SIR C R REDDY COLLEGE OF ENGINEERING

FIRST YEAR ENGINEERING DEPARTMENT

DATA STRUCTURES THROUGH C

LAB MANUAL

Subject: Data Structures Through C Lab **Code: R201234**

Class: I/IV B.Tech EEE (A & B) **Semester: II**

Prepared By

K.SREENU

2021-22

Experiment 1: Implement operations on Strings

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

int length(char str[]);
void reverse(char str[]);
void copy(char str1[], char str2[]);
int compare(char str1[], char str2[]);
void concat(char str1[], char str2[]);

void main() {
  char a[100], b[100];
  int result, option;
  do {
    printf("\n1.Length of a string");
    printf("\n2.Reverse the Given String");
    printf("\n3.Copy");
    printf("\n4.String Comparison");
    printf("\n5.String Concatenation");
    printf("\n6.Quit");
    printf("\n\nEnter Your Choice:");
    scanf("%d", &option);


    switch (option) {
    case 1:
      printf("\nEnter a String:");
      scanf("%s",a );
      result = length(a);
      printf("\nLength of %s=%d", a, result);
```

```
    printf("\nPress a Character");
    getch();
    break;

  case 2:
    printf("\nEnter a String:");
   scanf("%s",a );
    reverse(a);
    printf("\nResult=%s", a);
    printf("\nPress a Character");
    getch();
    break;

  case 3:
    printf("\nEnter a String:");
    scanf("%s",a );
    copy(b, a);
    printf("\nResult=%s", b);
    printf("\nPress a Character");
    getch();
    break;

  case 4:
    printf("\nEnter 1st string:");
    scanf("%s",a );
    printf("\nEnter 2nd string:");
    scanf("%s",b );
    result = compare(a, b);
    if (result == 0)
      printf("\nboth are same");
    else if (result > 0)
      printf("\n1st>2nd");
    else
```

```
        printf("\n1st<2nd");
      printf("\nPress a Character");
      getch();
      break;

    case 5:
      printf("\nEnter 1st string:");
      scanf("%s",a );
      printf("\nEnter 2nd string:");
      scanf("%s",b );
      concat(a, b);
      printf("\nresult=%s", a);
      printf("\nPress a Character");
      getch();
      break;

    default:
      printf("\nInvalid Choice:");
      break;
    }
  } while (option != 6);
}

int length(char str[]) {
  int i = 0;
  while (str[i] != '\0')
    i++;
  return (i);
}

void reverse(char str[]) {
  int i, j;
  char temp;
```

```
  i = j = 0;
  while (str[j] != '\0')
    j++;
  j--;
  while (i < j) {
    temp = str[i];
    str[i] = str[j];
    str[j] = temp;
    i++;
    j--;
  }
}

void copy(char str2[], char str1[]) {
  int i = 0;
  while (str1[i] != '\0') {
    str2[i] = str1[i];
    i++;
  }
  str2[i] = '\0';
}

int compare(char str1[], char str2[]) {
  int i;
  i = 0;
  while (str1[i] != '\0') {
    if (str1[i] > str2[i])
      return (1);
    if (str1[i] < str2[i])
      return (-1);
    i++;
  }
  return (0);
```

```
}


void concat(char str1[], char str2[]) {
  int i, j;
  i = 0;
  while (str1[i] != '\0')
    i++;
  for (j = 0; str2[j] != '\0'; i++, j++)
    str1[i] = str2[j];
  str1[i] = '\0';
}
```

OUTPUT:

```
1.Length of a string
2.Reverse the Given String
3.Copy
4.String Comparison
5.String Concatenation
6.Quit

Enter Your Choice: 1
Enter a String: sreenu

Length of sreenu=6
Press a Character
1.Length of a string
2.Reverse the Given String
3.Copy
4.String Comparison
5.String Concatenation
6.Quit
```

Enter Your Choice:2

Enter a String: sreenu

Result=uneers

Press a Character

1.Length of a string

2.Reverse the Given String

3.Copy

4.String Comparison

5.String Concatenation

6.Quit

Enter Your Choice:3

Enter a String:sreenu

Result=sreenu

Press a Character

1.Length of a string

2.Reverse the Given String

3.Copy

4.String Comparison

5.String Concatenation

6.Quit

Enter Your Choice:4

Enter 1st string:sreenu

Enter 2nd string:sree

1st>2nd

Press a Character

1.Length of a string

2.Reverse the Given String

3.Copy

4.String Comparison

5.String Concatenation

6.Quit

Enter Your Choice: 5

Enter 1st string: sree

Enter 2nd string: sree

result=sreesree

Press a Character

1.Length of a string

2.Reverse the Given String

3.Copy

4.String Comparison

5.String Concatenation

6.Quit

Enter Your Choice: 6

. ...Program finished with exit code 0

Press ENTER to exit con

Experiment 2: Implement basic operations on Stacks.

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 50
int a[MAX];
int top;
void initialize();
int isEmpty();
int isFull();
int size();
int peek();
void push(int x);
int pop();
void display();

main()
{
   int choice, x;
   initialize();
   while(1)
   {
      printf("1. Push an element on the top.\n");
      printf("2. Pop an element from the top.\n");
      printf("3. Display the top element.\n");
      printf("4. Display all stack elements.\n");
      printf("5. Display size of the stack.\n");
      printf("6. Quit.\n");
```

```
printf("Enter your choice : ");
scanf("%d", &choice);

if(choice == 6)
   break;
switch(choice)
{
case 1:
   printf("Enter the element to be pushed : ");
   scanf("%d", &x);
   push(x);
   break;
case 2:
   x = pop();
   printf("Popped element is : %d\n",x);
   break;
case 3:
   printf("Element at the top is : %d\n", peek());
   break;
case 4:
   display();
   break;
case 5:
   printf("Size of stack = %d\n", size());
   break;
default:
   printf("Wrong Choice!!\n");
```

```
            break;
        }
        printf("\n");
    }
}
void initialize()
{2
    top = -1;
}
int size()
{
    return top+1;
}
int isEmpty()
{
    if(top == -1)
        return 1;
    else
        return 0;
}
int isFull()
{
    if(top == MAX-1)
        return 1;
    else
        return 0;
}
```

```
void push(int x)
{
   if(isFull())
   {
      printf("Stack Overflow\n");
      return;
   }
   ///else
   top = top + 1;
   a[top] = x;
}
int pop()
{
   int x;
   if(isEmpty())
   {
      printf("Stack Underflow\n");
      exit(1);
   }
   ///else
   x = a[top];
   top = top - 1;
   return x;
}
int peek()  ///returns the top value
{
   if(isEmpty())
```

```
    {
        printf("Stack Underflow\n");
        exit(1);
    }
    ///else
    return a[top];
}
void display()
{
    int i;
    printf("top = %d\n", top);
    if(isEmpty())
    {
        printf("Stack is Empty\n");
    }
    ///else
    printf("Stack is : \n\n");
    for(i = top; i>=0; i--)
        printf(" %d\n", a[i]);
    printf("\n");
}
```

OUTPUT:

1. Push an element on the top.

2. Pop an element from the top.

3. Display the top element.

4. Display all stack elements.

5. Display size of the stack.

6. Quit.

Enter your choice : 1

Enter the element to be pushed : 30

1. Push an element on the top.

2. Pop an element from the top.

3. Display the top element.

4. Display all stack elements.

5. Display size of the stack.

6. Quit.

Enter your choice : 1

Enter the element to be pushed : 60

1. Push an element on the top.

2. Pop an element from the top.

3. Display the top element.

4. Display all stack elements.

5. Display size of the stack.

6. Quit.

Enter your choice : 1

Enter the element to be pushed : 100

1. Push an element on the top.

2. Pop an element from the top.

3. Display the top element.

4. Display all stack elements.

5. Display size of the stack.

6. Quit.

Enter your choice : 4

top = 2

Stack is :

100

60

30

1. Push an element on the top.

2. Pop an element from the top.

3. Display the top element.

4. Display all stack elements.

5. Display size of the stack.

6. Quit.

Enter your choice : 3

Element at the top is : 100

1. Push an element on the top.

2. Pop an element from the top.

3. Display the top element.

4. Display all stack elements.

5. Display size of the stack.

6. Quit.

Enter your choice : 5

Size of stack = 3

1. Push an element on the top.

2. Pop an element from the top.

3. Display the top element.

4. Display all stack elements.

5. Display size of the stack.

6. Quit.

Enter your choice : 2

Popped element is : 100

1. Push an element on the top.

2. Pop an element from the top.

3. Display the top element.

4. Display all stack elements.

5. Display size of the stack.

6. Quit.

Enter your choice : 4

top = 1

Stack is :

60

30

1. Push an element on the top.

2. Pop an element from the top.

3. Display the top element.

4. Display all stack elements.

5. Display size of the stack.

6. Quit.

Enter your choice : 6

...Program finished with exit code 0

Press ENTER to exit console.

Experiment 3: Implement basic operations on Queue.

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

int queue_arr[MAX];
int rear=-1;
int front=-1;

void insert(int item);
int del();
int peek();
void display();
int isFull();
int isEmpty();

main()
{
        int choice,item;
        while(1)
        {
                printf("1.Insert\n");
                printf("2.Delete\n");
                printf("3.Display element at the front\n");
                printf("4.Display all elements of the queue\n");
                printf("5.Quit\n");
                printf("Enter your choice : ");
```

```
        scanf("%d",&choice);

        switch(choice)
        {
        case 1:
                printf("Input the element for adding in queue : ");
                scanf("%d",&item);
                insert(item);
                break;
        case 2:
                item=del();
                printf("Deleted element is  %d\n",item);
                break;
        case 3:
                printf("Element at the front is %d\n",peek());
                break;
        case 4:
                display();
                break;
        case 5:
                exit(1);
        default:
                printf("Wrong choice\n");
        }/*End of switch*/
    }/*End of while*/
}/*End of main()*/
```

```
void insert(int item)
{
        if( isFull() )
        {
                printf("Queue Overflow\n");
                return;
        }
        if( front == -1 )
                front=0;
        rear=rear+1;
        queue_arr[rear]=item ;
}/*End of insert()*/

int del()
{
        int item;
        if( isEmpty() )
        {
                printf("Queue Underflow\n");
                exit(1);
        }
        item=queue_arr[front];
        front=front+1;
        return item;
}/*End of del()*/

int peek()
```

```
{
        if( isEmpty() )
        {
                printf("Queue Underflow\n");
                exit(1);
        }
        return queue_arr[front];
}/*End of peek()*/

int isEmpty()
{
        if( front==-1 || front==rear+1 )
                return 1;
        else
                return 0;
}/*End of isEmpty()*/

int isFull()
{
        if( rear==MAX-1 )
                return 1;
        else
                return 0;
}/*End of isFull()*/

void display()
{
```

```
        int i;
        if ( isEmpty() )
        {
                printf("Queue is empty\n");
                return;
        }
        printf("Queue is :\n\n");
        for(i=front;i<=rear;i++)
                printf("%d  ",queue_arr[i]);
        printf("\n\n");
}/*End of display() */
```

OUTPUT:

1.Insert

2.Delete

3.Display element at the front

4.Display all elements of the queue

5.Quit

Enter your choice : 1

Input the element for adding in queue : 1

1.Insert

2.Delete

3.Display element at the front

4.Display all elements of the queue

5.Quit

Enter your choice : 10

Wrong choice

1.Insert

2.Delete

3.Display element at the front

4.Display all elements of the queue

5.Quit

Enter your choice : 1

Input the element for adding in queue : 20

1.Insert

2.Delete

3.Display element at the front

4.Display all elements of the queue

5.Quit

Enter your choice : 1

Input the element for adding in queue : 30

1.Insert

2.Delete

3.Display element at the front

4.Display all elements of the queue

5.Quit

Enter your choice : 1

Input the element for adding in queue : 40

1.Insert

2.Delete

3.Display element at the front

4.Display all elements of the queue

5.Quit

Enter your choice : 4

Queue is :

1 20 30 40

1.Insert

2.Delete

3.Display element at the front

4.Display all elements of the queue

5.Quit

Enter your choice : 3

Element at the front is 1

1.Insert

2.Delete

3.Display element at the front

4.Display all elements of the queue

5.Quit

Enter your choice : 2

Deleted element is 1

1.Insert

2.Delete

3.Display element at the front

4.Display all elements of the queue

5.Quit

Enter your choice : 4

Queue is :

20 30 40

1.Insert

2.Delete

3.Display element at the front

4.Display all elements of the queue

5.Quit

Enter your choice : 5

...Program finished with exit code 1

Press ENTER to exit console.

Experiment 4: Implement basic operations on Circular Queue.

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

int cqueue_arr[MAX];
int front=-1;
int rear=-1;

void display( );
void insert(item);
int del();
int peek();
int isEmpty();
int isFull();

main()
{
        int choice,item;
        while(1)
        {
                printf("1.Insert\n");
                printf("2.Delete\n");
                printf("3.Peek\n");
                printf("4.Display\n");
                printf("5.Quit\n");
                printf("Enter your choice : ");
```

```
            scanf("%d",&choice);

            switch(choice)
            {
            case 1 :
                  printf("Input the element for insertion : ");
                  scanf("%d",&item);
                  insert(item);
                  break;
            case 2 :
                  printf("Element deleted is : %d\n",del());
                  break;
            case 3:
                  printf("Element at the front is  : %d\n",peek());
                  break;
            case 4:
                  display();
                  break;
            case 5:
                  exit(1);
            default:
                  printf("Wrong choice\n");
            }/*End of switch*/
      }/*End of while */
}/*End of main()*/

void insert(int item)
```

```
{
        if( isFull() )
        {
                printf("Queue Overflow\n");
                return;
        }
        if(front == -1 )
                front=0;

        if(rear==MAX-1)/*rear is at last position of queue*/
                rear=0;
        else
                rear=rear+1;
        cqueue_arr[rear]=item ;
}/*End of insert()*/

int del()
{
        int item;
        if( isEmpty() )
        {
                printf("Queue Underflow\n");
                exit(1);
        }
        item=cqueue_arr[front];
        if(front==rear) /* queue has only one element */
        {
```

```
                front=-1;
                rear=-1;
        }
        else if(front==MAX-1)
                front=0;
        else
                front=front+1;
        return item;
}/*End of del() */

int isEmpty()
{
        if(front==-1)
                return 1;
        else
                return 0;
}/*End of isEmpty()*/

int isFull()
{
        if((front==0 && rear==MAX-1) || (front==rear+1))
                return 1;
        else
                return 0;
}/*End of isFull()*/

int peek()
```

```
{
        if( isEmpty() )
        {
                printf("Queue Underflow\n");
                exit(1);
        }
        return cqueue_arr[front];
}/*End of peek()*/

void display()
{
        int i;
        if(isEmpty())
        {
                printf("Queue is empty\n");
                return;
        }
        printf("Queue elements :\n");
        i=front;
        if( front<=rear )
        {
                while(i<=rear)
                        printf("%d ",cqueue_arr[i++]);
        }
        else
        {
                while(i<=MAX-1)
```

```
                printf("%d ",cqueue_arr[i++]);
        i=0;
        while(i<=rear)
                printf("%d ",cqueue_arr[i++]);
    }
    printf("\n");
}/*End of display() */
```

Ouput:

```
1.Insert
2.Delete
3.Peek
4.Display
5.Quit
Enter your choice : 1
Input the element for insertion : 10
1.Insert
2.Delete
3.Peek
4.Display
5.Quit
Enter your choice : 2
Element deleted is : 10
1.Insert
2.Delete
3.Peek
4.Display
```

5.Quit

Enter your choice : 1

Input the element for insertion : 20

1.Insert

2.Delete

3.Peek

4.Display

5.Quit

Enter your choice : 1

Input the element for insertion : 30

1.Insert

2.Delete

3.Peek

4.Display

5.Quit

Enter your choice : 1

Input the element for insertion : 40

1.Insert

2.Delete

3.Peek

4.Display

5.Quit

Enter your choice : 4

Queue elements :

20 30 40

1.Insert

2.Delete

3.Peek

4.Display

5.Quit

Enter your choice : 3

Element at the front is : 20

1.Insert

2.Delete

3.Peek

4.Display

5.Quit

Enter your choice : 2

Element deleted is : 20

1.Insert

2.Delete

3.Peek

4.Display

5.Quit

Enter your choice : 4

Queue elements :

30 40

1.Insert

2.Delete

3.Peek

4.Display

5.Quit

Enter your choice : 5

Experiment 5: Implement multi stack in a single array

Multi stack using array

```
#include <stdio.h>
#define SIZE 20
 int array[SIZE];  // declaration of array type variable.
int top1 = -1;
int top2 = SIZE;

//Function to push data into stack1
void push1 (int data)
{
// checking the overflow condition
  if (top1 < top2 - 1)
  {
     top1++;
    array[top1] = data;
  }
  else
  {
    printf ("Stack is full");
  }
}
// Function to push data into stack2
void push2 (int data)
{
// checking overflow condition
if (top1 < top2 - 1)
```

```
  {
    top2--;
    array[top2] = data;
  }
  else
  {
    printf ("Stack is full..\n");
  }
}

//Function to pop data from the Stack1
void pop1 ()
{
// Checking the underflow condition
 if (top1 >= 0)
  {
    int popped_element = array[top1];
    top1--;

    printf ("%d is being popped from Stack 1\n", popped_element);
  }
  else
  {
    printf ("Stack is Empty \n");
  }
}
// Function to remove the element from the Stack2
```

```
void pop2 ()
{
// Checking underflow condition
if (top2 < SIZE)
 {
    int popped_element = array[top2];
   top2--;

   printf ("%d is being popped from Stack 1\n", popped_element);
 }
 else
 {
   printf ("Stack is Empty!\n");
 }
}

//Functions to Print the values of Stack1
void display_stack1 ()
{
 int i;
 for (i = top1; i >= 0; --i)
 {
   printf ("%d ", array[i]);
 }
 printf ("\n");
}
// Function to print the values of Stack2
```

```
void display_stack2 ()
{
  int i;
  for (i = top2; i < SIZE; ++i)
  {
    printf ("%d ", array[i]);
  }
  printf ("\n");
}

int main()
{
  int ar[SIZE];
  int i;
  int num_of_ele;

  printf ("We can push a total of 20 values\n");

  //Number of elements pushed in stack 1 is 10
  //Number of elements pushed in stack 2 is 10

// loop to insert the elements into Stack1
for (i = 1; i <= 10; ++i)
{
    push1(i);
    printf ("Value Pushed in Stack 1 is %d\n", i);
  }
```

```
// loop to insert the elements into Stack2.
for (i = 11; i <= 20; ++i)
 {
   push2(i);
   printf ("Value Pushed in Stack 2 is %d\n", i);
 }

 //Print Both Stacks
 display_stack1 ();
display_stack2 ();

 //Pushing on Stack Full
 printf ("Pushing Value in Stack 1 is %d\n", 11);
 push1 (11);

 //Popping All Elements from Stack 1
 num_of_ele = top1 + 1;
 while (num_of_ele)
 {
   pop1 ();
  --num_of_ele;
 }
   // Trying to Pop the element From the Empty Stack
 pop1 ();
   return 0;
}
```

OUTPUT:

```
We can push a total of 20 values
Value Pushed in Stack 1 is 1
Value Pushed in Stack 1 is 2
Value Pushed in Stack 1 is 3
Value Pushed in Stack 1 is 4
Value Pushed in Stack 1 is 5
Value Pushed in Stack 1 is 6
Value Pushed in Stack 1 is 7
Value Pushed in Stack 1 is 8
Value Pushed in Stack 1 is 9
Value Pushed in Stack 1 is 10
Value Pushed in Stack 2 is 11
Value Pushed in Stack 2 is 12
Value Pushed in Stack 2 is 13
Value Pushed in Stack 2 is 14
Value Pushed in Stack 2 is 15
Value Pushed in Stack 2 is 16
Value Pushed in Stack 2 is 17
Value Pushed in Stack 2 is 18
Value Pushed in Stack 2 is 19
Value Pushed in Stack 2 is 20
10 9 8 7 6 5 4 3 2 1
20 19 18 17 16 15 14 13 12 11
Pushing Value in Stack 1 is 11
Stack is full10 is being popped from Stack 1
9 is being popped from Stack 1
8 is being popped from Stack 1
7 is being popped from Stack 1
6 is being popped from Stack 1
5 is being popped from Stack 1
4 is being popped from Stack 1
3 is being popped from Stack 1
2 is being popped from Stack 1
1 is being popped from Stack 1
Stack is Empty
```

Experiment 6: Implement List data structure using i) array ii) singly linked list

Array using list

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define LIST_SIZE 30
void main()
{
int *element=NULL;
int ch,i,j,n;
int insdata,deldata,moddata,found;
int top=-1;
element=(int*)malloc(sizeof(int)* LIST_SIZE);
while(1)
{
fflush(stdin);
printf("\n\n basic Operations in a Linear List.....");
printf("\n 1.Create New List \t 2.Modify List \t 3.View List");
printf("\n 4.Insert First \t 5.Insert Last \t 6.Insert Middle");
printf("\n 7.Delete First \t 8.Delete Last \t 9.Delete Middle");
printf("\nEnter the Choice 1 to 10 : ");
scanf("%d",&ch);
switch(ch)
{
case 1:
```

```
top=-1;
printf("\n Enter the Limit (How many Elements):");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\n Enter The Element [%d]:",(i+1));
scanf("%d",&element[++top]);
}
break;
case 2:
if(top==-1)
{
printf("\n Linear List is Empty:");
break;
}
printf("\n Enter the Element for Modification:");
scanf("%d",&moddata);
found=0;
for(i=0;i<=top;i++)
{
if(element[i]==moddata)
{
found=1;
printf("\n Enter The New Element :");
scanf("%d",&element[i]);
break;
}
```

```
}
if(found==0)
printf("\n Element %d not found",moddata);
break;
case 3:
if(top==-1)
printf("\n \n Linear List is Empty:");
else if(top==LIST_SIZE -1)
printf("\n Linear LIst is Full:");
for(i=0;i<=top;i++)
printf("\n Element[%d]is-->%d",(i+1),element[i]);
break;
case 4:
if(top==LIST_SIZE-1)
{
printf("\n Linear List is Full:");
break;
}
top++;
for(i=top;i>0;i--)
element[i]=element[i-1];
printf("\n Enter the Element:");
scanf("%d",&element[0]);
break;
case 5:
if(top==LIST_SIZE-1)
{
```

```
printf("\n Linear List is Full:");
break;
}
printf("\n Enter the Element:");
scanf("%d",&element[++top]);
break;
case 6:
if(top==LIST_SIZE-1)
printf("\n Linear List is Full:");
else if(top==-1)
printf("\n linear List is Empty.");
else
{
found=0;
printf("\n Enter the Element after which the insertion is to be made:");
scanf("%d",&insdata);
for(i=0;i<=top;i++)
if(element[i]==insdata)
{
found=1;
top++;
for(j=top;j>i;j--)
element[j]=element[j-1];
printf("\n Enter the Element :");
scanf("%d",&element[i+1]);
break;
}
```

```
if(found==0)
printf("\n Element %d Not Found",insdata);
}
break;
case 7:
if(top==-1)
{
printf("\n Linear List is Empty:");
break;
}
printf("\n Deleted Data-->Element :%d",element[0]);
top--;
for(i=0;i<=top;i++)
element[i]=element[i+1];
break;
case 8:
if(top==-1)
printf("\n Linear List is Empty:");
else
printf("\n Deleted Data-->Element :%d",element[top--]);
break;
case 9:
if(top==-1)
{
printf("\n Linear List is Empty:");
break;
}
```

```
printf("\n Enter the Element for Deletion :");
scanf("%d",&deldata);
found=0;
for(i=0;i<=top;i++)
if(element[i]==deldata)
{
found=1;
printf("\n Deleted data-->Element :%d",element[i]);
top--;
for(j=i;j<=top;j++)
element[j]=element[j+1];
break;
}
if(found==0)
printf("\n Element %d Not Found ",deldata);
break;
default:
free(element);
printf("\n End Of Run Of Your Program.....");
exit(0);
}
}
}
```

OUTPUT:

```
basic Operations in a Linear List.....
 1.Create New List      2.Modify List   3.View List
 4.Insert First         5.Insert Last   6.Insert Middle
```

7.Delete First 8.Delete Last 9.Delete Middle

Enter the Choice 1 to 10 : 1

Enter the Limit (How many Elements):4

Enter The Element [1]:10

Enter The Element [2]:20

Enter The Element [3]:30

Enter The Element [4]:40

basic Operations in a Linear List.....

1.Create New List 2.Modify List 3.View List

4.Insert First 5.Insert Last 6.Insert Middle

7.Delete First 8.Delete Last 9.Delete Middle

Enter the Choice 1 to 10 : 3

Element[1]is-->10

Element[2]is-->20

Element[3]is-->30

Element[4]is-->40

basic Operations in a Linear List.....

1.Create New List 2.Modify List 3.View List

4.Insert First 5.Insert Last 6.Insert Middle

7.Delete First 8.Delete Last 9.Delete Middle

Enter the Choice 1 to 10 : 4

Enter the Element:5

basic Operations in a Linear List.....

1.Create New List 2.Modify List 3.View List

4.Insert First 5.Insert Last 6.Insert Middle

7.Delete First 8.Delete Last 9.Delete Middle

Enter the Choice 1 to 10 : 3

Element[1]is-->5

Element[2]is-->10

Element[3]is-->20

Element[4]is-->30

Element[5]is-->40

basic Operations in a Linear List.....

1.Create New List 2.Modify List 3.View List

4.Insert First 5.Insert Last 6.Insert Middle

7.Delete First 8.Delete Last 9.Delete Middle

Enter the Choice 1 to 10 : 5

Enter the Element:45

basic Operations in a Linear List.....

1.Create New List 2.Modify List 3.View List

4.Insert First 5.Insert Last 6.Insert Middle

7.Delete First 8.Delete Last 9.Delete Middle

Enter the Choice 1 to 10 : 3

Element[1]is-->5

Element[2]is-->10

Element[3]is-->20

Element[4]is-->30

Element[5]is-->40

Element[6]is-->45

basic Operations in a Linear List.....

1.Create New List 2.Modify List 3.View List

4.Insert First 5.Insert Last 6.Insert Middle

7.Delete First 8.Delete Last 9.Delete Middle

Enter the Choice 1 to 10 : 6

Enter the Element after which the insertion is to be made:20

Enter the Element :25

basic Operations in a Linear List.....

1.Create New List 2.Modify List 3.View List

4.Insert First 5.Insert Last 6.Insert Middle

7.Delete First 8.Delete Last 9.Delete Middle

Enter the Choice 1 to 10 : 3

Element[1]is-->5

Element[2]is-->10

Element[3]is-->20

Element[4]is-->25

Element[5]is-->30

Element[6]is-->40

Element[7]is-->45

basic Operations in a Linear List.....

1.Create New List 2.Modify List 3.View List

4.Insert First 5.Insert Last 6.Insert Middle

7.Delete First 8.Delete Last 9.Delete Middle

Enter the Choice 1 to 10 : 7

Deleted Data-->Element :5

basic Operations in a Linear List.....

1.Create New List 2.Modify List 3.View List

4.Insert First 5.Insert Last 6.Insert Middle

7.Delete First 8.Delete Last 9.Delete Middle

Enter the Choice 1 to 10 : 3

Element[1]is-->10

```
Element[2]is-->20

Element[3]is-->25

Element[4]is-->30

Element[5]is-->40

Element[6]is-->45


basic Operations in a Linear List.....

1.Create New List      2.Modify List   3.View List

4.Insert First         5.Insert Last   6.Insert Middle

7.Delete First         8.Delete Last   9.Delete Middle

Enter the Choice 1 to 10 : 8


Deleted Data-->Element :45


basic Operations in a Linear List.....

1.Create New List      2.Modify List   3.View List

4.Insert First         5.Insert Last   6.Insert Middle

7.Delete First         8.Delete Last   9.Delete Middle

Enter the Choice 1 to 10 : 3


Element[1]is-->10

Element[2]is-->20

Element[3]is-->25

Element[4]is-->30

Element[5]is-->40


basic Operations in a Linear List.....
```

```
1.Create New List      2.Modify List   3.View List
4.Insert First         5.Insert Last   6.Insert Middle
7.Delete First         8.Delete Last   9.Delete Middle
Enter the Choice 1 to 10 : 9

Enter the Element for Deletion :25

Deleted data-->Element :25

basic Operations in a Linear List.....
1.Create New List      2.Modify List   3.View List
4.Insert First         5.Insert Last   6.Insert Middle
7.Delete First         8.Delete Last   9.Delete Middle
Enter the Choice 1 to 10 : 3

Element[1]is-->10
Element[2]is-->20
Element[3]is-->30
Element[4]is-->40

basic Operations in a Linear List.....
1.Create New List      2.Modify List   3.View List
4.Insert First         5.Insert Last   6.Insert Middle
7.Delete First         8.Delete Last   9.Delete Middle
Enter the Choice 1 to 10 : 2

Enter the Element for Modification:40
```

```
Enter The New Element :50


basic Operations in a Linear List.....
1.Create New List      2.Modify List   3.View List
4.Insert First         5.Insert Last   6.Insert Middle
7.Delete First         8.Delete Last   9.Delete Middle
Enter the Choice 1 to 10 : 3

Element[1]is-->10
Element[2]is-->20
Element[3]is-->30
Element[4]is-->50

basic Operations in a Linear List.....
1.Create New List      2.Modify List   3.View List
4.Insert First         5.Insert Last   6.Insert Middle
7.Delete First         8.Delete Last   9.Delete Middle
Enter the Choice 1 to 10 :
```

Singly LINKED LIST

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

struct node
{
        int info;
        struct node *link;
};

struct node *createList(struct node *start);
void display(struct node *start);
void countNodes(struct node *start);
void search(struct node *start,int data);
struct node *insertInBeginning(struct node *start,int data);
struct node *insertAtEnd(struct node *start,int data);
struct node *insertAfter(struct node *start,int data,int item);
struct node *insertBefore(struct node *start,int data,int item );
struct node *insertAtPosition(struct node *start,int data,int pos);
struct node *deleteNode(struct node *start,int data);

main()
{
        struct node *start=NULL;
        int choice,data,item,pos;
```

```
while(1)
{
        printf("1.Create List\n");
        printf("2.Display\n");
        printf("3.Count\n");
        printf("4.Search\n");
        printf("5.Add to empty list / Add at beginning\n");
        printf("6.Add at end\n");
        printf("7.Add after node\n");
        printf("8.Add before node\n");
        printf("9.Add at position\n");
        printf("10.Delete\n");
        printf("11.Quit\n\n");

        printf("Enter your choice : ");
        scanf("%d",&choice);

        switch(choice)
        {
         case 1:
                start=createList(start);
                break;
         case 2:
                display(start);
                break;
         case 3:
                countNodes(start);
```

```
..              break;
                case 4:
                        printf("Enter the element to be searched : ");
                        scanf("%d",&data);
                        search(start,data);
                        break;
                case 5:
                        printf("Enter the element to be inserted : ");
                        scanf("%d",&data);
                        start=insertInBeginning(start,data);
                        break;
                case 6:
                        printf("Enter the element to be inserted : ");
                        scanf("%d",&data);
                        start=insertAtEnd(start,data);
                        break;
                case 7:
                        printf("Enter the element to be inserted : ");
                        scanf("%d",&data);
                        printf("Enter the element after which to insert : ");
                        scanf("%d",&item);
                        start=insertAfter(start,data,item);
                        break;
                case 8:
                        printf("Enter the element to be inserted : ");
                        scanf("%d",&data);
                        printf("Enter the element before which to insert : ");
```

```
                scanf("%d",&item);
                start=insertBefore(start,data,item);
                break;
            case 9:
                printf("Enter the element to be inserted : ");
                scanf("%d",&data);
                printf("Enter the position at which to insert : ");
                scanf("%d",&pos);
                start=insertAtPosition(start,data,pos);
                break;
            case 10:
                printf("Enter the element to be deleted : ");
                scanf("%d",&data);
                start=deleteNode(start, data);
                break;
            case 11:
                 exit(1);
            default:
                 printf("Wrong choice\n");
            }/*End of switch */
        }/*End of while */
}/*End of main()*/

struct node *createList(struct node *start)
{
        int i,n,data;
        printf("Enter the number of nodes : ");
```

```
        scanf("%d",&n);
        start=NULL;
        if(n==0)
                return start;

        printf("Enter the element to be inserted : ");
        scanf("%d",&data);
        start=insertInBeginning(start,data);

        for(i=2;i<=n;i++)
        {
                printf("Enter the element to be inserted : ");
                scanf("%d",&data);
                start=insertAtEnd(start,data);
        }
        return start;
}/*End of create_list()*/

void display(struct node *start)
{
        struct node *p;
        if(start==NULL)
        {
                printf("List is empty\n");
                return;
        }
        p=start;
```

```
        printf("List is :\n");
        while(p!=NULL)
        {
                printf("%d ",p->info);
                p=p->link;
        }
        printf("\n\n");
}/*End of display() */

void countNodes(struct node *start)
{
        struct node *p;
        int cnt=0;
        p=start;
        while(p!=NULL)
        {
                p=p->link;
                cnt++;
        }
        printf("Number of elements are %d\n",cnt);
}/*End of countNodes() */

void search(struct node *start,int item)
{
        struct node *p=start;
        int pos=1;
        while(p!=NULL)
```

```
        {
                if(p->info==item)
                {
                        printf("Item %d found at position %d\n",item,pos);
                        return;
                }
                p=p->link;
                pos++;
        }
        printf("Item %d not found in list\n",item);
}/*End of search()*/

struct node *insertInBeginning(struct node *start,int data)
{
        struct node *tmp;
        tmp=(struct node *)malloc(sizeof(struct node));
        tmp->info=data;
        tmp->link=start;
        start=tmp;
        return start;
}/*End of insertInBeginning()*/

struct node *insertAtEnd(struct node *start,int data)
{
        struct node *p,*tmp;
        tmp=(struct node *)malloc(sizeof(struct node));
        tmp->info=data;
```

```
        p=start;
        while(p->link!=NULL)
                p=p->link;
        p->link=tmp;
        tmp->link=NULL;
        return start;
}/*End of insertAtEnd()*/

struct node *insertAfter(struct node *start,int data,int item)
{
        struct node *tmp,*p;
        p=start;
        while(p!=NULL)
        {
                if(p->info==item)
                {
                        tmp=(struct node *)malloc(sizeof(struct node));
                        tmp->info=data;
                        tmp->link=p->link;
                        p->link=tmp;
                        return start;
                }
                p=p->link;
        }
        printf("%d not present in the list\n",item);
        return start;
}/*End of insertAfter()*/
```

```
struct node *insertBefore(struct node *start,int data,int item)
{
        struct node *tmp,*p;
        if(start==NULL )
        {
                printf("List is empty\n");
                return start;
        }
        /*If data to be inserted before first node*/
        if(item==start->info)
        {
                tmp=(struct node *)malloc(sizeof(struct node));
                tmp->info=data;
                tmp->link=start;
                start=tmp;
                return start;
        }
        p=start;
        while(p->link!=NULL)
        {
                if(p->link->info==item)
                {
                        tmp=(struct node *)malloc(sizeof(struct node));
                        tmp->info=data;
                        tmp->link=p->link;
                        p->link=tmp;
```

```
                return start;
            }
            p=p->link;
        }
        printf("%d not present in the list\n",item);
        return start;
}/*End of insertBefore()*/

struct node *insertAtPosition(struct node *start,int data,int pos)
{
        struct node *tmp,*p;
        int i;
        tmp=(struct node *)malloc(sizeof(struct node));
        tmp->info=data;
        if(pos==1)
        {
            tmp->link=start;
            start=tmp;
            return start;
        }
        p=start;
        for(i=1; i<pos-1 && p!=NULL; i++)
            p=p->link;
        if(p==NULL)
            printf("There are less than %d elements\n",pos);
        else
        {
```

```
                tmp->link=p->link;
                p->link=tmp;
        }
        return start;
}/*End of insertAtPosition()*/

struct node *deleteNode(struct node *start,int data)
{
        struct node *tmp,*p;
        if(start==NULL)
        {
                printf("List is empty\n");
                return start;
        }
        /*Deletion of first node*/
        if(start->info==data)
        {
                tmp=start;
                start=start->link;
                free(tmp);
                return start;
        }
        /*Deletion in between or at the end*/
        p=start;
        while(p->link!=NULL)
        {
                if(p->link->info==data)
```

```
                {
                        tmp=p->link;
                        p->link=tmp->link;
                        free(tmp);
                        return start;
                }
                p=p->link;
        }
        printf("Element %d not found\n",data);
        return start;
}/*End of deleteNode()*/
```

Output:

1.Create List

2.Display

3.Count

4.Search

5.Add to empty list / Add at beginning

6.Add at end

7.Add after node

8.Add before node

9.Add at position

10.Delete

11.Quit

Enter your choice : 1

Enter the number of nodes : 5

Enter the element to be inserted : 10

Enter the element to be inserted : 20

Enter the element to be inserted : 30

Enter the element to be inserted : 40

Enter the element to be inserted : 50

1.Create List

2.Display

3.Count

4.Search

5.Add to empty list / Add at beginning

6.Add at end

7.Add after node

8.Add before node

9.Add at position

10.Delete

11.Quit

Enter your choice : 2

List is :

10 20 30 40 50

1.Create List

2.Display

3.Count

4.Search

5.Add to empty list / Add at beginning

6.Add at end

7.Add after node

8.Add before node

9.Add at position

10.Delete

11.Quit

Enter your choice : 3

Number of elements are 5

1.Create List

2.Display

3.Count

4.Search

5.Add to empty list / Add at beginning

6.Add at end

7.Add after node

8.Add before node

9.Add at position

10.Delete

11.Quit

Enter your choice : 4

Enter the element to be searched : 40

Item 40 found at position 4

1.Create List

2.Display

3.Count

4.Search

5.Add to empty list / Add at beginning

6.Add at end

7.Add after node

8.Add before node

9.Add at position

10.Delete

11.Quit

Enter your choice : 5

Enter the element to be inserted : 5

1.Create List

2.Display

3.Count

4.Search

5.Add to empty list / Add at beginning

6.Add at end

7.Add after node

8.Add before node

9.Add at position

10.Delete

11.Quit

Enter your choice : 6

Enter the element to be inserted : 100

1.Create List

2.Display

3.Count

4.Search

5.Add to empty list / Add at beginning

6.Add at end

7.Add after node

8.Add before node

9.Add at position

10.Delete

11.Quit

Enter your choice : 2

List is :

5 10 20 30 40 50 100

1.Create List

2.Display

3.Count

4.Search

5.Add to empty list / Add at beginning

6.Add at end

7.Add after node

8.Add before node

9.Add at position

10.Delete

11.Quit

Enter your choice : 7

Enter the element to be inserted : 45

Enter the element after which to insert : 40

1.Create List

2.Display

3.Count

4.Search

5.Add to empty list / Add at beginning

6.Add at end

7.Add after node

8.Add before node

9.Add at position

10.Delete

11.Quit

Enter your choice : 2

List is :

5 10 20 30 40 45 50 100

1.Create List

2.Display

3.Count

4.Search

5.Add to empty list / Add at beginning

6.Add at end

7.Add after node

8.Add before node

9.Add at position

10.Delete

11.Quit

Enter your choice : 8

Enter the element to be inserted : 55

Enter the element before which to insert : 50

1.Create List

2.Display

3.Count

4.Search

5.Add to empty list / Add at beginning

6.Add at end

7.Add after node

8.Add before node

9.Add at position

10.Delete

11.Quit

Enter your choice : 2

List is :

5 10 20 30 40 45 55 50 100

1.Create List

2.Display

3.Count

4.Search

5.Add to empty list / Add at beginning

6.Add at end

7.Add after node

8.Add before node

9.Add at position

10.Delete

11.Quit

Enter your choice : 9

Enter the element to be inserted : 200

Enter the position at which to insert : 6

1.Create List

2.Display

3.Count

4.Search

5.Add to empty list / Add at beginning

6.Add at end

7.Add after node

8.Add before node

9.Add at position

10.Delete

11.Quit

Enter your choice : 2

List is :

5 10 20 30 40 200 45 55 50 100

1.Create List

2.Display

3.Count

4.Search

5.Add to empty list / Add at beginning

6.Add at end

7.Add after node

8.Add before node

9.Add at position

10.Delete

11.Quit

Enter your choice : 10

Enter the element to be deleted : 200

1.Create List

2.Display

3.Count

4.Search

5.Add to empty list / Add at beginning

6.Add at end

7.Add after node

8.Add before node

9.Add at position

10.Delete

11.Quit

Enter your choice : 2

List is :

5 10 20 30 40 45 55 50 100

1.Create List

2.Display

3.Count

4.Search

5.Add to empty list / Add at beginning

6.Add at end

7.Add after node

8.Add before node

9.Add at position

10.Delete

11.Quit

Enter your choice : 11

...Program finished with exit code 1

Press ENTER to exit console.

Experiment 7: Implement basic operations on doubly linked list

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

struct node
{
        struct node *prev;
        int info;
        struct node *next;
};

struct node *createList(struct node *start);
void display(struct node *start);
struct node *insertInEmpty(struct node *start,int data);
struct node *insertInBeginning(struct node *start,int data);
struct node *insertAtEnd(struct node *start,int data);
struct node *insertAfter(struct node *start,int data,int item);
struct node *insertBefore(struct node *start,int data,int item);
struct node *deleteNode(struct node *start,int data);

main()
{
        int choice,data,item;
        struct node *start=NULL;
        while(1)
        {
                printf("1.Create List\n");
```

```
printf("2.Display\n");
printf("3.Add to empty list\n");
printf("4.Add at beginning\n");
printf("5.Add at end\n");
printf("6.Add after\n");
printf("7.Add before\n");
printf("8.Delete\n");
printf("9.Quit\n");
printf("Enter your choice : ");
scanf("%d",&choice);
switch(choice)
{
 case 1:
        start=createList(start);
        break;
 case 2:
        display(start);
        break;
 case 3:
        printf("Enter the element to be inserted : ");
        scanf("%d",&data);
        start=insertInEmpty(start,data);
        break;
 case 4:
printf("Enter the element to be inserted : ");
        scanf("%d",&data);
        start=insertInBeginning(start,data);
```

```
            break;
case 5:
            printf("Enter the element to be inserted : ");
            scanf("%d",&data);
            start=insertAtEnd(start,data);
            break;
case 6:
            printf("Enter the element to be inserted : ");
            scanf("%d",&data);
            printf("Enter the element after which to insert : ");
            scanf("%d",&item);
            start=insertAfter(start,data,item);
            break;
case 7:
            printf("Enter the element to be inserted : ");
            scanf("%d",&data);
            printf("Enter the element before which to insert : ");
            scanf("%d",&item);
            start=insertBefore(start,data,item);
            break;
case 8:
            printf("Enter the element to be deleted : ");
            scanf("%d",&data);
            start=deleteNode(start,data);
            break;
case 9:
            exit(1);
```

```
                default:
                        printf("Wrong choice\n");
        }/*End of switch*/
  }/*End of while*/
}/*End of main()*/

struct node *createList(struct node *start)
{
        int i,n,data;
        printf("Enter the number of nodes : ");
        scanf("%d",&n);
        start=NULL;
        if(n==0)
                return start;
        printf("Enter the element to be inserted : ");
        scanf("%d",&data);
        start=insertInEmpty(start,data);

        for(i=2;i<=n;i++)
        {
                printf("Enter the element to be inserted : ");
                scanf("%d",&data);
                start=insertAtEnd(start,data);
        }
        return start;
}/*End of create_list()*/
```

```
void display(struct node *start)
{
        struct node *p;
        if(start==NULL)
        {
                printf("List is empty\n");
                return;
        }
        p=start;
        printf("List is :\n");
        while(p!=NULL)
        {
                printf("%d ",p->info);
                p=p->next;
        }
        printf("\n");
}/*End of display() */

struct node *insertInEmpty(struct node *start,int data)
{
        struct node *tmp;
        tmp=(struct node *)malloc(sizeof(struct node));
        tmp->info=data;
        tmp->prev=NULL;
        tmp->next=NULL;
        start=tmp;
        return start;
```

```
}/*End of insertInEmpty( )*/

struct node *insertInBeginning(struct node *start,int data)
{
        struct node *tmp;
        tmp = (struct node *)malloc(sizeof(struct node));
        tmp->info=data;
        tmp->prev=NULL;
        tmp->next=start;
        start->prev=tmp;
        start=tmp;
        return start;
}/*End of insertInBegiinning( )*/

struct node *insertAtEnd(struct node *start,int data)
{
        struct node *tmp,*p;
        tmp=(struct node *)malloc(sizeof(struct node));
        tmp->info=data;
        p=start;
        while(p->next!=NULL)
                p=p->next;
        p->next=tmp;
        tmp->next=NULL;
        tmp->prev=p;
        return start;
}/*End of insertAtEnd( )*/
```

```
struct node *insertAfter(struct node *start,int data,int item)
{
        struct node *tmp,*p;
        tmp=(struct node *)malloc(sizeof(struct node));
        tmp->info=data;
        p=start;
        while(p!=NULL)
        {
                if(p->info==item)
                {
                        tmp->prev=p;
                        tmp->next=p->next;
                        if(p->next!=NULL)
                                p->next->prev=tmp;
                        p->next=tmp;
                        return start;
                }
                p=p->next;
        }
        printf("%d not present in the list\n\n",item);
        return start;
}/*End of insertAfter()*/

struct node *insertBefore(struct node *start,int data,int item)
{
        struct node *tmp,*q;
```

```
if(start==NULL )
{
        printf("List is empty\n");
        return start;
}
if(start->info==item)
{
        tmp = (struct node *)malloc(sizeof(struct node));
        tmp->info=data;
        tmp->prev=NULL;
        tmp->next=start;
        start->prev=tmp;
        start=tmp;
        return start;
}
q=start;
while(q!=NULL)
{
        if(q->info==item)
        {
                tmp=(struct node *)malloc(sizeof(struct node));
                tmp->info=data;
                tmp->prev=q->prev;
                tmp->next = q;
                q->prev->next=tmp;
                q->prev=tmp;
                return start;
```

```
                }
                q=q->next;
        }
        printf("%d not present in the list\n",item);
        return start;
}/*End of insertBefore()*/

struct node *deleteNode(struct node *start,int data)
{
        struct node *tmp;
        if(start==NULL)
        {
                printf("List is empty\n");
                return start;
        }
        if(start->next==NULL)    /*only one node in the list*/
                if(start->info==data)
                {
                        tmp=start;
                        start=NULL;
                        free(tmp);
                        return start;
                }
                else
                {
                        printf("Element %d not found\n",data);
                        return start;
```

```
        }
/*Deletion of first node*/
if(start->info==data)
{
        tmp=start;
        start=start->next;
        start->prev=NULL;
        free(tmp);
        return start;
}
/*Deletion in between*/
tmp=start->next;
while(tmp->next!=NULL )
{
        if(tmp->info==data)
        {
                tmp->prev->next=tmp->next;
                tmp->next->prev=tmp->prev;
                free(tmp);
                return start;
        }
        tmp=tmp->next;
}
/*Deletion of last node*/
if(tmp->info==data)
{
        tmp->prev->next=NULL;
```

```
                free(tmp);
                return start;
        }
        printf("Element %d not found\n",data);
        return start;
}/*End of deleteNode()*/
```

OUTPUT:

```
1.Create List
2.Display
3.Add to empty list
4.Add at beginning
5.Add at end
6.Add after
7.Add before
8.Delete
9.Quit
Enter your choice : 1
Enter the number of nodes : 5
Enter the element to be inserted : 10
Enter the element to be inserted : 20
Enter the element to be inserted : 30
Enter the element to be inserted : 40
Enter the element to be inserted : 50
1.Create List
```

2.Display

3.Add to empty list

4.Add at beginning

5.Add at end

6.Add after

7.Add before

8.Delete

9.Quit

Enter your choice : 2

List is :

10 20 30 40 50

1.Create List

2.Display

3.Add to empty list

4.Add at beginning

5.Add at end

6.Add after

7.Add before

8.Delete

9.Quit

Enter your choice : 3

Enter the element to be inserted : 80

1.Create List

2.Display

3.Add to empty list

4.Add at beginning

5.Add at end

6.Add after

7.Add before

8.Delete

9.Quit

Enter your choice : 2

List is :

80

1.Create List

2.Display

3.Add to empty list

4.Add at beginning

5.Add at end

6.Add after

7.Add before

8.Delete

9.Quit

Enter your choice : 4

Enter the element to be inserted : 70

1.Create List

2.Display

3.Add to empty list

4.Add at beginning

5.Add at end

6.Add after

7.Add before

8.Delete

9.Quit

Enter your choice : 2

List is :

70 80

1.Create List

2.Display

3.Add to empty list

4.Add at beginning

5.Add at end

6.Add after

7.Add before

8.Delete

9.Quit

Enter your choice : 6

Enter the element to be inserted : 90

Enter the element after which to insert : 80

1.Create List

2.Display

3.Add to empty list

4.Add at beginning

5.Add at end

6.Add after

7.Add before

8.Delete

9.Quit

Enter your choice : 2

List is :

70 80 90

1.Create List

2.Display

3.Add to empty list

4.Add at beginning

5.Add at end

6.Add after

7.Add before

8.Delete

9.Quit

Enter your choice : 7

Enter the element to be inserted : 60

Enter the element before which to insert : 70

1.Create List

2.Display

3.Add to empty list

4.Add at beginning

5.Add at end

6.Add after

7.Add before

8.Delete

9.Quit

Enter your choice : 2

List is :

60 70 80 90

1.Create List

2.Display

3.Add to empty list

4.Add at beginning

5.Add at end

6.Add after

7.Add before

8.Delete

9.Quit

Enter your choice : 8

Enter the element to be deleted : 90

1.Create List

2.Display

3.Add to empty list

4.Add at beginning

5.Add at end

6.Add after

7.Add before

8.Delete

9.Quit

Enter your choice : 2

List is :

60 70 80

1.Create List

2.Display

3.Add to empty list

4.Add at beginning

5.Add at end

6.Add after

7.Add before

8.Delete

9.Quit

Enter your choice : 9

...Program finished with exit code 1

Press ENTER to exit console.

Experiment 8: Implement basic operations (insertion, deletion, search, find min and find max) on Binary Search trees

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
      struct node *lchild;
      int info;
      struct node *rchild;
};

struct node *insert(struct node *ptr, int ikey);
struct node *Min(struct node *ptr);
struct node *Max(struct node *ptr);
void display(struct node *ptr,int level);
struct node *search(struct node *p, int x);
int main( )
    {
     struct node *root=NULL,*ptr;
    int choice,k,x;

     while(1)
     {
           printf("\n");
           printf("1.Insert\n");
           printf("2.Display\n");
           printf("3.Find minimum and maximum\n");
           printf("4.Search\n");
           printf("5.Quit\n");
           printf("\nEnter your choice : ");
```

```
        scanf("%d",&choice);

        switch(choice)
        {
        case 1:
            printf("\nEnter the key to be inserted : ");
            scanf("%d",&k);
            root = insert(root, k);
            break;

    case 2:
      printf("\n");
      display(root,0);
      printf("\n");
      break;

        case 3:
            ptr = Min(root);
            if(ptr!=NULL)
                printf("\nMinimum key is %d\n", ptr->info );
            ptr = Max(root);
            if(ptr!=NULL)
                printf("\nMaximum key is %d\n", ptr->info );
            break;

case 4:
printf("Enter the key to be searched : ");
        scanf("%d",&x);
        ptr = search(root, x);
        if(ptr == NULL)
```

```
            printf("Key not found!!\n");
            else
            printf("key found.\n");
                                        break;



 case 5:
       exit(1);

 default:
          printf("\nWrong choice\n");
             }/*End of switch */
       }/*End of while */

       return 0;

}/*End of main( )*/



struct node *insert(struct node *ptr, int ikey )
{
       if(ptr==NULL)
       {
             ptr = (struct node *) malloc(sizeof(struct node));
             ptr->info = ikey;
             ptr->lchild = NULL;
             ptr->rchild = NULL;
       }
       else if(ikey < ptr->info) /*Insertion in left subtree*/
```

```
        ptr->lchild = insert(ptr->lchild, ikey);
    else if(ikey > ptr->info) /*Insertion in right subtree */
        ptr->rchild = insert(ptr->rchild, ikey);
    else
        printf("\nDuplicate key\n");
    return ptr;
}/*End of insert( )*/



struct node *Min(struct node *ptr)
{
    if(ptr==NULL)
        return NULL;
    else if(ptr->lchild==NULL)
    return ptr;
    else
        return Min(ptr->lchild);
}/*End of min()*/

struct node *Max(struct node *ptr)
{
    if(ptr==NULL)
        return NULL;
    else if(ptr->rchild==NULL)
    return ptr;
    else
        return Max(ptr->rchild);
}/*End of max()*/
```

```
void display(struct node *ptr,int level)
{
      int i;
      if(ptr == NULL )/*Base Case*/
            return;
      else
   {
            display(ptr->rchild, level+1);
            printf("\n");
            for (i=0; i<level; i++)
                  printf("    ");
            printf("%d", ptr->info);
            display(ptr->lchild, level+1);
      }
}/*End of display()*/

struct node *search(struct node *ptr, int x)
          {
          if(ptr == NULL)
          {
                              printf("key not found\n");
                              return NULL;
          }
          else if(x < ptr->info)/*search in left subtree*/
                              return search(ptr->lchild, x);
          else if(x > ptr->info)/*search in right subtree*/
                              return search(ptr->rchild, x);
                    else /*key found*/
                              return ptr;
          }/*End of search()*
```

OUTPUT:

1.Insert

2.Display

3.Find minimum and maximum

4.Search

5.Quit

Enter your choice : 1

Enter the key to be inserted : 6

1.Insert

2.Display

3.Find minimum and maximum

4.Search

5.Quit

Enter your choice : 1

Enter the key to be inserted : 8

1.Insert

2.Display

3.Find minimum and maximum

4.Search

5.Quit

Enter your choice : 1

Enter the key to be inserted : 7

1.Insert

2.Display

3.Find minimum and maximum

4.Search

5.Quit

Enter your choice : 1

Enter the key to be inserted : 9

1.Insert

2.Display

3.Find minimum and maximum

4.Search

5.Quit

Enter your choice : 1

Enter the key to be inserted : 3

1.Insert

2.Display

3.Find minimum and maximum

4.Search

5.Quit

Enter your choice : 1

Enter the key to be inserted : 4

1.Insert

2.Display

3.Find minimum and maximum

4.Search

5.Quit

Enter your choice : 2

```
      9
   8
      7
6
      4
   3
```

1.Insert

2.Display

3.Find minimum and maximum

4.Search

5.Quit

Enter your choice : 3

Minimum key is 3

Maximum key is 9

```
1.Insert

2.Display

3.Find minimum and maximum

4.Search

5.Quit


Enter your choice : 4

Enter the key to be searched : 7

key found.


1.Insert

2.Display

3.Find minimum and maximum

4.Search

5.Quit


Enter your choice : 4

Enter the key to be searched : 10

key not found

Key not found!!


1.Insert

2.Display

3.Find minimum and maximum

4.Search

5.Quit


Enter your choice : 5

...Program finished with exit code 1

Press ENTER to exit console.
```

Experiment 9: Implementation of Heaps.

```
#include<stdio.h>

#include<conio.h>


int ch,i,heap[50],n,k,left,right,target,temp,parent; void main()

{

printf("\n----------HEAP OPERATIONS ");

printf("\n[1].BUILD_HEAP\n[2].INSERT\n[3].DELETEMIN\n[4].DISPLAY\n[0].EXIT");

do

{

printf("\nChoose Operation:"); scanf("%d",&ch);

switch(ch)

{

case 1:

build_heap();

display();

break;

case 2: insert(); break;

case 3: deletemin(); break;

case 4: display(); break;

}

}while(ch!=0);

}

build_heap()

{

printf("\nEnter No of Elements in Heap:"); scanf("%d",&n);

printf("\nEnter %d elements:\n",n); for(i=1;i<=n;i++) scanf("%d",&heap[i]);

for(i=n/2;i>0;i--) percolate_down(i);

}

insert()
```

```
{
printf("Enter Element:"); scanf("%d",&temp); n++;
heap[n]=temp; printf("\nElement Inserted"); percolate_up(n);
}
deletemin()
{
if(n>0)
{
printf("\nElement Deleted Is:%d",heap[1]); heap[1]=heap[n];
heap[n]=0; n--;
percolate_down(1);
}
else printf("\nHeap Is Empty");
}
display()
{
if(n>0)
{
printf("\nThe Min-Heap is:\n"); i=1;
while(i<=n)
{
for(k=i;k<(2*i) && k<=n;k++) printf("%d ",heap[k]);
printf("\n"); i=k;
}
}
else printf("\nHeap is Empty");
}
percolate_up(int m)
{
while(m>1)
```

```
{
parent=m/2; if(heap[parent] > heap[m])
{
temp=heap[m]; heap[m]=heap[parent]; heap[parent]=temp; m=parent;
}
else break;
}
}
percolate_down(int m)
{
while(2*m <= n)
{
left=2*m; right=2*m+1;
if(right <=n && heap[right] < heap[left]) target=right;
else
target=left; if(heap[target] < heap[m])
{
temp=heap[m]; heap[m]=heap[target]; heap[target]=temp; m=target;
}
else break;
}
}
```

OUTPUT:

```
----------HEAP OPERATIONS
[1].BUILD_HEAP
[2].INSERT
[3].DELETEMIN
[4].DISPLAY
[0].EXIT
```

Choose Operation:1

Enter No of Elements in Heap:5

Enter 5 elements:

10

20

30

40

50

The Min-Heap is:

10

20 30

40 50

Choose Operation:2

Enter Element:60

Element Inserted

Choose Operation:4

The Min-Heap is:

10

20 30

40 50 60

Choose Operation:3

Element Deleted Is:10

Choose Operation:4

The Min-Heap is:

20

40 30

60 50

Choose Operation:0

...Program finished with exit code 0

Press ENTER to exit console.

Experiment 10: Implementation of Breadth First Search Techniques

```
#include<stdio.h>

#include<conio.h>

int am[10][10],n,v,visited[10],q[20],i,j,front=0,rear=0;

void main()

{

printf("Enter Number of Vertices:");

scanf("%d",&n);

printf("\nEnter Adjacency Matrix:");

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

scanf("%d",&am[i][j]);

visited[i]=0;

}

printf("Enter Starting Node:");

scanf("%d",&v);

bfs(v);

getch();

}

bfs(int st)

{

q[rear]=st; rear++;

while(front<rear)

{

v=q[front]; front++;

if(visited[v]==0)

{

visited[v]=1;

 printf("-->%d",v);

}
```

```
for(i=1;i<=n;i++)
{
if(am[v][i]==1 && visited[i]==0)
{
if(inqueue(i)==0)
{
q[rear]=i; rear++;
}   }     }  }
}
int inqueue(int n)
{
for(j=front;j<=rear;j++)
{
if(q[j]==n) return 1;
}
return 0;
}
```

OUTPUT:

```
Enter Number of Vertices:5
Enter Adjacency Matrix:1 0 0 1 1
1 1 1 0 1
1 0 0 0 1
1 1 1 1 1
1 0 0 0 0
Enter Starting Node:1
-->1-->4-->5-->2-->3


...Program finished with exit code 255
Press ENTER to exit console.
```

Experiment 11: Implementation of Depth First Search Techniques.

```
#include<stdio.h>
#include<conio.h>
int am[10][10],n,v,visited[10],stk[10]; int i,j,top=-1;
void main()
{
printf("Enter Number of Vertices:");
scanf("%d",&n);
printf("\nEnter Adjacency Matrix:");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
scanf("%d",&am[i][j]);
visited[i]=0;
}
printf("Enter Starting Node:");
scanf("%d",&v);
dfs(v);
}
dfs(int st)
{
top++; stk[top]=st;
while(top>=0)
{
v=stk[top]; top--;
if(visited[v]==0)
{
visited[v]=1;
printf("-->%d",v);
}
```

```
for(i=n;i>=1;i--)
{
if(am[v][i]==1 && visited[i]==0)
{
if(instack(i)==0)
{
top++; stk[top]=i;
}
}
}
}
}

int instack(int n)
{
for(j=0;j<=top;j++)
{
if(stk[j]==n) return 1;
}
return 0;
}
```

OUTPUT

```
Enter Number of Vertices:5

Enter Adjacency Matrix:1 1 1 0 0
0 0 0 0 1
1 0 1 0 1
1 1 1 1 1
```

0 0 0 1 1

Enter Starting Node:1

-->1-->2-->5-->4-->3

...Program finished with exit code 255

Press ENTER to exit console.

Experiment 12: Implementation of Prim's algorithm.

```
#include<stdio.h>

#include<conio.h>

int a,b,n,i,j,k,ne;

int min,mincost=0,cost[10][10],V[10];

void main()

{

printf("\n Enter the number of nodes:");

scanf("%d",&n);

printf("\n Enter the adjacency matrix:\n");

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

scanf("%d",&cost[i][j]);

if(cost[i][j]==0)

cost[i][j]=999;

}

}

V[1]=1;

ne=1;

while(ne<n)

{

min=999;

for(i=1;i<=n;i++)

{

if(inset(i)==1)

{

for(j=1;j<=n;j++)

{
```

```
if(inset(j)==0 && cost[i][j]<min)
{
min=cost[i][j]; a=i;
b=j;
}
}
}
}
printf("\n Edge %d:(%d %d) cost:%d",ne,a,b,min); ne++;
V[ne]=b; mincost=mincost+min; cost[a][b]=cost[b][a]=999;
}
printf("\n Minimun cost=%d",mincost);
getch();
}
int inset(int k)
{
int m=1; while(m<=ne)
{
if(V[m]==k)
return 1;
else
m++;
}
return 0;
}
```

OUTPUT

```
Enter the number of nodes:6

 Enter the adjacency matrix:
```

0 6 1 5 999 999

6 0 5 999 63 7

1 5 0 333 6 9

999 5 12 567 0 7

1 23 67 999 54 3

9 76 4 34 888 5

Edge 1:(1 3) cost:1

Edge 2:(1 4) cost:5

Edge 3:(3 2) cost:5

Edge 4:(3 5) cost:6

Edge 5:(5 6) cost:3

Minimun cost=20

...Program finished with exit code 255

Press ENTER to exit console.

Experiment 13: Implementation of Kruskal's Algorithm.

```
#include<stdio.h>

#include<conio.h>

int i,j,k,u,v,n,ne=1,a,b;

int min,mincost=0,cost[9][9],set[9];

void main()

{

printf("\n\tImplementation of Kruskal's algorithm\n");

printf("\nEnter the no. of vertices:");

scanf("%d",&n);

printf("\nEnter the cost adjacency matrix:\n");

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

scanf("%d",&cost[i][j]);

if(cost[i][j]==0)

cost[i][j]=999;

}

}

for(i=1;i<=n;i++) set[i]=i;

printf("The edges of Minimum Cost Spanning Tree are\n");

while(ne<n)

{

min=999; for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

if(cost[i][j] < min)

{
```

```
min=cost[i][j]; a=i;
b=j;
}
}
}
u=find(a); v=find(b); if(u!=v)
{
uni(u,v);
printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
mincost=mincost+min;
}
cost[a][b]=cost[b][a]=999;
}
printf("\n\tMinimum cost = %d\n",mincost);
getch();
}
int find(int i)
{
i=set[i]; return i;
}
int uni(int i,int j)
{
set[j]=i; for(k=1;k<=n;k++)
{
if(set[k]==j)
set[k]=i;
}
}
```

OUTPUT:

Implementation of Kruskal's algorithm

Enter the no. of vertices:6

Enter the cost adjacency matrix:

1 6 45 888 6 45

3 888 9 65 90 2

2 56 999 6 5 3

1 1 1 1 1 999

5 45 999 6 5 34

9 78 56 4 2 999

The edges of Minimum Cost Spanning Tree are

1 edge (4,1) =1

2 edge (4,2) =1

3 edge (4,3) =1

4 edge (4,5) =1

5 edge (2,6) =2

Minimum cost = 6

...Program finished with exit code 255

Press ENTER to exit console.

Experiment 14: Implementation of Linear search.

```
#include <stdio.h>

int main()

{

   int a[10], i, item,n;

   printf("\nEnter number of elements of an array:\n");

   scanf("%d",&n);

   printf("\nEnter elements: \n");

   for (i=0; i<n; i++)

      scanf("%d", &a[i]);

   printf("\nEnter item to search: ");

   scanf("%d", &item);

   for (i=0; i<=9; i++)

      if (item == a[i])

      {

         printf("\nItem found at location %d", i+1);

         break;

      }

   if (i > 9)

      printf("\nItem does not exist.");

   return 0;

}
```

OUTPUT

```
Enter number of elements of an array:

5


Enter elements:

54
```

7

39

8

90

Enter item to search: 7

Item found at location 2

...Program finished with exit code 0

Press ENTER to exit console.

Experiment 15: Implementation of Fibanocci search.

```
#include<stdio.h>

#define min(a,b) a<b?a:b

int main()

{

int n,ar[10],x,i,a,b,c,offset;


printf("\nEnter number of elements of an array:\n");

   scanf("%d",&n);

printf("Enter the elements of the array (in sorted order)");

   for (i=0; i<n; i++)

      scanf("%d", &ar[i]);

   printf("\nEnter item to search: ");

   scanf("%d", &x);

a=0;

b=1;

c=a+b;

while(c<n)

{

a=b;

b=c;

c=a+b;

}

offset= -1;

while(c>1)

{

i=min(offset+a, n-1);

if(ar[i] < x)

{

c=b;
```

```
b=a;
a=c-b;
offset= i;
}

else if(ar[i] > x)
{
c=a;
b=b-a;
a=c-b;
}
else
{
printf("Element %d is found at index %d", x,i);
return i;
}
}
if(b && ar[offset+1] == x)
{
printf("Element %d  is found at index %d " , x,offset);
return -1;
}
printf("Element not found");
}
```

OUTPUT:

```
Enter the elements of the array (in sorted order)
Segmentation fault (core dumped)
Enter number of elements of an array:
```

5

Enter the elements of the array (in sorted order)

10

20

30

40

60

Enter item to search: 40

Element 40 is found at index 3

Enter number of elements of an array:

5

Enter the elements of the array (in sorted order)

10

20

30

40

50

Enter item to search: 15

Element not found

Experiment 16: Implementation of Merge sort.

```
#include<stdio.h>
#include <conio.h>
#define MAX 10
void merge(int array[], int low, int mid, int high );
void merge_sort( int array[],int low, int high );
void merge(int array[], int low,int mid,int high )
{
        int temp[MAX];
        int i = low;
        int j = mid +1;
        int k = low ;
        while((i <= mid) && (j <= high))
        {
                if(array[i] <= array[j])
                        temp[k++] = array[i++];
                else
                        temp[k++] = array[j++];
        }
        while( i <= mid )
                temp[k++]=array[i++];
        while( j <= high )
                temp[k++]=array[j++];
        for(i= low; i <= high ; i++)
                array[i]=temp[i];
}
void merge_sort(int arr[],int low, int high )
{
        int mid;
        if( low != high )
        {
                mid = (low+high)/2;
                merge_sort(arr,low , mid );
                merge_sort(arr,mid+1, high );
                merge(arr,low, mid, high );
        }
```

```
}/*End of merge_sort*/
int main()
{
        int i,n,array[MAX];
        printf("Enter the number of elements : ");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("Enter element %d : ",i+1);
                scanf("%d",&array[i]);
        }
        printf("Unsorted list is :\n");
        for( i = 0 ; i<n ; i++)
                printf("%d ", array[i]);
        merge_sort(array,0, n-1);
        printf("\nSorted list is :\n");
        for( i = 0 ; i<n ; i++)
                printf("%d ", array[i]);
        printf("\n");
        return 0;
}/*End of main()*/
```

Output:

```
Enter no. of elements: 5
Enter element 1:  6
Enter element 1:  2
Enter element 1:  4
Enter element 1:  8
Enter element 1:  5

Unsorted list is:
6  2  4  8  5
Un sorted list is:
2  4  5  6  8
```

Experiment 17: Implementation of Quick sort.

```
#include<stdio.h>
int partition(int a[25],int beg,int end)
{
int left, right, pivot, temp;
pivot=beg;
left=beg;
right=end;
while(left<right)
{
        while(a[left]<=a[pivot]&& left<end)
                left++;
        while(a[right]>a[pivot])
                right--;
        if(left<right)
        {
                temp=a[left];
                a[left]=a[right];
                a[right]=temp;
        }
}
temp=a[pivot];
        a[pivot]=a[right];
        a[right]=temp;
        return right;
}
void quicksort(int a[25],int beg, int end)
{
        int j;
        if(beg<end)
        {
                j=partition(a,beg,end);
                quicksort(a,beg,j-1);
                quicksort(a,j+1,end);
        }
}
int main()
```

```
{
        int i, n, arr[25];
        printf("Enter no. of elements : ");
        scanf("%d",&n);
        printf("Enter %d elements: ",n);
        for(i=0;i<n;i++)
        scanf("%d",&arr[i]);
        quicksort(arr,0,n-1);
        printf("The Sorted Order is: ");
        for(i=0;i<n;i++)
        printf(" %d",arr[i]);
        return 0;
}
```

Output:

Enter no. of elements: 8
Enter 8 elements: 40 20 10 80 60 50 7 30
The sorted order is: 7 10 20 30 40 50 60 80