Q1. Write a program for implementing an ordered dictionary to hold a key value pair (both integer values) using array with insertions, deletion, display and find operations

Ans.
```
#include <iostream>

using namespace std;

struct Pair {
    int key;
    int value;
};
const int Max_size =100;
    Pair dictarr[Max_size];
    int Size=0;

    void insert(int key, int value) {
        if (Size >= Max_size) {
            cout << "Dictionary is full!\n";
            return;
        }
        for (int i = 0; i < Size; i++) {
            if (dictarr[i].key == key) {
                dictarr[i].value = value;
                cout << "Key already exists. Value updated.\n";
                return;
            }
        }
        int pos = Size - 1;
        while (pos >= 0 && dictarr[pos].key > key) {
            dictarr[pos + 1] = dictarr[pos];
            pos--;
        }
        dictarr[pos + 1] = {key, value};
        Size++;
        cout << "Inserted (" << key << ", " << value << ")\n";
    }
    void remove(int key) {
        int i;
        for (i = 0; i < Size; i++) {
            if (dictarr[i].key == key) break;
        }
        if (i == Size) {
            cout << "Key not found!\n";
            return;
        }
        // shift left
        for (int j = i; j < Size - 1; j++) {
            dictarr[j] = dictarr[j + 1];
        }
```

```
        Size--;
        cout << "Deleted key " << key << "\n";
    }
    void find(int key) {
        for (int i = 0; i < Size; i++) {
            if (dictarr[i].key == key) {
                cout << "Found: (" << dictarr[i].key << ", " <<
dictarr[i].value << ")\n";
                return;
            }
        }
        cout << "Key not found!\n";
    }
    void display() {
        if (Size == 0) {
            cout << "Dictionary is empty.\n";
            return;
        }
        cout << "Ordered Dictionary:\n";
        for (int i = 0; i < Size; i++) {
            cout << "(" << dictarr[i].key << ", " << dictarr[i].value << ") ";
        }
        cout << "\n";
    }
int main() {
    int choice;
    int key, value;
    do {
        cout << "\n--- Ordered Dictionary Menu ---\n";
        cout << "1. Insert\n2. Delete\n3. Find\n4. Display\n5. Exit\n";
        cout << "Enter choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter key: ";
                cin >> key ;
                cout<<"Enter value: ";
                cin >> value;
                insert(key, value);
                break;
            case 2:
                cout << "Enter key to delete: ";
                cin >> key;
                remove(key);
                break;
            case 3:
                cout << "Enter key to find: ";
```

```
                cin >> key;
                find(key);
                break;
            case 4:
                display();
                break;
            case 5:
                cout << "Exiting...\n";
                break;
            default:
                cout << "Invalid choice!\n";
        }
    } while (choice != 5);

    return 0;
}
Output:
--- Ordered Dictionary Menu ---
1. Insert
2. Delete
3. Find
4. Display
5. Exit
Enter choice: 1
Enter key: 1
Enter value: 23
Inserted (1, 23)
--- Ordered Dictionary Menu ---
1. Insert
2. Delete
3. Find
4. Display
5. Exit
Enter choice: 1
Enter key: 34
Enter value: 45
Inserted (34, 45)
--- Ordered Dictionary Menu ---
1. Insert
2. Delete
3. Find
4. Display
5. Exit
Enter choice: 1
Enter key: 34
Enter value: 67
Key already exists. Value updated.
--- Ordered Dictionary Menu ---
```

```
1. Insert
2. Delete
3. Find
4. Display
5. Exit
Enter choice: 3
Enter key to find: 34
Found: (34, 67)
--- Ordered Dictionary Menu ---
1. Insert
2. Delete
3. Find
4. Display
5. Exit
Enter choice: 5
Exiting...
```

Q2. Implement dictionary using single linked list, each node contains three elements with one key, one value and one forward pointer

Ans.
```cpp
#include <iostream>

using namespace std;
struct Node {
    int key;
    int value;
    Node* next;
};
Node* head = nullptr;
void insert(int key, int value) {
    Node* temp = head;
    while (temp != nullptr) {
        if (temp->key == key) {
            temp->value = value;
            cout << "Key already exists. Value updated.\n";
            return;
        }
        temp = temp->next;
    }
    Node* newNode = new Node{key, value, nullptr};

    if (head == nullptr || key < head->key) {
        newNode->next = head;
        head = newNode;
        cout << "Inserted (" << key << ", " << value << ")\n";
        return;
    }
```

```
    Node* prev = nullptr;
    Node* curr = head;
    while (curr != nullptr && curr->key < key) {
        prev = curr;
        curr = curr->next;
    }

    prev->next = newNode;
    newNode->next = curr;

    cout << "Inserted (" << key << ", " << value << ")\n";
}

void removeKey(int key) {
    if (head == nullptr) {
        cout << "Dictionary is empty!\n";
        return;
    }

    if (head->key == key) {
        Node* temp = head;
        head = head->next;
        delete temp;
        cout << "Deleted key " << key << "\n";
        return;
    }

    Node* curr = head;
    Node* prev = nullptr;
    while (curr != nullptr && curr->key != key) {
        prev = curr;
        curr = curr->next;
    }

    if (curr == nullptr) {
        cout << "Key not found!\n";
        return;
    }

    prev->next = curr->next;
    delete curr;
    cout << "Deleted key " << key << "\n";
}

void findKey(int key) {
    Node* temp = head;
    while (temp != nullptr) {
        if (temp->key == key) {
```

```
            cout << "Found: (" << temp->key << ", " << temp->value << ")\n";
            return;
        }
        temp = temp->next;
    }
    cout << "Key not found!\n";
}

void display() {
    if (head == nullptr) {
        cout << "Dictionary is empty.\n";
        return;
    }

    cout << "Ordered Dictionary:\n";
    Node* temp = head;
    while (temp != nullptr) {
        cout << "(" << temp->key << ", " << temp->value << ") ";
        temp = temp->next;
    }
    cout << "\n";
}

int main() {
    int choice, key, value;
    do {
        cout << "\n--- Ordered Dictionary (Linked List) Menu ---\n";
        cout << "1. Insert\n2. Delete\n3. Find\n4. Display\n5. Exit\n";
        cout << "Enter choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter key : ";
                cin >> key ;
                cout<<"Enter value: ";
                cin>> value;
                insert(key, value);
                break;
            case 2:
                cout << "Enter key to delete: ";
                cin >> key;
                removeKey(key);
                break;
            case 3:
                cout << "Enter key to find: ";
                cin >> key;
                findKey(key);
```

```
                break;
            case 4:
                display();
                break;
            case 5:
                cout << "Exiting...\n";
                break;
            default:
                cout << "Invalid choice!\n";
        }
    } while (choice != 5);

    return 0;
}
```

```
Output: --- Ordered Dictionary (Linked List) Menu ---
1. Insert
2. Delete
3. Find
4. Display
5. Exit
Enter choice: 1
Enter key and value: 23
45
Inserted (23, 45)
--- Ordered Dictionary Menu ---
1. Insert
2. Delete
3. Find
4. Display
5. Exit
Enter choice: 1
Enter key: 23
Enter value: 45
Inserted (23, 45)
--- Ordered Dictionary Menu ---
1. Insert
2. Delete
3. Find
4. Display
5. Exit
Enter choice: 1
Enter key: 23
Enter value: 46
Key already exists. Value updated.
--- Ordered Dictionary Menu ---
1. Insert
2. Delete
3. Find
```

```
4. Display
5. Exit
Enter choice: 1
Enter key: 45
Enter value: 67
Inserted (45, 67)
--- Ordered Dictionary Menu ---
1. Insert
2. Delete
3. Find
4. Display
5. Exit
Enter choice: 1
Enter key: 2
Enter value: 1
Inserted (2, 1)
```

Q3. Write a program to implement a priority queue (insertion or deletion).

Ans.
```cpp
#include <iostream>

using namespace std;

struct Element {
    int data;
    int priority;
};

const int MAX = 100;
Element pq[MAX];
int Size = 0;

void insert(int data, int priority) {
    if (Size >= MAX) {
        cout << "Priority Queue is full!\n";
        return;
    }
    pq[Size].data = data;
    pq[Size].priority = priority;
    Size++;
    cout << "Inserted (" << data << ", priority " << priority << ")\n";
}

void remove() {
    if (Size == 0) {
        cout << "Priority Queue is empty!\n";
        return;
    }
```

```
    int maxPriorityIndex = 0;
    for (int i = 1; i < Size; i++) {
        if (pq[i].priority > pq[maxPriorityIndex].priority) {
            maxPriorityIndex = i;
        }
    }

    cout << "Deleted (" << pq[maxPriorityIndex].data
         << ", priority " << pq[maxPriorityIndex].priority << ")\n";

    // Shift elements left
    for (int i = maxPriorityIndex; i < Size - 1; i++) {
        pq[i] = pq[i + 1];
    }
    Size--;
}

// Display the priority queue
void display() {
    if (Size == 0) {
        cout << "Priority Queue is empty.\n";
        return;
    }
    cout << "Priority Queue (data, priority):\n";
    for (int i = 0; i < Size; i++) {
        cout << "(" << pq[i].data << ", " << pq[i].priority << ") ";
    }
    cout << "\n";
}
int main() {
    int choice, data, priority;
    do {
        cout << "\n--- Priority Queue Menu ---\n";
        cout << "1. Insert\n2. Delete (highest priority)\n3. Display\n4.
Exit\n";
        cout << "Enter choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter data and priority: ";
                cin >> data >> priority;
                insert(data, priority);
                break;
            case 2:
                remove();
                break;
            case 3:
```

```
                display();
                break;
            case 4:
                cout << "Exiting...\n";
                break;
            default:
                cout << "Invalid choice!\n";
        }
    } while (choice != 4);

    return 0;
}
Output: --- Priority Queue Menu ---
1. Insert
2. Delete (highest priority)
3. Display
4. Exit
Enter choice: 1
Enter data and priority: 23 45
Inserted (23, priority 45)

--- Priority Queue Menu ---
1. Insert
2. Delete (highest priority)
3. Display
4. Exit
Enter choice: 1 23
Enter data and priority: 56
Inserted (23, priority 56)

--- Priority Queue Menu ---
1. Insert
2. Delete (highest priority)
3. Display
4. Exit
Enter choice: 2
Deleted (23, priority 56)
```

Q4: Write a program to implement a hash using first name, last name and contact number.

Ans.
```
#include <iostream>
#include <string>
using namespace std;

const int TABLE_SIZE = 10;
struct Node {
    string firstName;
    string lastName;
```

```
    string contact;
    Node* next;
    Node(string f, string l, string c) {
        firstName = f;
        lastName = l;
        contact = c;
        next = nullptr;
    }
};
class HashTable {
    Node* table[TABLE_SIZE];

public:
    HashTable() {
        for (int i = 0; i < TABLE_SIZE; i++) table[i] = nullptr;
    }
    int hashFunc(string f, string l) {
        int sum = 0;
        string key = f + l;
        for (char ch : key) sum += ch;
        return sum % TABLE_SIZE;
    }

    void insert(string f, string l, string c) {
        int index = hashFunc(f, l);
        Node* newNode = new Node(f, l, c);

        if (table[index] == nullptr) {
            table[index] = newNode;
        } else {
            Node* temp = table[index];
            while (temp->next != nullptr) temp = temp->next;
            temp->next = newNode;
        }
        cout << "Inserted: " << f << " " << l << " (" << c << ")\n";
    }
    void search(string f, string l) {
        int index = hashFunc(f, l);
        Node* temp = table[index];
        while (temp != nullptr) {
            if (temp->firstName == f && temp->lastName == l) {
                cout << "Found: " << temp->firstName << " "
                     << temp->lastName << " -> " << temp->contact << "\n";
                return;
            }
            temp = temp->next;
        }
        cout << "Contact not found!\n";
```

```
    }

    void remove(string f, string l) {
        int index = hashFunc(f, l);
        Node* temp = table[index];
        Node* prev = nullptr;

        while (temp != nullptr) {
            if (temp->firstName == f && temp->lastName == l) {
                if (prev == nullptr)
                    table[index] = temp->next;
                else
                    prev->next = temp->next;

                cout << "Deleted: " << f << " " << l << "\n";
                delete temp;
                return;
            }
            prev = temp;
            temp = temp->next;
        }
        cout << "Contact not found!\n";
    }
    void display() {
        for (int i = 0; i < TABLE_SIZE; i++) {
            cout << "Bucket[" << i << "]: ";
            Node* temp = table[i];
            while (temp != nullptr) {
                cout << "[" << temp->firstName << " " << temp->lastName
                     << " : " << temp->contact << "] -> ";
                temp = temp->next;
            }
            cout << "NULL\n";
        }
    }
};
int main() {
    HashTable ht;
    int choice;
    string f, l, c;

    do {
        cout << "\n--- Hash Table Menu ---\n";
        cout << "1. Insert Contact\n2. Search Contact\n3. Delete Contact\n4.
Display All\n5. Exit\n";
        cout << "Enter choice: ";
        cin >> choice;
```

```
        switch (choice) {
            case 1:
                cout << "Enter First Name, Last Name, Contact: ";
                cin >> f >> l >> c;
                ht.insert(f, l, c);
                break;
            case 2:
                cout << "Enter First Name and Last Name to search: ";
                cin >> f >> l;
                ht.search(f, l);
                break;
            case 3:
                cout << "Enter First Name and Last Name to delete: ";
                cin >> f >> l;
                ht.remove(f, l);
                break;
            case 4:
                ht.display();
                break;
            case 5:
                cout << "Exiting...\n";
                break;
            default:
                cout << "Invalid choice!\n";
        }
    } while (choice != 5);

    return 0;
}
Output:
--- Hash Table Menu ---
1. Insert Contact
2. Search Contact
3. Delete Contact
4. Display All
5. Exit
Enter choice: 1
Enter First Name, Last Name, Contact: Madhur Gahlot 6397887248
Inserted: Madhur Gahlot (6397887248)
--- Hash Table Menu ---
1. Insert Contact
2. Search Contact
3. Delete Contact
4. Display All
5. Exit
Enter choice: 1
Enter First Name, Last Name, Contact: Dev aaryan 901234534876
Inserted: Dev aaryan (901234534876)
```

```
--- Hash Table Menu ---
1. Insert Contact
2. Search Contact
3. Delete Contact
4. Display All
5. Exit
Enter choice: 2
Enter First Name and Last Name to search: Madhur Gahlot
Found: Madhur Gahlot -> 6397887248
```