Experiment 12 **Date: 19.10.2023**

Singly Linked List Operations

Aim:

12. To implement the following operations on a singly linked list
    a. Creation
    b. Insert a new node at front
    c. Insert an element after a particular
    d. Deletion from beginning
    e. Deletion from the end
    f. Searching
    g. Traversal.

Algorithm:

Program main()

1. Start
2. struct node{
   int data;
   struct node *next;
   }*head, *ptr, *temp;
3. Display choices.
4. Read option ch.
   a. if ch==1 call ins_beg().
   b. if ch==2 call ins_spec().
   c. if ch==3 call del_beg()
   d. if ch==4 call del_end()
   e. if ch==5 call search()
   f. if ch==6 call display()
5. Repeat step 3 while ch>0&&ch<7.
6. Stop.

void ins_beg()

1. Start
2. ptr = malloc(sizeof(struct node))
3. Read ptr->data
   if head==NULL
   ptr->next=NULL;
   head=ptr
   else
   ptr->next=head;

head=ptr

4. Exit

void ins_spec()

1. Start
2. ptr = malloc(sizeof(struct node))
3. Read ptr->data
4. set temp=head
5. for(int i=1;i<p;i++){
   temp=temp->next;
   if(temp==NULL){
   printf("Invalid Position");
   break;
   }
   }
   ptr->next=temp->next;
   temp->next=ptr;
6. Exit

void del_beg()

1. Start
2. if head==NULL print List Empty
3. else
   print head->data is deleted
   if head->next==NULL
   free(head);
   head=NULL;
   else
   ptr=head;
   head=ptr->next;
   free(ptr);
4. Exit.

void del_end()

1. Start
2. if head==NULL print List Empty
3. else

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

4. Exit.

void display()

1. Start
2. if head==NULL print List Empty
3. else

```
printf("Linked List : ");
while(ptr!=NULL){
        printf("%d\t",ptr->data);
        ptr=ptr->next;
}
```

4. Exit.

void search()

1. Start
2. Declare x,i=1,f=0
3. if head==NULL print List Empty
4. else
   read x
   for(ptr=head; ptr!=NULL; ptr=ptr->next){
   if(ptr->data==x){
   print element found at node i
   set f=1
   }
   i++
   }
   if f ==0 print Element not found
5. Exit.

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

void ins_beg(){
   ptr = malloc(sizeof(struct node));
   printf("Enter the item : ");
   scanf("%d",&ptr->data);
   if(head==NULL){
      ptr->next=NULL;
      head=ptr;
   }
   else{
      ptr->next=head;
      head=ptr;
   }
}
```

```
void ins_spec(){
   int p;
   ptr =  malloc(sizeof(struct node));
   printf("Enter the item and it's position : ");
   scanf("%d %d",&ptr->data,&p);
   temp=head;
   for(int i=1;i<p;i++){
      temp=temp->next;
      if(temp==NULL){
         printf("Invalid Position");
         break;
      }
   }
   ptr->next=temp->next;
   temp->next=ptr;
}

void del_beg(){
   if(head==NULL){
      printf("List Empty");
   }
   else{
      printf("%d is deleted",head->data);
      if(head->next==NULL){
         free(head);
         head=NULL;
      }
      else{
         ptr=head;
         head=ptr->next;
         free(ptr);
      }
   }
}

void del_end(){
   if(head==NULL){
      printf("List Empty");
   }
   else{
      if(head->next==NULL){
         printf("%d is deleted",head->data);
```

```
            free(head);
            head=NULL;
        }

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

void display(){
    if(head==NULL){
        printf("List Empty");
    }
    else{
        ptr=head;
        printf("Linked List : ");
        while(ptr!=NULL){
            printf("%d\t",ptr->data);
            ptr=ptr->next;
        }
    }
}

void search(){
    int x,i=1,f=0;
    if(head==NULL){
        printf("List Empty");
    }
    else{
        printf("Enter the item : ");
        scanf("%d",&x);
        for(ptr=head; ptr!=NULL; ptr=ptr->next){
            if(ptr->data==x){
                printf("Element found at node %d",i);
```

```
            f=1;
          }
          i++;
       }
       if(f==0){
          printf("Element not found");
       }
    }
}

void main(){
   int ch;
   do{
      printf("\n1. Insert at front\n2. Insert at Specific Position\n3. Delete at front\n4. Delete at rear\n5. Search\n6. Display\n7. Exit\nEnter your choice(1-7) : ");
      scanf("%d",&ch);
      switch(ch){
         case 1: ins_beg();
                 break;
         case 2: ins_spec();
                 break;
         case 3: del_beg();
                 break;
         case 4: del_end();
                 break;
         case 5: search();
                 break;
         case 6: display();
                 break;
      }
   }while(ch>0&&ch<7);
}
```

Output

```
mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc PGM12.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out PGM12.c

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 1
Enter the item : 10

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 1
Enter the item : 20

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 1
Enter the item : 30
```

```
1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 6
Linked List : 30     20      10

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 2
Enter the item and it's position : 25 1

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 6
Linked List : 30     25      20      10

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 2
Enter the item and it's position : 15 3
```

```
1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 6
Linked List : 30     25      20      15      10

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 5
Enter the item : 20
Element found at node 3

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 3
30 is deleted

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 6
```

```
Linked List : 25     20      15      10

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 4
10 is deleted

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 6
Linked List : 25     20      15

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 5
Enter the item : 30
Element not found
```

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit

Enter your choice(1-7) : 7

mits@mits:~/Desktop/S1MCA/ADS_lab$

Experiment 13 **Date: 20.10.2023**

Doubly Linked List Operations

Aim:

13. To implement the following operations on a singly linked list
    a. Creation
    b. Count the number of nodes
    c. Insert a new node at front
    d. Insert an element at end
    e. Deletion from beginning
    f. Deletion from the end
    g. Searching
    h. Traversal.

Algorithm:

main()

1. Start
2. struct node{
   int data;
   struct node *l, *r;
   }*head, *ptr, *temp;
   c=0
3. Display choices.
4. Read option ch.
   a. if ch==1 call ins_beg().
   b. if ch==2 call ins_end().
   c. if ch==3 call del_beg()
   d. if ch==4 call del_end()
   e. if ch==5 call search()
   f. if ch==6 print c
   g. if ch==7 call display()
5. Repeat step 3 while ch>0&&ch<8.
6. Stop.

void ins_beg()

1. Start
2. ptr = malloc(sizeof(struct node))
3. Read ptr->data
4. c++

```
if head==NULL
        ptr->r=ptr->l=NULL;
        head=ptr
else
        ptr->l=NULL;
        ptr->r=head;
        head=ptr;
```

5. Exit

void ins_end()

1. Start
2. ptr = malloc(sizeof(struct node))
3. Read ptr->data
4. c++

```
if head==NULL
        ptr->r=ptr->l=NULL;
        head=ptr
else
        temp=head;
        while(temp->r!=NULL){
                temp=temp->r;
        }
        temp->r=ptr;
        ptr->l=temp;
        ptr->r=NULL;
```

5. Exit

void del_beg()

1. Start
2. if head==NULL print List Empty
3. else

```
c--
print head->data is deleted
if(head->r==NULL){
        free(head);
        head=NULL;
```

```
        }
        else{
                ptr=head;
                head=head->r;
                head->l=NULL;
                free(ptr);
        }
```

4. Exit.

void del_end()

1. Start
2. if head==NULL print List Empty
3. else

```
        c--;
        if(head->r==NULL){
                printf("%d is deleted",head->data);
                free(head);
                head=NULL;
        }
        else{
                ptr=head;
                while(ptr->r!=NULL){
                        ptr=ptr->r;
                }
                printf("%d is deleted",ptr->data);
                ptr->l->r=NULL;
                free(ptr);
        }
```

4. Exit.

void display()

1. Start
2. if head==NULL print List Empty
3. else

```
        printf("Linked List : ");
        while(ptr!=NULL){
                printf("%d\t",ptr->r);
                ptr=ptr->r;
        }
```

4. Exit.

void search()

1. Start
2. Declare x,i=1,f=0
3. if head==NULL print List Empty
4. else
   read x
   for(ptr=head; ptr!=NULL; ptr=ptr->r){
   if(ptr->r==x){
   print element found at node i
   set f=1
   }
   i++
   }
   if f ==0 print Element not found
5. Exit.

Program

```
#include<stdio.h>
#include<stdlib.h>
int c=0;
struct node{
   int data;
   struct node *l, *r;
}*head, *ptr, *temp;

void ins_beg(){
   ptr = malloc(sizeof(struct node));
   printf("Enter the item : ");
   scanf("%d",&ptr->data);
   c++;
   if(head==NULL){
      ptr->r=ptr->l=NULL;
      head=ptr;
   }
   else{
      ptr->l=NULL;
      ptr->r=head;
      head=ptr;
   }
}
```

```
void ins_end(){
   ptr =  malloc(sizeof(struct node));
   printf("Enter the item : ");
   scanf("%d",&ptr->data);
   c++;
   if(head==NULL){
      ptr->r=ptr->l=NULL;
      head=ptr;
   }
   else{
      temp=head;
      while(temp->r!=NULL){
         temp=temp->r;
      }
      temp->r=ptr;
      ptr->l=temp;
      ptr->r=NULL;
   }
}

void del_beg(){
   if(head==NULL){
      printf("List Empty");
   }
   else{
      c--;
      printf("%d is deleted",head->data);
      if(head->r==NULL){
         free(head);
         head=NULL;
      }
      else{
         ptr=head;
         head=head->r;
         head->l=NULL;
         free(ptr);
      }
   }
}

void del_end(){
   if(head==NULL){
```

```
        printf("List Empty");
    }
    else{
        c--;
        if(head->r==NULL){
            printf("%d is deleted",head->data);
            free(head);
            head=NULL;
        }
        else{
            ptr=head;
            while(ptr->r!=NULL){
                ptr=ptr->r;
            }
            printf("%d is deleted",ptr->data);
            ptr->l->r=NULL;
            free(ptr);
        }
    }
}

void display(){
    ptr=head;
    if(ptr==NULL){
        printf("List Empty");
    }
    else{
        printf("Doubly Linked List : ");
        while(ptr!=NULL){
            printf("%d\t",ptr->data);
            ptr=ptr->r;
        }
    }
}

void search(){
    int x,i=1,f=0;
    if(head==NULL){
        printf("List Empty");
    }
    else{
        printf("Enter the item : ");
```

```
        scanf("%d",&x);
        for(ptr=head; ptr!=NULL; ptr=ptr->r){
            if(ptr->data==x){
                printf("Element found at node %d",i);
                f=1;
            }
            i++;
        }
        if(f==0){
            printf("Element not found");
        }
    }
}

void main(){
    int ch;
    do{
        printf("\n1. Insert at front\n2. Insert at rear\n3. Delete at front\n4. Delete at
rear\n5. Display\n6. Search\n7. Count\n8. Exit\nEnter your choice(1-8) : ");
        scanf("%d",&ch);
        switch(ch){
            case 1: ins_beg();
                    break;
            case 2: ins_end();
                    break;
            case 3: del_beg();
                    break;
            case 4: del_end();
                    break;
            case 5: display();
                    break;
            case 6: search();
                    break;
            case 7: printf("Number of nodes : %d",c);
                    break;
        }
    }while(ch>0&&ch<8);
}
```

Output

```
mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc PGM13.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out PGM13.c

1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 1
Enter the item : 10

1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 1
Enter the item : 20

1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 1
Enter the item : 30
```

1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit

Enter your choice(1-8) : 5
Doubly Linked List : 30 20 10

1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit

Enter your choice(1-8) : 2
Enter the item : 40

1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit

Enter your choice(1-8) : 2
Enter the item : 50

```
1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 5
Doubly Linked List : 30   20      10      40      50

1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 6
Enter the item : 10

Element found at node 3
1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 7
Number of nodes : 5
```

```
1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 3
30 is deleted

1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 5
Doubly Linked List : 20   10      40      50

1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 4
50 is deleted
```

```
1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 5
Doubly Linked List : 20   10      40

1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 8

mits@mits:~/Desktop/S1MCA/ADS_lab$
```

Experiment 14 **Date: 27.10.2023**

Linked Stack Operations

Aim:

14. To implement a menu driven program to perform following stack operations using linked list
    a. Push
    b. Pop
    c. Traversal

Algorithm:

main()

1. Start
2. struct node{
   int data;
   struct node *next;
   }*top, *ptr;
3. Display choices.
4. Read option ch.
   a. if ch==1 call push().
   b. if ch==2 call pop().
   c. if ch==3 call display()
5. Repeat step 3 while ch>0&&ch<4.
6. Stop.

void push()

1. Start
2. ptr = malloc(sizeof(struct node))
3. Read ptr->data
4. ptr->next=top; top=ptr;
5. Exit

void pop()

1. Start
2. if head==NULL print Stack Underflow
3. else
   ptr=top
   print ptr->data is deleted
   top=top->next;
   free(ptr);
4. Exit.

void display()

1. Start
2. if head==NULL print Stack Empty
3. else
   while(ptr!=NULL){
   print ptr->data
   ptr=ptr->next
   }
4. Exit.

Program

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

void push(){
   ptr = malloc(sizeof(struct node));
   printf("Enter the item : ");
   scanf("%d",&ptr->data);
   ptr->next=top;
   top=ptr;
}

void pop(){
   if(top==NULL){
      printf("Stack Underflow");
   }
   else{
```

```
        ptr=top;
        printf("%d is deleted",ptr->data);
        top=top->next;
        free(ptr);
    }
}

void display(){
    ptr=top;
    if(ptr==NULL){
        printf("Stack Empty");
    }
    else{
        printf("Stack : ");
        while(ptr!=NULL){
            printf("%d\t",ptr->data);
            ptr=ptr->next;
        }
    }
}

void main(){
    int ch;
    do{
        printf("\n1. Push\n2. Pop\n3. Display\n4. Exit\nEnter your choice(1-4) : ");
        scanf("%d",&ch);
        switch(ch){
            case 1: push();
                    break;
            case 2: pop();
                    break;
            case 3: display();
                    break;
        }
    }while(ch>0&&ch<4);
}
```

Output

```
mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc PGM14.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out PGM14.c

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the item : 10

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the item : 20

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the item : 30

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 3
Stack : 30     20      10

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 2
30 is deleted
```

```
1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 2
20 is deleted

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 2
10 is deleted

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 3
Stack Empty

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 4

mits@mits:~/Desktop/S1MCA/ADS_lab$
```

Experiment 15 **Date: 27.10.2023**

Linked Queue Operations

Aim:

15. To implement a menu driven program to perform following queue operations using linked list
    a. Enqueue
    b. Dequeue
    c. Traversal

Algorithm:

main()

1. Start
2. struct node{
   int data;
   struct node *next;
   }*top, *ptr, *f, *r;
3. Display choices.
4. Read option ch.
   a. if ch==1 call enqueue().
   b. if ch==2 call dequeue().
   c. if ch==3 call display()
5. Repeat step 3 while ch>0&&ch<4.
6. Stop.

void enqueue()

1. Start
2. ptr = malloc(sizeof(struct node))
3. Read ptr->data
4. if f==NULL
   f=r=ptr;
   else
   r->next=ptr;
   r=ptr;
5. Exit

void dequeue()

1. Start
2. if f==NULL print Queue is empty
3. else
   ptr=f
   print ptr->data is deleted
   f=ptr->next;
   free(ptr);
4. Exit.

void display()

1. Start
2. if head==NULL print Queue is empty
3. else
   while(ptr!=NULL){
   print ptr->data
   ptr=ptr->next
   }
4. Exit.

Program

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

void enqueue(){
   ptr = malloc(sizeof(struct node));
   printf("Enter the item : ");
   scanf("%d",&ptr->data);
   if(f==NULL){
      f=r=ptr;
   }
   else{
      r->next=ptr;
      r=ptr;
   }
}
```

```
void dequeue(){
   if(f==NULL){
      printf("Queue is empty");
   }
   else{
      ptr=f;
      printf("%d is deleted",ptr->data);
      f=ptr->next;
      free(ptr);
   }
}

void display(){
   if(f==NULL){
      printf("Queue is empty");
   }
   else{
      ptr=f;
      printf("Queue : ");
      while(ptr!=NULL){
         printf("%d\t",ptr->data);
         ptr=ptr->next;
      }
   }
}

void main(){
   int ch;
   do{
      printf("\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\nEnter your
choice(1-4) : ");
      scanf("%d",&ch);
      switch(ch){
         case 1: enqueue();
               break;
         case 2: dequeue();
               break;
         case 3: display();
               break;
      }
   }while(ch>0&&ch<4);
}
```

Output

```
mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc PGM15.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out PGM15.c

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the item : 10

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the item : 20

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the item : 30

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice(1-4) : 3
Queue : 10   20      30

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice(1-4) : 2
10 is deleted
```

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice(1-4) : 2
20 is deleted

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice(1-4) : 2
30 is deleted

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice(1-4) : 3
Queue is empty

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice(1-4) : 4

mits@mits:~/Desktop/S1MCA/ADS_lab$
```

Experiment 16 **Date: 02.11.2023**

Binary Search Tree Operations

Aim:

16. Menu Driven program to implement Binary Search Tree (BST) and to perform following operations
    a. Insertion of a node.
    b. Deletion of a node.
    c. In-order traversal.
    d. Pre-order traversal.
    e. Post-order traversal.

Algorithm:

main()

1. Start
2. struct node{
   int data;
   struct node *l,*r;
   }*root, *ptr, *succ, *succparent;
3. Declare ch and x
4. Display choices.
5. Read option ch.
   a. if ch==1 read x and call root=insert(root,x).
   b. if ch==2 read x and call root=del(root,x).
   c. if ch==3 call inorder(root)
   d. if ch==4 call preorder(root)
   e. if ch==5 call postorder(root)
6. Repeat step 3 while ch>0&&ch<6.
7. Stop.

struct node* create(int x)

1. Start
2. ptr=malloc(sizeof(struct node));
3. ptr->data=x;
4. ptr->l=ptr->r=NULL;
5. return ptr;
6. Exit

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

1. Start
2. if root==NULL return create(x)
3. if x>root->data
   root->r=insert(root->r,x);
   else
   root->l=insert(root->l,x);
4. return root;
5. Exit.

struct node* del(struct node* root, int x)

1. Start
2. if root==NULL return root
3. if x>root->data
   root->r=del(root->r,x)
   return root
   else if x<root->data
   root->l=del(root->l,x)
   return root
4. if root->l==NULL
   ptr=root->r
   free(root)
   return ptr
   else if root->r==NULL
   ptr=root->l
   free(root)
   return ptr
5. succparent=root
   succ=root->r;
   while(succ->l!=NULL){
   succparent=succ;
   succ=root->l;
   }
6. if succparent!=root
   succparent->l=succ->r
   else
   succparent->r=succ->r
7. root->data=succ->data
8. free(succ);
9. return root;
10. Exit.

void inorder(struct node* root)

1. Start
2. if(root!=NULL)
   inorder(root->l)
   print root->data
   inorder(root->r)
3. Exit.

void preorder(struct node* root)

1. Start
2. if(root!=NULL)
   print root->data
   inorder(root->l)
   inorder(root->r)
3. Exit.

void postorder(struct node* root)

1. Start
2. if(root!=NULL)
   inorder(root->l)
   inorder(root->r)
   print root->data
3. Exit.

Program

```
#include<stdio.h>
#include <stdlib.h>
struct node{
   int data;
   struct node *l,*r;
}*root, *ptr, *succ, *succparent;

struct node* create(int x){
   ptr=malloc(sizeof(struct node));
   ptr->data=x;
   ptr->l=ptr->r=NULL;
   return ptr;
}
```

```
struct node* insert(struct node* root, int x){
   if(root==NULL){
      return create(x);
   }
   if(x>root->data){
      root->r=insert(root->r,x);
   }
   else{
      root->l=insert(root->l,x);
   }

   return root;
}

struct node* del(struct node* root, int x){
   if(root==NULL){
      return root;
   }

   if(x>root->data){
      root->r=del(root->r,x);
      return root;
   }
   else if(x<root->data){
      root->l=del(root->l,x);
      return root;
   }

   if(root->l==NULL){
      ptr=root->r;
      free(root);
      return ptr;
   }
   else if(root->r==NULL){
      ptr=root->l;
      free(root);
      return ptr;
   }
```

```
    succparent=root;
    succ=root->r;
    while(succ->l!=NULL){
        succparent=succ;
        succ=root->l;
    }

    if(succparent!=root){
        succparent->l=succ->r;
    }
    else{
        succparent->r=succ->r;
    }

    root->data=succ->data;

    free(succ);
    return root;
}

void inorder(struct node* root){
    if(root!=NULL){
        inorder(root->l);
        printf("%d\t",root->data);
        inorder(root->r);
    }
}

void preorder(struct node* root){
    if(root!=NULL){
        printf("%d\t",root->data);
        inorder(root->l);
        inorder(root->r);
    }
}
```

```
void postorder(struct node* root){
   if(root!=NULL){
      inorder(root->l);
      inorder(root->r);
      printf("%d\t",root->data);
   }
}
void main(){
   int ch,x;
   do{
      printf("\n1. Insert\n2. Delete\n3. Inorder Traversal\n4. Preorder
Traversal\n5. Postorder Traversal\n6. Exit\nEnter your choice(1-6) : ");
      scanf("%d",&ch);
      switch(ch){
         case 1: printf("Enter the element : ");
               scanf("%d",&x);
               root=insert(root,x);
               break;
         case 2: printf("Enter the element : ");
               scanf("%d",&x);
               root=del(root,x);
               break;
         case 3: printf("Inorder Traversal : ");
               inorder(root);
               break;
         case 4: printf("Preorder Traversal : ");
               preorder(root);
               break;
         case 5: printf("Postorder Traversal : ");
               postorder(root);
               break;
      }
   }while(ch>0&&ch<6);
}
```

Output

```
mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc PGM16.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out PGM16.c

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice(1-6) : 1
Enter the element : 10

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice(1-6) : 1
Enter the element : 5

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice(1-6) : 1
Enter the element : 15

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice(1-6) : 3
Inorder Traversal : 5    10    15
```

```
1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice(1-6) : 4
Preorder Traversal : 10    5        15

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice(1-6) : 5
Postorder Traversal : 5    15      10

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice(1-6) : 2
Enter the element : 15

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice(1-6) : 3
Inorder Traversal : 5        10
```

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit

Enter your choice(1-6) : 6

mits@mits:~/Desktop/S1MCA/ADS_lab$

Experiment 17 **Date: 09.11.2023**

Bitstring Operations

Aim:

17. To implement set operations using bit strings.

Algorithm:

main()

1. Start
2. Declare int a[11], b[11], res[11], U[11]={1,2,3,4,5,6,7,8,9,10},s1,s2,ch;
3. Read size of bit-string 1 s1
4. Call input(a,s1) and display(a)
5. Read size of bit-string 2 s2
6. Call input(b,s2) and display(b)
7. Display choices.
8. Read option ch.
    a. if ch==1 call set_union().
    b. if ch==2 call set_intersection().
    c. if ch==3 call set_difference().
    d. if ch==3 if(set_equality())
        print Bit strings are equal.
    else
        print Bit strings are not equal.
9. Repeat step 3 while ch>0&&ch<4.
10. Stop.

void set_union()

1. Start
2. for(int i=1;i<11;i++)
        res[i]=a[i] | b[i];
3. display(res)
4. Exit.

void set_intersection()

1. Start
2. for(int i=1;i<11;i++)
        res[i]=a[i] & b[i];
3. display(res)
4. Exit.

void set_union()

1. Start
2. for(int i=1;i<11;i++)
   res[i]=a[i] & ~b[i];
3. display(res)
4. Exit.

bool set_equality()

1. Start
2. for(int i=1;i<11;i++)
   if a[i] != b[i]
   return false
3. return true
4. Exit.

void input(int bs[], int n)

1. Start
2. Declare x
3. for(int i=1;i<11;i++)
   read x
   bs[x]=1
4. Exit.

void display(int bs[])

1. Start
2. for(int i=1;i<11;i++)
   print bs[i]
3. Exit.

Program

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

void display(int bs[]){
   for(int i=1;i<11;i++){
      printf("%d\t",bs[i]);
   }
}
```

```
void input(int bs[], int n){
   int x;
   printf("Enter the elements : ");
   for(int i=0;i<n;i++){
      scanf("%d",&x);
      bs[x]=1;
   }
}

void set_union(){
   for(int i=1;i<11;i++){
      res[i]=a[i] | b[i];
   }

   printf("\nUnion Set : ");
   display(res);
}

void set_intersection(){
   for(int i=1;i<11;i++){
      res[i]=a[i] & b[i];
   }

   printf("\nIntersection Set : ");
   display(res);
}

void set_difference(){
   for(int i=1;i<11;i++){
      res[i]=a[i] & ~b[i];
   }

   printf("\nDifference Set : ");
   display(res);
}
```

```
bool set_equality(){
   for(int i=1;i<11;i++){
      if(a[i] != b[i]){
         return false;
      }
   }
   return true;
}
```

Output

```
mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc PGM17.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out PGM17.c

Enter the size of bit-string 1 : 5
Enter the elements : 1 3 5 7 9
Set A : 1      0      1      0      1      0      1      0      1      0

Enter the size of bit-string 2 : 5
Enter the elements : 2 4 6 8 10
Set B : 0      1      0      1      0      1      0      1      0      1

1. Union
2. Intersection
3. Difference
4. Equality
5. Exit
Enter your choice : 1
Union Set : 1      1      1      1      1      1      1      1      1      1

1. Union
2. Intersection
3. Difference
4. Equality
5. Exit
Enter your choice : 2
Intersection Set : 0 0      0      0      0      0      0      0      0      0
```

```
1. Union
2. Intersection
3. Difference
4. Equality
5. Exit
Enter your choice : 3
Difference Set : 1  0     1     0     1     0     1     0     1     0

1. Union
2. Intersection
3. Difference
4. Equality
5. Exit
Enter your choice : 4
Bit strings are not equal

1. Union
2. Intersection
3. Difference
4. Equality
5. Exit
Enter your choice : 5

mits@mits:~/Desktop/S1MCA/ADS_lab$
```