***Note: Just read all the comments and the explanations, they are for your understanding. You don't need to write them in the journal.***

**Program 1:**

```
#include<iostream>
using namespace std;
class BINARY {
private:
   short int list[30];
   short unsigned int N;

   void search(short int num) {
      int low = 0, high = N - 1;
      bool found = false;

      while (low <= high) {
         int mid = low + (high - low) / 2;

         if (list[mid] == num) {
            cout << "Number " << num << " found at index " << mid << "." << endl;
            found = true;
            break;
         } else if (list[mid] < num) {
            low = mid + 1;
         } else {
            high = mid - 1;
         }
      }

      if (!found) {
         cout << "Number " << num << " not found in the list." << endl;
      }
   }

public:
   BINARY() {
      cout << "Enter the total number of elements (N): ";
      cin >> N;

      cout << "Enter " << N << " numbers for the list (in ascending order):" << endl;
      for (short unsigned int i = 0; i < N; ++i) {
         cout << "Element " << i + 1 << ": ";
         cin >> list[i];
      }

      int numToSearch;
      cout << "Enter the number to search in the list: ";
      cin >> numToSearch;
```

```
        // Call the private search function
        search(numToSearch);
    }
};

int main() {
    // Create an object of the BINARY class
    BINARY binaryObject;

    return 0;
}
```

**Explanation:** This C++ program defines a class BINARY with the specified members and a default constructor that accepts the array size and elements. The constructor also calls the private member function search to perform a binary search on the array. The main function creates an object of the BINARY class to demonstrate its usage.

**Program 2:**

```
#include <iostream>
using namespace std;
class BUBBLE {
private:
    short int list[30];
    short unsigned int N;

    void sort() {
        for (int i = 0; i < N - 1; ++i) {
            for (int j = 0; j < N - i - 1; ++j) {
                if (list[j] > list[j + 1]) {
                    short int temp = list[j];
                    list[j] = list[j + 1];
                    list[j + 1] = temp;
                }
            }
        }
    }

    void show() {
        cout << "Sorted Array: ";
        for (int i = 0; i < N; ++i) {
            cout << list[i] << " ";
        }
        cout << endl;
    }

public:
    BUBBLE(short unsigned int num) : N(num) {
        cout << "Enter " << N << " numbers for the array:" << endl;
        for (int i = 0; i < N; ++i) {
```

```
            cin >> list[i];
        }

        sort();
        show();
    }
};

int main() {
    short unsigned int totalElements;
    cout << "Enter the total number of elements (<= 30): ";
    cin >> totalElements;

    BUBBLE bubbleObj(totalElements);

    return 0;
}
```

**Explanation:** In this example, the BUBBLE class has private members list and N, a default constructor, and two private member functions sort and show. The main function initializes an object of the class by accepting the total number of elements and then calls the class's default constructor, which, in turn, initializes the array, sorts it using the bubble sort technique, and displays the sorted array.

**Program 3:**

```
#include<iostream>
using namespace std;
class SELECT {
private:
    short int list[30];
    short unsigned int N;

    void sort() {
        // Selection sort implementation
        for (short unsigned int i = 0; i < N - 1; ++i) {
            short unsigned int minIndex = i;

            for (short unsigned int j = i + 1; j < N; ++j) {
                if (list[j] < list[minIndex]) {
                    minIndex = j;
                }
            }

            // Swap the found minimum element with the first element
            swap(list[i], list[minIndex]);
        }
    }

    void show() {
        // Display the content of the array
```

```
        cout << "Sorted Array: ";
        for (short unsigned int i = 0; i < N; ++i) {
            cout << list[i] << " ";
        }
        cout << endl;
    }

public:
    SELECT() {
        cout << "Enter the total number of elements (N): ";
        cin >> N;

        cout << "Enter " << N << " numbers for the list:" << endl;
        for (short unsigned int i = 0; i < N; ++i) {
            cout << "Element " << i + 1 << ": ";
            cin >> list[i];
        }

        // Call the private sort function
        sort();

        // Call the private show function to display the sorted array
        show();
    }
};

int main() {
    // Create an object of the SELECT class
    SELECT selectObject;

    return 0;
}
```

**Explanation:** This C++ program defines a class named SELECT with the specified members. The default constructor accepts the array size and elements, and then calls the private member function sort to perform selection sort on the array in ascending order. Finally, it calls the private member function to display the sorted array. The main function creates an object of the SELECT class to demonstrate its usage.

**Program 4:**

```
#include<iostream>
using namespace std;
class INSERT {
private:
    short int list[30];
    short unsigned int N;

    void sort() {
        // Insertion sort implementation
```

```
        for (short unsigned int i = 1; i < N; ++i) {
            short int key = list[i];
            short int j = i - 1;

            // Move elements greater than key to one position ahead of their current position
            while (j >= 0 && list[j] > key) {
                list[j + 1] = list[j];
                j--;
            }

            list[j + 1] = key;
        }
    }

    void show() {
        // Display the content of the array
        cout << "Sorted Array: ";
        for (short unsigned int i = 0; i < N; ++i) {
            cout << list[i] << " ";
        }
        cout << endl;
    }

public:
    INSERT() {
        cout << "Enter the total number of elements (N): ";
        cin >> N;

        cout << "Enter " << N << " numbers for the list:" << endl;
        for (short unsigned int i = 0; i < N; ++i) {
            cout << "Element " << i + 1 << ": ";
            cin >> list[i];
        }

        // Call the private sort function
        sort();

        // Call the private show function to display the sorted array
        show();
    }
};

int main() {
    // Create an object of the INSERT class
    INSERT insertObject;

    return 0;
}
```

**Explanation:** This C++ program defines an INSERT class with the specified members. The default constructor accepts the array size and elements, and then calls the private member function sort to perform insertion sort on the array in ascending order. Finally, it calls the private member function to display the sorted array. The main function creates an object of the INSERT class to demonstrate its usage.

**Program 5:**

```
#include<iostream>
using namespace std;
class MERGE {
private:
   short int A[50];
   short int B[50];
   short int C[50];
   short unsigned int M;
   short unsigned int N;

   void process() {
      // Merge arrays A and B into C in ascending order
      short unsigned int i = 0, j = 0, k = 0;

      while (i < M && j < N) {
         if (A[i] < B[j]) {
            C[k++] = A[i++];
         } else {
            C[k++] = B[j++];
         }
      }

      // Copy the remaining elements of A, if any
      while (i < M) {
         C[k++] = A[i++];
      }

      // Copy the remaining elements of B, if any
      while (j < N) {
         C[k++] = B[j++];
      }
   }

public:
   MERGE() {
      cout << "Enter the total number of elements for array A (M): ";
      cin >> M;

      cout << "Enter " << M << " numbers for array A in ascending order:" << endl;
      for (short unsigned int i = 0; i < M; ++i) {
         cout << "Element " << i + 1 << ": ";
         cin >> A[i];
```

```
        }

        cout << "Enter the total number of elements for array B (N): ";
        cin >> N;

        cout << "Enter " << N << " numbers for array B in ascending order:" << endl;
        for (short unsigned int i = 0; i < N; ++i) {
            cout << "Element " << i + 1 << ": ";
            cin >> B[i];
        }

        // Call the private process function to merge arrays A and B into C
        process();

        // Display the merged array C
        cout << "Merged Array (C) in ascending order: ";
        for (short unsigned int i = 0; i < M + N; ++i) {
            cout << C[i] << " ";
        }
        cout << endl;
    }
};

int main() {
    // Create an object of the MERGE class
    MERGE mergeObject;

    return 0;
}
```

**Explanation:** This C++ program defines a class named MERGE with the specified members. The default constructor accepts the array sizes and elements for arrays A and B, then calls the private member function process to merge arrays A and B into C in ascending order. Finally, it displays the merged array C. The main function creates an object of the MERGE class to demonstrate its usage.

**Program 6:**

```
#include<iostream>
using namespace std;
class MATRIX {
private:
    short unsigned int M, N, Q;
    int A[10][10];
    int B[10][10];
    int C[10][10];

    void product() {
        // Compute the product of matrices A and B and store in C
        for (short unsigned int i = 0; i < M; ++i) {
            for (short unsigned int j = 0; j < Q; ++j) {
```

```
            C[i][j] = 0;
            for (short unsigned int k = 0; k < N; ++k) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}

public:
    MATRIX() {
        cout << "Enter the number of rows for matrix A (M): ";
        cin >> M;

        cout << "Enter the number of columns for matrix A and rows for matrix B (N): ";
        cin >> N;

        cout << "Enter the number of columns for matrix B (Q): ";
        cin >> Q;

        cout << "Enter elements for matrix A (" << M << "x" << N << "):" << endl;
        for (short unsigned int i = 0; i < M; ++i) {
            for (short unsigned int j = 0; j < N; ++j) {
                cout << "Element A[" << i + 1 << "][" << j + 1 << "]: ";
                cin >> A[i][j];
            }
        }

        cout << "Enter elements for matrix B (" << N << "x" << Q << "):" << endl;
        for (short unsigned int i = 0; i < N; ++i) {
            for (short unsigned int j = 0; j < Q; ++j) {
                cout << "Element B[" << i + 1 << "][" << j + 1 << "]: ";
                cin >> B[i][j];
            }
        }

        // Call the private product function to compute the product and display matrix C
        product();

        // Display matrix C in tabular form
        cout << "Product Matrix C (" << M << "x" << Q << "):" << endl;
        for (short unsigned int i = 0; i < M; ++i) {
            for (short unsigned int j = 0; j < Q; ++j) {
                cout << C[i][j] << "\t";
            }
            cout << endl;
        }
    }
};
```

```
int main() {
    // Create an object of the MATRIX class
    MATRIX matrixObject;

    return 0;
}
```

**Explanation:** This C++ program defines a class named MATRIX with the specified members. The default constructor accepts the matrix sizes and elements for matrices A and B, then calls the private member function product to compute the product of matrices A and B and stores the result in matrix C. Finally, it displays matrix C in tabular form. The main function creates an object of the MATRIX class to demonstrate its usage.

**Program 7:**

```
#include<iostream>
#include<cstring>
using namespace std;
class Node {
private:
    unsigned short roll;
    char name[31];
    float percent;
    Node* next;

public:
    Node(unsigned short r = 0, const char* n = "", float p = 0.0) : roll(r), percent(p), next(nullptr) {
        strncpy(name, n, 30);
        name[30] = '\0';
    }

    friend class LinkedList;
};

class LinkedList {
private:
    Node* head;

public:
    LinkedList() : head(nullptr) {}

    void insertNode(unsigned short r, const char* n, float p) {
        Node* newNode = new Node(r, n, p);
        newNode->next = head;
        head = newNode;
    }

    void displayList() {
        cout << "ROLL\tNAME\t\tPERCENT" << endl;
        Node* current = head;
```

```
        while (current != nullptr) {
            cout << current->roll << "\t" << current->name << "\t\t" << current->percent << endl;
            current = current->next;
        }
    }

    ~LinkedList() {
        Node* current = head;
        Node* nextNode;

        while (current != nullptr) {
            nextNode = current->next;
            delete current;
            current = nextNode;
        }
        head = nullptr;
    }
};

int main() {
    LinkedList linkedList;
    int choice;

    do {
        cout << "\nMenu:\n1. Insert a Node\n2. Display Linked List\n3. Exit\nEnter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                unsigned short roll;
                char name[31];
                float percent;
                cout << "Enter Roll: ";
                cin >> roll;
                cout << "Enter Name: ";
                cin.ignore(); // Ignore the newline character in the buffer
                cin.getline(name, 30);
                cout << "Enter Percent: ";
                cin >> percent;
                linkedList.insertNode(roll, name, percent);
                break;

            case 2:
                linkedList.displayList();
                break;

            case 3:
                cout << "Exiting program.\n";
```

```
            break;

        default:
            cout << "Invalid choice. Please enter a valid option.\n";
    }

} while (choice != 3);

return 0;
}
```

**Explanation:** This program provides a menu to insert nodes into a singly linked list and display the list. The user can choose options until they decide to exit the program. The linked list is defined with a Node class, and the main program uses a LinkedList class to manage the list.

**Program 8:**

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

class Node {
public:
    char data;
    Node* next;
};

class Stack {
private:
    Node* top;

public:
    Stack() {
        top = nullptr;
    }

    void push(char value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = top;
        top = newNode;
    }

    void pop() {
        if (top == nullptr) {
            cout << "Stack is empty. Cannot pop." << endl;
            return;
        }
        Node* temp = top;
        top = top->next;
        delete temp;
```

```
    }

    void display() {
        if (top == nullptr) {
            cout << "Stack is empty." << endl;
            return;
        }

        Node* current = top;
        while (current != nullptr) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }
};

int main() {
    Stack stack;

    int choice;
    char value;

    do {
        cout << "1. Push" << endl
             << "2. Pop" << endl
             << "3. Display" << endl
             << "4. Exit" << endl
             << "Enter your choice: ";

        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter character to push onto stack: ";
                cin >> value;
                stack.push(value);
                break;

            case 2:
                stack.pop();
                break;

            case 3:
                stack.display();
                break;

            case 4:
                cout << "Exiting program." << endl;
```

```
            break;

        default:
            cout << "Invalid choice. Please try again." << endl;
            break;
      }
    } while (choice != 4);

    return 0;
}
```

**Explanation:** The Node class contains a character data field and a pointer to the next node in the list. The Stack class includes methods for pushing a new node onto the stack, popping the top node from the stack, and displaying all nodes' data fields horizontally.
In the main function, a menu interface is created using a do-while loop, allowing users to choose from options such as push, pop, display, and exit. Based on user input, corresponding methods of the Stack class are invoked to perform the selected operation.
This program provides an interactive way to manipulate a stack data structure using linked list concepts in C++.

**Program 9:**

```
#include <iostream>
using namespace std;
class Node {
public:
   char data;
   Node* next;
};

class Queue {
private:
   Node* front;
   Node* rear;

public:
   Queue() {
      front = nullptr;
      rear = nullptr;
   }

   void append(char value) {
      Node* newNode = new Node();
      newNode->data = value;
      newNode->next = nullptr;

      if (rear == nullptr) {
         front = newNode;
         rear = newNode;
      } else {
```

```
            rear->next = newNode;
            rear = newNode;
        }
    }

    void deleteNode() {
        if (front == nullptr) {
            cout << "Queue is empty. Cannot delete." << endl;
            return;
        }

        Node* temp = front;

        if (front == rear) {
            front = nullptr;
            rear = nullptr;
        } else {
            front = front->next;
        }

        delete temp;
    }

    void display() {
        if (front == nullptr) {
            cout << "Queue is empty." << endl;
            return;
        }

        Node* current = front;

        while (current != nullptr) {
            cout << current->data << " ";
            current = current->next;
         }

         cout << endl;
    }
};

int main() {
    Queue queue;

    int choice;
    char value;

    do {
        cout << "1. Append" << endl
             << "2. Delete" << endl
```

```
            << "3. Display" << endl
            << "4. Exit" << endl
            << "Enter your choice: ";

        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter character to append into the queue: "; cin >> value; queue.append(value);
break;

            case 2:
                queue.deleteNode(); break;

            case 3:
                queue.display(); break;

            case 4:
                cout<<"Exiting program."<<endl;break;

            default:cout<<"Invalid choice. Please try again."<<endl;break;
          }
      } while(choice != 4);

      return 0;
}
```

**Explanation:** The provided C++ program demonstrates a menu-driven implementation of a queue using a singly linked list. It comprises two classes: Node to represent individual elements in the queue, and Queue to manage the queue operations.
The Node class contains a character data field and a pointer to the next node in the list. The Queue class includes methods for appending a new node into the queue, deleting the first node in the queue, and displaying all nodes' data fields horizontally.
In the main function, a menu interface is created using a do-while loop, allowing users to choose from options such as append, delete, display, and exit. Based on user input, corresponding methods of the Queue class are invoked to perform the selected operation.
This program provides an interactive way to manipulate a queue data structure using linked list concepts in C++.

**Program 10:**

```
#include<iostream>
#include<fstream>
using namespace std;
class BILL {
private:
   unsigned short item_code;
   char item_name[30];
   float unit_price;
   unsigned short quantity;
```

```
    float total;

public:
    void get_data() {
        cout << "Enter Item Code: ";
        cin >> item_code;

        cout << "Enter Item Name (up to 30 characters): ";
        cin.ignore(); // Ignore the newline character in the buffer
        cin.getline(item_name, 30);

        cout << "Enter Unit Price: ";
        cin >> unit_price;

        cout << "Enter Quantity: ";
        cin >> quantity;

        // Compute total
        total = quantity * unit_price;
    }

    void put_data() const {
        cout << item_code << "\t\t" << item_name << "\t\t" << unit_price << "\t\t" << quantity << "\t\t" << total
<< endl;
    }

    void write_to_file() const {
        ofstream file("market.data", ios::binary | ios::app);
        if (file.is_open()) {
            file.write(reinterpret_cast<const char*>(this), sizeof(BILL));
            file.close();
        } else {
            cout << "Error: Unable to open file for writing.\n";
        }
    }

    void read_from_file() const {
        ifstream file("market.data", ios::binary);
        if (file.is_open()) {
            BILL temp;
            cout << "ITEM CODE\tITEM NAME\tUNIT-PRICE\tQUANTITY\tTOTAL\n";
            while (file.read(reinterpret_cast<char*>(&temp), sizeof(BILL))) {
                temp.put_data();
            }
            file.close();
        } else {
            cout << "Error: Unable to open file for reading.\n";
        }
    }
```

```
};

int main() {
    BILL bill;
    int choice;

    do {
        cout << "\nMenu:\n1. Add Item to market.data\n2. Display market.data\n3. Exit\nEnter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                bill.get_data();
                bill.write_to_file();
                cout << "Item added to market.data.\n";
                break;

            case 2:
                bill.read_from_file();
                break;

            case 3:
                cout << "Exiting program.\n";
                break;

            default:
                cout << "Invalid choice. Please enter a valid option.\n";
        }

    } while (choice != 3);

    return 0;
}
```

**Explanation:** BILL Class: Models a billing item with properties like code, name, price, quantity, and total.
File Handling: Uses fstream to read and write BILL objects to a binary file for efficient storage.
User Interaction: Provides a menu-driven interface for adding items, displaying file contents, and exiting.
Encapsulation: The BILL class encapsulates item data and related operations.
Error Handling: Includes basic error checks for file operations.
This program can be used to create and manage simple bill records, demonstrating object-oriented programming and file I/O concepts in C++.