MARRI LAXMAN REDDY
INSTITUTE OF TECHNOLOGY & MANAGEMENT
(Approved by AICTE-New Delhi, Accredited by NAAC with 'A' & Affliated to JNTU, Hyderabad)
Recognised Under Section 2(f) & 12(B) of The UGC act, 1956
Dundigal, Quthbullapur (M), Hyderabad-500043.

Data Structure Through C++ Lab Manual

Subject Code: CS306ES
Regulation: R16- JNTUH
Class: B. Tech II-I Semister

Prepared by
M. Jagadeesh

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that this manual is a bonafide record of practical work in the **Data Structures through C++** in **First Semester of II year B.Tech (CSE) programme** during the academic year **2017-18**. This book is prepared by **Mr.M.Jagadeesh (Asst.Professor), Mrs.A.Shalini (Asst.Professor), Mr.V.Dinesh (Asst.Professor)** Department of Computer Science and Engineering.

INDEX

PREFACE

This book "Data Structures through C++" lab manual is intended to teach the design and analysis of basic data structures and their implementation in an object-oriented language. Readers of this book need only be familiar with the basic syntax of C++ and similar languages. The "Data Structure" is increasingly becoming the default choice of the IT industry especially industries involved in software development at system level. Therefore, for proper development of "Data Structure" skills among the students this practical manual has been prepared. The manual contains the exercise programs and their solution for easy & quick understanding of the students. We hope that this practical manual will be helpful for students of Computer Science & Engineering for understanding the subject from the point of view of applied aspects. There is always scope for improvement in the manual. We would appreciate to receive valuable suggestions from readers and users for future use.

By,

M.Jagadeesh,
A.Shalini,
V.Dinesh

ACKNOWLEDGEMENT

It was really a good experience, working with ***Data Structures through C++*** lab. First we would like to thank Mr.K.Abdul Basith, Assoc.Professor, HOD of Department of Computer Science and Engineering, Marri Laxman Reddy Institute of technology & Management for his concern and giving the technical support in preparing the document.

We are deeply indebted and gratefully acknowledge the constant support and valuable patronage of Dr.R.Kotaih, Director, Marri Laxman Reddy Institute of technology & Management for giving us this wonderful opportunity for preparing the ***Data Structures through C++*** laboratory manual.

We express our hearty thanks to Dr.K.Venkateswara Reddy, Principal, Marri Laxman Reddy Institute of technology & Management, for timely corrections and scholarly guidance.

At last, but not the least I would like to thanks the entire CSE Department faculties those who had inspired and helped us to achieve our goal.

By,

M.Jagadeesh,
A.Shalini,
V.Dinesh

GENERAL INSTRUCTIONS

1. Students are instructed to come to Data Structures through C++ laboratory on time. Late comers are not entertained in the lab.

2. Students should be punctual to the lab. If not, the conducted experiments will not be repeated.

3. Students are expected to come prepared at home with the experiments which are going to be performed.

4. Students are instructed to display their identity cards before entering into the lab.

5. Students are instructed not to bring mobile phones to the lab.

6. Any damage/loss of system parts like keyboard, mouse during the lab session, it is student's responsibility and penalty or fine will be collected from the student.

7. Students should update the records and lab observation books session wise. Before leaving the lab the student should get his lab observation book signed by the faculty.

8. Students should submit the lab records by the next lab to the concerned faculty members in the staffroom for their correction and return.

9. Students should not move around the lab during the lab session.

10. If any emergency arises, the student should take the permission from faculty member concerned in written format.

11. The faculty members may suspend any student from the lab session on disciplinary grounds.

12. Never copy the output from other students. Write down your own outputs.

INSTITUTION VISION AND MISSION

VISION

To establish as an ideal academic institutions in the service of the nation, the world and the humanity by graduating talented engineers to be ethically strong, globally competent by conducting high quality research, developing breakthrough technologies, and disseminating and preserving technical knowledge.

MISSION

To fulfill the promised vision through the following strategic characteristics and aspirations:

- Contemporary and rigorous educational experiences that develop the engineers and managers.
- An atmosphere that facilitates personal commitment to the educational success of students in an environment that values diversity and community.
- Undergraduate programs that integrate global awareness, communication skills and team building.
- Education and Training that prepares students for interdisciplinary engineering research and advanced problem solving abilities.

DEPARTMENT VISION, MISSION , PROGRAMME EDUCATIONAL OBJECTIVES AND SPECIFIC OUTCOMES

VISION

The Civil Engineering department strives to impart quality education by extracting the innovative skills of students and to face the challenges in latest technological advancements and to serve the society.

MISSION

M-I Provide quality education and to motivate students towards professionalism

M-II Address the advanced technologies in research and industrial issues

PROGRAMME EDUCATIONAL OBJECTIVES

The Programme Educational Objectives (PEOs) that are formulated for the civil engineering programme are listed below;‖

PEO-I solving civil engineering problems in different circumstances PEO-II Pursue higher education and research for professional development.

PEO-III Inculcate qualities of leadership for technology innovation and entrepreneurship.

PROGRAM SPECIFIC OUUTCOMES

PSO1. UNDERSTANDING: Graduates will have an ability to describe, analyze, and solve problems using mathematics and systematic problem-solving techniques.

PSO2. ANALYTICAL SKILLS: Graduates will have an ability to design a system, component, or process to meet desired needs within realistic constraints such as economic, environmental, social, political, ethical, health and safety, manufacturability, and sustainability.

PSO3. BROADNESS: Graduates will have a broad education necessary to understand the impact of engineering solutions in a global, economic, and societal context.

PROGRAMME OUT COMES

a) An ability to apply knowledge of mathematics, science, and engineering

Graduates should transform knowledge of mathematics, Physics, chemistry, Engineering Mechanics, probability and statistics, and engineering drawing in solving a wide range of civil engineering problems.

b) **An ability to design, implement, evaluate a system and conduct experiments, as well as to analyze and interpret data**

Graduates should show that they can make decisions regarding type, and number of data points to be collected, duration of the experiment to obtain intended results, and demonstrate an understanding of accuracy and precision of data

(c) **An ability to design, implement and evaluate a system, or process to meet desired needs** Graduates should be able to: identify the project goal; define the project; search for alternative possibilities; choose the best of the possible solutions; create a design drawing, design plan, or computer simulation; evaluate the design; and justify the final design in written and oral forms.

d) **An ability to function effectively on multi-disciplinary teams**

Graduates should show that they can participate effectively as team members with people who bring different skills, expertise, and perspectives to a project; and with people from different sub-disciplines of civil engineering and interdisciplinary groups.

e) An ability to identify, formulate, analyse and solve engineering problems

Graduates should be able to describe the important components of a given problem, apply mathematical, engineering principles and to find the unknowns and arrive at appropriate and effective solutions.

f) An understanding of professional, ethical, legal, security and social responsibilities Graduates should be familiar with the applicable professional code of conduct for engineers. They should be able to apply the codes, where appropriate, to particular cases in which ethical issues arise. Graduates should also understand the importance of professionalism.

g) An ability to communicate effectively both in writing and orally

Civil engineering graduates should have the ability to speak and write effectively in various domains like laboratory reports, technical reports, technical presentations, project reports etc.

h) The broad education necessary to analyse the impact of engineering solutions on a global and societal context

Graduates should get exposed to the interactions among science, technology, and social values, understand the influence of science and technology on civilizations and how science and technology have been addressed for the betterment of humankind.

i) Recognition of the need for, and an ability to engage in continuing professional development and life-long learning

Graduates should show that they appreciate the need for further education and self improvement, understand the value of professional licensure the necessity of continuing professional developments, and the value of membership in appropriate professional organizations.

j) Knowledge of contemporary issues

Graduates should have knowledge and understand selected contemporary technical and social issues relevant to their field of study.

k) An ability to use the techniques, skills, and modern engineering tools necessary for engineering practice

Graduates should have ability to use practical methods readily and effectively in the performance of engineering analysis and design. Graduates should be able to select and use modern engineering tools used by practicing engineers, including computer software such as computer aided drawing (CAD)

l) An ability to apply design and development principles in the construction of software and hardware systems of varying complexity

Civil Graduates should have ability to design and develop principles involved in construction of different structures like buildings, shopping complexes, roads, water structures and to analyse the stability of structures using different softwares like stadpro. Studs etc.

COURSE STRUCTURE, OBJECTIVES & OUTCOMES

COURSE STRUCTURE

Environmental engineering lab will have a continuous evaluation during 7th semester for 25 sessional marks and 50 end semester examination marks.

Out of the 25 marks for internal evaluation, day-to-day work in the laboratory shall be evaluated for 15 marks and internal practical examination shall be evaluated for 10 marks conducted by the laboratory teacher concerned.

The end semester examination shall be conducted with an external examiner and internal examiner. The external examiner shall be appointed by the principal / Chief Controller of examinations

COURSE OBJECTIVE

The objective of this laboratory is to determine the qualities of water and waste water characteristics. The experiments include the determination of pH, turbidity, conductivity, and impurities in water and BOD, DO and COD of waste water.

The highlight of this laboratory is the spectrophotometer, muffle furnace, BOD incubator, COD Digestor etc. This laboratory course will help the students to understand the theoretical concepts learned in the course Environmental Engineering.

COURSE OUTCOME

1. Quantify the pollutant concentration in water, wastewater
2. Recommend the degree of treatment required for the water and wastewater

III. Course files – Soft copies

HODs and faculty members are directed to submit the soft copies of the course files in the director's office/L.M.S at the earliest, other wise it will be treated as not submitted

1. 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.

AIM: Implement the Single linked List
Algorithms:
Inserting a node into SLL:
Algorithm: Insert at front()
{
Allocate a memory for new node(new1)
if (new1== NULL)
then
display memory is full (insertion is not possible)
end if
else
read element ele;
new1->data=ele;/*copy the data into new1 node */
new1->next=header->next;
header->next=new1;
end else
}
Algorithm: Insert at End()
{
Allocate a memory for new node(new1)
if(new1== NULL)
then
display memory is full (insertion is not possible)
end if
else
ptr=header
while(ptr->next!=NULL)/*ptr moves end of list */
then
ptr=ptr->next
end
new1->next=NULL
new1->data=ele
ptr->next=new1
end else
}
Algorithm: Insert at middle ()
{
Allocate a memory for new node(new1)
if (new1== NULL)
then

```
display memory is full (insertion is not possible)
exit
else
read ele,pos
ptr=header
count=0
while(ptr->next!=NULL)
{
ptr=ptr->next;
count ++;
}
if(count<pos-1)
then
display position is not of range
end
ptr=header
count=0
while(count<pos-1)
then
ptr=ptr->next
count++
end
new1->data=ele
new1->next=ptr->next
ptr->next=new1
end else
}
```

Deleting a node from the SLL:

Algorithm: Delete at front()

```
{
if (header->next==NULL)
then
display list is empty
end if
else
temp=header->next;
header->next=header->next->next;
free(temp);
end if
}
```

Algorithm: Delete at end()

```
{
if(header->next==NULL)
then
display list is empty
end if
else
```

```
ptr=header;
while(ptr->next->next!=NULL)
{
ptr->=Ptr->next;
}
temp=ptr->next;
ptr->next=NULL;
free(temp);
end else
}
```

Algorithm:delete at anyposition()

```
{
if (header->next==NULL)
then
display list is empty /*deletion is not possible*/
end if
ptr=header;
count=0;
while(ptr->next!=NULL)
{
ptr=ptr->next;
count++;
}
if(count<pos-1)
{
Display position is out of range
}
ptr=header;
count=0;
while(count<pos-1)
{
ptr=ptr->next;
count++;
}
temp=ptr->next;
ptr->next=temp->next;
free(temp)
}
```

Traversing a List:

Algorithm: Traverse()

```
{
ptr=header;
if(ptr ->next==NULL)
then
display list is empty
end if
else
```

```
while(ptr -> next!=NULL)
{
display ptr-> data
ptr=ptr -> next /* move to next node */
}
end if
}
```

Source Code:

```
/*
 * C++ Program to Implement Singly Linked List
 */
#include<iostream>
#include<cstdio>
#include<cstdlib>
using namespace std;
/*
 * Node Declaration
 */
struct node
{
   int info;
   struct node *next;
}*start;

/*
 * Class Declaration
 */
class single_llist
{
   public:
      node* create_node(int);
      void insert_begin();
      void insert_pos();
      void insert_last();
      void delete_pos();
      void sort();
      void search();
      void update();
      void reverse();
      void display();
      single_llist()
      {
         start = NULL;
      }
};
```

```
/*
 * Main :contains menu
 */
main()
{
   int choice, nodes, element, position, i;
   single_llist sl;
   start = NULL;
   while (1)
   {
      cout<<endl<<"---------------------------------"<<endl;
      cout<<endl<<"Operations on singly linked list"<<endl;
      cout<<endl<<"---------------------------------"<<endl;
      cout<<"1.Insert Node at beginning"<<endl;
      cout<<"2.Insert node at last"<<endl;
      cout<<"3.Insert node at position"<<endl;
      cout<<"4.Sort Link List"<<endl;
      cout<<"5.Delete a Particular Node"<<endl;
      cout<<"6.Update Node Value"<<endl;
      cout<<"7.Search Element"<<endl;
      cout<<"8.Display Linked List"<<endl;
      cout<<"9.Reverse Linked List "<<endl;
      cout<<"10.Exit "<<endl;
      cout<<"Enter your choice : ";
      cin>>choice;
      switch(choice)
      {
      case 1:
         cout<<"Inserting Node at Beginning: "<<endl;
         sl.insert_begin();
         cout<<endl;
         break;
      case 2:
         cout<<"Inserting Node at Last: "<<endl;
         sl.insert_last();
         cout<<endl;
         break;
      case 3:
         cout<<"Inserting Node at a given position:"<<endl;
         sl.insert_pos();
         cout<<endl;
         break;
      case 4:
         cout<<"Sort Link List: "<<endl;
         sl.sort();
```

```
            cout<<endl;
            break;
        case 5:
            cout<<"Delete a particular node: "<<endl;
            sl.delete_pos();
            break;
        case 6:
            cout<<"Update Node Value:"<<endl;
            sl.update();
            cout<<endl;
            break;
        case 7:
            cout<<"Search element in Link List: "<<endl;
            sl.search();
            cout<<endl;
            break;
        case 8:
            cout<<"Display elements of link list"<<endl;
            sl.display();
            cout<<endl;
            break;
        case 9:
            cout<<"Reverse elements of Link List"<<endl;
            sl.reverse();
            cout<<endl;
            break;
        case 10:
            cout<<"Exiting..."<<endl;
            exit(1);
            break;
        default:
            cout<<"Wrong choice"<<endl;
        }
    }
}

/*
 * Creating Node
 */
node *single_llist::create_node(int value)
{
    struct node *temp, *s;
    temp = new(struct node);
    if (temp == NULL)
    {
        cout<<"Memory not allocated "<<endl;
```

```
        return 0;
    }
    else
    {
        temp->info = value;
        temp->next = NULL;
        return temp;
    }
}

/*
 * Inserting element in beginning
 */
void single_llist::insert_begin()
{
    int value;
    cout<<"Enter the value to be inserted: ";
    cin>>value;
    struct node *temp, *p;
    temp = create_node(value);
    if (start == NULL)
    {
        start = temp;
        start->next = NULL;
    }
    else
    {
        p = start;
        start = temp;
        start->next = p;
    }
    cout<<"Element Inserted at beginning"<<endl;
}

/*
 * Inserting Node at last
 */
void single_llist::insert_last()
{
    int value;
    cout<<"Enter the value to be inserted: ";
    cin>>value;
    struct node *temp, *s;
    temp = create_node(value);
    s = start;
    while (s->next != NULL)
```

```
    {
        s = s->next;
    }
    temp->next = NULL;
    s->next = temp;
    cout<<"Element Inserted at last"<<endl;
}

/*
 * Insertion of node at a given position
 */
void single_llist::insert_pos()
{
    int value, pos, counter = 0;
    cout<<"Enter the value to be inserted: ";
    cin>>value;
    struct node *temp, *s, *ptr;
    temp = create_node(value);
    cout<<"Enter the postion at which node to be inserted: ";
    cin>>pos;
    int i;
    s = start;
    while (s != NULL)
    {
        s = s->next;
        counter++;
    }
    if (pos == 1)
    {
        if (start == NULL)
        {
            start = temp;
            start->next = NULL;
        }
        else
        {
            ptr = start;
            start = temp;
            start->next = ptr;
        }
    }
    else if (pos > 1  && pos <= counter)
    {
        s = start;
        for (i = 1; i < pos; i++)
        {
```

```
            ptr = s;
            s = s->next;
        }
        ptr->next = temp;
        temp->next = s;
    }
    else
    {
        cout<<"Positon out of range"<<endl;
    }
}

/*
 * Sorting Link List
 */
void single_llist::sort()
{
    struct node *ptr, *s;
    int value;
    if (start == NULL)
    {
        cout<<"The List is empty"<<endl;
        return;
    }
    ptr = start;
    while (ptr != NULL)
    {
        for (s = ptr->next;s !=NULL;s = s->next)
        {
            if (ptr->info > s->info)
            {
                value = ptr->info;
                ptr->info = s->info;
                s->info = value;
            }
        }
        ptr = ptr->next;
    }
}

/*
 * Delete element at a given position
 */
void single_llist::delete_pos()
{
    int pos, i, counter = 0;
```

```
    if (start == NULL)
    {
       cout<<"List is empty"<<endl;
       return;
    }
    cout<<"Enter the position of value to be deleted: ";
    cin>>pos;
    struct node *s, *ptr;
    s = start;
    if (pos == 1)
    {
       start = s->next;
    }
    else
    {
       while (s != NULL)
       {
          s = s->next;
          counter++;
       }
       if (pos > 0 && pos <= counter)
       {
          s = start;
          for (i = 1;i < pos;i++)
          {
             ptr = s;
             s = s->next;
          }
          ptr->next = s->next;
       }
       else
       {
          cout<<"Position out of range"<<endl;
       }
       free(s);
       cout<<"Element Deleted"<<endl;
    }
}

/*
 * Update a given Node
 */
void single_llist::update()
{
    int value, pos, i;
    if (start == NULL)
```

```
    {
        cout<<"List is empty"<<endl;
        return;
    }
    cout<<"Enter the node postion to be updated: ";
    cin>>pos;
    cout<<"Enter the new value: ";
    cin>>value;
    struct node *s, *ptr;
    s = start;
    if (pos == 1)
    {
        start->info = value;
    }
    else
    {
        for (i = 0;i < pos - 1;i++)
        {
            if (s == NULL)
            {
                cout<<"There are less than "<<pos<<" elements";
                return;
            }
            s = s->next;
        }
        s->info = value;
    }
    cout<<"Node Updated"<<endl;
}

/*
 * Searching an element
 */
void single_llist::search()
{
    int value, pos = 0;
    bool flag = false;
    if (start == NULL)
    {
        cout<<"List is empty"<<endl;
        return;
    }
    cout<<"Enter the value to be searched: ";
    cin>>value;
    struct node *s;
    s = start;
```

```
    while (s != NULL)
    {
        pos++;
        if (s->info == value)
        {
            flag = true;
            cout<<"Element "<<value<<" is found at position "<<pos<<endl;
        }
        s = s->next;
    }
    if (!flag)
        cout<<"Element "<<value<<" not found in the list"<<endl;
}

/*
 * Reverse Link List
 */
void single_llist::reverse()
{
    struct node *ptr1, *ptr2, *ptr3;
    if (start == NULL)
    {
        cout<<"List is empty"<<endl;
        return;
    }
    if (start->next == NULL)
    {
        return;
    }
    ptr1 = start;
    ptr2 = ptr1->next;
    ptr3 = ptr2->next;
    ptr1->next = NULL;
    ptr2->next = ptr1;
    while (ptr3 != NULL)
    {
        ptr1 = ptr2;
        ptr2 = ptr3;
        ptr3 = ptr3->next;
        ptr2->next = ptr1;
    }
    start = ptr2;
}

/*
 * Display Elements of a link list
```

```
 */
void single_llist::display()
{
   struct node *temp;
   if (start == NULL)
   {
      cout<<"The List is Empty"<<endl;
      return;
   }
   temp = start;
   cout<<"Elements of list are: "<<endl;
   while (temp != NULL)
   {
      cout<<temp->info<<"->";
      temp = temp->next;
   }
   cout<<"NULL"<<endl;
}
```

Output:

```
---------------------------------
 Operations on singly linked list
 ---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 8
Display elements of link list
The List is Empty.
  ---------------------------------
 Operations on singly linked list
 ---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
```

```
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 5
Delete a particular node:
List is empty
 ---------------------------------
 Operations on singly linked list
 ---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 6
Update Node Value:
List is empty
 ---------------------------------
 Operations on singly linked list
 ---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 7
Search element in Link List:
List is empty
 ---------------------------------
 Operations on singly linked list
 ---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
```

```
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 3
Inserting Node at a given position:
Enter the value to be inserted: 1010
Enter the postion at which node to be inserted: 5
Positon out of range
---------------------------------
 Operations on singly linked list
 ---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 1
Inserting Node at Beginning:
Enter the value to be inserted: 100
Element Inserted at beginning
 ---------------------------------
 Operations on singly linked list
---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 1
Inserting Node at Beginning:
Enter the value to be inserted: 200
Element Inserted at beginning
```

```
 ---------------------------------
 Operations on singly linked list
 ---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 8
Display elements of link list
Elements of list are:
200->100->NULL
 ---------------------------------
 Operations on singly linked list
 ---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 2
Inserting node at last:
Enter the value to be inserted: 50
Element Inserted at last
 ---------------------------------
 Operations on singly linked list
 ---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
```

```
10.Exit
Enter your choice : 2
Inserting node at last:
Enter the value to be inserted: 150
Element Inserted at last
 ---------------------------------
 Operations on singly linked list
 ---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 8
Display elements of link list
Elements of list are:
200->100->50->150->NULL
 ---------------------------------
 Operations on singly linked list
 ---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 3
Inserting node at a given position:
Enter the value to be inserted: 1111
Enter the position at which node to be inserted: 4
 ---------------------------------
 Operations on singly linked list
 ---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
```

```
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 8
Display elements of link list
Elements of list are:
200->100->50->1111->150->NULL
 --------------------------------
 Operations on singly linked list
 --------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 3
Inserting node at a given position:
Enter the value to be inserted: 1010
Enter the position at which node to be inserted: 100
Position out of range
 --------------------------------
 Operations on singly linked list
 --------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 8
Display elements of link list
Elements of list are:
200->100->50->1111->150->NULL
```

```
--------------------------------
 Operations on singly linked list
 --------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 5
Delete a Particular node:
Enter the position of value to be deleted: 1
 --------------------------------
 Operations on singly linked list
 --------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 8
Display elements of link list
Elements of list are:
100->50->1111->150->NULL
--------------------------------
 Operations on singly linked list
 --------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
```

```
Enter your choice : 6
Update Node Value:
Enter the node position to be updated: 1
Enter the new value: 1010
Node Updated
 --------------------------------
 Operations on singly linked list
 --------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 8
Display elements of link list
Elements of list are:
1010->50->1111->150->NULL
--------------------------------
 Operations on singly linked list
 --------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 7
Search element in Link List:
Enter the value to be searched: 50
Element 50 is found at position 2
 --------------------------------
 Operations on singly linked list
 --------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
```

```
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 9
Reverse elements of Link List
 ---------------------------------
 Operations on singly linked list
 ---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 8
Display elements of link list
Elements of list are:
150->1111->50->1010->NULL
---------------------------------
 Operations on singly linked list
 ---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 4
Sort Link List:
 ---------------------------------
Operations on singly linked list
 ---------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
```

```
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 8
Display elements of link list
Elements of list are:
50->150->1010->1111->NULL
--------------------------------
 Operations on singly linked list
 --------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Sort Link List
5.Delete a Particular Node
6.Update Node Value
7.Search Element
8.Display Linked List
9.Reverse Linked List
10.Exit
Enter your choice : 10
```

Viva Questions:

1. List the differences between the linked array and linked list.
2. What is meant by single linked list.
3. What type of memory allocation is referred for Linked lists?
4. name the types of Linked Lists?
5. Mention what are the applications of Linked Lists?
6. List the disadvantages of single linked list.

2. 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.

AIM: Implement the double linked List
Algorithms:
Inserting a node into DLL:
Algorithm: Insert at front()
{
Allocate a memory for new node(new1)
if (new1== NULL)
then
display memory is full (insertion is not possible)
end if
else
read element ele;
new1->data=ele;/*copy the data into new1 node */
new1->next=header->next;
header->next->prev=new1;
header->next=new1;
new1->prev=header;
end else
}
Algorithm: Insert at End()
{
Allocate a memory for new node(new1)
if(new1== NULL)
then
display memory is full (insertion is not possible)
end if
else
ptr=header
while(ptr->next!=NULL)/*ptr moves end of list */
then
ptr=ptr->next
end
new1->next=NULL
new1->prev=ptr;
new1->data=ele
ptr->next=new1
end else
}
Algorithm: Insert at middle ()
{
Allocate a memory for new node(new1)

```
if (new1== NULL)
then
display memory is full (insertion is not possible)
exit
else
read ele,pos
ptr=header
count=0
while(ptr->next!=NULL)
{
ptr=ptr->next;
count ++;
}
if(count<pos-1)
then
display position is not of range
end
ptr=header
count=0
while(count<pos-1)
then
ptr=ptr->next
count++
end
new1->data=ele;
new1->next=ptr->next;
new1->prev=ptr;
ptr->next->prev=new1;
ptr->next=new1
end else
}
```

Deleting a node from the DLL:

Algorithm: delete at front()

```
{
if (header->next==NULL)
then
display list is empty
end if
else
temp=header->next;
header->next=header->next->next;
temp->next->prev=header;
free(temp);
end if
}
```

Algorithm: Delete at end()

```
{
```

```
if(header->next==NULL)
then
display list is empty
end if
else
ptr=header;
while(ptr->next->next!=NULL)
{
ptr->=Ptr->next;
}
temp=ptr->next;
ptr->next=NULL;
free(temp);
end else
}
```

Algorithm:delete at anyposition()

```
{
if (header->next==NULL)
then
display list is empty /*deletion is not possible*/
end if
ptr=header;
count=0;
while(ptr->next!=NULL)
{
ptr=ptr->next;
count++;
}
if(count<pos-1)
{
Display position is out of range
}
ptr=header;
count=0;
while(count<pos-1)
{
ptr=ptr->next;
count++;
}
temp=ptr->next;
ptr->next=temp->next;
temp->next->prev=Ptr;
free(temp)
}
```

Traversing a List:

Algorithm: Traverse()

```
{
ptr=header;
if(ptr ->next==NULL)
then
display list is empty
end if
else
while(ptr -> next!=NULL)
{
display ptr-> data
ptr=ptr -> next /* move to next node */
}
end if
}
```

Source Code:

```
/*
 * C++ Program to Implement Doubly Linked List
 */
#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.Count"<<endl;
      cout<<"7.Reverse"<<endl;
      cout<<"8.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:
            dl.count();
            break;
         case 7:
            if (start == NULL)
            {
               cout<<"List empty,nothing to reverse"<<endl;
               break;
            }
            dl.reverse();
            cout<<endl;
            break;
         case 8:
            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,nothing to display"<<endl;
        return;
    }
    q = start;
    cout<<"The Doubly Link List is :"<<endl;
    while (q != NULL)
    {
```

```
        cout<<q->info<<" <-> ";
        q = q->next;
    }
    cout<<"NULL"<<endl;
}

/*
 * Number of elements in Doubly Link List
 */
void double_llist::count()
{
    struct node *q = start;
    int cnt = 0;
    while (q != NULL)
    {
        q = q->next;
        cnt++;
    }
    cout<<"Number of elements are: "<<cnt<<endl;
}

/*
 * Reverse Doubly Link List
 */
void double_llist::reverse()
{
    struct node *p1, *p2;
    p1 = start;
    p2 = p1->next;
    p1->next = NULL;
    p1->prev = p2;
    while (p2 != NULL)
    {
        p2->prev = p2->next;
        p2->next = p1;
        p1 = p2;
        p2 = p2->prev;
    }
    start = p1;
    cout<<"List Reversed"<<endl;
}
```

Output:

```
---------------------------------
 Operations on Doubly linked list
 ---------------------------------
1.Create Node
2.Add at begining
3.Add after
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 2
Enter the element: 100
First Create the list.
  ---------------------------------
Operations on Doubly linked list
 ---------------------------------
1.Create Node
2.Add at begining
3.Add after
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 3
Enter the element: 200
Insert Element after postion: 1
First Create the list.
---------------------------------
 Operations on Doubly linked list
 ---------------------------------
1.Create Node
2.Add at begining
3.Add after
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 4
List empty,nothing to delete
 ---------------------------------
 Operations on Doubly linked list
 ---------------------------------
```

```
1.Create Node
2.Add at begining
3.Add after
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 5
List empty,nothing to display
 ---------------------------------
 Operations on Doubly linked list
 ---------------------------------
1.Create Node
2.Add at begining
3.Add after
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 6
Number of elements are: 0
 ---------------------------------
 Operations on Doubly linked list
 ---------------------------------
1.Create Node
2.Add at begining
3.Add after
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 7
List empty,nothing to reverse
 ---------------------------------
 Operations on Doubly linked list
 ---------------------------------
1.Create Node
2.Add at begining
3.Add after
4.Delete
5.Display
6.Count
7.Reverse
```

```
8.Quit
Enter your choice : 1
Enter the element: 100
 ---------------------------------
 Operations on Doubly linked list
 ---------------------------------
1.Create Node
2.Add at begining
3.Add after
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 5
The Doubly Link List is :
100 <-> NULL
 ---------------------------------
 Operations on Doubly linked list
 ---------------------------------
1.Create Node
2.Add at begining
3.Add after
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 2
Enter the element: 200
Element Inserted
 ---------------------------------
 Operations on Doubly linked list
 ---------------------------------
1.Create Node
2.Add at begining
3.Add after
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 5
The Doubly Link List is :
200 <-> 100 <-> NULL
```

```
---------------------------------
 Operations on Doubly linked list
 ---------------------------------
1.Create Node
2.Add at begining
3.Add after
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 3
Enter the element: 50
Insert Element after postion: 2
Element Inserted
 ---------------------------------
 Operations on Doubly linked list
 ---------------------------------
1.Create Node
2.Add at begining
3.Add after
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 5
The Doubly Link List is :
200 <-> 100 <-> 50 <-> NULL
 ---------------------------------
 Operations on Doubly linked list
 ---------------------------------
1.Create Node
2.Add at begining
3.Add after
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 3
Enter the element: 150
Insert Element after postion: 3
Element Inserted
  ---------------------------------
 Operations on Doubly linked list
```

```
 ---------------------------------
1.Create Node
2.Add at begining
3.Add after
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 5
The Doubly Link List is :
200 <-> 100 <-> 50 <-> 150 <-> NULL
 ---------------------------------
 Operations on Doubly linked list
 ---------------------------------
1.Create Node
2.Add at begining
3.Add after
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 6
Number of elements are: 4
 ---------------------------------
 Operations on Doubly linked list
 ---------------------------------
1.Create Node
2.Add at begining
3.Add after
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 4
Enter the element for deletion: 50
Element Deleted
---------------------------------
 Operations on Doubly linked list
 ---------------------------------
1.Create Node
2.Add at begining
3.Add after
4.Delete
```

```
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 5
The Doubly Link List is :
200 <-> 100 <-> 150 <-> NULL
 ---------------------------------
 Operations on Doubly linked list
 ---------------------------------
1.Create Node
2.Add at begining
3.Add after
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 6
Number of elements are: 3
---------------------------------
 Operations on Doubly linked list
 ---------------------------------
1.Create Node
2.Add at begining
3.Add after
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 7
List Reversed
 ---------------------------------
 Operations on Doubly linked list
 ---------------------------------
1.Create Node
2.Add at begining
3.Add after
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 5
The Doubly Link List is :
```

```
150 <-> 100 <-> 200 <-> NULL
 --------------------------------
 Operations on Doubly linked list
 --------------------------------
1.Create Node
2.Add at begining
3.Add after
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 3
Enter the element: 200
Insert Element after postion: 100
There are less than 100 elements.
 --------------------------------
 Operations on Doubly linked list
 --------------------------------
1.Create Node
2.Add at begining
3.Add after
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 4
Enter the element for deletion: 150
Element Deleted
 --------------------------------
 Operations on Doubly linked list
 --------------------------------
1.Create Node
2.Add at begining
3.Add after
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 5
The Doubly Link List is :
100 <-> 200 <-> NULL
 --------------------------------
 Operations on Doubly linked list
```

```
---------------------------------
1.Create Node
2.Add at begining
3.Add after
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 8
```

Viva Questions :

1. What is the double linked list?
2. What are the applications of double linked list?
3. What are the advantages of double linked list over single linked list?
4. How to add the new node in middle of the double linked list?
5. What are the memory allocation functions to create the double linked list?

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

Algorithm:

Step 1: Define a stack array.
Step 2: Scan each character in the infix string
Step 3: If it is between 0 to 9 or any alphabet, append it to postfix string.
Step 4: If it is left parenthesis push to stack
If it is operator *,+,-,/,%,^ then
If the stack is empty push it to the stack
If the stack is not empty then start a loop:
If the top of the stack has higher precedence
Then pop and append to output string
Else break
Push to the stack
If it is right parenthesis then
While stack not empty and top not equal to left brace
Pop from stack and append to output string
Finally pop out the left brace.
Step 5: If there is any input in the stack pop and append to the Postfix string.

Source Code:

```
#include <iostream.h>
#include <string.h>
#include <ctype.h>
const int MAX = 50 ;
class infix
{
        private :
                char target[MAX], stack[MAX] ;
                char *s, *t ;
                int top ;
        public :
                infix( ) ;
                void setexpr ( char *str ) ;
                void push ( char c ) ;
                char pop( ) ;
                void convert( ) ;
                int priority ( char c ) ;
                void show( ) ;
} ;
infix :: infix( )
{
        top = -1 ;
        strcpy ( target, "" ) ;
        strcpy ( stack, "" ) ;
```

```
        t = target ;
        s = ""  ;
}
void infix :: setexpr ( char *str )
{
        s = str ;
}
void infix :: push ( char c )
{
        if ( top == MAX )
                cout << "\nStack is full\n" ;
        else
        {
                top++ ;
                stack[top] = c ;
        }
}
char infix :: pop( )
{
        if ( top == -1 )
        {
                cout << "\nStack is empty\n" ;
                return -1 ;
        }
        else
        {
                char item = stack[top] ;
                top-- ;
                return item ;
        }
}
void infix :: convert( )
{
        while ( *s )
        {
                if ( *s == ' ' || *s == '\t' )
                {
                        s++ ;
                        continue ;
                }
                if ( isdigit ( *s ) || isalpha ( *s ) )
                {
                        while ( isdigit ( *s ) || isalpha ( *s ) )
                        {
                                *t = *s ;
                                s++ ;
```

```
                        t++ ;
                }
        }
        if ( *s == '(' )
        {
                push ( *s ) ;
                s++ ;
        }
        char opr ;
        if ( *s == '*' || *s == '+' || *s == '/' || *s == '%' || *s == '-' || *s == '$' )
        {
                if ( top != -1 )
                {
                        opr = pop( ) ;
                        while ( priority ( opr ) >= priority ( *s ) )
                        {
                                *t = opr ;
                                t++ ;
                                opr = pop( ) ;
                        }
                        push ( opr ) ;
                        push ( *s ) ;
                }
                else
                        push ( *s ) ;
                s++ ;
        }
        if ( *s == ')' )
        {
                opr = pop( ) ;
                while ( ( opr ) != '(' )
                {
                        *t = opr ;
                        t++ ;
                        opr =  pop( ) ;
                }
                s++ ;
        }
}
while ( top != -1 )
{
        char opr = pop( ) ;
        *t = opr ;
        t++ ;
}
*t = '\0' ;
```

```
}
int infix :: priority ( char c )
{
        if ( c == '$' )
                return 3 ;
        if ( c == '*' || c == '/' || c == '%' )
                return 2 ;
        else
        {
                if ( c == '+' || c == '-' )
                        return 1 ;
                else
                        return 0 ;
        }
}
void infix :: show( )
{
        cout << target ;
}
void main( )
{
        char expr[MAX] ;
        infix q ;

        cout << "\nEnter an expression in infix form: " ;
        cin.getline ( expr, MAX ) ;

        q.setexpr ( expr ) ;
        q.convert( ) ;

        cout << "\nThe postfix expression is: " ;
        q.show( ) ;
}
```

Output:

```
Enter the postfix experssion : a+b*c
Post fix expression is : abc*+
Enter the postfix experssion : a+(b+c*d)*f
Post fix expression is : abcd*+f*+
Enter the postfix experssion : a+(b*c+d)*f
Post fix expression is : abc*d+f*+
```

Viva Questions:

1. What are the applications of stack?
2. List the types of conversions.
3. What is the inorder?
4. What is the postorder?

4. Write a C++ program to implement a double ended queue ADT using an array, using a doubly linked list.

Algorithm:

i) array
AIM: Implement a DEQUEUE using arrays
Algorithm for Insertion at rear end
Step -1: [Check for overflow]
if(rear==MAX)
Print("Queue is Overflow");
return;
Step-2: [Insert element]
else
rear=rear+1;
q[rear]=no;
[Set rear and front pointer]
if rear=0
rear=1;
if front=0
front=1;
Step-3: return
Algorithm for Insertion at font end
Step-1 : [Check for the front position]
if(front<=1)
Print ("Cannot add item at front end");
return;
Step-2 : [Insert at front]
else
front=front-1;
q[front]=no;
Step-3 : Return
Algorithm for Deletion from front end
Step-1 [Check for front pointer]
if front=0
print(" Queue is Underflow");
return;
Step-2 [Perform deletion]
else
no=q[front];
print("Deleted element is",no);
[Set front and rear pointer]
if front=rear
front=0;
rear=0;
else
front=front+1;
Step-3 : Return

Algorithm for Deletion from rear end

Step-1 : [Check for the rear pointer]
if rear=0
print("Cannot delete value at rear end");
return;
Step-2: [perform deletion]
else
no=q[rear];
[Check for the front and rear pointer]
if front= rear
front=0;
rear=0;
else
rear=rear-1;
print("Deleted element is",no);
Step-3 : Return

Source Code:

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;

class queue
{
        int queue1[5];
        int rear,front;
    public:
        queue()
          {
             rear=-1;
             front=-1;
          }
        void insert(int x)
         {
            if(rear >  4)
             {
               cout <<"queue over flow";
               front=rear=-1;
               return;
             }
             queue1[++rear]=x;
             cout <<"inserted" <<x;
         }
        void delet()
         {
            if(front==rear)
```

```
                {
                   cout <<"queue under flow";
                   return;
                }
                cout <<"deleted" <<queue1[++front];
            }
           void display()
            {
               if(rear==front)
                {
                    cout <<" queue empty";
                    return;
                }
               for(int i=front+1;i<=rear;i++)
               cout <<queue1[i]<<" ";
            }
};

main()
{
    int ch;
    queue qu;
    while(1)
     {
          cout <<"\n1.insert  2.delet  3.display  4.exit\nEnter ur choice";
          cin >> ch;
          switch(ch)
            {
              case 1:    cout <<"enter the element";
                         cin >> ch;
                        qu.insert(ch);
                        break;
              case 2:  qu.delet();  break;
              case 3:  qu.display();break;
              case 4: exit(0);
              }
        }
return (0);
}
```

Output:

```
1.insert 2.delet 3.display 4.exit
Enter ur choice1
enter the element21
inserted21
```

1.insert 2.delet 3.display 4.exit
Enter ur choice1
enter the element22
inserted22

1.insert 2.delet 3.display 4.exit
Enter ur choice1
enter the element16
inserted16

1.insert 2.delet 3.display 4.exit
Enter ur choice3
21 22 16
1.insert 2.delet 3.display 4.exit
Enter ur choice2
deleted21

1.insert 2.delet 3.display 4.exit
Enter ur choice3
22 16

1.insert 2.delet 3.display 4.exit
Enter ur choice

ii) Linked List

AIM: To implement a Double ended Queue using Linked List

Algorithm: display()
{
ptr = front; //assign the ptr to front node
if(front==NULL || rear==NULL)
{
printf("List is empty");
}
while(ptr != NULL)
{
Display ptr ->data
Pointer move to next node i.e: ptr = ptr->next;
}
}
Algorithm: insert_begin(x)
{
Allocate a memory for new node(new1)
new1 -> data =x;
new1 ->previous = new1 ->next =NULL;
if(front == NULL||rear==NULL)

```
front = rear = new1;
else
{
new1 ->next = front;
front ->previous = new1;
front = new1;
}
}
```

Algorithm: insert_last(x)

```
{
Allocate a memory for new node (new1)
new1 ->data = x;
new1 -> previous = new1 ->next = NULL;
if (front == NULL||rear==NULL)
front = rear = new1;
else
{
rear ->next = new1;
new1 ->previous = rear;
rear = new1;
}
}
```

Algorithm: delete_begin()

```
{
if (front == NULL || rear==NULL)
{
Display List is empty
}
else
{
temp = front; /*assign the temp point at front node*/
x= temp->data;
if(front==rear) //verify list having only one node then update the list is empty
{
front=NULL;
rear=NULL;
}
else
{
front = front->next;
front->previous = NULL;
}
count --; // decrease the no.of nodes in the list
delete the temp node
}
}
```

Algorithm:

```
delete_last( )
{
```

```
if(rear == NULL || front==NULL)
{
Display List is empty
}
else
{
temp = rear; /*assign the temp point at rear node*/
if(front==rear) //verify list having only one node then update the list is empty
{
front=NULL;
rear=NULL;
}
else
{
rear = rear->previous;
rear -> next = NULL;
}
x= temp ->data;
delete the temp node
count --; // decrease the no.of nodes in the list
return x;
}
}
```

Source Code:

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;

class node
{
public:
int data;
class  node *next;
class node *prev;
};

class dqueue: public node
{
  node *head,*tail;
  int top1,top2;
  public:
   dqueue()
   {
   top1=0;
```

```
 top2=0;
 head=NULL;
 tail=NULL;
 }
void push(int x){
        node *temp;
        int ch;
        if(top1+top2 >=5)
        {
          cout <<"dqueue overflow";
          return ;
        }
        if( top1+top2 == 0)
          {
            head = new node;
            head->data=x;
            head->next=NULL;
            head->prev=NULL;
            tail=head;
            top1++;
          }
         else
         {
          cout <<" Add element 1.FIRST 2.LAST\n enter ur choice:";
          cin >> ch;


          if(ch==1)
          {
            top1++;
            temp=new node;
            temp->data=x;
            temp->next=head;
            temp->prev=NULL;
            head->prev=temp;
            head=temp;
          }
          else
          {
            top2++;
            temp=new node;
            temp->data=x;
            temp->next=NULL;
            temp->prev=tail;
            tail->next=temp;
            tail=temp;
```

```
            }

          }
 }
 void pop()
 {
  int ch;
  cout <<"Delete 1.First Node 2.Last Node\n Enter ur choice:";
  cin >>ch;
  if(top1 + top2 <=0)
  {
    cout <<"\nDqueue under flow";
    return;
  }
  if(ch==1)
  {
    head=head->next;
    head->prev=NULL;
    top1--;
  }
  else
  {
    top2--;
    tail=tail->prev;
    tail->next=NULL;
  }
 }

 void display()
 {
  int ch;
  node *temp;
  cout <<"display from 1.Staring 2.Ending\n Enter ur choice";
  cin >>ch;
  if(top1+top2 <=0)
  {
    cout <<"under flow";
    return ;
  }
  if (ch==1)
  {
   temp=head;
   while(temp!=NULL)
   {
     cout << temp->data <<" ";
     temp=temp->next;
```

```
  }
 }
 else
 {
  temp=tail;
  while( temp!=NULL)
  {
    cout <<temp->data << " ";
    temp=temp->prev;
  }
 }
  }
};

main()
{
  dqueue d1;
  int ch;
  while (1){
          cout <<"1.INSERT  2.DELETE  3.DISPLAU  4.EXIT\n Enter ur choice:";
  cin >>ch;
  switch(ch)
  {
  case 1:     cout <<"enter element";
                        cin >> ch;
                        d1.push(ch); break;
  case 2: d1.pop(); break;
  case 3: d1.display(); break;
  case 4: exit(1);
  }
}}
```

OUTPUT

```
1.INSERT 2.DELETE 3.DISPLAU 4.EXIT
Enter ur choice:1
enter element4

1.INSERT 2.DELETE 3.DISPLAU 4.EXIT
Enter ur choice:1
enter element5
Add element 1.FIRST 2.LAST
enter ur choice:1
```

```
1.INSERT 2.DELETE 3.DISPLAU 4.EXIT
Enter ur choice:1
enter element6
Add element 1.FIRST 2.LAST
enter ur choice:2

1.INSERT 2.DELETE 3.DISPLAU 4.EXIT

Enter ur choice:3
display from 1.Staring 2.Ending
Enter ur choice1
5 4 6

1.INSERT 2.DELETE 3.DISPLAU 4.EXIT

Enter ur choice:2
Delete 1.First Node 2.Last Node
Enter ur choice:1

1.INSERT 2.DELETE 3.DISPLAU 4.EXIT

Enter ur choice:3
display from 1.Staring 2.Ending

Enter ur choice1
4 6

1.INSERT 2.DELETE 3.DISPLAU 4.EXIT
Enter ur choice:4
```

Viva Questions:

1. **What is the double ended queue?**
2. **What are the applications of double ended queue?**
3. **Which data structure is used for double ended queue?**
4. **List the advantages of double ended queue over single queue.**
5. **List the disadvantages of double ended queue.**

5. Write a C++ program that uses functions to perform the following:
a) Create a binary search tree of characters.
b) Traverse the above Binary search tree recursively in preorder, in order and post order.

Algorithm

Algorithm: Insert(root, ele)
{
if tree is empty then insert a node as root node otherwise go to next step
if ele is less than the root node the insert left sub tree of root node
otherwise insert right sub tree of root node
}
Algorithm: inorder(struct node *ptr)
{
if(ptr==NULL)
return;
else
{
Visit inorder(ptr->left)
diplay ptr->data
Visit inorder(ptr->right)
}
}
Algorithm: preorder(struct node *ptr)
{
if(ptr==NULL)
return;
else
{
Display ptr->data
Visit preorder(ptr->left)
Visit preorder(ptr->right)
}
}
Algorithm: postorder(ptr)
{
if(ptr==NULL)
return;
else
{
Visit postorder(ptr->left);
Visity postorder(ptr->right);
display ptr->data);
}
}

Source Code:

```
#include <iostream>
#include <deque>
#include <climits>
#include <vector>
using namespace std;

struct Tree
{
        char data;
        Tree *left;
        Tree *right;
        Tree *parent;
};

Tree* lookUp(struct Tree *node, char key)
{
        if(node == NULL) return node;

        if(node->data == key) return node;
        else {
           if(node->data < key)
                return lookUp(node->right, key) ;
           else
                return lookUp(node->left, key);
        }
}

Tree* leftMost(struct Tree *node)
{
        if(node == NULL) return NULL;
        while(node->left != NULL)
                node = node->left;
        return node;
}

struct Tree *newTreeNode(int data)
{
        Tree *node = new Tree;
        node->data = data;
        node->left = NULL;
        node->right = NULL;
        node->parent = NULL;

        return node;
}
```

```
struct Tree* insertTreeNode(struct Tree *node, int data)
{
        static Tree *p;
        Tree *retNode;

        //if(node != NULL) p = node;

        if(node == NULL)  {
           retNode = newTreeNode(data);
           retNode->parent = p;
           return retNode;
        }
        if(data <= node->data ) {
           p = node;
           node->left = insertTreeNode(node->left,data);
        }
        else {
           p = node;
           node->right = insertTreeNode(node->right,data);
        }
        return node;
}

void isBST(struct Tree *node)
{
        static int lastData = INT_MIN;
        if(node == NULL) return;

        isBST(node->left);

        /* check if the given tree is BST */
        if(lastData < node->data)
           lastData = node->data;
        else {
           cout << "Not a BST" << endl;
           return;
        }

        isBST(node->right);
        return;
}

int treeSize(struct Tree *node)
{
        if(node == NULL) return 0;
```

```
        else
            return treeSize(node->left) + 1 + treeSize(node->right);
}

int maxDepth(struct Tree *node)
{
        if(node == NULL || (node->left == NULL && node->right == NULL))
      return 0;

        int leftDepth = maxDepth(node->left);
        int rightDepth = maxDepth(node->right);

        return leftDepth > rightDepth ?
                                    leftDepth + 1 : rightDepth + 1;
}

int minDepth(struct Tree *node)
{
        if(node == NULL || (node->left == NULL && node->right == NULL))
      return 0;

        int leftDepth = minDepth(node->left);
        int rightDepth = minDepth(node->right);

        return leftDepth < rightDepth ?
                                    leftDepth + 1 : rightDepth + 1;
}

bool isBalanced(struct Tree *node)
{
        if(maxDepth(node)-minDepth(node) <= 1)
            return true;
        else
            return false;
}

/* Tree Minimum */
Tree* minTree(struct Tree *node)
{
        if(node == NULL) return NULL;
        while(node->left)
            node = node -> left;
        return node;
}

/* Tree Maximum */
```

```
Tree* maxTree(struct Tree *node)
{
        while(node->right)
           node = node -> right;
        return node;
}

/* In Order Successor - a node which has the next higher key */
Tree *succesorInOrder(struct Tree *node)
{
        /* if the node has right child, seccessor is Tree-Minimum */
        if(node->right != NULL) return minTree(node->right);

        Tree *y = node->parent;
        while(y != NULL && node == y->right) {
           node = y;
           y = y->parent;
        }
        return y;
}

/* In Order Predecessor - a node which has the next lower key */
Tree *predecessorInOrder(struct Tree *node)
{
        /* if the node has left child, predecessor is Tree-Maximum */
        if(node->left != NULL) return maxTree(node->left);

        Tree *y = node->parent;
        /* if it does not have a left child,
        predecessor is its first left ancestor */
        while(y != NULL && node == y->left) {
           node = y;
           y = y->parent;
        }
        return y;
}

void reverseOrderPrint(struct Tree *node)
{
        if(node == NULL) return;
        if(node->left == NULL && node->right == NULL) {
                cout << node->data << " ";
                return;
        }

        reverseOrderPrint(node->right);
```

```
        cout << node->data << " ";
        reverseOrderPrint(node->left);
}

Tree *lowestCommonAncestor(Tree *node, Tree *p, Tree *q)
{
        Tree *left, *right;
        if(node == NULL) return NULL;
        if(node->left == p || node->left == q
                || node->right == p || node->right == q) return node;

        left = lowestCommonAncestor(node->left,p,q);
        right = lowestCommonAncestor(node->right, p,q);
        if(left && right)
           return node;
        else
           return (left) ? left : right;
}

void clear(struct Tree *node)
{
        if(node != NULL) {
           clear(node->left);
           clear(node->right);
           delete node;
        }
}
/* print tree in order */
/* 1. Traverse the left subtree.
   2. Visit the root.
   3. Traverse the right subtree.
*/

void printTreeInOrder(struct Tree *node)
{
        if(node == NULL) return;

        printTreeInOrder(node->left);
        cout << node->data << " ";
        printTreeInOrder(node->right);
}

/* print tree in postorder*/
/* 1. Traverse the left subtree.
   2. Traverse the right subtree.
   3. Visit the root.
```

```
*/
void printTreePostOrder(struct Tree *node)
{
        if(node == NULL) return;

        printTreePostOrder(node->left);
        printTreePostOrder(node->right);
        cout << node->data << " ";
}

/* print in preorder */
/* 1. Visit the root.
   2. Traverse the left subtree.
   3. Traverse the right subtree.
*/
void printTreePreOrder(struct Tree *node)
{
        if(node == NULL) return;

        cout << node->data << " ";
        printTreePreOrder(node->left);
        printTreePreOrder(node->right);
}

/* In reverse of printTreeInOrder() */
void printTreeReverseOrder(struct Tree *node)
{
        if(node == NULL) return;
        if(node->left == NULL && node->right == NULL) {
           cout << node->data << " ";
           return;
        }

        printTreeReverseOrder(node->right);
        cout << node->data << " ";
        printTreeReverseOrder(node->left);
}
/* recursion routine to find path */
void pathFinder(struct Tree *node, int path[], int level)
{
        if(node == NULL) return;
      // save leaf node
        if(node->left == NULL && node->right == NULL) {
           path[level] = node->data;
           for(int i = 0; i <= level; i++) {
                   cout << (char)path[i];
```

```
            }
            cout << endl;
            return;
        }
    // save parent node
        path[level] = node->data;
        pathFinder(node->left, path, level+1);
        pathFinder(node->right, path, level+1);
}

bool matchTree(Tree *r1, Tree *r2)
{
        /* Nothing left in the subtree */
        if(r1 == NULL && r2 == NULL)
            return true;
        /* Big tree empty and subtree not found */
        if(r1 == NULL || r2 == NULL)
            return false;
        /* Not matching */
        if(r1->data != r2->data)
            return false;
        return (matchTree(r1->left, r2->left) &&
                                matchTree(r1->right, r2->right));
}

bool subTree(Tree *r1, Tree *r2)
{
        /*Big tree empty and subtree not found */
        if(r1 == NULL)
            return false;
        if(r1->data == r2->data)
            if(matchTree(r1, r2)) return true;
        return
      (subTree(r1->left, r2) || subTree(r1->right, r2));
}

bool isSubTree(Tree *r1, Tree *r2)
{
        /* Empty tree is subtree */
        if(r2 == NULL)
            return true;
        else
            return subTree(r1, r2);
}

/* change a tree so that the roles of the left
```

```
and right hand pointers are swapped at every node */
void mirror(Tree *r)
{
        if(r == NULL) return;

        Tree *tmp;
        mirror(r->left);
        mirror(r->right);

        /* swap pointers */
        tmp = r->right;
        r->right = r->left;
        r->left = tmp;
}

/* create a new tree from a sorted array */
Tree *addToBST(char arr[], int start, int end)
{
        if(end < start) return NULL;
        int mid = (start + end)/2;

        Tree *r = new Tree;
        r->data = arr[mid];
        r->left = addToBST(arr, start, mid-1);
        r->right = addToBST(arr, mid+1, end);
        return r;
}

Tree *createMinimalBST(char arr[], int size)
{
        return addToBST(arr,0,size-1);
}

/* Breadth first traversal using queue */
void BreadthFirstTraversal(Tree *root)
{
        if (root == NULL) return;
        deque <Tree *> queue;
        queue.push_back(root);

        while (!queue.empty()) {
           Tree *p = queue.front();
           cout << p->data << " ";
           queue.pop_front();

           if (p->left != NULL)
```

```
                queue.push_back(p->left);
            if (p->right != NULL)
                queue.push_back(p->right);
        }
        cout << endl;
}

/* get the level of a node element: root level = 0 */
int getLevel(struct Tree *node, int elm, int level)
{
        if(node == NULL) return 0;
        if(elm == node->data)
            return level;
        else if(elm < node->data)
            return getLevel(node->left, elm, level+1);
        else
            return getLevel(node->right, elm, level+1);
}

/* This code prints out all nodes at the same depth (level) */
void BreadthFirst_LevelElement_Print
        (struct Tree *root, vector<vector<int> > &v;)
{
        if(root == NULL) return;
        deque<Tree *> q;
        q.push_back(root);
        while(!q.empty()) {
            Tree *p =  q.front();
            int lev = getLevel(root, p->data, 0);
            v[lev].push_back(p->data);
            q.pop_front();
            if(p->left) q.push_back(p->left);
            if(p->right)q.push_back(p->right);
        }
        return;
}

/* levelPrint()
prints nodes at the same level
This is simpler than the BreadthFirstTraversal(root) above
It takes 2D vector with the same size of level (= MaxDepth+1)
and fills elements as we traverse (preOrder) */

void levelPrint(struct Tree *node, vector<vector<char> >&elm;, int level)
{
        if(node == NULL) return;
```

```
        // leaf nodes
        if(node->left == NULL && node->right == NULL) {
           elm[level].push_back(node->data);
           return;
        }
        // other nodes
        elm[level++].push_back(node->data);
        levelPrint(node->left, elm, level);
        levelPrint(node->right, elm, level);
}

/* find n-th max node from a tree */
void NthMax(struct Tree* t)
{
        static int n_th_max = 5;
        static int num = 0;
        if(t == NULL) return;
        NthMax(t->right);
        num++;
        if(num == n_th_max)
           cout << n_th_max << "-th maximum data is "
            << t->data << endl;
        NthMax(t->left);
}

/* Converting a BST into an Array */
void TreeToArray(struct Tree *node, int a[]){
        static int pos = 0;

        if(node){
           TreeToArray(node->left,a);
           a[pos++] = node->data;
           TreeToArray(node->right,a);
     }
}

/* Separate even/odd level elements */
/* This function is using BFS */
void level_even_odd(struct Tree *node)
{
        vector<char> evenVec, oddVec;
        if (node == NULL) return;
        deque<struct Tree*> que;
        que.push_back(node);

        while(!que.empty())
```

```
        {
            struct Tree *p = que.front();
            int level = getLevel(node, p->data, 0) ;
            // even level
            if (level % 2 == 0)
                    evenVec.push_back(p->data);
            else
                    oddVec.push_back(p->data);
            que.pop_front();
            if(p->left)  que.push_back(p->left);
            if(p->right) que.push_back(p->right);
        }

        cout << "even level elements : ";
        for(int i = 0; i < evenVec.size(); i++)
      cout << evenVec[i] << " ";
        cout << endl << "odd level elements : ";
   for(int i = 0; i < oddVec.size(); i++)
      cout << oddVec[i] << " ";
        cout << endl;
}

int main(int argc, char **argv)
{
        char ch, ch1, ch2;
        Tree *found;
        Tree *succ;
        Tree *pred;
        Tree *ancestor;
        char charArr[9]
          = {'A','B','C','D','E','F','G','H','I'};

        Tree *root = newTreeNode('F');
        insertTreeNode(root,'B');
        insertTreeNode(root,'A');
        insertTreeNode(root,'D');
        insertTreeNode(root,'C');
        insertTreeNode(root,'E');
        insertTreeNode(root,'G');
        insertTreeNode(root,'I');
        insertTreeNode(root,'H');

        /* is the tree BST? */
        isBST(root);

        /* size of tree */
```

```
        cout << "size = " << treeSize(root) << endl;

        /* max depth */
        cout << "max depth = " << maxDepth(root) << endl;

        /* min depth */
        cout << "min depth = " << minDepth(root) << endl;

        /* balanced tree? */
        if(isBalanced(root))
           cout << "This tree is balanced!\n";
        else
           cout << "This tree is not balanced!\n";

        /* min value of the tree*/
        if(root)
           cout << "Min value = " << minTree(root)->data << endl;

        /* max value of the tree*/
        if(root)
           cout << "Max value = " << maxTree(root)->data << endl;

        /* get the level of a data: root level = 0 */
        cout << "Node B is at level: " << getLevel(root, 'B', 0) << endl;
        cout << "Node H is at level: " << getLevel(root, 'H', 0) << endl;
        cout << "Node F is at level: " << getLevel(root, 'F', 0) << endl;

    /* separate even/odd level elements */
    level_even_odd(root);

        ch = 'B';
        found = lookUp(root,ch);
        if(found) {
           cout << "Min value of subtree " << ch << " as a root is "
                                << minTree(found)->data << endl;
           cout << "Max value of subtree " << ch << " as a root is "
                                << maxTree(found)->data << endl;
        }

        ch = 'B';
        found = lookUp(root,ch);
        if(found) {
           succ = succesorInOrder(found);
           if(succ)
                   cout << "In Order Successor of " << ch << " is "
                       << succesorInOrder(found)->data << endl;
```

```
    else
            cout << "In Order Successor of " << ch << " is None\n";
}

ch = 'E';
found = lookUp(root,ch);
if(found) {
    succ = succesorInOrder(found);
    if(succ)
            cout << "In Order Successor of " << ch << " is "
                    << succesorInOrder(found)->data << endl;
    else
            cout << "In Order Successor of " << ch << " is None\n";
}

ch = 'I';
found = lookUp(root,ch);
if(found) {
    succ = succesorInOrder(found);
    if(succ)
            cout << "In Order Successor of " << ch << " is "
                    << succesorInOrder(found)->data << endl;
    else
            cout << "In Order Successor of " << ch << " is None\n";
}

ch = 'B';
found = lookUp(root,ch);
if(found) {
    pred = predecessorInOrder(found);
    if(pred)
            cout << "In Order Predecessor of " << ch << " is "
                    << predecessorInOrder(found)->data << endl;
    else
            cout << "In Order Predecessor of " << ch << " is None\n";
}

ch = 'E';
found = lookUp(root,ch);
if(found) {
    pred = predecessorInOrder(found);
    if(pred)
            cout << "In Order Predecessor of " << ch << " is "
                    << predecessorInOrder(found)->data << endl;
    else
            cout << "In Order Predecessor of " << ch << " is None\n";
```

```
}

ch = 'I';
found = lookUp(root,ch);
if(found) {
   pred = predecessorInOrder(found);
   if(pred)
        cout << "In Order Predecessor of " << ch << " is "
                << predecessorInOrder(found)->data << endl;
   else
        cout << "In Order Predecessor of " << ch << " is None\n";
}

/* Lowest Common Ancestor */
ch1 = 'A';
ch2 = 'C';
ancestor =
   lowestCommonAncestor(root,
                lookUp(root,ch1), lookUp(root,ch2));
if(ancestor)
   cout << "The lowest common ancestor of " << ch1 << " and "
        << ch2 << " is " << ancestor->data << endl;

ch1 = 'E';
ch2 = 'H';
ancestor =
   lowestCommonAncestor(root,
                lookUp(root,ch1), lookUp(root,ch2));
if(ancestor)
   cout << "The lowest common ancestor of " << ch1 << " and "
        << ch2 << " is " << ancestor->data << endl;

ch1 = 'D';
ch2 = 'E';
ancestor =
   lowestCommonAncestor(root,
                lookUp(root,ch1), lookUp(root,ch2));
if(ancestor)
   cout << "The lowest common ancestor of " << ch1 << " and "
        << ch2 << " is " << ancestor->data << endl;

ch1 = 'G';
ch2 = 'I';
ancestor =
   lowestCommonAncestor(root,
                lookUp(root,ch1), lookUp(root,ch2));
```

```
if(ancestor)
   cout << "The lowest common ancestor of " << ch1 << " and "
           << ch2 << " is " << ancestor->data << endl;

ch1 = 'H';
ch2 = 'I';
ancestor =
   lowestCommonAncestor(root,
                        lookUp(root,ch1), lookUp(root,ch2));
if(ancestor)
   cout << "The lowest common ancestor of " << ch1 << " and "
           << ch2 << " is " << ancestor->data << endl;

/* print tree in order */
cout << "increasing sort order\n";
printTreeInOrder(root);
cout << endl;

/* print tree in postorder*/
cout << "post order \n";
printTreePostOrder(root);
cout << endl;

/* print tree in preorder*/
cout << "pre order \n";
printTreePreOrder(root);
cout << endl;

/* print tree in reverse order*/
cout << "reverse order \n";
printTreeReverseOrder(root);
cout << endl;

/* lookUp */
ch = 'D';
found = lookUp(root,ch);
if(found)
   cout << found->data << " is in the tree\n";
else
   cout << ch << " is not in the tree\n";

/* lookUp */
ch = 'M';
found = lookUp(root,ch);
if(found)
   cout << found->data << " is in the tree\n";
```

```
else
   cout << ch << " is not in the tree\n";

/* printing all paths :
Given a binary tree, print out all of its root-to-leaf
paths, one per line. Uses a recursive helper to do the work. */
cout << "printing paths ..." << endl;
int path[10];
pathFinder(root, path, 0);

/* find n-th maximum node */
NthMax(root);

/* Traversing level-order.
We visit every node on a level before going to a lower level.
This is also called Breadth-first traversal.*/
cout << "printing with Breadth-first traversal" << endl;
BreadthFirstTraversal(root);

/* Prints all element at the same depth (level) */
int row = maxDepth(root);
vector<vector<int> > levVec(row+1);
BreadthFirst_LevelElement_Print(root, levVec);
for(int m = 0; m < levVec.size(); m++) {
   cout << "Level at " << m << ": ";
   for(int n = 0; n < levVec[m].size(); n++)
            cout << (char)levVec[m][n] << " ";
   cout << endl;
}

/* levelPrint()
prints nodes at the same level
This is simpler than the BreadthFirstTraversal(root) above
It takes 2D vector (elm) with the same size of level (= MaxDepth+1)
and fills elements as we traverse (preOrder) */
vector<vector<char> > elm;
int mxDepth = maxDepth(root);
elm.resize(mxDepth+1);
int level = 0;
levelPrint(root, elm, level);
cout << "levelPrint() " << endl;
for(int i = 0; i <= mxDepth; i++) {
   cout << "level " << i << ": " ;
   for(int j = 0; j < elm[i].size(); j++)
            cout << elm[i][j] << " ";
   cout << endl;
```

```
        }

        /* convert the tree into an array */
        int treeSz = treeSize(root);
        int *array = new int[treeSz];
        TreeToArray(root,array);
        cout << "New array: ";
        for (int i = 0; i < treeSz; i++)
           cout << (char)array[i] << ' ';
        cout << endl;
        delete [] array;

        /* subtree */
        Tree *root2 = newTreeNode('D');
        insertTreeNode(root2,'C');
        insertTreeNode(root2,'E');
        cout << "1-2 subtree: " << isSubTree(root, root2) << endl;

        Tree *root3 = newTreeNode('B');
        insertTreeNode(root3,'A');
        insertTreeNode(root3,'D');
        insertTreeNode(root3,'C');
        insertTreeNode(root3,'E');
        cout << "1-3 subtree: " << isSubTree(root, root3) << endl;

        Tree *root4 = newTreeNode('B');
        insertTreeNode(root4,'D');
        insertTreeNode(root4,'C');
        insertTreeNode(root4,'E');
        cout << "1-4 subtree: " << isSubTree(root, root4) << endl;

        cout << "2-3 subtree: " << isSubTree(root2, root3) << endl;
        cout << "3-2 subtree: " << isSubTree(root3, root2) << endl;

        /* swap left and right */
        mirror(root);

        /* deleting a tree */
        clear(root);

        /* make a new tree with minimal depth */
        Tree *newRoot = createMinimalBST(charArr,9);

        return 0;
}
```

Output:

```
size = 9
max depth = 3
min depth = 2
This tree is balanced!
Min value = A
Max value = I
Node B is at level: 1
Node H is at level: 3
Node F is at level: 0
even level elements : F A D I
odd level elements : B G C E H
Min value of subtree B as a root is A
Max value of subtree B as a root is E
In Order Successor of B is C
In Order Successor of E is F
In Order Successor of I is None
In Order Predecessor of B is A
In Order Predecessor of E is D
In Order Predecessor of I is H
The lowest common ancestor of A and C is B
The lowest common ancestor of E and H is F
The lowest common ancestor of D and E is B
The lowest common ancestor of G and I is F
The lowest common ancestor of H and I is G
increasing sort order
A B C D E F G H I
post order
A C E D B H I G F
pre order
F B A D C E G I H
reverse order
I H G F E D C B A
D is in the tree
M is not in the tree
printing paths ...
FBA
FBDC
FBDE
FGIH
5-th maximum data is E
printing with Breadth-first traversal
F B G A D I C E H
```

```
Level at 0: F
Level at 1: B G
Level at 2: A D I
Level at 3: C E H
levelPrint()
level 0: F
level 1: B G
level 2: A D I
level 3: C E H
New array: A B C D E F G H I
New array: A B C D E F G H I
1-2 subtree: 1
1-3 subtree: 1
1-4 subtree: 0
2-3 subtree: 0
3-2 subtree: 1
```

Viva Questions:

1. What is meant by binary search tree?
2. List the properties of binary search tree.
3. List the traversals of binary search tree.
4. What are the applications of binary search tree.
5. What is the complete binary tree?
6. What is the full binary tree?

6. .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.

a) Search for a key element in a list of elements using linear search.
Algorithm:

Given a list L of n elements with values or records L_0 ... L_{n-1}, and target value T, the following subroutine uses binary search to find the index of the target T in L.

1. Set i to 0.
2. If $L_i = T$, the search terminates successfully; return i.
3. Increase i by 1.
4. If $i < n$, go to step 2. Otherwise, the search terminates unsuccessfully.

Source Code:

```
#include<iostream.h>
#include<conio.h>
void main()
{
        clrscr();
        int arr[10], i, num, n, c=0, pos;
        cout<<"Enter the array size : ";
        cin>>n;
        cout<<"Enter Array Elements : ";
        for(i=0; i<n; i++)
        {
                cin>>arr[i];
        }
        cout<<"Enter the number to be search : ";
        cin>>num;
        for(i=0; i<n; i++)
        {
                if(arr[i]==num)
                {
                        c=1;
                        pos=i+1;
                        break;
                }
        }
        if(c==0)
        {
                cout<<"Number not found..!!";
        }
```

```
        else
        {
                cout<<num<<" found at position "<<pos;
        }
        getch();
}
```

Output:

```
Enter the array size 5
Enter array elements :
4
5
9
1
4
Enter the number to search: 9
 Found at position 3
```

b) Search for a key element in a list of sorted elements using binary search.

Algorithm:

Step 1: Start
Step 2: Initialize
    low = 1
    high = n
Step 3: Perform Search
    While(low <= high)
Step 4: Obtain index of midpoint of interval
    Middle = (low + high) / 2
Step 5: Compare
    if(X < K[middle])
     high = middle - 1
    else
     print “Element found at position”
     Return(middle)
    goto: step 2
Step 6: Unsuccessful Search
    print “Element found at position”
    Return (middle)
Step 7: Stop

Source Code:

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int a[20], i, n, key, low, high, mid;
 clrscr();
 printf("Enter the array elements in ascending order");
 for(i = 0; i < n; i++)
 {
  scanf("%d", &a[i]);
 }
 printf("Enter the key element\n");
 scanf("%d", &key);
 low = 0;
 high = n - 1;
 while(high >= low)
 {
  mid = (low + high) / 2;
  if(key == a[mid])
   break;
  else
  {
   if(key > a[mid])
    low = mid + 1;
   else
    high = mid - 1;
  }
 }
 if(key == a[mid])
  printf("The key element is found at location %d", mid + 1);
 else
  printf("the key element is not found");
 getch();
}
```

Output:

Enter the size of the array 7
Enter the array elements in ascending order 23 45 68 90 100 789 890
Enter the key element 789
The key Element is found at location 6

Viva Questions:

1. What is meant by linear search?
2. What is meant by binary search?
3. What are the applications of binary search?
4. What is the time complexity of liner search?
5. What is the time complexity of binary search?

7. Write a C++ program that implements Insertion sort algorithm to arrange a list of integers in ascending order.

Algorithm:

Insertion sort is another sorting technique used when sorting arrays. Unlike bubble sort, it uses much less number of passes to sort an array. As the name suggests, sorting is done by **inserting elements in their proper order** by comparing an element with the elements on either side. Hers is how it works.
Suppose we want to sort an array A with elements A[1],A[2]....A[N]. Then,
Step 1 : A[1] by itself is trivially sorted.
Step 2 : A[2] is inserted either before or after A[1] so that A[1],A[2] are sorted.
Step 3 : A[3] is inserted into proper place in A[1],A[2],that is, before A[1], between A[1] and A[2, or after A[2], so that A[1],A[2],A[3] is sorted.
The process keeps repeating until the array is fully sorted.

Source Code:

```
#include<iostream>
#include<stdlib.h>
using namespace std;

template<class T>
void isort(T a[],int n)
{
    int i;
    for(i=1;i<n;i++)
    {
          T t=a[i];
          int j;
          for(j=i-1;j>=0&&t<a[j];j--)
          {
                              a[j+1]=a[j];
          }
          a[j+1]=t;
    }
}


void main()
{
   int a[100],i,n;
   cout<<"Enter number of elements : ";
   cin>>n;
   cout<<"Enter elements (Use Spacebar as Separator)\n";
   for(i=0;i<n;i++)
   {
```

```
            cin>>a[i];
    }
    isort(a,n);
    cout<<"After sorting the elements are\n";
    for(i=0;i<n;i++)
    {
            cout<<a[i]<<"\n";
    }
    system("pause");
```

Output:

```
Enter number of elements : 5
Enter elements (Use Spacebar as Separator)
9
7
5
45
24
After sorting the elements are

5
7
9
24
45
```

Viva Questions:

1. List the types of sorting.
2. What is the best case time complexity of insertion sort?
3. What is the average case time complexity of insertion sort?
4. What is the worst case time complexity of insertion sort?
5. Give the example to insertion sort.

8. Write a template based C++ program that implements selection sort algorithm to arrange a list of elements in descending order.

Algorithm:

Let the values be store in an array "a"
Let n = a.length
for (i = 0 ; i < n ; i ++)
{
min=a[i];
for(j=i+1;j<n;j++) //select the min of the rest of array
{
if(min>a[j]) //ascending order for descending reverse
{
minat=j; //the position of the min element
min=a[j];
}
}
2. Swap this element a[min] and a[i]
int temp=a[i] ;
a[i]=a[minat]; //swap
a[minat]=temp;
}

Source Code:

```
#include<iostream>

using namespace std;

int main()
{
   int i,j,n,loc,temp,min,a[30];
   cout<<"Enter the number of elements:";
   cin>>n;
   cout<<"\nEnter the elements\n";

   for(i=0;i<n;i++)
   {
      cin>>a[i];
   }

   for(i=0;i<n-1;i++)
   {
      min=a[i];
      loc=i;
      for(j=i+1;j<n;j++)
```

```
        {
            if(min>a[j])
            {
                min=a[j];
                loc=j;
            }
        }

        temp=a[i];
        a[i]=a[loc];
        a[loc]=temp;
    }

    cout<<"\nSorted list is as follows\n";
    for(i=0;i<n;i++)
    {
        cout<<a[i]<<" ";
    }

    return 0;
}
```

Output:

```
Enter the number of elements: 6
Enter the elements
18 3 10 7 8 1
Sorted list is as follows
1 3 7 8 10 18
```

Viva Questions:

1. What is the best case time complexity of selection sort?
2. What is the average case time complexity of selection sort?
3. What is the worst case time complexity of selection sort?
4. Give the example to selection sort.

9. Write a template based C++ program that implements Quick sort algorithm to arrange a list of elements in ascending order.

Algorithm:

QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions given array around picked pivot. There are many different versions of quickSort that pick pivot in different ways.

1) Always pick first element as pivot.
2) Always pick last element as pivot (implemented below)
3) Pick a random element as pivot.
4) Pick median as pivot.

Key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Partition Algorithm

There can be many ways to do partition, following code adopts the method given in CLRS book. The logic is simple, we start from leftmost element and keep track of index of smaller (or equal to) elements as i. While traversing, if we find a smaller element, we swap current element with arr[i]. Otherwise we ignore current element.

```
Quick_Sort(a[],left,right)
{
i=left;
j=right;
pivot=left;
while(i<j)
{
while(a[i]<=a[pivot]&&i<=right)
i++;
while(a[j]>=a[pivot]&&j>left)
j--;
if(i<j)
{
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
}
if(i>j)
{
temp=a[pivot];
a[pivot]=a[j];
a[j]=temp;
pivot=j;
}
if(left<pivot)
Quick_Sort(a,left,pivot-1);
if(right>pivot)
Quick_Sort(a,pivot+1,right);
}
```

Source Code:

```
#include<iostream>
#include<conio.h>
using namespace std;

//Function for partitioning array
int part(int low,int high,int *a)
{
    int i,h=high,l=low,p,t;  //p==pivot
    p=a[low];
    while(low<high)
    {
                while(a[l]<p)
                {
                        l++;
                }
                while(a[h]>p)
                {
                        h--;
                }
                if(l<h)
                {
                        t=a[l];
                        a[l]=a[h];
                        a[h]=t;
                }
                else
                {
                   t=p;
                   p=a[l];
                   a[l]=t;
                   break;
                }
    }
    return h;
}

void quick(int l,int h,int *a)
{
  int index,i;
  if(l<h)
  {
        index=part(l,h,a);
        quick(l,index-1,a);
        quick(index+1,h,a);
```

```
  }
}

int main()
{
    int a[100],n,l,h,i;
    cout<<"Enter number of elements:";
    cin>>n;
    cout<<"Enter the elements (Use Space As A Separator):";
    for(i=0;i<n;i++)
    cin>>a[i];
    cout<<"\nInitial Array:\n";
    for(i=0;i<n;i++)
    {
                cout<<a[i]<<"\t";
    }
    h=n-1;
    l=0;
    quick(l,h,a);
    cout<<"\nAfter Sorting:\n";
    for(i=0;i<n;i++)
    {
           cout<<a[i]<<"\t";
    }
    getch();
    return 0;
}
```

Output:

```
Enter number of elements: 6
Enter the elements : 8 6 54 2 1
Initial Array:
8 6 54 2 1
After Sorting:
1 2 4 5 6 8
```

Viva Questions:

1. What is the best case time complexity of Quick sort?
2. What is the average case time complexity of quick sort?
3. What is the worst case time complexity of quick sort?
4. What is the methodology followed by quick sort?

10. Write a C++ program that implements Heap sort algorithm for sorting a list of integers in ascending order.

Algorithm:

The heapsort algorithm consists of these steps:

1. Build the heap using all elements of the array.
2. Poll the highest element of the heap.
3. Swap it with the last element of the heap (in array).
4. Reduce the heap size by 1 (elements at the end of the heap are already sorted).
5. Repair the heap (move element swapped in 3 to its correct place in the structure).
6. If there are any elements remaining in the heap GOTO: 2.
7. Array is sorted according to the priority of the elements in reverse order.

Source Code:

```
/*
 * C++ Program to Implement Heap Sort
 */
#include <iostream>
#include <conio.h>
using namespace std;
void max_heapify(int *a, int i, int n)
{
   int j, temp;
   temp = a[i];
   j = 2*i;
   while (j <= n)
   {
      if (j < n && a[j+1] > a[j])
         j = j+1;
      if (temp > a[j])
         break;
      else if (temp <= a[j])
      {
         a[j/2] = a[j];
         j = 2*j;
      }
   }
   a[j/2] = temp;
   return;
}
void heapsort(int *a, int n)
{
   int i, temp;
```

```
    for (i = n; i >= 2; i--)
    {
       temp = a[i];
       a[i] = a[1];
       a[1] = temp;
       max_heapify(a, 1, i - 1);
    }
  }
  void build_maxheap(int *a, int n)
  {
    int i;
    for(i = n/2; i >= 1; i--)
    {
       max_heapify(a, i, n);
    }
  }
  int main()
  {
    int n, i, x;
    cout<<"enter no of elements of array\n";
    cin>>n;
    int a[20];
    for (i = 1; i <= n; i++)
    {
       cout<<"enter element"<<(i)<<endl;
       cin>>a[i];
    }
    build_maxheap(a,n);
    heapsort(a, n);
    cout<<"sorted output\n";
    for (i = 1; i <= n; i++)
    {
       cout<<a[i]<<endl;
    }
    getch();
  }
```

Output

```
enter no of elements of array
7
enter element1
34
enter element2
45
enter element3
12
enter element4
```

40
enter element5
6
enter element6
75
enter element7
36
sorted output
6
12
34
36
40
45
75

Viva Questions:

1. What is meant by heap sort?
2. What is the best case time complexity of Heap sort?
3. What is the average case time complexity of heap sort?
4. What is the worst case time complexity of heap sort?
5. List the applications of heap sort.

11. Write a C++ program that implements Merge sort algorithm for sorting a list of integers in ascending order

Algorithm:

Merge sort is based on the divide – and – conquer paradigm. Its worst - case running time has a lower order of growth than insertion sort. Since we are dealing with sub - problems, we state each sub - problem as sorting a sub – array A [pr].

Initially, p= 1 and r=n , but these values change as we recurse through sub-problems.

To sort A[p..r]:

1.Divide Step

If a given array A has zero or one element, simply return; it is already sorted. Otherwise, split A[p..r] into two sub-arrays A[p..q] and A[q+ 1 ..r], each containing about half of the elements Of A[p..r]. That is, q is the halfway point of A[p..r].

2.Conquer Step Conquer by recursively sorting the two sub-arrays A[p..q] and A [q+ 1 ..r].

3. Combine Step Combine the elements back in A [p..r] by merging the two sorted sub-arrays A[p..q] and A[q+ 1 ..r] into a sorted sequence. To accomplish this step, we will define a procedure MERGE (A,p,q,r).

Note that the recursion bottoms out when the sub -array has just one element, so that it is trivially sorted. To sort the entire sequence A[1 .. n], make the initial call to the procedure MERGE-SORT (A, 1,n).MERGE-SORT (A,p,r)

```
1.IFp<r                          // Check for base case
2.THENq= FLOOR[(p+r)/2]          // Divide step
3.MERGE (A,p,q)                  // Conquer step.
4.MERGE (A,q+ 1,r)               // Conquer step.
5.MERGE (A,p,q,r)                // Conquer step
```

Source Code:

```
#include<iostream>
#include<stdlib.h>
using namespace std;

template<class T>
void m_sort(T nos[],T t[],int l,int r);
template<class T>
void mergesort(T nos[],T t[],int asize)
{
    m_sort<T>(nos,t,0,asize-1);
}

template<class T>
void m_sort(T nos[],T t[],int l,int r)
{
    int mid;
    if(r>l)
    {
```

```
            mid=(r+l)/2;
            m_sort(nos,t,l,mid);
            m_sort(nos,t,mid+1,r);
            merge(nos,t,l,mid+1,r);
    }
}

template<class T>
void merge(T nos[],T t[],int l,int mid,int r)
{
    int i,l_end,num_elements,t_pos;
       l_end=mid-1;
       t_pos=l;
       num_elements=r-l+1;
       while((l<=l_end)&&(mid<=r))
       {
                              if(nos[l]<=nos[mid])//(kept -nos instead )
                              {
                                                    t[t_pos]=nos[l];
                                                    t_pos++;
                                                    l++;
                              }
                              else
                              {
                                                    t[t_pos]=nos[mid];
                                                    t_pos++;
                                                    mid++;
                              }
       }
       while(l<=l_end)
       {
                     t[t_pos]=nos[l];
                     l++;
                     t_pos++;
       }
       while(mid<=r)
       {
                  t[t_pos]=nos[mid];
                  mid++;
                  t_pos++;
       }
       for(i=0;i<=num_elements;i++)
       {
                           nos[r]=t[r];
                           r--;
       }
```

```
}

int main()
{
   int a[20],b[20],n;
   cout<<"\nEnter The Total Number Of Elements:";
   cin>>n;
   cout<<"\nEnter "<<n<<" elements (Use Spacebar as separator) : ";
   for(int i=0;i<n;i++)
   {
            cin>>a[i];
   }

   mergesort(a,b,n);

   cout<<"\nThe sorted element list is \n";
   for(int i=0;i<n;i++)
   {
            cout<<a[i]<<"\t";
   }

   system("pause");
   return 0;
}
```

Output:

```
Enter the total number of elements 6
Enter 6 elements: 87 33 53 123 5 44
The sorted elements list is
5       33      44      53      87      123
```

Viva Questions:

1. What is meant by merge sort?
2. What is the best case time complexity of merge sort?
3. What is the average case time complexity of merge sort?
4. What is the worst case time complexity of merge sort?
5. What is the methodology followed by merge sort?

12. Write a C++ program to implement all the functions of a dictionary (ADT) using hashing.
Algorithm:
Dictionary : A Dictionary is a collection of pairs of the form (K,V). where 'K' is a Key and 'V' is the value associated with the key 'k'. No two pairs in the dictionary have a same key.
The Following operations performed on a dictionary :

1. Determine whether or not the dictionary is empty
2. Determine the dictionary Size (No of Pairs)
3. Find the pair with a specified key
4. Insert a pair into the dictionary
5. Delete or erase the pair into the specified key

ABSTRACT DATA TYPE :
INSTANCES : Collection of pairs with distinct keys
OPERATIONS :
Empty() : return true iff the dictionary is empty
Size() : return the no. of pairs in the dictionary
Find(k) : return the pair with the key , K.
Insert(p) : insert the pair 'P' into the dictionary
Erase(k) : delete the pair with key 'K'.

Source Code:

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;
# define max 10
typedef struct list
{
   int data;
   struct list *next;
} node_type;
node_type *ptr[max],*root[max],*temp[max];
class Dictionary
{
public:
   int index;
   Dictionary();
   void insert(int);
   void search(int);
   void delete_ele(int);
};
Dictionary::Dictionary()
{
   index=-1;
   for(int i=0; i<max; i++)
   {
```

```
        root[i]=NULL;
        ptr[i]=NULL;
        temp[i]=NULL;
    }
}
void Dictionary::insert(int key)
{
    index=int(key%max);
    ptr[index]=(node_type*)malloc(sizeof(node_type));
    ptr[index]->data=key;
    if(root[index]==NULL)
    {
        root[index]=ptr[index];
        root[index]->next=NULL;
        temp[index]=ptr[index];
    }
    else
    {
        temp[index]=root[index];
        while(temp[index]->next!=NULL)
            temp[index]=temp[index]->next;
        temp[index]->next=ptr[index];
    }
}
void Dictionary::search(int key)
{
    int flag=0;
    index=int(key%max);
    temp[index]=root[index];
    while(temp[index]!=NULL)
    {
        if(temp[index]->data==key)
        {
            cout<<"\nSearch key is found!!";
            flag=1;
            break;
        }
        else temp[index]=temp[index]->next;
    }
    if (flag==0)
        cout<<"\nsearch key not found.....";
}
void Dictionary::delete_ele(int key)
{
    index=int(key%max);
    temp[index]=root[index];
```

```
   while(temp[index]->data!=key && temp[index]!=NULL)
   {
      ptr[index]=temp[index];
      temp[index]=temp[index]->next;
   }
   ptr[index]->next=temp[index]->next;
   cout<<"\n"<<temp[index]->data<<" has been deleted.";
   temp[index]->data=-1;
   temp[index]=NULL;
   free(temp[index]);
}
main()
{
   int val,ch,n,num;
   char c;
   Dictionary d;
   do
   {
      cout<<"\nMENU:\n1.Create";
      cout<<"\n2.Search for a value\n3.Delete an value";
      cout<<"\nEnter your choice:";
      cin>>ch;
      switch(ch)
      {
      case 1:
         cout<<"\nEnter the number of elements to be inserted:";
         cin>>n;
         cout<<"\nEnter the elements to be inserted:";
         for(int i=0; i<n; i++)
         {
            cin>>num;
            d.insert(num);
         }
         break;
      case 2:
         cout<<"\nEnter the element to be searched:";
         cin>>n;
         d.search(n);
      case 3:
         cout<<"\nEnter the element to be deleted:";
         cin>>n;
         d.delete_ele(n);
         break;
      default:
         cout<<"\nInvalid Choice.";
      }
```

```
        cout<<"\nEnter y to Continue:";
        cin>>c;
    }
    while(c=='y');
    getch();
}
```

OUTPUT:

```
MENU:
1.Create
2.Search for a value
3.Delete an value
Enter your choice:1
Enter the number of elements to be inserted:3
Enter the elements to be inserted:12
23
56
 Enter y to Continue:y
 MENU:
1.Create
2.Search for a value
3.Delete an value
Enter your choice:2
 Enter the element to be searched:23
 Search key is found!!
Enter the element to be deleted:23
 23 has been deleted.
Enter y to Continue:y

MENU:
1.Create
2.Search for a value
3.Delete an value
Enter your choice:
```

Viva Questions:

1. What is hashing?
2. What is dictionary ADT?
3. List the operations performed on dictionary ADT.

13. Write a C++ program that implements Radix sort algorithm for sorting a list of integers in ascending order

Algorithm:

1. Find the length of the number that has maximum number of digits.
2. Initialize i=0, Repeat the below procedure till the length equals i.
3. Fill the bucket with all the digits in ith position.
4. Sort out the digits according to the order.
5. If length=i, i=i*10, goto to step 3. Else go to step 5
6. Print the sorted array.

Source Code:

```
#include <algorithm>
#include <iostream>
#include <iterator>

// Radix sort comparator for 32-bit two's complement integers
class radix_test
{
   const int bit; // bit position [0..31] to examine
public:
   radix_test(int offset) : bit(offset) {} // constructor

   bool operator()(int value) const // function call operator
   {
      if (bit == 31) // sign bit
         return value < 0; // negative int to left partition
      else
         return !(value & (1 << bit)); // 0 bit to left partition
   }
};

// Least significant digit radix sort
void lsd_radix_sort(int *first, int *last)
{
   for (int lsb = 0; lsb < 32; ++lsb) // least-significant-bit
   {
      std::stable_partition(first, last, radix_test(lsb));
   }
}

// Most significant digit radix sort (recursive)
void msd_radix_sort(int *first, int *last, int msb = 31)
{
```

```
    if (first != last && msb >= 0)
    {
        int *mid = std::partition(first, last, radix_test(msb));
        msb--; // decrement most-significant-bit
        msd_radix_sort(first, mid, msb); // sort left partition
        msd_radix_sort(mid, last, msb); // sort right partition
    }
}

// test radix_sort
int main()
{
    int data[] = { 170, 45, 75, -90, -802, 24, 2, 66 };

    lsd_radix_sort(data, data + 8);
    // msd_radix_sort(data, data + 8);

    std::copy(data, data + 8, std::ostream_iterator<int>(std::cout, " "));

    return 0;
}
```

Output:

-802 -90 2 24 45 66 75 170

Viva Questions:

1. What is meant by radix sort?
2. What is the best case time complexity of radix sort?
3. What is the average case time complexity of radix sort?
4. What is the worst case time complexity of radix sort?
5. Give the examples to radix sort?

14. 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 non recursively in inorder.

Algorithm: Insert(root, ele)
{
if tree is empty then insert a node as root node otherwise go to next step
if ele is less than the root node the insert left sub tree of root node
otherwise insert right sub tree of root node
}
Algorithm inorder(ptr)
{
Initially top=-1;
if(ptr==NULL)
return;
else
{
while(ptr!=NULL||top>=0)
{
if(ptr!=NULL)
{
top++;//push the value in stack
stack[top]=ptr;
ptr=ptr->left;
}
else
{
ptr=stack[top];
display ptr->data
top--; //pop the stack
ptr=ptr->right;
}
}
}
}

Source Code:

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

// Node class
class Node {
   int key;
   Node* left;
   Node* right;
public:
   Node() { key=-1; left=NULL; right=NULL; };
```

```
    void setKey(int aKey) { key = aKey; };
    void setLeft(Node* aLeft) { left = aLeft; };
    void setRight(Node* aRight) { right = aRight; };
    int Key() { return key; };
    Node* Left() { return left; };
    Node* Right() { return right; };
};

// Tree class
class Tree {
    Node* root;
public:
    Tree();
    ~Tree();
    Node* Root() { return root; };
    void addNode(int key);
    void inOrder(Node* n);
    void preOrder(Node* n);
    void postOrder(Node* n);
private:
    void addNode(int key, Node* leaf);
    void freeNode(Node* leaf);
};

// Constructor
Tree::Tree() {
    root = NULL;
}

// Destructor
Tree::~Tree() {
    freeNode(root);
}

// Free the node
void Tree::freeNode(Node* leaf)
{
   if ( leaf != NULL )
   {
      freeNode(leaf->Left());
      freeNode(leaf->Right());
      delete leaf;
   }
}

// Add a node
```

```
void Tree::addNode(int key) {
    // No elements. Add the root
    if ( root == NULL ) {
       cout << "add root node ... " << key << endl;
       Node* n = new Node();
       n->setKey(key);
       root = n;
    }
    else {
      cout << "add other node ... " << key << endl;
      addNode(key, root);
    }
}

// Add a node (private)
void Tree::addNode(int key, Node* leaf) {
   if ( key <= leaf->Key() ) {
      if ( leaf->Left() != NULL )
         addNode(key, leaf->Left());
      else {
         Node* n = new Node();
         n->setKey(key);
         leaf->setLeft(n);
      }
   }
   else {
      if ( leaf->Right() != NULL )
         addNode(key, leaf->Right());
      else {
         Node* n = new Node();
         n->setKey(key);
         leaf->setRight(n);
      }
   }
}

// Print the tree in-order
// Traverse the left sub-tree, root, right sub-tree
void Tree::inOrder(Node* n) {
   if ( n ) {
      inOrder(n->Left());
      cout << n->Key() << " ";
      inOrder(n->Right());
   }
}
```

```
// Print the tree pre-order
// Traverse the root, left sub-tree, right sub-tree
void Tree::preOrder(Node* n) {
   if ( n ) {
      cout << n->Key() << " ";
      preOrder(n->Left());
      preOrder(n->Right());
   }
}

// Print the tree post-order
// Traverse left sub-tree, right sub-tree, root
void Tree::postOrder(Node* n) {
   if ( n ) {
      postOrder(n->Left());
      postOrder(n->Right());
      cout << n->Key() << " ";
   }
}

// Test main program
int main() {
  Tree* tree = new Tree();
  tree->addNode(30);
  tree->addNode(10);
  tree->addNode(20);
  tree->addNode(40);
  tree->addNode(50);
  cout << "In order traversal" << endl;
  tree->inOrder(tree->Root());
  cout << endl;
  cout << "Pre order traversal" << endl;
  tree->preOrder(tree->Root());
  cout << endl;

  cout << "Post order traversal" << endl;
  tree->postOrder(tree->Root());
  cout << endl;
  delete tree;
  return 0;
}
.
```

OUTPUT:-

```
add root node ... 30
add other node ... 10
add other node ... 20
add other node ... 40
add other node ... 50
In order traversal
10 20 30 40 50
Pre order traversal
30 10 20 40 50
Post order traversal
20 10 50 40 30
```

15. Write a C++ program that uses functions to perform the following:
a) Create a binary search tree of integers.
b) Search for an integer key in the above binary search tree non recursively.
c) Search for an integer key in the above binary search tree recursively.

Algorithm:
Algorithm: Insert(root, ele)
{
if tree is empty then insert a node as root node otherwise go to next step
if ele is less than the root node the insert left sub tree of root node
otherwise insert right sub tree of root node
}

Algorithm: Search for the element
Here k is the key that is searched for and x is the start node.
BST -Search(x,k)
1:y<-x
2:whiley!=nil
do
3:if key[y] =k
Then return y
4:else if key[y]< k
then y <-right[y]
5:else y<- left [y]
6: return (“\NOT FOUND")

Source Code:

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;

void insert(int,int );
void display(int);
int search(int);
int search1(int,int);
int tree[40],t=1,s,x,i;

main()
{
        int ch,y;
        for(i=1;i<40;i++)
        tree[i]=-1;
        while(1)
        {
```

```
cout <<"1.INSERT\n2.DELETE\n3.DISPLAY\n4.SEARCH\n5.EXIT\nEnter your choice:";
                cin >> ch;
                switch(ch)
                {
                case 1:
                        cout <<"enter the element to insert";
                        cin >> ch;
                        insert(1,ch);
                        break;
                case 2: display(1);
                        cout<<"\n";
                        for(int i=0;i<=32;i++)
                        cout <<i;
                        cout <<"\n";
                        break;
        case 3:
                        cout <<"enter the element to search:";
                        cin >> x;
                        y=search(1);
                        if(y == -1) cout <<"no such element in tree";
                        else cout <<x << "is in" <<y <<"position";
                        break;
                case 5:
                        exit(0);
                }
        }
}

void insert(int s,int ch )
{
        int x;
        if(t==1)
        {
                tree[t++]=ch;
                return;
        }
        x=search1(s,ch);
        if(tree[x]>ch)
                tree[2*x]=ch;
        else
                tree[2*x+1]=ch;
        t++;
}

int search(int s)
{
```

```
if(t==1)
{
cout <<"no element in tree";
return -1;
}
if(tree[s]==-1)
return tree[s];
if(tree[s]>x)
search(2*s);
else if(tree[s]<x)
search(2*s+1);
else
return s;
}

void display(int s)
{
if(t==1)
{cout <<"no element in tree:";
return;}
for(int i=1;i<40;i++)
if(tree[i]==-1)
cout <<" ";
else cout <<tree[i];
return ;
}

int search1(int s,int ch)
{
if(t==1)
{
cout <<"no element in tree";
return -1;
}
if(tree[s]==-1)
return s/2;
if(tree[s] > ch)
search1(2*s,ch);
else search1(2*s+1,ch);
}
```

OUTPUT

1.INSERT
2.DELETE
3.DISPLAY
4.SEARCH

```
5.EXIT
Enter your choice:3

no element in tree:
01234567890111213141516171819202122232425262728293031 32

1.INSERT
2.DELETE
3.DISPLAY
4.SEARCH
5.EXIT
Enter your choice:1

Enter the element to insert 10
1.INSERT
2.DELETE
3.DISPLAY
4.SEARCH
5.EXIT
Enter your choice:4

Enter the element to search: 10
10 is in 1 position
1.INSERT
2.DELETE
3.DISPLAY
4.SEARCH
5.EXIT

Enter your choice:5
```