Lab 8(Understanding the Concept of Console and File Input/Output)

1. **Write a program to demonstrate the use of different `ios` flags and functions to format the output. Create a program to generate the bill invoice of a department store by using different formatting.**

```
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>

// Define a struct to represent an item in the invoice
struct InvoiceItem {
    std::string name;
    int quantity;
    double price;
};

// Function to generate and display the bill invoice
void generateInvoice(const std::vector<InvoiceItem>& items) {
    // Set up the header
     std::cout << std::left << std::setw(20) << "Item Name" << std::setw(10)
<< "Quantity" << std::setw(10) << "Price" << std::setw(15) << "Total" <<
std::endl;
       std::cout << std::setfill('-') << std::setw(55) << "" << std::setfill(' ') <<
std::endl;

    // Calculate and display the items and totals
    double totalAmount = 0.0;
    for (const auto& item : items) {
        double itemTotal = item.quantity * item.price;
         std::cout << std::left << std::setw(20) << item.name << std::setw(10)
<< item.quantity << std::fixed << std::setprecision(2) << std::setw(10) <<
item.price << std::setw(15) << itemTotal << std::endl;
        totalAmount += itemTotal;
    }

    // Display the total amount
       std::cout << std::setfill('-') << std::setw(55) << "" << std::setfill(' ') <<
std::endl;
           std::cout << std::setw(45) << "Total Amount" << std::fixed <<
std::setprecision(2) << std::setw(10) << totalAmount << std::endl;
}
```

```
int main() {
    // Create sample invoice items
    std::vector<InvoiceItem> items;
    items.push_back({"Item 1", 3, 10.50});
    items.push_back({"Item 2", 2, 25.75});
    items.push_back({"Item 3", 5, 5.99});

    // Display the bill invoice with different formatting
    std::cout << "Default Formatting:" << std::endl;
    generateInvoice(items);

    std::cout << "\nUsing Fixed Notation:" << std::endl;
    std::cout << std::fixed;
    generateInvoice(items);

    std::cout << "\nUsing Scientific Notation:" << std::endl;
    std::cout << std::scientific;
    generateInvoice(items);

    return 0;
}
```

Output:

```
Default Formatting:
Item Name          Quantity  Price     Total
----------------------------------------------------
Item 1             3         10.50     31.50
Item 2             2         25.75     51.50
Item 3             5         5.99      29.95
----------------------------------------------------
Total Amount                           112.95

Using Fixed Notation:
Item Name          Quantity  Price     Total
----------------------------------------------------
Item 1             3         10.50     31.50
Item 2             2         25.75     51.50
Item 3             5         5.99      29.95
----------------------------------------------------
Total Amount                           112.95
```

```
Using Scientific Notation:
Item Name          Quantity  Price     Total
----------------------------------------------------
Item 1             3         10.50     31.50
Item 2             2         25.75     51.50
Item 3             5         5.99      29.95
----------------------------------------------------
Total Amount                           112.95
```

2. Write a program to create a user-defined manipulator that will format the output by setting the width, precision, and fill character at the same time by passing arguments.

```
#include <iostream>
#include <iomanip>

// User-defined manipulator
struct FormatManipulator {
    int width;
    int precision;
    char fill;

    FormatManipulator(int w, int p, char f) : width(w), precision(p), fill(f) {}
};

// Overload the << operator for the user-defined manipulator
std::ostream& operator<<(std::ostream& os, const FormatManipulator&
manipulator) {
    os.width(manipulator.width);
    os.precision(manipulator.precision);
    os.fill(manipulator.fill);
    return os;
}

// Example usage
int main() {
    double number = 3.14159;

    std::cout << "Default output: " << number << std::endl;

    std::cout << "Formatted output: "
              << FormatManipulator(10, 4, '*') << number << std::endl;
```

```cpp
    return 0;
}
```
Output:

```
Default output: 3.14159
Formatted output: *****3.142
```

3. Write a program to overload stream operators to read a complex number and display the complex number in a+ib format.

```cpp
#include <iostream>

class Complex {
private:
    double real;
    double imaginary;

public:
    Complex(double real = 0.0, double imaginary = 0.0)
        : real(real), imaginary(imaginary) {}

    friend std::istream& operator>>(std::istream& in, Complex& complex);
      friend std::ostream& operator<<(std::ostream& out, const Complex&
complex);
};

std::istream& operator>>(std::istream& in, Complex& complex) {
    std::cout << "Enter the real part: ";
    in >> complex.real;

    std::cout << "Enter the imaginary part: ";
    in >> complex.imaginary;

    return in;
}

std::ostream& operator<<(std::ostream& out, const Complex& complex) {
    out << complex.real;
    if (complex.imaginary >= 0)
        out << "+";
    out << complex.imaginary << "i";

    return out;
}

int main() {
    Complex c;

    std::cout << "Enter a complex number:" << std::endl;
    std::cin >> c;

    std::cout << "Complex number in a+ib format: " << c << std::endl;

    return 0;
}
```

Output:

```
Enter a complex number:
Enter the real part: 1
Enter the imaginary part: 2
Complex number in a+ib format: 1+2i
```

4. Write a program that stores the information about students (name, student id, department, and address) in a structure and then transfers the information to a file in your directory. Finally, retrieve the information from your file and print it in the proper format on your output screen.

```
#include <iostream>
#include <fstream>
#include <string>

// Define a structure to store student information
struct Student {
    std::string name;
    int studentId;
    std::string department;
    std::string address;
};

// Function to write student information to a file
void writeStudentToFile(const Student& student, const std::string& filename) {
     std::ofstream outFile(filename, std::ios::app); // Open the file in append mode
    if (!outFile) {
        std::cerr << "Error opening the file for writing." << std::endl;
        return;
    }

    // Write student information to the file
    outFile << "Name: " << student.name << std::endl;
    outFile << "Student ID: " << student.studentId << std::endl;
    outFile << "Department: " << student.department << std::endl;
    outFile << "Address: " << student.address << std::endl;
    outFile << std::endl;

    outFile.close();
}

// Function to read and print student information from a file
void readStudentFromFile(const std::string& filename) {
    std::ifstream inFile(filename);
    if (!inFile) {
        std::cerr << "Error opening the file for reading." << std::endl;
        return;
    }

    std::string line;
    while (std::getline(inFile, line)) {
        std::cout << line << std::endl;
    }

    inFile.close();
}

int main() {
    // Create and initialize a student structure
     Student student1 = {"John Doe", 101, "Computer Science", "123 Main
St"};
      Student student2 = {"Jane Smith", 102, "Electrical Engineering", "456
Elm St"};

    // Write student information to a file
    writeStudentToFile(student1, "student_info.txt");
    writeStudentToFile(student2, "student_info.txt");

    // Read and print student information from the file
    std::cout << "Student Information from File:" << std::endl;
    readStudentFromFile("student_info.txt");

    return 0;
}
```

Output:

```
Name: Ram Joshi
Student ID: 101
Department: Computer Science
Address: 123 Main St

Name: Hari Yadav
Student ID: 102
Department: Electrical Engineering
Address: 456 Elm St

Name: Ram Joshi
Student ID: 101
Department: Computer Science
Address: 123 Main St

Name: Hari Yadav
Student ID: 102
Department: Electrical Engineering
Address: 456 Elm St
```

5. Write a program for transaction processing that write and read object randomly to and from a random access file so that user can add, update, delete and display the account information (account-number, last-name, first-name, total-balance).

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>

// Define the structure for account information
struct Account {
    int accountNumber;
    std::string lastName;
    std::string firstName;
    double totalBalance;
};
```

```
// Function to display an account's details
void displayAccount(const Account& account) {
        std::cout << "Account Number: " << account.accountNumber <<
std::endl;
    std::cout << "Last Name: " << account.lastName << std::endl;
    std::cout << "First Name: " << account.firstName << std::endl;
    std::cout << "Total Balance: $" << account.totalBalance << std::endl;
    std::cout << "-----------------------" << std::endl;
}

// Function to add a new account to the file
void addAccount(std::fstream& file, const Account& account) {
    file.write(reinterpret_cast<const char*>(&account), sizeof(Account));
}

// Function to update an account in the file
void updateAccount(std::fstream& file, int accountNumber, const
Account& updatedAccount) {
    Account account;
    while (file.read(reinterpret_cast<char*>(&account), sizeof(Account))) {
        if (account.accountNumber == accountNumber) {
                        file.seekp(-static_cast<std::streamoff>(sizeof(Account)),
std::ios::cur);
                    file.write(reinterpret_cast<const char*>(&updatedAccount),
sizeof(Account));
            break;
        }
    }
}

// Function to delete an account from the file
void deleteAccount(std::fstream& file, int accountNumber) {
    std::fstream tempFile("temp.txt", std::ios::out | std::ios::binary);
    if (!tempFile) {
        std::cerr << "Error creating temporary file." << std::endl;
        return;
    }

    Account account;
    while (file.read(reinterpret_cast<char*>(&account), sizeof(Account))) {
        if (account.accountNumber != accountNumber) {
```

```
                    tempFile.write(reinterpret_cast<const char*>(&account),
sizeof(Account));
        }
    }

    file.close();
    tempFile.close();
    remove("accounts.txt");
    rename("temp.txt", "accounts.txt");

    file.open("accounts.txt", std::ios::in | std::ios::out | std::ios::binary);
}

// Function to display all accounts in the file
void displayAllAccounts(std::fstream& file) {
    file.seekg(0, std::ios::beg);
    Account account;
    while (file.read(reinterpret_cast<char*>(&account), sizeof(Account))) {
        displayAccount(account);
    }
}

int main() {
            std::fstream   file("accounts.txt",   std::ios::in   |   std::ios::out   |
std::ios::binary);
    if (!file) {
        std::cerr << "Error opening the file." << std::endl;
        return 1;
    }

    int choice;
    do {
            std::cout << "1. Add Account\n2. Update Account\n3. Delete
Account\n4. Display All Accounts\n5. Exit\n";
        std::cout << "Enter your choice: ";
        std::cin >> choice;

        switch (choice) {
            case 1:
                {
                    Account newAccount;
                    std::cout << "Enter Account Number: ";
                    std::cin >> newAccount.accountNumber;
                    std::cout << "Enter Last Name: ";
                    std::cin >> newAccount.lastName;
                    std::cout << "Enter First Name: ";
                    std::cin >> newAccount.firstName;
                    std::cout << "Enter Total Balance: ";
                    std::cin >> newAccount.totalBalance;
                    addAccount(file, newAccount);
                    std::cout << "Account added successfully." << std::endl;
                }
                break;
            case 2:
                {
                    int accountNumber;
                    std::cout << "Enter Account Number to Update: ";
                    std::cin >> accountNumber;
                    Account updatedAccount;
                            std::cout << "Enter Updated Account Information:" <<
std::endl;
                    std::cout << "Enter Last Name: ";
                    std::cin >> updatedAccount.lastName;
                    std::cout << "Enter First Name: ";
                    std::cin >> updatedAccount.firstName;
                    std::cout << "Enter Total Balance: ";
                    std::cin >> updatedAccount.totalBalance;
                    updateAccount(file, accountNumber, updatedAccount);
                    std::cout << "Account updated successfully." << std::endl;
                }
                break;
            case 3:
                {
                    int accountNumber;
                    std::cout << "Enter Account Number to Delete: ";
                    std::cin >> accountNumber;
                    deleteAccount(file, accountNumber);
                    std::cout << "Account deleted successfully." << std::endl;
                }
                break;
            case 4:
                std::cout << "All Accounts:" << std::endl;
                displayAllAccounts(file);
                break;
```

```
            case 5:
                std::cout << "Exiting..." << std::endl;
                break;
            default:
                std::cout << "Invalid choice. Please try again." << std::endl;
        }
    } while (choice != 5);

    file.close();
    return 0;
}
```

Output

```
1. Add Account
2. Update Account
3. Delete Account
4. Display All Accounts
5. Exit
Enter your choice: 4
All Accounts:
Account Number: 123
Last Name: Frank
First Name: John
Total Balance: $3000
-----------------------
1. Add Account
2. Update Account
3. Delete Account
4. Display All Accounts
5. Exit
Enter your choice:
```