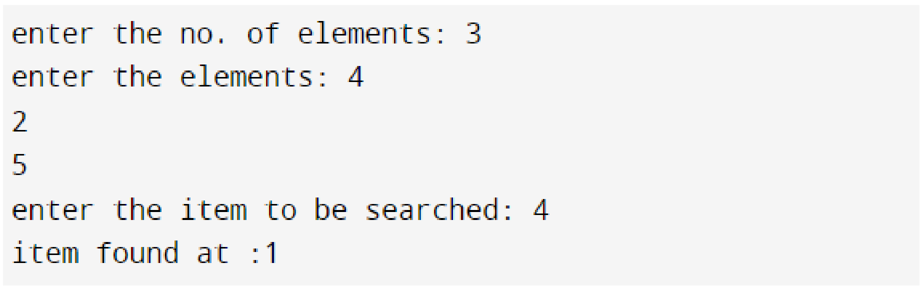
ADVANCED DATA STRUCTURES USING C

1.LINEAR SEARCH IMPLEMENTATION

CODE:

```
# include<stdio.h>
void main()
{
        int a[20],n,item,i;
        printf("enter the no. of elements: ");
        scanf("%d",&n);
        printf("enter the elements: ");
        for (i=0;i<n;i++)
                scanf("%d",&a[i]);
        printf("enter the item to be searched: ");
        scanf("%d",&item);
        for(i=0; i<=n;i++)
        if(item==a[i])
         {
                printf("item found at %d : ",i+1);
                break;
        }
        if(i>n)
        printf("item does not exist");
}
```

OUTPUT:-



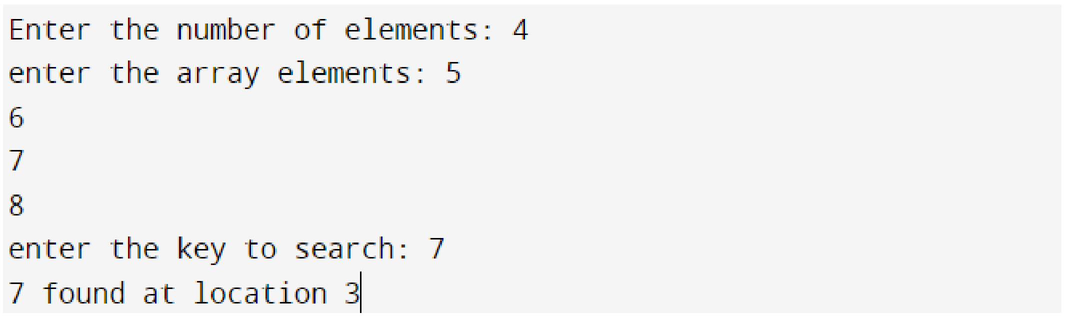
```
enter the no. of elements: 3
enter the elements: 4
2
5
enter the item to be searched: 4
item found at :1
```

2.BINARY SEARCH IMPLEMENTATION

CODE:

```
#include<stdio.h>
 void main()
{
        int a[100],n,key,i,first,last,middle;
       printf("Enter the number of elements: ");
       scanf("%d",&n);
       printf("enter the array elements: ");
        for(i=0;i<n;i++)
        scanf("%d",&a[i]);
        printf("enter the key to search: ");
        scanf("%d",&key);
        first=0;
        last=n-1;
        middle=(first+last)/2;
        while(first<=last)
        {
               if(a[middle]<key)
                      first=middle+1;
               else if (a[middle]==key)
               {
                       printf("%d found at location %d",key,middle+1);
                       break;
               }
               else
                      last=middle-1;
                       middle=(first+last)/2;
        }
        if(first>last)
                printf("%d item does not exist in the list",key);
}
```

OUTPUT :-



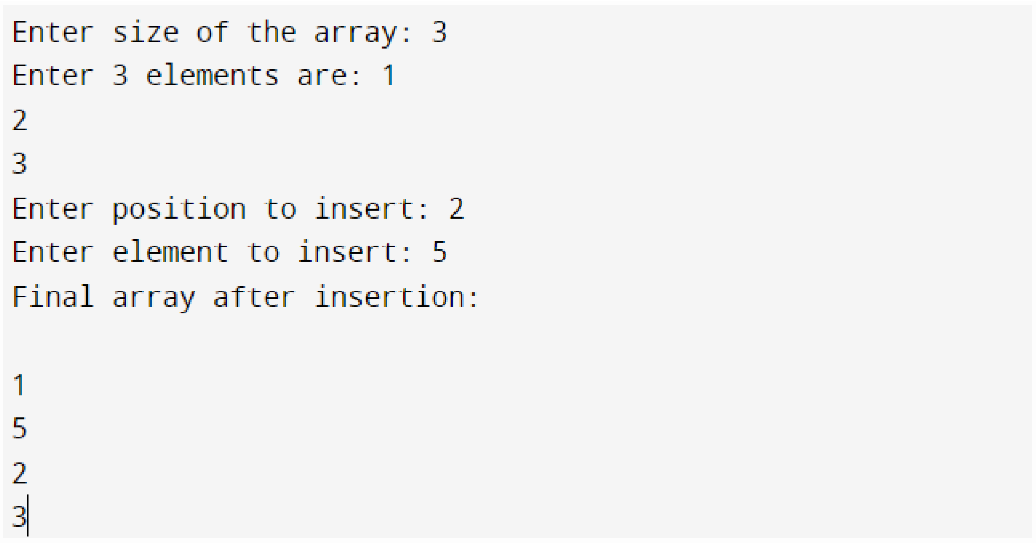
```
Enter the number of elements: 4
enter the array elements: 5
6
7
8
enter the key to search: 7
7 found at location 3
```

3.ARRAY INSERTION

CODE:

```
#include<stdio.h>
int main()
{
int a[100],n,pos,item,i;
printf("\nEnter size of the array: ");
scanf("%d",&n);
printf("Enter %d elements are: ",n);
for(i=0;i<n;i++)
scanf("%d",&a[i]);
printf("\nEnter position to insert: ");
scanf("%d",&pos);
printf("\nEnter element to insert: ");
scanf("%d",&item);
for(i=n-1;i>=pos-1;i--)
a[i+1]=a[i];
a[pos-1]=item;
printf("\nFinal array after insertion: \n");
for(i=0;i<n+1;i++)
printf("\n%d",a[i]);
return 0;
}
```

<u>OUTPUT:-</u>



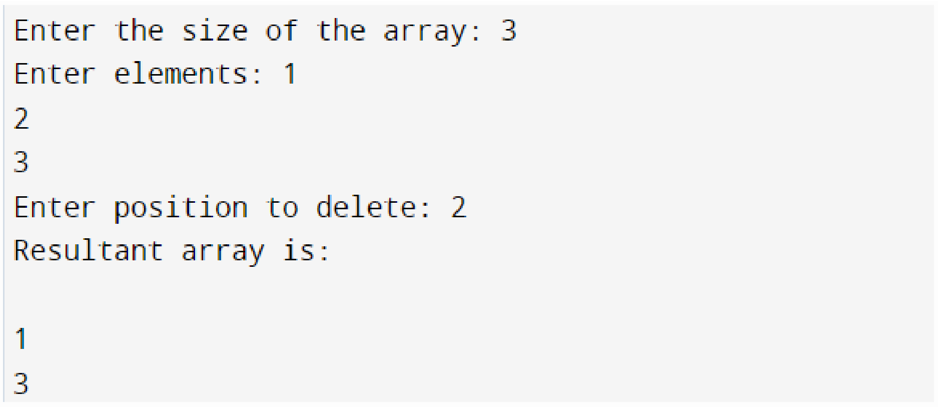
```
Enter size of the array: 3
Enter 3 elements are: 1
2
3
Enter position to insert: 2
Enter element to insert: 5
Final array after insertion:

1
5
2
3
```

<u>4.ARRAY DELETION</u>

<u>CODE:</u>

```
#include<stdio.h>
int main()
{
        int a[100],n,pos,i;
        printf("\nEnter the size of the array: ");
        scanf("%d",&n);
        printf("\n Enter elements: ");
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);
        printf("\nEnter position to delete: ");
        scanf("%d",&pos);
        if(pos>=n+1)
                printf("\nDeletion not possible\n");
        else
        {
                for(i=pos-1;i<n-1;i++)
                a[i]=a[i+1];
                printf("\nResultant array is: \n");
                for(i=0;i<n-1;i++)
                printf("\n%d",a[i]);
        }
        return 0;
}
```

<u>OUTPUT:-</u>



```
Enter the size of the array: 3
Enter elements: 1
2
3
Enter position to delete: 2
Resultant array is:

1
3
```

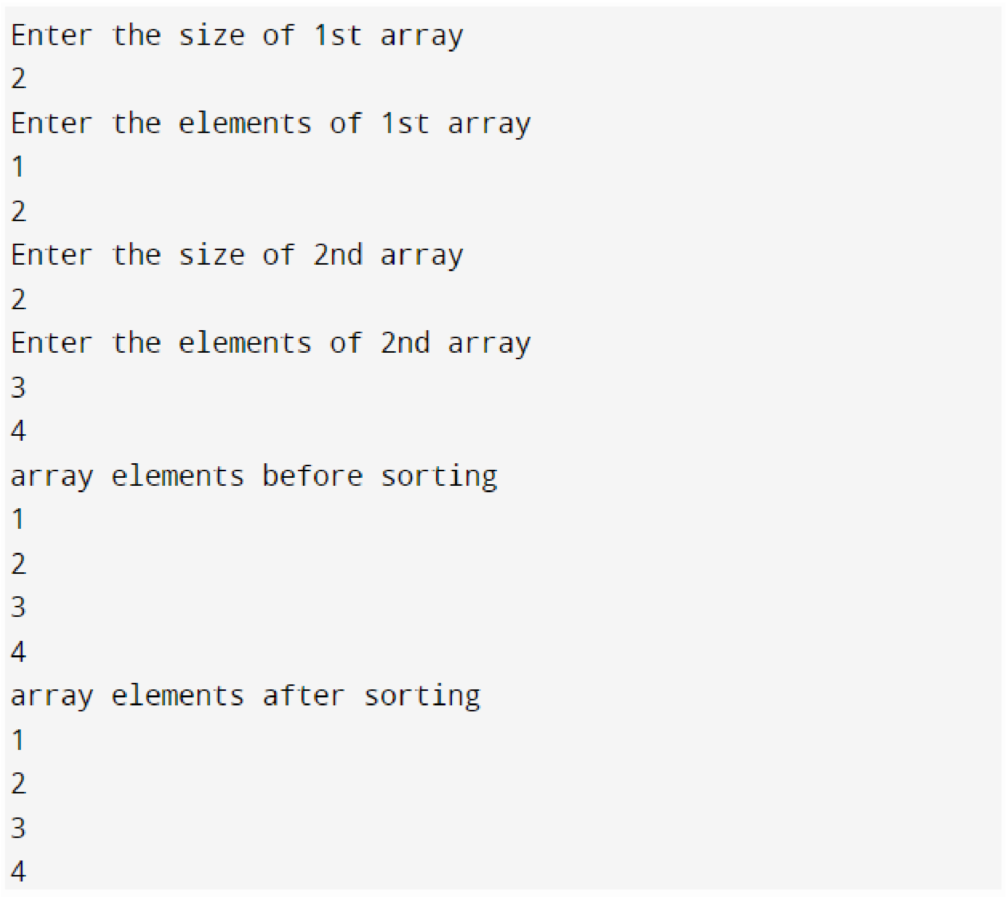
5.ARRAY MERGING

CODE:

```
#include <stdio.h>
void main()
{
	int array1[100],array2[100],array3[100],size1,size2,size3;
	printf("Enter the size of 1st array\n");
	scanf("%d",&size1);
	printf("Enter the elements of 1st array\n");
	for(int i=0;i<size1;i++)
		scanf("%d",&array1[i]);
	printf("Enter the size of 2nd array\n");
	scanf("%d",&size2);
	printf("Enter the elements of 2nd array\n");
	for(int i=0;i<size2;i++)
		scanf("%d",&array2[i]);
	size3=size1+size2;
	for(int i=0;i<size1;i++)
		array3[i]=array1[i];
	for(int i=0;i<size2;i++)
		array3[i+size1]=array2[i];
	printf("array elements before sorting\n");
	for(int i=0;i<size3;i++)
		printf("%d \n",array3[i]);
	for(int i = 0; i < size3; i++)
	{
		int temp;
		for(int j = i + 1; j < size3; j++)
		{
		if(array3[i] > array3[j])
```

```
                {
                        temp = array3[i];
                        array3[i] = array3[j];
                        array3[j] = temp;}
                }
        }
        printf("array elements after sorting\n");
        for(int i=0;i<size3;i++)
        printf("%d \n",array3[i]);
}
```

OUTPUT:-



```
Enter the size of 1st array
2
Enter the elements of 1st array
1
2
Enter the size of 2nd array
2
Enter the elements of 2nd array
3
4
array elements before sorting
1
2
3
4
array elements after sorting
1
2
3
4
```

6.MATRIX ADDITION

CODE:-

```
#include <stdio.h>
int main() {
  int r, c, a[100][100], b[100][100], sum[100][100], i, j;
  printf("Enter the number of rows (between 1 and 100): ");
  scanf("%d", &r);
  printf("Enter the number of columns (between 1 and 100): ");
  scanf("%d", &c);
  printf("\nEnter elements of 1st matrix:\n");
  for (i = 0; i < r; ++i)
            for (j = 0; j < c; ++j) {
            printf("Enter element a%d%d: ", i + 1, j + 1);
            scanf("%d", &a[i][j]);
            }
  printf("Enter elements of 2nd matrix:\n");
  for (i = 0; i < r; ++i)
            for (j = 0; j < c; ++j) {
            printf("Enter element b%d%d: ", i + 1, j + 1);
            scanf("%d", &b[i][j]);
            }
  for (i = 0; i < r; ++i)
            for (j = 0; j < c; ++j) {
            sum[i][j] = a[i][j] + b[i][j];
            }
  printf("\nSum of two matrices: \n");
  for (i = 0; i < r; ++i)
            for (j = 0; j < c; ++j) {
            printf("%d   ", sum[i][j]);
            if (j == c - 1) {
            printf("\n\n");
            }
            }
  return 0;
}
```

OUTPUT:-

```
Enter the number of rows (between 1 and 100): 2
Enter the number of columns (between 1 and 100): 2
Enter elements of 1st matrix:
Enter element a11: 1
Enter element a12: 2
Enter element a21: 3
Enter element a22: 4
Enter elements of 2nd matrix:
Enter element b11: 1
Enter element b12: 2
3Enter element b21: 3
Enter element b22: 4
Sum of two matrices:
2   4

6   8
```

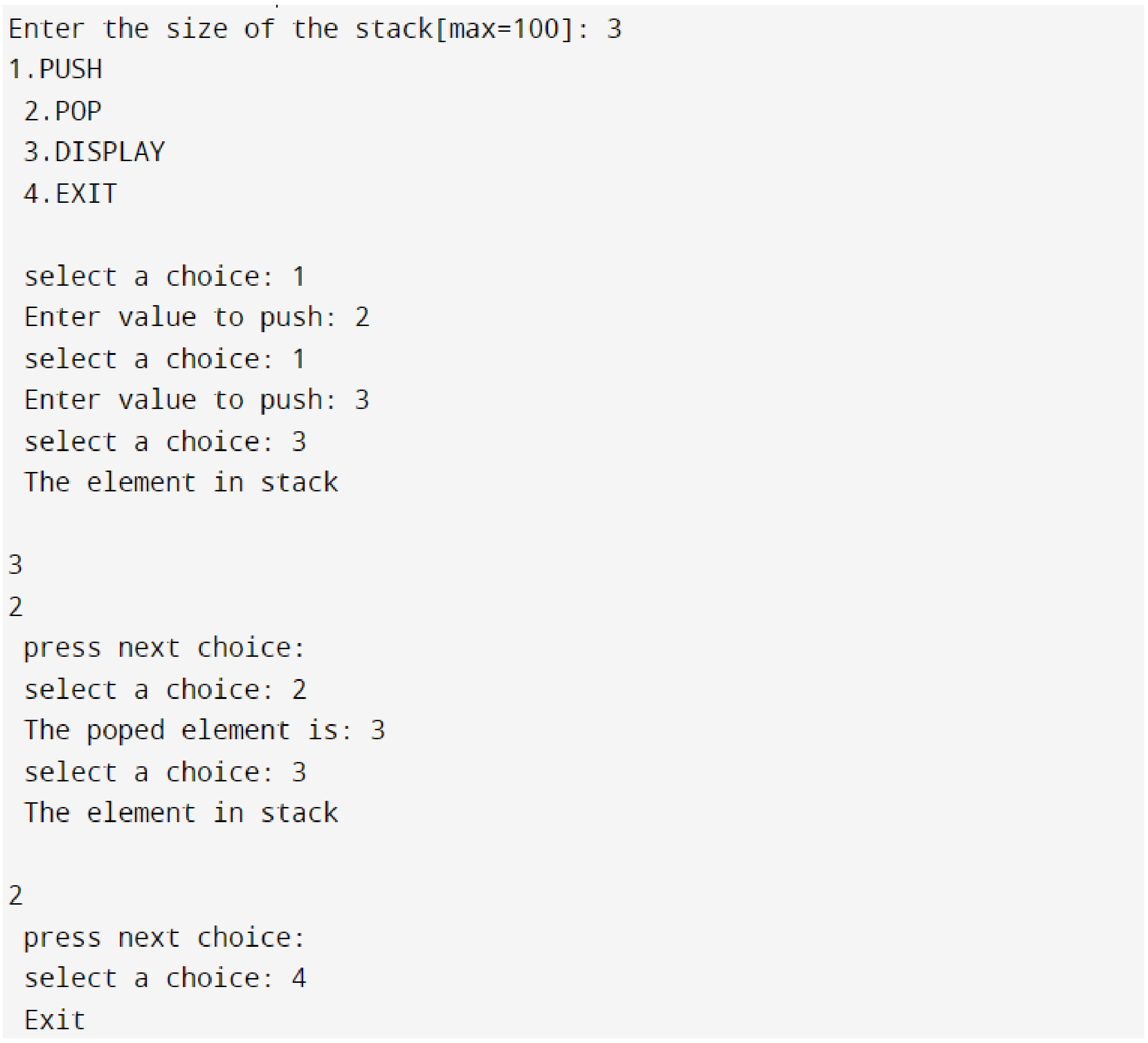
7.STACK OPERATION

Code:

```
#include<stdio.h>
int stack[100],choice,n,top,x,i;
void push(void);
void pop(void);
void display(void);
int main()
{
}
top=-1;
printf("Enter the size of the stack[max=100]: ");
scanf("%d",&n);
printf("\n 1.PUSH \n 2.POP \n 3.DISPLAY \n 4.EXIT \n");
do
{
printf("\n select a choice: ");
scanf("%d",&choice);
switch(choice)
{
case 1:
push();
break;
case 2:
pop();
break;
case 3:
display();
break;
case 4:
printf("Exit");
break;
default:
printf("\nEnter a valid choice\n");
}
}
while(choice!=4);
return 0;
}
void push()
{
```

```
if(top>=n-1)
{
printf("\nstack is overflow\n");
}
else
{
printf("Enter value to push: ");
scanf("%d",&x);
top++;
stack[top]=x;
}
}
void pop()
{
if(top<=-1)
{
printf("\nstack is underflow\n");
}
else
{
printf("The poped element is: %d",stack[top]);
top--;
}
}
void display()
{
if(top>=0)
{
printf("\nThe element in stack \t\n");
for(i=top;i>=0;i--)
printf("\n%d",stack[i]);
printf("\n press next choice: ");
}
else
{
printf("\nstack is empty\n");
}
}
```

OUTPUT:-



```
Enter the size of the stack[max=100]: 3
1.PUSH
 2.POP
 3.DISPLAY
 4.EXIT

 select a choice: 1
 Enter value to push: 2
 select a choice: 1
 Enter value to push: 3
 select a choice: 3
 The element in stack

3
2
 press next choice:
 select a choice: 2
 The poped element is: 3
 select a choice: 3
 The element in stack

2
 press next choice:
 select a choice: 4
 Exit
```

8.QUEUE OPERATION

CODE:

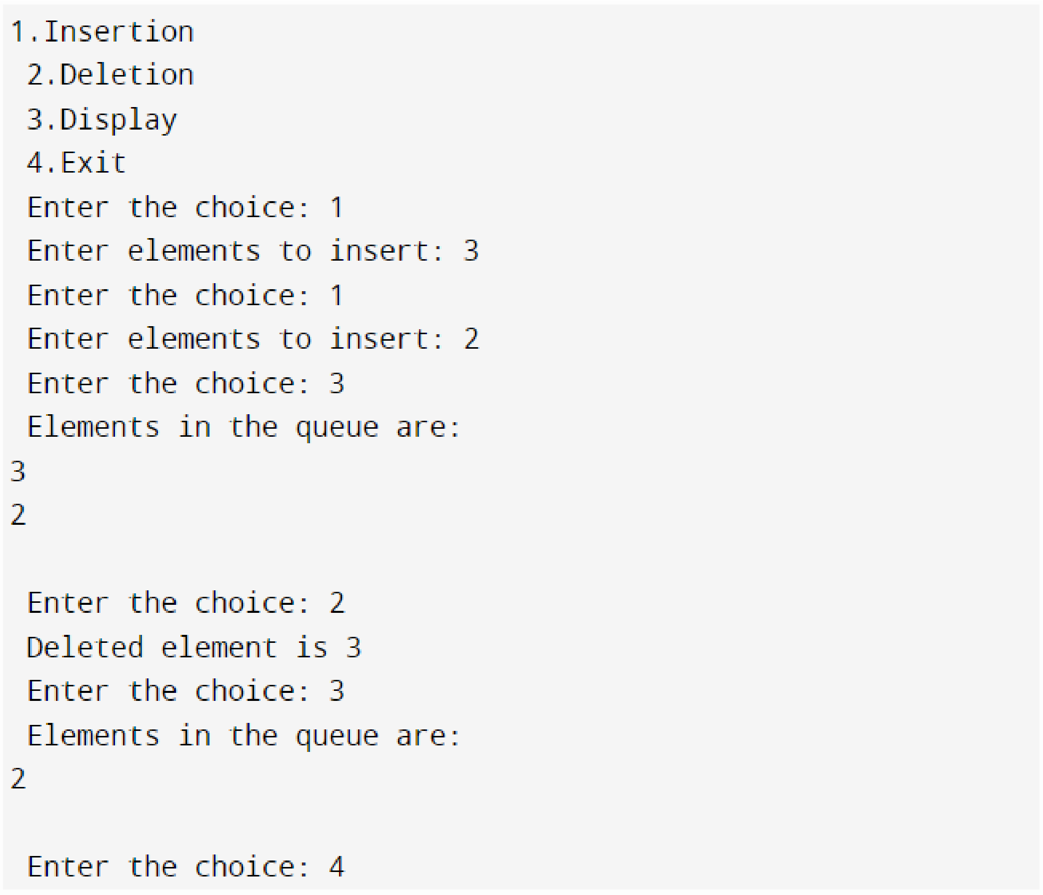
```
#include<stdio.h>
#include<stdlib.h>
#define n 5
int main()
{
int queue[n],choice=1,front=0,rear=0,i,x=n;
printf("\n 1.Insertion \n 2.Deletion \n 3.Display \n 4.Exit");
while(choice)
{
printf("\n Enter the choice: ");
scanf("%d",&choice);
switch(choice)
{
case 1:
if(rear==x)
{
printf("\n Queue is full");
}
else
{
printf("\n Enter elements to insert: ");
scanf("%d",&queue[rear]);
rear=rear+1;
}
break;
case 2:
```

```
if(front==rear)
{
printf("\n Queue is empty");
}
else
{
printf("\n Deleted element is %d",queue[front]);
front=front+1;
}
break;
case 3:
printf("\n Elements in the queue are: \n");
if(front==rear)
{
printf("\n Queue is empty");
}
else
{
for(i=front;i<rear;i++)
{
printf("%d",queue[i]);
printf("\n");
}
break;
case 4:
exit(0);
default:
printf("\n wrong choice");
}
}
}
```

return 0;

}

OUTPUT:-



```
1.Insertion
 2.Deletion
 3.Display
 4.Exit
 Enter the choice: 1
 Enter elements to insert: 3
 Enter the choice: 1
 Enter elements to insert: 2
 Enter the choice: 3
 Elements in the queue are:
3
2

 Enter the choice: 2
 Deleted element is 3
 Enter the choice: 3
 Elements in the queue are:
2

 Enter the choice: 4
```

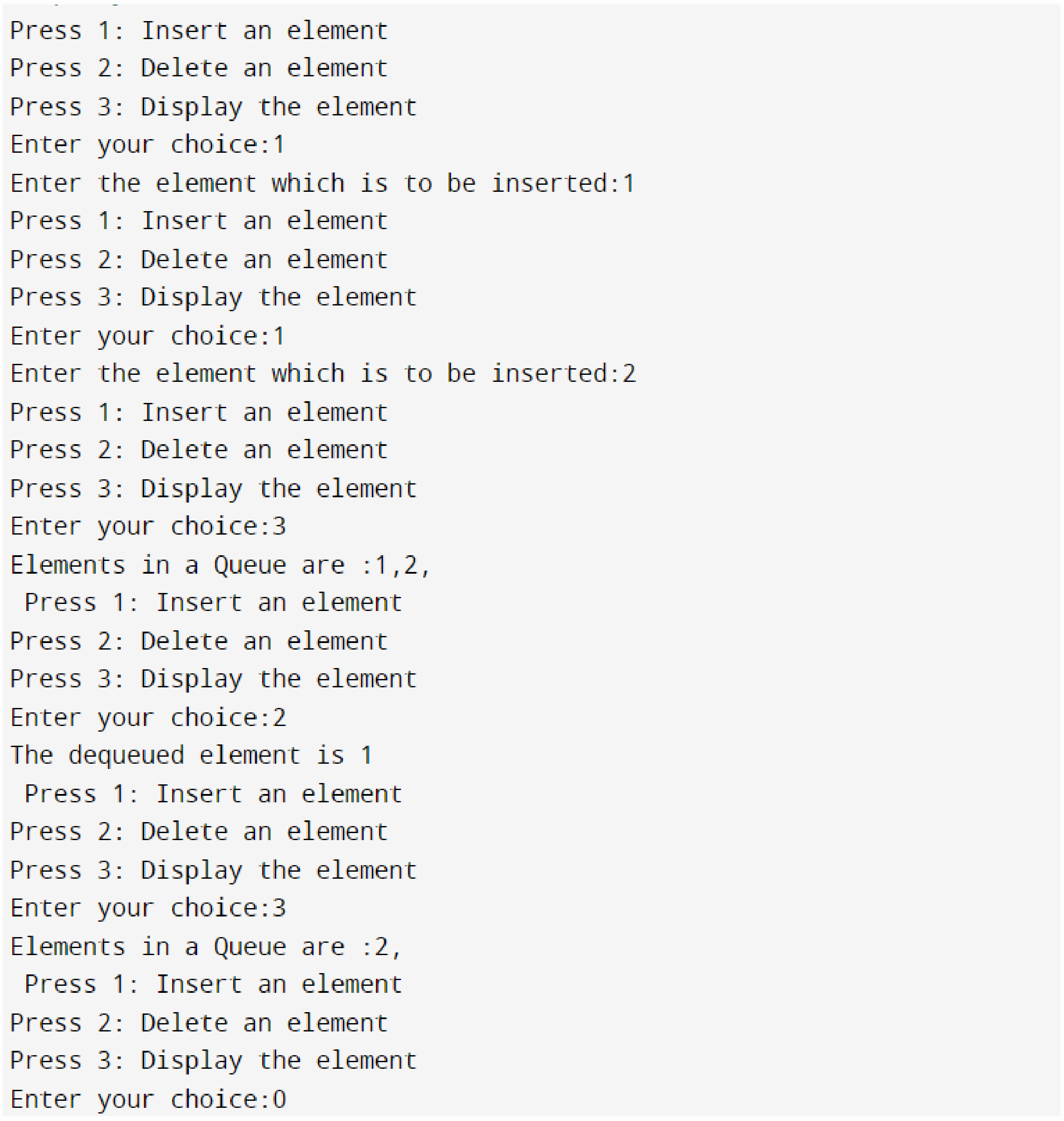
9.CIRCULAR QUEUE OPERATION

CODE:-

```
#include <stdio.h>
# define max 6
int queue[max];
int front=-1;
int rear=-1;
void enqueue(int element)
{
  if(front==-1 && rear==-1)
  {
      front=0;
      rear=0;
      queue[rear]=element;
  }
  else if((rear+1)%max==front)    {
      printf("Queue is overflow..");
  }
  else    {
      rear=(rear+1)%max;
      queue[rear]=element;
  }
}
int dequeue()
{
  if((front==-1) && (rear==-1))    {
      printf("\nQueue is underflow..");
  }
  else if(front==rear)    {
      printf("\nThe dequeued element is %d", queue[front]);
      front=-1;
      rear=-1;
  }
  else    {
      printf("\nThe dequeued element is %d", queue[front]);
      front=(front+1)%max;
  }
}
void display()
{
  int i=front;
  if(front==-1 && rear==-1)    {
      printf("\n Queue is empty..");
  }
  else    {
      printf("\nElements in a Queue are :");
      while(i<=rear)       {
          printf("%d,", queue[i]);
          i=(i+1)%max;
      }
  }
}
int main()
{
  int choice=1,x;
  while(choice<4 && choice!=0)    {
      printf("\n Press 1: Insert an element");
      printf("\nPress 2: Delete an element");
      printf("\nPress 3: Display the element");
      printf("\nEnter your choice:");
      scanf("%d", &choice);
      switch(choice)       {
          case 1:
          printf("Enter the element which is to be inserted:");
          scanf("%d", &x);
          enqueue(x);
          break;
          case 2:
          dequeue();
          break;
          case 3:
          display();
      }
```

```
  }
}
```

OUTPUT:-



```
Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice:1
Enter the element which is to be inserted:1
Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice:1
Enter the element which is to be inserted:2
Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice:3
Elements in a Queue are :1,2,
 Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice:2
The dequeued element is 1
 Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice:3
Elements in a Queue are :2,
 Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice:0
```

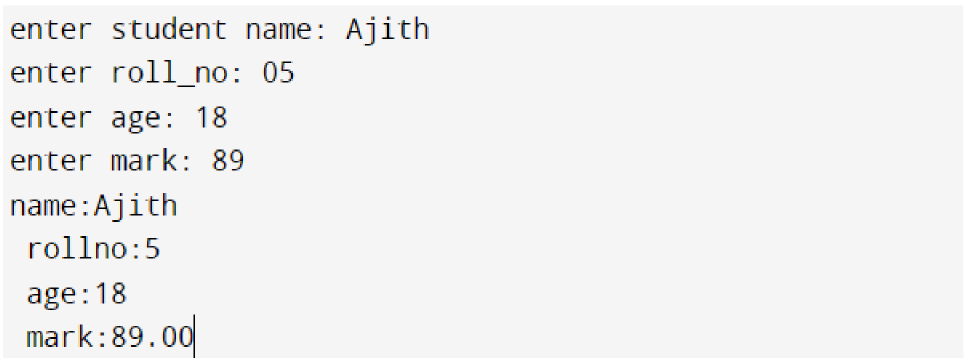
10.STRUCTURE IMPLEMENTATION

Code:

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

```
{
   char name[' '];
   int roll_no;
   int age;
   float mark;
}s;
void main()
{
printf("enter student name: ");
gets(s.name);
printf("enter roll_no: ");
scanf("%d",&s.roll_no);
printf("enter age: ");
scanf("%d",&s.age);
printf("enter mark: ");
scanf("%f",&s.mark);
printf("\n name:%s",s.name);
printf("\n rollno:%d",s.roll_no);
printf("\n age:%d",s.age);
printf("\n mark:%.2f",s.mark);
}
```

OUTPUT:-



```
enter student name: Ajith
enter roll_no: 05
enter age: 18
enter mark: 89
name:Ajith
 rollno:5
 age:18
 mark:89.00
```

<u>11.LINKED LIST IMPLEMENTATION</u>

<u>CODE:</u>

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node *head;
void beginsert ();
void lastinsert ();
void randominsert();
void begin_delete();
void last_delete();
void random_delete();
void display();
void search();
void main ()
{
   int choice =0;
   while(choice != 9)
   {
      printf("\n\n*********Main Menu*********\n");
      printf("\nChoose one option from the following list ...\n");
      printf("\n===============================================\n");
printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random location\n4.Delete
from Beginning\n 5.Delete from last\n6.Delete node after specified location\n7.Search for
an element\n8.Show\n9.Exit\n");
    printf("\nEnter your choice?\n");
    scanf("\n%d",&choice);
    switch(choice)
    {
      case 1:
      beginsert();
      break;
      case 2:
      lastinsert();
      break;
      case 3:
      randominsert();
      break;
      case 4:
      begin_delete();
      break;
      case 5:
      last_delete();
      break;
      case 6:
      random_delete();
      break;
      case 7:
      search();
      break;
      case 8:
      display();
      break;
```

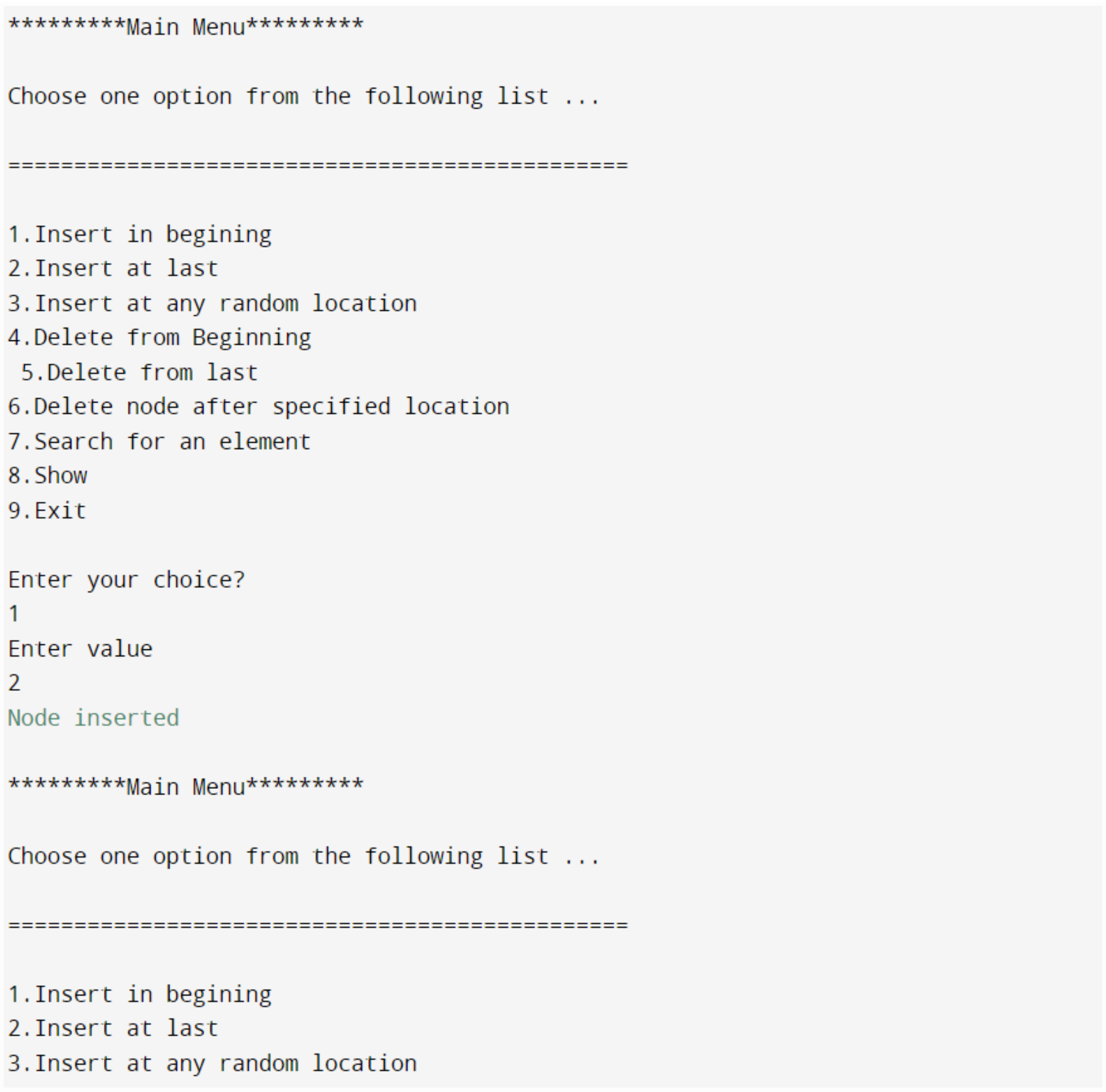
```
            case 9:
            exit(0);
            break;
            default:
            printf("Please enter valid choice..");
        }
    }
}
void beginsert()
{
    struct node *ptr;
    int item;
    ptr = (struct node *) malloc(sizeof(struct node *));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter value\n");
        scanf("%d",&item);
        ptr->data = item;
        ptr->next = head;
        head = ptr;
        printf("\nNode inserted");
    }

}
void lastinsert()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node*)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter value?\n");
        scanf("%d",&item);
        ptr->data = item;
        if(head == NULL)
        {
            ptr -> next = NULL;
            head = ptr;
            printf("\nNode inserted");
        }
        else
        {
            temp = head;
            while (temp -> next != NULL)
            {
                temp = temp -> next;
            }
            temp->next = ptr;
            ptr->next = NULL;
            printf("\nNode inserted");

        }
    }
}
```

```
void randominsert()
{
  int i,loc,item;
  struct node *ptr, *temp;
  ptr = (struct node *) malloc (sizeof(struct node));
  if(ptr == NULL)
  {
    printf("\nOVERFLOW");
  }
  else
  {
    printf("\nEnter element value");
    scanf("%d",&item);
    ptr->data = item;
    printf("\nEnter the location after which you want to insert ");
    scanf("\n%d",&loc);
    temp=head;
    for(i=0;i<loc;i++)
    {
      temp = temp->next;
      if(temp == NULL)
      {
        printf("\ncan't insert\n");
        return;
      }

    }
    ptr ->next = temp ->next;
    temp ->next = ptr;
    printf("\nNode inserted");
  }
```

```
}
void begin_delete()
{
   struct node *ptr;
   if(head == NULL)
   {
      printf("\nList is empty\n");
   }
   else
   {
      ptr = head;
      head = ptr->next;
      free(ptr);
      printf("\nNode deleted from the begining ...\n");
   }
}
void last_delete()
{
   struct node *ptr,*ptr1;
   if(head == NULL)
   {
      printf("\nlist is empty");
   }
   else if(head -> next == NULL)
   {
      head = NULL;
      free(head);
      printf("\nOnly node of the list deleted ...\n");
   }
   else
   {
        ptr = head;
        while(ptr->next != NULL)
        {
            ptr1 = ptr;
            ptr = ptr ->next;
        }
        ptr1->next = NULL;
        free(ptr);
        printf("\nDeleted Node from the last ...\n");
    }
}
void random_delete()
{
    struct node *ptr,*ptr1;
    int loc,i;
    printf("\n Enter the location of the node after which you want to perform deletion \n");
    scanf("%d",&loc);
    ptr=head;
    for(i=0;i<loc;i++)
    {
        ptr1 = ptr;
        ptr = ptr->next;
        if(ptr == NULL)
        {
            printf("\nCan't delete");
            return;
        }
    }
    ptr1 ->next = ptr ->next;
    free(ptr);
    printf("\nDeleted node %d ",loc+1);
```

```
}
void search()
{
   struct node *ptr;
   int item,i=0,flag;
   ptr = head;
   if(ptr == NULL)    {
      printf("\nEmpty List\n");
   }
   else     {
      printf("\nEnter item which you want to search?\n");
      scanf("%d",&item);
      while (ptr!=NULL)       {
         if(ptr->data == item)          {
            printf("item found at location %d ",i+1);
            flag=0;
         }
         else          {
            flag=1;
         }
         i++;
         ptr = ptr -> next;
      }
      if(flag==1)       {
         printf("Item not found\n");
      }
   }

}
void display()
{
    struct node *ptr;
    ptr = head;
    if(ptr == NULL)    {
        printf("Nothing to print");
    }
    else     {
        printf("\nprinting values . . . . .\n");
        while (ptr!=NULL)       {
            printf("\n%d",ptr->data);
            ptr = ptr -> next;
        }
    }
}
```

<u>OUTPUT:-</u>



```
*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert in begining
2.Insert at last
3.Insert at any random location
4.Delete from Beginning
 5.Delete from last
6.Delete node after specified location
7.Search for an element
8.Show
9.Exit

Enter your choice?
1
Enter value
2
Node inserted

*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert in begining
2.Insert at last
3.Insert at any random location
```

12.DOUBLY LINKED LIST IMPLEMENTATION

CODE:

```
#include<stdio.h>

#include<stdlib.h>

struct node

{

  struct node *prev;

  struct node *next;

  int data;

};
```

```
struct node *head;
void insertion_beginning();
void insertion_last();
void insertion_specified();
void deletion_beginning();
void deletion_last();
void deletion_specified();
void display();
void search();
void main ()
{
int choice =0;
    while(choice != 9)
    {
        printf("\n*********Main Menu*********\n");
        printf("\nChoose one option from the following list ...\n");
        printf("\n===============================================\n");
        printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random location\n4.Delete from Beginning\n5.Delete from last\n6.Delete the node after the given data\n7.Search\n8.Show\n9.Exit\n");
        printf("\nEnter your choice?\n");
        scanf("\n%d",&choice);
        switch(choice)
        {
            case 1:
            insertion_beginning();
            break;
            case 2:
                    insertion_last();
            break;
            case 3:
```

```
            insertion_specified();
            break;
            case 4:
            deletion_beginning();
            break;
            case 5:
            deletion_last();
            break;
            case 6:
            deletion_specified();
            break;
            case 7:
            search();
            break;
            case 8:
            display();
            break;
            case 9:
            exit(0);
            break;
            default:
            printf("Please enter valid choice..");
        }
    }
}
void insertion_beginning()
{
   struct node *ptr;
   int item;
   ptr = (struct node *)malloc(sizeof(struct node));
   if(ptr == NULL)
```

```
  {
      printf("\nOVERFLOW");
  }
  else
  {
   printf("\nEnter Item value");
   scanf("%d",&item);

  if(head==NULL)
  {
      ptr->next = NULL;
      ptr->prev=NULL;
      ptr->data=item;
      head=ptr;
  }
  else
  {
      ptr->data=item;
      ptr->prev=NULL;
      ptr->next = head;
      head->prev=ptr;
      head=ptr;
  }
  printf("\nNode inserted\n");
}

}
void insertion_last()
{
  struct node *ptr,*temp;
  int item;
  ptr = (struct node *) malloc(sizeof(struct node));
  if(ptr == NULL)
  {
      printf("\nOVERFLOW");
  }
  else
  {
      printf("\nEnter value");
      scanf("%d",&item);
       ptr->data=item;
      if(head == NULL)
      {
          ptr->next = NULL;
          ptr->prev = NULL;
          head = ptr;
      }
      else
      {
         temp = head;
         while(temp->next!=NULL)
         {
             temp = temp->next;
         }
         temp->next = ptr;
         ptr ->prev=temp;
         ptr->next = NULL;
         }
      }
    printf("\n node inserted\n");
   }
void insertion_specified()
```

```
{
  struct node *ptr,*temp;
  int item,loc,i;
  ptr = (struct node *)malloc(sizeof(struct node));
  if(ptr == NULL)
  {
      printf("\n OVERFLOW");
  }
  else
  {
      temp=head;
      printf("Enter the location");
      scanf("%d",&loc);
      for(i=0;i<loc;i++)
      {
          temp = temp->next;
          if(temp == NULL)
          {
              printf("\n There are less than %d elements", loc);
              return;
          }
      }
      printf("Enter value");
      scanf("%d",&item);
      ptr->data = item;
      ptr->next = temp->next;
      ptr -> prev = temp;
      temp->next = ptr;
      temp->next->prev=ptr;
      printf("\n node inserted\n");
  }
```

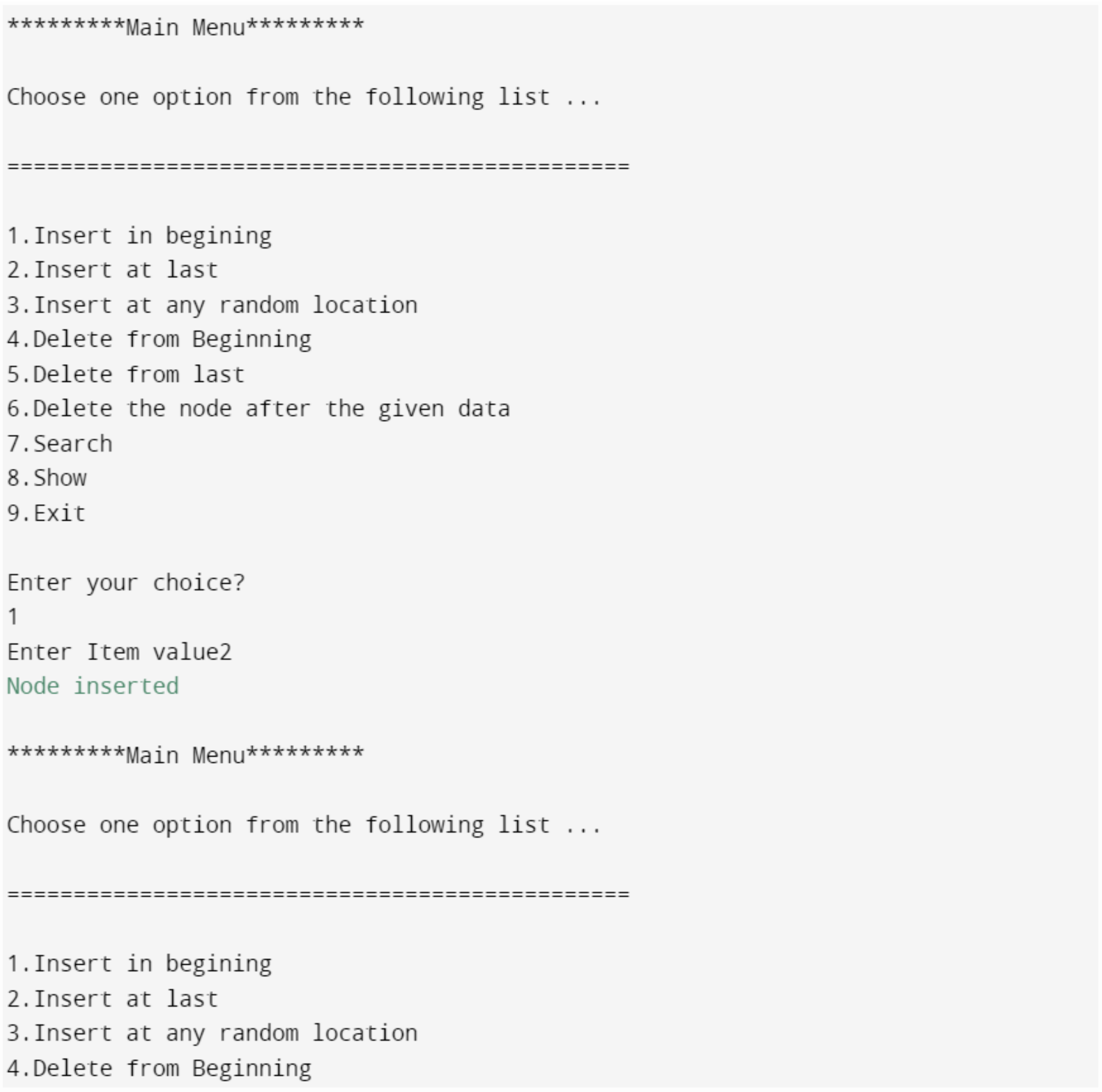
```
}
void deletion_beginning()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\n UNDERFLOW");
    }
    else if(head->next == NULL)
    {
        head = NULL;
        free(head);
        printf("\n node deleted\n");
    }
    else
    {
        ptr = head;
        head = head -> next;
        head -> prev = NULL;
        free(ptr);
        printf("\n node deleted\n");
    }
}
void deletion_last()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\n UNDERFLOW");
    }
    else if(head->next == NULL)
```

```
  {
    head = NULL;
    free(head);
    printf("\n node deleted\n");
  }
  else
  {
    ptr = head;
    if(ptr->next != NULL)
    {
      ptr = ptr -> next;
    }
    ptr -> prev -> next = NULL;
    free(ptr);
    printf("\n node deleted\n");
  }
}
void deletion_specified()
{
  struct node *ptr, *temp;
  int val;
  printf("\n Enter the data after which the node is to be deleted : ");
  scanf("%d", &val);
  ptr = head;
  while(ptr -> data != val)
  ptr = ptr -> next;
  if(ptr -> next == NULL)
  {
    printf("\nCan't delete\n");
  }
  else if(ptr -> next -> next == NULL)
```

```
    {
        ptr ->next = NULL;
    }
    else
    {
        temp = ptr -> next;
        ptr -> next = temp -> next;
        temp -> next -> prev = ptr;
        free(temp);
        printf("\n node deleted\n");
    }
}
void display()
{
    struct node *ptr;
    printf("\n printing values...\n");
    ptr = head;
    while(ptr != NULL)
    {
        printf("%d\n",ptr->data);
        ptr=ptr->next;
    }
}
void search()
{
    struct node *ptr;
    int item,i=0,flag;
    ptr = head;
    if(ptr == NULL)
    {
        printf("\nEmpty List\n");
```

```
  }
  else
  {
    printf("\nEnter item which you want to search?\n");
    scanf("%d",&item);
    while (ptr!=NULL)
    {
      if(ptr->data == item)
      {
        printf("\nitem found at location %d ",i+1);
        flag=0;
        break;
      }
      else
      {
        flag=1;
      }
      i++;
      ptr = ptr -> next;
    }
    if(flag==1)
    {
      printf("\nItem not found\n");
    }
  }

}
```

OUTPUT:-



```
*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert in begining
2.Insert at last
3.Insert at any random location
4.Delete from Beginning
5.Delete from last
6.Delete the node after the given data
7.Search
8.Show
9.Exit

Enter your choice?
1
Enter Item value2
Node inserted

*********Main Menu*********

Choose one option from the following list ...

===============================================

1.Insert in begining
2.Insert at last
3.Insert at any random location
4.Delete from Beginning
```

13.BINARY SEARCH TREE

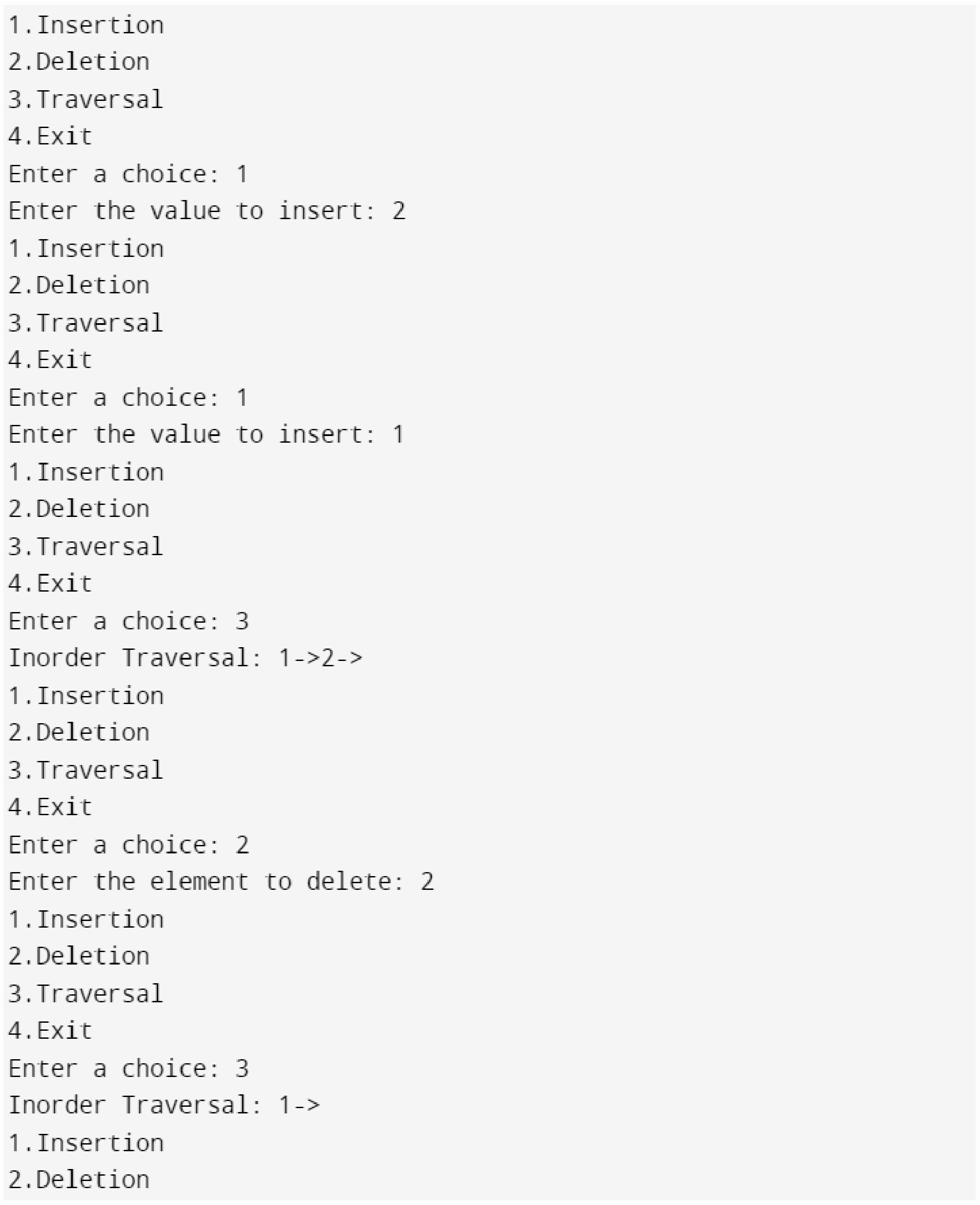
CODE:

```
#include<stdio.h>
#include<stdlib.h>
struct node {
int key;
```

```
struct node *left,*right;
};
struct node *newnode(int item) {
struct node *temp;
temp=(struct node *)malloc(sizeof(struct node));
temp->key=item;
temp->left=temp->right=NULL;
return temp;
}
void inorder(struct node *root) {
if(root!=NULL) {
inorder(root->left);
printf("%d->",root->key);
inorder(root->right);
}
}
struct node *insert(struct node *node,int key) {
if(node==NULL)
return newnode(key);
if(key<node->key)
node->left=insert(node->left,key);
else
node->right=insert(node->right,key);
return node;
}
struct node *minvalue(struct node *node) {
struct node *current=node;
while(current && current->left!=NULL){
current=current->left;
return current;
}
```

```
}
struct node *deletenode(struct node *root,int key) {
if(root==NULL)
return root;
if(key<root->key)
root->left=deletenode(root->left,key);
else if (key>root->key)
root->right=deletenode(root->right,key);
else {
if(root->left==NULL) {
struct node *temp=root->right;
free(root);
return temp;
}
else if(root->right==NULL) {
struct node *temp=root->left;
free(root);
return temp;
}
struct node *temp=minvalue(root->right);
root->key=temp->key;
root->right=deletenode(root->right,temp->key);
}
return root;
}
void main()
{
struct node *root=NULL;
int choice,n;
while(1) {
printf("\n1.Insertion \n2.Deletion \n3.Traversal \n4.Exit");
printf("\nEnter a choice: ");
scanf("%d",&choice);
switch(choice)
{
case 1:
printf("\nEnter the value to insert: ");
scanf("%d",&n);
root=insert(root,n);
break;
case 2:
printf("\nEnter the element to delete: ");
scanf("%d",&n);
root=deletenode(root,n);
break;
case 3:
printf("\nInorder Traversal: ");
inorder(root);
break;
case 4:
exit(0);
break;
default:
printf("\nWrong choice");
break;
}
}
}
```

Output: -



```
1.Insertion
2.Deletion
3.Traversal
4.Exit
Enter a choice: 1
Enter the value to insert: 2
1.Insertion
2.Deletion
3.Traversal
4.Exit
Enter a choice: 1
Enter the value to insert: 1
1.Insertion
2.Deletion
3.Traversal
4.Exit
Enter a choice: 3
Inorder Traversal: 1->2->
1.Insertion
2.Deletion
3.Traversal
4.Exit
Enter a choice: 2
Enter the element to delete: 2
1.Insertion
2.Deletion
3.Traversal
4.Exit
Enter a choice: 3
Inorder Traversal: 1->
1.Insertion
2.Deletion
```

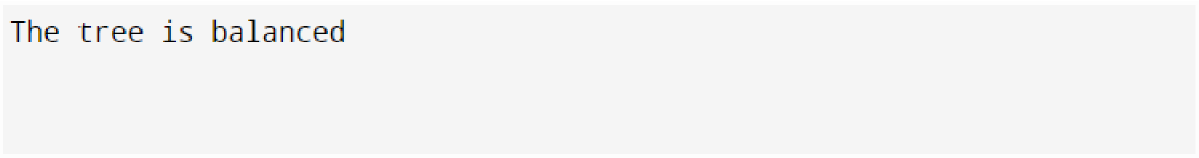
14.BALANCED BINARY SEARCH TREE IMPLEMENTATION

CODE:

```
#include <stdio.h>

#include <stdlib.h>
```

```
struct node {
 int item;
 struct node *left;
 struct node *right;
};
struct node *newNode(int item) {
 struct node *node = (struct node *)malloc(sizeof(struct node));
 node->item = item;
 node->left = NULL;
 node->right = NULL;
 return (node);
}
int checkHeightBalance(struct node *root, int *height) {
 int leftHeight = 0, rightHeight = 0;
 int l = 0, r = 0;
 if (root == NULL) {
   *height = 0;
   return 1;
 }

 l = checkHeightBalance(root->left, &leftHeight);
 r = checkHeightBalance(root->right, &rightHeight);
 *height = (leftHeight > rightHeight ? leftHeight : rightHeight) + 1;
 if ((leftHeight - rightHeight >= 2) || (rightHeight - leftHeight >= 2))
   return 0;
 else
   return l && r;
}
int main() {
 int height = 0;
 struct node *root = newNode(1);
 root->left = newNode(2);
 root->right = newNode(3);
 root->left->left = newNode(4);
 root->left->right = newNode(5);
 if (checkHeightBalance(root, &height))
   printf("The tree is balanced\n");
}
```

OUTPUT: -



```
The tree is balanced
```

15.SET OPERATION IMPLEMENTATION

CODE:

```
#include<stdio.h>
void main()
{
int a[10],b[10],c[10],i,j,k,p,choice,flag,n1,n2;
int wish;
printf("\nEnter size of set A: ");
scanf("%d",&n1);
printf("\nEnter elements of set A: ");
for(i=0;i<n1;i++)
{
scanf("%d",&a[i]);
}
printf("\nEnter size of set B: ");
scanf("%d",&n2);
printf("\nEnter elements of set B: ");
for(i=0;i<n2;i++)
{
```

```
scanf("%d",&b[i]);
}
do {
printf("\n1.Union \n2.Intersection \n3.Difference");
printf("\nEnter a choice: ");
scanf("%d",&choice);
switch(choice)
{
case 1:
k=0;
for(i=0;i<n1;i++)
{
c[k]=a[i];
k++;
}
for(i=0;i<n2;i++)
{
flag=1;
for(j=0;j<n1;j++)
{
if(b[i]==a[j])
{
flag=0;
break;
}
}
if(flag==1)
{
c[k]=b[i];
k++;
}
}
p=k;
for(k=0;k<p;k++)
{
printf("%d\t",c[k]);
}
break;
case 2:
k=0;
for(i=0;i<n2;i++)
{
flag=1;
for(j=0;j<n1;j++)
{
if(b[i]==a[j])
{
flag=0;
break;
}
}
if(flag==0)
{
c[k]=b[i];
k++;
}
}
p=k;
for(k=0;k<p;k++)
{
printf("%d\t",c[k]);
}
```

```
break;
case 3:
k=0;
for(i=0;i<n1;i++)
{
flag=1;
for(j=0;j<n2;j++)
{
if(a[i]==b[j])
{
flag=0;
break;
}
}
if(flag==1)
{
c[k]=a[i];
k++;
}
}
p=k;
for(k=0;k<p;k++)
{
printf("%d\t",c[k]);
}
break;
}
printf("\npress 1 for Stop and 0 for Continue");
printf("\nDo you want to continue(0/1): ");
scanf("%d",&wish);
}
```

```
while(wish==0);
}
```

Output:-

```
Enter size of set A: 2
Enter elements of set A: 1
2
Enter size of set B: 2
Enter elements of set B: 2
4
1.Union
2.Intersection
3.Difference
Enter a choice: 1
1   2   4
press 1 for Stop and 0 for Continue
Do you want to continue(0/1): 0
1.Union
2.Intersection
3.Difference
Enter a choice: 2
2
press 1 for Stop and 0 for Continue
Do you want to continue(0/1): 0
1.Union
2.Intersection
3.Difference
Enter a choice: 3
1
press 1 for Stop and 0 for Continue
Do you want to continue(0/1): 1
```

16.DISJOINT SET IMPLEMENTATION

Code:

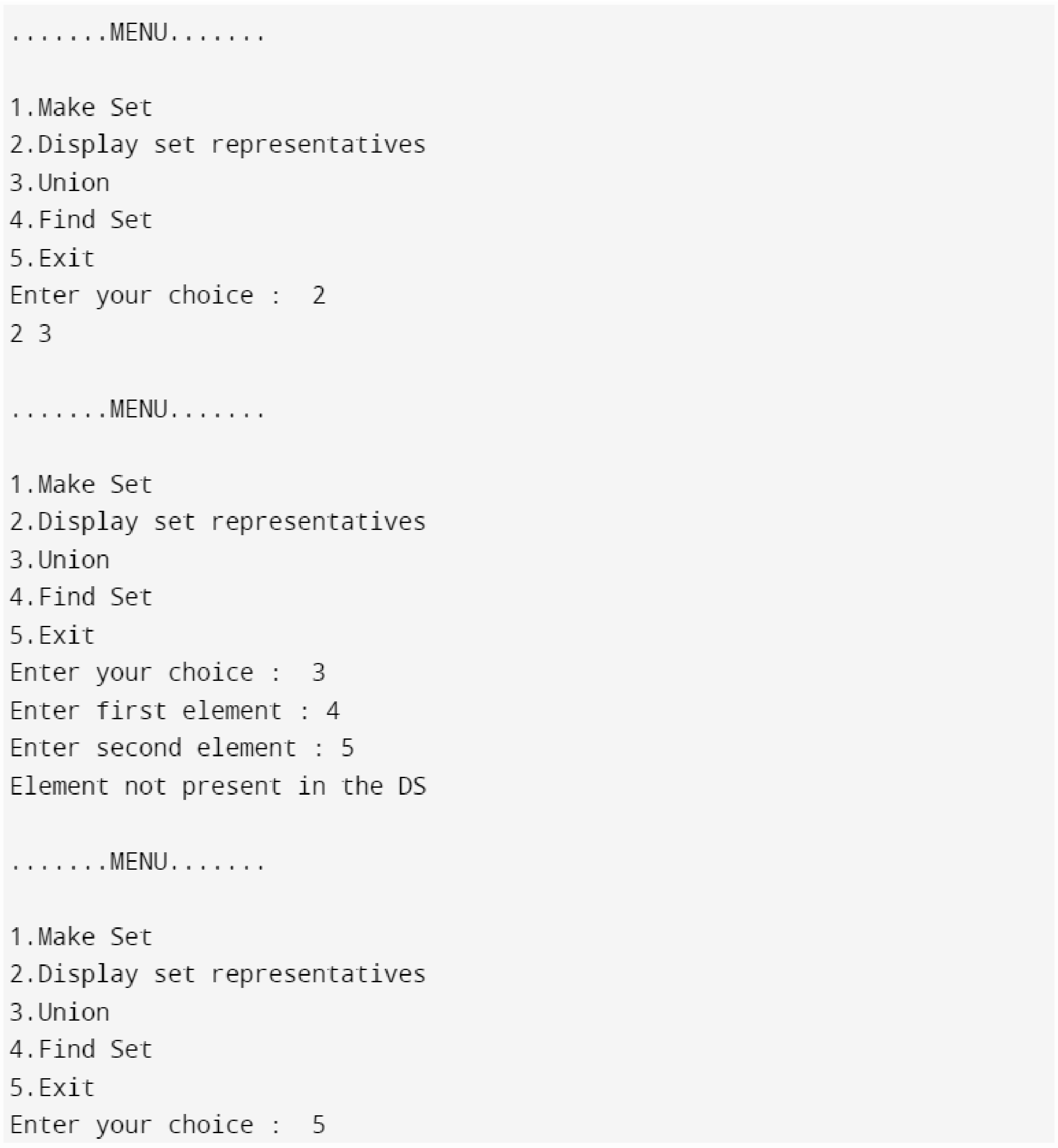
```
#include<stdio.h>
#include<stdlib.h>
```

```
struct node{
  struct node *rep;
  struct node *next;
  int data;
}*heads[50],*tails[50];
static int countRoot=0;
void makeSet(int x){
        struct node *new=(struct node *)malloc(sizeof(struct node));
        new->rep=new;
        new->next=NULL;
        new->data=x;
        heads[countRoot]=new;
        tails[countRoot++]=new;
}
struct node* find(int a){
        int i;
        struct node *tmp=(struct node *)malloc(sizeof(struct node));
        for(i=0;i<countRoot;i++){
                tmp=heads[i];
                while(tmp!=NULL){
                if(tmp->data==a)
                return tmp->rep;
                tmp=tmp->next;
                }
        }
        return NULL;
}
void unionSets(int a,int b){
        int i,pos,flag=0,j;
        struct node *tail2=(struct node *)malloc(sizeof(struct node));
        struct node *rep1=find(a);
```

```
	struct node *rep2=find(b);
	if(rep1==NULL||rep2==NULL){
		printf("\nElement not present in the DS\n");
		return;
	}
	if(rep1!=rep2){
		for(j=0;j<countRoot;j++){
			if(heads[j]==rep2){
				pos=j;
				flag=1;
				countRoot-=1;
				tail2=tails[j];
				for(i=pos;i<countRoot;i++){
					heads[i]=heads[i+1];
					tails[i]=tails[i+1];
				}
			}
			if(flag==1)
				break;
		}
		for(j=0;j<countRoot;j++){
			if(heads[j]==rep1){
				tails[j]->next=rep2;
				tails[j]=tail2;
				break;
			}
		}
		while(rep2!=NULL){
		rep2->rep=rep1;
		rep2=rep2->next;
		}
```

```
        }
}
int search(int x){
        int i;
        struct node *tmp=(struct node *)malloc(sizeof(struct node));
        for(i=0;i<countRoot;i++){
                tmp=heads[i];
                if(heads[i]->data==x)
                        return 1;
                while(tmp!=NULL){
                        if(tmp->data==x)
                                return 1;
                        tmp=tmp->next;
                }
        }
        return 0;
}
void main(){
int choice,x,i,j,y,flag=0;
        do{
                printf("\n.......MENU.......\n\n1.Make Set\n2.Display set
                representatives\n3.Union\n4.Find Set\n5.Exit\n");
                printf("Enter your choice :  ");
                scanf("%d",&choice);
                switch(choice){
                case 1:
                        printf("\nEnter new element : ");
                        scanf("%d",&x);
                        if(search(x)==1)
                                printf("\nElement already present in the disjoint set DS\n");
                        else
                                makeSet(x);
                        break;
                case 2:
                        printf("\n");
                        for(i=0;i<countRoot;i++)
                                printf("%d ",heads[i]->data);
                        printf("\n");
                        break;
                case 3:
                        printf("\nEnter first element : ");
                        scanf("%d",&x);
                        printf("\nEnter second element : ");
                        scanf("%d",&y);
                        unionSets(x,y);
                        break;
                case 4:
                        printf("\nEnter the element: ");
                        scanf("%d",&x);
                        struct node *rep=(struct node *)malloc(sizeof(struct node));
                        rep=find(x);
                        if(rep==NULL)
                        printf("\nElement not present in the DS\n");
                        else
                        printf("\nThe representative of %d is %d\n",x,rep->data);
                        break;
                case 5:
                        exit(0);
                default:
                        printf("\nWrong choice\n");
                        break;
                }
                }while(1);
}
```

Output:-



```
.......MENU.......

1.Make Set
2.Display set representatives
3.Union
4.Find Set
5.Exit
Enter your choice :  2
2 3

.......MENU.......

1.Make Set
2.Display set representatives
3.Union
4.Find Set
5.Exit
Enter your choice :  3
Enter first element : 4
Enter second element : 5
Element not present in the DS

.......MENU.......

1.Make Set
2.Display set representatives
3.Union
4.Find Set
5.Exit
Enter your choice :  5
```

17.MAX HEAP IMPLEMENTATION

CODE:

```
#include<stdio.h>
int size = 0;
void swap(int *a, int *b)
{
int temp = *b;
*b = *a;
*a = temp;
}
void heapify(int array[], int size, int i)
{
if(size == 0)
{
printf("Single element in heap");
}
else
{
int largest = i;
int l = 2 * i + 1;
int r = 2 * i + 2;
if(l < size && array[l] > array[largest])
largest = l;
if(r < size && array[r] > array[largest])
largest = r;
if(largest != i)
{
swap(&array[i], &array[largest]);
heapify(array, size, largest);
}
}
}
void insert(int array[], int newnum)
```

```
{
if(size == 0)
{
array[0] = newnum;
size += 1;
}
else
{
array[size] = newnum;
size += 1;
for(int i = size / 2 - 1 ; i >= 0; i--)
{
heapify(array, size, i);
}
}
}
void delete(int array[], int num)
{
int i;
for(i=0; i < size; i++)
{
if(num == array[i])
break;
}
swap(&array[i], &array[size - 1]);
size -= 1;
for(int i = size / 2 - 1; i >= 0; i--)
{
heapify(array, size, i);
}
}
```

```
void print(int array[], int size)
{
for(int i=0; i < size; i++)
printf("%d\t", array[i]);
printf("\n");
}
int main()
{
int array[10];
insert(array, 3);
insert(array, 4);
insert(array, 9);
insert(array, 5);
insert(array, 2);
printf("Max-heap: ");
print(array, size);
delete(array, 4);
printf("After deleting an element: ");
print(array, size);
}
```

Output

```
Max-heap: 9 5    4    3    2
After deleting an element: 9    5    2    3
```

18.MIN HEAP IMPLEMENTATION

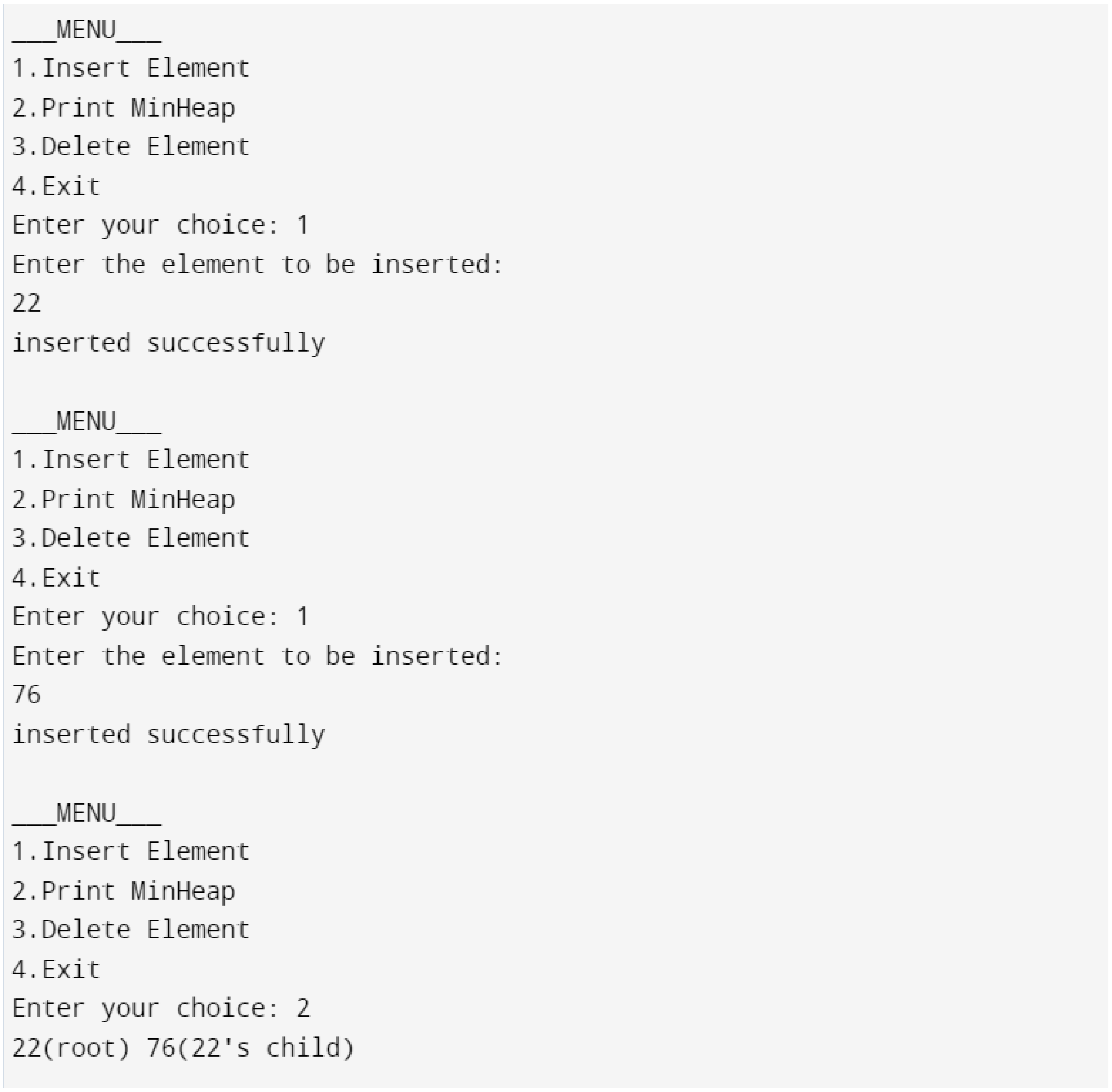
CODE:

```
#include <stdio.h>
#define HEAP_CAPACITY 10
#define SUCCESS_VAL 99999
#define FAIL_VAL -99999
int size = 0;
int i;
int heap[HEAP_CAPACITY];
void swap(int *a,int *b)
{
  int temp = *b;
  *b = *a;
  *a = temp;
}
void heapify(int i)
{
  if (size == 1)
  {
    return;
  }
  else{
    int smallest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    if(left < size && heap[left] < heap[smallest])
      smallest = left;
    if(right < size && heap[right] < heap[smallest])
      smallest = right;
    if (smallest != i)
    {
      swap(&heap[i], &heap[smallest]);
      heapify(smallest);
```

```
        }
    }
}
int insert(int newNum)
{
    if(size==0)
    {
        heap[0] = newNum;
        size += 1;
        return SUCCESS_VAL;
    }
    else if(size < HEAP_CAPACITY)
    {
        heap[size] = newNum;
        size += 1;
        for(i =(size-1)/2;i>=0;i--)
        {
            heapify(i);
        }
        return SUCCESS_VAL;
    }
    else
    {
        printf("Heap capacity reached. Insertion failed.\n");
        return FAIL_VAL;
    }
}
int delete(int number)
{
    int i,index=-1;
    if(size <=0)
    {
        printf("Empty min heap");
        return FAIL_VAL;
    }
    for(i=0;i<size;i++)
    {
        if(number == heap[i])
        {
            index = i;
            break;
        }
    }
    if(index == -1)
    {
        printf("Key is not found\n");
        return FAIL_VAL;
    }
    swap(&heap[i],&heap[size-1]);
    size -= 1;
    for(i=(size-1)/2; i>=0;i--)
    {
        heapify(i);
    }
    return SUCCESS_VAL;
}
void printHeap()
{
    for( i=0;i<size;++i)
    {
        if(i==0)
            printf("%d(root) ", heap[i]);
        else
            printf("%d(%d's child) ",heap[i],heap[(i-1)/2]);
    }
    printf("\n");
}
int main()
{
    while(1)    {
  printf("\n___MENU___\n1.Insert Element \n2.Print MinHeap \n3.Delete Element  \n4.Exit
\n");
        printf("Enter your choice: ");
        int choice;
        scanf("%d",&choice);
        if(choice==1)       {
            printf("Enter the element to be inserted: \n");
            int item;
            scanf("%d",&item);
            int res=insert(item);
            if(res==SUCCESS_VAL)
                printf("inserted successfully\n");
        }
        else if(choice==2)       {
            printHeap();
        }
        else if(choice==3)       {
            int res = delete(heap[0]);
            if(res==SUCCESS_VAL)
                printf("Delete Successfully\n");
            else       {
                printf("Deleted Unsuccessfully\n");
            }
```

```
      else if(choice==4)
      {
         break;
      }
   }
}
```

OUTPUT:-



```
___MENU___
1.Insert Element
2.Print MinHeap
3.Delete Element
4.Exit
Enter your choice: 1
Enter the element to be inserted:
22
inserted successfully

___MENU___
1.Insert Element
2.Print MinHeap
3.Delete Element
4.Exit
Enter your choice: 1
Enter the element to be inserted:
76
inserted successfully

___MENU___
1.Insert Element
2.Print MinHeap
3.Delete Element
4.Exit
Enter your choice: 2
22(root) 76(22's child)
```

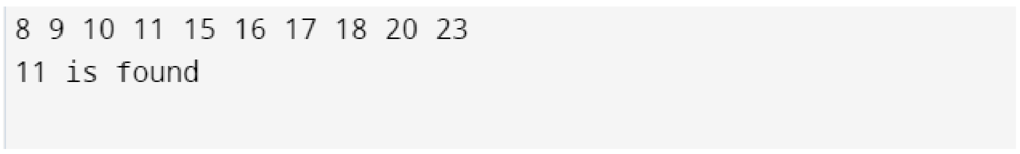
19.B-TREE IMPLEMENTATION

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 3
#define MIN 2
struct BTreeNode {
  int val[MAX + 1], count;
  struct BTreeNode *link[MAX + 1];
};
struct BTreeNode *root;
struct BTreeNode *createNode(int val, struct BTreeNode *child) {
  struct BTreeNode *newNode;
  newNode = (struct BTreeNode *)malloc(sizeof(struct BTreeNode));
  newNode->val[1] = val;
  newNode->count = 1;
  newNode->link[0] = root;
  newNode->link[1] = child;
  return newNode;
}
void insertNode(int val, int pos, struct BTreeNode *node,
        struct BTreeNode *child) {
  int j = node->count;
  while (j > pos) {
    node->val[j + 1] = node->val[j];
    node->link[j + 1] = node->link[j];
    j--;
  }
  node->val[j + 1] = val;
  node->link[j + 1] = child;
  node->count++;
}
void splitNode(int val, int *pval, int pos, struct BTreeNode *node,
        struct BTreeNode *child, struct BTreeNode **newNode) {
  int median, j;
  if (pos > MIN)
    median = MIN + 1;
  else
    median = MIN;
  *newNode = (struct BTreeNode *)malloc(sizeof(struct BTreeNode));
  j = median + 1;
  while (j <= MAX) {
    (*newNode)->val[j - median] = node->val[j];
    (*newNode)->link[j - median] = node->link[j];
    j++;
  }
  node->count = median;
  (*newNode)->count = MAX - median;
  if (pos <= MIN) {
    insertNode(val, pos, node, child);
  } else {
    insertNode(val, pos - median, *newNode, child);
  }
  *pval = node->val[node->count];
  (*newNode)->link[0] = node->link[node->count];
  node->count--;
}
int setValue(int val, int *pval,
          struct BTreeNode *node, struct BTreeNode **child) {
  int pos;
  if (!node) {
    *pval = val;
    *child = NULL;
    return 1;
```

```
 }
 if (val < node->val[1]) {
   pos = 0;
 } else {
   for (pos = node->count;
      (val < node->val[pos] && pos > 1); pos--)
     ;
   if (val == node->val[pos]) {
     printf("Duplicates are not permitted\n");
     return 0;
   }
 }
 if (setValue(val, pval, node->link[pos], child)) {
   if (node->count < MAX) {
     insertNode(*pval, pos, node, *child);
   } else {
     splitNode(*pval, pval, pos, node, *child, child);
     return 1;
   }
 }
 return 0;
}
void insert(int val) {
 int flag, i;
 struct BTreeNode *child;
 flag = setValue(val, &i, root, &child);
 if (flag)
   root = createNode(i, child);
}
void search(int val, int *pos, struct BTreeNode *myNode) {
 if (!myNode) {
   return;
 }
 if (val < myNode->val[1]) {
   *pos = 0;
 } else {
   for (*pos = myNode->count;
      (val < myNode->val[*pos] && *pos > 1); (*pos)--)
     ;
   if (val == myNode->val[*pos]) {
     printf("%d is found", val);
     return;
   }
 }
 search(val, pos, myNode->link[*pos]);
 return;
}
void traversal(struct BTreeNode *myNode) {
 int i;
 if (myNode) {
   for (i = 0; i < myNode->count; i++) {
     traversal(myNode->link[i]);
     printf("%d ", myNode->val[i + 1]);
   }
```

OUTPUT:-



```
8 9 10 11 15 16 17 18 20 23
11 is found
```

20.RED BLACK TREE IMPLEMENTATION

CODE:

```
#include <stdio.h>
#include <stdlib.h>
enum nodeColor {
  RED,
  BLACK
};
struct rbNode {
  int data, color;
  struct rbNode *link[2];
};
struct rbNode *root = NULL;
struct rbNode *createNode(int data) {
  struct rbNode *newnode;
  newnode = (struct rbNode *)malloc(sizeof(struct rbNode));
  newnode->data = data;
  newnode->color = RED;
  newnode->link[0] = newnode->link[1] = NULL;
  return newnode;
}
void insertion(int data) {
  struct rbNode *stack[98], *ptr, *newnode, *xPtr, *yPtr;
  int dir[98], ht = 0, index;
  ptr = root;
  if (!root) {
    root = createNode(data);
    return;
  }
  stack[ht] = root;
  dir[ht++] = 0;
  while (ptr != NULL) {
```

```
  if (ptr->data == data) {
    printf("Duplicates Not Allowed!!\n");
    return;
  }
  index = (data - ptr->data) > 0 ? 1 : 0;
  stack[ht] = ptr;
  ptr = ptr->link[index];
  dir[ht++] = index;
}
stack[ht - 1]->link[index] = newnode = createNode(data);
while ((ht >= 3) && (stack[ht - 1]->color == RED)) {
  if (dir[ht - 2] == 0) {
    yPtr = stack[ht - 2]->link[1];
    if (yPtr != NULL && yPtr->color == RED) {
      stack[ht - 2]->color = RED;
      stack[ht - 1]->color = yPtr->color = BLACK;
      ht = ht - 2;
    } else {
      if (dir[ht - 1] == 0) {
        yPtr = stack[ht - 1];
      } else {
        xPtr = stack[ht - 1];
        yPtr = xPtr->link[1];
        xPtr->link[1] = yPtr->link[0];
        yPtr->link[0] = xPtr;
        stack[ht - 2]->link[0] = yPtr;
      }
      xPtr = stack[ht - 2];
      xPtr->color = RED;
      yPtr->color = BLACK;
      xPtr->link[0] = yPtr->link[1];
      yPtr->link[1] = xPtr;
      if (xPtr == root) {
        root = yPtr;
      } else {
        stack[ht - 3]->link[dir[ht - 3]] = yPtr;
      }
      break;
    }
  } else {
    yPtr = stack[ht - 2]->link[0];
    if ((yPtr != NULL) && (yPtr->color == RED)) {
      stack[ht - 2]->color = RED;
      stack[ht - 1]->color = yPtr->color = BLACK;
      ht = ht - 2;
    } else {
      if (dir[ht - 1] == 1) {
        yPtr = stack[ht - 1];
      } else {
        xPtr = stack[ht - 1];
        yPtr = xPtr->link[0];
        xPtr->link[0] = yPtr->link[1];
        yPtr->link[1] = xPtr;
        stack[ht - 2]->link[1] = yPtr;
      }
      xPtr = stack[ht - 2];
      yPtr->color = BLACK;
      xPtr->color = RED;
      xPtr->link[1] = yPtr->link[0];
      yPtr->link[0] = xPtr;
      if (xPtr == root) {
        root = yPtr;
        } else {
          stack[ht - 3]->link[dir[ht - 3]] = yPtr;
        }
        break;
      }
    }
  }
  root->color = BLACK;
}
void deletion(int data) {
  struct rbNode *stack[98], *ptr, *xPtr, *yPtr;
  struct rbNode *pPtr, *qPtr, *rPtr;
  int dir[98], ht = 0, diff, i;
  enum nodeColor color;
  if (!root) {
    printf("Tree not available\n");
    return;
  }
  ptr = root;
  while (ptr != NULL) {
    if ((data - ptr->data) == 0)
      break;
    diff = (data - ptr->data) > 0 ? 1 : 0;
    stack[ht] = ptr;
    dir[ht++] = diff;
    ptr = ptr->link[diff];
  }
  if (ptr->link[1] == NULL) {
    if ((ptr == root) && (ptr->link[0] == NULL)) {
      free(ptr);
      root = NULL;
    } else if (ptr == root) {
      root = ptr->link[0];
      free(ptr);
    } else {
      stack[ht - 1]->link[dir[ht - 1]] = ptr->link[0];
    }
  } else {
    xPtr = ptr->link[1];
    if (xPtr->link[0] == NULL) {
      xPtr->link[0] = ptr->link[0];
      color = xPtr->color;
      xPtr->color = ptr->color;
      ptr->color = color;
      if (ptr == root) {
        root = xPtr;
      } else {
        stack[ht - 1]->link[dir[ht - 1]] = xPtr;
      }
      dir[ht] = 1;
      stack[ht++] = xPtr;
    } else {
      i = ht++;
      while (1) {
        dir[ht] = 0;
        stack[ht++] = xPtr;
        yPtr = xPtr->link[0];
        if (!yPtr->link[0])
          break;
        xPtr = yPtr;
      }
      dir[i] = 1;
      stack[i] = yPtr;
      if (i > 0)
        stack[i - 1]->link[dir[i - 1]] = yPtr;
      yPtr->link[0] = ptr->link[0];
      xPtr->link[0] = yPtr->link[1];
      yPtr->link[1] = ptr->link[1];
      if (ptr == root) {
        root = yPtr;
      }
      color = yPtr->color;
      yPtr->color = ptr->color;
      ptr->color = color;
    }
  }

  if (ht < 1)
    return;
  if (ptr->color == BLACK) {
    while (1) {
      pPtr = stack[ht - 1]->link[dir[ht - 1]];
      if (pPtr && pPtr->color == RED) {
        pPtr->color = BLACK;
        break;
      }
      if (ht < 2)
        break;
      if (dir[ht - 2] == 0) {
        rPtr = stack[ht - 1]->link[1];
        if (!rPtr)
          break;
```

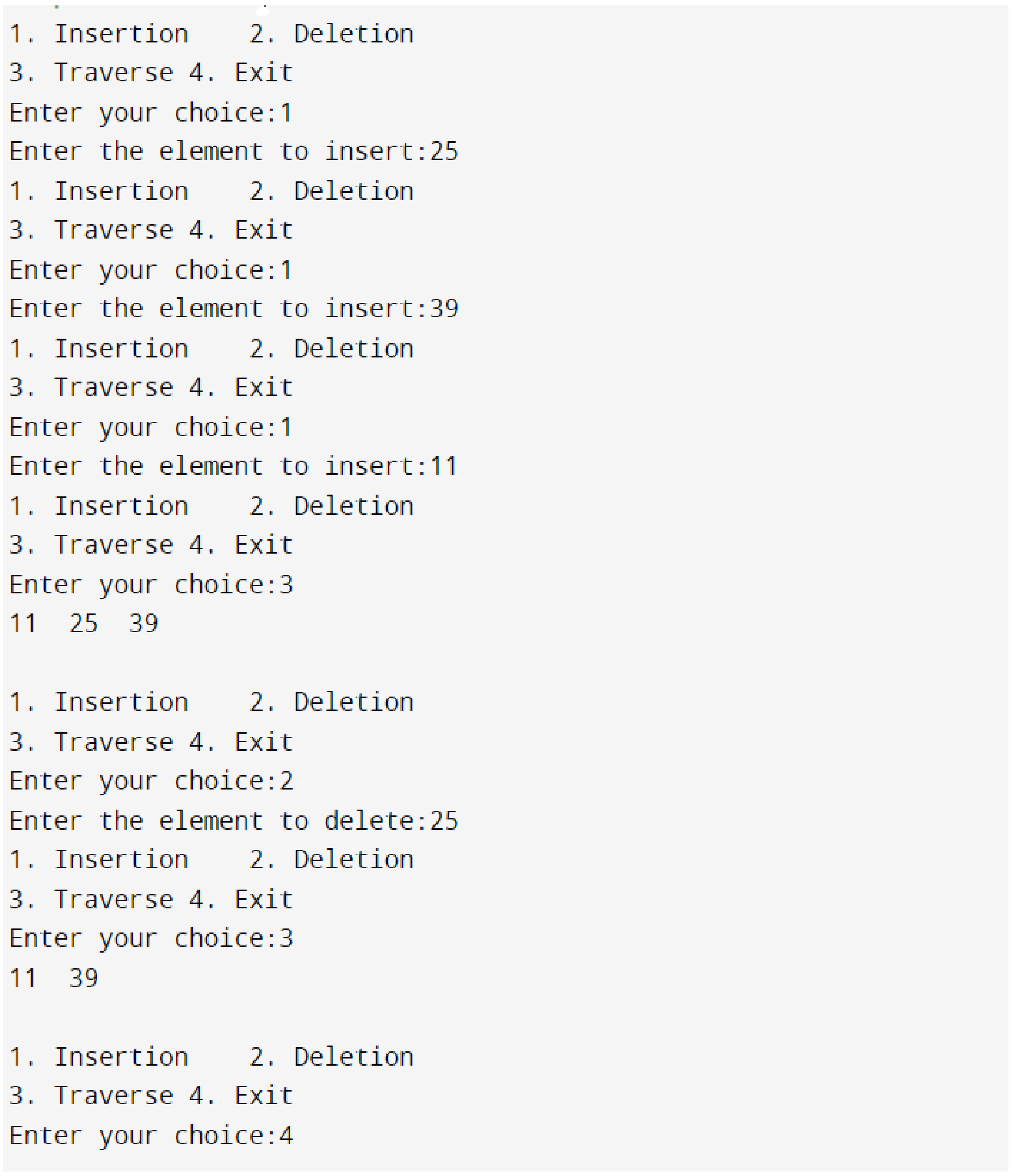
```
if (rPtr->color == RED) {
  stack[ht - 1]->color = RED;
  rPtr->color = BLACK;
  stack[ht - 1]->link[1] = rPtr->link[0];
  rPtr->link[0] = stack[ht - 1];
  if (stack[ht - 1] == root) {
    root = rPtr;
  } else {
    stack[ht - 2]->link[dir[ht - 2]] = rPtr;
  }
  dir[ht] = 0;
  stack[ht] = stack[ht - 1];
  stack[ht - 1] = rPtr;
  ht++;
  rPtr = stack[ht - 1]->link[1];
}
if ((!rPtr->link[0] || rPtr->link[0]->color == BLACK) &&
  (!rPtr->link[1] || rPtr->link[1]->color == BLACK)) {
  rPtr->color = RED;
} else {
  if (!rPtr->link[1] || rPtr->link[1]->color == BLACK) {
    qPtr = rPtr->link[0];
    rPtr->color = RED;
    qPtr->color = BLACK;
    rPtr->link[0] = qPtr->link[1];
    qPtr->link[1] = rPtr;
    rPtr = stack[ht - 1]->link[1] = qPtr;
  }
  rPtr->color = stack[ht - 1]->color;
  stack[ht - 1]->color = BLACK;
  rPtr->link[1]->color = BLACK;
```

```
          stack[ht - 1]->link[1] = rPtr->link[0];
          rPtr->link[0] = stack[ht - 1];
          if (stack[ht - 1] == root) {
            root = rPtr;
          } else {
            stack[ht - 2]->link[dir[ht - 2]] = rPtr;
          }
          break;
        }
      } else {
        rPtr = stack[ht - 1]->link[0];
        if (!rPtr)
          break;
        if (rPtr->color == RED) {
          stack[ht - 1]->color = RED;
          rPtr->color = BLACK;
          stack[ht - 1]->link[0] = rPtr->link[1];
          rPtr->link[1] = stack[ht - 1];
          if (stack[ht - 1] == root) {
            root = rPtr;
          } else {
            stack[ht - 2]->link[dir[ht - 2]] = rPtr;
          }
          dir[ht] = 1;
          stack[ht] = stack[ht - 1];
          stack[ht - 1] = rPtr;
          ht++;
          rPtr = stack[ht - 1]->link[0];
        }
        if ((!rPtr->link[0] || rPtr->link[0]->color == BLACK) &&  (!rPtr->link[1] ||
rPtr->link[1]->color == BLACK)) {
```

```
          rPtr->color = RED;
        } else {
          if (!rPtr->link[0] || rPtr->link[0]->color == BLACK) {
            qPtr = rPtr->link[1];
            rPtr->color = RED;
            qPtr->color = BLACK;
            rPtr->link[1] = qPtr->link[0];
            qPtr->link[0] = rPtr;
            rPtr = stack[ht - 1]->link[0] = qPtr;
          }
          rPtr->color = stack[ht - 1]->color;
          stack[ht - 1]->color = BLACK;
          rPtr->link[0]->color = BLACK;
          stack[ht - 1]->link[0] = rPtr->link[1];
          rPtr->link[1] = stack[ht - 1];
          if (stack[ht - 1] == root) {
            root = rPtr;
          } else {
            stack[ht - 2]->link[dir[ht - 2]] = rPtr;
          }
          break;
        }
      }
      ht--;
    }
  }
}
void inorderTraversal(struct rbNode *node) {
  if (node) {
    inorderTraversal(node->link[0]);
    printf("%d  ", node->data);
    inorderTraversal(node->link[1]);
  }
  return;
}
int main() {
  int ch, data;
  while (1) {
    printf("1. Insertion\t2. Deletion\n");
    printf("3. Traverse\t4. Exit");    printf("\nEnter your choice:");
    scanf("%d", &ch);
    switch (ch) {
      case 1:
        printf("Enter the element to insert:");
        scanf("%d", &data);
        insertion(data);
        break;
      case 2:
        printf("Enter the element to delete:");
        scanf("%d", &data);
        deletion(data);
        break;
      case 3:
        inorderTraversal(root);
        printf("\n");
        break;
      case 4:
        exit(0);
      default:
        printf("Not available\n");
        break;
    }
```

```
   printf("\n");
 }
 return 0;
}
```

<u>OUTPUT:-</u>



```
1. Insertion    2. Deletion
3. Traverse 4. Exit
Enter your choice:1
Enter the element to insert:25
1. Insertion    2. Deletion
3. Traverse 4. Exit
Enter your choice:1
Enter the element to insert:39
1. Insertion    2. Deletion
3. Traverse 4. Exit
Enter your choice:1
Enter the element to insert:11
1. Insertion    2. Deletion
3. Traverse 4. Exit
Enter your choice:3
11  25  39

1. Insertion    2. Deletion
3. Traverse 4. Exit
Enter your choice:2
Enter the element to delete:25
1. Insertion    2. Deletion
3. Traverse 4. Exit
Enter your choice:3
11  39

1. Insertion    2. Deletion
3. Traverse 4. Exit
Enter your choice:4
```

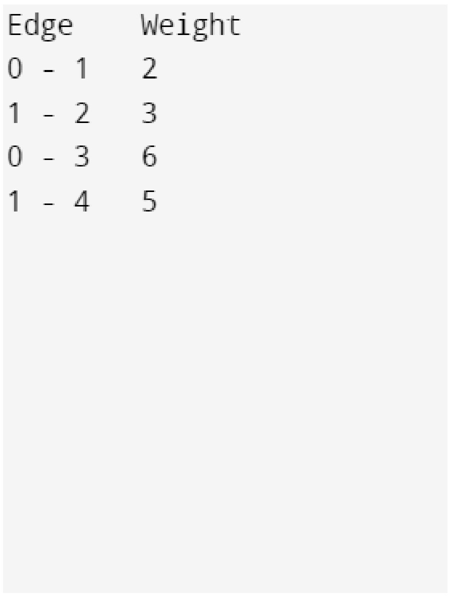
<u>21.IMPLEMENTATION OF PRIMS ALGORITHM</u>

<u>CODE:</u>

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
#define V 5
int minKey(int key[], bool mstSet[])
{
	int min = INT_MAX, min_index;
	for (int v = 0; v < V; v++)
		if (mstSet[v] == false && key[v] < min)
			min = key[v], min_index = v;
	return min_index;
}
int printMST(int parent[], int graph[V][V])
{
	printf("Edge \tWeight\n");
	for (int i = 1; i < V; i++)
		printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);
}
void primMST(int graph[V][V])
{
	int parent[V];
	int key[V];
	bool mstSet[V];
	for (int i = 0; i < V; i++)
		key[i] = INT_MAX, mstSet[i] = false;
	key[0] = 0;
	parent[0] = -1;
	for (int count = 0; count < V - 1; count++) {
		int u = minKey(key, mstSet);
		mstSet[u] = true;
		for (int v = 0; v < V; v++)
```

```
                    if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])
                        parent[v] = u, key[v] = graph[u][v];
    }
    printMST(parent, graph);
}
int main()
{
    int graph[V][V] = { { 0, 2, 0, 6, 0 },
                        { 2, 0, 3, 8, 5 },
                        { 0, 3, 0, 0, 7 },
                        { 6, 8, 0, 0, 9 },
                        { 0, 5, 7, 9, 0 } };
    primMST(graph);
    return 0;
}
```

OUTPUT:-



```
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
```