1. Design, Develop and Implement a menu driven Program in C/C++ for the following Array operations

a. Creating an Array of N Integer Elements

b. Display of Array Elements

c. Inserting an Element at a given valid Position

d. search an element

e. Exit.

Ans:

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

int main() {
    int a[50], n = 0;   // array and number of elements
    int choice;

    while (true) {
        cout << "\n--- Array Operations Menu ---\n";
        cout << "1. Create Array\n";
        cout << "2. Display Array\n";
        cout << "3. Insert Element at Position\n";
        cout << "4. Search Element\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        if (choice == 1) {
            cout << "Enter number of elements: ";
            cin >> n;
            cout << "Enter " << n << " elements:\n";
            for (int i = 0; i < n; i++)
                cin >> a[i];
        }
        else if (choice == 2) {
            if (n == 0) cout << "Array is empty.\n";
            else {
                cout << "Array elements: ";
                for (int i = 0; i < n; i++)
                    cout << a[i] << " ";
                cout << endl;
            }
        }
        else if (choice == 3) {
```

```
            int pos, val;
            cout << "Enter position (0 to " << n << "): ";
            cin >> pos;
            cout << "Enter value: ";
            cin >> val;

            if (pos < 0 || pos > n)
                cout << "Invalid Position!\n";
            else {
                for (int i = n; i > pos; i--)
                    a[i] = a[i - 1];
                a[pos] = val;
                n++;
                cout << "Element Inserted.\n";
            }
        }
        else if (choice == 4) {
            int key, found = 0;
            cout << "Enter element to search: ";
            cin >> key;

            for (int i = 0; i < n; i++) {
                if (a[i] == key) {
                    cout << "Element found at index " << i << endl;
                    found = 1;
                    break;
                }
            }
            if (!found)
                cout << "Element not found.\n";
        }
        else if (choice == 5) {
            cout << "Exiting program...\n";
            break;
        }
        else {
            cout << "Invalid Choice!\n";
        }
    }
    return 0;
}
```

2.Write a program to perform following singly linked list operations. a) Create list b) display list c) insert d) search
Ans:

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

struct Node {
   int data;
   Node *next;
};

Node *head = NULL;

// Create List
void createList(int n) {
   Node *newNode, *temp;
   int value;

   for(int i = 0; i < n; i++) {
      cout << "Enter value: ";
      cin >> value;
      newNode = new Node;
      newNode->data = value;
      newNode->next = NULL;

      if(head == NULL) {
         head = newNode;
      } else {
         temp = head;
         while(temp->next != NULL)
            temp = temp->next;
         temp->next = newNode;
      }
   }
}

// Display List
void displayList() {
   Node *temp = head;
   if(head == NULL) {
      cout << "List is Empty.\n";
      return;
```

```
    }
    cout << "Linked List: ";
    while(temp != NULL) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

// Insert at Beginning
void insertNode(int value) {
    Node *newNode = new Node;
    newNode->data = value;
    newNode->next = head;
    head = newNode;
    cout << "Node Inserted.\n";
}

// Search an Element
void searchNode(int key) {
    Node *temp = head;
    int index = 0, found = 0;

    while(temp != NULL) {
        if(temp->data == key) {
            cout << "Element found at position " << index << endl;
            found = 1;
            break;
        }
        temp = temp->next;
        index++;
    }
    if(!found)
        cout << "Element not found.\n";
}

int main() {
    int choice, n, value, key;

    while(true) {
        cout << "\n--- Singly Linked List Menu ---\n";
        cout << "1. Create List\n";
        cout << "2. Display List\n";
```

```
        cout << "3. Insert at Beginning\n";
        cout << "4. Search Element\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch(choice) {
            case 1:
                cout << "Enter number of nodes: ";
                cin >> n;
                createList(n);
                break;
            case 2:
                displayList();
                break;
            case 3:
                cout << "Enter value to insert: ";
                cin >> value;
                insertNode(value);
                break;
            case 4:
                cout << "Enter value to search: ";
                cin >> key;
                searchNode(key);
                break;
            case 5:
                cout << "Program Ended.\n";
                return 0;
            default:
                cout << "Invalid Choice!\n";
        }
    }
}
```

3. Design, Develop and Implement a menu driven Program in C/C++ for the following operations on STACK (Array Implementation of Stack with maximum size MAX)

a. Push an Element on to Stack

b. Pop an Element from Stack

c. Demonstrate how Stack can be used to check Palindrome

d. Demonstrate Overflow and Underflow situations on Stack

e. Display the status of Stack

f. Exit

Ans:

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

#define MAX 5   // Maximum size of Stack

int stackArr[MAX];
int top = -1;

// Push
void push(int x) {
    if (top == MAX - 1) {
        cout << "Stack Overflow! (No space)\n";
    } else {
        stackArr[++top] = x;
        cout << "Element Pushed.\n";
    }
}

// Pop
void pop() {
    if (top == -1) {
        cout << "Stack Underflow! (No elements)\n";
    } else {
        cout << "Popped Element: " << stackArr[top--] << endl;
    }
}

// Display
void display() {
    if (top == -1) {
        cout << "Stack is Empty.\n";
```

```
    } else {
        cout << "Stack Elements: ";
        for (int i = 0; i <= top; i++)
            cout << stackArr[i] << " ";
        cout << endl;
    }
}

// Check Palindrome using Stack
void checkPalindrome() {
    string s;
    cout << "Enter String: ";
    cin >> s;

    int len = s.length();
    char st[MAX];
    int t = -1;

    // Push characters
    for (int i = 0; i < len; i++) {
        if (t == MAX - 1) break; // prevent overflow
        st[++t] = s[i];
    }

    // Pop and compare
    bool flag = true;
    for (int i = 0; i < len; i++) {
        if (s[i] != st[t--]) {
            flag = false;
            break;
        }
    }

    if (flag) cout << "It is a Palindrome.\n";
    else cout << "Not a Palindrome.\n";
}

int main() {
    int choice, value;

    while (true) {
        cout << "\n--- STACK MENU ---\n";
        cout << "1. Push\n";
```

```
        cout << "2. Pop\n";
        cout << "3. Check Palindrome\n";
        cout << "4. Display Stack\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter value to push: ";
                cin >> value;
                push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
                checkPalindrome();
                break;
            case 4:
                display();
                break;
            case 5:
                cout << "Program Ended.\n";
                return 0;
            default:
                cout << "Invalid Choice!\n";
        }
    }
}
```

4. Write a program for queue operations

Ans:

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

#define MAX 5   // Maximum size of queue

int queueArr[MAX];
int front = 0, rear = -1;
```

```
// Enqueue (Insert)
void enqueue(int x) {
    if (rear == MAX - 1) {
        cout << "Queue Overflow! (No space)\n";
    } else {
        queueArr[++rear] = x;
        cout << "Inserted.\n";
    }
}

// Dequeue (Delete)
void dequeue() {
    if (front > rear) {
        cout << "Queue Underflow! (No elements)\n";
    } else {
        cout << "Deleted Element: " << queueArr[front++] << endl;
    }
}

// Display Queue
void display() {
    if (front > rear) {
        cout << "Queue is Empty.\n";
    } else {
        cout << "Queue Elements: ";
        for (int i = front; i <= rear; i++)
            cout << queueArr[i] << " ";
        cout << endl;
    }
}

int main() {
    int choice, value;

    while (true) {
        cout << "\n--- QUEUE MENU ---\n";
        cout << "1. Enqueue (Insert)\n";
        cout << "2. Dequeue (Delete)\n";
        cout << "3. Display Queue\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
```

```
        switch (choice) {
            case 1:
                cout << "Enter value to insert: ";
                cin >> value;
                enqueue(value);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            case 4:
                cout << "Program Ended.\n";
                return 0;
            default:
                cout << "Invalid Choice!\n";
        }
    }
}
```

5. 10. Write a program to perform following circular linked list operations.
a) Create list b) display list c) insert d) search

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

struct Node {
    int data;
    Node *next;
};

Node *last = NULL;   // last pointer points to last node (circular)

// Create List (Insert at end)
void createList(int n) {
    int value;
    for(int i = 0; i < n; i++) {
        cout << "Enter value: ";
        cin >> value;

        Node *newNode = new Node;
```

```
        newNode->data = value;

        if(last == NULL) {
            last = newNode;
            last->next = last;     // circular link
        } else {
            newNode->next = last->next;
            last->next = newNode;
            last = newNode;
        }
    }
}

// Display List
void displayList() {
    if(last == NULL) {
        cout << "List is Empty.\n";
        return;
    }
    Node *temp = last->next;
    cout << "Circular List: ";
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while(temp != last->next);
    cout << endl;
}

// Insert at Beginning
void insertNode(int value) {
    Node *newNode = new Node;
    newNode->data = value;

    if(last == NULL) {
        last = newNode;
        last->next = last;
    } else {
        newNode->next = last->next;
        last->next = newNode;
    }
    cout << "Node Inserted.\n";
}
```

```
// Search Element
void searchNode(int key) {
    if(last == NULL) {
        cout << "List is Empty.\n";
        return;
    }

    Node *temp = last->next;
    int pos = 0, found = 0;

    do {
        if(temp->data == key) {
            cout << "Element found at position " << pos << endl;
            found = 1;
            break;
        }
        temp = temp->next;
        pos++;
    } while(temp != last->next);

    if(!found)
        cout << "Element not found.\n";
}

int main() {
    int choice, n, value, key;

    while(true) {
        cout << "\n--- CIRCULAR LINKED LIST MENU ---\n";
        cout << "1. Create List\n";
        cout << "2. Display List\n";
        cout << "3. Insert at Beginning\n";
        cout << "4. Search Element\n";
        cout << "5. Exit\n";
        cout << "Enter choice: ";
        cin >> choice;

        switch(choice) {
            case 1:
                cout << "Enter number of nodes: ";
                cin >> n;
                createList(n);
                break;
```

```
            case 2:
                displayList();
                break;
            case 3:
                cout << "Enter value to insert: ";
                cin >> value;
                insertNode(value);
                break;
            case 4:
                cout << "Enter value to search: ";
                cin >> key;
                searchNode(key);
                break;
            case 5:
                cout << "Program Ended.\n";
                return 0;
            default:
                cout << "Invalid Choice!\n";
        }
    }
}
```

6.Design, Develop and Implement a menu driven Program in C for the following operations on Circular QUEUE of Characters
a. Insert an Element on to Circular QUEUE
b. Delete an Element from Circular QUEUE
c. Demonstrate Overflow and Underflow situations on Circular QUEUE
d. Display the status of Circular QUEUE
e. Exit
Support the program with appropriate functions for each of the above operations
Ans:

```
#include <stdio.h>
#define MAX 5     // Maximum size of Queue

char cqueue[MAX];
int front = -1, rear = -1;

// Insert (Enqueue)
void insert(char x) {
    if ((front == 0 && rear == MAX - 1) || (rear + 1 == front)) {
        printf("Queue Overflow! (No space)\n");
        return;
```

```
    }

    if (front == -1)  // First element
        front = rear = 0;
    else
        rear = (rear + 1) % MAX;

    cqueue[rear] = x;
    printf("Inserted.\n");
}

// Delete (Dequeue)
void delete() {
    if (front == -1) {
        printf("Queue Underflow! (No elements)\n");
        return;
    }

    printf("Deleted Element: %c\n", cqueue[front]);

    if (front == rear)  // Only one element
        front = rear = -1;
    else
        front = (front + 1) % MAX;
}

// Display Queue
void display() {
    if (front == -1) {
        printf("Queue is Empty.\n");
        return;
    }

    printf("Circular Queue: ");
    int i = front;
    while (1) {
        printf("%c ", cqueue[i]);
        if (i == rear)
            break;
        i = (i + 1) % MAX;
    }
    printf("\n");
}
```

```
int main() {
   int choice;
   char ch;

   while (1) {
      printf("\n--- CIRCULAR QUEUE MENU ---\n");
      printf("1. Insert\n");
      printf("2. Delete\n");
      printf("3. Display Queue\n");
      printf("4. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:
            printf("Enter a character to insert: ");
            scanf(" %c", &ch);
            insert(ch);
            break;

         case 2:
            delete();
            break;

         case 3:
            display();
            break;

         case 4:
            printf("Program Ended.\n");
            return 0;

         default:
            printf("Invalid Choice!\n");
      }
   }
}
```

7.Write a C++ program to store first year percentage of students in array.

Write function for sorting array of floating point numbers in ascending order using
a) Merge Sort b) Bubble sort and display top five scores.

Ans:

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

// --------- Merge Sort Functions ----------
void merge(float a[], int l, int m, int r) {
    int i = l, j = m + 1, k = 0;
    float temp[r - l + 1];

    while(i <= m && j <= r)
        temp[k++] = (a[i] < a[j]) ? a[i++] : a[j++];

    while(i <= m) temp[k++] = a[i++];
    while(j <= r) temp[k++] = a[j++];

    for(i = l, k = 0; i <= r; i++, k++)
        a[i] = temp[k];
}

void mergeSort(float a[], int l, int r) {
    if(l < r) {
        int m = (l + r) / 2;
        mergeSort(a, l, m);
        mergeSort(a, m + 1, r);
        merge(a, l, m, r);
    }
}

// --------- Bubble Sort ----------
void bubbleSort(float a[], int n) {
    for(int i = 0; i < n - 1; i++)
        for(int j = 0; j < n - i - 1; j++)
            if(a[j] > a[j + 1])
                swap(a[j], a[j + 1]);
}

// --------- Display Top Five ----------
void displayTopFive(float a[], int n) {
    cout << "\nTop Five Scores:\n";
```

```
    int count = min(5, n);
    for(int i = n - 1; i >= n - count; i--)
        cout << a[i] << " ";
    cout << endl;
}

int main() {
    int n;
    cout << "Enter number of students: ";
    cin >> n;

    float per[n], perMerge[n], perBubble[n];

    cout << "Enter percentages:\n";
    for(int i = 0; i < n; i++) {
        cin >> per[i];
        perMerge[i] = per[i];
        perBubble[i] = per[i];
    }

    // Sort using Merge Sort
    mergeSort(perMerge, 0, n - 1);
    cout << "\nAfter Merge Sort (Ascending): ";
    for(int i = 0; i < n; i++) cout << perMerge[i] << " ";
    cout << endl;
    displayTopFive(perMerge, n);

    // Sort using Bubble Sort
    bubbleSort(perBubble, n);
    cout << "\nAfter Bubble Sort (Ascending): ";
    for(int i = 0; i < n; i++) cout << perBubble[i] << " ";
    cout << endl;
    displayTopFive(perBubble, n);

    return 0;
}
```

8.Create a sorting application for an online bookstore, where customers can easily browse through a vast collection of books sorted alphabetically by their titles, enabling a seamless and user-friendly shopping experience.

Ans:

```
#include <iostream>
```

```
#include <string>
using namespace std;

int main() {
    int n;
    cout << "Enter number of books: ";
    cin >> n;
    cin.ignore();   // clear input buffer

    string books[n];

    // Accept book titles
    cout << "Enter book titles:\n";
    for (int i = 0; i < n; i++) {
        getline(cin, books[i]);
    }

    // Sorting (Bubble Sort based on alphabetical order)
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (books[j] > books[j + 1]) {
                swap(books[j], books[j + 1]);
            }
        }
    }

    // Display Sorted List
    cout << "\nBooks Sorted Alphabetically:\n";
    for (int i = 0; i < n; i++) {
        cout << books[i] << endl;
    }

    return 0;
}
```

9.Develop a C++ program for a student information system that implements direct access file organization. The program should allow users to add, update, delete, and search for student records efficiently using direct access techniques.

Ans:

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

```
using namespace std;

struct Student {
    int roll;
    char name[30];
    int age;
};

// File name
const char *filename = "students.dat";

// Calculate position in file
long getPosition(int roll) {
    return roll * sizeof(Student);
}

// Add or Update a student record
void addOrUpdate() {
    Student s;
    cout << "Enter Roll No: ";
    cin >> s.roll;
    cout << "Enter Name: ";
    cin >> s.name;
    cout << "Enter Age: ";
    cin >> s.age;

    fstream file(filename, ios::in | ios::out | ios::binary);
    if(!file) file.open(filename, ios::out | ios::binary);

    file.seekp(getPosition(s.roll));
    file.write((char*)&s, sizeof(s));
    file.close();
    cout << "Record Saved Successfully.\n";
}

// Search student record
void searchRecord() {
    Student s;
    int roll;
    cout << "Enter Roll No to Search: ";
    cin >> roll;

    ifstream file(filename, ios::binary);
```

```
    file.seekg(getPosition(roll));
    file.read((char*)&s, sizeof(s));

    if(file && s.roll != 0)
        cout << "Roll: " << s.roll << "  Name: " << s.name << "  Age: " << s.age <<
endl;
    else
        cout << "Record Not Found.\n";

    file.close();
}

// Delete record (overwrite empty record)
void deleteRecord() {
    int roll;
    cout << "Enter Roll No to Delete: ";
    cin >> roll;

    Student empty = {0, "", 0};
    fstream file(filename, ios::in | ios::out | ios::binary);
    file.seekp(getPosition(roll));
    file.write((char*)&empty, sizeof(empty));
    file.close();
    cout << "Record Deleted.\n";
}

int main() {
    int choice;
    while (true) {
        cout << "\n--- Student Information System (Direct Access) ---\n";
        cout << "1. Add / Update Record\n";
        cout << "2. Delete Record\n";
        cout << "3. Search Record\n";
        cout << "4. Exit\n";
        cout << "Enter Your Choice: ";
        cin >> choice;

        switch (choice) {
            case 1: addOrUpdate(); break;
            case 2: deleteRecord(); break;
            case 3: searchRecord(); break;
            case 4: cout << "Thank You!"; return 0;
            default: cout << "Invalid Choice.\n";
```

```
        }
    }
}
```

10. A department maintains employee information comprising employee ID, name, department, and address. Design a program to manage this employee data using a sequential file. The program should enable users to add, delete, and display information for each employee. When displaying an employee's information, if the record does not exist, the program should display an appropriate message. If the employee record is found, it should display the employee's details.

Ans:

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

struct Employee {
    int id;
    string name;
    string dept;
    string addr;
};

const string filename = "employees.txt";

// Add record
void addEmployee() {
    Employee e;
    cout << "Enter Employee ID: ";
    cin >> e.id;
    cout << "Enter Name: ";
    cin >> e.name;
    cout << "Enter Department: ";
    cin >> e.dept;
    cout << "Enter Address: ";
    cin >> e.addr;

    ofstream file(filename, ios::app);
    file << e.id << " " << e.name << " " << e.dept << " " << e.addr << "\n";
    file.close();
```

```
    cout << "Record Added Successfully.\n";
}

// Display all records
void displayAll() {
    ifstream file(filename);
    if (!file) {
        cout << "No employee records found.\n";
        return;
    }

    Employee e;
    cout << "\n--- Employee Records ---\n";
    while (file >> e.id >> e.name >> e.dept >> e.addr) {
        cout << "ID: " << e.id
             << "  Name: " << e.name
             << "  Dept: " << e.dept
             << "  Address: " << e.addr << endl;
    }
    file.close();
}

// Search record by ID
void searchEmployee() {
    int id;
    cout << "Enter Employee ID to Search: ";
    cin >> id;

    ifstream file(filename);
    Employee e;
    bool found = false;

    while (file >> e.id >> e.name >> e.dept >> e.addr) {
        if (e.id == id) {
            cout << "\nEmployee Found:\n";
            cout << "ID: " << e.id << "\nName: " << e.name
                 << "\nDepartment: " << e.dept
                 << "\nAddress: " << e.addr << endl;
            found = true;
            break;
        }
    }
    file.close();
```

```
    if (!found)
        cout << "Record Not Found.\n";
}

// Delete record by ID
void deleteEmployee() {
    int id;
    cout << "Enter Employee ID to Delete: ";
    cin >> id;

    ifstream file(filename);
    ofstream temp("temp.txt");

    Employee e;
    bool found = false;

    while (file >> e.id >> e.name >> e.dept >> e.addr) {
        if (e.id == id) {
            found = true;
            continue;
        }
        temp << e.id << " " << e.name << " " << e.dept << " " << e.addr << "\n";
    }

    file.close();
    temp.close();

    remove(filename.c_str());
    rename("temp.txt", filename.c_str());

    if (found)
        cout << "Record Deleted.\n";
    else
        cout << "Record Not Found.\n";
}

int main() {
    int choice;
    while (true) {
        cout << "\n--- Employee File System (Sequential) ---\n";
        cout << "1. Add Employee\n";
        cout << "2. Delete Employee\n";
```

```
        cout << "3. Display All Employees\n";
        cout << "4. Search Employee\n";
        cout << "5. Exit\n";
        cout << "Enter Choice: ";
        cin >> choice;

        switch (choice) {
            case 1: addEmployee(); break;
            case 2: deleteEmployee(); break;
            case 3: displayAll(); break;
            case 4: searchEmployee(); break;
            case 5: cout << "Program Ended.\n"; return 0;
            default: cout << "Invalid Choice!\n";
        }
    }
}
```

11. Develop a program to manage and sort student records based on different criteria. Each student record should include the following details:

- **Roll Number** (Integer)
- **Name** (String)
- **Marks** (Float)

The application should allow the user to:

Input student records (up to 100 records).

Display the records in their current order.

Sort the records based on:

Roll Number (Ascending order)

Name (Lexicographical order)

Marks (Descending order)

Ans:

```
#include <iostream>

#include <string>

using namespace std;
```

```
struct Student {
    int roll;
    string name;
    float marks;
};

int main() {
    Student s[100];
    int n, choice;

    cout << "Enter number of students (max 100): ";
    cin >> n;

    // Input student records
    for(int i = 0; i < n; i++) {
        cout << "\nEnter Roll, Name, Marks: ";
        cin >> s[i].roll >> s[i].name >> s[i].marks;
    }

    while(true) {
        cout << "\n--- Student Record Menu ---\n";
        cout << "1. Display Records\n";
        cout << "2. Sort by Roll Number (Ascending)\n";
        cout << "3. Sort by Name (Alphabetical)\n";
        cout << "4. Sort by Marks (Descending)\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
```

```
cin >> choice;

switch(choice) {

    case 1:
        cout << "\nRoll\tName\tMarks\n";
        for(int i = 0; i < n; i++)
            cout << s[i].roll << "\t" << s[i].name << "\t" << s[i].marks << endl;
        break;

    case 2:
        for(int i = 0; i < n - 1; i++)
            for(int j = i + 1; j < n; j++)
                if(s[i].roll > s[j].roll)
                    swap(s[i], s[j]);
        cout << "Sorted by Roll Number.\n";
        break;

    case 3:
        for(int i = 0; i < n - 1; i++)
            for(int j = i + 1; j < n; j++)
                if(s[i].name > s[j].name)
                    swap(s[i], s[j]);
        cout << "Sorted by Name.\n";
        break;

    case 4:
```

```
            for(int i = 0; i < n - 1; i++)
                for(int j = i + 1; j < n; j++)
                    if(s[i].marks < s[j].marks)
                        swap(s[i], s[j]);
            cout << "Sorted by Marks.\n";
            break;

        case 5:
            cout << "Program Ended.\n";
            return 0;

        default:
            cout << "Invalid Choice!\n";
    }
  }
}
```

12. Design a C++ application for an e-commerce platform to manage a product catalog. Each product has the following attributes: **Product ID, Name, Price, Ratings** The program should enable users to sort the products by:

1. Price (low to high).
2. Ratings (high to low).
3. Name (alphabetically).

Ans:

```
#include <iostream>

#include <string>

using namespace std;
```

```
struct Product {
    int id;
    string name;
    float price;
    float rating;
};

int main() {
    Product p[100];
    int n, choice;

    cout << "Enter number of products: ";
    cin >> n;

    // Input product records
    for(int i = 0; i < n; i++) {
        cout << "\nEnter Product ID, Name, Price, Rating: ";
        cin >> p[i].id >> p[i].name >> p[i].price >> p[i].rating;
    }

    while(true) {
        cout << "\n--- PRODUCT CATALOG MENU ---\n";
```

```
        cout << "1. Display Products\n";
        cout << "2. Sort by Price (Low to High)\n";
        cout << "3. Sort by Rating (High to Low)\n";
        cout << "4. Sort by Name (Alphabetically)\n";
        cout << "5. Exit\n";
        cout << "Enter choice: ";
        cin >> choice;

        switch(choice) {

            case 1: // Display
                cout << "\nID\tName\tPrice\tRating\n";
                for(int i = 0; i < n; i++)
                    cout << p[i].id << "\t" << p[i].name << "\t" << p[i].price <<
"\t" << p[i].rating << endl;
                break;

            case 2: // Price Low -> High
                for(int i = 0; i < n - 1; i++)
                    for(int j = i + 1; j < n; j++)
                        if(p[i].price > p[j].price)
                            swap(p[i], p[j]);
                cout << "Sorted by Price (Low → High).\n";
                break;
```

```
case 3: // Rating High -> Low
    for(int i = 0; i < n - 1; i++)
        for(int j = i + 1; j < n; j++)
            if(p[i].rating < p[j].rating)
                swap(p[i], p[j]);
    cout << "Sorted by Rating (High → Low).\n";
    break;

case 4: // Name Alphabetically
    for(int i = 0; i < n - 1; i++)
        for(int j = i + 1; j < n; j++)
            if(p[i].name > p[j].name)
                swap(p[i], p[j]);
    cout << "Sorted by Name (A → Z).\n";
    break;

case 5:
    cout << "Program Ended.\n";
    return 0;

default:
    cout << "Invalid Choice!\n";
```

```
        }
    }
}
```

13. Develop a C++ program to manage employee salary data. Each employee record should contain: **Employee ID, Name, Department, Salary** The program should allow the user to:

1. Enter and store records.
2. Display all records.
3. Sort employees by:
    - Employee ID (Ascending).
    - Name (Alphabetical).
    - Salary (Descending).

Ans:

```
#include <iostream>

#include <string>

using namespace std;


struct Employee {

    int id;

    string name;

    string dept;

    float salary;

};


int main() {
```

```
Employee e[100];
int n, choice;

cout << "Enter number of employees (max 100): ";
cin >> n;

// Input employee records
for(int i = 0; i < n; i++) {
    cout << "\nEnter Employee ID, Name, Department, Salary:\n";
    cin >> e[i].id >> e[i].name >> e[i].dept >> e[i].salary;
}

while(true) {
    cout << "\n--- EMPLOYEE SALARY MENU ---\n";
    cout << "1. Display All Records\n";
    cout << "2. Sort by Employee ID (Ascending)\n";
    cout << "3. Sort by Name (Alphabetical)\n";
    cout << "4. Sort by Salary (Descending)\n";
    cout << "5. Exit\n";
    cout << "Enter your choice: ";
    cin >> choice;

    switch(choice) {
```

```
            case 1:
                cout << "\nID\tName\tDept\tSalary\n";
                for(int i = 0; i < n; i++)
                    cout << e[i].id << "\t" << e[i].name << "\t" << e[i].dept << "\t"
<< e[i].salary << endl;
                break;

            case 2: // Sort by ID (ascending)
                for(int i = 0; i < n-1; i++)
                    for(int j = i+1; j < n; j++)
                        if(e[i].id > e[j].id)
                            swap(e[i], e[j]);
                cout << "Sorted by Employee ID (Ascending).\n";
                break;

            case 3: // Sort by Name (alphabetical)
                for(int i = 0; i < n-1; i++)
                    for(int j = i+1; j < n; j++)
                        if(e[i].name > e[j].name)
                            swap(e[i], e[j]);
                cout << "Sorted by Name (Alphabetical).\n";
                break;

            case 4: // Sort by Salary (descending)
```

```
                for(int i = 0; i < n-1; i++)
                    for(int j = i+1; j < n; j++)
                        if(e[i].salary < e[j].salary)
                            swap(e[i], e[j]);
                cout << "Sorted by Salary (High → Low).\n";
                break;

            case 5:
                cout << "Program Ended.\n";
                return 0;

            default:
                cout << "Invalid Choice!\n";
        }
    }
}
```

14. Design a C++ program to manage a library's book records using direct access files. Each book record contains the following fields: **Book ID** (unique integer), **Title** (string), **Author** (string), **Availability Status** (boolean: true for available, false for issued)

The program should allow the user to:

1. Add new book records.
2. Search for a book by its **Book ID** and display its details.
3. Update the availability status of a book.
4. Delete a book record.

Ans: #include <iostream>

#include <fstream>

#include <cstring>

using namespace std;

```
struct Book {
    int id;
    char title[50];
    char author[30];
    bool available;   // true = available, false = issued
};

const char *FILE_NAME = "library.dat";

// Get position based on Book ID
long getPosition(int id) {
    return (long)id * sizeof(Book);
}

// Add New Book or Update Existing
void addBook() {
    Book b;
    cout << "Enter Book ID: ";
```

```
    cin >> b.id;
    cin.ignore();
    cout << "Enter Title: ";
    cin.getline(b.title, 50);
    cout << "Enter Author: ";
    cin.getline(b.author, 30);
    b.available = true;

    fstream file(FILE_NAME, ios::in | ios::out | ios::binary);
    if(!file) file.open(FILE_NAME, ios::out | ios::binary);

    file.seekp(getPosition(b.id));
    file.write((char*)&b, sizeof(b));
    file.close();

    cout << "Book Record Saved.\n";
}

// Search Book Record
void searchBook() {
    int id;
    cout << "Enter Book ID to search: ";
    cin >> id;
```

```
    ifstream file(FILE_NAME, ios::binary);
    Book b;
    file.seekg(getPosition(id));
    file.read((char*)&b, sizeof(b));
    file.close();

    if(b.id == 0)
        cout << "Record Not Found.\n";
    else {
        cout << "\nBook Details:\n";
        cout << "ID: " << b.id << "\nTitle: " << b.title << "\nAuthor: " << b.author;
        cout << "\nStatus: " << (b.available ? "Available" : "Issued") << endl;
    }
}

// Update Availability (Issue / Return)
void updateStatus() {
    int id;
    cout << "Enter Book ID to update status: ";
    cin >> id;

    fstream file(FILE_NAME, ios::in | ios::out | ios::binary);
```

```
    Book b;
    file.seekg(getPosition(id));
    file.read((char*)&b, sizeof(b));

    if(b.id == 0) {
        cout << "Record Not Found.\n";
        return;
    }

    b.available = !b.available;  // Toggle availability
    file.seekp(getPosition(id));
    file.write((char*)&b, sizeof(b));
    file.close();

    cout << "Availability status updated.\n";
}

// Delete Book Record
void deleteBook() {
    int id;
    cout << "Enter Book ID to delete: ";
    cin >> id;
```

```
    Book empty = {0, "", "", false};
    fstream file(FILE_NAME, ios::in | ios::out | ios::binary);
    file.seekp(getPosition(id));
    file.write((char*)&empty, sizeof(empty));
    file.close();

    cout << "Book Record Deleted.\n";
}

int main() {
    int choice;
    while(true) {
        cout << "\n--- LIBRARY BOOK MANAGEMENT (DIRECT ACCESS FILE) ---\n";
        cout << "1. Add Book Record\n";
        cout << "2. Search Book by ID\n";
        cout << "3. Update Book Availability\n";
        cout << "4. Delete Book Record\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch(choice) {
            case 1: addBook(); break;
```

```
            case 2: searchBook(); break;
            case 3: updateStatus(); break;
            case 4: deleteBook(); break;
            case 5: cout << "Program Ended.\n"; return 0;
            default: cout << "Invalid Choice!\n";
        }
    }
}
```

15. Develop a C++ program to store and retrieve student examination results using sequential files. Each record includes: Roll Number (unique integer), Name (string), Subject Marks (array of 5 integers), The program should allow the user to:

Add new student records.

Retrieve a student's details by their Roll Number.

Update a student's marks for a specific subject.

Delete a student's record.

Ans:

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

struct Student {
    int roll;
    string name;
    int marks[5];
};
```

```
const string filename = "results.txt";

// Add new student record
void addRecord() {
    Student s;
    cout << "Enter Roll Number: ";
    cin >> s.roll;
    cout << "Enter Name: ";
    cin >> s.name;
    cout << "Enter marks of 5 subjects: ";
    for(int i = 0; i < 5; i++)
        cin >> s.marks[i];

    ofstream file(filename, ios::app);
    file << s.roll << " " << s.name;
    for(int i = 0; i < 5; i++)
        file << " " << s.marks[i];
    file << "\n";
    file.close();

    cout << "Record Added Successfully.\n";
}

// Retrieve student record by roll number
void retrieveRecord() {
    int r;
    cout << "Enter Roll Number to search: ";
```

```
    cin >> r;

    ifstream file(filename);
    Student s;
    bool found = false;

    while(file >> s.roll >> s.name >> s.marks[0] >> s.marks[1] >> s.marks[2]
>> s.marks[3] >> s.marks[4]) {
        if(s.roll == r) {
            cout << "\nRecord Found:\n";
            cout << "Roll No: " << s.roll << "\nName: " << s.name << "\nMarks:
";
            for(int i = 0; i < 5; i++)
                cout << s.marks[i] << " ";
            cout << endl;
            found = true;
            break;
        }
    }
    file.close();

    if(!found)
        cout << "Record Not Found.\n";
}

// Update marks of one subject
void updateMarks() {
    int r, subjectIndex, newMarks;
```

```
    cout << "Enter Roll Number to update: ";
    cin >> r;
    cout << "Enter subject index (0 to 4): ";
    cin >> subjectIndex;
    cout << "Enter new marks: ";
    cin >> newMarks;

    ifstream file(filename);
    ofstream temp("temp.txt");
    Student s;
    bool found = false;

    while(file >> s.roll >> s.name >> s.marks[0] >> s.marks[1] >> s.marks[2]
>> s.marks[3] >> s.marks[4]) {
        if(s.roll == r) {
            s.marks[subjectIndex] = newMarks;
            found = true;
        }
        temp << s.roll << " " << s.name;
        for(int i = 0; i < 5; i++)
            temp << " " << s.marks[i];
        temp << "\n";
    }
    file.close();
    temp.close();

    remove(filename.c_str());
```

```
    rename("temp.txt", filename.c_str());

    if(found)
        cout << "Marks Updated Successfully.\n";
    else
        cout << "Record Not Found.\n";
}

// Delete a student record
void deleteRecord() {
    int r;
    cout << "Enter Roll Number to delete: ";
    cin >> r;

    ifstream file(filename);
    ofstream temp("temp.txt");
    Student s;
    bool found = false;

    while(file >> s.roll >> s.name >> s.marks[0] >> s.marks[1] >> s.marks[2]
>> s.marks[3] >> s.marks[4]) {
        if(s.roll == r) {
            found = true;
            continue;
        }
        temp << s.roll << " " << s.name;
        for(int i = 0; i < 5; i++)
```

```
            temp << " " << s.marks[i];
        temp << "\n";
    }
    file.close();
    temp.close();

    remove(filename.c_str());
    rename("temp.txt", filename.c_str());

    if(found)
        cout << "Record Deleted.\n";
    else
        cout << "Record Not Found.\n";
}

int main() {
    int choice;
    while(true) {
        cout << "\n--- STUDENT RESULT MANAGEMENT SYSTEM ---\n";
        cout << "1. Add Record\n";
        cout << "2. Retrieve Record\n";
        cout << "3. Update Marks\n";
        cout << "4. Delete Record\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
```

```
        switch(choice) {
            case 1: addRecord(); break;
            case 2: retrieveRecord(); break;
            case 3: updateMarks(); break;
            case 4: deleteRecord(); break;
            case 5: cout << "Program Ended.\n"; return 0;
            default: cout << "Invalid Choice!\n";
        }
    }
}
```

16. Implement a C++ program that simulates an Undo/Redo feature using two stacks.

Ans:

```
#include <iostream>
#include <stack>
#include <string>
using namespace std;

int main() {
    stack<string> undoStack, redoStack;
    string text = "", input;
    int choice;

    while (true) {
        cout << "\n--- UNDO / REDO MENU ---\n";
```

```
cout << "1. Type Text (Append)\n";
cout << "2. Undo\n";
cout << "3. Redo\n";
cout << "4. Show Current Text\n";
cout << "5. Exit\n";
cout << "Enter choice: ";
cin >> choice;
cin.ignore();

switch(choice) {
   case 1:
      cout << "Enter text to append: ";
      getline(cin, input);

      undoStack.push(text);   // Save current state
      while(!redoStack.empty()) redoStack.pop(); // Clear redo

      text += input;          // Update text
      break;

   case 2:
      if(undoStack.empty())
         cout << "Nothing to Undo.\n";
```

```
    else {
        redoStack.push(text);       // Save current
        text = undoStack.top();     // Restore previous
        undoStack.pop();
    }
    break;

case 3:
    if(redoStack.empty())
        cout << "Nothing to Redo.\n";
    else {
        undoStack.push(text);       // Save current
        text = redoStack.top();     // Restore undone
        redoStack.pop();
    }
    break;

case 4:
    cout << "Current Text: " << text << endl;
    break;

case 5:
    cout << "Program Ended.\n";
```

```
            return 0;


        default:
            cout << "Invalid Choice!\n";
    }
  }
}
```