Experiment 11 Date: 12.10.2023

Circular Queue

Aim:

Program to implement circular queue using array

Algorithm:

```
main()
1: declare front,rear,count,q[5],size as global variable
2: declare item,choice,i
3: display 1.Insertion 2.deletion 3.traversal 4.exit
4: reapeat the step 5 to 8 while(true)
5: read choice
6: if(choice==1)
        read item
         call enqueue(item)
7: if(choice==2)
          item=call dequeue()
8: if(choice==3)
        Display queue elements
        i=front
        for j=0 and j<count
             dispalay q[i]
              i=(i+1)%size
9: if(choice==4)
          goto step 10
10: stop

void enqueue(item)
 1: if count=size
        Write overflow
        Go to step4
2: q[rear] = item
```

```
3:rear=(rear+1)%size
4:count=count+1
5: Exit
```

int dequeue()

```
1: declare item
2: if  count=0
       Display  Queue is empty
       goto step4
3:  else
      item = q[front]
      front=(front+1)%size
      count=count-1
4: return item
```

Program

```
#include<stdio.h>
 int q[5],size=5,front=0,rear=0,count=0;
 void enqueue(int item)
 {
    if(count==size)
          printf("queue overflow");
   else {
          q[rear]=item;
          rear=(rear+1)%size;
          count++;
   }
}
int dequeue()
{
 int item;
 if(count==0)
      printf("Underflow");
 else{
       item=q[front];
       front=(front+1)%size;
       count=count-1;
       return item;
```

```
  }
}



int main()
{
 int item,i,choice;
 printf("create a queue and perfrome the operations");
 printf("\n1.insertion\n2.deletion\n3.traversal\n4.exit\n");
 while(1)
 {
   printf("\nenter the choice");
   scanf("%d",&choice);
   if(choice==1)
   {
        printf("enter the item to be inserted");
        scanf("%d",&item);
        enqueue(item);
   }
    if(choice==2)
   {
       item=dequeue();
       printf("Item Deleted:%d",item);
   }
    if(choice==3)
   {
     if(count==0)
        printf("Queue is empty");
     printf("queue elements are:");
    i=front;
    for(j=0;j<count;j++)
     {
       printf("%d\t",q[i]);
       i=(i+1)%size;
      }
   }
    if(choice==4)
   {
    printf("exit");
    break;
   }
  }
```

```
return 0;
}
```

Output

```
mits@mits-Lenovo-S510:~/Desktop/s1mca$ gcc c_queue.c
 mits@mits-Lenovo-S510:~/Desktop/s1mca$ ./a.out
1.insertion
2.deletion
3.traversal
4.exit
enter the choice1
enter the item to be inserted56
enter the choice1
enter the item to be inserted89
enter the choice1
enter the item to be inserted34
enter the choice3
queue elements are:56   89      34
enter the choice 2
Item Deleted:56
enter the choice 3
queue elements are:89   34
```

Experiment 12 Date: 19.10.2023

Singly Linked List Operations

Aim

To implement the following operations on a singly linked list

i. Creation,

ii. Insert a new node at front

iii. Insert an element after a particular

iv. Deletion from beginning

v. Deletion from the end

vi. Searching

vii. Traversal.

Algorithm:

main()

```
 1: declare a structure node with data members data ,*next and variables *head,*temp,*ptr
 2: declare data,ch,key
 3: display 1.Insertfront 2.insertion after specific 3.Deletefront 4.DeleteEnd 5.Search
    6.traversal 7 Exit
 4: reapeat the step 5 to 8 while(choice>0 && choice<7)
 5: read choice
 6: if(choice=1)
         read data
          call insertfront(data)
6: if(choice=2)
         read data
         read key
         call insertspecific(data,key)
 7: if(choice=3)
           deletebeg()
 8: if(choice=4)
         deleteend()
 9: if(ch=5)
         read data
```

```
          search(data)
10: if(ch=6)
      if(head=NULL)
            display linked list is empty
      else
           temp=head
            while(temp!=NULL)
                 display temp->data
                 temp=temp->next
 11: if(choice==7)
            goto step 10
 12: stop
```

void insertfront(data)

```
1: allocate memory  for temp
2: read data into temp
3: if(head=NULL)
        temp->next=NULL
        head=temp
    else
        temp->next=head
        head=temp
4: stop
```

void insertspecific(data,key)

```
1:declare flag
2: allocate memory  for temp
3: read data into temp
4: if(head!=NULL)
        ptr=head
        while(ptr!=NULL)
           if(ptr->data=key)
                  flag=1
                  temp->next=ptr->next
```

```
            prt->next=temp;
            goto step 5
        else
            ptr=ptr->next
5: if(flag==0)
      Display key is not present insetion not possible
6: stop
```

void deletebeg()

```
1: if(head=NULL)
     Display linked  list is empty
2: ptr=head
3: display deleted element ptr->data
4: head=ptr->next
5: free(ptr)
6:  stop
```

void deleteend()

```
1: if(head=NULL)
      Display linked  list is empty
   else
      ptr=head
      while(ptr->next!=NULL)
          temp=ptr
          ptr=ptr->next
     temp->next=NULL
     display deleted element ptr->data
     free(ptr)
2:stop
```

void search(data)

```
1: declare flag
2:if(head=NULL)
     Display linked list is empty
```

```
3:temp=head
4:while(temp!=NULL)
    If(temp->data==item)
        Display element found
        Flag=1
        Goto step 5
     temp=temp->next

5: if(flag=0)
      Display element is not present
6:stop
```

Program

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
 int data;
 struct node *next;
}*head=NULL,*temp,*ptr;
void insertfront(int data){
 temp=malloc(sizeof(struct node));
 temp->data=data;
 if(head==NULL){
  temp->next=NULL;
  head=temp;
 }
 else{
  temp->next=head;
  head=temp;
 }
}
```

```
void  insertspecific(int data,int key){
 int flag=0;
 temp=malloc(sizeof(struct node));
 temp->data=data;
 if(head!=NULL)
{
  ptr=head;
  while(ptr!=NULL)
  {
   if(ptr->data==key)
   {
     flag=1;
     temp->next=ptr->next;
     ptr->next=temp;
     break;
   }
   else
   ptr=ptr->next;
  }
  if(flag== 0)
    printf("Key is not present");
  }
}
void deletebeg(){
  if(head==NULL)
    printf("linked list is empty");
  ptr=head;
  printf("deleted element:%d",ptr->data);
  head=ptr->next;
  free(ptr);
```

```
}
void deleteend()
{
  if(head==NULL)
    printf("linked list is empty");
  else{
   ptr=head;
   while(ptr->next!=NULL)
   {
    temp=ptr;
    ptr=ptr->next;
   }
  temp->next=NULL;
  printf("deleted element:%d",ptr->data);
  free(ptr);
  }
}
void search(int item)
{
  int flag=0;
  if(head==NULL)
    printf("linked list is empty");
  temp=head;
  while(temp!=NULL)
  {
   if(temp->data==item){
     printf("Element is present in linkedlist");
     flag=1;
     break;
    }
   temp=temp->next;
  }
  if(flag==0)
     printf("Element is not present in linked list");
}
void main()
{
 int data,ch,key;
 printf("Linked List Operation\n1. insertfront\n2.insertion after specific\n3.Deletion from begining\n4.Deletion from end\n5.Search\n6.Traversal\n7.Exit");
 do{
  printf("\nEnter the choice:");
  scanf("%d",&ch);
  if(ch==1){
   printf("Enter the data to be inserted");
   scanf("%d",&data);
   insertfront(data);
  }
  if(ch==2){
   printf("Enter the data to be inserted");
   scanf("%d",&data);
   printf("Enter the key");
   scanf("%d",&key);
   insertspecific(data,key);
  }
  if(ch==3)
    deletebeg();
  if(ch==4)
    deleteend();
  if(ch==5){
    printf("Enter the data to be searched");
   scanf("%d",&data);
   search(data) ;
  }
 if(ch==6){
     if(head==NULL)
         printf("linked list is empty");
      else{
       printf("linked list:");
        temp=head;
          while(temp!=NULL)
          {
          printf("%d->",temp->data);
          temp=temp->next;
          }printf("NULL");
}
 }
 if(ch==7){
   printf("Exit");
   break;
  }
 }while(ch>0 &&ch<7);
}
```

Output

```
mits@mits-Lenovo-S510:~/Desktop/s1mca$ gcc singllyll.c
mits@mits-Lenovo-S510:~/Desktop/s1mca$ gcc ./a.out
Linked List Operation
1. Insertfront
2.insertion after specific
3.Deletion from begining
4.Deletion from end
5.Search
6.Traversal
7.Exit
```

```
Enter the choice:1

Enter the data to be inserted 67

Enter the choice:6

linked list:67->NULL

Enter the choice:2

Enter the data to be inserted 89

Enter the key 67

Enter the choice:6

linked list:67->89->NULL

Enter the choice:2

Enter the data to be inserted 78

Enter the key 3

Key is not present

Enter the choice:3

deleted element:67

Enter the choice:6

linked list:89->NULL

Enter the choice:1

Enter the data to be inserted 45

Enter the choice:6

linked list:45->89->NULL

Enter the choice:4

deleted element:89

Enter the choice:6

linked list:45->NULL

Enter the choice:5

Enter the data to be searched45

Element is present in linkedlist

Enter the choice:5

Enter the data to be searched90
Element is not present in linked list
```

Experiment 13 Date: 20.10.2023

Doubly Linked List Operations

Aim

To implement the following operations on a Doubly linked list.

i. Creation

ii. Count the number of nodes

iii. Insert a node at first position

iv. Insert a node at l

v. Delete a node from the first position

vi. Delete a node from last

vii. Searching

viii. Traversal

Algorithm:

main()

```
 1: declare a structure node with data members data ,*next, *prev and variables
    *head,*temp,*ptr,*ptr1
 2: declare data,ch,key
 3: display 1.Insertfront 2.insertion at end 3.Deletefront 4.DeleteEnd 5.Search  6 count no of
    nodes  7.traversal 8 Exit
 4: reapeat the step 5 to 8 while(choice>0 && choice<8)
 5: read choice
 6: if(choice=1)
          read data
           call insertfront(data)
6: if(choice=2)
          read data
          call insertend(data)
 7: if(choice=3)
            deletebeg()
 8: if(choice=4)
          deleteend()
 9: if(ch=5)
```

```
           read data
           search(data)
10: if(ch=6)
       Declare count=0
        temp=head
        while(temp!=NULL)
              count=count+1
              temp=temp->next
     display count
 11: if(choice==7)
            Traversal()
 13: if(ch==8)
        Goto step14
 14: stop
```

void insertfront(data)

```
1: allocate memory  for temp
2: read data into temp
3: if(head=NULL)
        temp->next=temp->prev=NULL
        head=temp
    else
        temp->prev=NULL
        temp->next=head
        head=temp
4: stop
```

void insertend(data)

```
1: allocate memory  for temp
2: read data into temp
3: if(head=NULL)
        temp->next=temp->prev=NULL
        head=temp
    else
```

```
        ptr=head
        while(ptr->next!=NULL)
              ptr=ptr->next
        ptr->next=temp
        temp->prev=ptr
        temp->next=NULL
  4:stop
```

void deletebeg()

```
1: if(head=NULL)
      Display linked  list is empty
2: ptr=head
3: display deleted element ptr->data
4: head=ptr->next
5: free(ptr)
6:  stop
```

void deleteend()

```
1: if(head=NULL)
        Display linked  list is empty
    else
          ptr=head
          while(ptr->next!=NULL)
                ptr=ptr->next
         display deleted element ptr->data
          ptr->prev->next=NULL
         free(ptr)
2:stop
```

void search(data**)**

```
1: declare flag
2:if(head=NULL)
       Display linked list is empty
3:temp=head
```

```
4:while(temp!=NULL)
     If(temp->data==item)
         Display element found
         Flag=1
         Goto step 5
      temp=temp->next
5: if(flag=0)
       Display element is not present
6:stop
```

void traversal()

```
1:if(head=NULL)
          Display linked list is empty
2:temp=head
3:while(temp!=NULL)
      Display temp->data
      Temp=temp->next
4:stop
```

Program

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
 int data;
 struct node *next;
 struct node *prev;
}*head=NULL,*temp,*ptr,*ptr1;

void insertfront(int data)
{
 temp=malloc(sizeof(struct node));
 temp->data=data;
 if(head==NULL)
 {
  temp->next=temp->prev=NULL;
  head=temp;
 }
 else
 {
  temp->prev=NULL;
  temp->next=head;
  head=temp;
 }
}
void deletebeg()
{
  if(head==NULL)
    printf("linked list is empty");
  ptr=head;
  printf("deleted element:%d",ptr->data);
  head=ptr->next;
  head->prev=NULL;
  free(ptr);
}
void deleteend()
{
  if(head==NULL)
    printf("linked list is empty");

        ptr = head;
        while(ptr->next!=NULL)
        {
            ptr= ptr->next;
        }
  printf("deleted element:%d",ptr->data);
   ptr-> prev->next = NULL;
  free(ptr);
}
void insertend(int data)
{
 temp=malloc(sizeof(struct node));
 temp->data=data;
  if(head==NULL)
 {
  temp->next=temp->prev=NULL;
  head=temp;
 }
 else{
  ptr=head;
  while(ptr->next!=NULL){
  ptr=ptr->next; }

  ptr->next=temp;
  temp->prev=ptr;
  temp->next=NULL;
  printf("node is inserted at last");
}
}
void search(int item)
{
  int flag=0;
  if(head==NULL)
    printf("linked list is empty");
  temp=head;
  while(temp!=NULL)
  {
   if(temp->data==item){
     printf("Element is present in linkedlist");
     flag=1;
     break;
    }
   temp=temp->next;
  }
  if(flag==0)
     printf("Element is not present in linked list");
}
void traversal()
{
       if(head==NULL)
          printf("linked list is empty");
       else{
        printf("linked list:");
        temp=head;
while(temp!=NULL)
{
printf("%d->",temp->data);
temp=temp->next;
}
printf("NULL");
    }
}
void main()
{
 int data,ch,key;
 printf("Linked List Operation\n1. insertfront\n2.insertion at end\n3.Deletion from
begining\n4.Deletion from end\n5.Search\n6.count no nodes\n7.Traversal\n8.Exit");
 do
 {
  printf("\nEnter the choice:");
  scanf("%d",&ch);
  if(ch==1)
  {
   printf("Enter the data to be inserted");
   scanf("%d",&data);
   insertfront(data);
  }
  if(ch==2)
  {
   printf("Enter the data to be inserted");
   scanf("%d",&data);
   insertend(data);
  }
   if(ch==3)
  {
   deletebeg();
  }
   if(ch==4)
  {
   deleteend();
  }
  if(ch==5)
  {
   printf("Enter the key to be searched");
   scanf("%d",&data);
   search(data);
  }
  if(ch==6)
  {
  int count=0;
  temp=head;
  while(temp!=NULL)
  {
     count++;
    temp=temp->next;
  }
  printf("Count:%d",count);
  }
 if(ch==7)
 {
    traversal();
 }
 if(ch==8)
  {
   printf("Exit");
   break;
  }
 }while(ch>0 &&ch<9);
}
```

Output

```
mits@mits-Lenovo-S510:~/Desktop/s1mca$ gcc doublyll.c
 mits@mits-Lenovo-S510:~/Desktop/s1mca$ ./a.out
Linked List Operation
1. insertfront
2.insertion at end
3.Deletion from begining
4.Deletion from end
5.Search
6.count no nodes
7.Traversal
8.Exit
Enter the choice:1
Enter the data to be inserted67

Enter the choice:1
Enter the data to be inserted45

Enter the choice:7
linked list:45->67->NULL
Enter the choice:2
Enter the data to be inserted43
node is inserted at last
Enter the choice:7
linked list:45->67->43->NULL
Enter the choice:3
deleted element:45
Enter the choice:7
linked list:67->43->NULL
Enter the choice:5
Enter the key to be searched43
Element is present in linkedlist
Enter the choice:5
Enter the key to be searched56
Element is not present in linked list
Enter the choice:6
Count:2
Enter the choice:7
linked list:67->43->NULL
Enter the choice:8
Exit
```

Experiment 14 Date: 27.10.2023

Stack Using Linked List Opertions

Aim

To implement a menu driven program to perform following stack operations linked list

i. push
ii. pop
iii.Traversal

Algorithm

main()

```
1. Declare data,ch,key
2. Read  choice
3. Push(data)
4. Pop()
5. Traversal
     if(top==NULL)
          printf "Stack is empty"
        else
         printf "Stack elements:"
   temp=top
   while(temp!=NULL)
   printf "%d->",temp->data
   temp=temp->next;
   printf "NULL”
6. Exit
7. Stop
```

void push(int data)

```
1.  temp=malloc(sizeof(struct node));
2. temp->data=data;
3.  if(top==NULL)
4.   temp->next=NULL
     top=temp
5. else
     temp->next=top
     top=temp;
```

void pop()

```
1. if(top==NULL)
2. Print "stack is empty"
3. temp=top
     printf("deleted element:%d",ptr->data)
     top=temp->next
4. free(temp)
```

Program

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
 int data;
 struct node *next;
}*top=NULL,*temp,*ptr;

void push(int data)
{
 temp=malloc(sizeof(struct node));
 temp->data=data;
 if(top==NULL)
 {
  temp->next=NULL;
  top=temp;
 }
 else
 {
  temp->next=top;
  top=temp;
 }
}

void pop()
{
  if(top==NULL)
    printf("linked list is empty");
  ptr=top;
  printf("deleted element:%d",ptr->data);
  top=ptr->next;
  free(ptr);
}

void main()
{
 int data,ch,key;
 printf("1.Push\n2.Pop \n3.Traversal\n4.Exit");
 do
 {
  printf("\nEnter the choice:");
  scanf("%d",&ch);
  if(ch==1)
  {
   printf("Enter the data to be inserted");
   scanf("%d",&data);
   push(data);
  }
  if(ch==2)
    pop();
 if(ch==3)
 {   if(top==NULL)
        printf("Stack is empty");
      else{
       printf("Stack elements:");
temp=top;
while(temp!=NULL)
{


printf("%d->",temp->data);
temp=temp->next;
}
printf("NULL");
      }
 }
 if(ch==4)
  {
   printf("Exit");
   break;
  }
 }while(ch>0 &&ch<5);
}
```

Output

```
mits@mits-Lenovo-S510:~/Desktop/s1mca$ gcc stackll.c
mits@mits-Lenovo-S510:~/Desktop/s1mca$ gcc ./a.out
1.Push
2.Pop
3.Traversal
4.Exit
Enter the choice:1
Enter the data to be inserted54
Enter the choice:1
Enter the data to be inserted97
Enter the choice:3
Stack elements:97->54->NULL
Enter the choice:2
deleted element:97
Enter the choice:3
Stack elements:54->NULL
Enter the choice:4
Exit
```

Experiment 14 Date: 27.10.2023

Queue Using Linked List Operations

Aim:

To implement a menu driven program to perform following queue operations using linked list

1. enqueue
2. dequeue
3. Traversal

Algorithm

main()

```
1. Declare data,ch,key
2. Set front=rear=NULL
3. Read choice
4. Read data to be inserted
5. Enqueue(data)
6. Dequeue()
7. Traversal
8. if(rear==NULL)
          printf "Queue is empty"
       else
        printf "Queue elements:"
         temp=front
   while(temp!=NULL)
   printf("%d->",temp->data)
   temp=temp->next
   printf "NULL"
9. stop
```

void enqueue(int data)

```
1. temp=malloc(sizeof(struct node));
2. temp->data=data;
3. temp->next=NULL
4. if(rear==NULL)
    front=rear=temp
   else
    rear->next=temp
    rear=temp
5. if(front==NULL)
```

```
          printf "Queue is empty"
        temp=front
        front=front->next
6.      if(front==NULL)
         rear=NULL
        printf "deleted element:%d",temp->data
7.  free(temp)
8.  exit
```

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
 int data;
 struct node *next;
}*rear,*front,*temp;

void enqueue(int data)
{
 temp=malloc(sizeof(struct node));
 temp->data=data;
 temp->next=NULL;
 if(rear==NULL)
 {
  front=rear=temp;
 }
 else
 {
  rear->next=temp;
  rear=temp;
 }
}

void dequeue()
{
  if(front==NULL)
    printf("Queue is empty");
  temp=front;
  front=front->next;
  if(front==NULL)
  {
   rear=NULL;
```

```
      }
  printf("deleted element:%d",temp->data);
  free(temp);
}
void main()
{
 int data,ch,key;
 front=rear=NULL;
 printf("1.enqueue\n2.Dequeue\n3.Traversal\n4.Exit");
 do
 {
  printf("\nEnter the choice:");
  scanf("%d",&ch);
  if(ch==1)
  {
   printf("Enter the data to be inserted");
   scanf("%d",&data);
   enqueue(data);
  }
  if(ch==2)
    dequeue();

  if(ch==3)
  {    if(rear==NULL)
         printf("Queue is empty");
      else{
       printf("Queue elements:");
        temp=front;
while(temp!=NULL)
{

printf("%d->",temp->data);
temp=temp->next;
}
printf("NULL");
      }
  }
  if(ch==4)
   {
    printf("Exit");
    break;
   }
  }while(ch>0 &&ch<7);
}
```

Output

```
mits@mits-Lenovo-S510:~/Desktop/s1mca$ gcc queuell.c
mits@mits-Lenovo-S510:~/Desktop/s1mca$ gcc ./a.out
1.enqueue

2.Dequeue

3.Traversal

4.Exit

Enter the choice:1

Enter the data to be inserted45

Enter the choice:1

Enter the data to be inserted90

Enter the choice:3

Queue elements:45->90->NULL

Enter the choice:2

deleted element:45

Enter the choice:3

Queue elements:90->NULL

Enter the choice:4

Exit
```

Experiment 16 Date: 2.11.2023

BST Operations

Aim:

Menu Driven program to implement Binary Search Tree Operations- Insertion of node, Deletion of a node,inorder traversal, Pre-order traversal and post-order traversal.

Algorithm

main()

```
1. declare data,choice
2. print " menu"
3. read choice
   root = insert(root, data)
   root = delete(root, data)
   inorder(root)
  preorder(root)
  postorder(root)
4. exit
```

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

```
1. if (root == NULL)
        return createNode(data);
2.    if (data < root->data)
        root->left = insert(root->left, data);
      else if (data > root->data)
        root->right = insert(root->right, data)
3.    return root;
```

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

```
1. if (root == NULL)
        return root;
2.  else if (data < root->data)
        root->left = delete(root->left, data)
3. else if (data > root->data)
        root->right = delete(root->right, data
4.   else
          if (root->left == NULL && root->right == NULL)
```

```
                    free(root)
                    root = NULL
                  else if (root->left == NULL)
                    struct node* temp = root
                    root = root->right
                    free(temp);
                else if (root->right == NULL)
                    struct node* temp = root
                    root = root->left
                    free(temp)
                else
                    struct node* temp = findMin(root->right)
                    root->data = temp->data
                    root->right = delete(root->right, temp->data)
   5.     return root
```

void inorder(struct node* root**)**

```
  1. if (root != NULL)
     inorder(root->left);
     printf "%d ", root->data
     inorder(root->right);
```

void preorder(struct node* root**)**

```
  1.   if (root != NULL)
          printf "%d ", root->data
          preorder(root->left)
          preorder(root->right)
```

void postorder(struct node* root**)**

```
 1.  if (root != NULL)
       postorder(root->left);
       postorder(root->right)
       printf ("%d ", root->data
```

Program

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

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

struct node* createNode(int data) {
   struct node* newNode = (struct node*)malloc(sizeof(struct node));
   newNode->data = data;
   newNode->left = NULL;
   newNode->right = NULL;
   return newNode;
}

struct node* insert(struct node* root, int data) {
   if (root == NULL) {
      return createNode(data);
   }

   if (data < root->data) {
      root->left = insert(root->left, data);
   } else if (data > root->data) {
      root->right = insert(root->right, data);
   }

   return root;
}

struct node* findMin(struct node* root) {
   while (root->left != NULL) {
      root = root->left;
   }
   return root;
}

struct node* delete(struct node* root, int data) {
   if (root == NULL) {
      return root;
   } else if (data < root->data) {
      root->left = delete(root->left, data);
   } else if (data > root->data) {
      root->right = delete(root->right, data);
   } else {
      if (root->left == NULL && root->right == NULL) {
            free(root);
            root = NULL;
        } else if (root->left == NULL) {
            struct node* temp = root;
            root = root->right;
            free(temp);
        } else if (root->right == NULL) {
            struct node* temp = root;
            root = root->left;
            free(temp);
        } else {
            struct node* temp = findMin(root->right);
            root->data = temp->data;
            root->right = delete(root->right, temp->data);
        }
    }
    return root;
}

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

}

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

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

```
int main() {
   struct node* root = NULL;
   int choice, data;

   while (1) {
      printf("\nMenu:\n");
      printf("1. Insert a node\n");
      printf("2. Delete a node\n");
      printf("3. In-order traversal\n");
      printf("4. Pre-order traversal\n");
      printf("5. Post-order traversal\n");
      printf("6. Exit\n");

      printf("Enter your choice: ");
      scanf("%d", &choice);
      switch (choice) {
         case 1:
            printf("Enter the value to be inserted: ");
            scanf("%d", &data);
            root = insert(root, data);
            break;
         case 2:
            printf("Enter the value to be deleted: ");
            scanf("%d", &data);
            root = delete(root, data);
            break;
         case 3:
            printf("In-order Traversal: ");
            inorder(root);
            printf("\n");
            break;
         case 4:
            printf("Pre-order Traversal: ");
            preorder(root);
            printf("\n");
            break;
         case 5:
            printf("Post-order Traversal: ");
            postorder(root);
            printf("\n");
            break;
         case 6:
            exit(0);
         default:
                printf("Invalid choice!\n");
        }
    }

    return 0;
}
```

Output

```
mits@mits-Lenovo-S510:~/Desktop/s1mca$ gcc bst.c
mits@mits-Lenovo-S510:~/Desktop/s1mca$ gcc ./a.out
Menu:
1. Insert a node
2. Delete a node
3. In-order traversal
4. Pre-order traversal
5. Post-order traversal
6. Exit
Enter your choice: 1
Enter the value to be inserted: 30
Enter your choice: 1
Enter the value to be inserted: 20
Enter your choice: 1
Enter the value to be inserted: 60
Enter your choice: 1
Enter the value to be inserted: 70
Enter your choice: 1
Enter the value to be inserted: 40
Enter your choice: 1
Enter the value to be inserted: 10
Enter your choice: 1
Enter the value to be inserted: 25
Enter your choice: 3
In-order Traversal: 10 20 25 30 40 60 70
Enter your choice: 4
Pre-order Traversal: 30 20 10 25 60 40 70
Enter your choice: 5
Post-order Traversal: 10 25 20 40 70 60 30
Enter your choice: 2
Enter the value to be deleted: 30
Enter your choice: 3
In-order Traversal: 10 20 25 40 60 70
Enter your choice: 6
```

Experiment 17 Date: 9.11.2023

Bit String

Aim:

Program to implement set operations using bit string

Algorithm

main()

```
1. declare data,ch,key
2. read choice
3. seta()
4. setb()
5. union()
6. intersection()
7. difference()
8. equals()
9. exit
```

void seta()

```
1. declare s1,d1
2. read “Enter the size of first set\n”
3. read elements
4. read array
5. a[d1]=1
```

void setb()

```
1. declare s2,d2
2. read “ Enter the size of first set\n”
3. Read element
4. Read array
5. a[d2]=1
```

void display()

```
1.  printf "Bitstring of A:\n"
2. Read a[i]
3. Printf "Bitstring of B: \n"
4. Read b[i]
```

void Union()

```
1. for(i=1;i<11;i++)
2. if(a[i]==1 || b[i]==1)
3.     u[i]=1
   else
       u[i]=0
```

void difference()

```
1. for(i=1;i<11;i++)
        if(b[i]==1)
       b[i]=0
   else
       b[i]=1
2. for(i=1;i<11;i++)
   if(a[i]==1 && b[i]==1)
       u[i]=1
   else
       u[i]=0
```

void intersation()

```
1. if(a[i]==1 && b[i]==1)
2.     u[i]=1
   else
       u[i]=0
```

Program

```
#include<stdio.h>
#include<stdlib.h>
int a[11],b[11],u[11],i;
int us[11]={1,2,3,4,5,6,7,8,9,10,11};

void seta()
{
int s1,d1;
printf("Enter the size of first set\n");
scanf("%d",&s1);
printf("Enter elements\n");
for(i=0;i<s1;i++)
{
scanf("%d",&d1);
a[d1]=1;
}
}

void setb()
{
int s2,d2;
printf("Enter the size of second set\n");
scanf("%d",&s2);
printf("Enter elements\n");
```

```
for(i=0;i<s2;i++)
{
scanf("%d",&d2);
b[d2]=1;
}

}
void display()

{
 printf("Bitstring of A:\n");
for(i=1;i<11;i++)
{
     printf("%d\t",a[i]);
}
printf(" \n");
printf("Bitstring of B: \n");
for(i=1;i<11;i++)
{
       printf("%d \t",b[i]);
}
printf(" \n");

}


void Union()
{
for(i=1;i<11;i++)
{
if(a[i]==1 || b[i]==1)
    u[i]=1;
else
    u[i]=0;
}
display();
printf("Union: \n");
for(i=1;i<11;i++)
   printf("%d \t",u[i]);
}


void intersection()
{
for(i=1;i<11;i++)
{
if(a[i]==1 && b[i]==1)
    u[i]=1;
else
   u[i]=0;
}
display();
printf("Intersection: \n");
for(i=1;i<11;i++)
  printf("%d \t",u[i]);
}

void difference()
{
      for(i=1;i<11;i++)
{
if(b[i]==1)
   b[i]=0;
else
   b[i]=1;
}
for(i=1;i<11;i++)
{
if(a[i]==1 && b[i]==1)
   u[i]=1;
else
   u[i]=0;
}
display()
printf("differnce: \n");
for(i=1;i<11;i++)
  printf("%d \t",u[i]);
}


void equal()
{
      int f=0;
for(i=1;i<11;i++)
{
if(a[i]!=b[i]){
 f=1;
 break;}
}
if(f==1)
  printf("sets are not equal");
else
 printf("sets are  equal");
}
```

```
void main()
{
 int data,ch,key;
 printf("1.Set a and b \n2.Union of to sets\n3.Intersection of to sets\n4.Difference of to
sets\n5.Equal\n6.Exit");
 do{
  printf("\nEnter the choice:");
  scanf("%d",&ch);
 if(ch==1)
    {
     seta();
     setb();

     display();
    }
 if(ch==2)
     Union();
 if(ch==3)
     intersection();
 if(ch==4)
     difference();
 if(ch==5)
     equal();
  if(ch==6)
    {
     printf("Exit");
     break;
    }

  }while(ch<6);
}
```

Output

```
mits@mits-Lenovo-S510:~/Desktop/s1mca$ gcc bitstring.c
mits@mits-Lenovo-S510:~/Desktop/s1mca$ gcc ./a.out
1.Set a and b

2.Union of to sets

3.Intersection of to sets

4.Difference of to sets

5.Equal

6.Exit

Enter the choice:1
Enter the size of first set
4
Enter elements
1 3 4 6
Enter the size of second set
7
Enter elements
1 2 3 4 5 6 7
Bitstring of A:
1    0    1    1    0    1    0    0    0    0
Bitstring of B:
1    1    1    1    1    1    1    0    0    0
Enter the choice:2
Bitstring of A:
1    0    1    1    0    1    0    0    0    0
Bitstring of B:
1    1    1    1    1    1    1    0    0    0
Union:
1    1    1    1    1    1    1    0    0    0
Enter the choice:3
Bitstring of A:
1    0    1    1    0    1    0    0    0    0
Bitstring of B:
1    1    1    1    1    1    1    0    0    0
Intersection:
1    0    1    1    0    1    0    0    0    0
Enter the choice:4
Bitstring of A:
1    0    1    1    0    1    0    0    0    0
Bitstring of B:
```

```
0       0       0       0       0       0       0       1       1       1
differnce:
0       0       0       0       0       0       0       0       0       0
Enter the choice:5
sets are not equal
Enter the choice:6
Exit
```