1. **Write a program to implement circular queue using array**

```
#include<stdio.h>
#define MAX 5
void enqueue(int x);
void dequeue();
void display();
void exit();
int queue[MAX],front=-1,rear=-1;
int main()
{
 int ch,x;
 while(1)
 {
 printf("\n1.enqueue\n2.dequeue\n3.display\n4.exit\n");
 scanf("%d",&ch);
 switch(ch)
 {
 case 1: printf("enter value to insert ");
 scanf("%d",&x);
enqueue(x);
 break;
 case 2:dequeue();
 break;
 case 3:display();
 break;
 case 4: exit();
 break;
 default:printf("\nwrong choice");
 }
 }
```

```
}
void enqueue(int x)
{
if(front==-1&&rear==-1)
{
 front=rear=0;
 queue[rear]=x;
}
else if((rear+1)%MAX==front)
{
 printf("queue is full");
}
else
{
 rear=(rear+1)%MAX;
 queue[rear]=x;
}
}
void dequeue()
{
 if(front==-1&&rear==-1)
 {
 printf("queue is empty");
 }
 else if(front==rear)
 {
 printf("%d is deleted",queue[front]);
 front=rear=-1;
 }
 else
 {
 printf("%d is deleted",queue[front]);
 front=(front+1)%MAX;
 }
}
void display()
{
 int i=front;
 if(front==-1&&rear==-1)
 {
 printf("queue is empty");
 }
 else
 {
 while(i<=rear)
 {
 printf("\n%d",queue[i]);
 i=(i+1)%MAX;
 }
 }
}
```

****
*******OUTPUT

```
1.enqueue
2.dequeue
3.display
4.exit
1
enter value to insert 23
```

```
1.enqueue
2.dequeue
3.display
4.exit
1
enter value to insert 24

1.enqueue
2.dequeue
3.display
4.exit
3
23
24
1.enqueue
2.dequeue
3.display
4.exit
2
23 is deleted
1.enqueue
2.dequeue
3.display
4.exit
2
24 is deleted
1.enqueue
2.dequeue
3.display
4.exit
```

2. **Write the program for double ended queue**

```
#include<stdio.h>
#include <stdlib.h>
int deque[10];
int front = -1, rear = -1;

#define MAX 10
int front,rear;
void Insert_from_Front();
void Insert_from_Rear();
void Delete_from_Front();
void Delete_from_Rear();
void Display();
void main()
{
  int ch,x;
  while(1)
  {
    printf("\nEnter the Choice");
    printf("\n1.Insert from Front\n2.Insert from Rear\n3.Delete from Front\n4.Delete From
Rear\n5.Display\n6.exit ");
    scanf("%d",&ch);
    switch(ch)
    {
      case 1:printf("\nEnter The Val");
          scanf("%d",&x);
          Insert_from_Front( x);
          break;
      case 2:Insert_from_Rear( x);
          break;
```

```
            case 3:Delete_from_Front();
                   break;
            case 4:Delete_from_Rear();
                   break;
            case 5:Display();
                   break;
            case 6:exit(0);
                   break;
            Default:printf("\nWrong Choice");

        }

    }
}
void Insert_from_Front(int x)
{
  if((front==0&&rear==MAX-1)||(front==rear+1))
  {
    printf("Queue is full");
  }
  else if(front==-1&&rear==-1)
  {
     front=0;
     rear=0;
     deque[front]=x;
  }
  else if(front==0)
  {
    front=MAX-1;
    deque[front]=x;
```

```
 }
 else
 {
   front=front-1;
   deque[front]=x;
 }
}
void Insert_from_Rear(int x)
{
 if((front==0&&rear==MAX-1)||(front==rear+1))
 {
   printf("Queue is full");
 }
 else if(front==-1&&rear==-1)
 {
    front=0;
    rear=0;
    deque[rear]=x;
 }
 else if(rear==MAX-1)
 {
  rear=0;
  deque[rear]=x;
 }
 else
 {
   rear++;
  deque [rear]=x;
 }
}
```

```
void Delete_from_Front()
{
 if(front==-1&&rear==-1)
 {
  printf("Queue is Empty");
 }
 else if(front=rear)
 {
  printf("%d is deleted",deque[front]);
  front=rear-1;
 }
 else if(rear==MAX-1)
 {
   printf("%d is deleted",deque[front]);
 front=0;
 }
 else
 {
  printf("%d is deleted",front);
  front++;
 }
}
void Delete_from_Rear()
{
 if(front==-1&&rear==-1)
 {
  printf("Queue is Empty");
 }
 else if(front=rear)
 {
    printf("%d is deleted",deque[rear]);
    front=rear-1;
  }
  else if(rear==MAX-1)
  {
     printf("%d is deleted",deque[rear]);
    rear--;
  }
  else
  {
    printf("%d is deleted",deque[rear]);
    rear=MAX-1;
  }
}
void Display()
{
    int i = front;
    if (front == -1 && rear == -1)
    {
        printf("Queue is Empty");
    }
    else
    {
        do
        {
            printf("%d\t", deque[i]);
            i = (i + 1) % MAX;
        } while (i != (rear + 1) % MAX);
    }
}
```

```
*************************************************************************
*******OUTPUT
Enter the Choice
1.Insert from Front
2.Insert from Rear
3.Delete from Front
4.Delete From Rear
5.Display
6.exit
1

Enter The Val12

Enter the Choice
1.Insert from Front
2.Insert from Rear
3.Delete from Front
4.Delete From Rear
5.Display
6.exit
1

Enter The Val23

Enter the Choice
1.Insert from Front
2.Insert from Rear
3.Delete from Front
4.Delete From Rear
5.Display
6.exit
```

```
1

Enter The Val34

Enter the Choice
1.Insert from Front
2.Insert from Rear
3.Delete from Front
4.Delete From Rear
5.Display
6.exit 5
34      23      12
Enter the Choice
1.Insert from Front
2.Insert from Rear
3.Delete from Front
4.Delete From Rear
5.Display
6.exit 2

Enter the Choice
1.Insert from Front
2.Insert from Rear
3.Delete from Front
4.Delete From Rear
5.Display
6.exit 5
34      23      12      34
Enter the Choice
1.Insert from Front
```

```
2.Insert from Rear
3.Delete from Front
4.Delete From Rear
5.Display
6.exit 3
34 is deleted
Enter the Choice
1.Insert from Front
2.Insert from Rear
3.Delete from Front
4.Delete From Rear
5.Display
6.exit 4
34 is deleted
Enter the Choice
1.Insert from Front
2.Insert from Rear
3.Delete from Front
4.Delete From Rear
5.Display
6.exit 5
12      34
Enter the Choice
1.Insert from Front
2.Insert from Rear
3.Delete from Front
4.Delete From Rear
5.Display
6.exit
```

3. Write a program to implement priority queue using array

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

void create_queue();
void insert_element(int);
void delete_element(int);
void check_priority(int);
void display_priorityqueue();

int pqueue[MAX] ;
int front, rear;

void main()
{

int n, choice;
printf("\n enter 1 to insert element by priority ");
printf("\n enter 2 to delete element by priority");
printf("\n enter 3 to display priority queue");
printf("\n enter 4 to exit");

create_queue();
while (1)
{

 printf("\nEnter your choice: ");
 scanf("%d", &choice);
 switch(choice)
 {
 case 1: printf("\nEnter element to insert: ");
        scanf("%d",&n);
        insert_element(n);
        break;

 case 2: printf("\nEnter element to delete: ");
        scanf("%d",&n);
        delete_element(n);
        break;

 case 3: display_priorityqueue();
        break;

 case 4:exit(0);

 default:printf("n Please enter valid choice");
 }
}
}
void create_queue()
{
 front=rear=-1;
}

void insert_element(int data)
{
 if (rear>= MAX-1)
 {
   printf("QUEUE OVERFLOW");
   return;
 }

if ((front==-1) && (rear==-1))
  {
      front++;
      rear++;
      pqueue[rear]= data;
      return;
  }
else
check_priority(data);
rear++;
}

void check_priority(int data)
{
int i,j;
for (i=0; i <=rear; i++)
 {
    if(data>=pqueue[i])
    {
        for(j=rear+1;j>i;j--)
        {
           pqueue[j]=pqueue[j-1];
        }
        pqueue[i]=data;
        return;
      }
   }
   pqueue[i]=data;
  }
  void delete_element(int data)
  {
     int i;
     if((front==-1)&&(rear==-1))
     {
        printf("\nEmpty Queue");
        return;
     }
     for(i=0;i<=rear;i++)
     {
        if(data==pqueue[i])
        {
           for(i=0;i<rear;i++)
           {
              pqueue[i]=pqueue[i+1];
           }
           pqueue[i]=data;
           rear--;
           if(rear==-1)
           front=-1;
           return;
        }
     }
     printf("\n%d element not found in queue ",data);
  }
```

```
void display_priorityqueue()
{
   if((front==-1)&&(rear==-1))
   {
      printf("\nEmpty Queue");
      return;
   }
   for(front=0;front<=rear;front++)
   {
       printf("%d\n",pqueue[front]);
   }
   front=0;
}
```

```
OUTPUT
enter 1 to insert element by priority
 enter 2 to delete element by priority
 enter 3 to display priority queue
 enter 4 to exit
Enter your choice: 1

Enter element to insert: 23

Enter your choice: 1

Enter element to insert: 34

Enter your choice: 1

Enter element to insert: 12
Enter your choice: 1

Enter element to insert: 16

Enter your choice: 3
34
23
16
12

Enter your choice: 2

Enter element to delete: 34

Enter your choice: 3
23
16
12

Enter your choice: 2

Enter element to delete: 16

Enter your choice: 3
16
12

Enter your choice:
```

4. Write a c program to covert a given infix expression to postfix expression from using stack

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

#define SIZE 100

char stack[SIZE];
int top = -1;

void push(char item)
{
        if(top >= SIZE-1)
        {
                printf("\nStack Overflow.");
        }
        else
        {
                top = top+1;
                stack[top] = item;
        }
}

char pop()
{
        char item ;
```

```
	if(top <0)
	{
		printf("stack under flow: invalid infix expression");
		getchar();
		exit(1);
	}
	else
	{
		item = stack[top];
		top = top-1;
		return(item);
	}
}

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


int precedence(char symbol)
{
	if(symbol == '^')
```

```
        {
                return(3);
        }
        else if(symbol == '*' || symbol == '/')
        {
                return(2);
        }
        else if(symbol == '+' || symbol == '-')
        {
                return(1);
        }
        else
        {
                return(0);
        }
}

void InfixToPostfix(char infix_exp[], char postfix_exp[])
{
        int i, j;
        char item;
        char x;

        push('(');
        strcat(infix_exp,")");

        i=0;
        j=0;
        item=infix_exp[i];
```

```
    while(item != '\0')
    {
            if(item == '(')
            {
                    push(item);
            }
            else if( isdigit(item) || isalpha(item))
            {
                    postfix_exp[j] = item;
                    j++;
            }
            else if(is_operator(item) == 1)
            {
                    x=pop();
                    while(is_operator(x) == 1 && precedence(x)>= precedence(item))
                    {
                            postfix_exp[j] = x;
                            j++;
                            x = pop();
                    }
                    push(x);

                    push(item);
  }
            else if(item == ')')
            {
                    x = pop();
                    while(x != '(')
                    {
                            postfix_exp[j] = x;
                    j++;
                    x = pop();
                }
            }
            else
            {
                printf("\nInvalid infix Expression.\n");
                getchar();
                exit(1);
            }
            i++;


            item = infix_exp[i];
        }
        if(top>0)
        {
            printf("\nInvalid infix Expression.\n");
            getchar();
            exit(1);
        }
        if(top>0)
        {
            printf("\nInvalid infix Expression.\n");
            getchar();
            exit(1);
        }



        postfix_exp[j] = '\0';
```

```
}
int main()
{
	char infix[SIZE], postfix[SIZE];
	printf("\nEnter Infix expression : ");
	gets(infix);

	InfixToPostfix(infix,postfix);
	printf("Postfix Expression: ");
	puts(postfix);

	return 0;
}
```

OUTPUT

```
Enter Infix expression : (a+b-(c*d*f)/g-h)
Postfix Expression: ab+cd*f*g/-h-
```

5. Write a program to evaluate a given postfix expression using stack

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#defne MAX 50
int stack[MAX];
char post[MAX];
int top=-1;
void pushstack(int tmp);
void evaluate(char c);
void main()
{
  int i,l;
  printf("Insert a postfix notation :: ");
  scanf("%s",post);
  l=strlen(post);
  for(i=0;i<l;i++)
  {
    if(post[i]>='0' && post[i]<='9')
    {
       pushstack(i);
    }
    if(post[i]=='+' || post[i]=='-' || post[i]=='*' || post[i]=='/' || post[i]=='^')
    {
       evaluate(post[i]);
    }
  }
  printf("\n\nResult :: %d",stack[top]);
  getch();
}
```

```
void pushstack(int tmp)
{
  top++;
  stack[top]=(int)(post[tmp]-48);
}

void evaluate(char c)
{
  int a,b,ans;
  a=stack[top];
  stack[top]='\0';
  top--;
  b=stack[top];
  stack[top]='\0';
  top--;
  switch(c)
  {
    case '+':
       ans=b+a;
       break;
    case '-':
       ans=b-a;
       break;
    case '*':
       ans=b*a;
       break;
    case '/':
       ans=b/a;
       break;
    case '^':
        ans=b^a;
        break;
    default:
        ans=0;
  }
  top++;
  stack[top]=ans;
}
```

```
********************************************************************************
*******OUTPUT
Insert a postfix notation :: 5 4 *6 7+-



Result :: 7
```

6. **Write a C-program of implementing stack using two queues:**

```
#include <stdio.h>
#include<conio.h>
#define N 20
int queue1[N],queue2[N];
int f1= -1, r1= -1;
int f2= -1, r2= -1;
int count=0;
void enqueue1(int x);
int dequeue1();
void enqueue2(int x);
int dequeue2();
void push(int x);
int pop();
void display();
void main()
{
 int ch, num;
 while (ch != 4)
 {
    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);
              break;
```

```
        case 2: printf("Item deleted : %d",pop());
                break;
        case 3: display();
                break;
        case 4: exit(0);
                break;
        default:printf("\nInvalide Choice !!!\n");
      }
   }
}

void enqueue1(int x)
{
  if(r1==N-1)
  {
     printf("Overflow");
  }
  else
  {
     if(f1== -1)
     {
          f1=0;
     }
     r1=r1+1;
     queue1[r1]=x;
  }
}
int dequeue1()
{
  int temp;
  if(f1== -1 || f1 > r1)
  {
     printf("underflow");
  }
  else
  {
     temp = queue1[f1];
     f1++;
  }
  return(temp);
}
void enqueue2(int x)
{
  if(r2==N-1)
  {
    printf("Overflow");
  }
  else
  {
    if(f2== -1)
    {
       f2=0;
    }
    r2=r2+1;
    queue2[r2]=x;
 }
}
int dequeue2()
{
  int temp;
  if(f2== -1 || f2 > r2)
  {
    printf("Underflow");
  }
  else
  {
    temp = queue2[f2];
    f2++;
  }
  return(temp);
}

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());
 }
}
int pop()
{
 count--;
 return dequeue2();
```

```
}
void display()
{
  int i;
  printf("\nElements in Stack : ");
  for (i = f2; i <=r2 ; i++)
 {
    printf("%d ", queue2[i]);
 }
 printf("\n");
}
```

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

Enter your choice :1
Entre item to be inserted : 12

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

Enter your choice :1
Entre item to be inserted : 34

1.Push Item
2.Pop Item
```

```
3.Display Item
4.Exit

Enter your choice :1
Entre item to be inserted : 45

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

Enter your choice :1
Entre item to be inserted : 43

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

Enter your choice :3

Elements in Stack : 43 45 34 12

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

Enter your choice :2
Item deleted : 43
```

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

Enter your choice :2
Item deleted : 45
1.Push Item
2.Pop Item
3.Display Item
4.Exit

Enter your choice :2
Item deleted : 34
1.Push Item
2.Pop Item
3.Display Item
4.Exit

Enter your choice :2
Item deleted : 12
1.Push Item
2.Pop Item
3.Display Item
4.Exit
```

7. Write a program to implement queue using two stack

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int data;
   struct node *next;
};
void push(struct node** top, int data);
int pop(struct node** top);
struct queue
{
   struct node *stack1;
   struct node *stack2;
};
void enqueue(struct queue *q, int x)
{
   push(&q->stack1, x);
}
void dequeue(struct queue *q)
{
   int x;
   if (q->stack1 == NULL && q->stack2 == NULL) {
      printf("queue is empty");
      return;
   }
   if (q->stack2 == NULL) {
      while (q->stack1 != NULL) {
      x = pop(&q->stack1);
```

```
        push(&q->stack2, x);
        }
    }
    x = pop(&q->stack2);
    printf("%d\n", x);
}
void push(struct node** top, int data)
{
    struct node* newnode = (struct node*) malloc(sizeof(struct node));
        if (newnode == NULL) {
            printf("Stack overflow \n");
            return;
        }
    newnode->data = data;
    newnode->next = (*top);
    (*top) = newnode;
}
int pop(struct node** top)
{
    int buff;
    struct node *t;
    if (*top == NULL) {
        printf("Stack underflow \n");
        return;
    }
    else {
        t = *top;
        buff = t->data;
        *top = t->next;
        free(t);
        return buff;
    }
}
void display(struct node *top1,struct node *top2)
{
    while (top1 != NULL) {
        printf("%d\n", top1->data);
        top1 = top1->next;
    }
    while (top2 != NULL) {
        printf("%d\n", top2->data);
        top2 = top2->next;
    }
}
int main()
{
    struct queue *q = (struct queue*)malloc(sizeof(struct queue));
    int f = 0, a;
    char ch = 'y';
    q->stack1 = NULL;
    q->stack2 = NULL;
    while (ch == 'y'||ch == 'Y') {
        printf("enter ur choice\n1.add to queue\n2.remove from queue\n3.display\n4.exit\n");
        scanf("%d", &f);
        switch(f) {
            case 1 : printf("enter the element to be added to queue\n");
                     scanf("%d", &a);
                     enqueue(q, a);
                     break;
            case 2 : dequeue(q);
```

```
                break;
            case 3 : display(q->stack1, q->stack2);
                break;
            case 4 : exit(1);
                break;
            default : printf("invalid\n");
                 break;
        }
    }
}
```

OUTPUT

```
enter ur choice
1.add to queue
2.remove from queue
3.display
4.exit
1
enter the element to be added to queue
12
enter ur choice
1.add to queue
2.remove from queue
3.display
4.exit
1
enter the element to be added to queue
23
enter ur choice
1.add to queue
2.remove from queue
```

```
3.display
4.exit
1
enter the element to be added to queue
13
enter ur choice
1.add to queue
2.remove from queue
3.display
4.exit
1
enter the element to be added to queue
67
enter ur choice
1.add to queue
2.remove from queue
3.display
4.exit
3
67
13
23
12
enter ur choice
1.add to queue
2.remove from queue
3.display
4.exit
2
12
```

```
enter ur choice
1.add to queue
2.remove from queue
3.display
4.exit
2
23
enter ur choice
1.add to queue
2.remove from queue
3.display
4.exit
2
13
```

Write programme to implement the following data structure.

1. **Singly link list**

```
#include<stdio.h>
#include<stdlib.h>
struct node{
        int data;
        struct node*next;
};
struct node*head;
struct node*ptr;
struct node*temp;
void insert_begin(int val);
void insert_end(int val);
void insert_pos(int val);
void delete_begin();
void delete_end();
void delete_pos();
void display();
void exit();
int main()
{
        int ch,val;
        while(1)
        {
          printf("\n1.insert at begin \n2.insert at end \n3.insert at pos \n4.delete at begin
\n5.delete at end \n6.delete at pos \n7.display \n8.exit");
          printf("\nenter your choice");
          scanf("%d",&ch);
          switch(ch)
          {
```

```
                        case 1: printf("enter the value");
                                scanf("%d",&val);
                                insert_begin(val);
                                break;
                        case 2: insert_end(val);
                                    break;
                        case 3: insert_pos(val);
                                break;
                        case 4:  delete_begin();
                                break;
                        case 5:delete_end();
                                break;
                        case 6:delete_pos();
                                break;
                        case 7:display();
                                break;
                        case 8:exit(0);
                                    break;
                         defazult:printf("enter wrong choice");
                                 break;


                        }


                         }
                }
                void insert_begin(int val)
        {
           ptr = (struct node *)malloc(sizeof(struct node));
           if (ptr == NULL)
           {
```

```
        printf("memory is not allocated");
    }
    else
    {
        ptr->data = val;
        ptr->next = head; // Link the new node to the current head
        head = ptr;       // Make the new node the head of the list
    }
}

void insert_end(int val)
{
    ptr = (struct node *)malloc(sizeof(struct node));
    if (ptr == NULL)
    {
        printf("memory is not allocated");
    }
    else
    {
        ptr->data = val;
        ptr->next = NULL;

        if (head == NULL)
        {
            head = ptr;
        }
        else
        {
            temp = head;
            while (temp->next != NULL)
            {
               temp = temp->next;
            }
            temp->next = ptr;
        }
    }
}
                    void insert_pos(int val)
{
    int loc;
    ptr = (struct node *)malloc(sizeof(struct node));
    if (ptr == NULL)
    {
        printf("memory is not allocated");
    }
    else
    {
        printf("enter location where you want to insert");
        scanf("%d", &loc);

        if (loc == 1)
        {
            // Insert at the beginning
            ptr->data = val;
            ptr->next = head;
            head = ptr;
        }
        else
        {
            temp = head;
        for (int i = 0; i < loc - 2; i++) // Traverse to the node before the specified location
        {
          temp = temp->next;
          if (temp == NULL)
          {
            printf("cannot insert");
            free(ptr); // Free the allocated memory before returning
            return;
          }
        }

        ptr->data = val;
        ptr->next = temp->next;
        temp->next = ptr;
      }
    }
  }
                void delete_begin()
                {
                        temp=head;
                        if(head==NULL)
                {
                        printf("list is empty");
                }
                else
                {
                  head=temp->next;
                  temp->next=NULL;
                  printf("%d is deleted",temp->data);
                  free(temp);
```

```
                        }
                        }
        void delete_end()
    {
      struct node *temp1;
      struct node *ptr1;
      temp = head;

      if (head == NULL)
      {
        printf("list is empty");
      }
      else if (head->next == NULL)
      {
        printf("%d is deleted", temp->data);
        free(head);
        head = NULL;
      }
      else
      {
        while (temp->next != NULL)
        {
          ptr1 = temp;
          temp = temp->next;
        }

        ptr1->next = NULL;
        printf("%d is deleted", temp->data);
        free(temp);
      }
```

```
}
        void delete_pos()
                {
                        struct node*temp1;
                        temp=head;
                        int loc;
                        if(head==NULL)
                {
                        printf("cannot delete");
                }
                else
                {
                        printf("enter location");
                        scanf("%d",&loc);
                        for(int i=0;i<loc-1;i++)
                        {
                                temp1=temp;
                                temp=temp->next;
                                if(temp==NULL)
                                {
                                        printf("cannot delete");

                                }
                        }
                        temp1->next=temp->next;
                        temp->next=NULL;
                        printf("%d is deleted",temp->data);
                        free(temp);
                }
}
```

```
        void display()
        {
                temp=head;
                if(head==NULL)
                {
                  printf("list is empty");

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

```
*************************************************************************
*************OUTPUT
1.insert at begin
2.insert at end
3.insert at pos
4.delete at begin
5.delete at end
6.delete at pos
7.display
8.exit
enter your choice1
enter the value56

1.insert at begin
```

```
2.insert at end
3.insert at pos
4.delete at begin
5.delete at end
6.delete at pos
7.display
8.exit
enter your choice1
enter the value23

1.insert at begin
2.insert at end
3.insert at pos
4.delete at begin
5.delete at end
6.delete at pos
7.display
8.exit
enter your choice1
enter the value45

1.insert at begin
2.insert at end
3.insert at pos
4.delete at begin
5.delete at end
6.delete at pos
7.display
8.exit
enter your choice7
```

```
	45	23	56
1.insert at begin
2.insert at end
3.insert at pos
4.delete at begin
5.delete at end
6.delete at pos
7.display
8.exit
enter your choice2

1.insert at begin
2.insert at end
3.insert at pos
4.delete at begin
5.delete at end
6.delete at pos
7.display
8.exit
enter your choice7
	45	23	56	45
1.insert at begin
2.insert at end
3.insert at pos
4.delete at begin
5.delete at end
6.delete at pos
7.display
8.exit
enter your choice3
enter location where you want to insert2

1.insert at begin
2.insert at end
3.insert at pos
4.delete at begin
5.delete at end
6.delete at pos
7.display
8.exit
enter your choice7
        45      45      23      56      45
1.insert at begin
2.insert at end
3.insert at pos
4.delete at begin
5.delete at end
6.delete at pos
7.display
8.exit
enter your choice4
45 is deleted
1.insert at begin
2.insert at end
3.insert at pos
4.delete at begin
5.delete at end
6.delete at pos
7.display
8.exit
```

```
enter your choice7
        45      23      56      45
1.insert at begin
2.insert at end
3.insert at pos
4.delete at begin
5.delete at end
6.delete at pos
7.display
8.exit
enter your choice5
45 is deleted
1.insert at begin
2.insert at end
3.insert at pos
4.delete at begin
5.delete at end
6.delete at pos
7.display
8.exit
enter your choice5
56 is deleted
1.insert at begin
2.insert at end
3.insert at pos
4.delete at begin
5.delete at end
6.delete at pos
7.display
8.exit
```

```
          enter your choice6
          enter location2
          23 is deleted
          1.insert at begin
          2.insert at end
          3.insert at pos
          4.delete at begin
          5.delete at end
          6.delete at pos
          7.display
          8.exit
          enter your choice7
                45
          1.insert at begin
          2.insert at end
          3.insert at pos
          4.delete at begin
          5.delete at end
          6.delete at pos
          7.display
          8.exit
enter your choice
```

Doubly linked list

```
#include<stdio.h>
#include<stdlib.h>
struct node{
        int data;
        struct node*next;
        struct node*prev;
};
struct node*head;
struct node*ptr;
struct node*temp;
struct node*temp1;
void insert_begin(int val);
void insert_end(int val);
void insert_pos(int val);
void delete_begin();
void delete_end();
void delete_pos();
void display();
int main()
{
        int ch,val;
        while(1)
        {
           printf("\n1.insert at begin \n2.insert at end \n3.insert at pos \n4.delete at begin
\n5.delete at end \n6.delete at pos \n7.display \n8.exit");
           printf("\nenter your choice");
           scanf("%d",&ch);
           switch(ch)
           {
                   case 1: insert_begin(val);
                           break;
                   case 2: insert_end(val);
                                   break;
                   case 3: insert_pos(val);
                           break;
                   case 4:  delete_begin();
                           break;
                   case 5:delete_end();
                           break;
                   case 6:delete_pos();
                           break;
                   case 7:display();
                           break;
                   case 8:exit(0);
```

```
                                  break;
                      default:printf("enter wrong choice");
                              break;

                      }

              }
             }
             void insert_begin(int val)
             {
                printf("\n enter the number");
                      ptr=(struct node*)malloc(sizeof(struct node));
                      if(ptr==NULL)
                      {
                              printf("memory is not allocated");
                      }
                      else
                      {
                         scanf("%d",&ptr->data);
                         if(head==NULL)
                      {
                          ptr->next=NULL;
                               ptr->prev=NULL;
                               head=ptr;

                      }
                      else
                      {
                              ptr->next=head;
                         ptr->prev=NULL;
                         head->prev=ptr;
                               head=ptr;

                      }
          }
         }
         void insert_end(int val)
             {
                printf("\n enter the no.");
                      ptr=(struct node*)malloc(sizeof(struct node));
                      if(ptr==NULL)
                      {
                              printf("memory is not allocated");
                      }
                      else
                      {
                        scanf("%d",&ptr->data);
                        if(head==NULL)
                        {
                         ptr->next=NULL;
                                ptr->prev=NULL;
                                head=ptr;

                      }
                        else
                  {
                                temp=head;
                                while(temp->next!=NULL)
                                {
                                          temp=temp->next;
                                }
                                ptr->next=NULL;
                                temp->next=ptr;
                                ptr->prev=temp;

                 }
              }
              printf("node is inserted");
       }
       void insert_pos(int val)
              {
                      int i,loc;
                      printf("enter location where you want to insert");
                      scanf("%d",&loc);
                      temp=head;
                      for( i=0;i<loc-1;i++)
                                {
                                          temp=temp->next;
                                          if(temp==NULL)
                                          {
                                            printf("cannot insert");
                                            return;

                                          }
                                }
                      ptr=(struct node*)malloc(sizeof(struct node));
                      if(ptr==NULL)
                      {
                                printf("memory is not allocated");
                      }
                      else
                      {
```

```
                printf("\n enter value:");
                scanf("%d",&ptr->data);
            ptr->next=temp->next;
            ptr->next->prev=ptr;
                 temp->next=ptr;
                 ptr->prev=temp;
          }
          printf("node is inserted");
}

void delete_begin()
{
  if(head==NULL)
  {
     printf("can't delete");
  }
  else if(head->next==NULL)
  {
     printf("\n%d is deleted",head->data);
     free(head);
  }
  else
  {
     temp=head;
     printf("\n%d is deleted",temp->data);
     head=head->next;
     temp->next=NULL;
     head->prev=NULL;
     free(temp);
  }
}
void delete_end()
{
   if(head==NULL)
   {
      printf("linked list is empty");
   }
   else if(head->next==NULL)
   {
      printf("\n%d id deleted",head->data);
      free(head);
   }
   else
   {
      temp=head;
      while(temp->next!=NULL)
```

```
        {
            temp=temp->next;
        }
        temp->prev->next=NULL;
        temp->prev=NULL;
        printf("\n %d is deleted",temp->data);
        free(temp);
    }
}
void delete_pos()
{
    int i,loc;
    if(head==NULL)
    {
        printf("linked list is empty");
    }
    else
    {
        temp=head;
        printf("\n enter location");
        scanf("%d",&loc);
        for(i=0;i<loc;i++)
        {
            temp=temp->next;
            if(temp==NULL)
            {
                printf("can't delete");
            }
        }
        temp1=temp->next;
        temp->next=temp1->next;
        temp1->next->prev=temp;
        temp1->next=NULL;
        temp1->prev=NULL;
        printf("\n%d is deleted",temp1->data);
        free(temp1);
    }
}
 void display()
{
    temp = head; // Assuming 'temp' is a global variable or declared in the correct scope
    if (head == NULL)
    {
        printf("list is empty");
    }
    else
    {
        while (temp != NULL)
        {
            printf("\t %d ", temp->data);
            temp = temp->next;
        }
    }
}
```

```
Output
1.insert at begin
2.insert at end
3.insert at pos
4.delete at begin
5.delete at end
6.delete at pos
7.display
8.exit
enter your choice1

 enter the number34

1.insert at begin
2.insert at end
3.insert at pos
4.delete at begin
5.delete at end
6.delete at pos
7.display
8.exit
enter your choice1

 enter the number98

1.insert at begin
2.insert at end
3.insert at pos
4.delete at begin
5.delete at end
6.delete at pos
7.display
8.exit
enter your choice1

 enter the number78
```

```
1.insert at begin
2.insert at end
3.insert at pos
4.delete at begin
5.delete at end
6.delete at pos
7.display
8.exit
enter your choice7
        78      98      34
1.insert at begin
2.insert at end
3.insert at pos
4.delete at begin
5.delete at end
6.delete at pos
7.display
8.exit
enter your choice2

 enter the no.45
node is inserted
1.insert at begin
2.insert at end
3.insert at pos
4.delete at begin
5.delete at end
6.delete at pos
7.display
8.exit
enter your choice7
        78      98      34      45
1.insert at begin
2.insert at end
3.insert at pos
4.delete at begin
5.delete at end
6.delete at pos
7.display
8.exit
enter your choice3
enter location where you want to insert2

 enter value:22
node is inserted
```

```
1.insert at begin
2.insert at end
3.insert at pos
4.delete at begin
5.delete at end
6.delete at pos
7.display
8.exit
enter your choice7
        78      98      22      34      45
1.insert at begin
2.insert at end
3.insert at pos
4.delete at begin
5.delete at end
6.delete at pos
7.display
8.exit
enter your choice4

78 is deleted
1.insert at begin
2.insert at end
3.insert at pos
4.delete at begin
5.delete at end
6.delete at pos
7.display
8.exit
enter your choice5

 45 is deleted
1.insert at begin
2.insert at end
3.insert at pos
4.delete at begin
5.delete at end
6.delete at pos
7.display
8.exit
enter your choice6

 enter location3
can't delete
```

Write the programme to implement the circular singly linked list.

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
  int data;
  struct node *next;
};
struct node *head,*ptr,*temp,*head,*temp1;
void insert_begin();
void insert_last();
void delete_begin();
void delete_last();
void search();
void display();
void main()
{
  int ch;
  while(1)
  {
    printf("\n1. insert at begin\n2.insert at last\n3.delete at begin\n4.delete at
last\n5.search\n6.display\n7.exit\n");
    printf("enter your choice\n");
    scanf("%d",&ch);
    switch(ch)
    {
      case 1:insert_begin();
            break;
      case 2:insert_last();
            break;
      case 3:delete_begin();
             break;
      case 4:delete_last();
             break;
      case 5:search();
             break;
      case 6:display();
             break;
      case 7:exit(0);
             break;
      default:printf("invalid choice");
    }
  }
}
void insert_begin()
{
  ptr=(struct node*)malloc(sizeof(struct node));
  if(ptr==NULL)
  {
    printf("memory is not allocated\n");
  }
  else
  {
    printf("enter the value\n");
    scanf("%d",&ptr->data);
    if(head==NULL)
    {
      head=ptr;
      ptr->next=head;
    }
```

```
     else
     {
       temp=head;
       while(temp->next!=head)
       {
         temp=temp->next;
       }
       ptr->next=head;
       temp->next=ptr;
       head=ptr;
     }
   }
   printf("node is inserted\n");
 }
 void insert_last()
 {
     ptr = (struct node*)malloc(sizeof(struct node));
     if(ptr == NULL)
     {
         printf("memory is not allocated");
         return;
     }

     printf("enter the value: ");
     scanf("%d", &ptr->data);

     if(head == NULL)
     {
         head = ptr;
         ptr->next = head;
```

```
    }
    else
    {
        temp = head;
        while(temp->next != head)
        {
            temp = temp->next;
        }
        temp->next = ptr;
        ptr->next = head;
    }

    printf("node is inserted\n");
}
void delete_begin()
{
   if(head == NULL)
  {
   printf("\nUNDERFLOW");
  }
  else if(head->next == head)
  {
   head = NULL;
   free(head);
   printf("\nnode deleted\n");
  }

  else
  {
   ptr = head;
   while(ptr -> next != head)
   ptr = ptr -> next;
   ptr->next = head->next;
   free(head);
   head = ptr->next;
   printf("\n node deleted\n");
  }
}
void delete_last()
{
   if (head==NULL)
   {
    printf("linkedlist is empty\n");
   }
   else if (head->next==head)
   {
    printf("%d is deleted",head->data);
    free(head);
   }
   else
   {
    temp=head;
    while(temp->next!=head)
    {
      temp1=temp;
      temp=temp->next;
    }
    temp1->next=head;
    temp->next=NULL;
    printf("%d is deleted",temp->data);
    free(temp);
   }
   printf("\n node is deleted");
}
void search()
{
   int val,i=0,flag=0;
   temp = head;
  if(head == NULL)
  {
   printf("\nEmpty List\n");
  }
  else
  {
   printf("\nEnter item which you want to search?\n");
   scanf("%d",&val);
   while(temp!=head)
   {
    if(temp->data == val)
    {
      printf("item found at location %d",val);
    }
    else
    {
      flag=1;
    }
    i++;
    temp=temp->next;
   }
   if(flag==1)
  {
    printf("value is not found");
  }
 }
}
void display()
{
    ptr = head;
    if (head == NULL)
    {
        printf("\nnothing to print");
    }
    else
    {
        printf("\n printing values ... \n");
        do
        {
            printf("\n%d", ptr->data);
            ptr = ptr->next;
        } while (ptr != head);
    }
}
```

```
OUTPUT
1. insert at begin
2.insert at last
3.delete at begin
4.delete at last
5.search
6.display
```

```
7.exit
enter your choice
1
enter the value
24
node is inserted

1. insert at begin
2.insert at last
3.delete at begin
4.delete at last
5.search
6.display
7.exit
enter your choice
1
enter the value
46
node is inserted

1. insert at begin
2.insert at last
3.delete at begin
4.delete at last
5.search
6.display
7.exit
enter your choice
1
enter the value
```

```
79
node is inserted

1. insert at begin
2.insert at last
3.delete at begin
4.delete at last
5.search
6.display
7.exit
enter your choice
6

 printing values ...

79
46
24
1. insert at begin
2.insert at last
3.delete at begin
4.delete at last
5.search
6.display
7.exit
enter your choice
2
enter the value: 4
node is inserted
```

```
1. insert at begin
2.insert at last
3.delete at begin
4.delete at last
5.search
6.display
7.exit
enter your choice
2
enter the value: 64
node is inserted

1. insert at begin
2.insert at last
3.delete at begin
4.delete at last
5.search
6.display
7.exit
enter your choice
6

 printing values ...

79
46
24
4
64
1. insert at begin
```

```
2.insert at last
3.delete at begin
4.delete at last
5.search
6.display
7.exit
enter your choice
3

 node deleted

1. insert at begin
2.insert at last
3.delete at begin
4.delete at last
5.search
6.display
7.exit
enter your choice
6

 printing values ...

46
24
4
64
1. insert at begin
2.insert at last
3.delete at begin
```

```
4.delete at last
5.search
6.display
7.exit
enter your choice
4
64 is deleted
 node is deleted
1. insert at begin
2.insert at last
3.delete at begin
4.delete at last
5.search
6.display
7.exit
enter your choice
6

 printing values ...

46
24
4
1. insert at begin
2.insert at last
3.delete at begin
4.delete at last
5.search
6.display
7.exit
```

```
enter your choice
5

Enter item which you want to search?
2

1. insert at begin
2.insert at last
3.delete at begin
4.delete at last
5.search
6.display
7.exit
enter your choice
6

 printing values ...

46
24
4
1. insert at begin
2.insert at last
3.delete at begin
4.delete at last
5.search
6.display
7.exit
enter your choice
```

Write programme to implement to binary tree

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

struct node
{
   int data;
   struct node *left;
   struct node *right;
};

struct node *root = NULL;

// Function prototype
struct node *create();

void pre_order_traversal(struct node* root) {
   if(root != NULL) {
      printf("%d ", root->data);
      pre_order_traversal(root->left);
      pre_order_traversal(root->right);
   }
}

void inorder_traversal(struct node* root) {
   if(root != NULL) {
      inorder_traversal(root->left);
      printf("%d ", root->data);
```

```
        inorder_traversal(root->right);
    }
}

void post_order_traversal(struct node* root) {
    if(root != NULL) {
        post_order_traversal(root->left);
        post_order_traversal(root->right);
        printf("%d ", root->data);
    }
}

struct node *create() {
    struct node *temp;
    int data, choice;

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

    printf("Press 0 to exit");
    printf("\nPress 1 for a new node");
    printf("\nEnter your choice: ");
    scanf("%d", &choice);

    if(choice == 0) {
        return NULL;
    } else {
        printf("Enter the data: ");
        scanf("%d", &data);
        temp->data = data;
        printf("Enter the left child of %d\n", data);
        temp->left = create();
        printf("Enter the right child of %d\n", data);
        temp->right = create();
        return temp;
    }
```

```
OUTPUT
1. Create
2. Inorder
3. Preorder
4. Postorder
5. Exit
Enter your choice: 1
Press 0 to exit
Press 1 for a new node
Enter your choice: 1
Enter the data: 24
Enter the left child of 24
Press 0 to exit
Press 1 for a new node
Enter your choice: 1
Enter the data: 12
Enter the left child of 12
Press 0 to exit
Press 1 for a new node
Enter your choice: 1
Enter the data: 78
Enter the left child of 78
Press 0 to exit
Press 1 for a new node
Enter your choice: 0
Enter the right child of 78
Press 0 to exit
Press 1 for a new node
Enter your choice: 1
Enter the data: 45
Enter the left child of 45
Press 0 to exit
Press 1 for a new node
Enter your choice: 1
Enter the data: 90
Enter the left child of 90
Press 0 to exit
Press 1 for a new node
Enter your choice: 1
Enter the data: 89
Enter the left child of 89
Press 0 to exit
Press 1 for a new node
Enter your choice: 0
Enter the right child of 89
Press 0 to exit
Press 1 for a new node
Enter your choice: 0
Enter the right child of 90
Press 0 to exit
Press 1 for a new node
Enter your choice: 2
Enter the data: 12
Enter the left child of 12
```

4 .Write programme to implement Binary search tree.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.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
   {
      /* Now We can delete this node and replace with either minimum element in the right sub
tree or maximum element in the left subtree */
      if(root->right && root->left)
      { /* Here we will replace with minimum element in the right sub tree */
         temp = findmin(root->right);
         root->data = temp->data;
         /* As we replaced it with some other node, we have to delete that node */
         root->right = delet(root->right,temp->data);
      }
      else
```

```
        {
            /* If there is only one or zero children then we can directly remove it from the tree and
connect its parent to its child */
            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 NULL;
    }
    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");
}
```

```
OUTPUT
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: 2

 enter values: 12
23

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: 2

 enter values: 78 45

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: 78

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 23 45
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 23 45
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:
45 23 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

 45 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
```

Write programme to implement Hash table.

```
#include<stdio.h>
#include<conio.h>
#define size 10
int arr[size];
void init();
void insert(int value);
void del(int value);
void print();
int main()
{
  int ch =0,temp;
  init();
  while (ch!=4)
  {
     printf("\n1.Insert\n2.Delete\n3.Print hash table\n4.Exit\nEntre your choice: ");
     scanf("%d",&ch);
     switch(ch)
     {
        case 1 : printf("Entre element to be inserted : ");
                 scanf("%d",&temp);
                 insert(temp);
                 break;
        case 2 : printf("Entre element to be deleted : ");
                 scanf("%d",&temp);
                 del(temp);
                 break;
        case 3 : print();
                 break;
```

```
         case 4 : exit(0);
                  break;
         default : printf("wrong input !!!");
       }
     }
  return 0;
}

void init()
{
   int i;
   for(i = 0; i < size; i++)
       arr[i] = -1;
}

void insert(int value)
{
    int flag=1;
    int key = value % size;
    if(arr[key] == -1)
    {
       arr[key] = value;
       printf("%d inserted with key %d !\n", value,key);


    }
    else
    {
        printf("Collision : key %d has element %d already!\n",key,arr[key]);
        while(flag)
        {
            while(arr[key]!=-1)
            {
               key++;
            }
            if(key != size-1)
            {
                //if(arr[key] == -1)
                //{
                     arr[key] = value;
                  printf("%d inserted with key %d !\n", value,key);
                  flag=0;
            }
            else
               printf("\n Unable to insert");
         }
      }
   }

   //delete item from table
   void del(int value)
   {
      /*int key = value % size;
      if(arr[key] == value)
          arr[key] = -1;
      else
          printf("%d not present in the hash table\n",value);*/
      int i;
      for(i =0; i<size-1; i++)
      {
         if(arr[i] == value)
```

```
        {
           arr[i] = -1;
           printf("\n %d is deletd",value);
        }
    }


}
//hash table print method
void print()
{
    int i;
    printf("\nVALUE\t---\tKEY");
    printf("\n===================\n");
    for(i = 0; i < size; i++)
    {
        if(arr[i]!=-1)
        {
            printf("%d\t---\t%d\n",arr[i],i);
        }
    }
}
```

OUTPUT

```
1.Insert
2.Delete
3.Print hash table
4.Exit
Entre your choice: 1
Entre element to be inserted : 45
45 inserted with key 5 !
```

```
1.Insert
2.Delete
3.Print hash table
4.Exit
Entre your choice: 1
Entre element to be inserted : 67
67 inserted with key 7 !

1.Insert
2.Delete
3.Print hash table
4.Exit
Entre your choice: 1
Entre element to be inserted : 90
90 inserted with key 0 !

1.Insert
2.Delete
3.Print hash table
4.Exit
Entre your choice: 1
Entre element to be inserted : 34
34 inserted with key 4 !

1.Insert
2.Delete
3.Print hash table
4.Exit
Entre your choice: 3
```

```
VALUE   ---    KEY
==================
90      ---    0
34      ---    4
45      ---    5
67      ---    7

1.Insert
2.Delete
3.Print hash table
4.Exit
Entre your choice: 2
Entre element to be deleted : 34

 34 is deletd
1.Insert
2.Delete
3.Print hash table
4.Exit
Entre your choice: 3

VALUE   ---    KEY
==================
90      ---    0
45      ---    5
67      ---    7

1.Insert
2.Delete
```

```
3.Print hash table
4.Exit
Entre your choice:
```

Write a program to implement selection sort

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

void selection_sort(int a[],int n);
void display(int a[], int n);

int a[10], n;
int main()
{
  int i,ch;

  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: ");
    }
  }

 // getch();
}

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]);
  }
}
```

OUTPUT

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

 enter the elements: 23 45 67 78 12 2

1. Sort array
2. display sorted array
3. exit
 Enter your choice:
 sorted array is:
 23
 45
 67
 78
 12
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:
 12
 23
 45
 67
 78
1. Sort array
2. display sorted array
3. exit
 Enter your choice:
```

Write a program to implement insertion sort

```c
#include <stdio.h>
void insertionsort(int a[], int n)
{
   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;
   }
}

int main()
{
   int a[20],n,i;
   printf("\n how many elements you want to insert: ");
   scanf("%d",&n);
   printf("\n enter values: ");
   for(i=0;i<n;i++)
   {
      scanf("%d",&a[i]);
   }
   insertionsort(a,n);
   printf("\n sorted array is : ");
   for(i=0; i<n; i++)
   {
      printf("\t %d ",a[i]);
   }
   return 0;
}
```

OUTPUT

```
how many elements you want to insert: 5

 enter values: 12 34 56 23 1

 sorted array is :       1       12      23      34      56
```

write a program to implement merge sort

```
#include<stdio.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];
  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=0;k<=ub;k++)
  {
   a[k]=b[k];
  }
}
```

OUTPUT

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

 Enter the 5 elemets in array: 12 34 76 9 6

 sorted array is:  6  9  12  34  76
```

Write a program to implement quick sort

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

void quicksort(int a[], int first, int last);

void main()
{
  int i,n,a[20];
  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);
      }
   }
```

OUTPUT

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

 Enter the 5 elemets in array: 12 2 1 34 6

 sorted array is:  1  2  6  12  34
```