BHARATI VIDYAPEETH'S
INSTITUTE OF COMPUTER APPLICATIONS & MANAGEMENT

(Affiliated to Guru Gobind Singh Indraprastha University,
Approved by AICTE, New Delhi)

Data and File Structures Lab (MCA-106)

Practical File

Submitted to:
Dr. Saumya Bansal
(Associate Professor)

Submitted by:
AMAN
01211604422

Index

Set	Practical	Date
A	*AP1.* Write a program which takes an array of n integers and displays the frequency of each element present in the array.	
	AP2. Write a program which takes an array of n integers and performs searching of an element by implementing linear search and binary search techniques.	
	AP3. Write a program which takes an array of n integers and sorts the integers in descending order using bubble sort and selection sort techniques.	
	AP4. Write a program which takes an array of n integers and sorts the integers in ascending order using insertion sort technique.	
	AP5. Write a program which takes an array of n integers and sorts the integers in ascending order using quick sort technique.	
B	*BP1.* Write a menu-driven program which implements a linear linked list	
	BP2. A polynomial is composed of different terms where each of them holds a coefficient and an exponent. Write a program to represent the following polynomials: $4x^4 + 4x^3 -2x^2 + x$ and $11x^3 + 7x^2 - 4x$ with linear linked list, and then perform addition of the given polynomials.	
C	*CP1.* Write a menu-driven program which implements a stack	
	CP2. Write a menu-driven program which implements a linear queue	
	CP3. Write a menu-driven program which implements a circular queue	
D	*DP1.* Write a menu-driven program which implements a binary tree	
	DP2. Write a menu-driven program which implements a heap	
	DP3. Write a menu-driven program which implements a binary search tree	
	DP4. Write a program which takes an array of n integers and sorts the integers in ascending order using heap sort technique	
E	*EP1.* Write a menu-driven program which implements a graph	
	EP2. Write a program to travers the graph using breadth first search and depth first search techniques.	
	EP3. Write a program to determine shortest path	
	EP4. Write a program to determine shortest paths between every pair of vertices	
F	*FP1.* Write a program that generates n random integers and stores them in a text file, named as “All.txt”. Then, retrieve the stored integers from this file and copy to “Odd.txt” and „Even.txt‟ based upon the type of number, i.e. if the retrieved integer is odd number then store in “Odd.txt” file or if the retrieved integer is even then store in “Even.txt” file. Finally, display the contents of all three files.	
	FP2. Write a program to find the the highest grade, and list all the students who have highest grade. Also, list the details of students‟ having 3rd highest grade.	
	FP3. Write a program to implement 20 integers (with some duplicate integers) in a hash table (with separate chaining for collision resolution).	

Assignment Set: A (Searching and Sorting in Array)

Array Implementation: DataStructures/Arrays/StaticArray.hpp

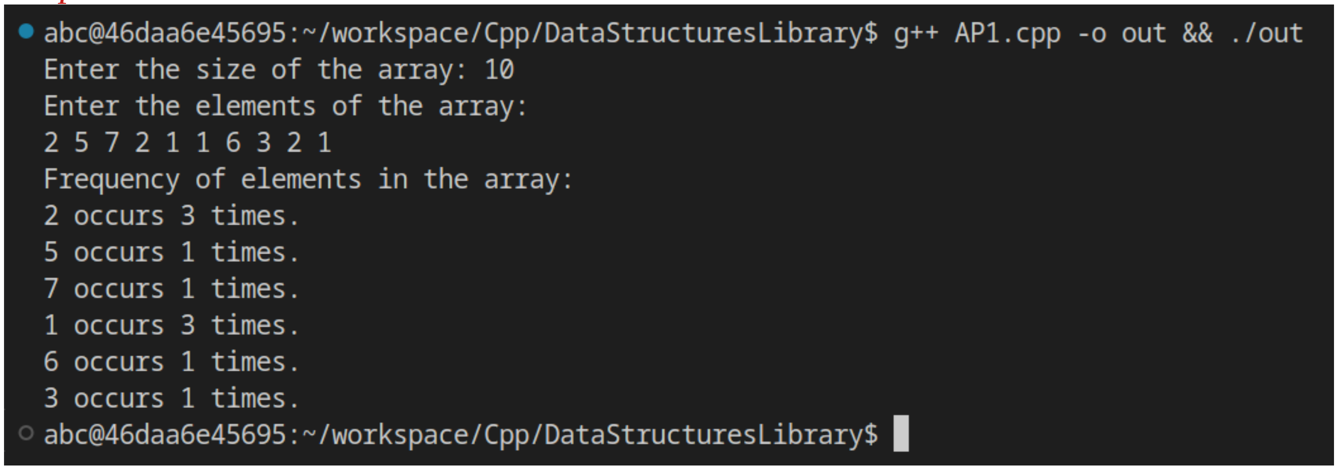
```
#pragma once
#include <stdexcept>

namespace DS {
  template<typename T, int Max_size>
  class StaticArray {
    T elements[Max_size] = {};
    int max_size = Max_size;
    int size = 0;
  public:
    StaticArray() = default;
    StaticArray(int _size) {
      size = _size;
    }
    StaticArray(const StaticArray<T, Max_size>& array) {
      size = array.size;
      for (int i = 0; i < size; i++) {
        elements[i] = array.elements[i];
      }
    }

    T& operator[](int n) {
      if (n < size && n >= 0) {
        return elements[n];
      }
      throw std::out_of_range("Index out of Range");
    }

    bool PushBack(const T& item) {
      if (size != max_size) {
        elements[size] = item;
        size++;
        return true;
      }
      return false;
    }

    T PopBack() {
      if (size > 0) {
        size -= 1;
        return elements[size];
      }
      else {
        throw std::out_of_range("Array is empty");
      }
    }

    bool PushAt(T item, int location) {
      if (size != max_size) {
        size++;
        for (int i = size - 1; i >= location; i--) {
          elements[i] = elements[i - 1];
        }
        elements[location] = item;
        return true;
      }
      return false;
    }

    T PopAt(int location) {
      if (size > 0) {
        if (location >= 0 && location < size) {
          T& item = elements[location];
          for (int i = location; i < size; i++) {
            elements[i] = elements[i + 1];
          }
          size--;
          return item;
        }
        throw std::out_of_range("Index out of range");
      }
      throw std::out_of_range("Array is empty");
    }

    int MaxSize() {
      return max_size;
    }

    void MaxSize(int _max_size) {
      max_size = _max_size;
    }

    int Size() {
      return size;
    }

    void setSize(int n) {
      size = n;
    }

    void Clear() {
      size = 0;
    }

    void Swap(int i1, int i2) {
      std::swap(elements[i1], elements[i2]);
    }
  };
 }
```

AP1 Write a program which takes an array of n integers and displays the frequency of each element present in the array.

Code:

```
#include <iostream>
#include "DataStructures/Arrays/StaticArray.hpp"

using namespace std;
using namespace DS;

int main() {
    StaticArray<int, 100> arr;
    int n, x;
    cout << "Enter the size of the array: ";
    cin >> n;
    cout << "Enter the elements of the array:\n";
    for (int i = 0; i < n; i++) {
        cin >> x;
        arr.PushBack(x);
    }

    StaticArray<int, 100> freq;
    freq.setSize(n);
    for (int i = 0; i < n; i++) {
        freq[i] = -1;
    }

    for (int i = 0; i < n; i++) {
        int count = 1;
        for (int j = i + 1; j < n; j++) {
            if (arr[i] == arr[j]) {
                count++;
                freq[j] = 0;
            }
        }
        if (freq[i] != 0) {
            freq[i] = count;
        }
    }

    cout << "Frequency of elements in the array:\n";
    for (int i = 0; i < n; i++) {
        if (freq[i] != 0) {
            cout << arr[i] << " occurs " << freq[i] << " times.\n";
        }
    }

    return 0;
}
```

Output:



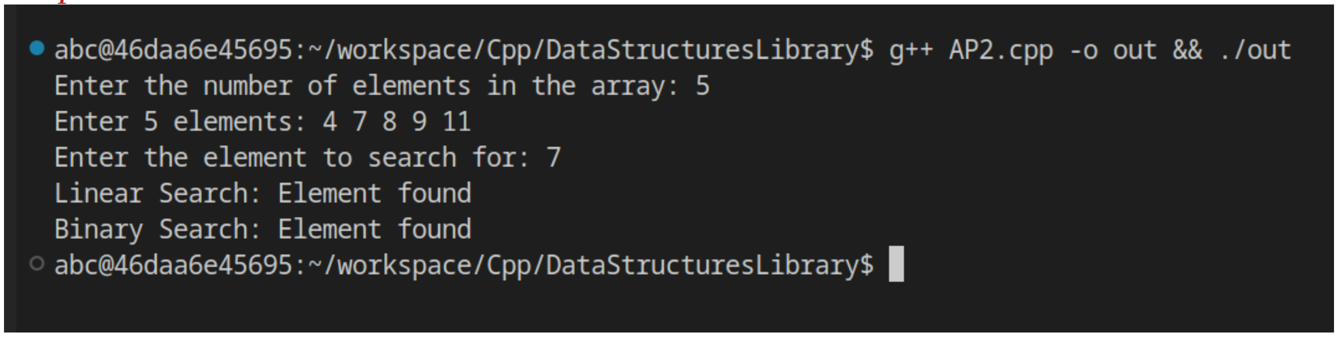
```
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$ g++ AP1.cpp -o out && ./out
Enter the size of the array: 10
Enter the elements of the array:
2 5 7 2 1 1 6 3 2 1
Frequency of elements in the array:
2 occurs 3 times.
5 occurs 1 times.
7 occurs 1 times.
1 occurs 3 times.
6 occurs 1 times.
3 occurs 1 times.
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$
```

AP2 Write a program which takes an array of n integers and performs searching of an element by implementing linear search and binary search techniques.

Code:

```
#include <iostream>
#include "DataStructures/Arrays/StaticArray.hpp"

using namespace std;
using namespace DS;

bool LinearSearch(StaticArray<int, 100>& arr, int key) {
    for (int i = 0; i < arr.Size(); i++) {
        if (arr[i] == key) {
            return true;
        }
    }
    return false;
}

bool BinarySearch(StaticArray<int, 100>& arr, int key) {
    int low = 0;
    int high = arr.Size() - 1;

    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] == key) {
            return true;
        }
        else if (arr[mid] < key) {
            low = mid + 1;
        }
        else {
            high = mid - 1;
        }
    }

    return false;
}

int main() {
    StaticArray<int, 100> arr;
    int n;

    cout << "Enter the number of elements in the array: ";
    cin >> n;

    cout << "Enter " << n << " elements: ";
    for (int i = 0; i < n; i++) {
        int x;
        cin >> x;
        arr.PushBack(x);
    }

    int key;
    cout << "Enter the element to search for: ";
    cin >> key;
    if (LinearSearch(arr, key)) {
        cout << "Linear Search: Element found" << endl;
    }
    else {
        cout << "Linear Search: Element not found" << endl;
    }

    if (BinarySearch(arr, key)) {
        cout << "Binary Search: Element found" << endl;
    }
    else {
        cout << "Binary Search: Element not found" << endl;
    }

    return 0;
}
```

Output:

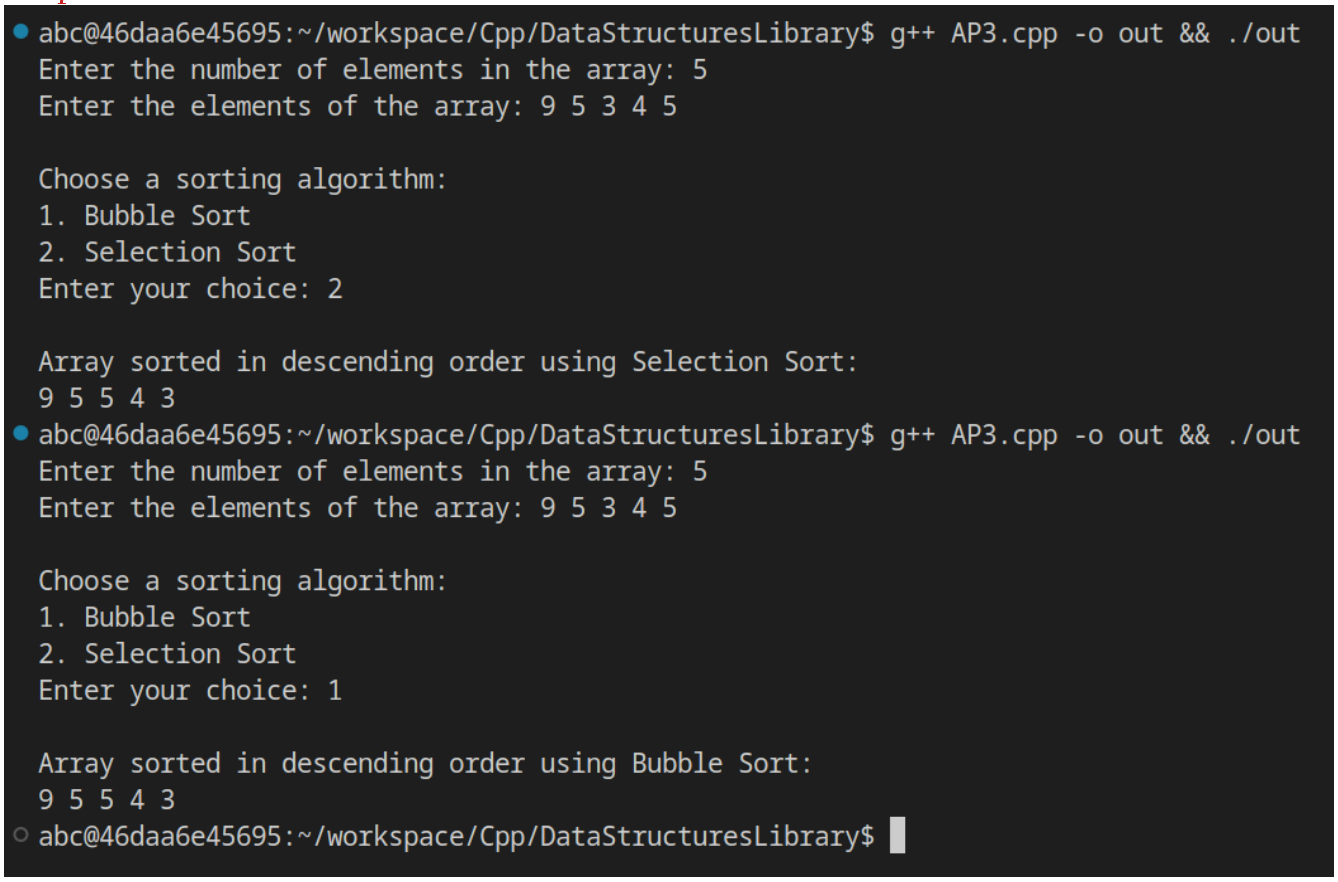


```
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$ g++ AP2.cpp -o out && ./out
Enter the number of elements in the array: 5
Enter 5 elements: 4 7 8 9 11
Enter the element to search for: 7
Linear Search: Element found
Binary Search: Element found
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$
```

AP3 Write a program which takes an array of n integers and sorts the integers in descending order using bubble sort and selection sort techniques.

Code:
```cpp
#include <iostream>
#include "DataStructures/Arrays/StaticArray.hpp"

using namespace std;
using namespace DS;

void bubbleSort(StaticArray<int, 100>& arr, int n) {
    bool swapped;
    for (int i = 0; i < n - 1; i++) {
        swapped = false;
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] < arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
                swapped = true;
            }
        }
        if (!swapped) break;
    }
}

void selectionSort(StaticArray<int, 100>& arr, int n) {
    for (int i = 0; i < n - 1; i++) {
        int max_idx = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] > arr[max_idx]) {
                max_idx = j;
            }
        }
        swap(arr[i], arr[max_idx]);
    }
}

int main() {
    int n, choice;
    StaticArray<int, 100> arr;

    cout << "Enter the number of elements in the array: ";
    cin >> n;
    cout << "Enter the elements of the array: ";
    for (int i = 0; i < n; i++) {
        int x;
        cin >> x;
        arr.PushBack(x);
    }

    cout << "\nChoose a sorting algorithm:\n";
    cout << "1. Bubble Sort\n2. Selection Sort\n";
    cout << "Enter your choice: ";
    cin >> choice;

    switch (choice) {
        case 1:
            bubbleSort(arr, n);
            cout << "\nArray sorted in descending order using Bubble Sort:\n";
            break;
        case 2:
            selectionSort(arr, n);
            cout << "\nArray sorted in descending order using Selection Sort:\n";
            break;
        default:
            cout << "\nInvalid choice!\n";
            return 0;
    }

    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

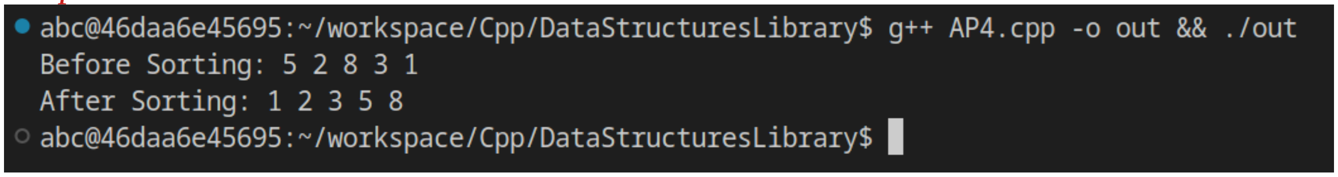
Output:



```
● abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$ g++ AP3.cpp -o out && ./out
  Enter the number of elements in the array: 5
  Enter the elements of the array: 9 5 3 4 5

  Choose a sorting algorithm:
  1. Bubble Sort
  2. Selection Sort
  Enter your choice: 2

  Array sorted in descending order using Selection Sort:
  9 5 5 4 3
● abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$ g++ AP3.cpp -o out && ./out
  Enter the number of elements in the array: 5
  Enter the elements of the array: 9 5 3 4 5

  Choose a sorting algorithm:
  1. Bubble Sort
  2. Selection Sort
  Enter your choice: 1

  Array sorted in descending order using Bubble Sort:
  9 5 5 4 3
○ abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$
```

AP4 Write a program which takes an array of n integers and sorts the integers in ascending order using insertion sort technique.

Code:
```
#include <iostream>
#include "DataStructures/Arrays/StaticArray.hpp"
using namespace DS;
template<typename T, int size>
void insertionSort(StaticArray<T, size>& arr) {
    for (int i = 1; i < arr.Size(); i++) {
        T key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

int main() {
    StaticArray<int, 5> arr;
    arr.PushBack(5);
    arr.PushBack(2);
    arr.PushBack(8);
    arr.PushBack(3);
    arr.PushBack(1);
    std::cout << "Before Sorting: ";
    for (int i = 0; i < arr.Size(); i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;
    insertionSort(arr);
    std::cout << "After Sorting: ";
    for (int i = 0; i < arr.Size(); i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

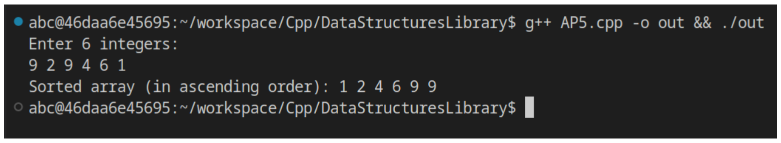
Output:

```
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$ g++ AP4.cpp -o out && ./out
Before Sorting: 5 2 8 3 1
After Sorting: 1 2 3 5 8
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$
```

AP5 Write a program which takes an array of n integers and sorts the integers in ascending order using quick sort technique.

Code:

```cpp
#include <iostream>
#include "DataStructures/Arrays/StaticArray.hpp"
using namespace std;
template<typename T, int N>
int Partition(DS::StaticArray<T, N>& arr, int start, int end) {
    T pivot = arr[end];
    int i = start - 1;
    for (int j = start; j <= end - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            arr.Swap(i, j);
        }
    }
    arr.Swap(i + 1, end);
    return i + 1;
}
template<typename T, int N>
void QuickSort(DS::StaticArray<T, N>& arr, int start, int end) {
    if (start < end) {
        int partition_index = Partition(arr, start, end);
        QuickSort(arr, start, partition_index - 1);
        QuickSort(arr, partition_index + 1, end);
    }
}
int main() {
    const int n = 6;
    DS::StaticArray<int, 6> arr;
    cout << "Enter " << n << " integers:" << endl;
    for (int i = 0; i < n; i++) {
        int x;
        cin >> x;
        arr.PushBack(x);
    }
    QuickSort(arr, 0, n - 1);
    cout << "Sorted array (in ascending order): ";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
    return 0;
}
```

Output:



```
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$ g++ AP5.cpp -o out && ./out
Enter 6 integers:
9 2 9 4 6 1
Sorted array (in ascending order): 1 2 4 6 9 9
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$
```

SLL Node Implementation:

```
#pragma once
#include <memory>

namespace DS {
  template<typename T>
  struct SLLNode {
    T value;
    std::shared_ptr<SLLNode<T>> next;
  };
}
```

Linked List Implementation: DataStructures/LinkedList.hpp

```
#pragma once
#include "../Nodes.hpp"
#include <functional>
#include <stdexcept>

namespace DS {
  template<typename T = int>
  class SinglyLinkedList {
  private:
    std::shared_ptr<SLLNode<T>> head = nullptr, tail = nullptr;
    int size = 0;

  public:
    SinglyLinkedList() = default;

    SinglyLinkedList(T Val) {
      tail = head = std::make_shared<SLLNode<T>>(new SLLNode<T>());
      head->value = Val;
      size++;
    }

    SinglyLinkedList(const SinglyLinkedList& list) {
      std::shared_ptr<SLLNode<T>> Next = list.head;
      while (Next) {
        PushBack(Next->value);
        Next = Next->next;
      }
    }

    ~SinglyLinkedList() {
      Clear();
    }

    void Clear() {
      std::shared_ptr<SLLNode<T>> current_ptr = head;
      while (current_ptr)
      {
        head = current_ptr->next;
        current_ptr = head;
      }
      head = tail = nullptr;
      size = 0;
    }

    void PushFront(const T& value) {
      if (!head) {
        PushBack(value);
        return;
      }
      std::shared_ptr<SLLNode<T>> new_node(new SLLNode<T>());
      new_node->value = value;
      new_node->next = head;
      head = new_node;
      if (!head->next) {
        tail = head;
      }
      size++;
    }

    void PushAfter(std::shared_ptr<SLLNode<T>> Node, const T& value) {
      std::shared_ptr<SLLNode<T>> new_node(new SLLNode<T>());
      new_node->value = value;
      new_node->next = Node->next;
      Node->next = new_node;
      if (Node == tail) {
        tail = new_node;
      }
      size++;
    }

    void PushBack(const T& value) {
      std::shared_ptr<SLLNode<T>> new_node(new SLLNode<T>());
      new_node->value = value;
      if (!tail) {
        tail = head = new_node;
      }
      else {
        tail->next = new_node;
        tail = new_node;
      }
      size++;
    }

    void PushBefore(std::shared_ptr<SLLNode<T>> Node, const T& value) {
      if (Node == head) {
        PushFront(value);
        return;
      }
      std::shared_ptr<SLLNode<T>> start = head;
      while (start->next != Node)
```

```
    {
      start = start->next;
      if (!start) {
        break;
      }
    }
    if (start->next == Node) {
      std::shared_ptr<SLLNode<T>> new_node(new SLLNode<T>());
      new_node->value = value;
      new_node->next = start->next;
      start->next = new_node;
      size++;
      return;
    }
    throw std::out_of_range("Node not in linked list");
  }

  T PopFront() {
    std::shared_ptr<SLLNode<T>> Node = head;
    if (head == tail) {
      head = tail = nullptr;
      size = 0;
      return Node->value;
    }
    head = head->next;
    size -= 1;
    return Node->value;
  }

  T PopAfter(std::shared_ptr<SLLNode<T>> Node) {
    std::shared_ptr<SLLNode<T>> nNode = Node->next;
    if (Node == tail) {
      throw std::out_of_range("End of Linked List Cant Remove");
    }
    if (nNode == tail) {
      Node->next = nullptr;
      tail = Node;
      size -= 1;
      return nNode->value;
    }
    std::shared_ptr<SLLNode<T>> nnNode = Node->next->next;
    Node->next = nnNode;
    size -= 1;
    return nNode->value;
  }

  T PopBack() {
    std::shared_ptr<SLLNode<T>> Node = head;
    if (head == tail) {
      head = tail = nullptr;
      size = 0;
      return Node->value;
    }

    while (Node->next != tail) {
      Node = Node->next;
    }

    std::shared_ptr<SLLNode<T>> pNode = Node->next;
    Node->next = nullptr;
    tail = Node;
    size -= 1;
    return pNode->value;
  }

  T PopBefore(std::shared_ptr<SLLNode<T>> Node) {
    if (Node == head) {
      throw std::out_of_range("Cant Remove Before Head");
    }

    if (head->next == Node) {
      return PopFront();
    }

    std::shared_ptr<SLLNode<T>> ppNode = head;
    while (ppNode->next->next != Node) {
      ppNode = ppNode->next;
    }

    std::shared_ptr<SLLNode<T>> pNode = ppNode->next;
    ppNode->next = ppNode->next->next;

    size -= 1;
    return pNode->value;
  }

  T Pop(std::shared_ptr<SLLNode<T>> Node) {
    if (Node == tail) return PopFront();
    if (Node == head) return PopBack();
    return PopBefore(Node->next);
  }

  std::shared_ptr<SLLNode<T>> GetNodeAt(int n) {
    if (n < 0 || n >= size) {
      return nullptr;
    }

    std::shared_ptr<SLLNode<T>> Node = head;
    for (int i = 0; i < n; i++) {
      Node = Node->next;
    }
    return Node;
  }

  std::shared_ptr<SLLNode<T>> operator[](int n) {
    return GetNodeAt(n);
  }

  std::shared_ptr<SLLNode<T>> find(const T& Val) {
    std::shared_ptr<SLLNode<T>> Node = head;
    while (Node) {
      if (Node->value == Val) {
        return Node;
      }
      Node = Node->next;
    }
    return nullptr;
  }

  template <typename T2>
  std::shared_ptr<SLLNode<T>> findBy(const T2& Val) {
    std::shared_ptr<SLLNode<T>> Node = head;
    while (Node) {
      if (Node->value == Val) {
        return Node;
      }
      Node = Node->next;
    }
    return nullptr;
  }

  bool Empty() {
    return size == 0;
  }

  int Size() {
    return size;
  }

  std::shared_ptr<SLLNode<T>> Head() {
    return head;
  }

  std::shared_ptr<SLLNode<T>> Tail() {
    return tail;
  }
 };
}
```

BP1 Write a menu-driven program which implements a linear linked list

Code:

```
#include <iostream>
#include "DataStructures/LinkedList.hpp"

int main() {
    DS::SinglyLinkedList<int> list;
    char choice;
    int val, pos;
    std::shared_ptr<DS::SLLNode<int>> node;

    do {
        std::cout << "\n\nMenu\n";
        std::cout << "a. Insert element at beginning\n";
        std::cout << "b. Insert element at specific location\n";
        std::cout << "c. Insert element at end\n";
        std::cout << "d. Delete element from beginning\n";
        std::cout << "e. Delete element from specific location\n";
        std::cout << "f. Delete element from end\n";
        std::cout << "g. Display all elements\n";
        std::cout << "h. Search element\n";
        std::cout << "i. Exit\n";
        std::cout << "Enter your choice: ";
        std::cin >> choice;

        switch(choice) {
            case 'a':
                std::cout << "Enter value: ";
                std::cin >> val;
                list.PushFront(val);
                break;

            case 'b':
                std::cout << "Enter position: ";
                std::cin >> pos;
                std::cout << "Enter value: ";
                std::cin >> val;

                node = list.GetNodeAt(pos);
                if (node == nullptr) {
                    std::cout << "Invalid position\n";
                }
                else {
                    list.PushAfter(node, val);
                }
                break;

            case 'c':
                std::cout << "Enter value: ";
                std::cin >> val;
                list.PushBack(val);
```

```cpp
            break;

        case 'd':
            if (list.Size() == 0) {
                std::cout << "List is empty\n";
            }
            else {
                std::cout << "Deleted element: " << list.PopFront() << "\n";
            }
            break;

        case 'e':
            std::cout << "Enter position: ";
            std::cin >> pos;

            node = list.GetNodeAt(pos);
            if (node == nullptr) {
                std::cout << "Invalid position\n";
            }
            else {
                std::cout << "Deleted element: " << list.PopAfter(node) << "\n";
            }
            break;

        case 'f':
            if (list.Size() == 0) {
                std::cout << "List is empty\n";
            }
            else {
                std::cout << "Deleted element: " << list.PopBack() << "\n";
            }
            break;

        case 'g':
            if (list.Size() == 0) {
                std::cout << "List is empty\n";
            }
            else {
                std::cout << "Elements: ";
                auto node = list.Head();
                while (node != nullptr) {
                    std::cout << node -> value << " -> ";
                    node = node -> next;
                }
                std::cout << "nullptr\n";
            }
            break;

        case 'h':
            std::cout << "Enter value: ";
            std::cin >> val;
```

```
                if (list.find(val)) {
                    std::cout << "Element found\n";
                }
                else {
                    std::cout << "Element not found\n";
                }
                break;

            case 'i':
                std::cout << "Exiting program\n";
                break;

            default:
                std::cout << "Invalid choice\n";
        }

    } while (choice != 'i');

    return 0;
}
```

Output:

```
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$ g++ BP1.cpp -o out && ./out


Menu
a. Insert element at beginning
b. Insert element at specific location
c. Insert element at end
d. Delete element from beginning
e. Delete element from specific location
f. Delete element from end
g. Display all elements
h. Search element
i. Exit
Enter your choice: a
Enter value: 5


Menu
a. Insert element at beginning
b. Insert element at specific location
c. Insert element at end
d. Delete element from beginning
e. Delete element from specific location
f. Delete element from end
g. Display all elements
h. Search element
i. Exit
Enter your choice: a
Enter value: 10


Menu
a. Insert element at beginning
b. Insert element at specific location
c. Insert element at end
d. Delete element from beginning
e. Delete element from specific location
f. Delete element from end
g. Display all elements
h. Search element
i. Exit
Enter your choice: b
Enter position: 1
Enter value: 12
```

```
Menu
a. Insert element at beginning
b. Insert element at specific location
c. Insert element at end
d. Delete element from beginning
e. Delete element from specific location
f. Delete element from end
g. Display all elements
h. Search element
i. Exit
Enter your choice: c
Enter value: 30


Menu
a. Insert element at beginning
b. Insert element at specific location
c. Insert element at end
d. Delete element from beginning
e. Delete element from specific location
f. Delete element from end
g. Display all elements
h. Search element
i. Exit
Enter your choice: g
Elements: 10 -> 5 -> 12 -> 30 -> nullptr


Menu
a. Insert element at beginning
b. Insert element at specific location
c. Insert element at end
d. Delete element from beginning
e. Delete element from specific location
f. Delete element from end
g. Display all elements
h. Search element
i. Exit
Enter your choice: d
Deleted element: 10


Menu
a. Insert element at beginning
b. Insert element at specific location
c. Insert element at end
d. Delete element from beginning
e. Delete element from specific location
f. Delete element from end
g. Display all elements
h. Search element
i. Exit
Enter your choice: e
Enter position: 12
Invalid position


Menu
a. Insert element at beginning
b. Insert element at specific location
c. Insert element at end
d. Delete element from beginning
e. Delete element from specific location
f. Delete element from end
g. Display all elements
h. Search element
i. Exit
Enter your choice: f
Deleted element: 30


Menu
a. Insert element at beginning
b. Insert element at specific location
c. Insert element at end
```

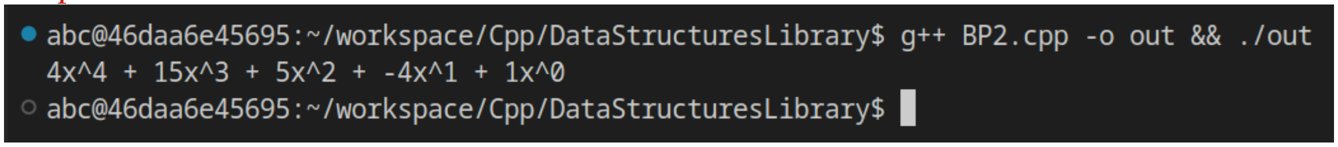
```
d. Delete element from beginning
e. Delete element from specific location
f. Delete element from end
g. Display all elements
h. Search element
i. Exit
Enter your choice: h
Enter value: 9
Element not found


Menu
a. Insert element at beginning
b. Insert element at specific location
c. Insert element at end
d. Delete element from beginning
e. Delete element from specific location
f. Delete element from end
g. Display all elements
h. Search element
i. Exit
Enter your choice: i
Exiting program
```

BP2 A polynomial is composed of different terms where each of them holds a coefficient and an exponent. Write a program to represent the following polynomials: $4x^4 + 4x^3 - 2x^2 + x$ and $11x^3 + 7x^2 - 4x$ with linear linked list, and then perform addition of the given polynomials.

Code:

```cpp
#include "./DataStructures/LinkedList.hpp"
#include <iostream>

using namespace DS;

struct polyterm {
    int pow, coeff;
};

LinkedList<polyterm> addPoly(LinkedList<polyterm> p1, LinkedList<polyterm> p2) {
    LinkedList<polyterm> p3;
    auto n1 = p1.Head();
    auto n2 = p2.Head();

    while (n1 != nullptr && n2 != nullptr) {
        if (n1->value.pow > n2->value.pow) {
            p3.PushBack(n2->value);
            n2 = n2->next;
        }
        else if (n1->value.pow < n2->value.pow) {
            p3.PushBack(n1->value);
            n1 = n1->next;
        }
        else if (n1->value.pow == n2->value.pow) {
            p3.PushBack({n2->value.pow, n1->value.coeff + n2->value.coeff});
            n1 = n1->next;
            n2 = n2->next;
        }
    }

    auto n = n1 == nullptr ? n2 : n1;

    while (n != nullptr) {
        p3.PushBack(n->value);
        n = n->next;
    }

    return p3;
}

int main() {
    LinkedList<polyterm> poly1;
    poly1.PushBack({ 0, 1 });
    poly1.PushBack({ 2, -2 });
    poly1.PushBack({ 3, 4 });
    poly1.PushBack({ 4, 4 });

    LinkedList<polyterm> poly2;
    poly2.PushBack({ 1, -4 });
    poly2.PushBack({ 2, 7 });
    poly2.PushBack({ 3, 11 });

    auto poly3 = addPoly(poly1, poly2);

    for (auto node = poly3.Tail(); node != nullptr; node = node->prev) {
        std::cout << node->value.coeff << "x^" << node->value.pow << (node->prev == nullptr ? "" : " + 
");
    }
    std::cout << std::endl;

    return 0;
}
```

Output:



```
● abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$ g++ BP2.cpp -o out && ./out
  4x^4 + 15x^3 + 5x^2 + -4x^1 + 1x^0
○ abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$ 
```

Assignment Set: C (Stack and Queue)

Stack Implementation: DataStructures/Stack.hpp

```
#pragma once
#include "../LinkedList.hpp"

namespace DS {
  template<typename T, template <typename> class ListType = LinkedList>
  class LinkedListStack {
    ListType<T> stack;
  public:
    LinkedListStack() = default;

    LinkedListStack(const T& val) {
      stack.PushFront(val);
    }

    LinkedListStack(const LinkedListStack& Stack) {
      stack = ListType<T>(Stack.stack);
    }

    T& Top() {
      return stack.Tail()->value;
    }

    T Pop() {
      return stack.PopBack();
    }

    void Push(const T& val) {
      stack.PushBack(val);
    }

    int Size() {
      return stack.Size();
    }

    bool Empty() {
      return stack.Empty();
    }

    ListType<T>& List() {
      return stack;
    }
  };
}
```

Circular Queue Implementation: DataStructures/Queues/StaticArrayQueue.hpp

```
#pragma once
#include "../Array.hpp"
namespace DS {
  template<typename T, int max_size, template <typename, int> class ArrayType = Array>
  class StaticArrayQueue {
    ArrayType<T, max_size + 1> queue;
    int rd = 0, wr = 0;
  public:
    StaticArrayQueue() : queue(max_size + 1), wr(0), rd(0) {
    }
    StaticArrayQueue(const T& val) : queue(max_size + 1), wr(0), rd(0) {
      queue[wr] = val;
      wr++;
    }
    StaticArrayQueue(const StaticArrayQueue& Queue) {
      queue = ArrayType<T, max_size + 1>(Queue.queue);
      rd = Queue.rd;
      wr = Queue.wr;
    }
    void Enqueue(const T& val) {
      int w1 = (wr + 1) % (max_size + 1);
      if (rd - (w1 - 1) != 1) {
        queue[wr] = val;
        wr = w1;
      }
      else
        throw std::out_of_range("Queue is full");
    }
    T Dequeue() {
      int r1 = rd;
      rd = (rd + 1) % (max_size + 1);
      return queue[r1];
    }
    int Size() {
      if (rd <= wr)
        return wr - rd;
      else
        return (max_size - rd + 1) + wr;
    }
    bool Empty() {
      return rd == wr;
    }
    ArrayType<T, max_size + 1>& Array() {
      return queue;
    }
    int MaxSize() {
      return max_size;
    }
  };
}
```

Queue Implementation: DataStructures/Queues/LinkedListQueue.hpp

```
#pragma once
#include "../LinkedList.hpp"

namespace DS {
  template<typename T, template <typename> class ListType = LinkedList>
  class LinkedListQueue {
    ListType<T> queue;
  public:
    LinkedListQueue() = default;
    LinkedListQueue(const T& val) {
      queue.PushFront(val);
    }
    LinkedListQueue(const LinkedListQueue& Queue) {
      queue = ListType<T>(Queue.queue);
    }

    void Enqueue(const T& val) {
      queue.PushBack(val);
    }

    T Dequeue() {
      return queue.PopFront();
    }

    int Size() {
      return queue.Size();
    }

    bool Empty() {
      return queue.Empty();
    }

    ListType<T>& List() {
      return queue;
    }
  };
}
```

CP1. Write a menu-driven program which implements a stack

code:

```
#include <iostream>
#include "DataStructures/Stack.hpp"
int main() {
   DS::StaticStack<int, 5> stack;
   int choice, val;
   do {
      std::cout << "\n\nMenu:\n";
      std::cout << "1. Push\n";
      std::cout << "2. Pop\n";
      std::cout << "3. Print Stack\n";
      std::cout << "4. Exit\n";
      std::cout << "Enter your choice: ";
      std::cin >> choice;
      switch (choice) {
         case 1:
            std::cout << "Enter the value to push: ";
            std::cin >> val;
            try {
               stack.Push(val);
               std::cout << "Pushed " << val << " onto the stack\n";
            }
            catch (const std::out_of_range& e)
               std::cerr << "Error: " << e.what() << "\n";
            break;
         case 2:
            try {
               val = stack.Pop();
               std::cout << "Popped " << val << " from the stack\n";
            }
            catch (const std::out_of_range& e) {
               std::cerr << "Error: " << e.what() << "\n";
            }
            break;
         case 3:
            std::cout << "Elements in the stack: ";
            for (int i = 0; i < stack.Array().Size(); i++)
               std::cout << stack.Array()[i] << " ";
            std::cout << "\n";
            break;
         case 4:
            std::cout << "Exiting...\n";
            break;
         default:
            std::cout << "Invalid choice, please try again\n";
            break;
      }
   } while (choice != 4);
   return 0;
}
```

output:

```
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$ g++ CP1.cpp -o out && ./out


Menu:
Push
Pop
Print Stack
Exit
Enter your choice: 1
Enter the value to push: 20
Pushed 20 onto the stack


Menu:
Push
Pop
Print Stack
Exit
Enter your choice: 1
Enter the value to push: 30
Pushed 30 onto the stack


Menu:
Push
Pop
Print Stack
Exit
Enter your choice: 1
Enter the value to push: 40
Pushed 40 onto the stack


Menu:
Push
Pop
Print Stack
Exit
Enter your choice: 2
Popped 40 from the stack


Menu:
Push
Pop
Print Stack
Exit
Enter your choice: 3
Elements in the stack: 20 30


Menu:
Push
Pop
Print Stack
Exit
Enter your choice: 4
Exiting...
```

CP2. Write a menu-driven program which implements a linear queue

code:
```cpp
#include <iostream>
#include "./DataStructures/Queues/LinkedListQueue.hpp"

using namespace std;
using namespace DS;

int main() {
   LinkedListQueue<int> queue;
   int choice, element;
   do {
      cout << "Queue Operations:" << endl;
      cout << "1. Enqueue" << endl;
      cout << "2. Dequeue" << endl;
      cout << "3. Display" << endl;
      cout << "4. Exit" << endl;
      cout << "Enter your choice: ";
      cin >> choice;
      switch (choice) {
         case 1:
            cout << "Enter the element to enqueue: ";
            cin >> element;
            queue.Enqueue(element);
            cout << "Element enqueued successfully." << endl;
            break;
         case 2:
            if (!queue.Empty()) {
               element = queue.Dequeue();
               cout << "Dequeued element: " << element << endl;
            } else {
               cout << "Queue is empty. Unable to dequeue." << endl;
            }
            break;
         case 3:
            if (!queue.Empty()) {
               cout << "Queue elements: ";
               auto head = queue.List().Head();
               while(head) {
                  cout << head->value << " ";
                  head = head->next;
               }
               cout << endl;
            } else {
               cout << "Queue is empty." << endl;
            }
            break;
         case 4:
            cout << "Exiting program." << endl;
            break;
            default:
                cout << "Invalid choice. Please try again." << endl;
                break;
        }

        cout << endl;
    } while (choice != 4);

    return 0;
}
```

output:

```
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$ g++ CP2.cpp -o out && ./out
Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the element to enqueue: 20
Element enqueued successfully.

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the element to enqueue: 30
Element enqueued successfully.

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the element to enqueue: 40
Element enqueued successfully.

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued element: 20

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 30 40

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
Exiting program.
```

CP3. Write a menu-driven program which implements a circular queue

code:

```cpp
#include <iostream>
#include "./DataStructures/Queues/StaticArrayQueue.hpp"
using namespace std;
using namespace DS;
int main() {
    const int max_size = 10;
    StaticArrayQueue<int, max_size> queue;
    int choice, element;
    do {
        cout << "Queue Operations:" << endl;
        cout << "1. Enqueue" << endl;
        cout << "2. Dequeue" << endl;
        cout << "3. Display" << endl;
        cout << "4. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice) {
            case 1:
                cout << "Enter the element to enqueue: ";
                cin >> element;
                try {
                    queue.Enqueue(element);
                    cout << "Element enqueued successfully." << endl;
                } catch (const std::out_of_range& e)
                    cout << "Queue is full. Unable to enqueue." << endl;
                break;
            case 2:
                if (!queue.Empty()) {
                    element = queue.Dequeue();
                    cout << "Dequeued element: " << element << endl;
                } else
                    cout << "Queue is empty. Unable to dequeue." << endl;
                break;
            case 3:
                if (!queue.Empty()) {
                    cout << "Queue elements: ";
                    for(int i = 0; i < queue.Size(); i++) {
                        auto element = queue.Dequeue();
                        cout << element << " ";
                        queue.Enqueue(element);
                    }
                    cout << endl;
                } else
                    cout << "Queue is empty." << endl;
                break;
            case 4:
                cout << "Exiting program." << endl;
                break;
            default:
                cout << "Invalid choice. Please try again." << endl;
                break;
        }
        cout << endl;
    } while (choice != 4);
    return 0;
}
```

output:

```
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$ g++ CP3.cpp -o out && ./out
Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the element to enqueue: 20
Element enqueued successfully.

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the element to enqueue: 30
Element enqueued successfully.

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the element to enqueue: 40
Element enqueued successfully.

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued element: 20

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 30 40

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
Exiting program.
```

Assignment Set: D (Tree)

Binary Tree Implementation: DataStructures/Trees.hpp

```
#pragma once
#include "../Nodes.hpp"
#include "../Array.hpp"
#include "../Queue.hpp"
#include <iostream>

namespace DS {
  template<typename T>
  class BinaryTree {
  public:
    static const int inOrderSort = 0;
    static const int preOrderSort = 1;
    static const int postOrderSort = 2;
    static const int levelOrderSort = 3;

    private:
    std::shared_ptr<BSTNode<T>> Root;
    int size;

    DynamicArray<std::shared_ptr<BSTNode<T>>> List;

    public:
        BinaryTree() {}
    BinaryTree(const T& RootValue) {
      Root = std::make_shared<BSTNode<T>>(BSTNode<T>());
      Root->value = RootValue;
    }
    BinaryTree(const BinaryTree& BST) {
      size = BST.size;
      LevelOrderTraversal(BST.Root, [&](T& val) -> void { Insert(val); });
    }
    ~BinaryTree() {
      DeleteTree(Root);
    }

        void DeleteTree(std::shared_ptr<BSTNode<T>> node) {
            if (!node) return;
      DeleteTree(node->Left);
      DeleteTree(node->Right);

            if (node == Root) {
        Root = nullptr;
        return;
      }
      if (node->Parent->value <= node->value) {
        node->Parent->Right.reset();
        node.reset();
        return;
      }
      else {
```

```
    node->Parent->Left.reset();
    node.reset();
    return;
   }
    }

    void Insert(const T& value) {
        if (!Root) {
            Root = std::make_shared<BSTNode<T>>(BSTNode<T>());
      Root->value = value;
        } else {
            Queue<std::shared_ptr<BSTNode<T>>> queue;
            queue.Enqueue(Root);

            auto node = std::make_shared<BSTNode<T>>(BSTNode<T>());
            node->value = value;

            while (!queue.Empty()) {
                auto current = queue.Dequeue();
                if (!current->Left) {
                    current->Left = node;
                    node->Parent = current;
                    break;
                } else {
                    queue.Enqueue(current->Left);
                }

                if (!current->Right) {
                    current->Right = node;
                    node->Parent = current;
                    break;
                } else {
                    queue.Enqueue(current->Right);
                }
            }
            size++;
        }
    }

    std::shared_ptr<BSTNode<T>> Find(const T& Value, std::shared_ptr<BSTNode<T>> node) {
   if (!node || node->value == Value) {
            return node;
        }

        auto left_search = Find(Value, node->Left);
        if (left_search) return left_search;

        auto right_search = Find(Value, node->Right);
        if (right_search) return right_search;

        return nullptr;
 }
```

```
  std::shared_ptr<BSTNode<T>> Find(const T& Value) {
   return Find(Value, Root);
  }

    std::shared_ptr<BSTNode<T>> FindMin(std::shared_ptr<BSTNode<T>> root) {
        auto current = Root;
        while (current->Left) {
            current = current->Left;
        }
        return current;
  }

    T Delete(std::shared_ptr<BSTNode<T>> node) {
        T Value = node->value;
        if (isLeaf(node)) {
            if (node == Root) {
                Root = nullptr;
            } else if (node->Parent->Left == node) {
                node->Parent->Left = nullptr;
            } else {
                node->Parent->Right = nullptr;
            }
        }
        else if (!node->Left || !node->Right) {
            auto child = (node->Left) ? node->Left : node->Right;
            if (node == Root) {
                Root = child;
                child->Parent = nullptr;
            } else if (node->Parent->Left == node) {
                node->Parent->Left = child;
                child->Parent = node->Parent;
            } else {
                node->Parent->Right = child;
                child->Parent = node->Parent;
            }
        }
        else {
            auto successor = FindMin(node->Right);
            node->value = successor->value;
            Delete(successor);
        }
        size--;
   return Value;
    }

    bool isLeaf(std::shared_ptr<BSTNode<T>> node) {
   return !node->Left && !node->Right;
  }

    int Size() {
   return size;
  }
  bool Empty() {
    return !Root;
  }

  std::shared_ptr<BSTNode<T>> getRoot() {
    return Root;
  }

    DynamicArray<std::shared_ptr<BSTNode<T>>>& getSortedArray(const int& sortType =
inOrderSort) {
    List.Clear();
    auto Function = [&](std::shared_ptr<BSTNode<T>> node) -> void {
      List.PushBack(node);
    };
    switch (sortType)
    {
    case inOrderSort:
      InOrderTraversal(Root, Function);
      break;
    case preOrderSort:
      PreOrderTraversal(Root, Function);
      break;
    case postOrderSort:
      PostOrderTraversal(Root, Function);
      break;
    case levelOrderSort:
      LevelOrderTraversal(Root, Function);
      break;
    default:
      InOrderTraversal(Root, Function);
      break;
    }
    return List;
  }

  void InOrderTraversal(std::shared_ptr<BSTNode<T>> node,
std::function<void(std::shared_ptr<BSTNode<T>>)> function) {
    if (!node) return;
    InOrderTraversal(node->Left, function);
    function(node);
    InOrderTraversal(node->Right, function);
  }

  void InOrderTraversal(std::function<void(std::shared_ptr<BSTNode<T>>)> function) {
    InOrderTraversal(Root, function);
  }

  void InOrderTraversal(std::shared_ptr<BSTNode<T>> node, std::function<void(T&)> function) {
    if (!node) return;
    InOrderTraversal(node->Left, function);
    function(node->value);
    InOrderTraversal(node->Right, function);
  }

  void InOrderTraversal(std::function<void(T&)> function) {
    InOrderTraversal(Root, function);
  }

  void PreOrderTraversal(std::shared_ptr<BSTNode<T>> node,
std::function<void(std::shared_ptr<BSTNode<T>>)> function) {
    if (!node) return;
    function(node);
    PreOrderTraversal(node->Left, function);
    PreOrderTraversal(node->Right, function);
  }

  void PreOrderTraversal(std::function<void(std::shared_ptr<BSTNode<T>>)> function) {
    PreOrderTraversal(Root, function);
  }

  void PreOrderTraversal(std::shared_ptr<BSTNode<T>> node, std::function<void(T&)> function) {
    if (!node) return;
    function(node->value);
    PreOrderTraversal(node->Left, function);
    PreOrderTraversal(node->Right, function);
  }

  void PreOrderTraversal(std::function<void(T&)> function) {
    PreOrderTraversal(Root, function);
  }

  void PostOrderTraversal(std::shared_ptr<BSTNode<T>> node,
std::function<void(std::shared_ptr<BSTNode<T>>)> function) {
    if (!node) return;
    PostOrderTraversal(node->Left, function);
    PostOrderTraversal(node->Right, function);
    function(node);
  }

  void PostOrderTraversal(std::function<void(std::shared_ptr<BSTNode<T>>)> function) {
    PostOrderTraversal(Root, function);
  }

  void PostOrderTraversal(std::shared_ptr<BSTNode<T>> node, std::function<void(T&)> function) {
    if (!node) return;
    PostOrderTraversal(node->Left, function);
    PostOrderTraversal(node->Right, function);
    function(node->value);
  }

  void PostOrderTraversal(std::function<void(T&)> function) {
    PostOrderTraversal(Root, function);
  }
```

```
    void LevelOrderTraversal(std::shared_ptr<BSTNode<T>> node,
std::function<void(std::shared_ptr<BSTNode<T>>)> function) {
      if (!node) return;
      Queue<std::shared_ptr<BSTNode<T>>> nodes;
      nodes.Enqueue(node);
      while (!nodes.Empty()) {
        node = nodes.Dequeue();
        function(node);
        if (node->Left) nodes.Enqueue(node->Left);
        if (node->Right) nodes.Enqueue(node->Right);
      }
    }

    void LevelOrderTraversal(std::function<void(std::shared_ptr<BSTNode<T>>)> function) {
      LevelOrderTraversal(Root, function);
    }

    void LevelOrderTraversal(std::shared_ptr<BSTNode<T>> node, std::function<void(T&)> function)
{
      if (!node) return;
      Queue<std::shared_ptr<BSTNode<T>>> nodes;
      nodes.Enqueue(node);
      while (!nodes.Empty()) {
        node = nodes.Dequeue();
        function(node->value);
        if (node->Left) nodes.Enqueue(node->Left);
        if (node->Right) nodes.Enqueue(node->Right);
      }
    }

    void LevelOrderTraversal(std::function<void(T&)> function) {
      LevelOrderTraversal(Root, function);
    }

  };
}
```

Binary Search Tree Implementation: DataStructures/Trees.hpp

```
#pragma once
#include "../Nodes.hpp"
#include "../Array.hpp"
#include "../Queue.hpp"
#include <functional>

namespace DS {
  template<typename T>
  class BinarySearchTree {
  public:
    static const int inOrderSort = 0;
    static const int preOrderSort = 1;
    static const int postOrderSort = 2;
    static const int levelOrderSort = 3;

  private:
    std::shared_ptr<BSTNode<T>> Root;
    int size;

    DynamicArray<std::shared_ptr<BSTNode<T>>> List;

  public:
    BinarySearchTree() {}
    BinarySearchTree(const T& RootValue) {
      Root = std::make_shared<BSTNode<T>>(BSTNode<T>());
      Root->value = RootValue;
    }
    BinarySearchTree(const BinarySearchTree& BST) {
      size = BST.size;
      LevelOrderTraversal(BST.Root, [&](T& val) -> void { Insert(val); });
    }
    ~BinarySearchTree() {
      DeleteTree(Root);
    }

    void DeleteTree(std::shared_ptr<BSTNode<T>> node) {
      if (!node) return;
      DeleteTree(node->Left);
      DeleteTree(node->Right);

      if (node == Root) {
        Root = nullptr;
        return;
      }
      if (node->Parent->value <= node->value) {
        node->Parent->Right.reset();
        node.reset();
        return;
      }
      else {
        node->Parent->Left.reset();
        node.reset();
        return;
      }
    }

    void Insert(const T& Value) {
      std::shared_ptr<BSTNode<T>> next = Root;
      if (!next) {
        Root = std::make_shared<BSTNode<T>>(BSTNode<T>());
        Root->value = Value;
        size++;
        return;
      }
      while (true) {
        if (Value < next->value) {
          if (!next->Left) {
            next->Left = std::make_shared<BSTNode<T>>(BSTNode<T>());
            next->Left->value = Value;
            next->Left->Parent = next;
            break;
          }
          next = next->Left;
        }
        else {
          if (!next->Right) {
            next->Right = std::make_shared<BSTNode<T>>(BSTNode<T>());
            next->Right->value = Value;
            next->Right->Parent = next;
            break;
          }
          next = next->Right;
        }
      }
      size++;
    }

    std::shared_ptr<BSTNode<T>> Find(const T& Value, std::shared_ptr<BSTNode<T>> node) {
      std::shared_ptr<BSTNode<T>> next = node;
      while (true) {
        if (next->value == Value) {
          return next;
        }
        if (Value < next->value) {
          if (next->Left) next = next->Left;
          else return next;
        }
        else {
          if (next->Right) next = next->Right;
          else return next;
        }
      }
```

```
}

std::shared_ptr<BSTNode<T>> Find(const T& Value) {
  return Find(Value, Root);
}

std::shared_ptr<BSTNode<T>> Next(std::shared_ptr<BSTNode<T>> node) {
  if (!node->Right) {
    std::shared_ptr<BSTNode<T>> next = node;
    while (true)
    {
      if (next == Root) return next;
      if (next->value < next->Parent->value) return next->Parent;
      next = next->Parent;
    }
  }
  else {
    std::shared_ptr<BSTNode<T>> next = node->Right;
    while (true) {
      if (!next->Left) return next;
      next = next->Left;
    }
  }
}

std::shared_ptr<BSTNode<T>> Prev(std::shared_ptr<BSTNode<T>> node) {
  if (!node->Left) {
    std::shared_ptr<BSTNode<T>> next = node;
    while (true)
    {
      if (next == Root) return next;
      if (next->value >= next->Parent->value) return next->Parent;
      next = next->Parent;
    }
  }
  else {
    std::shared_ptr<BSTNode<T>> next = node->Left;
    while (true) {
      if (!next->Right) return next;
      next = next->Right;
    }
  }
}

std::shared_ptr<BSTNode<T>> Next(const T& Value) {
  return Next(Find(Value));
}

std::shared_ptr<BSTNode<T>> Prev(const T& Value) {
  return Prev(Find(Value));
}
```

```
  DynamicArray<std::shared_ptr<BSTNode<T>>>& RangeSearch(const T& val1, const T& val2) {
    List.Clear();
    std::shared_ptr<BSTNode<T>> next = Find(val1);
    while (next->value <= val2) {
      if (next->value >= val1) {
        List.PushBack(next);
      }
      next = Next(next);
    }
    return List;
  }

  DynamicArray<std::shared_ptr<BSTNode<T>>>& Neighbours(const T& val) {
    List.Clear();
    List.PushBack(Prev(val));
    List.PushBack(Next(val));
    return List;
  }

  DynamicArray<std::shared_ptr<BSTNode<T>>>& Neighbours(std::shared_ptr<BSTNode<T>>
node) {
    List.Clear();
    List.PushBack(Prev(node));
    List.PushBack(Next(node));
    return List;
  }

  T Delete(std::shared_ptr<BSTNode<T>> node) {
    T value = node->value;
    size--;
    if (isLeaf(node)) {
      if (node->Parent) {
        if (node == node->Parent->Right) node->Parent->Right = nullptr;
        else node->Parent->Left = nullptr;
      }
      else {
        Root = nullptr;
      }
      return value;
    }
    std::shared_ptr<BSTNode<T>> next;
    if ((node != Root) && (node->Right)) next = Next(node);
    else next = Prev(node);
    node->value = next->value;
    if (!isLeaf(next)) {
      if (next->Right) {
        next->Right->Parent = next->Parent;
        if (next == next->Parent->Right) next->Parent->Right = next->Right;
        else next->Parent->Left = next->Right;
      }
      else {
        next->Left->Parent = next->Parent;
```

```
          if (next == next->Parent->Right) next->Parent->Left = next->Left;
          else next->Parent->Left = next->Left;
        }
      }
      else {
        if (next == next->Parent->Right) next->Parent->Right = nullptr;
        else next->Parent->Left = nullptr;
      }
      return value;
    }

    bool isLeaf(std::shared_ptr<BSTNode<T>> node) {
      return !node->Left && !node->Right;
    }

    int Size() {
      return size;
    }

    bool Empty() {
      return !Root;
    }

    std::shared_ptr<BSTNode<T>> getRoot() {
      return Root;
    }

    DynamicArray<std::shared_ptr<BSTNode<T>>>& getSortedArray(const int& sortType =
inOrderSort) {
      List.Clear();
      auto Function = [&](std::shared_ptr<BSTNode<T>> node) -> void {
        List.PushBack(node);
      };
      switch (sortType)
      {
      case inOrderSort:
        InOrderTraversal(Root, Function);
        break;
      case preOrderSort:
        PreOrderTraversal(Root, Function);
        break;
      case postOrderSort:
        PostOrderTraversal(Root, Function);
        break;
      case levelOrderSort:
        LevelOrderTraversal(Root, Function);
        break;
      default:
        InOrderTraversal(Root, Function);
        break;
      }
      return List;
```

```
  }

  void InOrderTraversal(std::shared_ptr<BSTNode<T>> node,
std::function<void(std::shared_ptr<BSTNode<T>>)> function) {
    if (!node) return;
    InOrderTraversal(node->Left, function);
    function(node);
    InOrderTraversal(node->Right, function);
  }

  void InOrderTraversal(std::function<void(std::shared_ptr<BSTNode<T>>)> function) {
    InOrderTraversal(Root, function);
  }

  void InOrderTraversal(std::shared_ptr<BSTNode<T>> node, std::function<void(T&)> function) {
    if (!node) return;
    InOrderTraversal(node->Left, function);
    function(node->value);
    InOrderTraversal(node->Right, function);
  }

  void InOrderTraversal(std::function<void(T&)> function) {
    InOrderTraversal(Root, function);
  }

  void PreOrderTraversal(std::shared_ptr<BSTNode<T>> node,
std::function<void(std::shared_ptr<BSTNode<T>>)> function) {
    if (!node) return;
    function(node);
    PreOrderTraversal(node->Left, function);
    PreOrderTraversal(node->Right, function);
  }

  void PreOrderTraversal(std::function<void(std::shared_ptr<BSTNode<T>>)> function) {
    PreOrderTraversal(Root, function);
  }

  void PreOrderTraversal(std::shared_ptr<BSTNode<T>> node, std::function<void(T&)> function) {
    if (!node) return;
    function(node->value);
    PreOrderTraversal(node->Left, function);
    PreOrderTraversal(node->Right, function);
  }

  void PreOrderTraversal(std::function<void(T&)> function) {
    PreOrderTraversal(Root, function);
  }

  void PostOrderTraversal(std::shared_ptr<BSTNode<T>> node,
std::function<void(std::shared_ptr<BSTNode<T>>)> function) {
    if (!node) return;
    PostOrderTraversal(node->Left, function);
    PostOrderTraversal(node->Right, function);
```

```
      function(node);
    }

    void PostOrderTraversal(std::function<void(std::shared_ptr<BSTNode<T>>)> function) {
      PostOrderTraversal(Root, function);
    }

    void PostOrderTraversal(std::shared_ptr<BSTNode<T>> node, std::function<void(T&)> function) {
      if (!node) return;
      PostOrderTraversal(node->Left, function);
      PostOrderTraversal(node->Right, function);
      function(node->value);
    }

    void PostOrderTraversal(std::function<void(T&)> function) {
      PostOrderTraversal(Root, function);
    }

    void LevelOrderTraversal(std::shared_ptr<BSTNode<T>> node,
std::function<void(std::shared_ptr<BSTNode<T>>)> function) {
      if (!node) return;
      Queue<std::shared_ptr<BSTNode<T>>> nodes;
      nodes.Enqueue(node);
      while (!nodes.Empty()) {
        node = nodes.Dequeue();
        function(node);
        if (node->Left) nodes.Enqueue(node->Left);
        if (node->Right) nodes.Enqueue(node->Right);
      }
    }

    void LevelOrderTraversal(std::function<void(std::shared_ptr<BSTNode<T>>)> function) {
      LevelOrderTraversal(Root, function);
    }

    void LevelOrderTraversal(std::shared_ptr<BSTNode<T>> node, std::function<void(T&)> function)
{
      if (!node) return;
      Queue<std::shared_ptr<BSTNode<T>>> nodes;
      nodes.Enqueue(node);
      while (!nodes.Empty()) {
        node = nodes.Dequeue();
        function(node->value);
        if (node->Left) nodes.Enqueue(node->Left);
        if (node->Right) nodes.Enqueue(node->Right);
      }
    }

    void LevelOrderTraversal(std::function<void(T&)> function) {
      LevelOrderTraversal(Root, function);
    }
    std::shared_ptr<BSTNode<T>> Max() {
      std::shared_ptr<BSTNode<T>> next = Root;
      while (true) {
        if (!next->Right) return next;
        next = next->Right;
      }
    }

    std::shared_ptr<BSTNode<T>> Min() {
      std::shared_ptr<BSTNode<T>> next = Root;
      while (true) {
        if (!next->Left) return next;
        next = next->Left;
      }
    }
  };
}
```

Heap Implementation: DataStructures/Stack.hpp

```
#pragma once
#include "../Array.hpp"

namespace DS {
  template <typename T, int max_size, template<typename, int> class ArrayType = Array>
  class BalancedBT {
    ArrayType<T, max_size> Heap;
  public:
    BalancedBT() = default;

    BalancedBT(const BalancedBT& BST) {
      Heap = ArrayType<T, max_size>(BST.Heap);
    }

    int Parent(int i) {
      return i / 2;
    }

    int LeftChild(int i) {
      return 2 * i + 1;
    }

    int RightChild(int i) {
      return 2 * i + 2;
    }

    void ShiftUP(int i) {
      while (i > 0 and Heap[Parent(i)] < Heap[i]) {
        Heap.Swap(Parent(i), i);
        i = Parent(i);
      }
    }

    void ShiftDown(int i) {
      while (true) {
        int maxIndex = i;
        int l = LeftChild(i);
        if (l < Size() && Heap[l] > Heap[maxIndex]) {
          maxIndex = l;
        }
        int r = RightChild(i);
        if (r < Size() && Heap[r] > Heap[maxIndex]) {
          maxIndex = r;
        }
        if (i != maxIndex) {
          Heap.Swap(i, maxIndex);
          i = maxIndex;
        }
        else {
          break;
        }
      }
    }

    void Insert(const T& p) {
      Heap.PushBack(p);
      ShiftUP(Size() - 1);
    }

    T& Extract() {
      Heap.Swap(0, Size() - 1);
      T& result = Heap[Size() - 1];
      Heap.setSize(Size() - 1);
      ShiftDown(0);
      return result;
    }

    T& operator[](int i) {
      return Heap[i];
    }

    int Size() {
      return Heap.Size();
    }

    bool Empty() {
      return Heap.Size() == 0;
    }

    ArrayType<T, max_size>& Array() {
      return Heap;
    }

    int MaxSize() {
      return Heap.MaxSize();
    }
  };
}
```

Heap Sort Implementation: DataStructures/Algorithms/HeapSort.hpp

```
#pragma once
#include "../Queues/PriorityQueue.hpp"
namespace DS {
  template<typename T, int max_size, template <typename, int> class ArrayType = DS::Array>
  void HeapSort(ArrayType<T, max_size>& array) {
    DS::PriorityQueue<T, max_size, ArrayType> pQ;
    for (int i = 0; i < array.Size(); i++)
      pQ.Insert(array[i]);
    array.Clear();
    while (!pQ.Empty())
      array.PushBack(pQ.ExtractMax());
    for (int i = 0; i < array.Size() / 2; i++)
      array.Swap(i, array.Size() - i - 1);
  }
  template<typename T, int max_size, template <typename, int> class ArrayType = DS::Array>
  void ShiftDown(int i, ArrayType<T, max_size>& Heap) {
    while (true) {
      int maxIndex = i;
      int l = 2 * i + 1;
      if (l < Heap.Size() && Heap[l] > Heap[maxIndex]) maxIndex = l;
      int r = 2 * i + 2;
      if (r < Heap.Size() && Heap[r] > Heap[maxIndex]) maxIndex = r;
      if (i != maxIndex) {
        Heap.Swap(i, maxIndex);
        i = maxIndex;
      }
      else break;
    }
  }
  template<typename T, int max_size, template <typename, int> class ArrayType =   DS::Array>
  void BuildHeap(ArrayType<T, max_size>& Heap) {
    for (int j = Heap.Size() / 2; j >= 0; j--)
      ShiftDown<T, max_size, ArrayType>(j, Heap);
  }
  template<typename T, int max_size, template <typename, int> class ArrayType = DS::Array>
  void inPlaceHeapSort(ArrayType<T, max_size>& array) {
    BuildHeap<T, max_size, ArrayType>(array);
    int n = array.Size();
    for (int i = 0; i < n - 1; i++) {
      array.Swap(0, array.Size() - 1);
      array.setSize(array.Size() - 1);
      ShiftDown<T, max_size, ArrayType>(0, array);
    }
    array.setSize(n);
  }
}
```

DP1. Write a menu-driven program which implements a binary tree

code:

```
#include <iostream>
#include "DataStructures/Trees.hpp"

using namespace std;
using namespace DS;

void displayMenu() {
    cout << "Menu:\n";
    cout << "a) Insertion of a node\n";
    cout << "b) Deletion of a node\n";
    cout << "c) Preorder traversal\n";
    cout << "d) Inorder traversal\n";
    cout << "e) Postorder traversal\n";
    cout << "f) Determine total number of leaf nodes\n";
    cout << "q) Quit\n";
    cout << "Enter your choice: ";
}

int main() {
    BinaryTree<int> tree;
    char choice;
    int value;

    do {
        displayMenu();
        cin >> choice;

        int leafCount = 0;

        switch (choice) {
            case 'a':
                cout << "Enter the value to insert: ";
                cin >> value;
                tree.Insert(value);
                cout << "Node inserted.\n";
                break;
            case 'b':
                cout << "Enter the value to delete: ";
                cin >> value;

                if (tree.Delete(tree.Find(value))) {
                    cout << "Node deleted.\n";
                } else {
                    cout << "Node not found.\n";
                }
                break;
            case 'c':
                cout << "Preorder traversal: ";
                tree.PreOrderTraversal([](int& val) {
```

```
                cout << val << " ";
            });
            cout << endl;
            break;
        case 'd':
            cout << "Inorder traversal: ";
            tree.InOrderTraversal([](int& val) {
                cout << val << " ";
            });
            cout << endl;
            break;
        case 'e':
            cout << "Postorder traversal: ";
            tree.PostOrderTraversal([](int& val) {
                cout << val << " ";
            });
            cout << endl;
            break;
        case 'f':
            tree.PostOrderTraversal([&leafCount](std::shared_ptr<BSTNode<int>> node) {
                if (!node->Left && !node->Right) leafCount++;
            });
            cout << "Total number of leaf nodes: " << leafCount << endl;
            break;
        case 'q':
            cout << "Quitting program.\n";
            break;
        default:
            cout << "Invalid choice. Please try again.\n";
            break;
        }
    } while (choice != 'q');

    return 0;
}
```

output:

```
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$ g++ DP1.cpp -o out && ./out
Menu:
a) Insertion of a node
b) Deletion of a node
c) Preorder traversal
d) Inorder traversal
e) Postorder traversal
f) Determine total number of leaf nodes
q) Quit
Enter your choice: a
Enter the value to insert: 10
Node inserted.
Menu:
a) Insertion of a node
b) Deletion of a node
c) Preorder traversal
d) Inorder traversal
e) Postorder traversal
f) Determine total number of leaf nodes
q) Quit
Enter your choice: a
Enter the value to insert: 20
Node inserted.
Menu:
a) Insertion of a node
b) Deletion of a node
c) Preorder traversal
d) Inorder traversal
e) Postorder traversal
f) Determine total number of leaf nodes
q) Quit
Enter your choice: a
Enter the value to insert: 30
Node inserted.
Menu:
a) Insertion of a node
b) Deletion of a node
c) Preorder traversal
d) Inorder traversal
e) Postorder traversal
f) Determine total number of leaf nodes
q) Quit
Enter your choice: a
Enter the value to insert: 40
Node inserted.
Menu:
a) Insertion of a node
b) Deletion of a node
c) Preorder traversal
d) Inorder traversal
e) Postorder traversal
f) Determine total number of leaf nodes
q) Quit
Enter your choice: b
Enter the value to delete: 40
Node deleted.
Menu:
a) Insertion of a node
b) Deletion of a node
c) Preorder traversal
d) Inorder traversal
e) Postorder traversal
f) Determine total number of leaf nodes
q) Quit
Enter your choice: c
Preorder traversal: 10 20 30
Menu:
a) Insertion of a node
b) Deletion of a node
c) Preorder traversal
d) Inorder traversal
e) Postorder traversal
f) Determine total number of leaf nodes
q) Quit
Enter your choice: d
Inorder traversal: 20 10 30
Menu:
a) Insertion of a node
b) Deletion of a node
c) Preorder traversal
d) Inorder traversal
e) Postorder traversal
f) Determine total number of leaf nodes
q) Quit
Enter your choice: e
Postorder traversal: 20 30 10
Menu:
a) Insertion of a node
b) Deletion of a node
c) Preorder traversal
d) Inorder traversal
e) Postorder traversal
f) Determine total number of leaf nodes
q) Quit
Enter your choice: f
Total number of leaf nodes: 2
Menu:
a) Insertion of a node
```

```
b) Deletion of a node
c) Preorder traversal
d) Inorder traversal
e) Postorder traversal
f) Determine total number of leaf nodes
q) Quit
Enter your choice: q
Quitting program.
```

DP2. Write a menu-driven program which implements a heap

code:

```
#include <iostream>
#include "DataStructures/Trees.hpp"
using namespace std;
using namespace DS;
void displayMenu() {
   cout << "Menu:\n";
   cout << "a) Insertion of a node\n";
   cout << "b) Deletion of a node\n";
   cout << "c) Display\n";
   cout << "q) Quit\n";
   cout << "Enter your choice: ";
}
int main() {
   BalancedBT<int, 100> heap;
   char choice;
   int value;
   do {
      displayMenu();
      cin >> choice;
      switch (choice) {
         case 'a':
            cout << "Enter the value to insert: ";
            cin >> value;
            heap.Insert(value);
            cout << "Node inserted.\n";
            break;
         case 'b':
            if (!heap.Empty()) {
               int extractedValue = heap.Extract();
               cout << "Node with value " << extractedValue << " deleted.\n";
            } else cout << "Heap is empty. Unable to delete.\n";
            break;
         case 'c':
            if (!heap.Empty()) {
               cout << "Heap elements: ";
               for (int i = 0; i < heap.Size(); i++) cout << heap[i] << " ";
               cout << endl;
            } else cout << "Heap is empty.\n";
            break;
         case 'q':
            cout << "Quitting program.\n";
            break;
         default:
            cout << "Invalid choice. Please try again.\n";
            break;
      }
   } while (choice != 'q');
   return 0;
```

```
}
```

output:

```
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$ g++ DP2.cpp -o out && ./out
Menu:
a) Insertion of a node
b) Deletion of a node
c) Display
q) Quit
Enter your choice: a
Enter the value to insert: 10
Node inserted.
Menu:
a) Insertion of a node
b) Deletion of a node
c) Display
q) Quit
Enter your choice: a
Enter the value to insert: 5
Node inserted.
Menu:
a) Insertion of a node
b) Deletion of a node
c) Display
q) Quit
Enter your choice: a
Enter the value to insert: 30
Node inserted.
Menu:
a) Insertion of a node
b) Deletion of a node
c) Display
q) Quit
Enter your choice: a
Enter the value to insert: 100
Node inserted.
Menu:
a) Insertion of a node
b) Deletion of a node
c) Display
q) Quit
Enter your choice: b
Node with value 100 deleted.
Menu:
a) Insertion of a node
b) Deletion of a node
c) Display
q) Quit
Enter your choice: c
Heap elements: 30 10 5
Menu:
a) Insertion of a node
b) Deletion of a node
c) Display
q) Quit
Enter your choice: q
Quitting program.
```

DP3. Write a menu-driven program which implements a binary search tree

code:

```
#include <iostream>
#include "DataStructures/Trees.hpp"

using namespace DS;
using namespace std;

int main() {
    BinarySearchTree<int> bst;
    int choice, value;

    do {
        cout << "Binary Search Tree Menu\n";
        cout << "1. Insert a node\n";
        cout << "2. Delete a node\n";
        cout << "3. Preorder traversal\n";
        cout << "4. Inorder traversal\n";
        cout << "5. Postorder traversal\n";
        cout << "6. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter the value to insert: ";
                cin >> value;
                bst.Insert(value);
                cout << "Node inserted.\n";
                break;
            case 2:
                cout << "Enter the value to delete: ";
                cin >> value;
                if (bst.Find(value)) {
                    bst.Delete(bst.Find(value));
                    cout << "Node deleted.\n";
                } else {
                    cout << "Node not found.\n";
                }
                break;
            case 3:
                cout << "Preorder Traversal: ";
                bst.PreOrderTraversal([](int& val) {
                    cout << val << " ";
                });
                cout << endl;
                break;
            case 4:
                cout << "Inorder Traversal: ";
                bst.InOrderTraversal([](int& val) {
```

```
                cout << val << " ";
            });
            cout << endl;
            break;
        case 5:
            cout << "Postorder Traversal: ";
            bst.PostOrderTraversal([](int& val) {
                cout << val << " ";
            });
            cout << endl;
            break;
        case 6:
            cout << "Exiting program.\n";
            break;
        default:
            cout << "Invalid choice. Please try again.\n";
            break;
        }
        cout << endl;

    } while (choice != 6);

    return 0;
}
```

output:

```
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$ g++ DP3.cpp -o out && ./out
Binary Search Tree Menu
1. Insert a node
2. Delete a node
3. Preorder traversal
4. Inorder traversal
5. Postorder traversal
6. Exit
Enter your choice: 1
Enter the value to insert: 20
Node inserted.

Binary Search Tree Menu
1. Insert a node
2. Delete a node
3. Preorder traversal
4. Inorder traversal
5. Postorder traversal
6. Exit
Enter your choice: 1
Enter the value to insert: 30
Node inserted.

Binary Search Tree Menu
1. Insert a node
2. Delete a node
3. Preorder traversal
4. Inorder traversal
5. Postorder traversal
6. Exit
Enter your choice: 1
Enter the value to insert: 10
Node inserted.

Binary Search Tree Menu
1. Insert a node
2. Delete a node
```

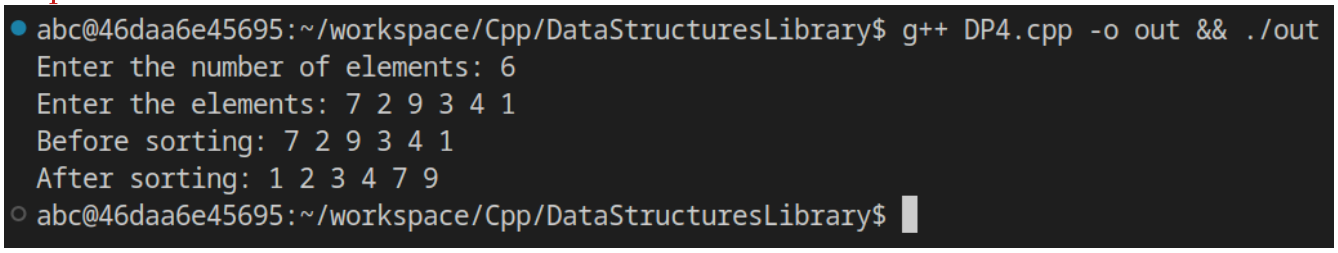
```
3. Preorder traversal
4. Inorder traversal
5. Postorder traversal
6. Exit
Enter your choice: 1
Enter the value to insert: 5
Node inserted.

Binary Search Tree Menu
1. Insert a node
2. Delete a node
3. Preorder traversal
4. Inorder traversal
5. Postorder traversal
6. Exit
Enter your choice: 2
Enter the value to delete: 5
Node deleted.

Binary Search Tree Menu
1. Insert a node
2. Delete a node
3. Preorder traversal
4. Inorder traversal
5. Postorder traversal
6. Exit
Enter your choice: 3
Preorder Traversal: 20 10 30

Binary Search Tree Menu
1. Insert a node
2. Delete a node
3. Preorder traversal
4. Inorder traversal
5. Postorder traversal
6. Exit
Enter your choice: 4
Inorder Traversal: 10 20 30

Binary Search Tree Menu
1. Insert a node
2. Delete a node
3. Preorder traversal
4. Inorder traversal
5. Postorder traversal
6. Exit
Enter your choice: 5
Postorder Traversal: 10 30 20

Binary Search Tree Menu
1. Insert a node
2. Delete a node
3. Preorder traversal
4. Inorder traversal
5. Postorder traversal
6. Exit
Enter your choice: 6
Exiting program.
```

DP4. Write a program which takes an array of n integers and sorts the integers in ascending order using heap sort technique

code:

```cpp
#include <iostream>
#include "DataStructures/Algorithms/HeapSort.hpp"

using namespace DS;
using namespace std;

int main() {
    DynamicArray<int> array;

    int n = 0;
    cout << "Enter the number of elements: ";
    cin >> n;
    cout << "Enter the elements: ";
    for (int i = 0; i < n; i++) {
        int x;
        cin >> x;
        array.PushBack(x);
    }

    cout << "Before sorting: ";
    for (int i = 0; i < array.Size(); ++i) {
        cout << array[i] << " ";
    }
    cout << endl;

    inPlaceHeapSort<int, 1, DynamicArray>(array);

    cout << "After sorting: ";
    for (int i = 0; i < array.Size(); ++i) {
        cout << array[i] << " ";
    }
    cout << endl;

    return 0;
}
```

output:



```
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$ g++ DP4.cpp -o out && ./out
Enter the number of elements: 6
Enter the elements: 7 2 9 3 4 1
Before sorting: 7 2 9 3 4 1
After sorting: 1 2 3 4 7 9
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$
```

Assignment Set: E (Graphs)

EP1. Write a menu-driven program which implements a graph

code:

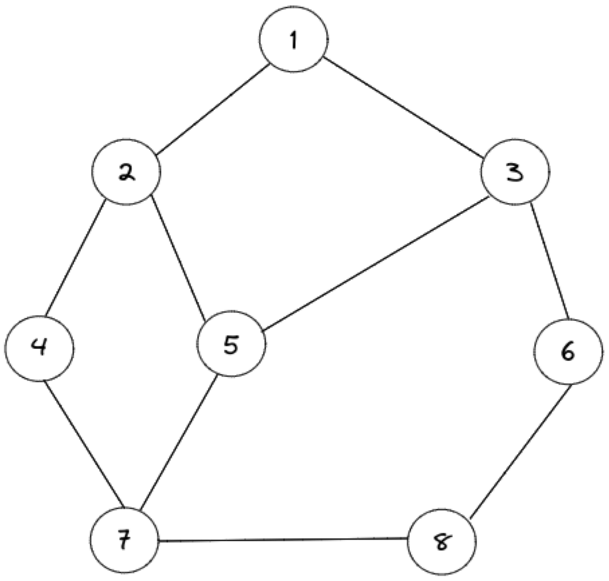
```
#include <iostream>
#include <vector>

class Graph {
private:
    int numVertices;
    std::vector<std::vector<bool>> adjacencyMatrix;

public:
    Graph(int n) {
        numVertices = n;
        adjacencyMatrix.resize(numVertices, std::vector<bool>(numVertices, false));
    }

    void insertVertex() {
        numVertices++;
        adjacencyMatrix.resize(numVertices, std::vector<bool>(numVertices, false));
        for (int i = 0; i < numVertices - 1; i++) {
            adjacencyMatrix[i].resize(numVertices, false);
        }
        std::cout << "Vertex inserted successfully!" << std::endl;
    }

    void insertEdge(int u, int v) {
        if (u >= 0 && u < numVertices && v >= 0 && v < numVertices) {
            adjacencyMatrix[u][v] = true;
            adjacencyMatrix[v][u] = true;
            std::cout << "Edge inserted successfully!" << std::endl;
        } else {
            std::cout << "Invalid vertex indices!" << std::endl;
        }
    }

    void deleteVertex(int v) {
        if (v >= 0 && v < numVertices) {
            adjacencyMatrix.erase(adjacencyMatrix.begin() + v);
            for (int i = 0; i < numVertices; i++) {
                adjacencyMatrix[i].erase(adjacencyMatrix[i].begin() + v);
            }
            numVertices--;
            std::cout << "Vertex deleted successfully!" << std::endl;
        } else {
            std::cout << "Invalid vertex index!" << std::endl;
        }
    }

    void deleteEdge(int u, int v) {
        if (u >= 0 && u < numVertices && v >= 0 && v < numVertices) {
            adjacencyMatrix[u][v] = false;
            adjacencyMatrix[v][u] = false;
            std::cout << "Edge deleted successfully!" << std::endl;
        } else {
            std::cout << "Invalid vertex indices!" << std::endl;
        }
    }

    void calculateDegree() {
        for (int i = 0; i < numVertices; i++) {
            int degree = 0;
            for (int j = 0; j < numVertices; j++) {
                if (adjacencyMatrix[i][j])
                    degree++;
            }
            std::cout << "Degree of vertex " << i << ": " << degree << std::endl;
        }
    }

    void calculateSelfLoops() {
        int count = 0;
        for (int i = 0; i < numVertices; i++) {
            if (adjacencyMatrix[i][i])
                count++;
        }
        std::cout << "Number of self-loops: " << count << std::endl;
    }
};

int main() {
    int numVertices;
    std::cout << "Enter the number of vertices: ";
    std::cin >> numVertices;

    Graph graph(numVertices);

    int choice;
    do {
        std::cout << "\n--- Graph Operations ---" << std::endl;
        std::cout << "1. Insert a vertex" << std::endl;
        std::cout << "2. Insert an edge" << std::endl;
        std::cout << "3. Delete a vertex" << std::endl;
        std::cout << "4. Delete an edge" << std::endl;
        std::cout << "5. Calculate degree of each vertex" << std::endl;
        std::cout << "6. Calculate number of self-loops" << std::endl;
        std::cout << "0. Exit" << std::endl;
        std::cout << "Enter your choice: ";
        std::cin >> choice;

        switch (choice) {
            case 1: {
                graph.insertVertex();
                break;
            }
            case 2: {
                int u, v;
                std::cout << "Enter the indices of the vertices to connect: ";
                std::cin >> u >> v;
                graph.insertEdge(u, v);
                break;
            }
            case 3: {
                int v;
                std::cout << "Enter the index of the vertex to delete: ";
                std::cin >> v;
                graph.deleteVertex(v);
                break;
            }
            case 4: {
                int u, v;
                std::cout << "Enter the indices of the vertices to disconnect: ";
                std::cin >> u >> v;
                graph.deleteEdge(u, v);
                break;
            }
            case 5: {
                graph.calculateDegree();
                break;
            }
            case 6: {
                graph.calculateSelfLoops();
                break;
            }
            case 0: {
                std::cout << "Exiting..." << std::endl;
                break;
            }
            default: {
                std::cout << "Invalid choice!" << std::endl;
                break;
            }
        }
    } while (choice != 0);

    return 0;
}
```

Output:

```
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$ g++ EP1.cpp -o out && ./out
Enter the number of vertices: 7

--- Graph Operations ---
1. Insert a vertex
2. Insert an edge
3. Delete a vertex
4. Delete an edge
5. Calculate degree of each vertex
6. Calculate number of self-loops
0. Exit
Enter your choice: 1
Vertex inserted successfully!

--- Graph Operations ---
1. Insert a vertex
2. Insert an edge
3. Delete a vertex
4. Delete an edge
5. Calculate degree of each vertex
6. Calculate number of self-loops
0. Exit
Enter your choice: 2
Enter the indices of the vertices to connect: 0 2
Edge inserted successfully!

--- Graph Operations ---
1. Insert a vertex
2. Insert an edge
3. Delete a vertex
4. Delete an edge
5. Calculate degree of each vertex
6. Calculate number of self-loops
0. Exit
Enter your choice: 2
Enter the indices of the vertices to connect: 0 0
Edge inserted successfully!

--- Graph Operations ---
1. Insert a vertex
2. Insert an edge
3. Delete a vertex
4. Delete an edge
5. Calculate degree of each vertex
6. Calculate number of self-loops
0. Exit
Enter your choice: 2
Enter the indices of the vertices to connect: 6 6
Edge inserted successfully!

--- Graph Operations ---
1. Insert a vertex
2. Insert an edge
3. Delete a vertex
4. Delete an edge
5. Calculate degree of each vertex
6. Calculate number of self-loops
0. Exit
Enter your choice: 2
Enter the indices of the vertices to connect: 1 2
Edge inserted successfully!

--- Graph Operations ---
1. Insert a vertex
2. Insert an edge
3. Delete a vertex
4. Delete an edge
5. Calculate degree of each vertex
6. Calculate number of self-loops
0. Exit
Enter your choice: 4
Enter the indices of the vertices to disconnect: 1 2
Edge deleted successfully!

--- Graph Operations ---
```

```
1. Insert a vertex
2. Insert an edge
3. Delete a vertex
4. Delete an edge
5. Calculate degree of each vertex
6. Calculate number of self-loops
0. Exit
Enter your choice: 5
Degree of vertex 0: 2
Degree of vertex 1: 0
Degree of vertex 2: 1
Degree of vertex 3: 0
Degree of vertex 4: 0
Degree of vertex 5: 0
Degree of vertex 6: 1
Degree of vertex 7: 0

--- Graph Operations ---
1. Insert a vertex
2. Insert an edge
3. Delete a vertex
4. Delete an edge
5. Calculate degree of each vertex
6. Calculate number of self-loops
0. Exit
Enter your choice: 6
Number of self-loops: 2

--- Graph Operations ---
1. Insert a vertex
2. Insert an edge
3. Delete a vertex
4. Delete an edge
5. Calculate degree of each vertex
6. Calculate number of self-loops
0. Exit
Enter your choice: 0
Exiting...
```

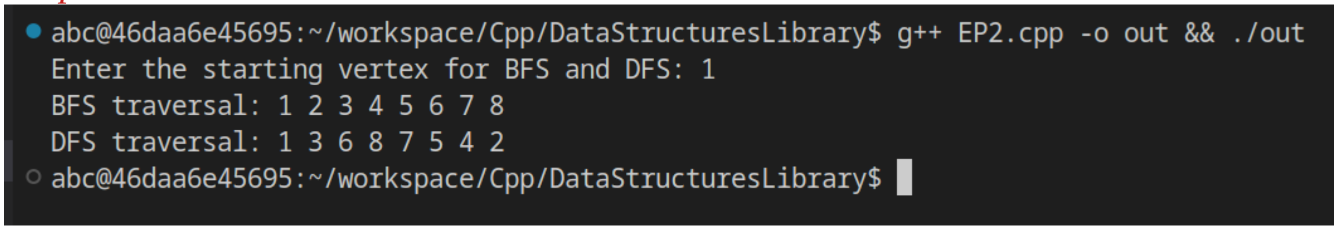
EP2. Write a program to travers the following graph using breadth first search and depth first search techniques.



Code:

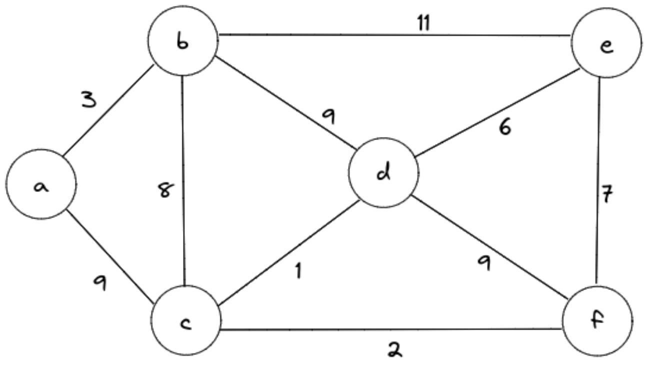
```
#include <iostream>
#include <queue>
#include <stack>
#include <vector>

class Graph {
private:
    int numVertices;
    std::vector<std::vector<int>> adjacencyList;

public:
    Graph(int n) {
        numVertices = n;
        adjacencyList.resize(numVertices);
    }

    void addEdge(int u, int v) {
        if (u >= 0 && u < numVertices && v >= 0 && v < numVertices) {
            adjacencyList[u].push_back(v);
            adjacencyList[v].push_back(u);
        } else {
            std::cout << "Invalid vertex indices!" << std::endl;
        }
    }

    void breadthFirstSearch(int startVertex) {
        std::vector<bool> visited(numVertices, false);
        std::queue<int> q;

        visited[startVertex] = true;
        q.push(startVertex);

        std::cout << "BFS traversal: ";

        while (!q.empty()) {
            int currentVertex = q.front();
            q.pop();
            std::cout << currentVertex + 1 << " ";

            for (int adjacentVertex : adjacencyList[currentVertex]) {
                if (!visited[adjacentVertex]) {
                    visited[adjacentVertex] = true;
                    q.push(adjacentVertex);
                }
            }
        }

        std::cout << std::endl;
    }

    void depthFirstSearch(int startVertex) {
        std::vector<bool> visited(numVertices, false);
        std::stack<int> s;

        visited[startVertex] = true;
        s.push(startVertex);

        std::cout << "DFS traversal: ";

        while (!s.empty()) {
            int currentVertex = s.top();
            s.pop();
            std::cout << currentVertex + 1 << " ";

            for (int adjacentVertex : adjacencyList[currentVertex]) {
                if (!visited[adjacentVertex]) {
                    visited[adjacentVertex] = true;
                    s.push(adjacentVertex);
                }
            }
        }

        std::cout << std::endl;
    }
};

int main() {
    Graph graph(8);

    graph.addEdge(0, 1);
    graph.addEdge(0, 2);
    graph.addEdge(1, 3);
    graph.addEdge(1, 4);
    graph.addEdge(2, 5);
    graph.addEdge(3, 6);
    graph.addEdge(4, 6);
    graph.addEdge(5, 7);
    graph.addEdge(6, 7);

    int startVertex;
    std::cout << "Enter the starting vertex for BFS and DFS: ";
    std::cin >> startVertex;
    startVertex--;
    graph.breadthFirstSearch(startVertex);
    graph.depthFirstSearch(startVertex);

    return 0;
}
```

Output:



```
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$ g++ EP2.cpp -o out && ./out
Enter the starting vertex for BFS and DFS: 1
BFS traversal: 1 2 3 4 5 6 7 8
DFS traversal: 1 3 6 8 7 5 4 2
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$
```

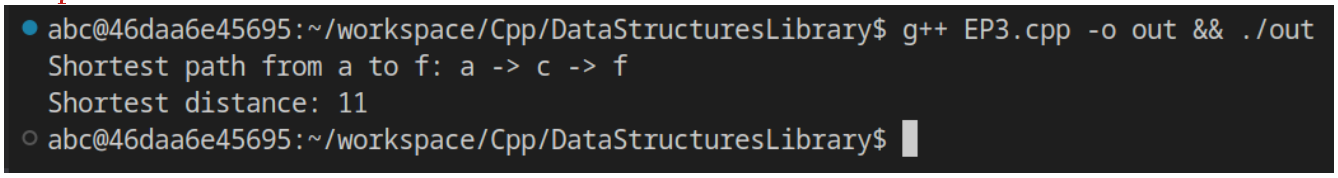
EP3. Write a program to determine shortest path from a to f (using Dijkstra's algorithm) in the following graph



Code:

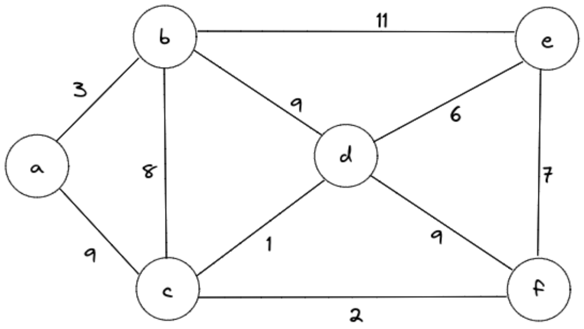
```
#include <iostream>
#include <limits>
#include <vector>
class DijkstraShortestPath {
private:
    int numVertices;
    std::vector<std::vector<int>> graph;
public:
    DijkstraShortestPath(int vertices)
        : numVertices(vertices), graph(vertices, std::vector<int>(vertices, 0)) {}
    void addEdge(int source, int destination, int weight) {
        graph[source - 'a'][destination - 'a'] = weight;
        graph[destination - 'a'][source - 'a'] = weight;
    }
    int findMinDistance(const std::vector<int>& distances, const std::vector<bool>& visited) {
        int minDistance = std::numeric_limits<int>::max();
        int minIndex = -1;
        for (int i = 0; i < numVertices; ++i) {
            if (!visited[i] && distances[i] < minDistance) {
                minDistance = distances[i];
                minIndex = i;
            }
        }
        return minIndex;
    }
    void printPath(const std::vector<int>& parent, int destination) {
        if (parent[destination] == -1) {
            std::cout << char('a' + destination);
            return;
        }
        printPath(parent, parent[destination]);
        std::cout << " -> " << char('a' + destination);
    }
    void findShortestPath(int source, int destination) {
        std::vector<int> distances(numVertices, std::numeric_limits<int>::max());
        std::vector<bool> visited(numVertices, false);
        std::vector<int> parent(numVertices, -1);

        distances[source] = 0;
        for (int count = 0; count < numVertices - 1; ++count) {
            int current = findMinDistance(distances, visited);
            visited[current] = true;
            for (int i = 0; i < numVertices; ++i) {
                if (!visited[i] && graph[current][i] != 0 &&
                    distances[current] != std::numeric_limits<int>::max() &&
                    distances[current] + graph[current][i] < distances[i]) {
                    distances[i] = distances[current] + graph[current][i];
                    parent[i] = current;
                }
            }
        }
        std::cout << "Shortest path from " << char('a' + source) << " to " << char('a' + destination)
<< ": ";
        printPath(parent, destination);
        std::cout << "\nShortest distance: " << distances[destination] << std::endl;
    }
};

int main() {
    DijkstraShortestPath graph(6);
    // Adding edges to the graph
    graph.addEdge('a', 'b', 3);
    graph.addEdge('a', 'c', 9);
    graph.addEdge('b', 'c', 8);
    graph.addEdge('b', 'd', 9);
    graph.addEdge('b', 'e', 11);
    graph.addEdge('c', 'd', 1);
    graph.addEdge('c', 'f', 2);
    graph.addEdge('d', 'e', 6);
    graph.addEdge('d', 'f', 9);
    graph.addEdge('e', 'f', 7);
    graph.findShortestPath(0, 5);
    return 0;
}
```

Output:



```
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$ g++ EP3.cpp -o out && ./out
Shortest path from a to f: a -> c -> f
Shortest distance: 11
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$
```

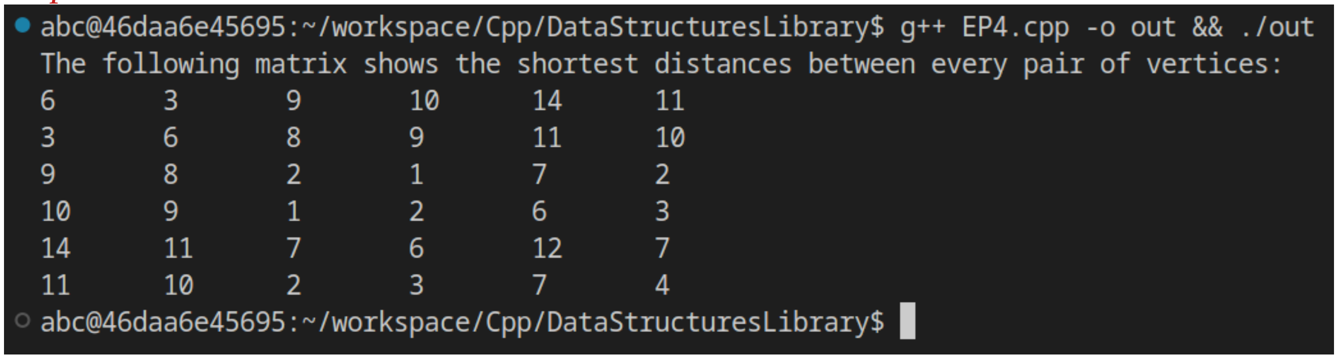
EP4. Write a program to determine shortest paths between every pair of vertices (using Floyd Warshall‟s algorithm) in the following graph.



code:

```
#include <iostream>
#include <climits>
#include <vector>

using namespace std;

class Graph {
private:
    std::vector<std::vector<int>> graph;
    int numVertices;
public:
    Graph(int vertices) : numVertices(vertices), graph(vertices, std::vector<int>(vertices, INT_MAX))
{}
    void addEdge(int i, int j, int weight) {
        graph[i-'a'][j-'a'] = weight;
        graph[j-'a'][i-'a'] = weight;
    }
    void floydWarshall() {
        std::vector<std::vector<int>> dist(graph);
        for (int k = 0; k < numVertices; k++) {
            for (int i = 0; i < numVertices; i++) {
                for (int j = 0; j < numVertices; j++) {
                    if (dist[i][k] != INT_MAX && dist[k][j] != INT_MAX && dist[i][k] + dist[k][j] <
dist[i][j]) {
                        dist[i][j] = dist[i][k] + dist[k][j];
                    }
                }
            }
        }
        printSolution(dist);
    }
    void printSolution(std::vector<std::vector<int>> dist) {
        cout << "The following matrix shows the shortest distances between every pair of vertices:\n";
        for (int i = 0; i < numVertices; i++) {
            for (int j = 0; j < numVertices; j++) {
                if (dist[i][j] == INT_MAX) {
                    cout << "INF ";
                } else {
                    cout << dist[i][j] << "\t";
                }
            }
            cout << endl;
        }
    }
};
int main() {
    Graph graph(6);
    graph.addEdge('a', 'b', 3);
    graph.addEdge('a', 'c', 9);
    graph.addEdge('b', 'c', 8);
    graph.addEdge('b', 'd', 9);
    graph.addEdge('b', 'e', 11);
    graph.addEdge('c', 'd', 1);
    graph.addEdge('c', 'f', 2);
    graph.addEdge('d', 'e', 6);
    graph.addEdge('d', 'f', 9);
    graph.addEdge('e', 'f', 7);
    graph.floydWarshall();
    return 0;
}
```

output:



```
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$ g++ EP4.cpp -o out && ./out
The following matrix shows the shortest distances between every pair of vertices:
6       3       9       10      14      11
3       6       8       9       11      10
9       8       2       1       7       2
10      9       1       2       6       3
14      11      7       6       12      7
11      10      2       3       7       4
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$
```

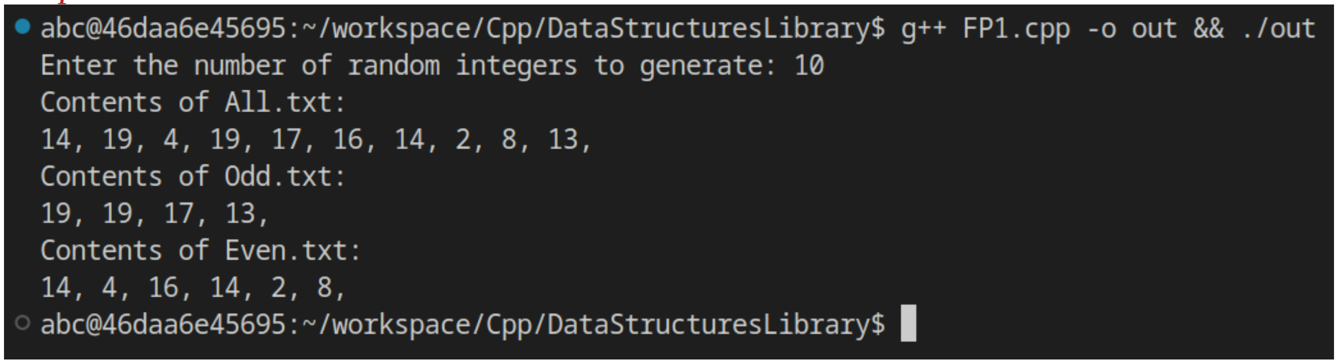
FP1. Write a program that generates n random integers and stores them in a text file, named as “All.txt”. Then, retrieve the stored integers from this file and copy to “Odd.txt” and „Even.txt‟ based upon the type of number, i.e. if the retrieved integer is odd number then store in “Odd.txt” file or if the retrieved integer is even then store in “Even.txt” file. Finally, display the contents of all three files.

Code:

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>

void generateRandomIntegers(int n) {
    std::ofstream file("All.txt");
    if (!file) {
        std::cout << "Error opening file.";
        return;
    }
    std::srand(std::time(0));
    for (int i = 0; i < n; i++) {
        int randomNumber = std::rand();
        file << randomNumber << "\n";
    }
    file.close();
}

void separateNumbers() {
    std::ifstream inputFile("All.txt");
    std::ofstream oddFile("Odd.txt");
    std::ofstream evenFile("Even.txt");

    if (!inputFile) {
        std::cout << "Error opening input file.";
        return;
    }

    if (!oddFile || !evenFile) {
        std::cout << "Error opening output files.";
        return;
    }

    int number;

    while (inputFile >> number) {
        if (number % 2 == 0) {
            evenFile << number << "\n";
        } else {
            oddFile << number << "\n";
        }
    }
```

```cpp
    inputFile.close();
    oddFile.close();
    evenFile.close();
}

void displayFileContents(const std::string& filename) {
    std::ifstream file(filename);
    std::string line;
    if (!file) {
        std::cout << "Error opening file: " << filename;
        return;
    }
    std::cout << "Contents of " << filename << ":\n";
    while (std::getline(file, line)) {
        std::cout << line << " ";
    }
    std::cout << std::endl;
    file.close();
}

int main() {
    int n;
    std::cout << "Enter the number of random integers to generate: ";
    std::cin >> n;
    generateRandomIntegers(n);
    separateNumbers();
    displayFileContents("All.txt");
    displayFileContents("Odd.txt");
    displayFileContents("Even.txt");
    return 0;
}
```

Output:



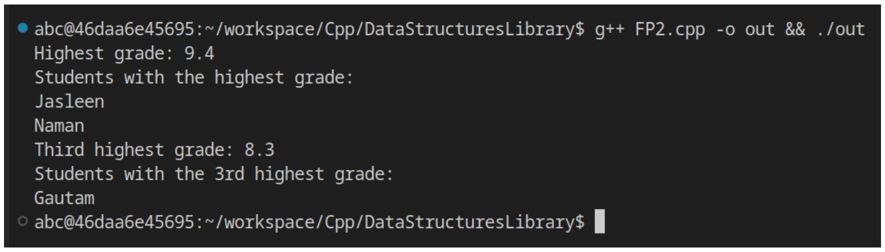
```
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$ g++ FP1.cpp -o out && ./out
Enter the number of random integers to generate: 10
Contents of All.txt:
14, 19, 4, 19, 17, 16, 14, 2, 8, 13,
Contents of Odd.txt:
19, 19, 17, 13,
Contents of Even.txt:
14, 4, 16, 14, 2, 8,
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$
```

FP2. Write a program to find the the highest grade, and list all the students who have highest grade. Also, list the details of students‟ having 3rd highest grade.

Code:

```
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <string>
using namespace std;
bool compareGrades(const pair<float, string>& a, const pair<float, string>& b) {
    return a.first > b.first;
}
int main() {
    ifstream inputFile("grades.txt");
    vector<pair<float, string>> students;
    if (inputFile.is_open()) {
        float grade;
        string name;
        while (inputFile >> grade >> name)
            students.push_back(make_pair(grade, name));
        inputFile.close();
    } else {
        cout << "Unable to open file." << endl;
        return 1;
    }
    sort(students.begin(), students.end(), compareGrades);
    float highestGrade = students[0].first;
    float thirdHighestGrade = students[2].first;
    cout << "Highest grade: " << highestGrade << endl;
    cout << "Students with the highest grade:" << endl;
    for (const auto& student : students)
        if (student.first == highestGrade)
            cout << student.second << endl;
    cout << "Third highest grade: " << thirdHighestGrade << endl;
    cout << "Students with the 3rd highest grade:" << endl;
    for (const auto& student : students)
        if (student.first == thirdHighestGrade)
            cout << student.second << endl;
    return 0;
}
```

Output:



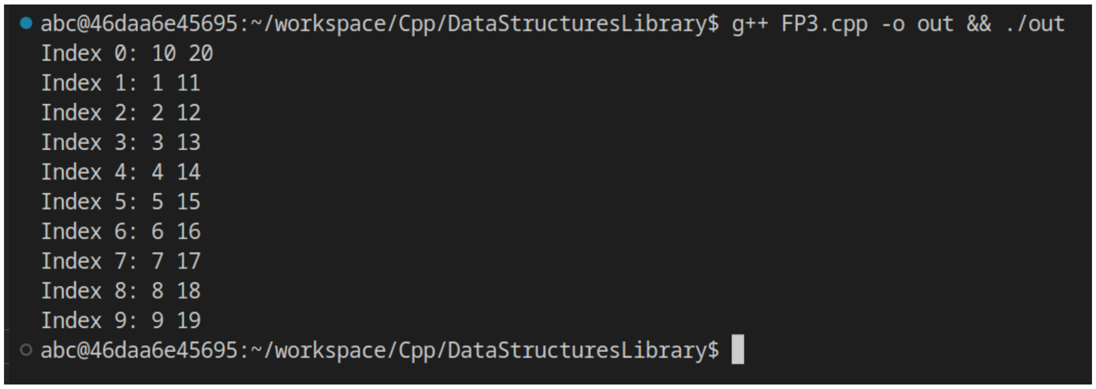
```
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$ g++ FP2.cpp -o out && ./out
Highest grade: 9.4
Students with the highest grade:
Jasleen
Naman
Third highest grade: 8.3
Students with the 3rd highest grade:
Gautam
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$
```

FP3. Write a program to find the the highest grade, and list all the students who have highest grade. Also, list the details of students‟ having 3rd highest grade.

Code:

```
#include <iostream>
#include <list>
using namespace std;
class HashTable {
private:
    static const int TABLE_SIZE = 10;  // Size of the hash table
    list<int> table[TABLE_SIZE];        // Array of linked lists for separate chaining
    int hashFunction(int key) {
        return key % TABLE_SIZE;
    }
public:
    void insert(int key) {
        int index = hashFunction(key);
        table[index].push_back(key);
    }
    void display() {
        for (int i = 0; i < TABLE_SIZE; i++) {
            cout << "Index " << i << ": ";
            for (int num : table[i])
                cout << num << " ";
            cout << endl;
        }
    }
};
int main() {
    HashTable hashTable;
    int integers[20] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20};
    for (int i = 0; i < 20; i++)
        hashTable.insert(integers[i]);
    hashTable.display();
    return 0;
}
```

Output:



```
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$ g++ FP3.cpp -o out && ./out
Index 0: 10 20
Index 1: 1 11
Index 2: 2 12
Index 3: 3 13
Index 4: 4 14
Index 5: 5 15
Index 6: 6 16
Index 7: 7 17
Index 8: 8 18
Index 9: 9 19
abc@46daa6e45695:~/workspace/Cpp/DataStructuresLibrary$
```