NEW HORIZON
COLLEGE OF ENGINEERING
Autonomous College Permanently Affiliated to VTU, Approved by AICTE & UGC
Accredited by NAAC with 'A' Grade, Accredited by NBA

DEPARTMENT OF MCA
II SEMESTER

Data Structures Using C++ Laboratory (20MCAL28)

VINAYAKA D
1NH21MC107

COURSE COORDINATOR
Dr. B Nithya Ramesh

2021-22

NEW HORIZON
COLLEGE OF ENGINEERING
Autonomous College Permanently Affiliated to VTU, Approved by AICTE & UGC
Accredited by NAAC with 'A' Grade, Accredited by NBA

DEPARTMENT OF MCA

CERTIFICATE

This is to certify **VINAYAKA D** bearing **USN No. 1NH21MC107** of **MCA II SEMESTER** has satisfactorily completed the Laboratory Programs in **Data Structures Using C++ Lab,** Subject Code – **20MCAL28** as prescribed by the Syllabus during the year **2021-22.**

--------------------------------------	--
Signature of Course Co-ordinator	Signature of Head of the Department
-----------------------------------	---
Internal Examiner	External Examiner

Date of Examination:

INDEX

1. Example programs on arrays:

a) Write a C++ program to find the largest element of a given array of integers.

```
#include<iostream>
using namespace std;
int main()
{
int a[10],n,i;
cout<<"Enter Array Size:";
cin>>n;
cout<<"Enter Array Input:";
for(i=0;i<n;i++)
cin>>a[i];
int largest=a[0];
cout<<"Output..\n";
for(i=1;i<n;i++)
if(a[i]>largest)
largest=a[i];
cout<<"Largest Number:"<<largest;
return 0;
}
```

Output:

```
ENTER ARRAY SIZE... 5

ENTER ARRAY INPUT... 4 7 3 56 2

OUTPUT...

Largest number= 56
```

b) C++ program to sort an array in Ascending Order.

```
#include<iostream>
using namespace std;
int main()
{
```

```
int arr[10];
int size,i,j,temp;
cout<<"Enter size of array:";
cin>>size;
cout<<"Enter element in array";
for(i=0;i<size;i++)
cin>>arr[i];
for(i=1;i<size;i++)
{
for(j=0;j<size-1;j++)
{
if(arr[j]>arr[j+1])
{
temp=arr[j];
arr[j]=arr[j+1];
arr[j+1]=temp;
}
}
}
cout<<"Elements of array in sorted ascending order:"<<endl;
for(i=0;i<size;i++)
{
cout<<arr[i]<<endl;
}
return 0;
}
```

Output:

```
Enter size of array: 6

Enter elements in array: 7 3 0 -8 4 1

Elements of array in sorted ascending order:
-8
0
1
3
4
7
```

c) C++ Program to Add Two Matrix Using Multi-dimensional Arrays

```
#include<iostream>
using namespace std;
int main()
{

int r,c,a[10][10],b[10][10],sum[10][10],i,j;
cout<<"Enter no of rows :";
cin>>r;
cout<<"Enter no of cols:";
cin>>c;
cout<<"Enter elements of 1st matrix:"<<endl;
for(i=0;i<r;i++)
for(j=0;j<c;j++)
{
cin>>a[i][j];
}
cout<<"Enter elements of 2nd matrix:"<<endl;
for(i=0;i<r;i++)
for(j=0;j<c;j++)
{
cin>>b[i][j];
}
for(i=0;i<r;i++)
for(j=0;j<c;j++)
sum[i][j]=a[i][j]+b[i][j];
cout<<endl<<"Sum of 2 matrix is:"<<endl;
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
cout<<sum[i][j]<<" ";
}
cout<<endl;
}
return 0;
}
```

Output:

```
Enter number of rows: 3

Enter number of columns: 3

Enter elements of 1st matrix:

4 5 1
6 7 1
4 2 1

Enter elements of 2nd matrix:

0 9 3
2 5 1
6 3 4

Sum of two matrix is:

4  14  4
8  12  2
10  5  5
```

2. C++ program on String operations.

```
#include<iostream>
#include<cstring>
using namespace std;
int main()
{
char str1[10]="Hello";
char str2[10]="World";
char str3[10];
int len;
strcpy(str3,str1);
cout<<"strcpy(str3,str1):"<<str3<<endl;
strcat(str1,str2);
cout<<"strcat(str1,str2):"<<str1<<endl;
len=strlen(str1);
cout<<"strlen(str1):"<<len<<endl;
cout<<"strcmp(str1,str2):"<<strcmp(str1,str2)<<endl;
cout<<"strchr(str1,'e'):"<<strchr(str1,'e')<<endl;
return 0;
}
```

Output:

```
strcpy( str3, str1) : Hello

strcat( str1, str2): HelloWorld

strlen(str1) : 10

strcmp(str1, str2): 72

strchr(str1, 'e'): elloWorld
```

3. Define a STUDENT class with USN, Name, and Marks in 3 tests of a subject. Declare an array of 10 STUDENT objects. Using appropriate functions, find the average of the two better marks for each student. Print the USN, Name and the average marks of all the students.

```
#include<iostream>
using namespace std;
class student
{
char usn[10];
char name[10];
float m1,m2,m3,avg;
public:
void read_data();
void compute();
void write_data();
};
void student::read_data()
{
cout<<"Enter the USN:";
cin>>usn;
cout<<"Enter the Name:";
cin>>name;
cout<<"Enter 3 tests marks of a subject:";
cin>>m1>>m2>>m3;
}
void student::compute()
{
if(m1<m2 && m1<m3)
avg=(m2+m3)/2;
else if(m2<m3)
avg=(m1+m3)/2;
else
avg=(m1+m2)/2;
}
void student::write_data()
{
cout<<"Name:"<<name<<endl;
cout<<"m1:"<<m1<<endl;
cout<<"m2:"<<m2<<endl;
cout<<"m3:"<<m3<<endl;
cout<<"USN:"<<usn<<endl;
cout<<"Avg:"<<avg<<endl;
}
```

```
int main()
{
student s[10];
int n,i;
cout<<"Enter the number of students:";
cin>>n;
for(i=0;i<n;i++)
s[i].read_data();
for(i=0;i<n;i++)
s[i].compute();
for(i=0;i<n;i++)
s[i].write_data();
return 0;
}
```

Output:

```
enter the number of students:
2

enter the usn:
101

enter the name:
Hari

enter the marks of 3 subjects:
15 18 21

enter the usn:
102

enter the name:
Vino

enter the marks of 3 subjects:
18 20 22

name is:Hari

 m1 is:15

 m2 is:18

 m3 is:21
```

```
avg. of best of two subjects:19.5

name is:Vino

m1 is:18

m2 is:20

m3 is:22

avg. of best of two subjects:21
```

4. Write a C++ program that uses stack operations to convert a given infix expression into its postfix equivalent, Implement the stack using an array.

```
#include<iostream>
using namespace std;
//FUNCTION TO RETURN PRECEDENCE OF OPERATORS
int prec(char c)
{
    if(c=='^'||c=='$')
    return 3;
    else if(c=='*'||c=='/')
    return 2;
    else if(c=='+'||c=='-')
    return 1;
    else
    return -1; //#
}
// FUNCTION TO CONVERT INFIX TO POSTFIX EXPRESSION
void infixToPostfix(string s)
{
   char st[20];
   int top=-1;
   string result;
   for(int i=0;i<s.length();i++)
   {
     char c=s[i];
// IF SCANNED CHARACTER IS AN OPERAND ADD TO OUTPUT STRING
    if((c>='a'&&c<='z')||(c>='A'&&c<='Z')||(c>='0'&&c<='9'))
    result=result+c;
// IF SCANNED CHARACTER IS AN '(' PUSH TO STACK
    else if(c=='(')
    st[++top]='(';
// IF SCANNED CHARACTER IS AN ')' POP ALL SYMBOLS
    else if(c==')')
      {
       while(top!=-1 && st[top]!='(')
      {
       char temp=st[top];
       st[top--];
       result=result+temp;
      }
       st[top--];
      }
// IF AN OPERATOR IS SCANNED
```

```
    else
      {
        while(top!=-1 && prec(s[i])<=prec(st[top]))
          {
            char temp=st[top];
            st[top--];
            result=result+temp;
          }
          st[++top]=c;
          }
       }
// POP ALL REMAINING ELEMENTS FROM STACK
        while(top!=-1)
     {
       char temp=st[top];
       st[top--];
       result=result+temp;
     }
       cout<<result<<endl;
}
// MAIN PROGRAM TO TEST ABOVE FUNCTIONS
int main()
{
   string exp;
   cout<<"Enter an Infix Expression:"<<endl;
   cin>>exp;
   cout<<"Postfix Expression:"<<endl;
   infixToPostfix(exp);
   return 0;
}
```

Output:

```
Enter an infix expression: a+b*(c^d-e)^(f+g*h)-i

Postfix expression: abcd^e-fgh*+^*+i-
```

5. Using recursion,

a) Solving Towers of Hanoi Problem
b) Finding factorial of a given number
c) Calculation of GCD and LCM of 3 integer Numbers

```
#include<iostream>
#include<math.h>
#include<stdlib.h>
using namespace std;
void hanoi(int d,char source,char temp,char dest)
{
if(d==1)
{
cout<<" Move Disk "<< d <<" from "<< source <<" to "<< dest <<endl;
}
else
{
hanoi(d-1,source,dest,temp);
cout<<"\n Move Disk "<< d <<" from "<< source <<" to "<< dest <<endl;
hanoi(d-1,temp,source,dest);
}
}
int factorial(int n)
{
if(n<0)
return(-1);
if(n==0)
return(1);
else
{
return(n*factorial(n-1));
}
}
int gcd(int m,int n)
{
int r;
if(m<n)
return gcd(n,m);
r=m%n;
if(r==0)
return n;
else
return(gcd(n,r));
}
```

```
int lcm(int a,int b,int n)
{
if(n%a==0 && n%b==0)
return n;
else
return(lcm(a,b,n+1));
}
int main()
{
int g,l,x,y,z,n,ch,disk,move,fact,value;
while(1)
{
cout<<"\n 1.TOWERS OF HANOI";
cout<<"\n 2.FACTORIAL OF A GIVEN NUMBER";
cout<<"\n 3.GCD & LCM";
cout<<"\n ENTER YOUR CHOICE :";
cin>>ch;
switch(ch)
{
case 1: {
        cout<<"ENTER THE NO OF DISK:";
        cin>>disk;
        move=(pow(2,disk)-1);
          cout<<"NO OF MOVES ARE:"<<move<<endl;
          hanoi(disk,'A','B','C');
          break;
          }
case 2: {
        cout<<"ENTER THE NUMBER:";
        cin>>value;
        fact=factorial(value);
          cout<<"FACTORIAL OF A NUMBER:"<<fact<<endl;
          break;
          }
case 3: {
        cout<<"ENTER THREE INTEGERS:";
        cin>>x>>y>>z;
          n=1;
        g=gcd(gcd(x,y),z);
          l=lcm(lcm(x,y,n),z,n);
          cout<<"GCD IS:"<<g<<"\n LCM IS:"<<l;
          break;
          }
case 4:{
          exit(0);
          break;
        }
default:cout<<"\n INVALID CHOICE...";
}
}
return 0;
}
```

Output:

```
1.Tower of Hanoi
2.Factorial of a given number
3.GCD & LCM
4.Exit

Enter your choice...1

Enter the no of disk:3

No of moves are 7Move disk 1from Ato C
Move disk2from Ato BMove disk 1from Cto B
Move disk3from Ato CMove disk 1from Bto A
Move disk2from Bto CMove disk 1from Ato C

1.Tower of Hanoi
2.Factorial of a given number
3.GCD & LCM
4.Exit

Enter your choice...2

Enter a number: 5

Factorial of a number is:120

1.Tower of Hanoi
2.Factorial of a given number
3.GCD & LCM
4.Exit

Enter your choice...3

Enter Three integers: 42 24 32

GCD is: 2
LCM is: 672
```

```
1.Tower of Hanoi
2.Factorial of a given number
3.GCD & LCM
4.Exit

Enter your choice...4
```

6. Simulating the working of linear queue.

```
#include<iostream>
using namespace std;
int queue[5],n=5,front=-1,rear=-1;
void Insert()
{
int val;
if(rear == n-1)
cout<<"QUEUE OVERFLOW"<<endl;
else {
if(front == -1)
front=0;
cout<<"ENTER THE ELEMENT TO BE INSERTED IN QUEUE:"<<endl;
cin>>val;
rear++;
queue[rear]=val;
}
}
void Delete()
{
if(front == -1 || front > rear)
{
cout<<"QUEUE UNDERFLOW"<<endl;
return;
}
else {
cout<<"ELEMENTS DELETED FROM QUEUE IS:"<<queue[front]<<endl;
front++;
}
}
void Display()
{
if(front == -1 || front > rear)
cout<<"QUEUE IS EMPTY"<<endl;
else {
cout<<"QUEUE ELEMENTS ARE:";
for(int i=front;i<=rear;i++)
cout<<queue[i]<<" ";
cout<<endl;
}
}
int main()
{
int ch;
cout<<"1) INSERT ELEMENTS TO QUEUE"<<endl;
cout<<"2) DELETE ELEMENTS FROM QUEUE"<<endl;
cout<<"3) DISPLAY ALL THE ELEMENTS OF QUEUE"<<endl;
cout<<"4) EXIT "<<endl;
do {
cout<<"ENTER YOUR CHOICE:"<<endl;
cin>>ch;
switch(ch)
{
case 1:Insert();
break;
case 2:Delete();
break;
case 3:Display();
break;
case 4:cout<<" EXIT "<<endl;
break;
default : cout<<"INVALID CHOICE"<<endl;
}
} while(ch!=4);
return 0;
}
```

Output:

```
1) Insert element to queue
2) Delete element from queue
3) Display all the elements of queue
4) Exit

Enter your choice : 1

Enter the element to be inserted in queue :
3

Enter your choice : 1

Enter the element to be inserted in queue :
7

Enter your choice : 1

Enter the element to be inserted in queue :
5
```

```
Enter your choice : 3

Queue elements are : 3 7 5

Enter your choice : 2

Element deleted from queue is : 3

Enter your choice : 3

Queue elements are : 7 5

Enter your choice : 2

Element deleted from queue is : 7

Enter your choice : 2

Element deleted from queue is : 5

Enter your choice : 2

Queue Underflow

Enter your choice : 4

Exit
```

7. Simulating the working of circular queue.

```
#include<iostream>
using namespace std;
int MAX=3;
int cq[4], rear=0, front=1, count=0;
void cqins()
{
int elt;
if (count==MAX)
{
cout<<"\n cq is full";
}

else
{
cout<<"\n enter the element to insert";
cin>>elt;
rear=rear%MAX+1;
cq[rear]=elt;
count++;
}
}
void cqdel()
{
if (count==0)
{cout<<"\n cq is empty";
}
else
{
cout<<"\n delete item is "<<cq[front];
front=front%MAX+1;
count--;
}
}
void cqdis()
{
int i,j;
i=front;
if(count==0)
{
cout<<"\nempty";
}
else
{
```

```
cout<<"\n cq elements are :";
for(j=1; j<=count; j++)
{
cout<<"\n"<<cq[i];
i=i%MAX+1;
}

cout<<"\n front of the cq: "<< cq[front];
cout<<"\n rear of the cq:"<<cq[rear];
}}
int main()
{
int ch;
for (;;)
{
cout<<"\n cq operation";
cout<<"\n 1. Insert";
cout<<"\n 2. delete";
cout<<"\n 3. display";
cout<<"\n 4. exit";
cout<<"\n enter your choice";
cin>>ch;
switch(ch)
{
case 1: cqins();
break;
case 2: cqdel();
break;
case 3: cqdis();
break;
case 4: cout<<"exit";
break;
default: cout<<"\n invalid choice";
}
}
return 0;
}
```

Output:

```
1)Insert
2)Delete
3)Display
4)Exit
```

```
Enter choice :
1

Input for insertion:
4

Enter choice :
1

Input for insertion:
6

Enter choice :
1

Input for insertion:
7

Enter choice :
1

Input for insertion:
8

Enter choice :
3

Queue elements are :
4 6 7 8

Enter choice :
2

Element deleted from queue is : 4

Enter choice :
3

Queue elements are :
6 7 8

Enter choice :
2

Element deleted from queue is : 6
```

```
Enter choice :
3

Queue elements are :
7 8

Enter choice :
4

Exit
```

8. Write a C++ program that uses functions to perform the following:
a) Create a singly linked list of integers.
b) Delete a given integer from the above linked list.
c) Display the contents of the above list after deletion.

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;
class Node
{
public:
   int info;
   Node* next;
};
class List:public Node
{

   Node *first,*last;
public:
   List()
   {
      first=NULL;
      last=NULL;
   }
   void create();
   void insert();
   void delet();
   void display();
};
void List::create()
{
   Node *temp;
   temp=new Node;
   int n;
   cout<<"\nEnter an Element:";
   cin>>n;
   temp->info=n;
   temp->next=NULL;
   if(first==NULL)
   {
      first=temp;
      last=first;
   }

   Else
```

```
    {
        last->next=temp;
        last=temp;
    }
}
void List::insert()
{
    Node *prev,*cur;
    prev=NULL;
    cur=first;
    int count=1,pos,ch,n;
    Node *temp=new Node;
    cout<<"\nEnter an Element:";
    cin>>n;
    temp->info=n;
    temp->next=NULL;
    cout<<"\nINSERT AS\n1:FIRSTNODE\n2:LASTNODE\n3:IN BETWEEN
FIRST&LAST NODES";
    cout<<"\nEnter Your Choice:";
    cin>>ch;
    switch(ch)
    {
    case 1:
        temp->next=first;
        first=temp;
        break;
    case 2:
        last->next=temp;
        last=temp;
        break;
    case 3:
        cout<<"\nEnter the Position to Insert:";
        cin>>pos;
        while(count!=pos)
        {
            prev=cur;
            cur=cur->next;
            count++;
        }
        if(count==pos)
        {
            prev->next=temp;
            temp->next=cur;
        }
        else
            cout<<"\nNot Able to Insert";
        break;

    }
}
void List::delet()
{
    Node *prev=NULL,*cur=first;
    int count=1,pos,ch;
    cout<<"\nDELETE\n1:FIRSTNODE\n2:LASTNODE\n3:IN BETWEEN
FIRST&LAST NODES";
    cout<<"\nEnter Your Choice:";
    cin>>ch;
    switch(ch)
    {
    case 1:
        if(first!=NULL)
        {
            cout<<"\nDeleted Element is "<<first->info;
            first=first->next;
        }
        else
            cout<<"\nNot Able to Delete";
        break;
    case 2:
        while(cur!=last)
        {
            prev=cur;
            cur=cur->next;
        }
        if(cur==last)
        {
            cout<<"\nDeleted Element is: "<<cur->info;
            prev->next=NULL;
            last=prev;
        }
        else
            cout<<"\nNot Able to Delete";
        break;
    case 3:
        cout<<"\nEnter the Position of Deletion:";
        cin>>pos;
        while(count!=pos)
        {
            prev=cur;
            cur=cur->next;
            count++;
        }
        if(count==pos)
        {
           cout<<"\nDeleted Element is: "<<cur->info;
           prev->next=cur->next;
        }
        else
           cout<<"\nNot Able to Delete";
        break;
     }
  }
  void List::display()
  {
     Node *temp=first;
     if(temp==NULL)
     {
        cout<<"\nList is Empty";
     }
     while(temp!=NULL)
     {
        cout<<temp->info;
        cout<<"-->";
        temp=temp->next;
     }
     cout<<"NULL";
  }
  int main()
  {
     List l;
     int ch;
     while(1)
     {
        cout<<"\n**** MENU ****";
        cout<<"\n1:CREATE\n2:INSERT\n3:DELETE\n4:DISPLAY\n5:EXIT\n";
        cout<<"\nEnter Your Choice:";
        cin>>ch;
        switch(ch)
        {
        case 1:
           l.create();
           break;
        case 2:
           l.insert();
           break;
        case 3:
           l.delet();
```

```
            break;
        case 4:
            l.display();
            break;
        case 5:
            return 0;
        }
    }
    return 0;
}
```

Output:

```
**** MENU ****
1:CREATE
2:INSERT
3:DELETE
4:DISPLAY
5:EXIT

Enter Your Choice:1
Enter an Element:10

**** MENU ****
1:CREATE
2:INSERT
3:DELETE
4:DISPLAY
5:EXIT

Enter Your Choice:2

Enter an Element:20

INSERT AS
1:FIRSTNODE
2:LASTNODE
3:IN BETWEEN FIRST&LAST NODES
Enter Your Choice:1

**** MENU ****
1:CREATE
2:INSERT
3:DELETE
4:DISPLAY
5:EXIT
```

```
Enter Your Choice:3
DELETE
1:FIRSTNODE
2:LASTNODE
3:IN BETWEEN FIRST&LAST NODES
Enter Your Choice:1

Deleted Element is 20
**** MENU ****
1:CREATE
2:INSERT
3:DELETE
4:DISPLAY
5:EXIT

Enter Your Choice:2
Enter an Element:5

INSERT AS
1:FIRSTNODE
2:LASTNODE
3:IN BETWEEN FIRST&LAST NODES
Enter Your Choice:1

**** MENU ****
1:CREATE
2:INSERT
3:DELETE
4:DISPLAY
5:EXIT

Enter Your Choice:4
5-->10-->NULL
**** MENU ****
1:CREATE
2:INSERT
3:DELETE
4:DISPLAY
5:EXIT

Enter Your Choice:2
Enter an Element:20

INSERT AS
1:FIRSTNODE
2:LASTNODE
```

```
3:IN BETWEEN FIRST&LAST NODES
Enter Your Choice:2

**** MENU ****
1:CREATE
2:INSERT
3:DELETE
4:DISPLAY
5:EXIT

Enter Your Choice:4
5-->10-->20-->NULL
**** MENU ****
1:CREATE
2:INSERT
3:DELETE
4:DISPLAY
5:EXIT

Enter Your Choice:2
Enter an Element:15

INSERT AS
1:FIRSTNODE
2:LASTNODE
3:IN BETWEEN FIRST&LAST NODES
Enter Your Choice:3

Enter the Position to Insert:3

**** MENU ****
1:CREATE
2:INSERT
3:DELETE
4:DISPLAY
5:EXIT
Enter Your Choice:4
5-->10-->15-->20-->NULL
**** MENU ****
1:CREATE
2:INSERT
3:DELETE
4:DISPLAY
5:EXIT

Enter Your Choice:5
```

9. Write a template-based C++ program that uses functions to perform the following:
a) Create a doubly linked list of elements.
b) Delete a given element from the above doubly linked list.
c) Display the contents of the above list after deletion.

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
/*
 * Node Declaration
 */
using namespace std;
struct node
{
    int info;
    struct node *next;
    struct node *prev;
}*start;

/*
 Class Declaration
 */
class double_llist
{
    public:
        void create_list(int value);
        void add_begin(int value);
        void add_after(int value, int position);
        void delete_element(int value);
        void search_element(int value);
        void display_dlist();
        void count();
        void reverse();
        double_llist()
        {
            start = NULL;
        }
};

/*
 * Main: Conatins Menu
 */

int main()
```

```
      {

        int choice, element, position;
          double_llist dl;
          while (1)
          {
              cout<<endl<<"----------------------------"<<endl;
              cout<<endl<<"Operations on Doubly linked list"<<endl;
              cout<<endl<<"----------------------------"<<endl;
              cout<<"1.Create Node"<<endl;
              cout<<"2.Add at begining"<<endl;
              cout<<"3.Add after position"<<endl;
              cout<<"4.Delete"<<endl;
              cout<<"5.Display"<<endl;
              cout<<"6.Quit"<<endl;
              cout<<"Enter your choice : ";
              cin>>choice;
              switch ( choice )
              {
              case 1:
                  cout<<"Enter the element: ";
                  cin>>element;
                  dl.create_list(element);
                  cout<<endl;
                  break;
              case 2:
                  cout<<"Enter the element: ";
                  cin>>element;
                  dl.add_begin(element);
                  cout<<endl;
                  break;
              case 3:
                  cout<<"Enter the element: ";
                  cin>>element;
                  cout<<"Insert Element after postion: ";
                  cin>>position;
                  dl.add_after(element, position);
                  cout<<endl;
                  break;
              case 4:
                  if (start == NULL)
                  {
                      cout<<"List empty,nothing to delete"<<endl;
                      break;

                  }
                  cout<<"Enter the element for deletion: ";
```

```
            cin>>element;

            dl.delete_element(element);
            cout<<endl;
            break;
        case 5:
            dl.display_dlist();
            cout<<endl;
            Break;
        case 6:
            exit(1);
        default:
            cout<<"Wrong choice"<<endl;
        }
    }
    return 0;
}

/*
 * Create Double Link List
 */
void double_llist::create_list(int value)
{
    struct node *s, *temp;
    temp = new(struct node);
    temp->info = value;
    temp->next = NULL;
    if (start == NULL)
    {
        temp->prev = NULL;
        start = temp;
    }
    else
    {
        s = start;
        while (s->next != NULL)
            s = s->next;
        s->next = temp;
        temp->prev = s;
    }
}

/*
 * Insertion at the beginning
 */
```

```
void double_llist::add_begin(int value)

{
   if (start == NULL)
   {
      cout<<"First Create the list."<<endl;
      return;
   }
   struct node *temp;
   temp = new(struct node);
   temp->prev = NULL;
   temp->info = value;
   temp->next = start;
   start->prev = temp;
   start = temp;
   cout<<"Element Inserted"<<endl;
}

/*
 * Insertion of element at a particular position
 */
void double_llist::add_after(int value, int pos)
{
   if (start == NULL)
   {
      cout<<"First Create the list."<<endl;
      return;
   }
   struct node *tmp, *q;
   int i;
   q = start;
   for (i = 0;i < pos - 1;i++)
   {
      q = q->next;
      if (q == NULL)
      {
         cout<<"There are less than ";
         cout<<pos<<" elements."<<endl;
         return;
      }
   }
   tmp = new(struct node);

   tmp->info = value;
   if (q->next == NULL)
   {
```

```
            q->next = tmp;
            tmp->next = NULL;

            tmp->prev = q;
        }
        else
        {
            tmp->next = q->next;
            tmp->next->prev = tmp;
            q->next = tmp;
            tmp->prev = q;
        }
        cout<<"Element Inserted"<<endl;
    }

    /*
     * Deletion of element from the list
     */
    void double_llist::delete_element(int value)
    {
        struct node *tmp, *q;
         /*first element deletion*/
        if (start->info == value)
        {
            tmp = start;
            start = start->next;
            start->prev = NULL;
            cout<<"Element Deleted"<<endl;
            free(tmp);
            return;
        }
        q = start;
        while (q->next->next != NULL)
        {
            /*Element deleted in between*/
            if (q->next->info == value)
            {
                tmp = q->next;
                q->next = tmp->next;
                tmp->next->prev = q;
                cout<<"Element Deleted"<<endl;
                free(tmp);
                return;

    }
            q = q->next;
```

```
}
    /*last element deleted*/
   if (q->next->info == value)

   {
      tmp = q->next;
      free(tmp);
      q->next = NULL;
      cout<<"Element Deleted"<<endl;
      return;
   }
   cout<<"Element "<<value<<" not found"<<endl;
}

/*
 * Display elements of Doubly Link List
 */
void double_llist::display_dlist()
{
   struct node *q;
   if (start == NULL)
   {
      cout<<"List empty,"<<endl;
      return;
   }
   q = start;
   cout<<"The Doubly Link List is :"<<endl;
   while (q != NULL)
   {
      cout<<q->info<<" <-> ";
      q = q->next;
   }
   cout<<"NULL"<<endl;
}
```

Output:

```
Operations on Doubly linked list

1.Create Node
2.Add at begining
3.Add after position

4.Delete
5.Display
6.Quit
```

```
Enter your choice : 1
Enter the element: 10


Operations on Doubly linked list

1.Create Node
2.Add at begining
3.Add after position
4.Delete
5.Display
6.Quit

Enter your choice : 5
The Doubly Link List is :
10 <-> NULL

Operations on Doubly linked list

1.Create Node
2.Add at begining
3.Add after position
4.Delete
5.Display
6.Quit

Enter your choice : 2
Enter the element: 15
Element Inserted

Operations on Doubly linked list

1.Create Node
2.Add at begining
3.Add after position
4.Delete
5.Display
6.Quit

Enter your choice : 5
The Doubly Link List is :

15 <-> 10 <-> NULL

Operations on Doubly linked list
```

```
1.Create Node
2.Add at begining
3.Add after position
4.Delete

5.Display
6.Quit

Enter your choice : 3
Enter the element: 35
Insert Element after postion: 1
Element Inserted

Operations on Doubly linked list

1.Create Node
2.Add at begining
3.Add after position
4.Delete
5.Display
6.Quit

Enter your choice : 5
The Doubly Link List is :
15 <-> 35 <-> 10 <-> NULL

Operations on Doubly linked list

1.Create Node
2.Add at begining
3.Add after position
4.Delete
5.Display
6.Quit

Enter your choice : 4
Enter the element for deletion: 35
Element Deleted

Operations on Doubly linked list


1.Create Node
2.Add at begining
3.Add after position
4.Delete
```

```
5.Display
6.Quit

Enter your choice : 6
```

10. Implement the techniques of Selection Sort, Insertion Sort, Quick sort, Heap sort

Selection Sort :

```
#include<iostream>
using namespace std;
void swapping(int &a, int &b) {   //swap the content of a and b
   int temp;
   temp = a;
   a = b;
   b = temp;
}
void display(int *array, int size) {
   for(int i = 0; i<size; i++)
      cout << array[i] << " ";
   cout << endl;
}
void selectionSort(int *array, int size) {
   int i, j, imin;
   for(i = 0; i<size-1; i++) {
      imin = i;   //get index of minimum data
      for(j = i+1; j<size; j++)
         if(array[j] < array[imin])
            imin = j;
         //placing in correct position
         swap(array[i], array[imin]);
   }
}
int main() {
   int n;
   cout << "Enter the number of elements: ";
   cin >> n;
   int arr[n];       //create an array with given number of elements
   cout << "Enter elements:" << endl;
   for(int i = 0; i<n; i++) {
      cin >> arr[i];
   }
   cout << "Array before Sorting: ";
   display(arr, n);
   selectionSort(arr, n);
   cout << "Array after Sorting: ";
   display(arr, n);
}
```

Output:

```
Enter the number of elements: 5

Enter elements:
4
7
0
-4
23

Array before Sorting: 4 7 0 -4 23

Array after Sorting: -4 0 4 7 23
```

Insertion Sort :

```
#include<iostream>
using namespace std;
void display(int *array, int size) {
  for(int i = 0; i<size; i++)
    cout << array[i] << " ";
  cout << endl;
}
void insertionSort(int *array, int size) {
  int key, j;
  for(int i = 1; i<size; i++) {
    key = array[i];//take value
    j = i;
    while(j > 0 && array[j-1]>key) {
      array[j] = array[j-1];
      j--;
    }
    array[j] = key;   //insert in right place
  }
}
int main() {
  int n;
  cout << "Enter the number of elements: ";
  cin >> n;
  int arr[n];   //create an array with given number of elements
  cout << "Enter elements:" << endl;
  for(int i = 0; i<n; i++) {
    cin >> arr[i];
  }
  cout << "Array before Sorting: ";
  display(arr, n);
  insertionSort(arr, n);
  cout << "Array after Sorting: ";
  display(arr, n);
}
```

Output:

```
Enter the number of elements: 5

Enter elements:
9
0
-3
2
33

Array before Sorting: 9 0 -3 2 33

Array after Sorting: -3 0 2 9 33
```

Quick sort :

```
#include <iostream>
using namespace std;
int partition(int arr[], int start, int end)
{

        int pivot = arr[start];

        int count = 0;
        for (int i = start + 1; i <= end; i++) {
                if (arr[i] <= pivot)
                        count++;
        }

        // Giving pivot element its correct position
        int pivotIndex = start + count;
        swap(arr[pivotIndex], arr[start]);

        // Sorting left and right parts of the pivot element
        int i = start, j = end;
        while (i < pivotIndex && j > pivotIndex) {

                while (arr[i] <= pivot) {
                        i++;
                }

                while (arr[j] > pivot) {
                        j--;
                }

                if (i < pivotIndex && j > pivotIndex) {
                        swap(arr[i++], arr[j--]);
                }
        }

        return pivotIndex;
}

void quickSort(int arr[], int start, int end)
{

        // base case
        if (start >= end)
                return;

        // partitioning the array
        int p = partition(arr, start, end);

        // Sorting the left part
        quickSort(arr, start, p - 1);

        // Sorting the right part
        quickSort(arr, p + 1, end);
}

int main()
{
        int n;
        cout<<"Enter the number of elements: ";
        cin>>n;
            int arr[n];
            cout<<"Enter the elements: "<<endl;
        for (int i = 0; i < n; i++) {
                cin>>arr[i];
        }
  quickSort(arr,0,n-1);
  cout<<"Sorted Array : "<<endl;
  for (int i = 0; i < n; i++) {
                cout<<arr[i]<<" "<<endl;
        }
        return 0;
}
```

Output:

```
Enter the number of elements: 6

Enter elements:
3
0
-4
5
32
1

Sorted Array:
-4
0
1
3
5
32
```

Heap sort :

```
#include<iostream>
using namespace std;
void heapify(int arr[], int n, int i) {
   int temp;
   int largest = i;
   int l = 2 * i + 1;
   int r = 2 * i + 2;
   if (l < n && arr[l] > arr[largest])
      largest = l;
   if (r < n && arr[r] > arr[largest])
      largest = r;
   if (largest != i) {
      temp = arr[i];
      arr[i] = arr[largest];
      arr[largest] = temp;
      heapify(arr, n, largest);
   }
}
void heapSort(int arr[], int n) {
  int temp;
  for (int i = n / 2 - 1; i >= 0; i--)
    heapify(arr, n, i);
  for (int i = n - 1; i >= 0; i--) {
    temp = arr[0];
    arr[0] = arr[i];
    arr[i] = temp;
    heapify(arr, i, 0);
  }
}
int main() {
  int i,n;
  cout<<"Enter the number of elements :";
  cin>>n;
  int arr[n];
  cout<<"Enter elements :"<<endl;
  for (i = 0; i < n; i++){
  cin>>arr[i];
  }
  heapSort(arr, n);
  cout<< "Sorted array is:"<<endl;
  for (i = 0; i < n; ++i)
  cout<<arr[i]<<" ";
}
```

Output:

```
Enter the number of elements: 6

Enter elements:
2
0
-7
43
21
6

Sorted array is:
-7  0  2  6  21  43
```

11. Write a C++ program that uses function templates to perform the following:
a) Search for a key element in a list of elements using linear search.
b) Search for a key element in a list of sorted elements using binary search.

```
#include<iostream>
using namespace std;

template<class T>
void Lsearch(T *a,T item,int n)
{
int z=0;
for(int i=0;i<n;i++)
{
if(a[i]==item)
{
z=1;
cout<<"\n Item Found at Position="<<i+1<<"\n\n";
}
else
if(z!=1)
{
z=0;
}
}
if(z==0)
cout<<"\n Item Not Found in the list \n\n";
}

template<class T>
void Bsearch(T *a,T item,int n)
{
int beg=0,end=n-1;
int mid=(beg+end)/2;
while((a[mid]!=item)&&(n>0))
{
if(item>a[mid])
beg=mid;
else
end=mid;
mid=(beg+end)/2;
n--;
}
if(a[mid]==item)
cout<<"\n Item Found at Position="<<mid+1<<"\n\n";
else
cout<<"\n Item Not Found in the List"<<"\n\n";
}

int main()
{
int iarr[10]={2,42,56,86,87,99,323,546,767,886};
double darr[6]={2.4,5.53,44.4,54.45,65.7,89.54};
int iele;
double dele;
cout<<"\n Elements of integer Array\n";
for(int i=0;i<10;i++)
{
cout<<" "<<iarr[i]<<",";
}
cout<<"\n Enter an item to be searched: \n";
cin>>iele;
cout<<"\n Linear Search Method \n";
Lsearch(iarr,iele,10);
cout<<"\n Binary Search Method \n";
Bsearch(iarr,iele,10);
cout<<"\n Elements of double array \n";
for(int i=0;i<6;i++)
{
cout<<" "<<darr[i]<<" , ";
}
cout<<"\n\n Enter an item to be searched:";
cin>>dele;
cout<<"\n Linear Search Method \n";
Lsearch(darr,dele,6);
cout<<"\n Binary Search Method \n";
Bsearch(darr,dele,6);
}
```

Output:

```
Elements of Integer Array
 2 , 42 , 56 , 86 , 87 , 99 , 323 , 546 , 767 , 886 ,

 Enter an item to be search: 888
 Linear Search Method

 Item not found in the list



 Binary Search method
```

```
Item not found in the list


Elements of double Array
2.4 , 5.53 , 44.4 , 54.45 , 65.7 , 89.54 ,

Enter an item to be search: 65.7
Linear Search Method

Item found at position = 5


Binary Search method

Item found at position = 5
```

12. Write a C++ program that uses functions to perform the following:
a) Create a binary search tree of integers.
b) Traverse the above Binary search tree in inorder, preorder and postorder

```
#include<iostream>
using namespace std;

struct st

{

int data;

struct st *left;

struct st *right;

};

struct st *root=NULL,*temp;

void insertElements();

void preorder(struct st *);

void inorder(struct st *);

void postorder(struct st *);

int main()

{

int ch;

while(1)

{

cout<<"\n 1.Insert Elements \n 2.Preorder \n 3.Inorder \n 4.Postorder \n 5.Exit";

cout<<"\n Enter your choice ";


cin>>ch;
switch(ch)

{

case 1:insertElements();
break;

case 2:preorder(root);
break;

case 3:inorder(root);
break;

case 4:postorder(root);
break;

case 5:exit(1);
break;

default:
cout<<"Invalid operation";

}

}

}

void insertElements()

{

struct st *nc,*pNode;

int v;

cout<<"\n Enter the value ";

cin>>v;

temp=new st;

temp->data=v;


temp->left=NULL;
```

```
temp->right=NULL;
if(root==NULL)
{
root=temp;
}
else
{
nc=root;
while(nc!=NULL)
{
pNode=nc;
if(v<nc->data)
nc=nc->left;
else
nc=nc->right;
}
if(v < pNode->data)
{
pNode->left=temp;
}
else
pNode->right=temp;

}
```

```
}
void preorder(struct st *temp)
{
if(temp!=NULL)
{
cout<<" "<<temp->data;
preorder(temp->left);
preorder(temp->right);
}
}
void inorder(struct st *temp)
{
if(temp!=NULL)
{
inorder(temp->left);
cout<<" "<<temp->data;
inorder(temp->right);
}
}
void postorder(struct st *temp)
{
if(temp!=NULL)

{
```

```
postorder(temp->left);

postorder(temp->right);

cout<<" "<<temp->data;

}

}
```

Output:

```
1.insert Elements
 2.preorder
 3.inorder
 4.postorder
5.exit

 enter ur choice 1
 enter the value 5

 1.insert Elements
 2.preorder
 3.inorder
 4.postorder
5.exit

 enter ur choice 1
 enter the value 4

 1.insert Elements
 2.preorder
 3.inorder
 4.postorder
5.exit

 enter ur choice 1
 enter the value 1

 1.insert Elements
 2.preorder
 3.inorder
 4.postorder
5.exit

 enter ur choice 1
 enter the value 8


 1.insert Elements
 2.preorder
 3.inorder
 4.postorder
5.exit

 enter ur choice 3
 1 4 5 8

 1.insert Elements
 2.preorder
 3.inorder
 4.postorder
5.exit

 enter ur choice
 2
 5 4 1 8

 1.insert Elements
 2.preorder
 3.inorder
 4.postorder
5.exit

 enter ur choice 4
 1 4 8 5

 1.insert Elements
 2.preorder
 3.inorder
 4.postorder
5.exit

 enter ur choice 5
```

13. Find the shortest path in a given graph using Dijkstra’s algorithm.

```
#include <bits/stdc++.h>
#include<iostream>




using namespace std;

// Number of vertices in the graph
#define V 9

// A utility function to find the vertex with minimum
// distance value, from the set of vertices not yet included
// in shortest path tree
int minDistance(int dist[], bool sptSet[])
{
    // Initialize min value
    int min = INT_MAX, min_index;
    for (int i = 0; i < V; i++)
        if (sptSet[i] == false && dist[i] <= min)
            min = dist[i], min_index = i;
    return min_index;
}

// Function to print shortest path from source to j using
// parent array
void printPath(int parent[], int j)
{
    // Base Case : If j is source
    if (parent[j] == -1)
        return;
    printPath(parent, parent[j]);
    cout << j << " ";
}

// A utility function to print the constructed distance
// array
int printSolution(int dist[], int n, int parent[])
{
    int src = 0;
    cout << "Vertex\t Distance\tPath";
    for (int i = 1; i < V; i++) {
      cout <<"\n"<< src << "->" << i << "\t\t" << dist[i] <<"\t\t"
         << src  << " ";
        printPath(parent, i);
    }
}

// Function that implements Dijkstra's single source
// shortest path algorithm for a graph represented using
// adjacency matrix representation
void dijkstra(int graph[V][V], int src)
{
    // The output array. dist[i] will hold the shortest
    // distance from src to i
    int dist[V];

    // sptSet[i] will true if vertex i is included / in
    // shortest path tree or shortest distance from src to i
    // is finalized
    bool sptSet[V] = { false };

    // Parent array to store shortest path tree
    int parent[V] = { -1 };

    // Initialize all distances as INFINITE
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX;

    // Distance of source vertex from itself is always 0
    dist[src] = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < V - 1; count++) {
        // Pick the minimum distance vertex from the set of
        // vertices not yet processed. u is always equal to
        // src in first iteration.
        int u = minDistance(dist, sptSet);
        // Mark the picked vertex as processed
        sptSet[u] = true;
        // Update dist value of the adjacent vertices of the
        // picked vertex.
        for (int v = 0; v < V; v++)
            // Update dist[v] only if is not in sptSet,
            // there is an edge from u to v, and total
            // weight of path from src to v through u is
            // smaller than current value of dist[v]
            if (!sptSet[v] && graph[u][v]
                && dist[u] + graph[u][v] < dist[v]) {
                parent[v] = u;
                dist[v] = dist[u] + graph[u][v];
            }
    }
    // print the constructed distance array
    printSolution(dist, V, parent);
}

// Driver Code
int main()
{
    // Let us create the example graph discussed above
    int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                        { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                        { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                        { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                        { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                        { 0, 0, 4, 0, 10, 0, 2, 0, 0 },
                        { 0, 0, 0, 14, 0, 2, 0, 1, 6 },
                        { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                        { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };
    dijkstra(graph, 0);
    return 0;
}
```

Output:

```
Vertex      Distance        Path
0 -> 1          4           0 1
0 -> 2          12          0 1 2
0 -> 3          19          0 1 2 3
0 -> 4          21          0 7 6 5 4
0 -> 5          11          0 7 6 5
0 -> 6          9           0 7 6
0 -> 7          8           0 7
0 -> 8          14          0 1 2 8
```