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

EXPLANATION:
1)#include <iostream>
This line includes the iostream library, which provides input and output functionalities in C++. It allows us t o use objects like cin and cout for input and output operations.

2)using namespace std;
This line is a using directive that allows us to use names from the std namespace without having to prefix them with std::. It simplifies the code by allowing us to write cout instead of std::cout and cin instead of std ::cin.

```
3)class telebook {
  string name;
  long int tel;
public:
  telebook() {
    tel = 0;
  }
  friend class hashing;
};
```
This block of code defines a class named telebook to represent a contact in the telephone book. It has tw o member variables: name of type string, which stores the name of the contact, and tel of type long int, w hich stores the telephone number.
The class also has a default constructor telebook(), which initializes the tel variable to 0.
The friend class hashing declaration indicates that the hashing class can access the private members of t he telebook class.

```
4)class hashing {
  telebook data[100];
  string nm;
  long int t;
  int index, size;
public:
  hashing() {
    t = 0;
    size = 100;
  }
  void create_record() {
    // ...
  }
  void search_record() {
    // ...
  }
  void display_record() {
    // ...
  }
};
```
This block of code defines the hashing class, which handles the operations related to the telephone book.

The class has member variables such as data, which is an array of telebook objects to store the contact r

ecords, nm and t to temporarily store the name and telephone number when creating a record, and index and size for indexing and sizing purposes.
The class also has a default constructor hashing(), which sets t to 0 and size to 100.
It declares three member functions: create_record(), search_record(), and display_record(), which are res ponsible for creating new records, searching for records, and displaying all records in the telephone book, respectively.

5)int main() {
  hashing s;
  int cho = 0;
  while (cho < 4) {
    // ...
  }
  return 0;
}
This is the main() function, which is the entry point of the program.
Inside main(), an object s of the hashing class is created.
A menu-driven loop is implemented using a while loop, which continues until the user enters a value great er than or equal to 4.
The user is prompted with a menu of options to choose from: creating a new client record, displaying all r ecords, searching for a record, or exiting the program.

6)cout << endl << " Enter 1: Create new Client record ";
cout << endl << "Enter 2: Display all record";
cout << endl << "Enter 3: Search ";
cout << endl << "Enter 4: Exit";
cout << endl << "Enter your choice ";
cin >> cho;
These lines of code display the menu options to the user and prompt them to enter their choice. The choic e is stored in the cho variable using cin.

7)switch (cho) {
  case 1:
    cout << "\n1.CREATE Record ";
    s.create_record();
    break;
  case 2:
    cout << "\n\n\n\n2.DISPLAY Record ";
    s.display_record();
    break;
  case 3:
    cout << "\n\n\n\n3.SEARCH Record";
    s.search_record();
    break;
}
This switch statement is used to execute different actions based on the user's choice.
If cho is 1, it prompts the user to create a new record by calling the create_record() function of the hashin g object s.
If cho is 2, it displays all records by calling the display_record() function of s.
If cho is 3, it prompts the user to search for a record by calling the search_record() function of s.

8)while (cho < 4) {
  // ...
}
This while loop ensures that the menu is repeatedly displayed until the user chooses to exit the program (

choosing option 4).

9)return 0;
This line indicates the end of the main() function and returns 0 to indicate successful program execution.

****

GROUP A
2)Implement all the functions of a dictionary (ADT) using hashing and handle collisions using chaining with / without replacement.
Data: Set of (key, value) pairs, Keys are mapped to values, Keys must be comparable, Keys must be unique
Standard Operations: Insert(key, value), Find(key), Delete(key)

EXPLANATION:
1)#include<iostream>
using namespace std;
This code includes the ‘iostream‘ library, which provides input and output functionalities in C++. The ‘using namespace std;‘ line allows us to use names from the ‘std‘ namespace without having to prefix them with ‘std::‘. It simplifies the code by allowing us to write ‘cout‘ instead of ‘std::cout‘ and ‘cin‘ instead of ‘std::cin‘.

2)struct node
{
  string data;
  struct node *next;
};
This block of code defines a structure ‘node‘ to represent a node in a linked list. Each node has a ‘string‘ member variable ‘data‘ to store the data, and a pointer ‘next‘ of type ‘struct node‘ to point to the next node in the list.

3)class HashTable
{
  struct node *tbl[10], *nw, *t;
public:
  HashTable()
  {
    for (int i = 0; i < 10; i++)
      tbl[i] = NULL;
  }
  void insertValue()
  {
    // ...
  }
  void display()
  {
    // ...
  }
  void search()
  {
    // ...

```
  }
};
```
This block of code defines a class named ‘HashTable‘ that represents a hash table.
The class has three private member variables: ‘tbl‘, which is an array of pointers to ‘node‘ objects to repre sent the hash table; ‘nw‘ and ‘t‘, which are pointers to ‘node‘ objects used for inserting and traversing the li nked list, respectively.
The class provides three member functions: ‘insertValue()‘, ‘display()‘, and ‘search()‘, which are responsib le for inserting values into the hash table, displaying the content of the hash table, and searching for a spe cific value in the hash table, respectively.

4)
```
int main()
{
  HashTable h;
  int ch = 0;
  while (ch < 4)
  {
    // ...
  }
  return 0;
}
```
This is the ‘main()‘ function, which is the entry point of the program.
Inside ‘main()‘, an object ‘h‘ of the ‘HashTable‘ class is created.
A menu-driven loop is implemented using a ‘while‘ loop, which continues until the user enters a value gre ater than or equal to 4.
The user is prompted with a menu of options to choose from: inserting a string into the hash table, display ing the content of the hash table, searching for a string in the hash table, or exiting the program.

5)
```
cout << "\n 1: insert String :";
cout << "\n 2: display :";
cout << "\n 3: search :";
cout << "\n 4: Exit";
cout << "\n Enter your choice :";
cin >> ch;
```
These lines of code display the menu options to the user and prompt them to enter their choice. The choic e is stored in the ‘ch‘ variable using ‘cin‘.

6)
```
switch (ch)
{
  case 1:
    h.insertValue();
    break;
  case 2:
    h.display();
    break;
  case 3:
    h.search();
    break;
}
```
This ‘switch‘ statement is used to execute different actions based on the user’s choice.
If ‘ch‘ is 1, it prompts the user to insert a string into the hash table by calling the ‘insertValue()‘ function of t he ‘h‘ object.
If ‘ch‘ is 2, it displays the content of the hash table by calling the ‘display()‘ function of ‘h‘.
If ‘ch‘ is 3, it prompts the user to search for a string in the hash table by calling the ‘search()‘ function of ‘h‘ .

7)while (ch < 4)
{
  // ...
}
This ‘while‘ loop ensures that the menu is repeatedly displayed until the user chooses to exit the program (choosing option 4).

8)return 0;
This line indicates the end of the ‘main()‘ function and returns 0 to indicate successful program execution.

****

GROUP A
4)To create ADT that implement the "set" concept.
a. Add (newElement) -Place a value into the set
b. Remove (element) Remove the value
c. Contains (element) Return true if element is in collection
d. Size () Return number of values in collection Iterator () Return an iterator used to loop over collection
e. Intersection of two sets
f. Union of two sets
g. Difference between two sets
h.Subset

EXPLANATION:
1)#include <iostream>
#include <list>
#include <cstdlib>
using namespace std;
This code includes the necessary libraries: ‘iostream‘ for input and output operations, ‘list‘ for using the list container, and ‘cstdlib‘ for using the ‘exit()‘ function. The ‘using namespace std;‘ line allows us to use names from the ‘std‘ namespace without having to prefix them with ‘std::‘.

2)class set {
private:
 int num, flag = 1;
public:
 list<int> l, l1, u, I, d;
 list<int>::iterator t, t1, t2, t3, t4;
 void createSet();
 void add();
 void delete1(int);
 void search(int);
 void display();
 void union1();
 void Intersection();
 void Difference();
};
This code defines a class named ‘set‘ that represents a set. It has private member variables ‘num‘ and ‘flag‘, and several public member functions for various set operations such as creating a set, adding elements, deleting elements, searching elements, displaying the set, performing union, intersection, and difference operations.

```
3)void set::createSet() {
 int n, m;
 cout << "\nSET A:\n";
 cout << "How many Elements You want Add in Set A:\n";
 cin >> n;
 cout << "Enter Elements\n";

 for (int i = 0; i < n; i++) {
  cin >> num;
  l.push_back(num);
 }
 cout << "\nSET B:\n";
 cout << "How many Elements You want Add in Set B:\n";
 cin >> m;
 cout << "Enter Elements\n";

 for (int i = 0; i < m; i++) {
  cin >> num;
  l1.push_back(num);
 }
}
```

The ‘createSet()‘ function allows the user to create two sets, Set A and Set B. It prompts the user to enter the number of elements for each set, and then the elements themselves. The elements are added to the r espective lists (‘l‘ for Set A and ‘l1‘ for Set B) using the ‘push_back()‘ function.

```
4)void set::add() {
 char c;
 cout << "In Which Set do you want to Add Element (A/B)\n";
 cin >> c;
 cout << "Enter Element\n";
 cin >> num;

 if (c == ’A’ || c == ’a’) {
  l.push_back(num);
  cout << "\nElement Inserted\n";
 }
 else if (c == ’B’ || c == ’b’) {
  l1.push_back(num);
  cout << "\nElement Inserted\n";
 }
 else
  cout << "Invalid Set!!!";
}
```

The ‘add()‘ function allows the user to add an element to either Set A or Set B. It prompts the user to ente r the set in which they want to add the element and the element itself. The element is added to the corres ponding list (‘l‘ for Set A and ‘l1‘ for Set B) using the ‘push_back()‘ function.

```
5)void set::display() {
 cout << "The Elements of Set A:\n{\t";
 for (t = l.begin(); t != l.end(); t++) {
  cout << *t << "\t";
 }
 cout << "}\n\n";
 cout << "The Elements of Set B:\n{\t";
```

```
 for (t1 = l1.begin(); t1 != l1.end(); t1++) {
  cout << *t1 << "\t";
 }
 cout << "}\n";
}
```
The ‘display()‘ function displays the elements of Set A and Set B. It iterates over the list ‘l‘ (Set A) and list ‘ l1‘ (Set B) using iterators ‘t‘ and ‘t1‘, and prints each element using ‘cout‘.

```
6)void set::search(int key) {
 for (t = l.begin(), t1 = l1.begin(); t != l.end(); t++, t1++) {
  if (*t == key || *t1 == key) {
   cout << "The Element is Present\n";
   flag = 1;
   break;
  }
  else
   flag = 0;
 }
 if (flag == 0) {
  cout << "The Element is not Present\n";
 }
}
```
The ‘search()‘ function searches for a given ‘key‘ in both Set A and Set B. It iterates over the lists ‘l‘ and ‘l 1‘ using iterators ‘t‘ and ‘t1‘, and checks if the current element is equal to the ‘key‘. If a match is found, it s ets ‘flag‘ to 1 and displays that the element is present. If no match is found, it sets ‘flag‘ to 0 and displays t hat the element is not present.

```
7)void set::delete1(int key) {
 if (l.empty() && l1.empty()) {
  cout << "Set A & Set B are Empty\n";
 }
 else {
  search(key);
  if (flag == 1) {
   l.remove(key);
   l1.remove(key);
   cout << "Element Deleted\n";
  }
  else
   cout << "Element not Deleted\n";
 }
}
```
The ‘delete1()‘ function deletes a given ‘key‘ from both Set A and Set B. It first checks if both sets are emp ty. If they are not empty, it calls the ‘search()‘ function to check if the element exists in either set. If the ele ment is found (‘flag‘ is 1), it removes the element using the ‘remove()‘ function and displays that the eleme nt is deleted. If the element is not found, it displays that the element is not deleted.

```
8)void set::union1() {
 int flag = 0;
 for (t = l.begin(); t != l.end(); t++) {
  u.push_back(*t);
 }
 for (t1 = l1.begin(); t1 != l1.end(); t1++) {
  for (t2 = u.begin(); t2 != u.end(); t2++) {
   if (*t1 == *t2) {
```

```
    flag = 0;
    break;
   }
   else
    flag = 1;
  }
  if (flag == 1) {
   u.push_back(*t1);
  }
 }

 cout << "The Union Set of A & B is: {\t";
 for (t2 = u.begin(); t2 != u.end(); t2++) {
  cout << *t2 << "\t";
 }
 cout << "}\n";
}
```

The ‘union1()‘ function calculates the union of Set A and Set B. It iterates over the list ‘l‘ (Set A) and adds each element to the list ‘u‘ (union set). Then, it iterates over the list ‘l1‘ (Set B) and checks if each element is already present in the union set (‘u‘). If an element is not found, it adds it to the union set. Finally, it dis plays the elements of the union set.

```
9)void set::Intersection() {
 for (t = l.begin(); t != l.end(); t++) {
  for (

t1 = l1.begin(); t1 != l1.end(); t1++) {
    if (*t == *t1) {
     l.push_back(*t);
     break;
    }
   }
 }
 if (l.empty()) {
  cout << "There is no common element in Set A & Set B\n";
 }
 else {
  cout << "The Intersection Set of A & B is: {\t";
  for (t3 = l.begin(); t3 != l.end(); t3++) {
   cout << *t3 << "\t";
  }
  cout << "}\n";
 }
}
```

The ‘Intersection()‘ function calculates the intersection of Set A and Set B. It iterates over the list ‘l‘ (Set A) and compares each element with the elements in the list ‘l1‘ (Set B). If a match is found, it adds the elem ent to the intersection set (‘I‘). Finally, it displays the elements of the intersection set.

```
10)void set::Difference() {
 int flag = 0;
 for (t = l.begin(); t != l.end(); t++) {
  for (t1 = l1.begin(); t1 != l1.end(); t1++) {
   if (*t == *t1) {
    flag = 0;
    break;
```

```
   }
   else
    flag = 1;
  }
  if (flag == 1) {
   d.push_back(*t);
  }
 }

 if (d.empty()) {
  cout << "Set A & Set B are equal\n";
 }
 else {
  cout << "The Difference Set of A & B is: {\t";
  for (t4 = d.begin(); t4 != d.end(); t4++) {
   cout << *t4 << "\t";
  }
  cout << "}\n";
 }
}
```

The ‘Difference()‘ function calculates the difference between Set A and Set B (A - B). It iterates over the li st ‘l‘ (Set A) and checks if each element is present in the list ‘l1‘ (Set B). If an element is not found, it adds it to the difference set (‘d‘). Finally, it displays the elements of the difference set.

11)

```
int main() {
 set s;
 int ch, key;
 s.createSet();

 while (1) {
  cout << "\n-----------------------------\n";
  cout << "Set Theory\n";
  cout << "\n-----------------------------\n";
  cout << "1. Add Element\n";
  cout << "2. Delete Element\n";
  cout << "3. Search Element\n";
  cout << "4. Display\n";
  cout << "5. Union\n";
  cout << "6. Intersection\n";
  cout << "7. Difference\n";
  cout << "8. Exit\n";
  cout << "Enter Your Choice: ";
  cin >> ch;

  switch (ch) {
   case 1:
    s.add();
    break;
   case 2:
    cout << "Enter the Element to be Deleted: ";
    cin >> key;
    s.delete1(key);
    break;
   case 3:
    cout << "Enter the Element to be Searched: ";
```

```
    cin >> key;
    s.search(key);
    break;
   case 4:
    cout << endl;
    s.display();
    break;
   case 5:
    s.union1();
    break;
   case 6:
    s.Intersection();
    break;
   case 7:
    s.Difference();
    break;
   case 8:
    cout << "Exiting...";
    exit(1);
    break;
   default:
    cout << "Invalid Choice!!!";
    break;
  }
 }
 return 0;
}
```

The ‘main()‘ function acts as the driver program. It creates an instance of the ‘set‘ class (‘s‘) and calls the ‘ createSet()‘ function to create Set A and Set B. It then enters a while loop that displays a menu of set ope rations and prompts the user for their choice. Depending on the user’s choice, it calls the corresponding member function of the ‘set‘ class. The program continues to execute until the user chooses to exit (choic e 8).

**********//////////**********//////////

GROUP B

5)A book consists of chapters, chapters consist of sections and sections consist of subsections. Construct a tree and print the nodes. Find the time and space requirements of your method.

EXPLANATION:

1)
```
#include <iostream>
#include <string.h>
using namespace std;
```

The code includes the necessary header files for input/output (‘iostream‘) and string manipulation (‘string. h‘).

2)
```
struct node
{
    string label;
    int ch_count;
    struct node *child[10];
} * root;
```

A structure ‘node‘ is defined to represent a node in the tree. Each node contains a label (string) and the n

umber of children (‘ch_count‘). It also has an array of pointers to child nodes (‘child‘). The variable ‘root‘ is a pointer to the root node of the tree.

3)class Tree
```
{
public:
    void create_tree();
    void display(node *r1);
    Tree()
    {
        root = NULL;
    }
};
```
A class ‘Tree‘ is defined to encapsulate the tree-related operations. It has two public member functions: ‘create_tree()‘ to create the tree structure, and ‘display()‘ to display the tree hierarchy. The constructor initializes the ‘root‘ pointer to ‘NULL‘.

4)void Tree::create_tree()
```
{
    int tbooks, tchapters, i, j, k;
    root = new node;
    cout << "Enter name of book : ";
    cin >> root->label;
    cout << "Enter number of chapters in book : ";
    cin >> tchapters;
    root->ch_count = tchapters;
    for (i = 0; i < tchapters; i++)
    {
        root->child[i] = new node;
        cout << "Enter the name of Chapter " << i + 1 << " : ";
        cin >> root->child[i]->label;
        cout << "Enter number of sections in Chapter : " << root->child[i]->label << " : ";
        cin >> root->child[i]->ch_count;
        for (j = 0; j < root->child[i]->ch_count; j++)
        {
            root->child[i]->child[j] = new node;
            cout << "Enter Name of Section " << j + 1 << " : ";
            cin >> root->child[i]->child[j]->label;
        }
    }
}
```
The ‘create_tree()‘ function is responsible for creating the tree structure. It prompts the user to enter the name of the book, the number of chapters in the book, and for each chapter, the name of the chapter and the number of sections in that chapter. It dynamically allocates memory for each node and assigns the input values.

5)void Tree::display(node *r1)
```
{
    int i, j, k, tchapters;
    if (r1 != NULL)
    {
        cout << "\n-----Book Hierarchy---";
        cout << "\n Book title : " << r1->label;
        tchapters = r1->ch_count;
        for (i = 0; i < tchapters; i++)
```

```
        {
            cout << "\nChapter " << i + 1;
            cout << " : " << r1->child[i]->label;
            cout << "\nSections : ";
            for (j = 0; j < r1->child[i]->ch_count; j++)
            {
                cout << "\n" << r1->child[i]->child[j]->label;
            }
        }
    }
    cout << endl;
}
```

The ‘display()‘ function is used to display the tree hierarchy. It takes a node pointer ‘r1‘ as a parameter and checks if it is not ‘NULL‘. If it’s not ‘NULL‘, it displays the book title, chapters, and sections. It iterates over the chapters and sections using nested loops and prints their labels.

6)

```
int main()
{
    int choice = 0;
    Tree gt;
    while (choice < 3)
    {
        cout << "1.Create" << endl;
        cout << "2.Display" << endl;
        cout << "3.Quit" << endl;
        cout << "Enter your choice : ";
        cin >> choice;
        switch (choice)
        {
        case 1:
            gt.create_tree();
            break;
        case 2:
            gt.display(root);
            break;
        default:
            cout << "Wrong choice!!!" << endl;
        }
    }
    return 0;
}
```

The ‘main()‘ function acts as the driver program. It creates an instance of the ‘Tree‘ class (‘gt‘) and enters a loop that displays a menu of options: create the tree (choice 1), display the tree (choice 2), or quit (choice 3). Based on the user’s choice, it calls the corresponding member function of the ‘Tree‘ class. The loop continues until the user chooses to quit.

****

GROUP B

6)Beginning with an empty binary search tree, Construct binary search tree by inserting the values in the order given. After constructing a binary tree -

i. Insert new node

ii. Find number of nodes in longest path from root

iii. Minimum data value found in the tree

iv. Change a tree so that the roles of the left and right pointers are swapped at every node

v. Search a value

EXPLANATION:
1)#include<iostream>
#include<math.h>
using namespace std;
The code includes the necessary header files for input/output (‘iostream‘) and mathematical functions (‘m
ath.h‘).

2)struct Bstnode
{
 int data;
 Bstnode *left = NULL;
 Bstnode *right = NULL;
};
A structure ‘Bstnode‘ is defined to represent a node in a binary search tree (BST). Each node contains an integer ‘data‘ and pointers to the left and right child nodes (‘left‘ and ‘right‘).

3)class Btree
{
  int n;
  int x;
  int flag;
public:
  Bstnode *root;
 Btree()
 {
  root = NULL;
 }
A class ‘Btree‘ is defined to encapsulate the BST-related operations. It has a private member ‘n‘ to store t
he number of elements, ‘x‘ to store temporary data during insertion, and ‘flag‘ to store a flag value. The pu
blic member ‘root‘ is a pointer to the root node of the BST. The constructor initializes the ‘root‘ pointer to ‘
NULL‘.

4)Bstnode *GetNewNode(int in_data)
{
  Bstnode *ptr = new Bstnode();
  ptr->data = in_data;
  ptr->left = NULL;
  ptr->right = NULL;
  return ptr;
}
A private member function ‘GetNewNode()‘ is defined to create a new BST node. It takes an integer ‘in_d
ata‘ as input, dynamically allocates memory for a new node, initializes its data and pointers, and returns th
e pointer to the new node.

5)Bstnode *insert(Bstnode *temp, int in_data)
{
  if (temp == NULL)
  {
    temp = GetNewNode(in_data);
  }
  else if (temp->data > in_data)
  {
    temp->left = insert(temp->left, in_data);

```
  }
  else
  {
    temp->right = insert(temp->right, in_data);
  }
  return temp;
}
```
A private member function ‘insert()‘ is defined to insert a node into the BST. It takes a pointer to the curre nt node ‘temp‘ and an integer ‘in_data‘ as input. If ‘temp‘ is ‘NULL‘, it creates a new node using ‘GetNewN ode()‘ and assigns it to ‘temp‘. If ‘in_data‘ is less than the data of ‘temp‘, it recursively inserts the node into the left subtree. Otherwise, it recursively inserts the node into the right subtree. It returns the modified ‘te mp‘ pointer.

```
6)void createBST()
{
  cout << "ENTER NUMBER OF ELEMENTS IN THE BST : ";
  cin >> n;
  for (int i = 0; i < n; i++)
  {
    cout << "NUMBER = ";
    cin >> x;
    root = insert(root, x);
  }
}
```
A public member function ‘createBST()‘ is defined to create the BST. It prompts the user to enter the num ber of elements in the BST (‘n‘) and then prompts for each element. It inserts each element into the BST u sing the ‘insert()‘ function.

```
7)int search(Bstnode *temp, int in_data)
{
  if (temp != NULL)
  {
    if (temp->data == in_data)
    {
      cout << ":-- RECORD FOUND --:" << endl;
      return 1;
    }
    else if (in_data < temp->data)
    {
      this->search(temp->left, in_data);
    }
    else if (in_data > temp->data)
    {
      this->

search(temp->left, in_data);
    }
  }
  else
  {
    return 0;
  }
}
```
A public member function ‘search()‘ is defined to search for a given element in the BST. It takes a pointer to the current node ‘temp‘ and an integer ‘in_data‘ as input. If ‘temp‘ is not ‘NULL‘, it checks if the data of ‘t

emp‘ is equal to ‘in_data‘. If found, it prints a message and returns 1. If ‘in_data‘ is less than the data of ‘te mp‘, it recursively searches in the left subtree. If ‘in_data‘ is greater than the data of ‘temp‘, it recursively s earches in the right subtree. If ‘temp‘ is ‘NULL‘, it returns 0.

```
8)void minvalue(Bstnode *temp)
{
  while (temp->left != NULL)
  {
    temp = temp->left;
  }
  cout << "MINIMUM VALUE = " << temp->data << endl;
}
```

A public member function ‘minvalue()‘ is defined to find the minimum value in the BST. It takes a pointer t o the current node ‘temp‘ as input. It traverses to the leftmost node of the BST until ‘temp->left‘ becomes ‘ NULL‘ and then prints the data of that node as the minimum value.

```
9)void mirror(Bstnode *temp)
{
  if (temp == NULL)
  {
    return;
  }
  else
  {
    Bstnode *ptr;
    mirror(temp->left);
    mirror(temp->right);
    ptr = temp->left;
    temp->left = temp->right;
    temp->right = ptr;
  }
}
```

A public member function ‘mirror()‘ is defined to create a mirror image of the BST. It takes a pointer to the current node ‘temp‘ as input. If ‘temp‘ is ‘NULL‘, it returns. Otherwise, it recursively swaps the left and righ t child pointers of each node in the BST, effectively creating a mirror image of the tree.

```
10)void display()
{
  cout << endl
       << "--- INORDER TRAVERSAL ---" << endl;
  inorder(root);
  cout << endl;
}
```

A public member function ‘display()‘ is defined to display the BST using inorder traversal. It calls the privat e member function ‘inorder()‘ with the ‘root‘ pointer as input.

```
11)void inorder(Bstnode *temp)
{
  if (temp != NULL)
  {
    inorder(temp->left);
    cout << temp->data << "  ";
    inorder(temp->right);
  }
}
```

A private member function ‘inorder()‘ is defined to perform inorder traversal of the BST. It takes a pointer t o the current node ‘temp‘ as input. If ‘temp‘ is not ‘NULL‘, it recursively visits the left subtree, prints the dat a of ‘temp‘, and then recursively visits the right subtree.

12)int depth(Bstnode *temp)
{
  if (temp == NULL)
    return 0;
  return (max((depth(temp->left)), (depth(temp->right))) + 1);
}
A public member function ‘depth()‘ is defined to calculate the depth (maximum height) of the BST. It takes a pointer to the current node ‘temp‘ as input. If ‘temp‘ is ‘NULL‘, it returns 0. Otherwise, it recursively calc ulates the depth of the left and right subtrees and returns the maximum depth plus 1.

13)int main()
{
 Btree obj;
 int cho = 0, a, e;

 while (cho < 4)
 {
   cout << endl<<
 " Enter 1: create BST ";
   cout << endl
        << " Enter 2: Display tree inorder";
   cout << endl
        << " Enter 3: search a number ";
   cout << endl
        << " Enter 4 : Find minimum value ";
   cout << endl
        << " Enter 5 : create a mirror image";
   cout << endl
        << " Enter 6 : calculate depth ";
   cout << endl
        << " Enter your choice ";
   cin >> cho;
   switch (cho)
   {
   case 1:
     obj.createBST();
     break;
   case 2:
     obj.display();
     break;
   case 3:
     a = 0;
     cout << endl
          << " Enter element to search";
     cin >> e;
     a = obj.search(obj.root, e);
     if (a == 0)
     {
       cout << "ELEMENT NOT FOUND" << endl;
     }
     else

```
      cout << "ELEMENT FOUND" << endl;
    cout << endl
         << a << endl;
    break;
  case 4:
    obj.minvalue(obj.root);
    break;
  case 5:
    obj.mirror(obj.root);
    obj.inorder(obj.root);
    break;
  case 6:
    cout << endl
         << obj.depth(obj.root);
    break;
  }
 }

 return 0;
}
```

The ‘main()‘ function is the entry point of the program. It creates an object ‘obj‘ of the ‘Btree‘ class and pre sents a menu-driven interface to the user. The user can choose different operations such as creating a B ST, displaying the tree using inorder traversal, searching for an element, finding the minimum value, creat ing a mirror image of the tree, and calculating the depth of the tree. The program continues to execute unt il the user enters a choice greater than 4.

**********//////////**********//////////

GROUP C

13)Represent a given graph using adjacency matrix/list to perform DFS and using adjacency list to perform BFS. Use the map of the area around the college as the graph. Identify the prominent land marks as nodes and perform DFS and BFS on that.

EXPLANATION:

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

This code includes necessary header files for input/output operations and standard library functions.

2)int cost[10][10], i, j, k, n, qu[10], front, rear, v, visit[10], visited[10];
int stk[10], top, visit1[10], visited1[10];

This line declares various variables used in the program, such as the adjacency matrix ‘cost‘, loop control variables ‘i‘, ‘j‘, ‘k‘, the number of vertices ‘n‘, arrays for queue ‘qu‘ and stack ‘stk‘, variables for queue poi nters ‘front‘ and ‘rear‘, vertex variables ‘v‘, and arrays for tracking visited vertices ‘visit‘ and ‘visited‘.

3)int main()
{
    int m;
    cout << "Enter number of vertices : ";
    cin >> n;
    cout << "Enter number of edges : ";
    cin >> m;

The ‘main()‘ function is the entry point of the program. It prompts the user to enter the number of vertices ‘ n‘ and the number of edges ‘m‘ in the graph.

4)
```
cout << "\nEnter EDGES : i j :\n";
   for (k = 1; k <= m; k++)
   {
      cin >> i >> j;
      cost[i][j] = 1;
      cost[j][i] = 1;
   }
```
This part of the code takes input for the edges of the graph and updates the adjacency matrix ‘cost‘ accor dingly. For each edge, it sets ‘cost[i][j]‘ and ‘cost[j][i]‘ to 1, indicating the presence of an edge between vert ices ‘i‘ and ‘j‘.

5)
```
//display function
   cout << "The adjacency matrix of the graph is : " << endl;
   for (i = 0; i < n; i++)
   {
      for (j = 0; j < n; j++)
      {
         cout << " " << cost[i][j];
      }
      cout << endl;
   }
```
This code snippet displays the adjacency matrix of the graph, showing the connections between vertices.

6)
```
cout << "Enter initial vertex for BFS Traversal : ";
   cin >> v;
   cout << "The BFS of the Graph is\n";
   cout << v<<endl;
   visited[v] = 1;
   k = 1;
   while (k < n)
   {
      for (j = 1; j <= n; j++)
         if (cost[v][j] != 0 && visited[j] != 1 && visit[j] != 1)
         {
            visit[j] = 1;
            qu[rear++] = j;
         }
      v = qu[front++];
      cout << v << " ";
      k++;
      visit[v] = 0;
      visited[v] = 1;
   }
```
This section of the code performs Breadth-First Search (BFS) traversal on the graph starting from the use r-defined initial vertex ‘v‘. It uses a queue ‘qu‘ to keep track of the vertices to be visited. It iteratively visits t he neighboring vertices of the current vertex ‘v‘ and marks them as visited. The BFS traversal sequence i s printed as the output.

7)
```
cout <<endl<<"Enter initial vertex for DFS Traversal: ";
   cin >> v;
   cout << "The DFS of the Graph is\n";
   cout << v<<endl;
```

```
   visited[v] = 1;
   k = 1;
   while (k < n)
   {
      for (j = n; j >= 1; j--)
         if (cost[v][j] != 0 && visited1[j] != 1 && visit1[j] != 1)
         {
            visit1[j] = 1;
            stk[top] = j;
            top++;
         }
      v = stk[--top];
      cout << v << " ";
      k++;
      visit1[v] = 0;
      visited1[v] = 1;
   }
```

This part of the code performs Depth-First Search (DFS) traversal on the graph starting from the user-defi ned initial vertex ‘v‘. It uses a stack ‘stk‘ to keep track of the vertices to be visited. It iteratively visits the ne ighboring vertices of the current vertex ‘v‘ and marks them as visited. The DFS traversal sequence is print ed as the output.

8)return 0;
}
This line ends the main() function and returns 0, indicating successful program execution.

****

GROUP C

14)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

EXPLANATION:

1)#include<iostream>
#include<stdlib.h>
#include<string.h>
using namespace std;
This code includes necessary header files for input/output operations, standard library functions, and strin g operations.

2)struct node
{
   string vertex;
   int time;
   node *next;
};
This code defines a structure ‘node‘ that represents a node in the adjacency list. It has three members: ‘v ertex‘ (string) to store the name of the city, ‘time‘ (integer) to store the time required to reach the city, and ‘ next‘ (pointer to node) to point to the next node in the adjacency list.

3)class adjmatlist

```
{
    int m[10][10], n, i, j;
    char ch;
    string v[20];
    node *head[20];
    node *temp = NULL;
public:
    adjmatlist();
    void getgraph();
    void adjlist();
    void displaym();
    void displaya();
};
```

This code defines a class ‘adjmatlist‘ that represents an adjacency matrix and adjacency list. It has private data members ‘m‘ for the adjacency matrix, ‘n‘ for the number of cities, ‘i‘ and ‘j‘ for loop control, ‘ch‘ to store user input for path existence, ‘v‘ to store the names of cities, ‘head‘ as an array of pointers to the first node of the adjacency list for each city, and ‘temp‘ as a temporary pointer for traversing the adjacency list. It also declares public member functions to get the graph, create the adjacency list, display the adjacency matrix, and display the adjacency list.

4)

```
void adjmatlist::getgraph()
{
    cout << "\n enter no. of cities (max. 20)";
    cin >> n;
    cout << "\n enter the name of cities";
    for (i = 0; i < n; i++)
        cin >> v[i];
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            cout << "\n if a path is present between city " << v[i] << " and " << v[j] << " then press ’y’, otherwise
 ’n’";
            cin >> ch;
            if (ch == ’y’)
            {
                cout << "\n enter time required to reach city " << v[j] << " from " << v[i] << " in minutes";
                cin >> m[i][j];
            }
            else if (ch == ’n’)
            {
                m[i][j] = 0;
            }
            else
            {
                cout << "\n unknown entry";
            }
        }
    }
    adjlist();
}
```

This function ‘getgraph()‘ is a member function of the ‘adjmatlist‘ class. It prompts the user to enter the number of cities and their names. Then it asks the user whether a path exists between each pair of cities and the time required to reach each city. Based on the user’s input, the adjacency matrix ‘m‘ is updated. After getting the graph, it calls the ‘adjlist()‘ function to create the adjacency list.

```
5)void adjmatlist::adjlist()
{
    cout << "\n ****";
    for (i = 0; i < n; i++)
    {
        node *p = new (struct node);
        p->next = NULL;
        p->vertex = v[i];
        head[i] = p;
        cout << "\n" << head[i]->vertex;
    }
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            if (m[i][j] != 0)
            {
                node *p = new (struct node);
                p->vertex = v[j];
                p->time = m[i][j];
                p->next = NULL;
                if (head[i]->next == NULL)
                {
                    head[i]->next = p;
                }
                else
                {
                    temp = head[i];
                    while (temp->next != NULL)
                    {
                        temp = temp->next;
                    }
                    temp->next = p;
                }
            }
        }
    }
}
```

This function ‘adjlist()‘ is a member function of the ‘adjmatlist‘ class. It creates an adjacency list from the a djacency matrix ‘m‘. It iterates over each element of the matrix and if a path exists between two cities (non -zero value in the matrix), it creates a new node and adds it to the adjacency list of the corresponding city. The ‘head‘ array stores the pointers to the first nodes of the adjacency lists.

```
6)void adjmatlist::displaym()
{
    cout << "\n";
    for (j = 0; j < n; j++)
    {
        cout << "\t" << v[j];
    }
    for (i = 0; i < n; i++)
    {
        cout << "\n " << v[i];
        for (j = 0; j < n; j++)
```

```
        {
            cout << "\t" << m[i][j];
        }
        cout << "\n";
    }
}
```

This function ‘displaym()‘ is a member function of the ‘adjmatlist‘ class. It displays the adjacency matrix ‘m ‘ on the console. It prints the city names as column headers and then iterates over each element of the m atrix to print the corresponding values.

```
7)void adjmatlist::displaya()
{
    cout << "\n adjacency list is";
    for (i = 0; i < n; i++)
    {
        if (head[i] == NULL)
        {
            cout << "\n adjacency list not present";
            break;
        }
        else
        {
            cout << "\n" << head[i]->vertex;
            temp = head[i]->next;
            while (temp != NULL)
            {
                cout << "-> " << temp->vertex;
                temp = temp->next;
            }
        }
    }
    cout << "\n path and time required to reach cities is";
    for (i = 0; i < n; i++)
    {
        if (head[i] == NULL)
        {
            cout << "\n adjacency list not present";
            break;
        }
        else
        {
            temp = head[i]->next;
            while (temp != NULL)
            {
                cout << "\n" << head[i]->vertex << "-> " << temp->vertex << "\n   [time required: " << temp->time
 << " min ]";
                temp = temp->next;
            }
        }
    }
}
```

This function ‘displaya()‘ is a member function of the ‘adjmatlist‘ class. It displays the adjacency list and th e path with the time required to reach each city. It iterates over each city and its corresponding adjacency list. It prints the city name and then traverses the adjacency list to print the connected cities. It also prints t he time required to reach each connected city.

```
8)int main()
{
   int m;
   adjmatlist a;
   while (1)
   {
      cout << "\n 1. enter graph";
      cout << "\n 2. display adjacency matrix for cities";
      cout << "\n 3. display adjacency list for cities";
      cout << "\n 4. exit";
      cout << "\n\n enter the choice";
      cin >> m;
      switch (m)
      {
         case 1:
            a.getgraph();
            break;
         case 2:
            a.displaym();
            break;
         case 3:
            a.displaya();
            break;
         case 4:
            exit(0);
         default:
            cout << "\n unknown choice";
      }
   }
   return 0;
}
```

This is the ‘main()‘ function where the program execution starts. It creates an object ‘a‘ of the ‘adjmatlist‘ c lass. It provides a menu-driven interface to the user, allowing them to enter the graph, display the adjacen cy matrix, display the adjacency list, or exit the program based on their choice. The program keeps runnin g until the user chooses to exit.

I hope this explanation helps you understand the code! Let me know if you have any further questions.

**********//////////**********//////////

GROUP F

23)Department maintains a student information. The file contains roll number, name, division and address. Allow user to add, delete information of student. Display information of particular employee. If record of student does not exist an appropriate message is displayed. If it is, then the system displays the student details. Use sequential file to main the data.

EXPLANATION:

1)
```
#include<iostream>
#include<fstream>
#include<cstring>
```

using namespace std;
These lines include the necessary header files for input/output operations (‘iostream‘), file operations (‘fstr eam‘), and string operations (‘cstring‘). The ‘using namespace std;‘ statement allows you to use standard l ibrary objects and functions without specifying the ‘std::‘ namespace prefix.

2)class studRec {
public:
    int rollNo;
    char name[10];
    char div;
    char address[20];
    void accept() {
        cout << "\n\tEnter Roll Number: ";
        cin >> rollNo;
        cout << "\n\tEnter the Name: ";
        cin >> name;
        cout << "\n\tEnter the Division: ";
        cin >> div;
        cout << "\n\tEnter the Address: ";
        cin >> address;
    }
    void show() {
        cout << "\n\t" << rollNo << "\t\t" << name << "\t\t" << div << "\t\t" << address;
    }
};
The ‘studRec‘ class represents a student record. It has member variables ‘rollNo‘, ‘name‘, ‘div‘, and ‘addre ss‘ to store the student’s information. The ‘accept()‘ function is used to input the values for these variables , and the ‘show()‘ function is used to display the student record.

3)int main() {
    int n, ch, ch1, rec, count;
    studRec t1;
    count = 0;
    fstream g, f;
The ‘main()‘ function is the entry point of the program. It declares variables ‘n‘, ‘ch‘, ‘ch1‘, ‘rec‘, and ‘count‘ . ‘t1‘ is an object of the ‘studRec‘ class. ‘g‘ and ‘f‘ are file stream objects used for file operations.

4)do {
    cout << "\n>>>>>>>>>>>>>>>>>>>>>>MENU<<<<<<<<<<<<<<<<<<<<";
    cout << "\n1.Add Student Record";
    cout << "\n2.Display All Student Record";
    cout << "\n3.Search Student";
    cout << "\n4.Delete a Student Record";
    cout << "\n5.Exit";
    cout << "\n\tEnter the Choice\t:";
    cin >> ch;
    switch (ch) {
        case 1:
            f.open("StuRecord.txt", ios::out);
            x:
                t1.accept();
                f.write((char*)&t1, (sizeof(t1)));
                cout << "\nDo you want to enter more records?\n1.Yes\n2.No";
                cin >> ch1;
                if (ch1 == 1)

```
            goto x;
        else {
            f.close();
            break;
        }
```

The code inside the ‘do-while‘ loop displays a menu to the user and reads their choice. Based on the choi ce, it performs the corresponding operation.

- Case 1: Adding a Student Record
  - It opens a file named "StuRecord.txt" in output mode (‘ios::out‘).
  - It prompts the user to enter the student record using the ‘accept()‘ function.
  - The student record is written to the file using the ‘write()‘ function.
  - If the user wants to enter more records, it goes back to the label ‘x‘ using the ‘goto‘ statement.
  - If the user doesn’t want to enter more records, the file is closed, and the loop continues.

5)case 2:

```
    f.open("StuRecord.txt", ios::in);
    f.read((char*)&t1, (sizeof(t1)));
    cout << "\n\tRoll No.\t\tName \t\t Division \t\t Address";
    while (f) {
        t1.show();
        f.read((char*)&t1, (sizeof(t1)));
    }
    f.close();
    break;
```

- Case 2: Displaying All Student Records
  - It opens the "StuRecord.txt" file in input mode (‘ios::in‘).
  - It reads the student record from the file using the ‘read()‘ function.
  - It displays the header of the table.
  - It loops through the file and displays each student record using the ‘show()‘ function.
  - Finally, the file is closed.

6)case 3:

```
    cout << "\nEnter the roll number you want to find";
    cin >> rec;
    f.open("StuRecord.txt", ios::in);
    f.read((char*)&t1, (sizeof(t1)));
    while (f) {
        if (rec == t1.rollNo) {
            count++;
            cout << "\nRecord found";
            t1.show();
        }
        f.read((char*)&t1, (sizeof(t1)));
    }
    if (count == 0)
        cout << "\nRecord not found";
    f.close();
    break;
```

- Case 3: Searching for a Student Record
  - It prompts the user to enter the roll number to search for.
  - It opens the "StuRecord.txt" file in input mode (‘ios::in‘).
  - It reads the student record from the file using the ‘read()‘ function.
  - It compares the roll number with the entered value and displays the student record if found.
  - If the record is not found, it displays a message.
  - Finally, the file is closed.

7)case 4:

```
    int roll;
    cout << "Please Enter the Roll No. of Student Whose Info You Want to Delete: ";
    cin >> roll;
    f.open("StuRecord.txt", ios::in);
    g.open("temp.txt", ios::out);
    f.read((char*)&t1, sizeof(t1));
    while (!f.eof()) {
        if (t1.rollNo != roll)
            g.write((char*)&t1, sizeof(t1));
        f.read((char*)&t1, sizeof(t1));
    }
    cout << "The record with the roll no. " << roll << " has been deleted " << endl;
    f.close();
    g.close();
    remove("StuRecord.txt");
    rename("temp.txt", "StuRecord.txt");
    break;
}
```

- Case 4: Deleting a Student Record
  - It prompts the user to enter the roll number of the student to delete.
  - It opens the "StuRecord.txt" file in input mode (‘ios::in‘).
  - It opens a temporary file named "temp.txt" in output mode (‘ios::out‘).
  - It reads the student record from the file using the ‘read()‘ function.
  - If the roll number matches, it skips writing that record to the temporary file.
  - After processing all the records, it displays a message indicating the deletion of the record.
  - The original file is closed, the temporary file is closed, the original file is deleted using ‘remove()‘, and t he temporary file is renamed to "StuRecord.txt" using ‘rename()‘.

The ‘do-while‘ loop continues until the user enters a choice less than 5.

I hope this explanation helps you understand the code! Let me know if you have any further questions.

****

GROUP F

24)Company maintains employee information as employee ID, name, designation and salary. Allow user to add, delete information of employee. Display information of particular employee. If employee does not exist an appropriate message is displayed. If it is, then the system displays the employee details. Use index sequential file to maintain the data.

EXPLANATION:

1)

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

These lines include the necessary header files for input/output operations (‘iostream‘), file operations (‘fstr eam‘), and string operations (‘cstring‘). The ‘using namespace std;‘ statement allows you to use standard l ibrary objects and functions without specifying the ‘std::‘ namespace prefix.

2)

```
class empRec {
public:
    int empId, salary;
    char name[10], design[10];
    void accept() {
```

```
        cout << "\n\tEnter Emp ID : ";
        cin >> empId;
        cout << "\n\tEnter the Name of Emp : ";
        cin >> name;
        cout << "\n\tEnter the Designation : ";
        cin >> design;
        cout << "\n\tEnter the salary: ";
        cin >> salary;
    }
    void show() {
        cout << "\n\t" << empId << "\t" << name << "\t" << design << "\t" << salary;
    }
};
```

The ‘empRec‘ class represents an employee record. It has member variables ‘empId‘, ‘salary‘, ‘name‘, an d ‘design‘ to store the employee’s information. The ‘accept()‘ function is used to input the values for these variables, and the ‘show()‘ function is used to display the employee record.

3)

```
int main() {
    int n, ch, ch1, rec, count;
    empRec t1;
    count = 0;
    fstream g, f;
    do {
        cout << "\n>>>>>>>>>>>>>>>>>>>>>>>MENU<<<<<<<<<<<<<<<<<<<";
        cout << "\n1.Add Employee Record";
        cout << "\n2.Display All Employee Record";
        cout << "\n3.Search Employee";
        cout << "\n4.Delete an Employee Record";
        cout << "\n5.Exit";
        cout << "\n\tEnter the Choice\t: ";
        cin >> ch;
        switch (ch) {
            case 1:
                f.open("EmpRecord.txt", ios::out);
                x:
                    t1.accept();
                    f.write((char*)&t1, sizeof(t1));
                    cout << "\nDo you want to enter more records?\n1.Yes\n2.No";
                    cin >> ch1;
                    if (ch1 == 1)
                        goto x;
                    else {
                        f.close();
                        break;
                    }
            case 2:
                f.open("EmpRecord.txt", ios::in);
                f.read((char*)&t1, sizeof(t1));
                cout << "\n\tEmpId\t\tName \t\t Designation \t\t Salary";
                while (f) {
                    t1.show();
                    f.read((char*)&t1, sizeof(t1));
                }
                f.close();
                break;
```

```
        case 3:
            cout << "\nEnter the Emp ID you want to find: ";
            cin >> rec;
            f.open("EmpRecord.txt", ios::in);
            f.read((char*)&t1, sizeof(t1));
            while (f) {
                if (rec == t1.empId) {
                    count++;
                    cout << "\nRecord found";
                    t1.show();
                }
                f.read((char*)&t1, sizeof(t1));
            }
            if (count == 0)
                cout << "\nRecord not found";
            f.close();
            break;
        case 4:
            int eid;
            cout << "Please Enter the Emp ID of Employee Whose Info You Want to Delete: ";
            cin >> eid;
            f.open("EmpRecord.txt", ios::in);
            g.open("temp.txt",ios::out);
            f.read((char*)&t1, sizeof(t1));
            while (!f.eof()) {
                if (t1.empId != eid)
                    g.write((char*)&t1, sizeof(t1));
                f.read((char*)&t1, sizeof(t1));
            }
            cout << "The record with the Emp ID " << eid << " has been deleted." << endl;
            f.close();
            g.close();
            remove("EmpRecord.txt");
            rename("temp.txt", "EmpRecord.txt");
            break;
    }
  } while (ch < 5);
}
```

The ‘main()‘ function contains the menu-driven program logic for managing employee records.

- It declares variables ‘n‘, ‘ch‘, ‘ch1‘, ‘rec‘, and ‘count‘ for use in the program.
- An instance of the ‘empRec‘ class, ‘t1‘, is created to hold employee records.
- File streams ‘g‘ and ‘f‘ are declared for reading from and writing to files.
- A ‘do-while‘ loop is used to display a menu and execute the corresponding actions based on the user’s c hoice.
- Inside the loop, the menu options are displayed using ‘cout‘.
- Based on the user’s choice (‘ch‘), different actions are performed:
  - Case 1: Adding an Employee Record
    - The file "EmpRecord.txt" is opened in output mode (‘ios::out‘).
    - A label ‘x‘ is defined for looping back to the input prompt.
    - The ‘accept()‘ function is called to input employee details and store them in ‘t1‘.
    - The employee record is written to the file using ‘write()‘.
    - The user is asked if they want to enter more records. If yes, the program jumps to label ‘x‘.
    - If no, the file is closed and the loop continues to the next iteration.
  - Case 2: Displaying All Employee Records
    - The file "EmpRecord.txt" is opened in input mode (‘ios::in‘).

- Employee records are read from the file using ‘read()‘.
- The header of the table is displayed.
- Records are shown using the ‘show()‘ function.
- The file is closed.

- Case 3: Searching for an Employee Record
  - The user is prompted to enter the employee ID to search for.
  - The file "EmpRecord.txt" is opened in input mode (‘ios::in‘).
  - Employee records are read from the file using ‘read()‘.
  - If the employee ID matches, the record is displayed using ‘show()‘.
  - If no matching record is found, a message is displayed.
  - The file is closed.
- Case 4: Deleting an Employee Record
  - The user is prompted to enter the employee ID of the record to delete.
  - The file "EmpRecord.txt" is opened in input mode (‘ios::in‘).
  - A temporary file "temp.txt" is opened in output mode (‘ios::out‘).
  - Employee records are read from the original file, and if the employee ID doesn’t match the one to dele te, the record is written to the temporary file.
  - The original file is closed, the temporary file is closed, and the original file is deleted using ‘remove()‘.
  - The temporary file is renamed to "EmpRecord.txt" using ‘rename()‘.
- The ‘do-while‘ loop continues until the user enters a choice less than 5.

This program allows you to add, display, search, and delete employee records using file operations.

If you have any further questions, please let me know!