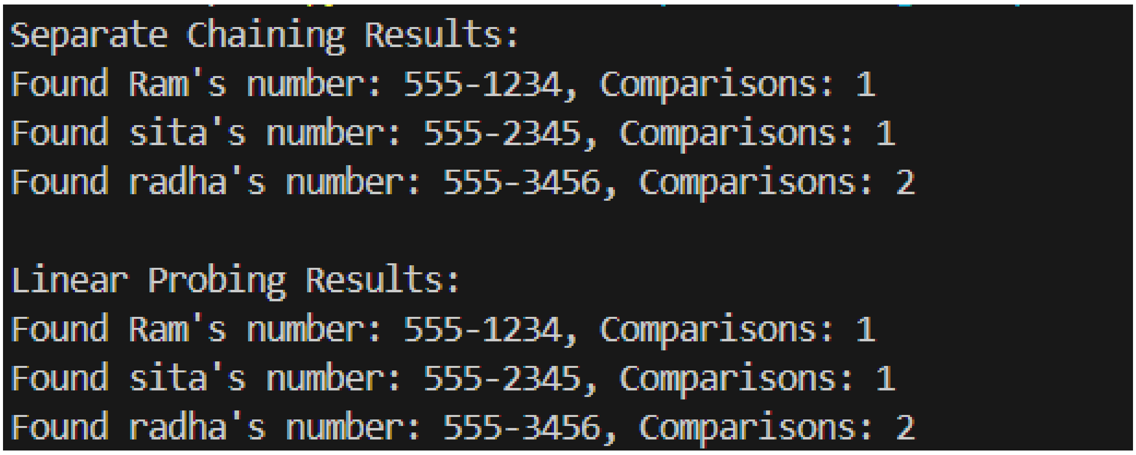
Group A

1.Consider telephone book database of N clients. Make use of a hash table implementation to quickly look up client‘s telephone number. Make use of two collision handling techniques and compare them using number of comparisons required to find a set of telephone numbers

```
class HashTableSeparateChaining:
  def __init__(self, size):
    self.size = size
    self.table = [[] for _ in range(size)]
  def hash(self, key):
    return hash(key) % self.size
  def insert(self, key, value):
    index = self.hash(key)
    self.table[index].append((key, value))
  def find(self, key):
    index = self.hash(key)
    comparisons = 0
    for item in self.table[index]:
      comparisons += 1
      if item[0] == key:
        return item[1], comparisons
    return None, comparisons
# Hash Table Implementation with Linear Probing (Open Addressing)
class HashTableLinearProbing:
  def __init__(self, size):
    self.size = size
    self.table = [None] * size
  def hash(self, key):
    return hash(key) % self.size
    def insert(self, key, value):
        index = self.hash(key)
        while self.table[index] is not None:
            index = (index + 1) % self.size
        self.table[index] = (key, value)
    def find(self, key):
        index = self.hash(key)
        comparisons = 0
        while self.table[index] is not None:
            comparisons += 1
            if self.table[index][0] == key:
                return self.table[index][1], comparisons
            index = (index + 1) % self.size
        return None, comparisons
# Test the two hash table implementations
def test_collision_handling():
    # Initialize two hash tables with the same size
    size = 10
    hash_table_chaining = HashTableSeparateChaining(size)
    hash_table_linear = HashTableLinearProbing(size)
    # Insert some clients into both hash tables
    clients = [
        ("Ram", "555-1234"),
        ("sita", "555-2345"),
        ("radha", "555-3456"),
        ("Kartiki", "555-4567"),
        ("nita", "555-5678"),
    ]

    for name, number in clients:

        hash_table_chaining.insert(name, number)

        hash_table_linear.insert(name, number)


    # Search for telephone numbers and count comparisons

    search_names = ["Ram", "sita", "radha"]


    print("Separate Chaining Results:")

    for name in search_names:

        number, comparisons = hash_table_chaining.find(name)

        print(f"Found {name}'s number: {number}, Comparisons: {comparisons}")


    print("\nLinear Probing Results:")

    for name in search_names:

        number, comparisons = hash_table_linear.find(name)

        print(f"Found {name}'s number: {number}, Comparisons: {comparisons}")


# Run the test

test_collision_handling()
```

output:



```
Separate Chaining Results:
Found Ram's number: 555-1234, Comparisons: 1
Found sita's number: 555-2345, Comparisons: 1
Found radha's number: 555-3456, Comparisons: 2

Linear Probing Results:
Found Ram's number: 555-1234, Comparisons: 1
Found sita's number: 555-2345, Comparisons: 1
Found radha's number: 555-3456, Comparisons: 2
```

Group A

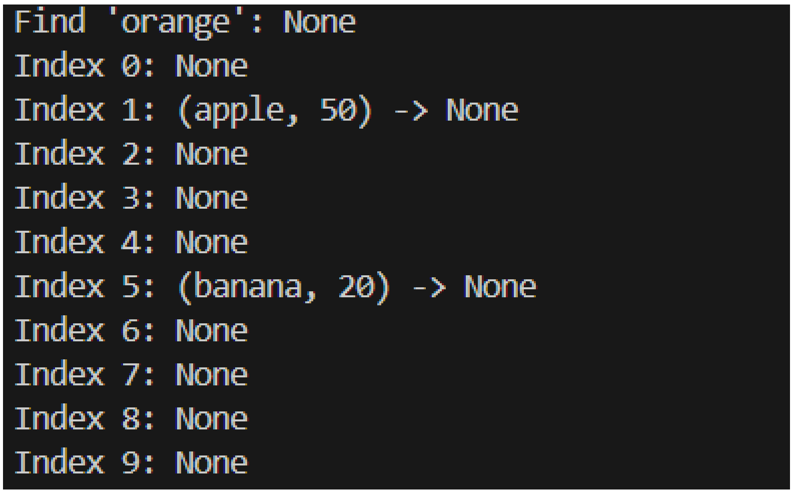
2. Implement functions of ADT using Hashing and handle collision with/without replacement.

```
class HashNode:
    def __init__(self, key, value):
        self.key = key
        self.value = value
        self.next = None

class HashTable:
    def __init__(self, size=10):
        # Initializing the hash table with a default size
        self.size = size
        self.table = [None] * size

    def _hash(self, key):
        # Hash function to calculate index based on the key
        return hash(key) % self.size

    def insert(self, key, value):
        index = self._hash(key)
        new_node = HashNode(key, value)

        # If no collision, insert the new node at the table index
        if self.table[index] is None:
            self.table[index] = new_node
        else:
```

```
        # Collision: Chain the new node at the index
        current = self.table[index]
        while current:
            if current.key == key:
                # Key already exists, replace the value
                current.value = value
                return
            if current.next is None:
                break
            current = current.next
        # If we reach the end of the chain, add the new node
        current.next = new_node

def find(self, key):
    index = self._hash(key)
    current = self.table[index]

    # Traverse the chain at the index
    while current:
        if current.key == key:
            return current.value
        current = current.next
    return None  # Key not found

def delete(self, key):
    index = self._hash(key)
    current = self.table[index]
    prev = None
```

```
        # Traverse the chain to find the key
        while current:
            if current.key == key:
                if prev is None:
                    # Node to delete is the first node in the chain
                    self.table[index] = current.next
                else:
                    # Bypass the current node to delete it
                    prev.next = current.next
                return True
            prev = current
            current = current.next
        return False  # Key not found

    def display(self):
        for i in range(self.size):
            print(f"Index {i}: ", end="")
            current = self.table[i]
            while current:
                print(f"({current.key}, {current.value})", end=" -> ")
                current = current.next
            print("None")

# Usage Example
if __name__ == "__main__":
    ht = HashTable()
    ht.insert("apple", 10)
    ht.insert("banana", 20)
    ht.insert("grape", 30)
    ht.insert("apple", 50)  # Update the value for "apple"

    ht.display()  # Display the hash table

    print("\nFind 'banana':", ht.find("banana"))  # Find value for key "banana"
    print("Find 'orange':", ht.find("orange"))  # Find value for non-existent key

    ht.delete("grape")  # Delete key "grape"
    ht.display()  # Display the updated hash table
```

Output:-



```
Find 'orange': None
Index 0: None
Index 1: (apple, 50) -> None
Index 2: None
Index 3: None
Index 4: None
Index 5: (banana, 20) -> None
Index 6: None
Index 7: None
Index 8: None
Index 9: None
```

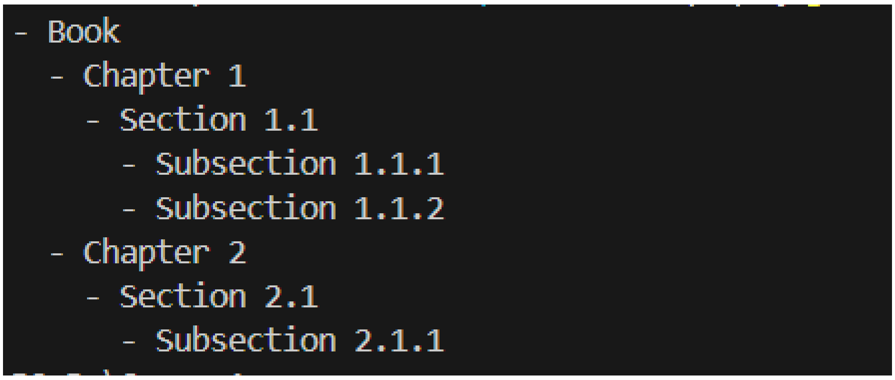
Group B

3. Constructing a Tree Representation for Book Chapters, Sections, and Subsections, with Analysis of Time and Space Complexity.

```
#include <iostream>
#include <vector>
using namespace std;
class TreeNode {
public:
  string name;
  vector<TreeNode*> children;
  TreeNode(string name) {
    this->name = name;
  }
  ~TreeNode() {
    for (TreeNode* child : children) {
      delete child;
    } }
  void addChild(TreeNode* child) {
    children.push_back(child);
  }
  void printTree(int level = 0) {
    for (int i = 0; i < level; i++) cout << "  ";
    cout << "- " << name << endl;
    for (TreeNode* child : children) {
      child->printTree(level + 1);
    }
  } };
```

```
// Constructing the book structure
TreeNode* constructBook() {
  TreeNode* book = new TreeNode("Book");
  TreeNode* chapter1 = new TreeNode("Chapter 1");
  TreeNode* section1_1 = new TreeNode("Section 1.1");
  TreeNode* subsection1_1_1 = new TreeNode("Subsection 1.1.1");
  TreeNode* subsection1_1_2 = new TreeNode("Subsection 1.1.2");
  section1_1->addChild(subsection1_1_1);
  section1_1->addChild(subsection1_1_2);
  chapter1->addChild(section1_1);
  TreeNode* chapter2 = new TreeNode("Chapter 2");
  TreeNode* section2_1 = new TreeNode("Section 2.1");
  TreeNode* subsection2_1_1 = new TreeNode("Subsection 2.1.1");
  section2_1->addChild(subsection2_1_1);
  chapter2->addChild(section2_1);
  book->addChild(chapter1);
  book->addChild(chapter2);
  return book;
}
int main() {
  TreeNode* bookTree = constructBook();
  bookTree->printTree();
  // Clean up dynamically allocated memory
  delete bookTree;
  return 0;
}
```

Output:-



```
- Book
  - Chapter 1
    - Section 1.1
      - Subsection 1.1.1
      - Subsection 1.1.2
  - Chapter 2
    - Section 2.1
      - Subsection 2.1.1
```

Group B

4. Building and Manipulating a Binary Search Tree: Insertion, Longest Path, Minimum Value,Role Swapping, and Search Operations.

```
#include <iostream>

using namespace std;

//Structre to create node

struct Node{

  int data;

  Node* right;

  Node* left;

};

//Function to create node

Node* create(int data){

  Node* temp=new Node();

  temp->data=data;

  temp->left=temp->right=NULL;

  return temp;

}

//Function to insert node

void insert(Node* &root,int data){

  if(root==NULL){

    root=create(data);       //say 5 root

  }                 //Next element n

  else if(root->data > data){    //if (5 > n)

    insert(root->left,data);   //n will go to left side

  }

  else{              //else
```

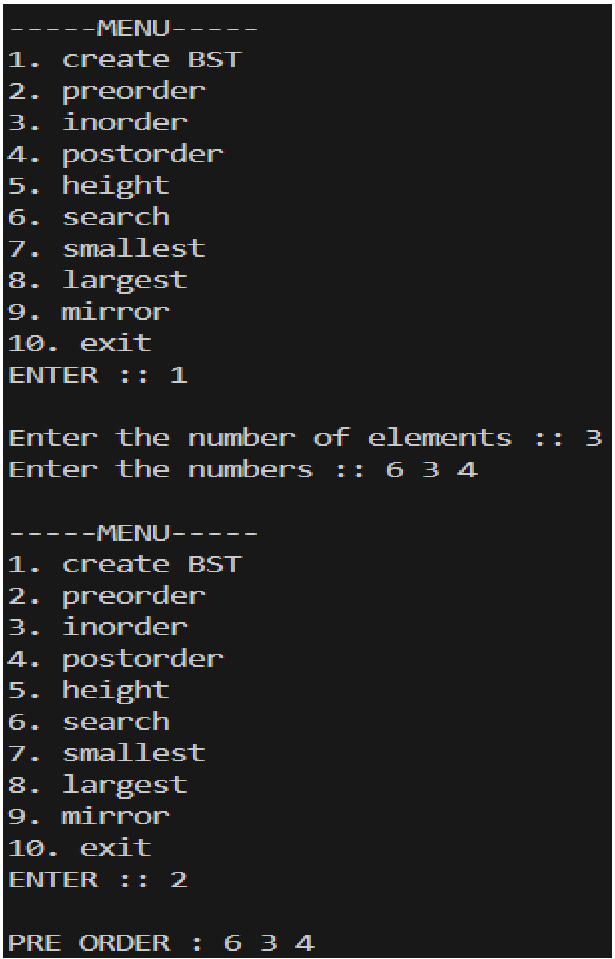
```
    insert(root->right,data);      //n will go to right side
    }
}
//Preorder
void displayPre(Node* root){
    if(root!=NULL){
        cout<<root->data<<" ";     //PARENT
        displayPre(root->left);    //LEFT
        displayPre(root->right);   //RIGHT
    }
}
//Inorder
void displayIn(Node* root){
    if(root!=NULL){
        displayIn(root->left);     //LEFT
        cout<<root->data<<" ";     //PARENT
        displayIn(root->right);    //RIGHT
    }
}
//Postorder
void displayPost(Node* root){
    if(root!=NULL){
        displayPost(root->left);     //LEFT
        displayPost(root->right);    //RIGHT
        cout<<root->data<<" ";       //PARENT
    }
}
//Function to calculate Height
```

```
int height(Node* root){
   if(root==NULL){
      return 0;
   }
   else{
      int l_h=height(root->left);
      int r_h=height(root->right);
      if(l_h>=r_h){
         return l_h+1;
      }
      else{
         return r_h+1;
      }
   }
}
//Function to seach for value
void search(Node* root,int value){
   if(root!=NULL){
   if(root->data>value){
      search(root->left,value);
   }
   else if(root->data<value){
      search(root->right,value);
   }
   else if(root->data==value){
      cout<<"\nelement FOUND";
   }
   }
```

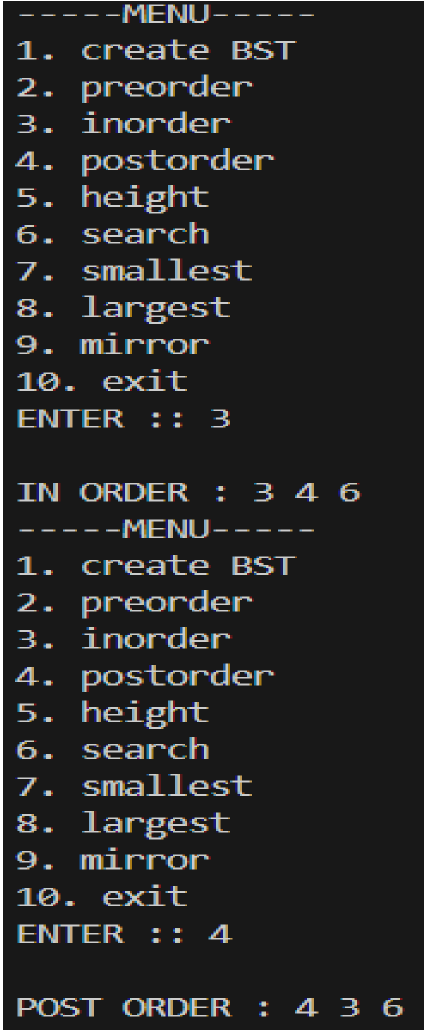
```
  else{
    cout<<"\nelement NOT found";
  }
}
//Function to find smallest element i.e display extreme left
void smallest(Node* root){
  if(root->left!=NULL){
    smallest(root->left);
  }
  else{
    cout<<"Smallest :: "<<root->data;
  }
}
//Function to display largest element i.e display extreme right
void largest(Node* root){
  if(root->right!=NULL){
    largest(root->right);
  }
  else{
    cout<<"\nlargest :: "<<root->data;
  }
}
//Function mirror the tree
void mirror(Node* root){
  if(root==NULL){
    return;
  }
  mirror(root->left);
```

```
    mirror(root->right);
    swap(root->left,root->right);
}
int main(){
    bool loop=1;
    Node * root=NULL;
    int ch,n,num;
    while(loop==1){
    //Menu
    cout<<"\n-----MENU-----"<<endl
        <<"1. create BST"<<endl
        <<"2. preorder"<<endl
        <<"3. inorder"<<endl
        <<"4. postorder"<<endl
        <<"5. height"<<endl
        <<"6. search"<<endl
        <<"7. smallest"<<endl
        <<"8. largest"<<endl
        <<"9. mirror"<<endl
        <<"10. exit"<<endl
        <<"ENTER :: ";
    cin>>ch;
    switch (ch)
    { case 1:
        {
            cout<<"\nEnter the number of elements :: ";
            cin>>n;
            cout<<"Enter the numbers :: ";
            for(int i=0;i<n;i++){
                cin>>num;
                insert(root,num);
            } break;
        }
        case 2: {
            cout<<"\nPRE ORDER : ";
            displayPre(root);
            break;
        }
        case 3: {
            cout<<"\nIN ORDER : ";
            displayIn(root);
            break;
        }
        case 4: {
            cout<<"\nPOST ORDER : ";
            displayPost(root);
            break;
        }
        case 5: {
            int h=height(root);
            cout<<"\nHeight of BST :: "<<h;
            break;
        }
        case 6: {
            int value;
            cout<<"Enter the element to search :: ";
            cin>>value;
            search(root, value);
            break;
          }
          case 7: {
            smallest(root);
            break;
          }
          case 8: {
            largest(root);
            break;
          }
          case 9: {
            cout<<"\nBEFORE MIRROR"
              <<"\nInorder :: ";
            displayIn(root);
            mirror(root);
            cout<<"\nAFTER MIRROR"
              <<"\nInorder :: ";
            displayIn(root);
            break;
          }
          case 10: {
            loop=0;
            break;
          }
        }}
        return 0; }
```

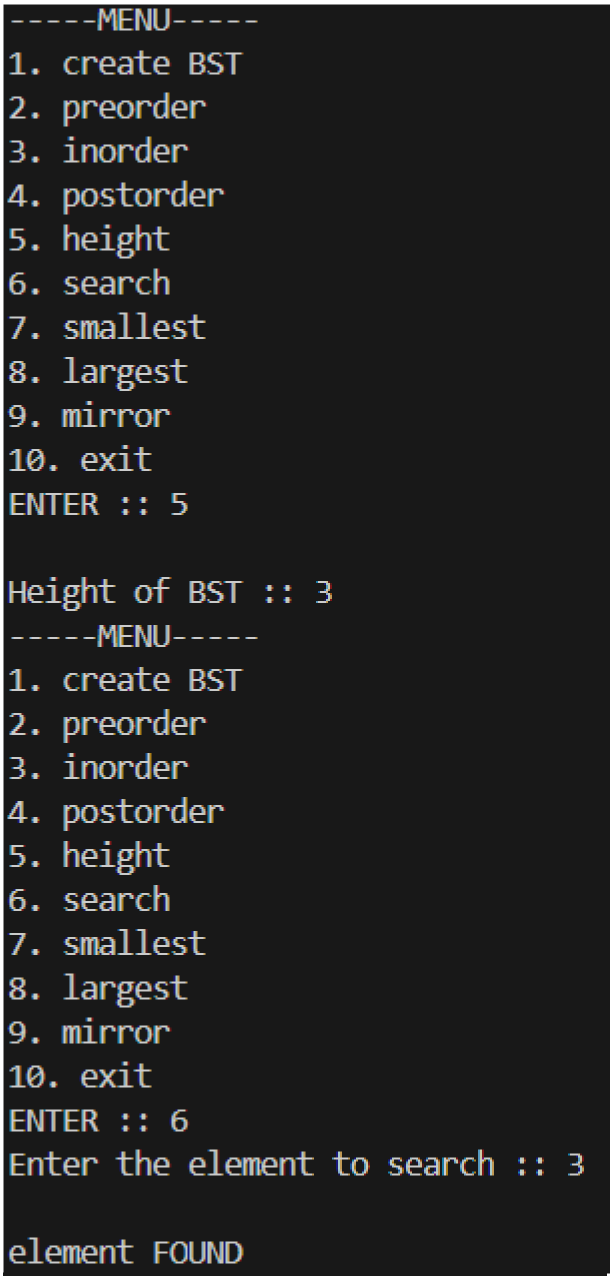
Output:-



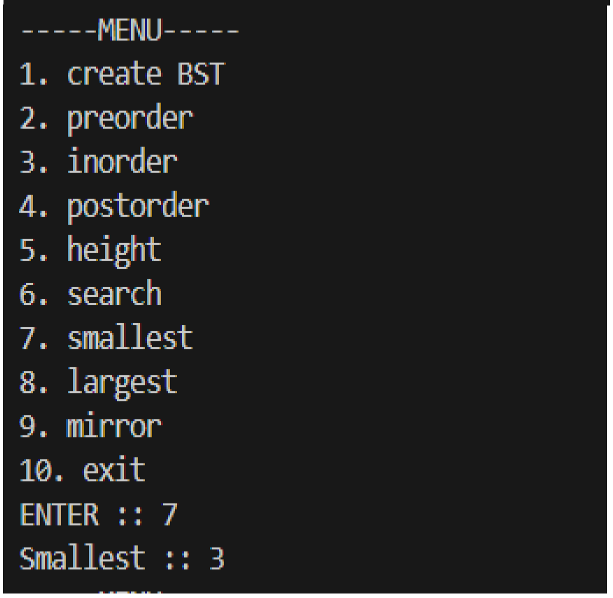
```
-----MENU-----
1. create BST
2. preorder
3. inorder
4. postorder
5. height
6. search
7. smallest
8. largest
9. mirror
10. exit
ENTER :: 1

Enter the number of elements :: 3
Enter the numbers :: 6 3 4

-----MENU-----
1. create BST
2. preorder
3. inorder
4. postorder
5. height
6. search
7. smallest
8. largest
9. mirror
10. exit
ENTER :: 2

PRE ORDER : 6 3 4
```



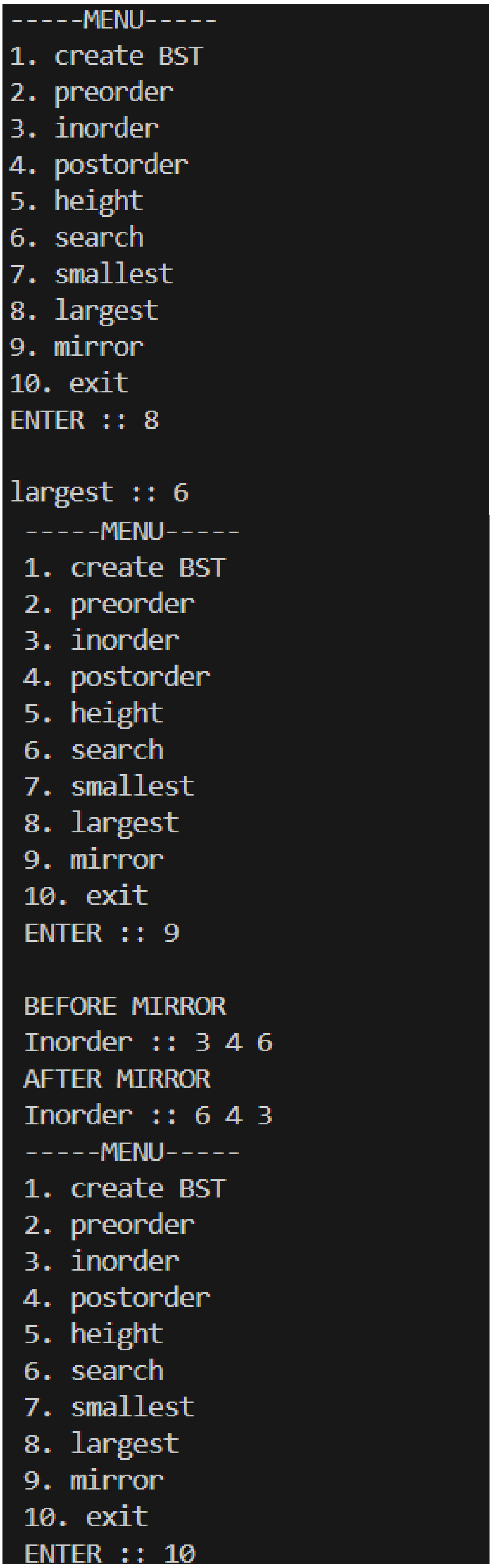
```
-----MENU-----
1. create BST
2. preorder
3. inorder
4. postorder
5. height
6. search
7. smallest
8. largest
9. mirror
10. exit
ENTER :: 3

IN ORDER : 3 4 6
-----MENU-----
1. create BST
2. preorder
3. inorder
4. postorder
5. height
6. search
7. smallest
8. largest
9. mirror
10. exit
ENTER :: 4

POST ORDER : 4 3 6
```



```
-----MENU-----
1. create BST
2. preorder
3. inorder
4. postorder
5. height
6. search
7. smallest
8. largest
9. mirror
10. exit
ENTER :: 5

Height of BST :: 3
-----MENU-----
1. create BST
2. preorder
3. inorder
4. postorder
5. height
6. search
7. smallest
8. largest
9. mirror
10. exit
ENTER :: 6
Enter the element to search :: 3

element FOUND
```



```
-----MENU-----
1. create BST
2. preorder
3. inorder
4. postorder
5. height
6. search
7. smallest
8. largest
9. mirror
10. exit
ENTER :: 7
Smallest :: 3
```



```
-----MENU-----
1. create BST
2. preorder
3. inorder
4. postorder
5. height
6. search
7. smallest
8. largest
9. mirror
10. exit
ENTER :: 8

largest :: 6
 -----MENU-----
 1. create BST
 2. preorder
 3. inorder
 4. postorder
 5. height
 6. search
 7. smallest
 8. largest
 9. mirror
 10. exit
 ENTER :: 9

 BEFORE MIRROR
 Inorder :: 3 4 6
 AFTER MIRROR
 Inorder :: 6 4 3
 -----MENU-----
 1. create BST
 2. preorder
 3. inorder
 4. postorder
 5. height
 6. search
 7. smallest
 8. largest
 9. mirror
 10. exit
 ENTER :: 10
```

Group B

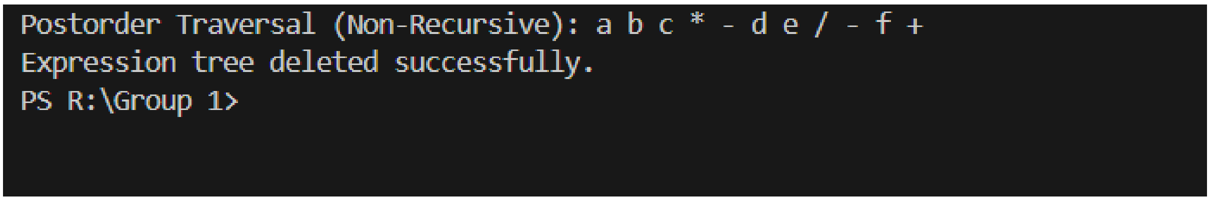
5. Construct an expression tree from given prefix expression and traverse it using post order (non recursive) traversal then delete entire tree

```
#include <iostream>
#include <stack>
#include <cctype>

using namespace std;

// Tree node
struct Node {
    char data;
    Node* left;
    Node* right;

    Node(char val) : data(val), left(nullptr), right(nullptr) {}
};

// Check if the character is an operator
bool isOperator(char ch) {
    return ch == '+' || ch == '-' || ch == '*' || ch == '/';
}

// Construct expression tree from prefix expression
Node* constructTree(const string& prefix) {
    stack<Node*> s;
    // Traverse the prefix expression from right to left
    for (int i = prefix.size() - 1; i >= 0; --i) {
        char ch = prefix[i];

        Node* node = new Node(ch);

        if (isOperator(ch)) {
            // Operator must have two operands
            if (s.size() < 2) {
                cerr << "Invalid expression!\n";
                return nullptr;
            }
            node->left = s.top(); s.pop();
            node->right = s.top(); s.pop();
        }

        s.push(node);
    }

    if (s.size() != 1) {
        cerr << "Invalid expression!\n";
        return nullptr;
    }

    return s.top();
}

// Non-recursive postorder traversal using two stacks
```

```
void postorderNonRecursive(Node* root) {
   if (!root) return;

   stack<Node*> s1, s2;
   s1.push(root);

   while (!s1.empty()) {
      Node* node = s1.top(); s1.pop();
      s2.push(node);

      if (node->left)
         s1.push(node->left);
      if (node->right)
         s1.push(node->right);
   }

   cout << "Postorder Traversal (Non-Recursive): ";
   while (!s2.empty()) {
      cout << s2.top()->data << " ";
      s2.pop();
   }
   cout << endl;
}

// Delete tree (recursive method is okay here)
void deleteTree(Node* root) {
   if (!root) return;
   deleteTree(root->left);
```

```
    deleteTree(root->right);

    delete root;

}


// Main function

int main() {

    string prefix = "+--a*bc/def";

    Node* root = constructTree(prefix);


    if (root) {

        postorderNonRecursive(root);

        deleteTree(root);

        cout << "Expression tree deleted successfully.\n";

    }


    return 0;

}
```

Output:-



```
Postorder Traversal (Non-Recursive): a b c * - d e / - f +
Expression tree deleted successfully.
PS R:\Group 1>
```

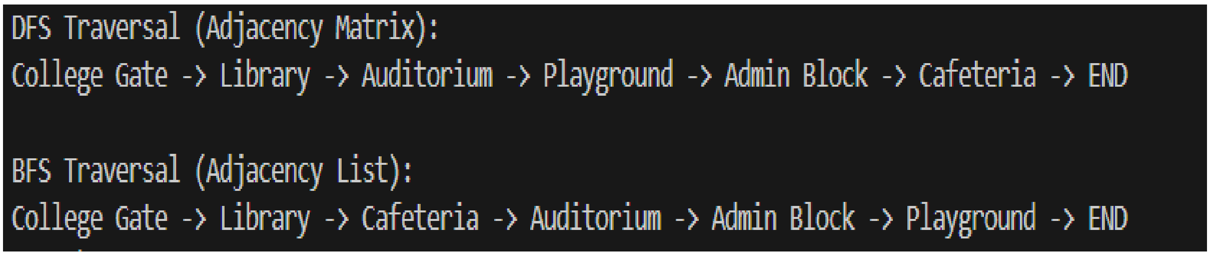
Group C

6. Represent graph using adjacency matrix/list to perform DFS and using adjacency list to perform BFS .Use map of area around the college as graph .identify the prominent land marks as nodes and perform DFS BFS on that.

```
#include <iostream>

#include <vector>

#include <queue>

using namespace std;

#define NUM_NODES 6

// Landmarks

string landmarks[NUM_NODES] = {

  "College Gate", "Library", "Cafeteria",

  "Auditorium", "Admin Block", "Playground"

};


// ---------------- DFS using Adjacency Matrix ----------------

class DFSGraph {

private:

  int adjMatrix[NUM_NODES][NUM_NODES];

  bool visited[NUM_NODES];


public:

  DFSGraph() {

    for (int i = 0; i < NUM_NODES; i++) {

      visited[i] = false;

      for (int j = 0; j < NUM_NODES; j++) {

        adjMatrix[i][j] = 0;

      }
    }
  }

  void addEdge(int u, int v) {
    adjMatrix[u][v] = 1;
    adjMatrix[v][u] = 1;
  }
  void DFS(int node) {
    visited[node] = true;
    cout << landmarks[node] << " -> ";
    for (int i = 0; i < NUM_NODES; i++) {
      if (adjMatrix[node][i] == 1 && !visited[i]) {
        DFS(i);
      }
    }
  }
};
// ---------------- BFS using Adjacency List ----------------
class BFSGraph {
private:
  vector<int> adjList[NUM_NODES];
public:
  void addEdge(int u, int v) {
    adjList[u].push_back(v);
    adjList[v].push_back(u);
  }
  void BFS(int start) {
    vector<bool> visited(NUM_NODES, false);
```

```
        queue<int> q;
        visited[start] = true;
        q.push(start);
        while (!q.empty()) {
            int node = q.front();
            q.pop();
            cout << landmarks[node] << " -> ";
            for (int neighbor : adjList[node]) {
                if (!visited[neighbor]) {
                    visited[neighbor] = true;
                    q.push(neighbor);
                }
            }
        }
    }
};
// ---------------- Main ----------------
int main() {
    // Creating and populating DFS graph
    DFSGraph dfsGraph;
    dfsGraph.addEdge(0, 1); // College Gate - Library
    dfsGraph.addEdge(0, 2); // College Gate - Cafeteria
    dfsGraph.addEdge(1, 3); // Library - Auditorium
    dfsGraph.addEdge(2, 4); // Cafeteria - Admin Block
    dfsGraph.addEdge(3, 5); // Auditorium - Playground
    dfsGraph.addEdge(4, 5); // Admin Block - Playground
    cout << "DFS Traversal (Adjacency Matrix):\n";
    dfsGraph.DFS(0);
    cout << "END\n\n";

    // Creating and populating BFS graph

    BFSGraph bfsGraph;

    bfsGraph.addEdge(0, 1);

    bfsGraph.addEdge(0, 2);

    bfsGraph.addEdge(1, 3);

    bfsGraph.addEdge(2, 4);

    bfsGraph.addEdge(3, 5);

    bfsGraph.addEdge(4, 5);

    cout << "BFS Traversal (Adjacency List):\n";

    bfsGraph.BFS(0);

    cout << "END\n";

    return 0;

}
```

Output:-



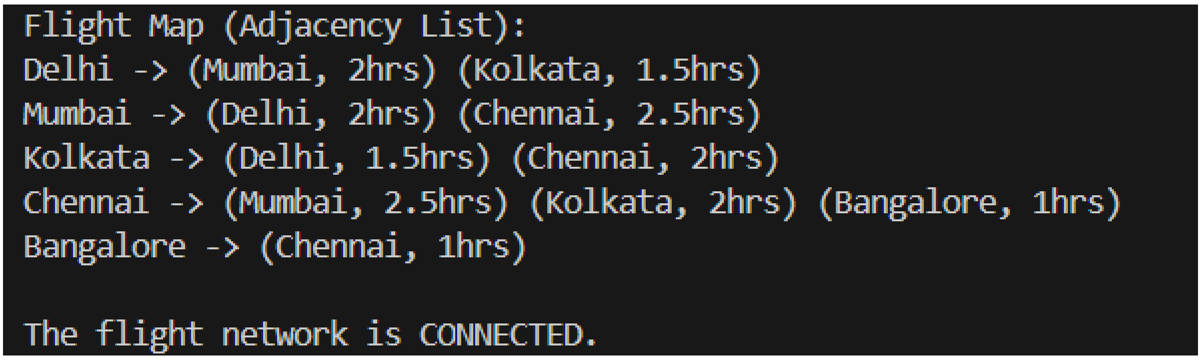
```
DFS Traversal (Adjacency Matrix):
College Gate -> Library -> Auditorium -> Playground -> Admin Block -> Cafeteria -> END

BFS Traversal (Adjacency List):
College Gate -> Library -> Cafeteria -> Auditorium -> Admin Block -> Playground -> END
```

Group C

7. There are flight paths between cities. If there is a flight between city A and city B then there is an edge between the cities. The cost of the edge can be the time that flight take to reach city B from A, or the amount of fuel used for the journey. Represent this as a graph. The node can be represented by airport name or name of the city. Use adjacency list representation of the graph or use adjacency matrix representation of the graph. Check whether the graph is connected or not. Justify the storage representation used.

```
#include <iostream>

#include <vector>

#include <queue>

using namespace std;

#define NUM_CITIES 5

string cityNames[NUM_CITIES] = {

    "Delhi", "Mumbai", "Kolkata", "Chennai", "Bangalore"

};

// Each edge is a pair: (destination, cost)

vector<pair<int, double>> adjList[NUM_CITIES];

// Add a directed or undirected flight

void addFlight(int from, int to, double time) {

    adjList[from].push_back({to, time});

    adjList[to].push_back({from, time}); // Assuming undirected (bidirectional) flights

}

// Check if graph is connected using BFS

bool isConnected() {

    vector<bool> visited(NUM_CITIES, false);

    queue<int> q;

    visited[0] = true;

    q.push(0);
    while (!q.empty()) {
        int city = q.front();
        q.pop();
        for (auto neighbor : adjList[city]) {
            int dest = neighbor.first;
            if (!visited[dest]) {
                visited[dest] = true;
                q.push(dest);
            }
        }
    }
    // Check if all cities are visited
    for (bool v : visited) {
        if (!v) return false;
    }
    return true;
}
void printGraph() {
    cout << "Flight Map (Adjacency List):\n";
    for (int i = 0; i < NUM_CITIES; i++) {
        cout << cityNames[i] << " -> ";
        for (auto edge : adjList[i]) {
            cout << "(" << cityNames[edge.first] << ", " << edge.second << "hrs) ";
        }
        cout << "\n";
    }
}
int main() {
    // Adding flights
    addFlight(0, 1, 2);    // Delhi - Mumbai
    addFlight(0, 2, 1.5);  // Delhi - Kolkata
    addFlight(1, 3, 2.5);  // Mumbai - Chennai
    addFlight(2, 3, 2);    // Kolkata - Chennai
    addFlight(3, 4, 1);    // Chennai - Bangalore
    printGraph();
    // Check connectivity
    if (isConnected()) {
        cout << "\nThe flight network is CONNECTED.\n";
    } else {
        cout << "\nThe flight network is NOT CONNECTED.\n";
    }
    return 0;
}
```

Output:-



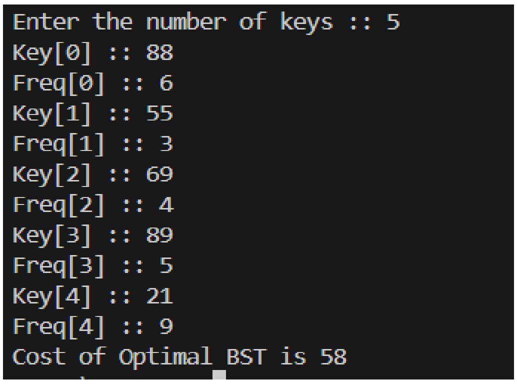
```
Flight Map (Adjacency List):
Delhi -> (Mumbai, 2hrs) (Kolkata, 1.5hrs)
Mumbai -> (Delhi, 2hrs) (Chennai, 2.5hrs)
Kolkata -> (Delhi, 1.5hrs) (Chennai, 2hrs)
Chennai -> (Mumbai, 2.5hrs) (Kolkata, 2hrs) (Bangalore, 1hrs)
Bangalore -> (Chennai, 1hrs)

The flight network is CONNECTED.
```

Group D

8. Given sequence k = k1 <k2 < ... <kn of n sorted keys, with a search probability pi for eachkey ki.Build the Binary search tree that has the least search cost given the access probabilityfor each key?

```
#include <iostream>

#include <limits.h>

using namespace std;

int sum(int freq[], int i, int j)

{

  int s = 0;

  for (int k = i; k <= j; k++)

    s += freq[k];

  return s;

}

int optCost(int keys[], int freq[], int n) {

  int cost[n][n];

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

    cost[i][i] = freq[i];

  for (int length=2; length<=n; length++)

  {

    for (int i=0; i<=n-length+1; i++)

    {

      int j = i+length-1;

      cost[i][j] = INT_MAX;

      for (int r=i; r<=j; r++)

      {

        int c = ((r > i)?cost[i][r-1]:0)+((r < j)?cost[r+1][j]:0)+sum(freq, i, j);

        if (c < cost[i][j])
        cost[i][j] = c;
      }
    }
  }
  return cost[0][n-1];
}
int main()
{
  int n;
  cout<<"Enter the number of keys :: ";
  cin>>n;
  int keys[10],freq[10];
  for(int i=0;i<n;i++){
    cout<<"Key["<<i<<"] :: ";
    cin>>keys[i];
    cout<<"Freq["<<i<<"] :: ";
    cin>>freq[i];
  }
  cout << "Cost of Optimal BST is "
    << optCost(keys, freq, n);
  return 0;
}
```

Output:-



```
Enter the number of keys :: 5
Key[0] :: 88
Freq[0] :: 6
Key[1] :: 55
Freq[1] :: 3
Key[2] :: 69
Freq[2] :: 4
Key[3] :: 89
Freq[3] :: 5
Key[4] :: 21
Freq[4] :: 9
Cost of Optimal BST is 58
```

Group D

9. Implementing a Height-Balanced Tree Dictionary with CRUD Operations, Sorting, and Complexity Analysis for Keyword Retrieval

```
#include<iostream>
#include<string.h>
using namespace std;
struct node
{
char k[20];
char m[20];
class node *left;
class node * right;
};
class dict
{
public:
node *root;
void create();
void disp(node *);
void insert(node * root,node *temp);
int search(node *,char []);
int update(node *,char []);
node* del(node *,char []);
node * min(node *);
};
void dict :: create()
{
```

```
class node *temp;

char ch;

do

{

temp = new node;

cout<<"\nEnter Keyword :: ";

cin>>temp->k;

cout<<"Enter Meaning :: ";

cin>>temp->m;

temp->left = NULL;

temp->right = NULL;

if(root == NULL)

{

root = temp;

}

else

{

insert(root, temp);

}

cout<<"\nDo u want to add more (y/n):";

cin>>ch;

}

while(ch =='y' || ch=='Y');

}

void dict :: insert(node * root,node *temp)

{

if(strcmp (temp->k, root->k) < 0 )

{
```

```
if(root->left == NULL)

root->left = temp;

else

insert(root->left,temp);

}

else

{

if(root->right == NULL)

root->right = temp;

else

insert(root->right,temp);

}

}

void dict:: disp(node * root)

{

if( root != NULL)

{

disp(root->left);

cout<<"\n"<<root->k<<"\t\t"<<root->m;

disp(root->right);

}

}

int dict :: search(node * root,char k[20])

{

int c=0;

while(root != NULL)

{

c++;
```

```
if(strcmp (k,root->k) == 0)

{

cout<<"\nNo of Comparisons ::"<<c;

return 1;

}

if(strcmp (k, root->k) < 0)

root = root->left;

if(strcmp (k, root->k) > 0)

root = root->right;

}

return 0;

}

int dict :: update(node * root,char k[20])

{

while(root != NULL)

{

if(strcmp (k,root->k) == 0)

{

cout<<"\nEnter New Meaning of Keyword "<<root->k<<" :: ";

cin>>root->m;

return 1;

}

if(strcmp (k, root->k) < 0)

root = root->left;

if(strcmp (k, root->k) > 0)

root = root->right;

}return 0;

}
```

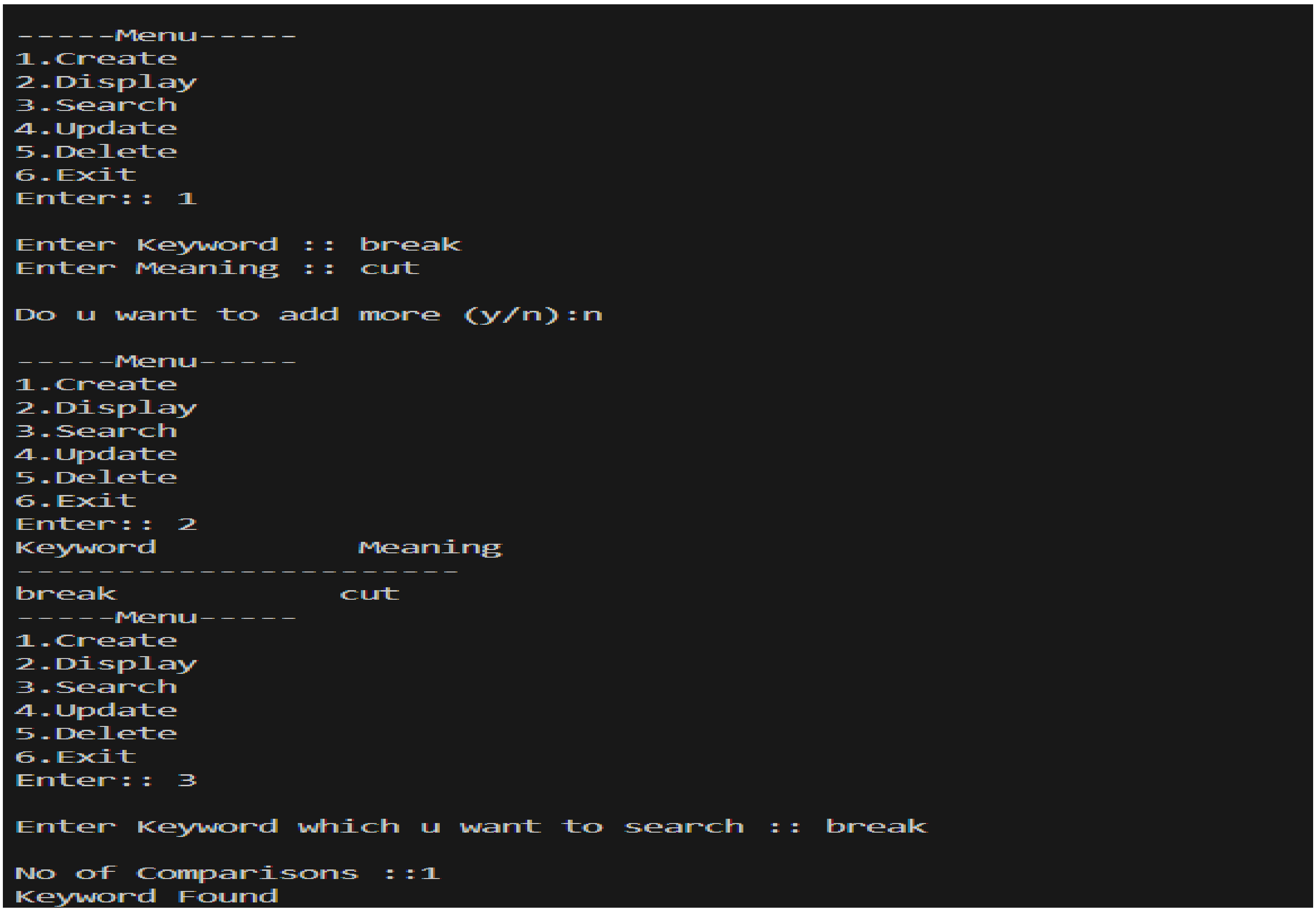
```
node* dict :: del(node * root,char k[20])

{

node *temp;

if(root == NULL)

{

cout<<"\nNo Element Found";

return root;

}

if (strcmp(k,root->k) < 0)

{

root->left = del(root->left, k);

return root;

}

if (strcmp(k,root->k) > 0)

{

root->right = del(root->right, k);

return root;

}

if (root->right==NULL&&root->left==NULL)

{

temp = root;

delete temp;

return NULL;

}

if(root->right==NULL)

{

temp = root;

root = root->left;
delete temp;
return root;
}
else if(root->left==NULL)
{
temp = root;
root = root->right;
delete temp;
return root;
}
temp = min(root->right);
strcpy(root->k,temp->k);
root->right = del(root->right, temp->k);
return root;
}
node * dict :: min(node *q)
{
while(q->left != NULL)
{
q = q->left;
}
return q;
}
int main()
{
int ch,loop=1;
dict d;
d.root = NULL;
```

```
while(loop==1)

{

cout<<"\n-----Menu-----"

<<"\n1.Create\n2.Display\n3.Search\n4.Update\n5.Delete\n6.Exit\nEnter:: ";

cin>>ch;

switch(ch)

{

case 1:

d.create();

break;

case 2:

if(d.root == NULL)

{

cout<<"\nDictionary is Empty";

}

else

{

cout<<"Keyword \t Meaning\n";

cout<<"----------------------";

d.disp(d.root);

}break;

case 3:

if(d.root == NULL)

{

cout<<"\nDictionary is Empty";

}

else

{
```

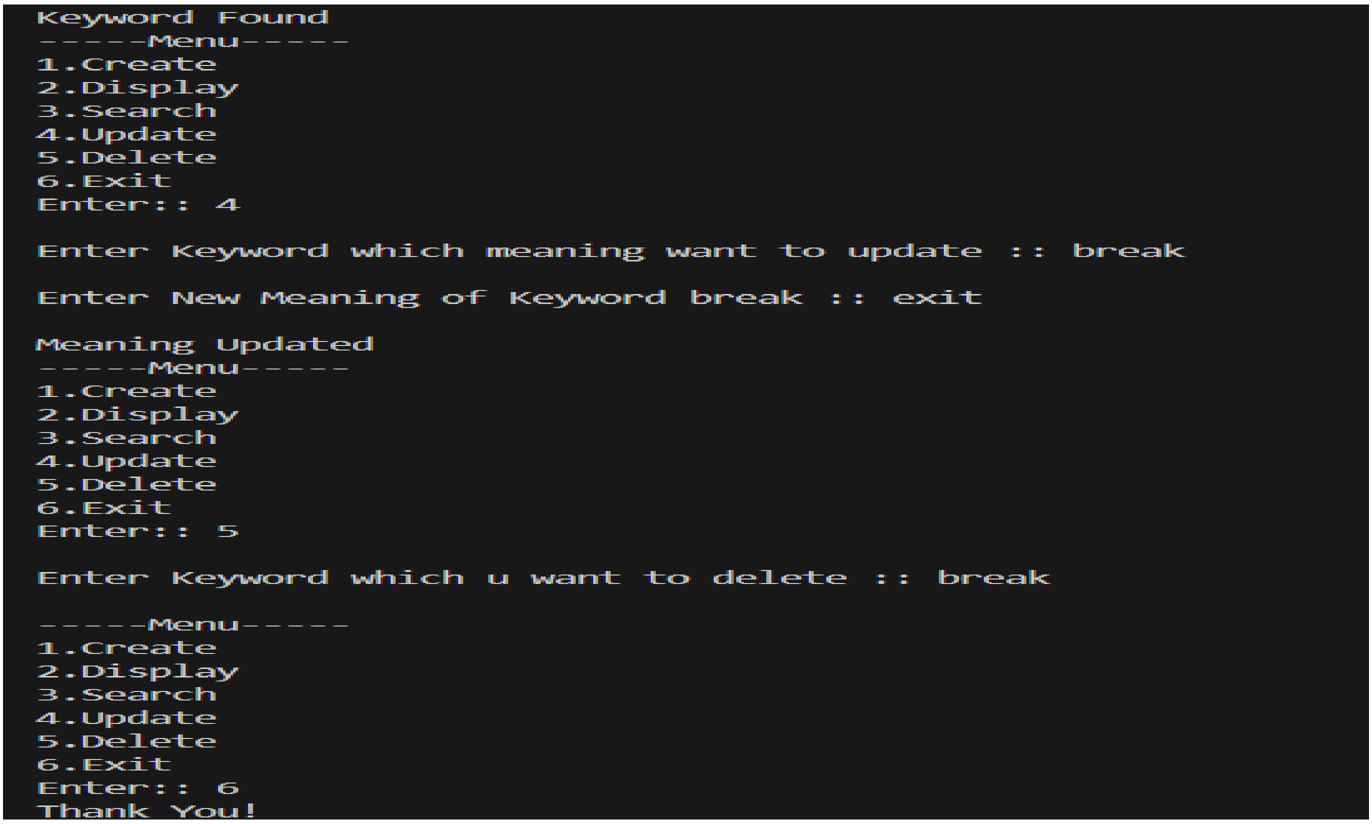
```
cout<<"\nEnter Keyword which u want to search :: ";

char k[20];

cin>>k;

int f=d.search(d.root,k);

if( f == 1)

cout<<"\nKeyword Found";

else

cout<<"\nKeyword Not Found";

}break;

case 4:

if(d.root == NULL)

{

cout<<"\nDictionary is Empty";

}

else

{

cout<<"\nEnter Keyword which meaning want to update :: ";

char k[20];

cin>>k;

if(d.update(d.root,k) == 1)

cout<<"\nMeaning Updated";

else

cout<<"\nKeyword Not Found";

}break;

case 5:

if(d.root == NULL)

{

cout<<"\nDictionary is Empty";
```

```
}
else
{
cout<<"\nEnter Keyword which u want to delete :: ";
char k[20];
cin>>k;
if(d.root == NULL)
{
cout<<"\nKeyword Not Found";
}
else
{
d.root = d.del(d.root,k);
}
}
break;
case 6:
loop=0;
cout<<"Thank You!";
break;
default:
cout<<"You entered something wrong";
break;
}
}
return 0;
}
```

Output:-



```
-----Menu-----
1.Create
2.Display
3.Search
4.Update
5.Delete
6.Exit
Enter:: 1

Enter Keyword :: break
Enter Meaning :: cut

Do u want to add more (y/n):n

-----Menu-----
1.Create
2.Display
3.Search
4.Update
5.Delete
6.Exit
Enter:: 2
Keyword                 Meaning
----------------------
break                 cut
-----Menu-----
1.Create
2.Display
3.Search
4.Update
5.Delete
6.Exit
Enter:: 3

Enter Keyword which u want to search :: break

No of Comparisons ::1
Keyword Found
```



```
Keyword Found
-----Menu-----
1.Create
2.Display
3.Search
4.Update
5.Delete
6.Exit
Enter:: 4

Enter Keyword which meaning want to update :: break

Enter New Meaning of Keyword break :: exit

Meaning Updated
-----Menu-----
1.Create
2.Display
3.Search
4.Update
5.Delete
6.Exit
Enter:: 5

Enter Keyword which u want to delete :: break

-----Menu-----
1.Create
2.Display
3.Search
4.Update
5.Delete
6.Exit
Enter:: 6
Thank You!
```

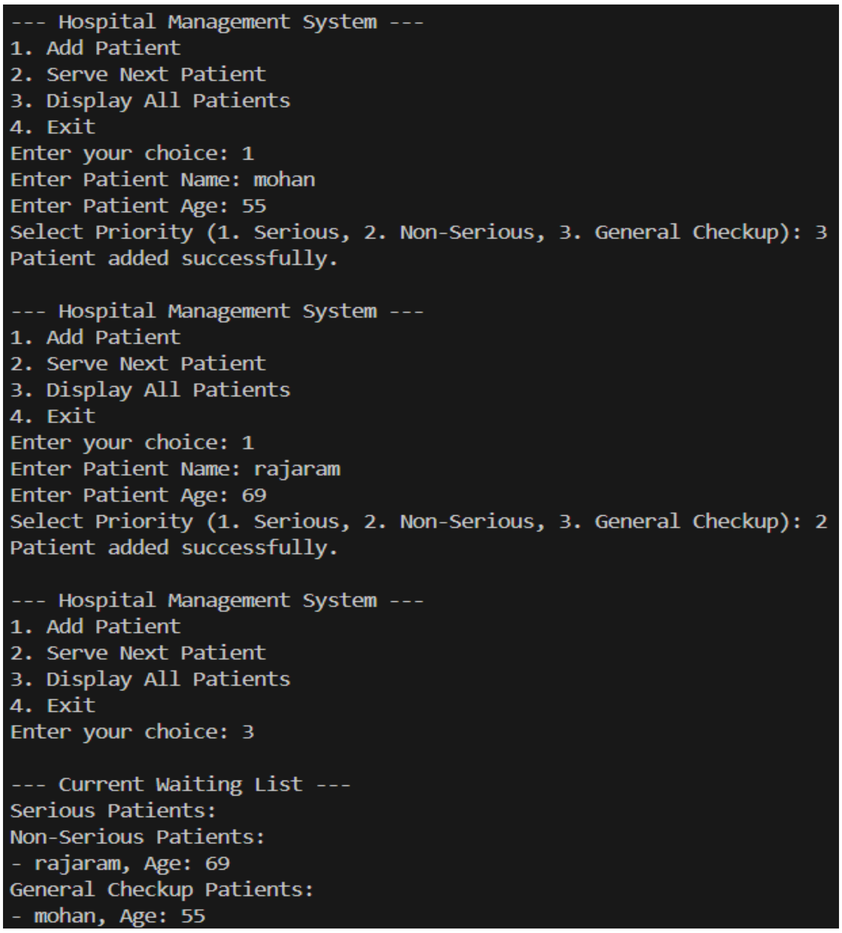
Group E

10. Designing a Priority Queue System for Hospital Services Catering to Various Patient Categories.

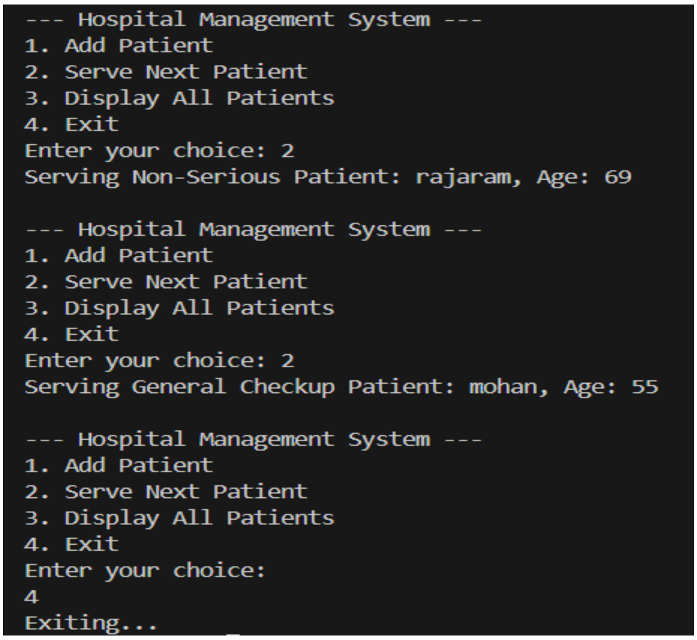
```
#include <iostream>
#include <queue>
#include <string>
using namespace std;
// Define a patient structure
struct Patient {
   string name;
   int age; };
// Priority queues for different types of patients
queue<Patient> seriousQueue;      // Priority 1
queue<Patient> nonSeriousQueue;   // Priority 2
queue<Patient> generalQueue;      // Priority 3
// Add patient to the correct queue
void addPatient() {
   Patient p;
   int priority;
   cout << "Enter Patient Name: ";
   cin.ignore(); // To clear buffer
   getline(cin, p.name);
   cout << "Enter Patient Age: ";
   cin >> p.age;
   cout << "Select Priority (1. Serious, 2. Non-Serious, 3. General Checkup): ";
   cin >> priority;
   switch (priority) {
    case 1: seriousQueue.push(p); break;
    case 2: nonSeriousQueue.push(p); break;
    case 3: generalQueue.push(p); break;
    default: cout << "Invalid priority selected.\n"; return; }
  cout << "Patient added successfully.\n";
}
// Serve next patient based on priority
void servePatient() {
  if (!seriousQueue.empty()) {
    Patient p = seriousQueue.front();
    seriousQueue.pop();
    cout << "Serving Serious Patient: " << p.name << ", Age: " << p.age << endl;
 }
  else if (!nonSeriousQueue.empty()) {
    Patient p = nonSeriousQueue.front();
    nonSeriousQueue.pop();
    cout << "Serving Non-Serious Patient: " << p.name << ", Age: " << p.age << endl;
  }
  else if (!generalQueue.empty()) {
    Patient p = generalQueue.front();
    generalQueue.pop();
    cout << "Serving General Checkup Patient: " << p.name << ", Age: " << p.age << endl }
  else {
    cout << "No patients in queue.\n";
  } }
// Display all waiting patients
void displayQueues() {
  cout << "\n--- Current Waiting List ---\n";
  queue<Patient> temp;
  cout << "Serious Patients:\n";
  temp = seriousQueue;
  while (!temp.empty()) {
    cout << "- " << temp.front().name << ", Age: " << temp.front().age << endl;
    temp.pop();
  }
  cout << "Non-Serious Patients:\n";
  temp = nonSeriousQueue;
  while (!temp.empty()) {
    cout << "- " << temp.front().name << ", Age: " << temp.front().age << endl;
    temp.pop();
  }
  cout << "General Checkup Patients:\n";
  temp = generalQueue;
  while (!temp.empty()) {
    cout << "- " << temp.front().name << ", Age: " << temp.front().age << endl;
    temp.pop();
  } }
// Main menu
int main() {
  int choice;
  do {
    cout << "\n--- Hospital Management System ---\n";
    cout << "1. Add Patient\n";
    cout << "2. Serve Next Patient\n";
    cout << "3. Display All Patients\n";
    cout << "4. Exit\n";
```

```
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice) {
            case 1: addPatient(); break;
            case 2: servePatient(); break;
            case 3: displayQueues(); break;
            case 4: cout << "Exiting...\n"; break;
            default: cout << "Invalid choice. Try again.\n";
        }
    } while (choice != 4);
    return 0;
}
```

Output:-



```
--- Hospital Management System ---
1. Add Patient
2. Serve Next Patient
3. Display All Patients
4. Exit
Enter your choice: 1
Enter Patient Name: mohan
Enter Patient Age: 55
Select Priority (1. Serious, 2. Non-Serious, 3. General Checkup): 3
Patient added successfully.

--- Hospital Management System ---
1. Add Patient
2. Serve Next Patient
3. Display All Patients
4. Exit
Enter your choice: 1
Enter Patient Name: rajaram
Enter Patient Age: 69
Select Priority (1. Serious, 2. Non-Serious, 3. General Checkup): 2
Patient added successfully.

--- Hospital Management System ---
1. Add Patient
2. Serve Next Patient
3. Display All Patients
4. Exit
Enter your choice: 3

--- Current Waiting List ---
Serious Patients:
Non-Serious Patients:
- rajaram, Age: 69
General Checkup Patients:
- mohan, Age: 55
```



```
--- Hospital Management System ---
1. Add Patient
2. Serve Next Patient
3. Display All Patients
4. Exit
Enter your choice: 2
Serving Non-Serious Patient: rajaram, Age: 69

--- Hospital Management System ---
1. Add Patient
2. Serve Next Patient
3. Display All Patients
4. Exit
Enter your choice: 2
Serving General Checkup Patient: mohan, Age: 55

--- Hospital Management System ---
1. Add Patient
2. Serve Next Patient
3. Display All Patients
4. Exit
Enter your choice:
4
Exiting...
```

Group F

11. Creating a Student Information Management System Using Sequential File Operations.

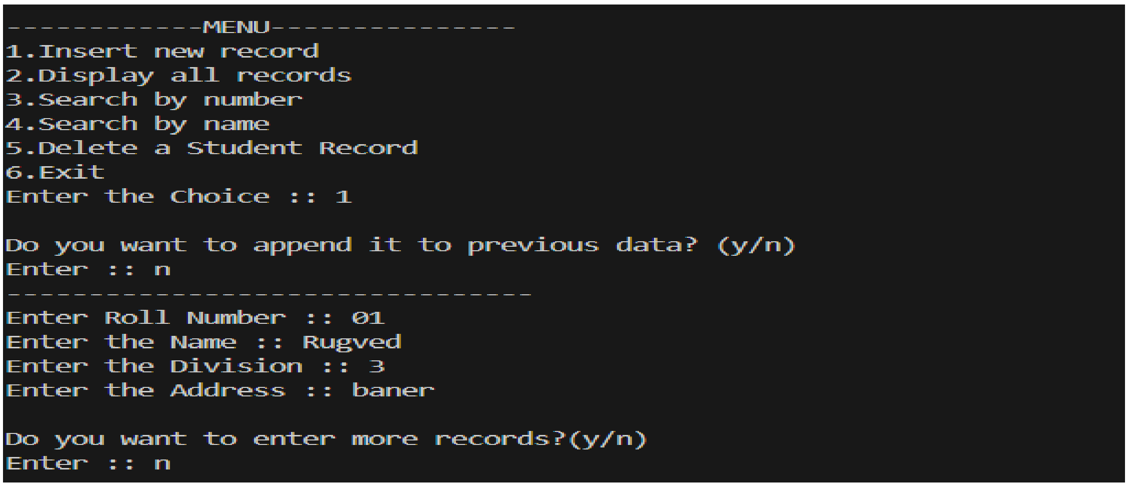
```
#include<iostream>

#include<fstream>

#include<cstring>

using namespace std;

class Student

{

  public:

    int rollNo,roll1;

    char name[10];

    char div;

    char address[20];


    void accept()

    {

      cout<<"------------------------------------";

      cout<<"\nEnter Roll Number :: ";

      cin>>rollNo;

      cout<<"Enter the Name :: ";

      cin>>name;

      cout<<"Enter the Division :: ";

      cin>>div;

      cout<<"Enter the Address :: ";

      cin>>address;

    }
    int getRollNo()
    {
     return rollNo;
    }

    void show()
    {
      cout<<"\n\t"<<rollNo<<"\t\t"<<name<<"\t\t"<<div<<"\t\t"<<address;
    }

    void show1()
    {
      cout<<"\nRoll no :: "<<rollNo
        <<"\nName :: "<<name
        <<"\nDivision :: "<<div
        <<"\nAddress :: "<<address;
    }
};

int main()
{
  int ch,rec,count,y,loop=1;
  char c,name[20];
  Student s;
  count=0;
  fstream g,f;
  while(loop==1)
```

```
{
  cout<<"\n\n------------MENU---------------";
  cout<<"\n1.Insert new record"
    <<"\n2.Display all records"
    <<"\n3.Search by number"
    <<"\n4.Search by name"
    <<"\n5.Delete a Student Record"
    <<"\n6.Exit"
    <<"\nEnter the Choice :: ";
  cin>>ch;
  switch(ch)
  {

    case 1:
    cout<<"\nDo you want to append it to previous data? (y/n) \nEnter :: ";
    cin>>c;
    if(c=='y'||c=='Y')
      f.open("StuRecord.txt",ios::app);
    else
      f.open("StuRecord.txt",ios::out);
    x:s.accept();
    f.write((char*) &s,(sizeof(s)));
    cout<<"\nDo you want to enter more records?(y/n)\nEnter :: ";
    cin>>c;
      if(c=='y' || c=='Y')
        goto x;
      else
      {
```

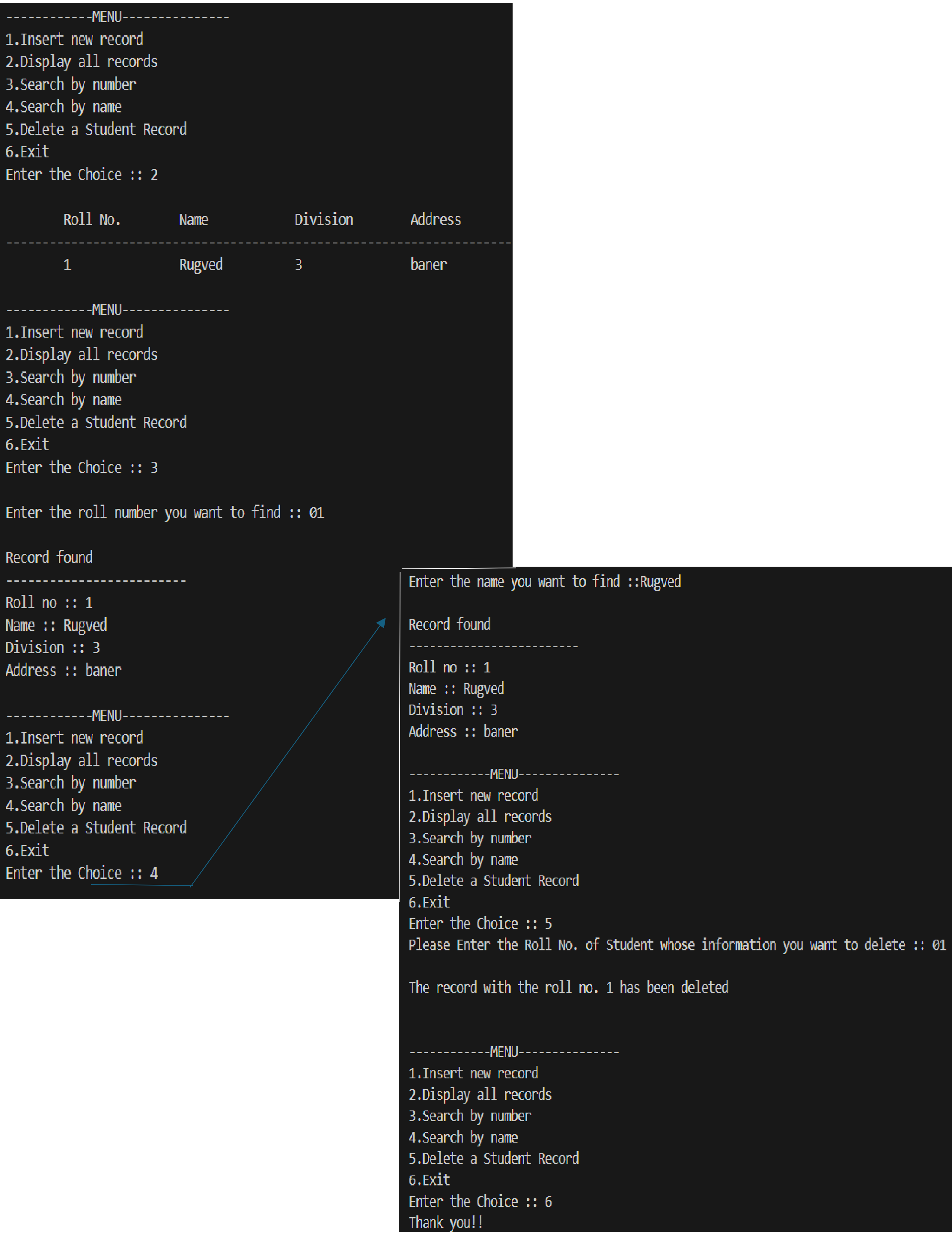
```
      f.close();
      break;
    }
//--------------------------------------------------------------

  case 2:
  f.open("StuRecord.txt",ios::in);
  f.read((char*) &s,(sizeof(s)));
  cout<<"\n\tRoll No.\tName\t\tDivision\tAddress";
  cout<<"\n-------------------------------------------------------------------";
  while(f)
  {
    s.show();
    f.read((char*) &s,(sizeof(s)));
  }
  f.close();
  break;
//---------------------------------------------------------------

  case 3:
  count=0;
    cout<<"\nEnter the roll number you want to find :: ";
    cin>>rec;
    f.open("StuRecord.txt",ios::in|ios::out);
    f.read((char*)&s,(sizeof(s)));
    while(f)
    {
      if(rec==s.rollNo)
      {
        cout<<"\nRecord found";
        cout<<"\n-------------------------";
        s.show1();
        f.close();
        count=1;
        break;
      }
      f.read((char*)&s,(sizeof(s)));
    }
    if(count==0)
      cout<<"\nRecord not found";
    f.close();
  break;
//--------------------------------------------------------------------


  case 4:
    count=0;
    cout<<"\nEnter the name you want to find ::";
    cin>>name;
    f.open("StuRecord.txt",ios::in|ios::out);
    f.read((char*)&s,(sizeof(s)));
    while(f)
    {
      y=(strcmp(name,s.name));
      if(y==0)
      {
        cout<<"\nRecord found";
      cout<<"\n------------------------";
      count=1;
      s.show1();
      break;
     }
     f.read((char*)&s,(sizeof(s)));
    }
    if(count==0)
     cout<<"\nRecord not found";
    f.close();
   break;
  //----------------------------------------------------------------

   case 5:
    count=1;
    int roll;
    cout<<"Please Enter the Roll No. of Student whose information you want to
delete :: ";
    cin>>roll;
    f.open("StuRecord.txt",ios::in);
    g.open("temp.txt",ios::out);
    f.read((char *)&s,sizeof(s));
     while(!f.eof())
     {
      if (s.getRollNo() != roll)
       g.write((char *)&s,sizeof(s));
       f.read((char *)&s,sizeof(s));
      if(s.getRollNo()==roll)
```

```
            count=0;

        }
      if(count==0){
        cout << "\nThe record with the roll no. " << roll << " has been deleted " << endl;
      }
      else{
        cout << "\nRecord not found" << endl;
      }
      g.close();
      f.close();
      remove("StuRecord.txt");
      rename("temp.txt","StuRecord.txt");
    break;
  //-------------------------------------------------------------------


    case 6:
      loop=0;
      cout<<"Thank you!!";
    break;
  }}}
```

Output:-



```
------------MENU---------------
1.Insert new record
2.Display all records
3.Search by number
4.Search by name
5.Delete a Student Record
6.Exit
Enter the Choice :: 1

Do you want to append it to previous data? (y/n)
Enter :: n
--------------------------------
Enter Roll Number :: 01
Enter the Name :: Rugved
Enter the Division :: 3
Enter the Address :: baner

Do you want to enter more records?(y/n)
Enter :: n
```



```
------------MENU--------------
1.Insert new record
2.Display all records
3.Search by number
4.Search by name
5.Delete a Student Record
6.Exit
Enter the Choice :: 2

        Roll No.        Name            Division        Address
----------------------------------------------------------------------
        1               Rugved          3               baner

------------MENU--------------
1.Insert new record
2.Display all records
3.Search by number
4.Search by name
5.Delete a Student Record
6.Exit
Enter the Choice :: 3

Enter the roll number you want to find :: 01

Record found
------------------------
Roll no :: 1
Name :: Rugved
Division :: 3
Address :: baner

------------MENU--------------
1.Insert new record
2.Display all records
3.Search by number
4.Search by name
5.Delete a Student Record
6.Exit
Enter the Choice :: 4
```

```
Enter the name you want to find ::Rugved

Record found
------------------------
Roll no :: 1
Name :: Rugved
Division :: 3
Address :: baner

------------MENU--------------
1.Insert new record
2.Display all records
3.Search by number
4.Search by name
5.Delete a Student Record
6.Exit
Enter the Choice :: 5
Please Enter the Roll No. of Student whose information you want to delete :: 01

The record with the roll no. 1 has been deleted


------------MENU--------------
1.Insert new record
2.Display all records
3.Search by number
4.Search by name
5.Delete a Student Record
6.Exit
Enter the Choice :: 6
Thank you!!
```

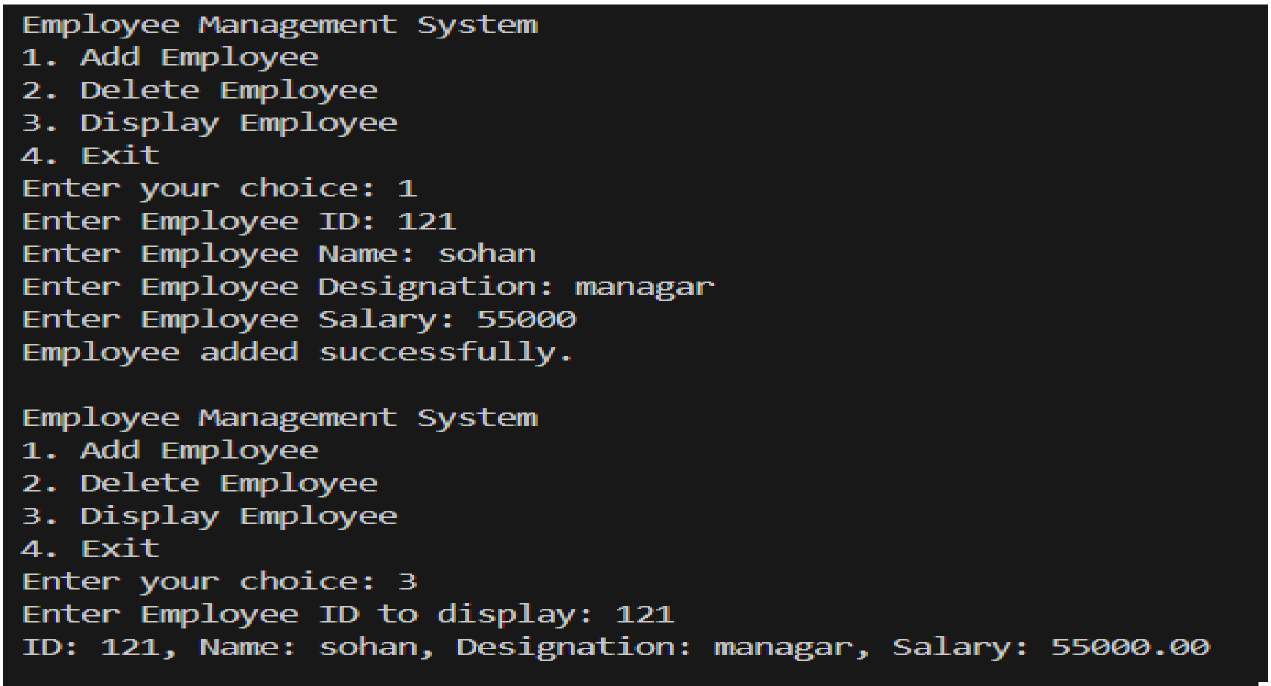
Group F

12. Building an Employee Information Management System with Index Sequential File Operations.

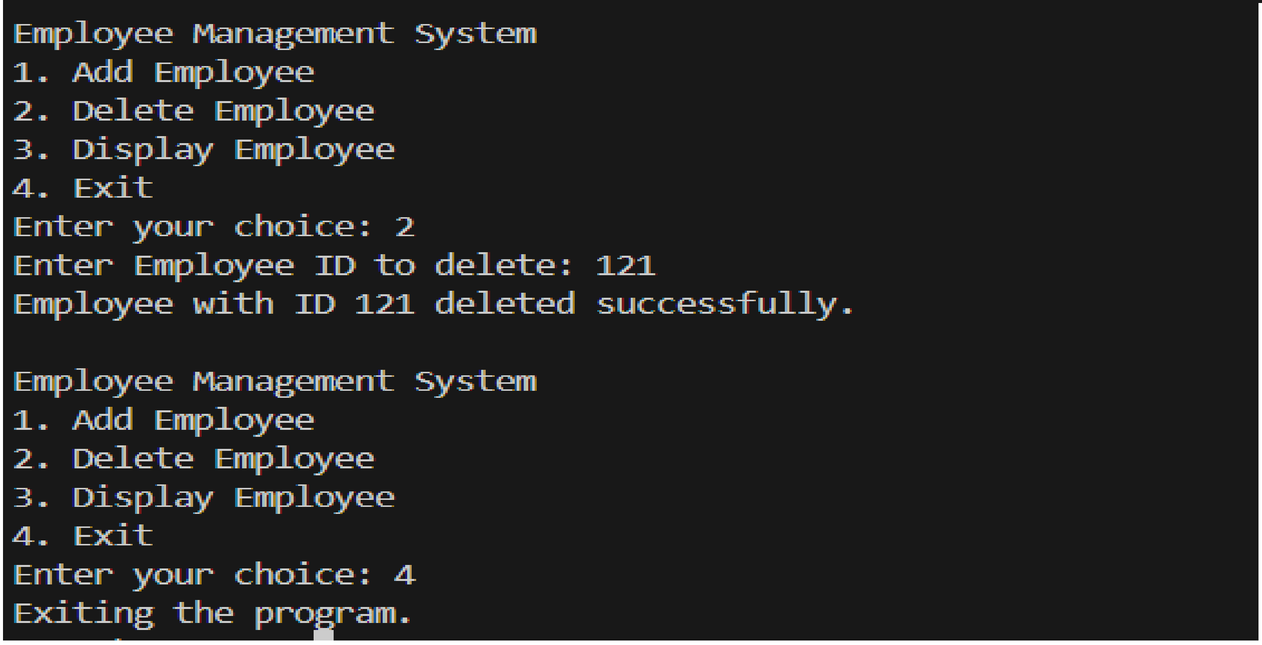
```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <iomanip>
using namespace std;
struct Employee {
  int id;
  string name;
  string designation;
  double salary;
  // Function to display employee details
  void display() const {
    cout << "ID: " << id << ", Name: " << name
       << ", Designation: " << designation
       << ", Salary: " << fixed << setprecision(2) << salary << endl;
  }
};
class EmployeeManager {
private:
  const string filename = "employees.dat";
public:
  void addEmployee(const Employee& emp) {
    ofstream outFile(filename, ios::app | ios::binary);
    if (outFile) {
        outFile.write(reinterpret_cast<const char*>(&emp), sizeof(Employee));
        outFile.close();
        cout << "Employee added successfully." << endl;
    } else {
        cout << "Error opening file." << endl;
    }
}
void deleteEmployee(int id) {
    vector<Employee> employees = loadEmployees();
    bool found = false;
    ofstream outFile(filename, ios::binary);
    for (const auto& emp : employees) {
        if (emp.id != id) {
            outFile.write(reinterpret_cast<const char*>(&emp), sizeof(Employee));
        } else {
            found = true;
        }
    }
    outFile.close();
    if (found) {
        cout << "Employee with ID " << id << " deleted successfully." << endl;
    } else {
        cout << "Employee with ID " << id << " not found." << endl;
    }
}
void displayEmployee(int id) {
    vector<Employee> employees = loadEmployees();
    bool found = false;
    for (const auto& emp : employees) {
      if (emp.id == id) {
        emp.display();
        found = true;
        break;
      }  }
    if (!found) {
      cout << "Employee with ID " << id << " does not exist." << endl;
    } }
private:
  vector<Employee> loadEmployees() {
    vector<Employee> employees;
    Employee emp;
    ifstream inFile(filename, ios::binary);
    while (inFile.read(reinterpret_cast<char*>(&emp), sizeof(Employee))) {
      employees.push_back(emp);
    }
    inFile.close();
    return employees;
  } };
int main() {
  EmployeeManager manager;
  int choice;
  do {
    cout << "\nEmployee Management System\n";
    cout << "1. Add Employee\n";
    cout << "2. Delete Employee\n";
cout << "3. Display Employee\n";

cout << "4. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

   case 1: {

      Employee emp;

      cout << "Enter Employee ID: ";

      cin >> emp.id;

      cout << "Enter Employee Name: ";

      cin.ignore();

      getline(cin, emp.name);

      cout << "Enter Employee Designation: ";

      getline(cin, emp.designation);

      cout << "Enter Employee Salary: ";

      cin >> emp.salary;

      manager.addEmployee(emp);

      break;  }

   case 2: {

      int id;

      cout << "Enter Employee ID to delete: ";

      cin >> id;

      manager.deleteEmployee(id);

      break;  }

   case 3: {

      int id;

      cout << "Enter Employee ID to display: ";

      cin >> id;
```

```
            manager.displayEmployee(id);

            break;}

        case 4:

            cout << "Exiting the program." << endl;

            break;

        default:

            cout << "Invalid choice. Please try again." << endl;  }

    } while (choice != 4);

    return 0;

}
```

Output:-



```
Employee Management System
1. Add Employee
2. Delete Employee
3. Display Employee
4. Exit
Enter your choice: 1
Enter Employee ID: 121
Enter Employee Name: sohan
Enter Employee Designation: managar
Enter Employee Salary: 55000
Employee added successfully.

Employee Management System
1. Add Employee
2. Delete Employee
3. Display Employee
4. Exit
Enter your choice: 3
Enter Employee ID to display: 121
ID: 121, Name: sohan, Designation: managar, Salary: 55000.00
```



```
Employee Management System
1. Add Employee
2. Delete Employee
3. Display Employee
4. Exit
Enter your choice: 2
Enter Employee ID to delete: 121
Employee with ID 121 deleted successfully.

Employee Management System
1. Add Employee
2. Delete Employee
3. Display Employee
4. Exit
Enter your choice: 4
Exiting the program.
```