Experiment No- **Date-**

Aim – To study File Processing

Theory –

FILE PROCESSING

File processing in computer science refers to the handling and manipulation of data stored in files by a computer program. Files provide a means to store data persistently, allowing information to be saved and retrieved later. In file processing, a program performs operations such as creating, opening, reading, writing, and closing files to manage data.

Types of Files

1. **Text Files**: Store data in human-readable format, using characters. Examples include .txt and .csv files. Data is stored as a sequence of lines, where each line may end with a newline character.
2. **Binary Files**: Store data in a machine-readable format (binary form), not easily readable by humans. Examples include image files (.png, .jpg), executable files (.exe), and compiled program files.

File Operations

- **Creating a File**: Involves allocating space for the file in the storage system. This operation typically includes specifying the file name and format.
- **Opening a File**: Prepares the file for reading, writing, or updating. A file must be opened before any operation can be performed on it. Modes for opening a file include:
  - Read (r): Open an existing file for reading.
  - Write (w): Create a new file or overwrite an existing file.
  - Append (a): Open a file and write data at the end.
  - Update (r+, w+, a+): Open a file for both reading and writing.
- **Reading from a File**: Involves accessing the file's content and retrieving data. Methods vary depending on the type of file (text or binary).
- **Writing to a File**: Involves adding data to the file, either by creating a new file, overwriting an existing file, or appending data to the end.
- **Closing a File**: Finalizes the file operations, ensuring all data is saved, and resources are freed.

File Access Methods

1. **Sequential Access**: Data is accessed in a specific order, typically from the beginning to the end. It's suitable for reading files line by line or processing data in a fixed sequence.
2. **Random Access**: Data can be accessed directly at any position in the file, without reading through preceding data. It allows updating or retrieving specific parts of the file efficiently.

File Handling in C++

C++ provides a set of library functions in the <fstream> header for file handling:

- **ifstream**: Used for reading from files.
- **ofstream**: Used for writing to files.
- **fstream**: Supports both reading and writing.

The <fstream> library in C++ provides various file processing functions to perform input and output operations on files. The main classes used are ifstream (input file stream), ofstream (output file stream), and fstream (file stream for both input and output). Here are some key file processing functions in <fstream>:

1. open()

- **Opens a file and associates it with a stream object.**
- **Syntax: fileStream.open("filename", mode);**
- **Modes include:**
  - **ios::in: Open for reading.**
  - **ios::out: Open for writing.**
  - **ios::app: Append to the end of the file.**
  - **ios::binary: Open in binary mode.**
  - **ios::ate: Open and seek to the end.**
  - **ios::trunc: Truncate the file if it exists.**

2. close()

- **Closes the file associated with the stream, releasing the resources.**
- **Syntax: fileStream.close();**

3. is_open()

- **Checks if a file is open.**
- **Syntax: fileStream.is_open();**
- **Returns true if the file is open, otherwise false.**

4. getline()

- **Reads a line from the file into a string.**
- **Syntax: getline(fileStream, stringVariable);**

5. write()

- **Writes data to a file in binary mode.**
- **Syntax: fileStream.write(reinterpret_cast<const char*>(&variable), sizeof(variable));**

6. read()

- **Reads data from a file in binary mode.**
- **Syntax: fileStream.read(reinterpret_cast<char*>(&variable), sizeof(variable));**

7. seekg() and seekp()

- **seekg(): Moves the "get" (read) position in a file.**
- **seekp(): Moves the "put" (write) position in a file.**
- **Syntax: fileStream.seekg(offset, direction);**
  - **direction can be ios::beg (beginning), ios::cur (current position), or ios::end (end).**

8. tellg() and tellp()

- **tellg(): Returns the current "get" (read) position.**
- **tellp(): Returns the current "put" (write) position.**
- **Syntax: fileStream.tellg();**

9. eof()

- **Checks if the end of the file has been reached.**
- **Syntax: fileStream.eof();**

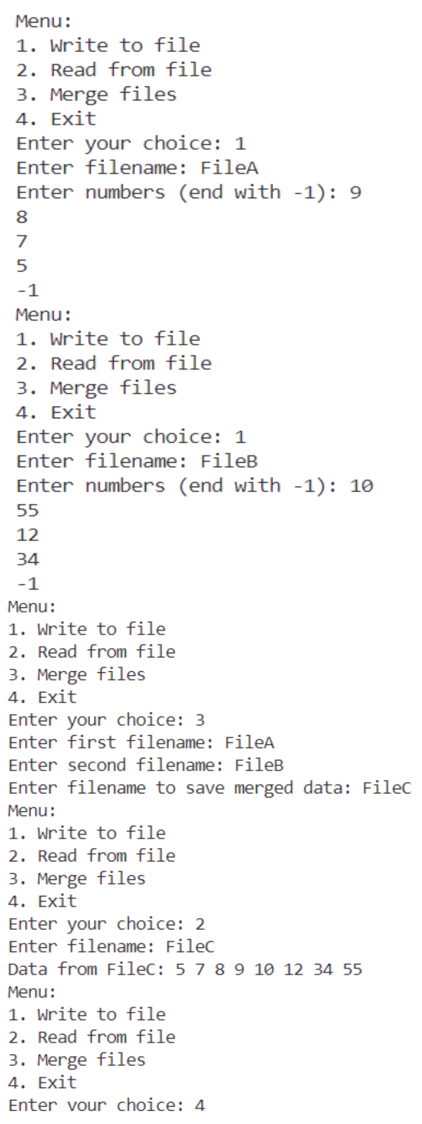
‘

[A] Write a C++ program to insert 5 elements in first file and 3 elements in second file. Merge the contents of both files into third file into ascending order.

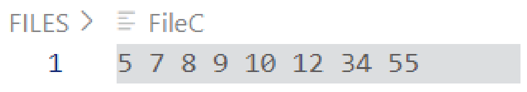
Program-

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

using namespace std;

const int MAX_SIZE = 1000;

void writeToFile(const string& filename, int data[], int size) {
   ofstream outFile(filename);
   for (int i = 0; i < size; ++i) {
      outFile << data[i] << " ";
   }
   outFile.close();
}

int readFromFile(const string& filename, int data[]) {
   ifstream inFile(filename);
   int num, size = 0;
   while (inFile >> num && size < MAX_SIZE) {
      data[size++] = num;
   }
   inFile.close();
   return size;
}

void displayMenu() {
   cout << "Menu:\n";
   cout << "1. Write to file\n";
   cout << "2. Read from file\n";
   cout << "3. Merge files\n";
```

OUTPUT –



```
Menu:
1. Write to file
2. Read from file
3. Merge files
4. Exit
Enter your choice: 1
Enter filename: FileA
Enter numbers (end with -1): 9
8
7
5
-1
Menu:
1. Write to file
2. Read from file
3. Merge files
4. Exit
Enter your choice: 1
Enter filename: FileB
Enter numbers (end with -1): 10
55
12
34
-1
Menu:
1. Write to file
2. Read from file
3. Merge files
4. Exit
Enter your choice: 3
Enter first filename: FileA
Enter second filename: FileB
Enter filename to save merged data: FileC
Menu:
1. Write to file
2. Read from file
3. Merge files
4. Exit
Enter your choice: 2
Enter filename: FileC
Data from FileC: 5 7 8 9 10 12 34 55
Menu:
1. Write to file
2. Read from file
3. Merge files
4. Exit
Enter your choice: 4
```



```
FILES > FileC
1    5 7 8 9 10 12 34 55
```

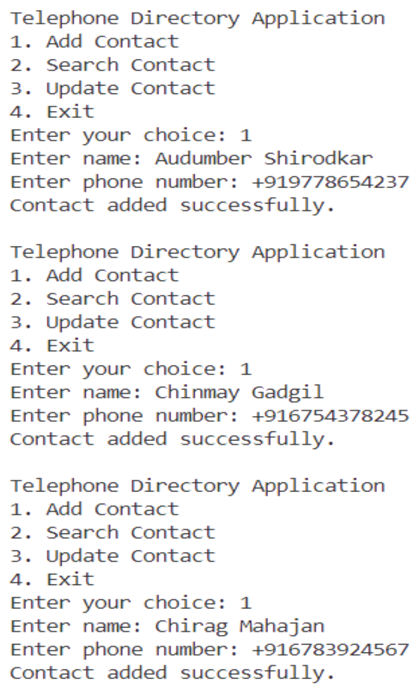
```
    cout << "4. Exit\n";
    cout << "Enter your choice: ";
}

int main() {
    int file1Data[MAX_SIZE], file2Data[MAX_SIZE],
mergedData[2 * MAX_SIZE];
    string filename;
    int choice, size1, size2, mergedSize;

    while (true) {
        displayMenu();
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter filename: ";
                cin >> filename;
                cout << "Enter numbers (end with -1): ";
                size1 = 0;
                int num;
                while (cin >> num && num != -1 && size1 <
MAX_SIZE) {
                    file1Data[size1++] = num;
                }
                writeToFile(filename, file1Data, size1);
                break;

            case 2:
                cout << "Enter filename: ";
                cin >> filename;
                size1 = readFromFile(filename, file1Data);
                cout << "Data from " << filename << ": ";
                for (int i = 0; i < size1; ++i) {
                    cout << file1Data[i] << " ";
                }
                cout << endl;
                break;

            case 3:
                cout << "Enter first filename: ";
                cin >> filename;
                size1 = readFromFile(filename, file1Data);
                cout << "Enter second filename: ";
                cin >> filename;
                size2 = readFromFile(filename, file2Data);

                mergedSize = size1 + size2;
                copy(file1Data, file1Data + size1, mergedData);
                copy(file2Data, file2Data + size2, mergedData +
size1);
                sort(mergedData, mergedData + mergedSize);

                cout << "Enter filename to save merged data: ";
                cin >> filename;
                writeToFile(filename, mergedData, mergedSize);
                break;

            case 4:
                return 0;

            default:
                cout << "Invalid choice. Please try again.\n";
        }
    }

    return 0;
}
```

[B] Write a C++ program to simulate a telephone directory application. Program should prompt user to enter name and telephone number of users. Also the program should allow the user to search and update the telephone number of a specific user depending upon the name entered.

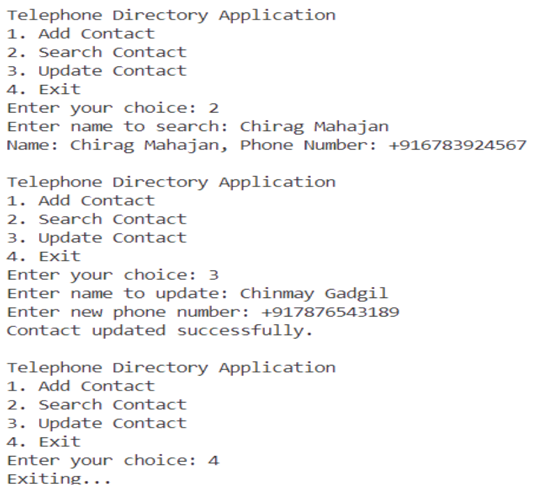
Program –

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

using namespace std;

struct Contact {
   string name;
   string phoneNumber;
};

class TelephoneDirectory {
private:
   const string filename = "contacts.txt";

   void saveContact(const Contact& contact) {
      ofstream file(filename, ios::app);
      if (file.is_open()) {
        file << contact.name << "," << contact.phoneNumber
<< "\n";
         file.close();
      } else {
         cout << "Unable to open file for writing.\n";
      }
   }

   bool loadContacts(vector<Contact>& contacts) {
      ifstream file(filename);
      if (file.is_open()) {
         string line;
         while (getline(file, line)) {
            size_t pos = line.find(',');
            if (pos != string::npos) {
               Contact contact;
               contact.name = line.substr(0, pos);
               contact.phoneNumber = line.substr(pos + 1);
               contacts.push_back(contact);
            }
         }
         file.close();
         return true;
      } else {
         cout << "Unable to open file for reading.\n";
         return false;
      }
   }
```

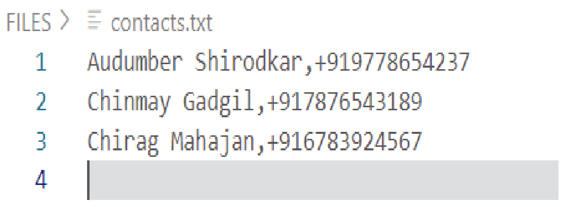
Output –



```
Telephone Directory Application
1. Add Contact
2. Search Contact
3. Update Contact
4. Exit
Enter your choice: 1
Enter name: Audumber Shirodkar
Enter phone number: +919778654237
Contact added successfully.

Telephone Directory Application
1. Add Contact
2. Search Contact
3. Update Contact
4. Exit
Enter your choice: 1
Enter name: Chinmay Gadgil
Enter phone number: +916754378245
Contact added successfully.

Telephone Directory Application
1. Add Contact
2. Search Contact
3. Update Contact
4. Exit
Enter your choice: 1
Enter name: Chirag Mahajan
Enter phone number: +916783924567
Contact added successfully.
```



```
Telephone Directory Application
1. Add Contact
2. Search Contact
3. Update Contact
4. Exit
Enter your choice: 2
Enter name to search: Chirag Mahajan
Name: Chirag Mahajan, Phone Number: +916783924567

Telephone Directory Application
1. Add Contact
2. Search Contact
3. Update Contact
4. Exit
Enter your choice: 3
Enter name to update: Chinmay Gadgil
Enter new phone number: +917876543189
Contact updated successfully.

Telephone Directory Application
1. Add Contact
2. Search Contact
3. Update Contact
4. Exit
Enter your choice: 4
Exiting...
```



FILES > contacts.txt

```
Audumber Shirodkar,+919778654237
Chinmay Gadgil,+917876543189
Chirag Mahajan,+916783924567

```

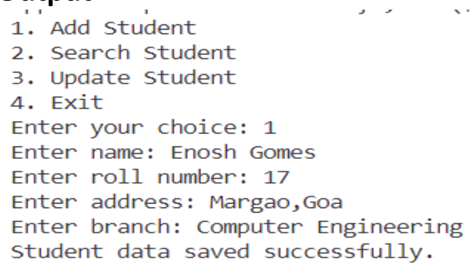
```
  void saveAllContacts(const vector<Contact>& contacts)
{
     ofstream file(filename);
     if (file.is_open()) {
        for (const auto& contact : contacts) {
                    file << contact.name << "," <<
contact.phoneNumber << "\n";
        }
        file.close();
     } else {
        cout << "Unable to open file for writing.\n";
     }
  }

public:
   void addContact(const string& name, const string&
phoneNumber) {
     Contact contact = {name, phoneNumber};
     saveContact(contact);
     cout << "Contact added successfully.\n";
  }

   bool searchContact(const string& name, Contact&
contact) {
     vector<Contact> contacts;
     if (loadContacts(contacts)) {
        for (const auto& c : contacts) {
           if (c.name == name) {
              contact = c;
              return true;
           }
        }
     }
     return false;
  }

  bool updateContact(const string& name, const string&
newPhoneNumber) {
     vector<Contact> contacts;
     if (loadContacts(contacts)) {
        for (auto& c : contacts) {
           if (c.name == name) {
              c.phoneNumber = newPhoneNumber;
              saveAllContacts(contacts);
              return true;
           }
        }
     }
     return false;
  }
};

int main() {
  TelephoneDirectory directory;
  int choice;
  Contact contact;
  string name, phoneNumber;

  while (true) {
     cout << "\nTelephone Directory Application\n";
     cout << "1. Add Contact\n";
     cout << "2. Search Contact\n";
     cout << "3. Update Contact\n";
     cout << "4. Exit\n";
     cout << "Enter your choice: ";
     cin >> choice;
      cin.ignore(numeric_limits<streamsize>::max(), '\n');
// Clear input buffer after numeric input

     switch (choice) {
        case 1:
           cout << "Enter name: ";
           getline(cin, name);
           cout << "Enter phone number: ";
           getline(cin, phoneNumber);
           directory.addContact(name, phoneNumber);
           break;
        case 2:
           cout << "Enter name to search: ";
           getline(cin, name);

           if (directory.searchContact(name, contact)) {
              cout << "Name: " << contact.name << ", Phone
Number: " << contact.phoneNumber << "\n";
           } else {
              cout << "Contact not found.\n";
           }
           break;
        case 3:
           cout << "Enter name to update: ";
           getline(cin, name);
           cout << "Enter new phone number: ";
           getline(cin, phoneNumber);
                 if (directory.updateContact(name,
phoneNumber)) {
              cout << "Contact updated successfully.\n";
           } else {
              cout << "Contact not found.\n";
           }
           break;
        case 4:
           cout << "Exiting...\n";
           return 0;
        default:
           cout << "Invalid choice. Please try again.\n";
     }
  }
Return 0 ;
}
```

[C] Write a C++ program to create a student's database application using "files". Create a unique file for each student depending upon the student name entered. Store the student data like name, roll no, address, and branch into the file. Allow the user to search and update all the student details depending upon the entered roll-no and display the details.

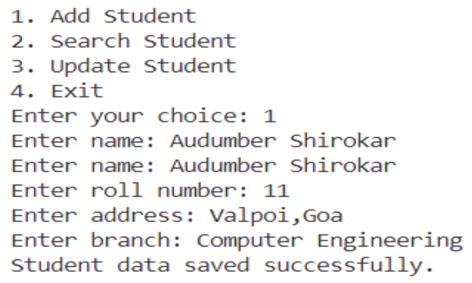
Program –

```
#include <iostream>
#include <fstream>
#include <string>
#include <limits>  // For numeric_limits

using namespace std;

class Student {
public:
   string name;
   int rollNo;
   string address;
   string branch;

   void getData() {
      cout << "Enter name: ";
       cin.ignore();  // Ignore buffer before getline after using
cin
      getline(cin, name);
      cout << "Enter roll number: ";
      cin >> rollNo;
      cin.ignore();  // Ignore buffer before getline
      cout << "Enter address: ";
      getline(cin, address);
      cout << "Enter branch: ";
      getline(cin, branch);
   }

   void displayData() {
      cout << "Name: " << name << endl;
      cout << "Roll Number: " << rollNo << endl;
      cout << "Address: " << address << endl;
      cout << "Branch: " << branch << endl;
   }

   string getFileName() {
        return to_string(rollNo) + ".txt";  // File now named
based on roll number
   }

   void saveToFile() {
      ofstream file(getFileName());
      if (file) {
         file << name << endl;
         file << rollNo << endl;
         file << address << endl;
         file << branch << endl;
         file.close();
           cout << "Student data saved successfully.\n\n" <<
endl;
      } else {
```

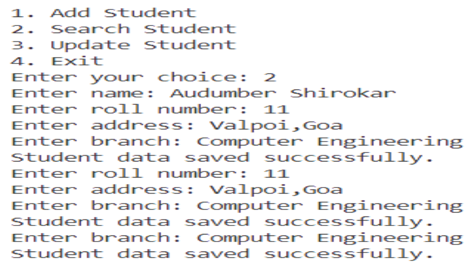
Output –



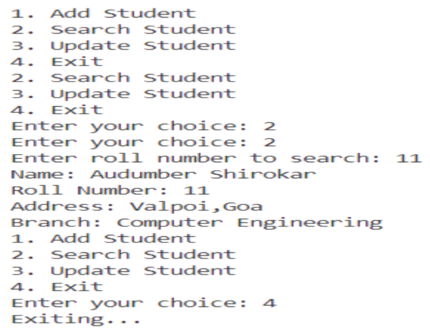
```
1. Add Student
2. Search Student
3. Update Student
4. Exit
Enter your choice: 1
Enter name: Enosh Gomes
Enter roll number: 17
Enter address: Margao,Goa
Enter branch: Computer Engineering
Student data saved successfully.
```



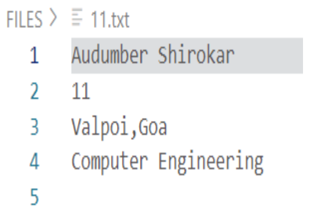
```
1. Add Student
2. Search Student
3. Update Student
4. Exit
Enter your choice: 1
Enter name: Audumber Shirokar
Enter name: Audumber Shirokar
Enter roll number: 11
Enter address: Valpoi,Goa
Enter branch: Computer Engineering
Student data saved successfully.
```



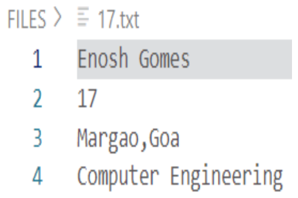
```
1. Add Student
2. Search Student
3. Update Student
4. Exit
Enter your choice: 2
Enter name: Audumber Shirokar
Enter roll number: 11
Enter address: Valpoi,Goa
Enter branch: Computer Engineering
Student data saved successfully.
Enter roll number: 11
Enter address: Valpoi,Goa
Enter branch: Computer Engineering
Student data saved successfully.
Enter branch: Computer Engineering
Student data saved successfully.
```



```
1. Add Student
2. Search Student
3. Update Student
4. Exit
2. Search Student
3. Update Student
4. Exit
Enter your choice: 2
Enter your choice: 2
Enter roll number to search: 11
Name: Audumber Shirokar
Roll Number: 11
Address: Valpoi,Goa
Branch: Computer Engineering
1. Add Student
2. Search Student
3. Update Student
4. Exit
Enter your choice: 4
Exiting...
```



```
FILES > 11.txt
1  Audumber Shirokar
2  11
3  Valpoi,Goa
4  Computer Engineering
5
```



```
FILES > 17.txt
1  Enosh Gomes
2  17
3  Margao,Goa
4  Computer Engineering
```

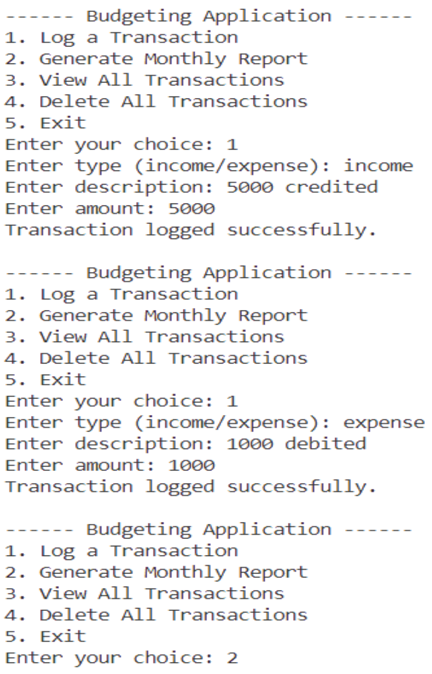
```
         cout << "Error saving student data!" << endl;
      }
   }

   void loadFromFile(string fileName) {
      ifstream file(fileName);
      if (file.is_open()) {
         getline(file, name);
         string rollNoStr;
         getline(file, rollNoStr);
         rollNo = stoi(rollNoStr);
         getline(file, address);
         getline(file, branch);
         file.close();
      } else {
         cout << "File not found!" << endl;
      }
   }
};

void updateStudentDetails(int rollNo) {
   string fileName = to_string(rollNo) + ".txt";

   Student student;
   student.loadFromFile(fileName);

   if (student.rollNo == rollNo) {
      cout << "Current details:" << endl;
      student.displayData();
      cout << "Enter new details:" << endl;
      student.getData();
      student.saveToFile();
      cout << "Details updated successfully." << endl;
   } else {
      cout << "Roll number does not match!" << endl;
   }
}

void searchStudentDetails(int rollNo) {
   string fileName = to_string(rollNo) + ".txt";

   Student student;
   student.loadFromFile(fileName);

   if (student.rollNo == rollNo) {
      student.displayData();
   } else {
      cout << "Roll number does not match!" << endl;
   }
}

int main() {
   int choice;
   do {
      cout << "1. Add Student" << endl;
      cout << "2. Search Student" << endl;
      cout << "3. Update Student" << endl;
      cout << "4. Exit" << endl;
      cout << "Enter your choice: ";
      cin >> choice;

      switch (choice) {
         case 1: {
            Student student;
            student.getData();
            student.saveToFile();
            break;
         }
         case 2: {
            int rollNo;
            cout << "Enter roll number to search: ";
            cin >> rollNo;
            searchStudentDetails(rollNo);
            break;
         }
         case 3: {
            int rollNo;
            cout << "Enter roll number to update: ";
            cin >> rollNo;
            updateStudentDetails(rollNo);
            break;
         }
         case 4:
            cout << "Exiting..." << endl;
            break;
         default:
            cout << "Invalid choice!" << endl;
      }
   } while (choice != 4);

   return 0;
}
```

D} Write a C++ program to a budgeting application that allows user to log income and expense.Store transactions in a CSV file and generate monthy reports.

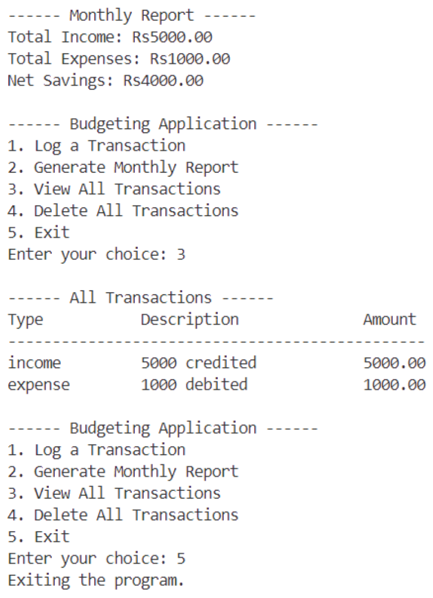
Program –

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

// Function prototypes with definitions
void displayMenu() {
   cout << "\n------ Budgeting Application ------\n";
   cout << "1. Log a Transaction\n";
   cout << "2. Generate Monthly Report\n";
   cout << "3. View All Transactions\n";
   cout << "4. Delete All Transactions\n";
   cout << "5. Exit\n";
}

void logTransaction() {
   ofstream file("transactions.csv", ios::app);
   if (!file) {
      cerr << "Error opening file." << endl;
      return;
   }

   string type, description;
   double amount;
   cout << "Enter type (income/expense): ";
   cin >> type;
   cout << "Enter description: ";
   cin.ignore();
   getline(cin, description);
   cout << "Enter amount: ";
   cin >> amount;

   file << type << "," << description << "," << amount << "\n";
   file.close();

   cout << "Transaction logged successfully." << endl;
}

void generateMonthlyReport() {
   ifstream file("transactions.csv");
   if (!file) {
      cerr << "Error opening file." << endl;
      return;
   }

   string line, type, description;
   double amount, totalIncome = 0, totalExpense = 0;

   while (getline(file, line)) {
      size_t pos1 = line.find(',');
      size_t pos2 = line.rfind(',');

      if (pos1 != string::npos && pos2 != string::npos && pos1
!= pos2) {
```

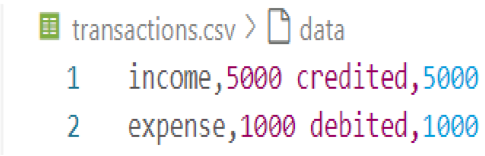
OUTPUT –



```
------ Budgeting Application ------
1. Log a Transaction
2. Generate Monthly Report
3. View All Transactions
4. Delete All Transactions
5. Exit
Enter your choice: 1
Enter type (income/expense): income
Enter description: 5000 credited
Enter amount: 5000
Transaction logged successfully.

------ Budgeting Application ------
1. Log a Transaction
2. Generate Monthly Report
3. View All Transactions
4. Delete All Transactions
5. Exit
Enter your choice: 1
Enter type (income/expense): expense
Enter description: 1000 debited
Enter amount: 1000
Transaction logged successfully.

------ Budgeting Application ------
1. Log a Transaction
2. Generate Monthly Report
3. View All Transactions
4. Delete All Transactions
5. Exit
Enter your choice: 2
```



```
------ Monthly Report ------
Total Income: Rs5000.00
Total Expenses: Rs1000.00
Net Savings: Rs4000.00

------ Budgeting Application ------
1. Log a Transaction
2. Generate Monthly Report
3. View All Transactions
4. Delete All Transactions
5. Exit
Enter your choice: 3

------ All Transactions ------
Type           Description              Amount
-----------------------------------------------
income         5000 credited            5000.00
expense        1000 debited             1000.00

------ Budgeting Application ------
1. Log a Transaction
2. Generate Monthly Report
3. View All Transactions
4. Delete All Transactions
5. Exit
Enter your choice: 5
Exiting the program.
```



```
transactions.csv > data
1   income,5000 credited,5000
2   expense,1000 debited,1000
```

```
            type = line.substr(0, pos1);
            description = line.substr(pos1 + 1, pos2 - pos1 - 1);
            amount = stod(line.substr(pos2 + 1));

            if (type == "income") {
                totalIncome += amount;
            } else if (type == "expense") {
                totalExpense += amount;
            }
        }
    }

    file.close();

    cout << fixed << setprecision(2);
    cout << "\n------ Monthly Report ------\n";
    cout << "Total Income: Rs" << totalIncome << endl;
    cout << "Total Expenses: Rs" << totalExpense << endl;
    cout << "Net Savings: Rs" << (totalIncome - totalExpense)
<< endl;
}

void viewAllTransactions() {
    ifstream file("transactions.csv");
    if (!file) {
        cerr << "Error opening file." << endl;
        return;
    }

    string line, type, description;
    double amount;

    cout << "\n------ All Transactions ------\n";
     cout << left << setw(15) << "Type" << setw(25) <<
"Description" << "Amount" << endl;
    cout << "---------------------------------------------\n";
    while (getline(file, line)) {
        size_t pos1 = line.find(',');
        size_t pos2 = line.rfind(',');

       if (pos1 != string::npos && pos2 != string::npos && pos1
!= pos2) {
            type = line.substr(0, pos1);
            description = line.substr(pos1 + 1, pos2 - pos1 - 1);
            amount = stod(line.substr(pos2 + 1));

                cout << left << setw(15) << type << setw(25) <<
description << fixed << setprecision(2) << amount << endl;
        }
    }

    file.close();
}

void deleteAllTransactions() {
    ofstream file("transactions.csv", ios::trunc);
    if (!file) {
        cerr << "Error opening file." << endl;
        return;
    }
    file.close();
    cout << "All transactions have been deleted." << endl;
}

int main() {
    int choice;

    while (true) {
        displayMenu();
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                logTransaction();
                break;
            case 2:
                generateMonthlyReport();

                break;
            case 3:
                viewAllTransactions();
                break;
            case 4:
                deleteAllTransactions();
                break;
            case 5:
                cout << "Exiting the program." << endl;
                return 0;
            default:
                cout << "Invalid choice. Please try again." << endl;
        }
    }

    return 0;
}
```

Conclusion – All the codes were successfully executed using the concepts of File Processing.