ANNAMALAI UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

B.E. [Computer Science and Engineering]

III – SEMESTER

CSCP308 - Data Structures and Algorithms Lab Manual

Annamalai University
Faculty of Engineering and Technology
Department of Computer Science and Engineering
B. E. (CSE) – III Semester

CSCP308 - Data Structures and Algorithms Lab

List of Exercises

1. Write a C++ program to search a key element from the given list by sequential search technique.
2. Write a C++ program to search a key element from the given list by binary search technique.
3. Write a C++ program to sort the given list by selection sort technique.
4. Write a C++ program to sort the given list by bubble sort technique.
5. Write a C++ program to sort the given list by merge sort technique using divide and conquer method.
6. Write a C++ program to sort the given list by quick sort technique using divide and conquer method.
7. Write a C++ program to implement a stack data structure.
8. Write a C++ program to implement a queue data structure.
9. Write a C++ program to implement a single linked list data structure.
10. Write a C++ program to create a binary search tree and perform in-order, pre-order and post-order traversals.
11. Write a C++ program to traverse a graph data structure using depth first searching algorithm.
12. Write a C++ program to traverse a graph using breadth first searching algorithm.

A1 - Batch – Dr. R. Ragupathy

A2 - Batch – Dr. N.J.Nalini

B1- Batch – Dr. R. Priya

B2 – Batch – Dr. S. G. Santhi

Signature of staff-in-charge

Ex. No. 1
Date:

Sequential Search

Aim:

To write a C++ program to search a key element from the given list by sequential search technique.

Procedure:

Member variables in the class Sequential

n and key as integer
a as array of integer of size 100

Member functions in the class Sequential

void GetData(void)	- To get the input from user
void SequentialSearch(void)	- To search the key in the given array a using sequential search technique

Algorithm

void GetData(void)

1. Get the list of numbers.
2. Get the Key value to be searched.

void SequentialSearch(void)

1. Compare the key value with the numbers in the list from the first position.
2. If the key value is present in the list, then print key value and its position.
3. If the key value is not present in any position of the list then, print "The key element is not present in the list".

int main()

1. Create an object S of class Sequential.
2. Call member function GetData() of object S.
3. Call member function SequentialSearch() of object S.

Source Code:

```
// Sequential Search
#include <iostream.h>
#include <conio.h>
class Sequential
{
 int a[100],n,key;
 public:
        void GetData(void);
        void SequentialSearch(void);
};
```

```
void Sequential::GetData(void)
{
 cout<<"\nEnter the size of the list to be searched : ";
 cin>>n;
 cout<<"\nEnter the elements of the list : ";
 for(int i=0;i<n;i++)
 {
  cin>>a[i];
 }
 cout<<"Enter the key value to be searched in the above list : ";
 cin>>key;
}

void Sequential::SequentialSearch(void)
{
 for(int i=0;i<n;i++)
 {
  if(a[i]==key)
  {
   cout<<"The key value "<<key<<" is present at location "<<i+1;
   return;
  }
 }
cout<<"The key value "<<key<<" is not present in the above list";
}

int main()
{
 Sequential S;
 clrscr();
 S.GetData();
 S.SequentialSearch();
 getch();
 return 0;
}
```

Sample Input/Output:

Enter the size of the list to be searched : 11

Enter the elements of the list : -12 23 1 -15 3 5 2 4 4 66 -11

Enter the key value to be searched in the above list : 3

The key value 3 is present at location 5

Result:

Thus, a C++ program to search a key element from the given list by sequential search technique has been successfully written and outputs have been verified.

Ex. No. 2
Date:

Binary Search

Aim:

To write a C++ program to search a key element from the given list by binary search technique.

Procedure:

Member variables in the class Binary

n and key as integer
a as array of integer of size 100

Member functions in the class Binary

void GetData(void)	- To get the input from user
void Sort(void)	- To sort the input in ascending order
void Display(void)	- To display all elements of the list
void BinarySearch(void)	- To search the key in the given array a using binary search technique

Algorithm

void GetData(void)

1. Get the list of numbers.
2. Get the Key value to be searched.

void Sort(void)

1. for i=0 to n-1 do step 2
2. for j=i+1 to n do step 3
3. if(a[i]>a[j]) exchange a[i] and a[j]

void Display(void)

1. Display elements of the list one by one.

void BinarySearch(void)

1. Assign:
   high= total number of elements in the list -1.
   low=0.
2. until low <= high repeat the following steps,
   a. mid= (high + low)/2.
   b. Check the key value with the value present at the middle index position of the sorted list.
   c. If present, display its position in the list and stop the process.
   d. Else if key value < the value present in the middle position, then
      i. high=mid+1;
   e. Else if key value > the value present in the middle position, then
      i. low=mid+1;
3. Display “The key value is not present in the list”.

int main()

1. Create an object B of class Binary.
2. Call member function GetData() of object B.
3. Call member function Sort() of object B.
4. Call member function Display() of object B.
5. Call member function BinarySearch() of object B.

Source Code:

```
// Binary Search
#include <iostream.h>
#include <conio.h>
class Binary
{
 int a[100],n,key;
 public:
        void GetData(void);
        void Sort(void);
        void Display(void);
        void BinarySearch(void);
};

void Binary::GetData(void)
{
 cout<<"\nEnter the size of the list to be searched : ";
 cin>>n;
 cout<<"\nEnter the elements of the list :";
 for(int i=0;i<n;i++)
 {
  cin>>a[i];
 }
 cout<<"Enter the key value to be searched in the above list : ";
 cin>>key;
}

void Binary::Sort(void)
{
 for(int i=0;i<n-1;i++)
  for(int j=i+1;j<n;j++)
  {
   if(a[i]>a[j])
   {
    int temp=a[i];
    a[i]=a[j];
    a[j]=temp;
   }
  }
}
```

```
void Binary::Display(void)
{
for(int i=0;i<n;i++)
   cout<<" Element "<< i+1 <<" is " << a[i]<<endl;
}

void Binary::BinarySearch(void)
{
 int high,low,mid;
 high=n-1;
 low=0;
 while (low<=high)
 {
  mid=(low+high)/2;
  if(a[mid]==key)
  {
   cout<<"\n The key value "<< key <<" is present at location " <<mid+1;
   return;
  }
  else if (key<a[mid])
  {
   high=mid-1;
  }
  else
  {
   low=mid+1;
  }
 }
 cout<<"\n The key value "<< key<<" is not present in the above list";
}

int main()
{
  Binary B;
  clrscr();
  B.GetData();
  B.Sort();
  cout<<"\n The Sorted List \n";
  cout<<"  ---------------- \n";
  B.Display();
  B.BinarySearch();
  getch();
  return 0;
}
```

Sample Input/Output:

Enter the size of the list to be searched : 11

Enter the elements of the list. : -12 23 1 -15 3 5 2 4 4 66 -11

Enter the key value to be searched in the above list : 3

```
The Sorted List
-----------------
Element  1 is -15.
Element  2 is -12.
Element  3 is -11.
Element  4 is  1.
Element  5 is  2.
Element  6 is  3.
Element  7 is  4.
Element  8 is  4.
Element  9 is  5.
Element 10 is 23.
Element 11 is 66.
```

The key value 3 is present at location 6.

Result:

Thus, a C++ program to search a key element from the given list by binary search technique has been successfully written and outputs have been verified.

Ex. No. 3
Date:

Selection Sort

Aim:

To write a C++ program to sort the given list by selection sort technique.

Procedure:

Create the base class DataArray with the following member variables and functions.

Member variables in the base class DataArray

n as integer
a as array of integer of size 100

Member functions in the base class DataArray

void GetData(void) - To get the input from user
void Display(void) - To display elements of the list

Algorithm

void GetData(void)

1. Get the list of numbers.

void Display(void)

1. Display all elements of the list one by one.

Create a derived class Selection, derived from the base class DataArray with public access.

Member variable in the derived class Selection is none.

Member functions in the derived class Selection

void SelectionSort(void) - To sort elements in ascending order

Algorithm

void SelectionSort(void)

1. Find the largest number and place it in last position in the list.
2. Consider rest of the elements only as new list and repeat the step 1 until there is only one element in the list.

int main()

1. Create an object S of class Selection.
2. Call member function GetData() of object S.
3. Call member function SelectionSort() of object S.
4. Call member function Display() of object S.

Source Code:

```
// Selection sorting.
#include<iostream.h>
#include<conio.h>

class DataArray
{
 protected:
        int a[100],n;
 public:
        void GetData(void);
        void Display(void);
};

void DataArray::GetData(void)
{
 cout<<"\n Enter the number of elements to be sorted : ";
 cin>>n;
 for(int i=0;i<n;i++)
 {
  cout<<" Enter element no. "<<i+1<< " :";
  cin>>a[i];
 }
}

void DataArray::Display(void)
{
 for(int i=0;i<n;i++)
   cout<<"  "<<a[i];
 cout<<endl;
}

class Selection : public DataArray
{
public:
        void SelectionSort(void);
};

void Selection::SelectionSort(void)
{
 int large,index,i,j;
 for(i=n-1;i>0;i--)
 {
  large=a[0];
  index=0;
  for(j=1;j<=i;j++)
  {
   if(a[j]>large)
```

```
  {
   large=a[j];
   index=j;
   }
 }
 a[index]=a[i];
 a[i]=large;
 cout<<"\n Array after iteration "<<n-i<<": ";
 Display();
 }
}

int main()
{
 Selection S;
 clrscr();
 S.GetData();
 S.SelectionSort();
 cout<<"\n\t The Sorted Array";
 cout<<"\n\t ~~~~~~~~~~~~~~~~\n";
 S.Display();
 getch();
 return 0;
}
```

Sample Input/output:

```
Enter the number of elements to be sorted : 10
Enter element no. 1 : 19
Enter element no. 2 : 77
Enter element no. 3 : -60
Enter element no. 4 : -12
Enter element no. 5 : 7
Enter element no. 6 : 41
Enter element no. 7 : 2
Enter element no. 8 : 1
Enter element no. 9 :99
Enter element no. 10 :0

Array after iteration 1:  19  77  -60  -12  7  41  2  1  0  99

Array after iteration 2:  19  0  -60  -12  7  41  2  1  77  99

Array after iteration 3:  19  0  -60  -12  7  1  2  41  77  99

Array after iteration 4:  2  0  -60  -12  7  1  19  41  77  99

Array after iteration 5:  2  0  -60  -12  1  7  19  41  77  99

Array after iteration 6:  1  0  -60  -12  2  7  19  41  77  99
```

```
Array after iteration 7:  -12  0  -60  1  2  7  19  41  77  99

Array after iteration 8:  -12  -60  0  1  2  7  19  41  77  99

Array after iteration 9:  -60  -12  0  1  2  7  19  41  77  99


      The Sorted Array
      ~~~~~~~~~~~~
 -60  -12  0  1  2  7  19  41  77  99
```

Result:

Thus, a C++ program to sort the given list by selection sort technique has been successfully written and outputs have been verified.

Ex. No. 4
Date:

Bubble Sort

Aim:

To write a C++ program to sort the given list by bubble sort technique.

Procedure:

Create the base class DataArray with the following member variables and functions.

Member variables in the base class DataArray

n as integer
a as array of integer of size 100

Member functions in the base class DataArray

void GetData(void) - To get the input from user
void Display(void) - To display elements of the list

Algorithm

void GetData(void)

1. Get the list of numbers.

void Display(void)

1. Display all elements of the list one by one.

Create a derived class Bubble, derived from the base class DataArray with public access.

Member variable in the derived class Bubble is none.

Member functions in the derived class Bubble

void BubbleSort(void) - To sort elements in ascending order

Algorithm

void BubbleSort(void)

1. Compare pairs of adjacent elements from the first location towards right for the entire list with the size of the list decreasing by one for each repetition.
2. For each comparison, if the first element is greater than the second element then interchange its position.
3. Repeat the above steps 1 and 2 until no interchanges has happened during any comparison from left to right

int main()

1. Create an object B of class Bubble.
2. Call member function GetData() of object B.
3. Call member function BubbleSort() of object B.
4. Call member function Display() of object B.

Source Code:

```
// Bubble sorting.
#include<iostream.h>
#include<conio.h>
class DataArray
{
 protected:
        int a[100],n;
 public:
        void GetData(void);
        void Display(void);
};

void DataArray::GetData(void)
{
 cout<<"\n Enter the number of elements to be sorted : ";
 cin>>n;
 for(int i=0;i<n;i++)
 {
  cout<<" Enter element no. "<<i+1<< " :";
  cin>>a[i];
 }
}

void DataArray::Display(void)
{
 for(int i=0;i<n;i++)
   cout<<"  "<<a[i];
 cout<<endl;
}

class Bubble : public DataArray
{
public:
        void BubbleSort(void);
};

void Bubble::BubbleSort(void)
{
 int flag=1,i,j,temp;
 for(i=0;i<n-1;i++)
 {
  for(j=0;j<n-i-1;j++)
  if(a[j]>a[j+1])
  {
   temp=a[j];
   a[j]=a[j+1];
   a[j+1]=temp;
```

```
  flag=0;
 }
 cout<<"\n Array after iteration "<<i<<": ";
 Display();
 if(flag)
  break;
 else
  flag=1;
 }
}

int main()
{
 Bubble B;
 clrscr();
 B.GetData();
 B.BubbleSort();
 cout<<"\n\t The Sorted Array";
 cout<<"\n\t ~~~~~~~~~~~~~~~~\n";
 B.Display();
 getch();
 return 0;
}
```

Sample Input/output:

```
Enter the number of elements to be sorted : 10
Enter element no. 1 : 19
Enter element no. 2 : 77
Enter element no. 3 : -60
Enter element no. 4 : -12
Enter element no. 5 : 7
Enter element no. 6 : 41
Enter element no. 7 : 2
Enter element no. 8 : 1
Enter element no. 9 :99
Enter element no. 10 :0

Array after iteration 0:   19  -60  -12  7  41  2  1  77  0  99

Array after iteration 1:   -60  -12  7  19  2  1  41  0  77  99

Array after iteration 2:   -60  -12  7  2  1  19  0  41  77  99

Array after iteration 3:   -60  -12  2  1  7  0  19  41  77  99

Array after iteration 4:   -60  -12  1  2  0  7  19  41  77  99

Array after iteration 5:   -60  -12  1  0  2  7  19  41  77  99
```

```
Array after iteration 6:  -60  -12  0  1  2  7  19  41  77  99

Array after iteration 7:  -60  -12  0  1  2  7  19  41  77  99

      The Sorted Array
      ~~~~~~~~~~~~
 -60  -12  0  1  2  7  19  41  77  99
```

Result:

Thus, a C++ program to sort the given list by bubble sort technique has been successfully written and outputs have been verified.

Ex. No. 5
Date:

Merge Sort

Aim:

To write a C++ program to sort the given list by merge sort technique using divide and conquer method.

Procedure:

Create the base class DataArray with the following member variables and functions.

Member variables in the base class DataArray

n as integer
a as array of integer of size 100

Member functions in the base class DataArray

int Getn(void)	- To return value of n
void GetData(void)	- To get the input from user
void Display(void)	- To display elements of the list

Algorithm

void GetData(void)

1. Get the list of numbers.

void Display(void)

1. Display all elements of the list one by one.

Create a derived class Merging, derived from the base class DataArray with public access.

Member variable in the derived class Merging is none.

Member functions in the derived class Merging

void Merge(int low, int high, int mid)	- To merge two lists
void MergeSort(int low, int high)	- To divide the list into two part
void MergeDisplay(int l,int h)	- To Display elements from l to h

Algorithm

void Merge(int low, int high, int mid)

1. Merge the recently divided two lists by comparing its value and arrange it in the ascending order.
2. Repeat the step 2 until the entire divided list is merged in to a single sorted list.

void MergeSort(int low, int high)

1. Repeatedly divide the given list of elements into two parts until each part cannot be divided further, that is up to one element in the list.

void MergeDisplay(int l,int h)

1. Display elements from l to h in the list one by one.

int main()

1. Create an object M of class Merging.
2. Call member function GetData() of object M.
3. Call member function MergeSort() of object M.
4. Call member function Display() of object M.

Source Code:

```
// Merge sorting.
#include<iostream.h>
#include<conio.h>
class DataArray
{
 protected:
        int a[100],n;
 public:
        int Getn(void) { return n; }
        void GetData(void);
        void Display(void);
};

void DataArray::GetData(void)
{
 cout<<"\n Enter the number of elements to be sorted : ";
 cin>>n;
 for(int i=0;i<n;i++)
 {
  cout<<" Enter element no. "<<i+1<< " :";
  cin>>a[i];
 }
}

void DataArray::Display(void)
{
 for(int i=0;i<n;i++)
   cout<<"  "<<a[i];
 cout<<endl;
}

class Merging : public DataArray
{
public:
        void Merge(int low, int high, int mid);
        void MergeSort(int low, int high);
        void MergeDisplay(int l,int h);
};
```

```
void Merging::MergeSort(int low, int high)
{
 int mid;
 if(low<high)
 {
  mid=(low+high)/2;
                cout<<"Dividing :";                MergeDisplay(low,mid);
  MergeSort(low,mid);
                cout<<"Dividing :";                MergeDisplay(mid+1,high);
  MergeSort(mid+1,high);
                cout<<"Merging list 1 :";    MergeDisplay(low,mid);
                cout<<"Merging list 2 :";    MergeDisplay(mid+1,high);
  Merge(low,high,mid);
                cout<<"After merging :";           MergeDisplay(low,high);
 }
}

void Merging::Merge(int low, int high, int mid)
{
 int i, j, k, c[50];
 i=low;
 j=mid+1;
 k=low;
 while((i<=mid)&&(j<=high))
 {
  if(a[i]<a[j])
  {
   c[k]=a[i];
   k++;
   i++;
  }
  else
  {
   c[k]=a[j];
   k++;
   j++;
  }
 }
 while(i<=mid)
 {
  c[k]=a[i];
  k++;
  i++;
 }
 while(j<=high)
 {
  c[k]=a[j];
  k++;
  j++;
```

```
 }
 for(i=low;i<k;i++)
 {
  a[i]=c[i];
 }
}

void Merging::MergeDisplay(int l,int h)
{
 for(int i=l;i<=h;i++)
   cout<<"  "<<a[i];
 cout<<endl;
}

int main()
{
 Merging M;
 clrscr();
 M.GetData();          cout<<"Dividing :";          M.MergeDisplay(0,M.Getn()-1);
 M.MergeSort(0,M.Getn()-1);
 cout<<"\n\t The Sorted Array";
 cout<<"\n\t ~~~~~~~~~~~~~~~~\n";
 M.Display();
 getch();
 return 0;
}
```

Sample Input/output:

```
Enter the number of elements to be sorted : 5
Enter element no. 1 : 12
Enter element no. 2 : 34
Enter element no. 3 : 54
Enter element no. 4 : 20
Enter element no. 5 : 15
Dividing :  12  34  54 20 15
Dividing :  12  34  54
Dividing :  12  34
Dividing :  12
Dividing :  34
Merging list 1 :  12
Merging list 2 :  34
After merging :  12  34
Dividing :  54
Merging list 1 :  12  34
Merging list 2 :  54
After merging :  12  34  54
Dividing :  20  15
Dividing :  20
```

```
Dividing :  15
Merging list 1 :  20
Merging list 2 :  15
After merging :  15  20
Merging list 1 :  12  34  54
Merging list 2 :  15  20
After merging :  12  15  20  34  54

        The Sorted Array
        ~~~~~~~~~~~~~
 12  15  20  34  54
```

Result:

Thus, a C++ program to sort the given list by merge sort technique using divide and conquer method has been successfully written and outputs have been verified.

Ex. No. 6
Date:

Quick Sort

Aim:

To write a C++ program to sort the given list by quick sort technique using divide and conquer method.

Procedure:

Create the base class DataArray with the following member variables and functions.

Member variables in the base class DataArray

n as integer
a as array of integer of size 100

Member functions in the base class DataArray

int Getn(void)	- To return value of n
void GetData(void)	- To get the input from user
void Display(void)	- To display elements of the list

Algorithm

void GetData(void)

1. Get the list of numbers.

void Display(void)

1. Display all elements of the list one by one.

Create a derived class Quick, derived from the base class DataArray with public access.

Member variable in the derived class Quick is none.

Member functions in the derived class Quick

void QuickSort(int low, int high)	- To divide the list into two part based on pivot element
void QuickDisplay(int l,int h)	- To Display elements from l to h

Algorithm

void QuickSort(int low, int high)

1. Set the first element in the list as pivot element.
2. Assume two pointers left and right moving from either end of the list towards each other
3. Move left pointer until finding the element larger than the pivot element's value.
4. Move the right pointer until finding the element not larger than the pivot element value.
5. Swap the contents of the location pointed by these two pointers.

6. Repeat the steps 3, 4 and 5 until the pointers crossed each other.
7. Swap the contents of the location pointed by pivot pointer and right pointer.
8. Now, Repeat the above steps from 1 to 8 by considering two new lists, partitioned by the pivot pointer until further partitioning is not possible.

void QuickDisplay(int l,int h)

1. Display elements from l to h in the list one by one.

int main()

1. Create an object Q of class Quick.
2. Call member function GetData() of object Q.
3. Call member function QuickSort() of object Q.
4. Call member function Display() of object Q.

Source Code:

```
// Quick sorting.
#include<iostream.h>
#include<conio.h>
class DataArray
{
 protected:
        int a[100],n;
 public:
        int Getn(void) { return n; }
        void GetData(void);
        void Display(void);
};

void DataArray::GetData(void)
{
 cout<<"\n Enter the number of elements to be sorted : ";
 cin>>n;
 for(int i=0;i<n;i++)
 {
  cout<<" Enter element no. "<<i+1<< " :";
  cin>>a[i];
 }
}

void DataArray::Display(void)
{
 for(int i=0;i<n;i++)
   cout<<"  "<<a[i];
 cout<<endl;
}
```

```
class Quick : public DataArray
{
public:
        void QuickSort(int low, int high);
        void QuickDisplay(int l,int h);
};

void Quick::QuickSort(int low, int high)
{
 int left = low+1;
 int right = high;
 int temp = 0;

 int pivot = a[low];  /* first element as pivot element */

 if ( low > high ) return;
 /* partition */
 do
 {
  while(a[left] < pivot && left < high) left++;          /* find element above ... */
  while(a[right] >= pivot && right > low) right--;       /* find element below ... */
  if (left<right)
  {
  temp = a[left];
  a[left] = a[right];
  a[right] = temp;
  }
  else break;
 }while(1);

   temp = a[low];
   a[low] = a[right];
   a[right] = temp;
   cout<<"Left :";          QuickDisplay(low,right-1);
   cout<<" Pivot : [ " <<pivot<<" ] ";
   cout<<"Right :";         QuickDisplay(right+1,high);
   cout<<endl;
   /* recursive */
  QuickSort(low, right-1);
  QuickSort(right+1, high);
}

void Quick::QuickDisplay(int l,int h)
{
 cout<<" {";
 for(int i=l;i<=h;i++)
   cout<<"  "<<a[i];
 cout<<" } ";
}
```

```
int main()
{
 Quick Q;
 clrscr();
 Q.GetData();
 cout<<"Orginal : ";  Q.QuickDisplay(0,Q.Getn()-1);      cout<<endl;
 Q.QuickSort(0,Q.Getn()-1);
 cout<<"\n\t The Sorted Array";
 cout<<"\n\t ~~~~~~~~~~~~~~~~\n";
 Q.Display();
 getch();
 return 0;
}
```

Sample Input/Output:

```
 Enter the number of elements to be sorted : 7
 Enter element no. 1 :30
 Enter element no. 2 :10
 Enter element no. 3 :20
 Enter element no. 4 :15
 Enter element no. 5 :45
 Enter element no. 6 :40
 Enter element no. 7 :50
Orginal :  {  30  10  20  15  45  40  50 }
Left : {  15  10  20 }  Pivot : [ 30 ] Right : {  45  40  50 }
Left : {  10 }  Pivot : [ 15 ] Right : {  20 }
Left : { }  Pivot : [ 10 ] Right : { }
Left : { }  Pivot : [ 20 ] Right : { }
Left : {  40 }  Pivot : [ 45 ] Right : {  50 }
Left : { }  Pivot : [ 40 ] Right : { }
Left : { }  Pivot : [ 50 ] Right : { }

      The Sorted Array
      ~~~~~~~~~~~~
 10  15  20  30  40  45  50
```

Result:

Thus, a C++ program to sort the given list by quick sort technique using divide and conquer method has been successfully written and outputs have been verified.

Ex. No. 7
Date:

Stack Operations

Aim:

To write a C++ program to implement a Stack data structure.

Procedure:

Member variables in the class Stack

size, top as integer
StackData as array of integer of size 100

Member functions in the class Stack

Stack() {top=-1;}	- Constructor to set top as -1
void SetSize(int n)	- To set the size of stack
void Push(int x)	- To insert value x into stack
int Pop(void)	- To remove a value from stack
void Display(void)	- To display all elements of stack

Algorithm

Stack()

1. Initialize the top to –1.

void SetSize(int n)

1. Set the size of the Stack as n

void Push(int x)

1. Check whether the top is less than the size of the stack.
2. If there is space then, increment the top by one and place the new element in the position pointed by top.
3. Otherwise, display “Stack is over flow”.

int Pop(void)

1. Check whether the top is greater than –1.
2. If there is an element then, store the element present at the location pointed by top in x and decrement the value of the pointer by one and return the value of x.
3. Otherwise, display “Stack is under flow”.

void Display(void)

1. Display the elements present in the locations pointed by top to location zero.
2. Otherwise, display “Stack is empty”.

int main()

1. Display menu to perform Stack operations.
2. Get user choice and perform corresponding operation.

Source Code:

```
//Stack Operations
#include<iostream.h>
#include<conio.h>

class Stack
{
 int StackData[100],size,top;
 public:
        Stack() {top=-1;}
        void SetSize(int n) {size=n;}
        void Push(int x);
        int Pop(void);
        void Display(void);
};

void Stack::Push(int x)
{
 if(top>=size-1)
 {
  cout<<"\n\t\tStack is over flow...";
  getch();
 }
 else
 {
  top++;
  StackData[top]=x;
  cout<<"\n\t\t"<<x<<" is pushed \n\n\n";
 }
}

int Stack::Pop(void)
{
 int x;
 if(top<=-1)
 {
  cout<<"\n\t\tStack is under flow";
  return -9999;
 }
 else
 {
  x=StackData[top];
  top--;
  return x;
 }
}
```

```
void Stack::Display(void)
{
 if(top>=0)
 {
  cout<<"\nThe elements in stack.";
  cout<<"\n~~~~~~~~~~~~~~~~~~~~~~~~";
  for(int i=top;i>=0;i--)
   cout<<"\n\nDATA "<<i+1<<" = "<<StackData[i];
  cout<<"\n\n\n\n Press any key to continue...";
 }
 else
 {
  cout<<"\nThe Stack is empty.";
 }
}

int main()
{
 Stack S;
 clrscr();
 int option,n;
 cout<<"\nEnter the size of stack [MAX=100] : ";
 cin>>n;
 S.SetSize(n);
 do
 {
  clrscr();
  cout<<"\n\t\tStack Operation";
  cout<<"\n\t\t~~~~~~~~~~~~~~~~";
  cout<<"\n\t1.PUSH\n\t2.POP\n\t3.DISPLAY\n\t4.EXIT";
  cout<<"\n\nEnter your option : ";
  cin>>option;
  int x,topvalue;
  switch(option)
  {
   case 1: cout<<"\nEnter a value to be pushed : ";
           cin>>x;
           S.Push(x);
           S.Display();
           break;
   case 2: topvalue=S.Pop();
           if (topvalue!=-9999)
                cout<<"\n\t\t The popped element is "<<topvalue;
           S.Display();
           break;
   case 3: S.Display(); break;
   case 4: cout<<"\n\n\t\t You are on EXIT..."; break;
   default:cout<<"\n\n\t Enter only 1,2,3,4.";
  }
  getch();
```

```
 }while(option!=4);
 return 0;
}
```

Sample Input/Output:

```
Enter the size of stack [MAX=10] : 5

        Stack Operation
        ~~~~~~~~~~~
    1.PUSH
    2.POP
    3.DISPLAY
    4.EXIT

Enter your option : 1

Enter a value to be pushed : 5

        5 is Pushed

The elements in stack.
~~~~~~~~~~~~~~~

DATA 1 = 5

 Press any key to continue...

        Stack Operation
        ~~~~~~~~~~~
    1.PUSH
    2.POP
    3.DISPLAY
    4.EXIT

Enter your option : 1

Enter a value to be pushed : 4

        4 is Pushed

The elements in stack.
~~~~~~~~~~~~~~~

DATA 2 = 4
DATA 1 = 5

 Press any key to continue...
```

```
        Stack Operation
        ~~~~~~~~~~~
    1.PUSH
    2.POP
    3.DISPLAY
    4.EXIT

Enter your option : 1

Enter a value to be pushed : 3

        3 is Pushed

The elements in stack.
~~~~~~~~~~~~~~~

DATA 3 = 3
DATA 2 = 4
DATA 1 = 5

 Press any key to continue...

        Stack Operation
        ~~~~~~~~~~~
    1.PUSH
    2.POP
    3.DISPLAY
    4.EXIT

Enter your option : 1

Enter a value to be pushed : 22

        22 is Pushed

The elements in stack.
~~~~~~~~~~~~~~~

DATA 4 = 22
DATA 3 = 3
DATA 2 = 4
DATA 1 = 5
 Press any key to continue...

        Stack Operation
        ~~~~~~~~~~~
    1.PUSH
    2.POP
    3.DISPLAY
```

```
    4.EXIT

Enter your option : 1

Enter a value to be pushed : 11

        11 is Pushed


The elements in stack.
~~~~~~~~~~~~~~~

DATA 5 = 11
DATA 4 = 22
DATA 3 = 3
DATA 2 = 4
DATA 1 = 5

 Press any key to continue...

        Stack Operation
        ~~~~~~~~~~~
    1.PUSH
    2.POP
    3.DISPLAY
    4.EXIT

Enter your option : 1

        Stack is over flow...
 Press any key to continue...

        Stack Operation
        ~~~~~~~~~~~
    1.PUSH
    2.POP
    3.DISPLAY
    4.EXIT

Enter your option : 2

        The popped element is  11.

The elements in stack.
~~~~~~~~~~~~~~~

DATA 4 = 22
DATA 3 = 3
DATA 2 = 4
DATA 1 = 5
```

```
 Press any key to continue...

        Stack Operation
        ~~~~~~~~~~~
    1.PUSH
    2.POP
    3.DISPLAY
    4.EXIT

Enter your option : 2

        The popped element is  22.
The elements in stack.
~~~~~~~~~~~~~~~

DATA 3 = 3
DATA 2 = 4
DATA 1 = 5

 Press any key to continue...

        Stack Operation
        ~~~~~~~~~~~
    1.PUSH
    2.POP
    3.DISPLAY
    4.EXIT

Enter your option : 2

        The popped element is  3.
The elements in stack.
~~~~~~~~~~~~~~~

DATA 2 = 4
DATA 1 = 5

 Press any key to continue...

        Stack Operation
        ~~~~~~~~~~~
    1.PUSH
    2.POP
    3.DISPLAY
    4.EXIT

Enter your option : 2

        The popped element is  4.
```

```
The elements in stack.
~~~~~~~~~~~~~~~
DATA 1 = 5

 Press any key to continue...


        Stack Operation
        ~~~~~~~~~~~
    1.PUSH
    2.POP
    3.DISPLAY
    4.EXIT

Enter your option : 2

        The popped element is  5.
The Stack is empty.

        Stack Operation
        ~~~~~~~~~~~
    1.PUSH
    2.POP
    3.DISPLAY
    4.EXIT

Enter your option : 2

        Stack is under flow
The Stack is empty.

        Stack Operation
        ~~~~~~~~~~~
    1.PUSH
    2.POP
    3.DISPLAY
    4.EXIT

Enter your option : 4
        You are on EXIT...
```

Result:

Thus, a C++ program to implement the Stack data structure has been successfully written and its operations have been verified.

Ex. No. 8
Date:

Queue Operations

Aim:

To write a C++ program to implement a queue data structure.

Procedure:

Member variables in the class Queue

size, front, rear as integer
QueueData as array of integer of size 100

Member functions in the class Queue

Queue()	- Constructor to set front and rear as 0
void SetSize(int n)	- To set the size of queue
void Enque(int);	- To insert value x into queue
int Deque(void);	- To remove a value from queue
void View(void);	- To display all elements of queue

Algorithm

Queue()

1. Initialize the front and rear to 0.

void SetSize(int n)

1. Set the size of the Queue as n.

void Enque(int x)

1. Check whether there is any space for new element to be added.
2. If there is any space then, place the new element in the position pointed by rear and increment the rear by one.
3. Otherwise, display “Queue is over flow”.

int Deque(void)

1. Check whether there is any element present in the queue.
2. If there is an element then, store the element present at the location pointed by front in x and increment the front by one and return the value of x.
3. Otherwise, display “Queue is under flow”.

void Display(void)

1. If there is element Display the elements present in the locations pointed by front to rear.
2. Otherwise, display “Queue is empty”.

int main()

1. Display menu to perform queue operations.
2. Get user choice and perform corresponding operation.

Source Code:

```
//Queue Operation
#include<iostream.h>
#include<conio.h>

class Queue
{
int QueueData[100],size,front,rear;
public:
        Queue() {front=0;rear=0;}
        void SetSize(int n) {size=n;}
        void Enque(int);
        int Deque(void);
        void View(void);
};

void Queue::Enque(int x)
{
 if((rear-front+1)>size)
 {
  cout<<"\n\t\tQueue is over flow...";
 }
 else
 {
  QueueData[rear]=x;
  cout<<"\n\t\t"<<x<<" is added.\n";
  rear++;
 }
}

int Queue::Deque(void)
{
 int x;
 if(front>=rear)
 {
  cout<<"\n\t\tQueue is under flow...";
  return -9999;
 }
 else
 {
  x=QueueData[front];
  front++;
  return x;
 }
}

void Queue::View()
{
 int i=front,j=rear;
```

```
 if(i>=j)
 {
  cout<<"\n\t\tQueue is empty...";
 }
 else
 {
  cout<<"\n\n\tData in Queue :";
  for(;i<=j-1;i++)
   cout<<"  "<<QueueData[i];
 }
}

int main()
{
 clrscr();
 int n;
 cout<<"\nEnter the size of Queue [MAX=100] : ";
 cin>>n;
 Queue Q;
 Q.SetSize(n);
 int choice;
 do
 {
  clrscr();
  cout<<"\n\t\tQueue Operation";
  cout<<"\n\t\t~~~~~~~~~~~~~~~~";
  cout<<"\n\n\t1.INSERT\n\t2.DELETE\n\t3.VIEW\n\t4.EXIT";
  cout<<"\n\nEnter your choice : ";
  cin>>choice;
  int x;
  switch(choice)
  {
   case 1: cout<<"\nEnter a value : ";
           cin>>x;
           Q.Enque(x);
           Q.View();
           break;
   case 2: x=Q.Deque();
           if (x!=-9999)
                cout<<"\n\n\t"<<x<<" is deleted.\n\n";
           Q.View();
           break;
   case 3: Q.View();break;
   case 4: cout<<"\n\n\t\t  Exit...";break;
  }
  getch();
 }while(choice!=4);
 return 0;
}
```

Sample Input/Output:

```
Enter The size of Queue [MAX=100] : 3


        Queue Operation
        ~~~~~~~~~~~~~

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

Enter your choice : 1

Enter a value : 11

        11 is added.


    Data in Queue : 11

        Queue Operation
        ~~~~~~~~~~~~~

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

Enter your choice : 1

Enter a value : 22

        22 is added.


    Data in Queue : 11  22

        Queue Operation
        ~~~~~~~~~~~~~

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

Enter your choice : 1
```

```
Enter a value : 33

        33 is added.


    Data in Queue : 11  22  33

        Queue Operation
        ~~~~~~~~~~~~

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

Enter your choice : 1

            Queue is over flow...

Data in Queue : 11  22  33

        Queue Operation
        ~~~~~~~~~~~~

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

Enter your choice : 3

      Data in Queue : 11  22  33


        Queue Operation
        ~~~~~~~~~~~~

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

Enter your choice : 2

    11 is deleted.

    Data in Queue : 22  33
```

Queue Operation
~~~~~~~~~~~~

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

Enter your choice : 2

22 is deleted.

Data in Queue : 33

Queue Operation
~~~~~~~~~~~~

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

Enter your choice : 2

33 is deleted.

Queue is empty...

Queue Operation
~~~~~~~~~~~~

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

Enter your choice : 2

Queue is under flow...
Queue is empty...

Queue Operation
~~~~~~~~~~~~

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

Enter your choice : 4

Exit...

Result:

Thus, a C++ program to implement the queue data structure has been successfully written and its operations have been verified.

Ex. No. 9
Date:

Single Linked List

Aim:

To write a C++ program to implement a single linked list data structure.

Procedure:

Create user defined data type called node using structure in C++ with the following members

data as integer
link as a pointer to hold the address of the next node

Member variables in the class LinkedList

Head as pointer to the node

Member functions in the LinkedList

LinkedList()	- Constructor to initialize Head as NULL
~LinkedList(){delete Head;}	- Destructor to release the memory of Head
void Append(int x);	- To add a node at the end of the linked list
void AddAfter(int loc,int x);	- To add a node after the specific location in the linked list
void AddAtBegin(int x);	- To add a node at the beginning of the linked list
void Display();	- To display all nodes of linked list
void Count();	- To count number of nodes in the linked list
void Delete(int x);	- To delete a node in the linked list

Algorithm

LinkedList()

1. Initialize Head as NULL.

~LinkedList()

1. Release the memory of Head.

void Append(int x)

1. Create a node temp and get the data in the data field of the temp node and next-address field of temp to hold NULL.
2. Make the next-address field of the last node to hold the address of the temp node.

void AddAfter(int loc,int x)

1. Create a node temp and get the data in the data field of the temp node and next-address field of temp to hold NULL.
2. After traversing N nodes in the list i.e., specified by loc, make the N^{th} node next-address field to hold the address of the temp node and then make the next-address filed of the temp node to hold the address of the $(N+1)^{th}$ node.

void AddAtBegin(int x)

1. Create a node temp and get the data in the data field of the temp node and next-address field of temp to hold NULL.
2. Make the temp node as front node by making the next-address field of the temp node to hold the address of the front node of the list. [If there is no elements already present in the list then, the list would have only this one element in the data field and NULL in the address field]

void Display()

1. Traverse the list from the first node to the last node by displaying the contents of its data field.

void Count()

1. Count the nodes by traversing the node one by one from the first node till the address field of the node contains NULL.

void Delete(int x)

1. Locate the node having the data x in its data field.
2. If the node to be deleted is present at the first location, then make the next (second) node as the first node by making the first node's next-address field to NULL and free the memory location of the deleted node.
3. If the node to be deleted is present as intermediate node (having previous node and next node in the list) then, make the previous node's next-address field to hold the address of the next node and free the memory location of the deleted node.
4. If the node to be deleted is present at the last in the list, then make the previous node address to hold the NULL value and free the memory location of the deleted node.
5. If the list consists of only one node then, make the list as Null and free the memory location of the deleted node.

int main()

1. Display menu to perform linked list operations.
2. Get user choice and perform corresponding operation.

Source Code:

```
//Single Linked List
#include<iostream.h>
#include<conio.h>

typedef struct node
{
 int data;
 struct node *link;
};
```

```
class LinkedList
{
node *Head;
public:
        LinkedList(){Head=NULL; }
        ~LinkedList(){delete Head;}
        void Append(int x);
        void AddAfter(int loc,int x);
        void AddAtBegin(int x);
        void Display();
        void Count();
        void Delete(int x);
};

void LinkedList::Append(int num)
{
 node *temp,*r;
 temp=Head;

 if(Head==NULL)
 {
  temp=new node;
  temp->data=num;
  temp->link=NULL;
  Head=temp;
 }
 else
 {
  while(temp->link!=NULL)
   temp=temp->link;
  r=new node;
  r->data=num;
  r->link=NULL;
  temp->link=r;
 }
}

void LinkedList::AddAfter(int loc,int num)
{
 node *temp,*r;
 int i;
 temp=Head;
 if(temp==NULL)
 {
  cout<<"\n\n\t\t There are less than "<<loc<<" elements in list.";
  return;
 }
 for(i=0;i<loc-1;i++)
 {
  if(temp==NULL)
```

```
 {
  cout<<"\n\n\t\tThere are less than "<<loc<<" elements in list.";
  return;
 }
 temp=temp->link;
}
 r=new node;
 r->data=num;
 r->link=temp->link;
 temp->link=r;
}

void LinkedList::AddAtBegin(int num)
{
 node *temp;
 temp = new node;
 temp->data=num;
 temp->link=Head;
 Head=temp;
}

void LinkedList::Display()
{
 cout<<"\n\n\n";
 node *temp=Head;
 while(temp!=NULL)
 {
  cout<<temp->data<<" --> ";
  temp=temp->link;
 }
 cout<<"NULL";
 getch();
}

void LinkedList::Count()
{
 int c=0;
 node *temp=Head;
 while(temp!=NULL)
 {
  temp=temp->link;
  c++;
 }
 cout<<"\n\n\n\t\tCount = "<<c;
}
```

```
void LinkedList::Delete(int num)
{
 node *old,*temp;
 temp=Head;
 while(temp!=NULL)
 {
  if(temp->data==num)
  {
   if(temp==Head)
   {
    Head=temp->link;
    delete temp;
    return;
   }
   else
   {
    old->link=temp->link;
    delete temp;
    return;
   }
  }
  else
  {
   old=temp;
   temp=temp->link;
  }
 }
 cout<<"\n\n\n\t\tThe data not found.";
}

int main()
{
 LinkedList L;
 int choice,x,loc;

 do
 {
  clrscr();
  cout<<"\n\t\t\tSingle Linked List";
  cout<<"\n\t\t\t~~~~~~~~~~~~~~~~~~";
  cout<<"\n\t1. CREATE A NODE";
  cout<<"\n\t2. ADD AT THE BEGINING";
  cout<<"\n\t3. ADD AT END";
  cout<<"\n\t4. ADD AFTER A SPECIFIC LOCATION";
  cout<<"\n\t5. DISPLAY ";
  cout<<"\n\t6. COUNT NO OF NODES";
  cout<<"\n\t7. DELETE A SPECIFIED NODE";
  cout<<"\n\t8. EXIT";
  cout<<"\n\n\nEnter your choice : ";
  cin>>choice;
```

```
 switch(choice)
 {
  case 1 :
        cout<<"\n Enter a value : ";
        cin>>x;
        L.Append(x);
        L.Display();
        break;
  case 2 :
        cout<<"\n Enter a value : ";
        cin>>x;
        L.AddAtBegin(x);
        L.Display();
        break;
  case 3 :
        cout<<"\n Enter a value : ";
        cin>>x;
        L.Append(x);
        L.Display();
        break;
  case 4 :
        cout<<"\n Enter the location : ";
        cin>>loc;
        cout<<"\n Enter a value : ";
        cin>>x;
        L.AddAfter(loc,x);
        L.Display();
        break;
  case 5 :
        L.Display();
        break;
  case 6 :
        L.Count();
        L.Display();
        break;
  case 7 :
        cout<<"\n Enter a value to delete : ";
        cin>>x;
        L.Delete(x);
        L.Display();
        break;
  case 8 :
        cout<<"\n\n\n\t\t You are on Exit . . . ";
        getch();
        break;
 }
 }while(choice!=8);
 return 0;
}
```

Sample Input/Output:

```
                Single Linked List
                ~~~~~~~~~~~~~
        1. CREATE A NODE
        2. ADD AT THE BEGINING
        3. ADD AT END
        4. ADD AFTER A SPECIFIC LOCATION
        5. DISPLAY
        6. COUNT NO OF NODES
        7. DELETE A SPECIFIED NODE
        8. EXIT

Enter your choice : 1

 Enter a value : 1

1 --> NULL

                Single Linked List
                ~~~~~~~~~~~~~
        1. CREATE A NODE
        2. ADD AT THE BEGINING
        3. ADD AT END
        4. ADD AFTER A SPECIFIC LOCATION
        5. DISPLAY
        6. COUNT NO OF NODES
        7. DELETE A SPECIFIED NODE
        8. EXIT

Enter your choice : 2

 Enter a value : 11

11 --> 1 --> NULL

                Single Linked List
                ~~~~~~~~~~~~~
        1. CREATE A NODE
        2. ADD AT THE BEGINING
        3. ADD AT END
        4. ADD AFTER A SPECIFIC LOCATION
        5. DISPLAY
        6. COUNT NO OF NODES
        7. DELETE A SPECIFIED NODE
        8. EXIT

Enter your choice : 3
```

```
 Enter a value : 33
11 --> 1 --> 33 --> NULL

                Single Linked List
                ~~~~~~~~~~~~~
        1. CREATE A NODE
        2. ADD AT THE BEGINING
        3. ADD AT END
        4. ADD AFTER A SPECIFIC LOCATION
        5. DISPLAY
        6. COUNT NO OF NODES
        7. DELETE A SPECIFIED NODE
        8. EXIT

Enter your choice : 4

 Enter the location : 2

 Enter a value : 23

11 --> 1 --> 23 --> 33 --> NULL

                Single Linked List
                ~~~~~~~~~~~~~
        1. CREATE A NODE
        2. ADD AT THE BEGINING
        3. ADD AT END
        4. ADD AFTER A SPECIFIC LOCATION
        5. DISPLAY
        6. COUNT NO OF NODES
        7. DELETE A SPECIFIED NODE
        8. EXIT

Enter your choice : 5

11 --> 1 --> 23 --> 33 --> NULL

                Single Linked List
                ~~~~~~~~~~~~~
        1. CREATE A NODE
        2. ADD AT THE BEGINING
        3. ADD AT END
        4. ADD AFTER A SPECIFIC LOCATION
        5. DISPLAY
        6. COUNT NO OF NODES
        7. DELETE A SPECIFIED NODE
        8. EXIT

Enter your choice : 6
```

```
        Count = 4

11 --> 1 --> 23 --> 33 --> NULL

                Single Linked List
                ~~~~~~~~~~~~~
    1. CREATE A NODE
    2. ADD AT THE BEGINING
    3. ADD AT END
    4. ADD AFTER A SPECIFIC LOCATION
    5. DISPLAY
    6. COUNT NO OF NODES
    7. DELETE A SPECIFIED NODE
    8. EXIT

Enter your choice : 7

 Enter a value to delete : 23

11 --> 1 --> 33 --> NULL

                Single Linked List
                ~~~~~~~~~~~~~
    1. CREATE A NODE
    2. ADD AT THE BEGINING
    3. ADD AT END
    4. ADD AFTER A SPECIFIC LOCATION
    5. DISPLAY
    6. COUNT NO OF NODES
    7. DELETE A SPECIFIED NODE
    8. EXIT

Enter your choice : 8

        You are on Exit . . .
```

Result:

Thus, a C++ program to implement the single linked list data structure has been successfully written and its operations have been verified.

Ex. No. 10
Date:

Binary Search Tree

Aim:

To write a C++ program to create a binary search tree and perform in-order, pre-order and post-order traversals.

Procedure:

Create user defined data type called node using structure in C++ with the following members

data as integer
left as a pointer to hold the address of the left child
right as a pointer to hold the address of the right child

Member variables in the class LinkedList

Root as pointer to the root node

Member functions in the LinkedList

BinarySearchTree()	- Constructor to initialize Root as NULL
~BinarySearchTree()	- Destructor to release the memory of Root
BSTnode *GetRoot(void)	- To know the Root
void SetRoot(BSTnode *root)	- To set the Root
BSTnode *CreateNode(void)	- To create a node
void Insert(BSTnode *root, BSTnode *New)	- To insert a new node in binary search tree
void Preorder(BSTnode *temp)	- To traverse preorder
void Inorder(BSTnode *temp)	- To traverse inorder
void Postorder(BSTnode *temp)	- To traverse preorder
void Display(BSTnode *temp,int row,int low,int high)	-To display the nodes in tree structure

Algorithm

BinarySearchTree()

1. Initialize Root as NULL

~BinarySearchTree()

1. Release the memory of Root

BSTnode *GetRoot(void)

1. Return the Root

void SetRoot(BSTnode *root)

1. Set the Root as root

BSTnode *CreateNode(void)

1. Create a node with default values and return its address.

void Insert(BSTnode *root, BSTnode *New)

1. If the tree is empty or left-address field and right-address field is holding NULL values, then store the value in root node itself.
2. Compare the new with the root for its place until the leaf node is reached by traverse through left child or right child based on the root data.
3. If the new value is greater than the leaf node value then, place the data in a new node and make the right-address field of the leaf node to hold the address of the new node.
4. If the new value is lesser than the leaf node value then, place the data in a new node and make the left-address field of the leaf node to hold the address of the new node.

void Preorder(BSTnode *temp)

1. Display the present root node's value
2. Traverse the left sub-tree by taking the left node as new root node in pre-orderly.
3. Traverse the right sub-tree by taking the left node as new root node in pre-orderly.

void Inorder(BSTnode *temp)

1. Traverse the left sub-tree by taking the left node as new root node in pre-orderly.
2. Display the present root node's value.
3. Traverse the right sub-tree by taking the left node as new root node in pre-orderly.

void Postorder(BSTnode *temp)

1. Traverse the left sub-tree by taking the left node as new root node in pre-orderly.
2. Traverse the right sub-tree by taking the left node as new root node in pre-orderly.
3. Display the present root node's value.

void Display(BSTnode *temp,int row,int low,int high)

1. Find the place to print the data and print it to display like a binary search tree.

int main()

1. Display menu to perform binary search tree creation and its traversals.
2. Get user choice and perform corresponding operation.

Source Code:

```
//Binary Search Tree Traversal
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
```

```
typedef struct BSTnode
{
 int data;
 BSTnode *left,*right;
};

class BinarySearchTree
{
BSTnode *Root;
public:
        BinarySearchTree() {Root=NULL; }
        ~BinarySearchTree() {delete Root;}
        BSTnode *GetRoot(void) {return Root;}
        void SetRoot(BSTnode *root) {Root=root;}
        BSTnode *CreateNode(void);
        void Insert(BSTnode *root, BSTnode *New);
        void Preorder(BSTnode *temp);
        void Inorder(BSTnode *temp);
        void Postorder(BSTnode *temp);
        void Display(BSTnode *temp,int row,int low,int high);
};

/*Create node*/
BSTnode *BinarySearchTree::CreateNode()
{
 BSTnode *temp;
 temp=new BSTnode;
 temp->left=NULL;
 temp->right=NULL;
 return temp;
}

/*Insert Function*/
void BinarySearchTree::Insert(BSTnode *root,BSTnode *New)
{
 if(New->data < root->data)
 {
  if(root->left==NULL)
   root->left=New;
  else
   Insert(root->left,New);
 }
 if(New->data >= root->data)
 {
  if(root->right==NULL)
   root->right=New;
  else
   Insert(root->right,New);
 }
}
```

```
/*Preorder Traversals*/
void BinarySearchTree::Preorder(BSTnode *temp)
{
 if(temp!=NULL)
 {
  cout<<"-> "<<temp->data<<" ";
  Preorder(temp->left);
  Preorder(temp->right);
 }
}

/*Inorder Traversals*/
void BinarySearchTree::Inorder(BSTnode *temp)
{
 if(temp!=NULL)
 {
  Inorder(temp->left);
  cout<<"-> "<<temp->data<<" ";
  Inorder(temp->right);
 }
}

/*Postorder Traversals*/
void BinarySearchTree::Postorder(BSTnode *temp)
{
 if(temp!=NULL)
 {
  Postorder(temp->left);
  Postorder(temp->right);
  cout<<"-> "<<temp->data<<" ";
 }
}

/*display of binary search tree in tree format*/
void BinarySearchTree::Display(BSTnode *temp,int row,int low,int high)
{
if(temp!=NULL)
 {
  Display(temp->left,row+2,low,(low+high)/2);
  gotoxy((low+high)/2,row);
  cout<<temp->data;
  Display(temp->right,row+2,(low+high)/2,high);
 }
}

int main()
{
 BinarySearchTree B;
 int ch;
```

```
BSTnode *New=NULL,*root=NULL;

while(1)
{
 clrscr();
 cout<<"\n\t\t\t Binary Search Tree";
 cout<<"\n\t\t\t ~~~~~~~~~~~~~~~~~~";
 cout<<"\n1-Create";
 cout<<"\n2-Preorder";
 cout<<"\n3-Inorder";
 cout<<"\n4-Postorder";
 cout<<"\n5-Display";
 cout<<"\n6-Exit\n";
 cout<<"\nEnter your choice : ";
 cin>>ch;
 root=B.GetRoot();
 switch(ch)
 {
  case 1:
        cout<<"\nEnter elements 1 by 1: (0 to stop entering)\n";
        do
        {
         New=B.CreateNode();
         cin>>New->data;
         if(New->data==0) break;
          if(root==NULL)
           root=New;
          else
           B.Insert(root,New);
        }while(1);
        B.SetRoot(root);
        break;
  case 2:
        if(root==NULL)
         cout<<"\nNo element in Tree\n";
        else
        {
         cout<<"\n~~~Preorder Traversal~~~\n\nThe Tree is:\n";
         B.Preorder(root);
        }
        getch();
        break;
  case 3:
        if(root==NULL)
         cout<<"\nNo element in Tree\n";
        else
        {
         cout<<"\n~~~Inorder Traversal~~~\n\nThe Tree is:\n";
         B.Inorder(root);
        }
```

```
        getch();
        break;
  case 4:
        if(root==NULL)
           cout<<"\nNo element in Tree\n";
        else
        {
         cout<<"\n~~~Postorder Traversal~~~\n\nThe Tree is:\n";
         B.Postorder(root);
        }
        getch();
        break;
  case 5:
        clrscr();
        B.Display(root,1,1,80);
        getch();
        break;
  case 6:
        cout<<"\n~~~Exit~~~\n";
        getch();
        exit(0);
  default:
        cout<<"\n Enter between 1 to 5\n\n";
        break;
 }
}
 return 0;
}
```

Sample Input/Output:

```
                    Binary Search Tree
                    ~~~~~~~~~~~~~~
1-Create
2-Preorder
3-Inorder
4-Postorder
5-Display
6-Exit

Enter your choice : 1

Enter elements 1 by 1: (0 to stop entering)
100
70
120
50
80
110
```

```
130
5
55
75
85
105
115
125
135
0

                    Binary Search Tree
                    ~~~~~~~~~~~~~~
1-Create
2-Preorder
3-Inorder
4-Postorder
5-Display
6-Exit

Enter your choice : 2

~~~Preorder Traversal~~~

The Tree is:
-> 100 -> 70 -> 50 -> 5 -> 55 -> 80 -> 75 -> 85 -> 120 -> 110 -> 105 -> 115 -> 130 ->
125 -> 135

                    Binary Search Tree
                    ~~~~~~~~~~~~~~
1-Create
2-Preorder
3-Inorder
4-Postorder
5-Display
6-Exit

Enter your choice : 3

~~~Inorder Traversal~~~

The Tree is:
-> 5 -> 50 -> 55 -> 70 -> 75 -> 80 -> 85 -> 100 -> 105 -> 110 -> 115 -> 120 -> 125 ->
130 -> 135

                    Binary Search Tree
                    ~~~~~~~~~~~~~~
1-Create
2-Preorder
3-Inorder
```

```
4-Postorder
5-Display
6-Exit

Enter your choice :4

~~~Postorder Traversal~~~

The Tree is:
-> 5 -> 55 -> 50 -> 75 -> 85 -> 80 -> 70 -> 105 -> 115 -> 110 -> 125 -> 135 -> 130 ->
120 -> 100

                    Binary Search Tree
                    ~~~~~~~~~~~~~~
1-Create
2-Preorder
3-Inorder
4-Postorder
5-Display
6-Exit

Enter your choice : 5

                          100

              70                              120

      50              80              110             130

   5      55      75      85      105     115     125     135


                    Binary Search Tree
                    ~~~~~~~~~~~~~~
1-Create
2-Preorder
3-Inorder
4-Postorder
5-Display
6-Exit

Enter your choice : 6

~~~Exit~~~
```

Result:

Thus, a C++ program to create the binary search tree and perform in-order, pre-order and post-order traversals has been written and outputs have been verified.

Ex. No. 11
Date:

Depth First Search Graph Traversal

Aim:

To write a program to traverse a graph using Depth First Search (DFS) algorithm.

Procedure:

Member variables in the class Graph

n and s as integer
a as two dimensional array of integer of size [10] [10]

Member functions in the class Graph

void getdata()	- To get the input from user
void dfs_traverse()	- To traverse a graph from one node to another node in depth first order.

Algorithm

void getdata()

1. Get the number of vertices in the graph.
2. Get the adjacency matrix of the graph.
3. Get the starting vertex for traversal.

void dfs_traverse()

1. Push starting vertex into a stack and make it visited.
2. Pop a vertex from stack.
3. Find all adjacent vertices of popped vertex in the reverse order in such a way that it is not already visited.
4. Push them into stack and make them visited.
5. Repeat steps 2 to 4 until the stack is empty.

int main()

1. Call the getdata( ) function to get input from user.
2. Call the dfs_traverse( ) function to do the traversal operation.

Source Code:

```
//Depth First Search
#include<iostream.h>
#include<conio.h>

class Graph
{
   int a[10][10],n,s;
```

```
public:
   void getdata();
   void dfs_traverse();
};

void Graph::getdata()
{
   cout<<"\n Enter the number of vertices in the graph: ";
   cin>>n;
   cout<<"\n Enter the adjacency matrix of graph: "<<endl;
   for(int i=0;i<n;i++)
         for(int j=0;j<n;j++)
            cin>>a[i][j];
   cout<<"\n Enter the vertex from which you want to traverse: ";
   cin>>s;
}

void Graph::dfs_traverse()
{
   int *visited= new int[n];
   int stack[10],top=-1,i;
   for(int j=0;j<n;j++)
         visited[j]=0;
   cout<<"\n The Depth First Search Traversal : "<<endl;
   i=stack[++top]=s;
   visited[s]=1;
   while(top>=0)
   {
         i=stack[top];
         cout<<stack[top--]<<endl;
         for(int j=n-1;j>=0;j--)
            if(a[i][j]!=0&&visited[j]!=1)
            {
              stack[++top]=j;
              visited[j]=1;
            }
   }
}

int main()
{
   Graph DFS;
   clrscr();
   DFS.getdata();
   DFS.dfs_traverse();
   getch();
   return 0;
}
```

Sample Input/ Output:

Enter the number of vertices in the graph: 6

Enter the adjacency matrix of graph:
0 1 1 0 0 0
1 0 0 1 1 0
1 0 0 0 1 0
0 1 0 0 1 1
0 1 1 1 0 1
0 0 0 1 1 0

Enter the vertex from which you want to traverse: 0

The Depth First Search Traversal:
0
1
3
5
4
2

Result:

Thus, a C++ program to traverse a graph using Depth First Search algorithm has been successfully written and outputs have been verified.

Ex. No. 12
Date:

Breadth First Search Graph Traversal

Aim:

To write a program to traverse a graph using Breadth First Search (BFS) algorithm.

Procedure:

Member variables in the class BGraph

n and start as integer
a as two dimensional array of integer of size [10] [10]

Member functions in the class BGraph

void getdata() - To get the input from user
void bfs_traverse() - To traverse a graph from one node to another node in breadth first order.

Algorithm

void getdata()
1. Get the number of vertices in the graph.
2. Get the adjacency matrix of the graph.
3. Get the starting vertex for traversal.

void bfs_traverse()
1. Insert starting vertex into a queue and make it visited.
2. Remove a vertex from queue.
3. Find all adjacent vertices of removed vertex in such a way that it is not already visited.
4. Insert them into queue and make them visited.
5. Keep repeating steps 2 to 4 until the queue is empty.

int main()
1. Call the getdata( ) function to get input from user.
2. Call the bfs_ traverse( ) function to do the traversal operation.

Source Code:

```
//Breadth First Search
#include<iostream.h>
#include<conio.h>

class BGraph
{
  int a[10][10],n,start;
public:
  void getdata();
```

```
   void bfs_traverse();
};

void BGraph::getdata()
{
   cout<<"\n Enter the number of vertices in the graph: ";
   cin>>n;
   cout<<"\n Enter the adjacency matrix of graph: "<<endl;
   for(int i=0;i<n;i++)
      for(int j=0;j<n;j++)
         cin>>a[i][j];
   cout<<"\n Enter the vertex from which you want to traverse: ";
   cin>>start;
}

void BGraph::bfs_traverse()
{
   int *visited= new int[n];
   int queue[10],front=-1,rear=0,i;
   for(int j=0;j<n;j++)
      visited[j]=0;
   cout<<"\n Traversing the graph using breadth first search algorithm : "<<endl;
   queue[rear]=start;
   visited[start]=1;
   while(front!=rear)
   {
      cout<<queue[++front]<<endl;
      i=queue[front];
      for(int j=0;j<n;j++)
         if(a[i][j]!=0 && visited[j]!=1)
         {
           queue[++rear]=j;
           visited[j]=1;
         }
   }
}

int main()
{
   BGraph bfs;
   bfs.getdata();
   bfs.bfs_traverse();
   getch();
   return 0;
}
```

Sample Input/Output:

Enter the number of vertices in the graph: 5

```
Enter the adjacency matrix of graph:
0  1  1  0  0
1  0  0  1  0
1  0  0  1  1
0  1  1  0  0
0  0  1  1  0

Enter the vertex from which you want to traverse: 1

Traversing the graph using breadth first search algorithm:
1
0
3
2
4
```

Result:

Thus, a C++ program to traverse a graph using Breadth First Search algorithm has been successfully written and outputs have been verified.