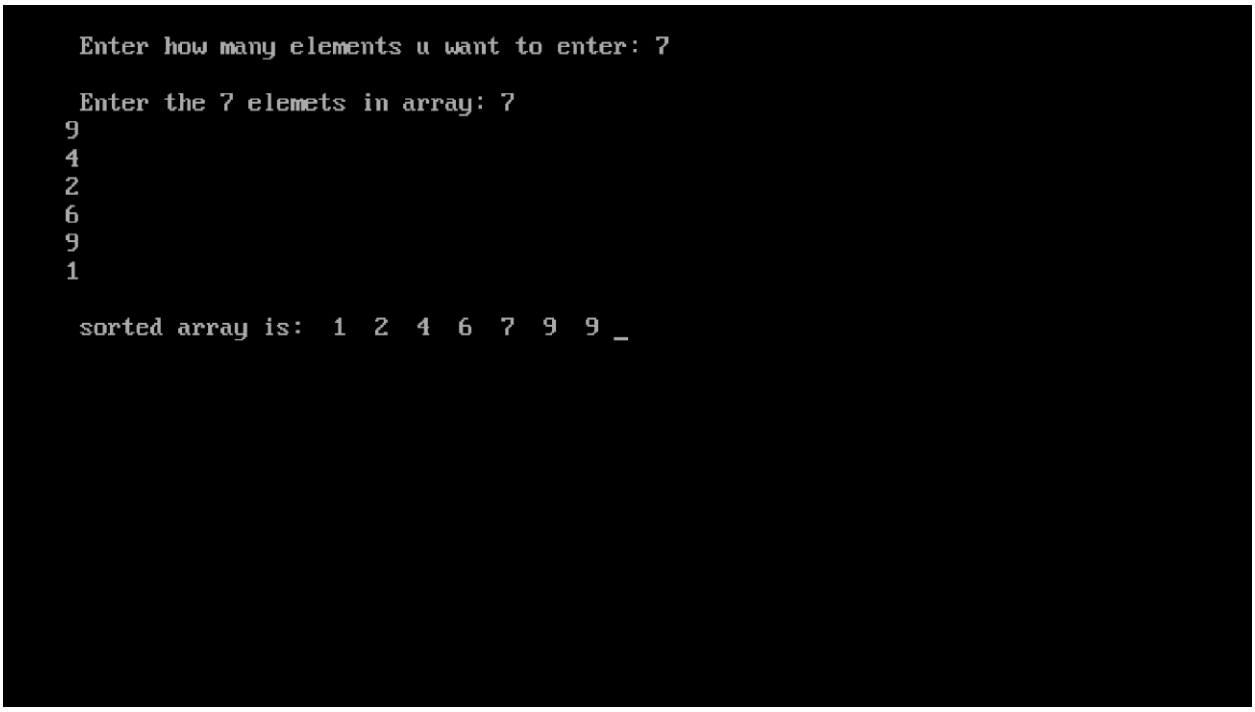
```
 enter the elements: 6 3 8 1 9 3

1. Sort array
2. display sorted array
3. exit
 Enter your choice: 1

 array is sorted:
1. Sort array
2. display sorted array
3. exit
 Enter your choice: 2

 sorted array is:
 1
 3
 3
 6
 8
 9
1. Sort array
2. display sorted array
3. exit
 Enter your choice: 3_
```

Implement the following sorting algorithms:

(a) Quick sort

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void quicksort(int a[], int first, int last);
void main()
{
        int i,n,a[20];
        clrscr();
        printf("\n Enter how many elements u want to enter: ");
        scanf("%d",&n);
        printf("\n Enter the %d elemets in array: ",n);
        for(i=0;i<n;i++)
        {
                scanf("%d", &a[i]);
        }
        quicksort(a,0,n-1);
        printf("\n sorted array is: ");
        for(i=0;i<n;i++)
        {
                printf(" %d ",a[i]);
        }
        getch();
}

void quicksort(int a[], int first, int last)
{
        int i,j,pivot,temp;
        if(first<last)
```

```
    {
        pivot=first;
        i=first;
        j=last;
        while(i<j)
        {
                while(a[i]<=a[pivot] && i<last)
                {
                    i++;
                }
                while(a[j]>a[pivot])
                {
                j--;
                }
                if(i<j)
                {
                temp=a[i];
                a[i]=a[j];
                a[j]=temp;
                }
         }
        temp=a[pivot];
        a[pivot]=a[j];
        a[j]=temp;
        quicksort(a,first,j-1);
        quicksort(a,j+1,last);
    }
}
```



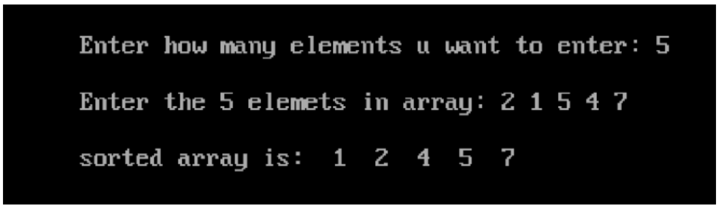
```
 Enter how many elements u want to enter: 7

 Enter the 7 elemets in array: 7
9
4
2
6
9
1

 sorted array is:  1  2  4  6  7  9  9 _
```

Implement the following sorting algorithms:

(b)Merge sort

```
#include<stdio.h>
#include<conio.h>
void mergesort(int a[], int lb, int ub);
void merge(int a[], int lb, int mid, int ub);
void main()
{
        int i,n,a[20];
        clrscr();
        printf("\n Enter how many elements u want to enter: ");
        scanf("%d",&n);
        printf("\n Enter the %d elemets in array: ",n);
        for(i=0;i<n;i++)
        {
                scanf("%d", &a[i]);
        }
        mergesort(a,0,n-1);
        printf("\n sorted array is: ");
        for(i=0;i<n;i++)
        {
                printf(" %d ",a[i]);
        }
        getch();
}
void mergesort(int a[], int lb, int ub)
{
        int mid;
        if(lb<ub)
        {
                mid=(lb+ub)/2;
                mergesort(a,lb,mid);
                mergesort(a, mid+1,ub);
                merge(a,lb,mid,ub);
        }
}
void merge(int a[], int lb, int mid, int ub)
{
        int i,j,k;
        int b[20];
        i=lb;
        j=mid+1;
        k=lb;
        while(i<=mid && j<=ub)
        {
                if(a[i]<=a[j])
                {
                b[k]=a[i];
                k++;
                i++;
                }
                else
                {
                b[k]=a[j];
                j++;
                k++;
                }
        }
        while(i<=mid)
        {
                b[k]=a[i];
                i++;
```

```
		k++;
	}
	while(j<=ub)
	{
		b[k]=a[j];
		k++;
		j++;
	}
	for(k=lb;k<=ub;k++)  // Changed from k=0 to k=lb
	{
		a[k]=b[k];
	}
}
```



```
Enter how many elements u want to enter: 5

Enter the 5 elemets in array: 2 1 5 4 7

sorted array is:  1  2  4  5  7
```