LAB 14: File Handling

CLOs: CLO-4

Objective(s):

- To understand how to open, check, and close files in C++ using the <fstream> library.

Introduction

The programs you have written so far require you to re-enter data each time the program runs. This is because the data stored in RAM disappears once the program stops running or the computer is shut down. If a program is to retain data between the times it runs, it must have a way of saving it. Data is saved in a file, which is usually stored on a computer's disk. Once the data is saved in a file, it will remain there after the program stops running. The data can then be retrieved and used at a later time. There are always three steps that must be taken when a file is used by a program:

1. The file must be ***opened***. If the file does not yet exist, opening it means creating it.
2. Data is then **saved** to the file, **read** from the file, or both.
3. When the program is finished using the file, the file must be ***closed***

Data Type	Description
`ifstream`	Input File Stream. This data type can be used only to read data from files into memory.
`ofstream`	Output File Stream. This data type can be used to create files and write data to them.
`fstream`	File Stream. This data type can be used to create files, write data to them, and read data from them.

class	default mode parameter
ofstream	ios::out
ifstream	ios::in
fstream	ios::in | ios::out

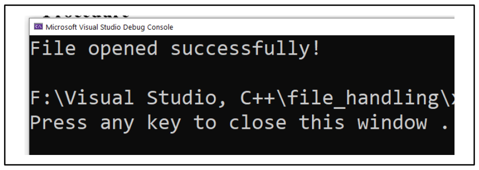
File handling in C++ uses streams: ofstream for output (writing), ifstream for input (reading), and fstream for both. Files are opened with constructors or open() method, and modes like ios::out, ios::in, ios::app control behavior. Always check if the file opened successfully with is_open() and close it with close() to avoid resource leaks.

Procedure

1. Include necessary headers.
2. Declare an ofstream object and open a file.
3. Check if open, write a message if successful.
4. Close the file.
5. Compile and run; check for the created file.

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

```cpp
int main() {
    ofstream myFile("lab1.txt");
    if (myFile.is_open()) {
        cout << "File opened successfully!" << endl;
        myFile.close();
    }
    else {
        cout << "Unable to open file." << endl;
    }
    return 0;
}
```



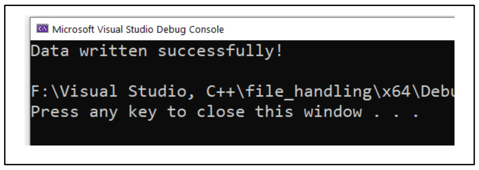
```
Microsoft Visual Studio Debug Console
File opened successfully!

F:\Visual Studio, C++\file_handling\
Press any key to close this window .
```

Writing uses the << operator, similar to cout. You can write strings, numbers, or variables. Files are overwritten by default in ios::out mode; use ios::app to add without erasing.

Procedure

1. Open a file in write mode.
2. Write multiple lines of text.
3. Close the file.
4. Manually open the file to verify contents.

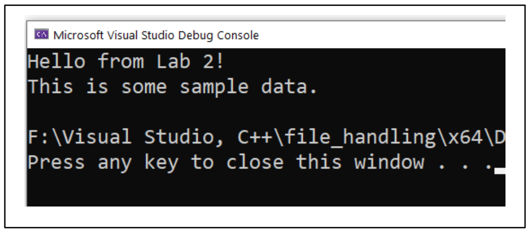
```cpp
#include <iostream>
#include <fstream>
using namespace std;

int main() {
    ofstream myFile("lab2.txt");
    if (myFile.is_open()) {
        myFile << "Hello from Lab 2!" << endl;
        myFile << "This is some sample data." << endl;
        myFile.close();
        cout << "Data written successfully!" << endl;
    }
    return 0;
}
```



```
Microsoft Visual Studio Debug Console
Data written successfully!

F:\Visual Studio, C++\file_handling\x64\Debu
Press any key to close this window . . .
```

Reading uses >> for words/numbers or getline() for entire lines. Loop until end-of-file (EOF) with while (myFile >> var) or while (getline(myFile, line)). Open the file in read mode (ifstream). Read line by line and print to console. Handle errors if file doesn't exist.

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

int main() {
    ifstream myFile("lab2.txt");
    string line;
    if (myFile.is_open()) {
        while (getline(myFile, line)) {
            cout << line << endl;
        }
        myFile.close();
    }
    else {
        cout << "Unable to open file." << endl;
    }
    return 0;
}
```



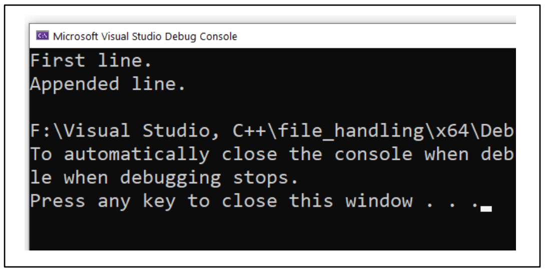
```
Microsoft Visual Studio Debug Console
Hello from Lab 2!
This is some sample data.

F:\Visual Studio, C++\file_handling\x64\D
Press any key to close this window . . .
```

File Access Flag	Meaning
`ios::app`	Append mode. If the file already exists, its contents are preserved and all output is written to the end of the file. By default, this flag causes the file to be created if it does not exist.
`ios::ate`	If the file already exists, the program goes directly to the end of it. Output may be written anywhere in the file.
`ios::binary`	Binary mode. When a file is opened in binary mode, data are written to or read from it in pure binary format. (The default mode is text.)
`ios::in`	Input mode. Data will be read from the file. If the file does not exist, it will not be created and the open function will fail.
`ios::out`	Output mode. Data will be written to the file. By default, the file's contents will be deleted if it already exists.
`ios::trunc`	If the file already exists, its contents will be deleted (truncated). This is the default mode used by `ios::out`.

1. ios::app (Append Mode)

This flag opens the file for writing at the end (appending). If the file exists, new data is added without overwriting existing content. If it doesn't exist, the file is created. It's great for logs where you don't want to lose old data.

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

int main() {
    // First run: Create and write
    ofstream file("append.txt", ios::app);
    if (file.is_open()) {
        file << "First line." << endl;
        file.close();
    }
    // Second run: Append more
    ofstream file2("append.txt", ios::app);
    if (file2.is_open()) {
        file2 << "Appended line." << endl;
        file2.close();
    }
    // Read back to verify
    ifstream readFile("append.txt");
    string line;
    while (getline(readFile, line)) {
        cout << line << endl;
    }
    readFile.close();
    return 0;
}
```



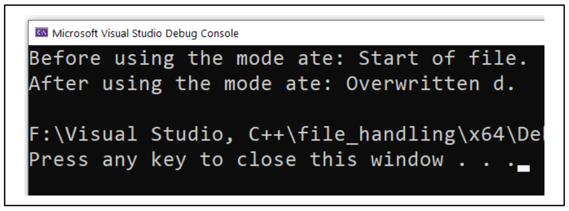
```
First line.
Appended line.

F:\Visual Studio, C++\file_handling\x64\Deb
To automatically close the console when deb
le when debugging stops.
Press any key to close this window . . .
```

Notes: Without ios::app, reopening in ios::out would overwrite the file. Always close files to ensure data flushes.

2. ios::ate (At End)

This positions the file pointer at the end of the file immediately after opening (if the file exists). Unlike ios::app, you can then seek and write anywhere in the file, not just at the end. If the file doesn't exist, it's created, and the pointer starts at the beginning.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main() {
    // Write initial content
    ofstream initFile("ate.txt");
    initFile << "Start of file.";
    initFile.close();
    cout << "Before using the mode ate: ";
    ifstream readFile("ate.txt");
    string content;
    getline(readFile, content);
    cout << content << endl;

    // Open with ate: Pointer at end
    fstream file("ate.txt", ios::out | ios::ate);
    if (file.is_open()) {
        // Write at end (current position)
        file << " Added at end.";
        // Seek to beginning and overwrite
        file.seekp(0);
        file << "Overwritten ";
        file.close();
    }

    cout << "After using the mode ate: ";
    ifstream readFile1("ate.txt");
    getline(readFile1, content);
    cout << content << endl;
    readFile.close();
    return 0;
}
```



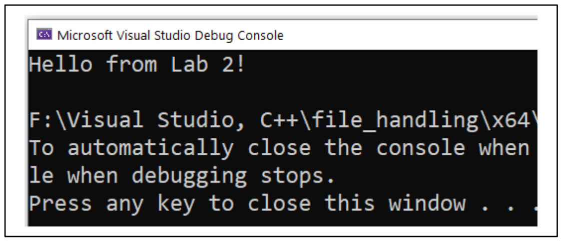
```
Before using the mode ate: Start of file.
After using the mode ate: Overwritten d.

F:\Visual Studio, C++\file_handling\x64\De
Press any key to close this window . . .
```

Notes: Combine with ios::in | ios::out for read/write. ios::ate doesn't force appending like ios::app; it just sets the initial position.

3. ios::in (Input Mode)

Opens the file for reading. The file must exist; otherwise, opening fails. No writing is allowed in this mode alone (use with ios::out for read/write).

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

int main() {
    // Assume "lab2.txt" exists
    ifstream file("lab2.txt", ios::in);
    if (file.is_open()) {
        string content;
        getline(file, content);
        cout << content << endl;
        file.close();
    }
    else {
        cout << "File does not exist or cannot be opened." << endl;
    }
    return 0;
}
```



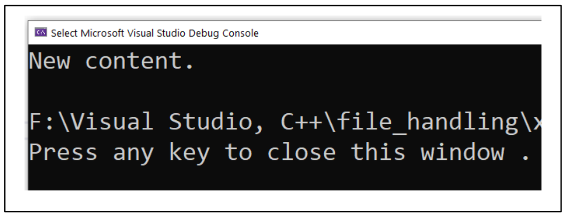
```
Hello from Lab 2!

F:\Visual Studio, C++\file_handling\x64\
To automatically close the console when
le when debugging stops.
Press any key to close this window . .
```

Notes: This is the default for ifstream. Trying to write will fail or cause undefined behavior.

4. ios::out (Output Mode)

Opens the file for writing. If it exists, contents are deleted (truncated) by default. If it doesn't exist, it's created. Use for creating or overwriting files.

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

int main() {
    // Write to file (overwrites if exists)
    ofstream file("output.txt", ios::out);
    if (file.is_open()) {
        file << "New content." << endl;
        file.close();
    }

    // Read back
    ifstream readFile("output.txt");
    string line;
    getline(readFile, line);
    cout << line << endl;
    readFile.close();
    return 0;
}
```



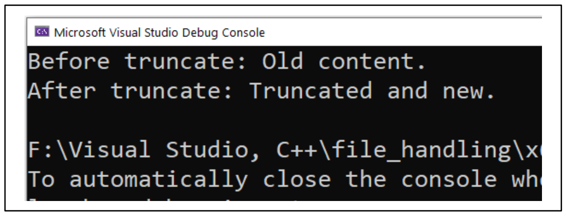
```
New content.

F:\Visual Studio, C++\file_handling\x
Press any key to close this window .
```

Notes: This is the default for ofstream. Combine with ios::app to avoid truncation.

5. ios::trunc (Truncate Mode)

If the file exists, its contents are deleted (truncated to zero length) before opening. This is automatically used with ios::out by default, but you can specify it explicitly.

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

int main() {
    ofstream initFile("trunc.txt");
    initFile << "Old content.";
    initFile.close();

    ifstream readFile("trunc.txt");
    string line;
    getline(readFile, line);
    cout << "Before truncate: " << line << endl;
    // Open with trunc: Deletes old content
    ofstream file("trunc.txt", ios::out | ios::trunc);
    if (file.is_open()) {
        file << "Truncated and new." << endl;
        file.close();
    }

    ifstream readFile1("trunc.txt");
    getline(readFile1, line);
    cout << "After truncate: " << line << endl;
    readFile.close();
    return 0;
}
```



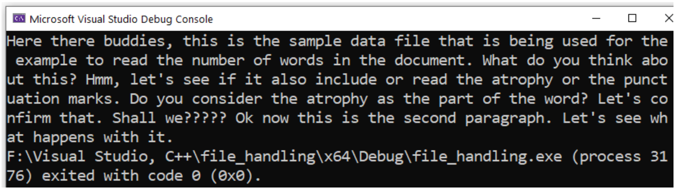
```
Before truncate: Old content.
After truncate: Truncated and new.

F:\Visual Studio, C++\file_handling\x
To automatically close the console wh
```

Notes: Redundant with plain ios::out, but useful for clarity or when combining modes. Doesn't create the file if not used with ios::out.

Example: Let us learn how to count to words in the file.

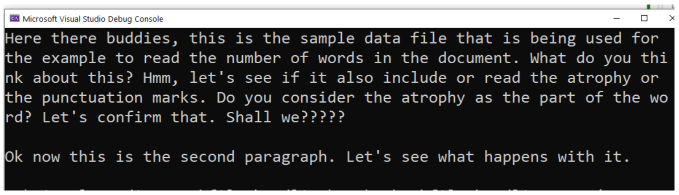
```
#include<iostream>
#include<fstream>
#include<string>
using namespace std;
```

```
int main() {
    ifstream myFile("data.txt");
    string word;
    if (myFile.is_open()) {
        while (myFile >> word) {
            cout << word << " ";
        }
        myFile.close();
    }


    return 0;
}
```



```
Here there buddies, this is the sample data file that is being used for the
 example to read the number of words in the document. What do you think abo
ut this? Hmm, let's see if it also include or read the atrophy or the punct
uation marks. Do you consider the atrophy as the part of the word? Let's co
nfirm that. Shall we????? Ok now this is the second paragraph. Let's see wh
at happens with it.
F:\Visual Studio, C++\file_handling\x64\Debug\file_handling.exe (process 31
76) exited with code 0 (0x0).
```

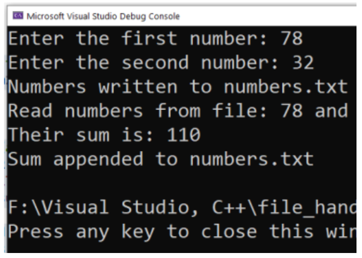
```
int main() {
    ifstream myFile("data.txt");
    string line;
    if (myFile.is_open()) {
        while (getline(myFile, line)) {
            cout << line << endl;
        }
        myFile.close();
    }
    return 0;
}
```



```
Here there buddies, this is the sample data file that is being used for
the example to read the number of words in the document. What do you thi
nk about this? Hmm, let's see if it also include or read the atrophy or
the punctuation marks. Do you consider the atrophy as the part of the wo
rd? Let's confirm that. Shall we?????

Ok now this is the second paragraph. Let's see what happens with it.
```

Example: The C++ program where we are taking the input in a variable and saving in the file and applying manipulation on it, let's say, take two variables from user, write those in a file and then read those two variables and add them and write that also on the file.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main() {
    int num1, num2;
    string filename = "numbers.txt";

    cout << "Enter the first number: ";
    cin >> num1;
    cout << "Enter the second number: ";
    cin >> num2;

    // Use ofstream to create and write to the file. This overwrites if the file already exists.
    ofstream outFile(filename);
    if (!outFile.is_open()) {
        cerr << "Error: Could not open file for writing." << endl;
        return 1;
    }
    outFile << num1 << " " << num2 << endl;
    outFile.close();
    cout << "Numbers written to " << filename << endl;

    ifstream inFile(filename);
    if (!inFile.is_open()) {
        cerr << "Error: Could not open file for reading." << endl;
        return 1;
    }

    int readNum1, readNum2;
    inFile >> readNum1 >> readNum2;

    inFile.close();

    int sum = readNum1 + readNum2;
    cout << "Read numbers from file: " << readNum1 << " and " << readNum2 << endl;
```



```
Enter the first number: 78
Enter the second number: 32
Numbers written to numbers.txt
Read numbers from file: 78 and
Their sum is: 110
Sum appended to numbers.txt

F:\Visual Studio, C++\file_hand
Press any key to close this wir
```

```cpp
    cout << "Their sum is: " << sum << endl;

    // Re-open the file in append mode using ofstream with ios::app
    // Opening with 'app' ensures data is added to end rather than overwriting existing content.
    ofstream appendFile(filename, ios::app);
    if (!appendFile.is_open()) {
        cerr << "Error: Could not open file for appending." << endl;
        return 1;
    }
    appendFile << "Sum: " << sum << endl;
    appendFile.close();
    cout << "Sum appended to " << filename << endl;
    return 0;
}
```

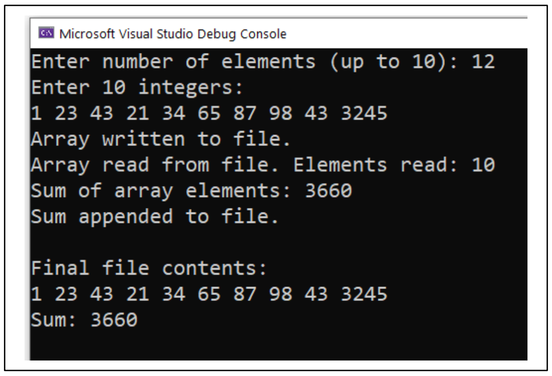
C++ File Handling with Arrays and Functions

To demonstrate the file handling with array and functions, let us understand this through the following example. The program will:

- Take user input to fill an array.
- Use a function to write the array to a text file.
- Use another function to read the array back from the file into a new array.
- Use a third function to manipulate the array (e.g., calculate the sum of elements).
- Append the sum to the file using append mode.

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

// Function to write an array to a file
void writeArrayToFile(const string& filename, int arr[], int size) {
    ofstream outFile(filename);
    if (outFile.is_open()) {
        for (int i = 0; i < size; ++i) {
            outFile << arr[i] << " ";
        }
        outFile << endl;
        outFile.close();
        cout << "Array written to file." << endl;
    }
    else {
        cout << "Unable to open file for writing." << endl;
    }
}

// Function to read an array from a file
void readArrayFromFile(const string& filename, int arr[], int& size, int maxSize) {
    ifstream inFile(filename);
    if (inFile.is_open()) {
        size = 0;  // Reset size
        while (inFile >> arr[size] && size < maxSize) {
            ++size;
        }
        inFile.close();
        cout << "Array read from file. Elements read: " << size << endl;
    }
    else {
        cout << "Unable to open file for reading." << endl;
        size = 0;
    }
}

// Function to manipulate the array (e.g., calculate sum)
```

```cpp
int calculateSum(int arr[], int size) {
    int sum = 0;
    for (int i = 0; i < size; ++i) {
        sum += arr[i];
    }
    return sum;
}

int main() {
    const int MAX_SIZE = 10;
    int inputArray[MAX_SIZE];
    int readArray[MAX_SIZE];
    int inputSize = 0;

    cout << "Enter number of elements (up to " << MAX_SIZE << "): ";
    cin >> inputSize;
    if (inputSize > MAX_SIZE) {
        inputSize = MAX_SIZE;
    }
    cout << "Enter " << inputSize << " integers:" << endl;
    for (int i = 0; i < inputSize; ++i) {
        cin >> inputArray[i];
    }

    string filename = "array_data.txt";
    writeArrayToFile(filename, inputArray, inputSize);

    int readSize;
    readArrayFromFile(filename, readArray, readSize, MAX_SIZE);

    if (readSize > 0) {
        int sum = calculateSum(readArray, readSize);
        cout << "Sum of array elements: " << sum << endl;
        ofstream appendFile(filename, ios::app);
        if (appendFile.is_open()) {
            appendFile << "Sum: " << sum << endl;
            appendFile.close();
            cout << "Sum appended to file." << endl;
        }
    }

    cout << "\nFinal file contents:" << endl;
    ifstream finalRead(filename);
    string line;
    while (getline(finalRead, line)) {
        cout << line << endl;
    }
    finalRead.close();

    return 0;
}
```



```
Enter number of elements (up to 10): 12
Enter 10 integers:
1 23 43 21 34 65 87 98 43 3245
Array written to file.
Array read from file. Elements read: 10
Sum of array elements: 3660
Sum appended to file.

Final file contents:
1 23 43 21 34 65 87 98 43 3245
Sum: 3660
```

Tasks

Task 1: Write a program that asks the user to enter the names of 10 employees and store the names in a file “Names.txt”. Display the total number of words in the file.

Task 2: Create a file "numbers.txt" and write integers 1 to 10, each on a new line. Read them back and sum them up, printing the total.

Task 3: Write a C++ program that takes a file as an input and search and replace: Read "data.txt", replace all occurrences of "old" with "new", and write to "updated.txt".

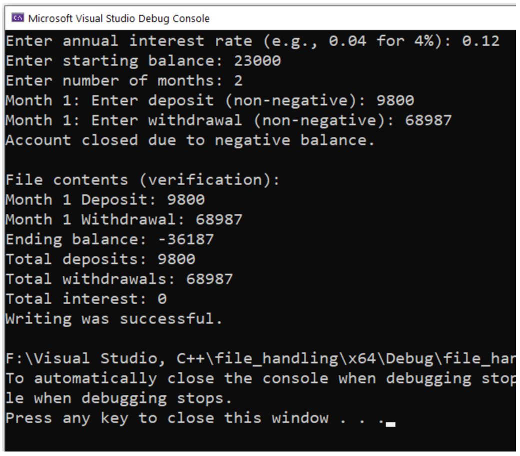
Task 4:

Write a program that calculates the balance of a savings account at the end of a period of time. It should ask the user for the annual interest rate, the starting balance, and the number of months that have passed since the account was established. A loop should then iterate once for every month, performing the following:

- Ask the user for the amount deposited into the account during the month. (Do not accept negative numbers.) This amount should be added to the balance. Write these in a file "Accounts.txt".
- Ask the user for the amount withdrawn from the account during the month. (Do not accept negative numbers.) This amount should be subtracted from the balance. Read these from the file "Accounts.txt".
- Calculate the monthly interest. The monthly interest rate is the annual interest rate divided by twelve. Multiply the monthly interest rate by the balance and add the result to the balance. Write the results in "Accounts.txt".

After the last iteration, the program should display the ending balance, the total amount of deposits, the total amount of withdrawals, and the total interest earned. Write these in "Accounts.txt" to save them and then read the file "Accounts.txt" to see if the writing was successful or not.

NOTE: If a negative balance is calculated at any point, a message should be displayed indicating the account has been closed and the loop should terminate.



```
Enter annual interest rate (e.g., 0.04 for 4%): 0.12
Enter starting balance: 23000
Enter number of months: 2
Month 1: Enter deposit (non-negative): 9800
Month 1: Enter withdrawal (non-negative): 68987
Account closed due to negative balance.

File contents (verification):
Month 1 Deposit: 9800
Month 1 Withdrawal: 68987
Ending balance: -36187
Total deposits: 9800
Total withdrawals: 68987
Total interest: 0
Writing was successful.

F:\Visual Studio, C++\file_handling\x64\Debug\file_har
To automatically close the console when debugging stop
le when debugging stops.
Press any key to close this window . . .
```

Task 5: **File Handling with Arrays of Strings and Functions for Sorting**

Create a C++ program that:

- Uses an array of strings (e.g., a fixed-size array of std::string for names or words, say, max 10 elements).
- Takes user input to fill the array (e.g., enter a list of words).
- Defines a function to write the array to a text file (one string per line).
- Defines another function to read the array back from the file.
- Defines a third function to manipulate the array (e.g., sort the strings alphabetically using bubble sort).
- Appends the sorted list to the same file (in append mode) with a header like "Sorted List:".
- Include basic error handling and display the final file contents.