Digital Empowerment Pakistan

Internship Batch 3 DEN

Domain:
C++ programming

task :03

Submitted By:

Laiba Husna

Gmail:

Laibahusna163@gmail.com

Task3:

Implementing a Simple File Compression Algorithm:

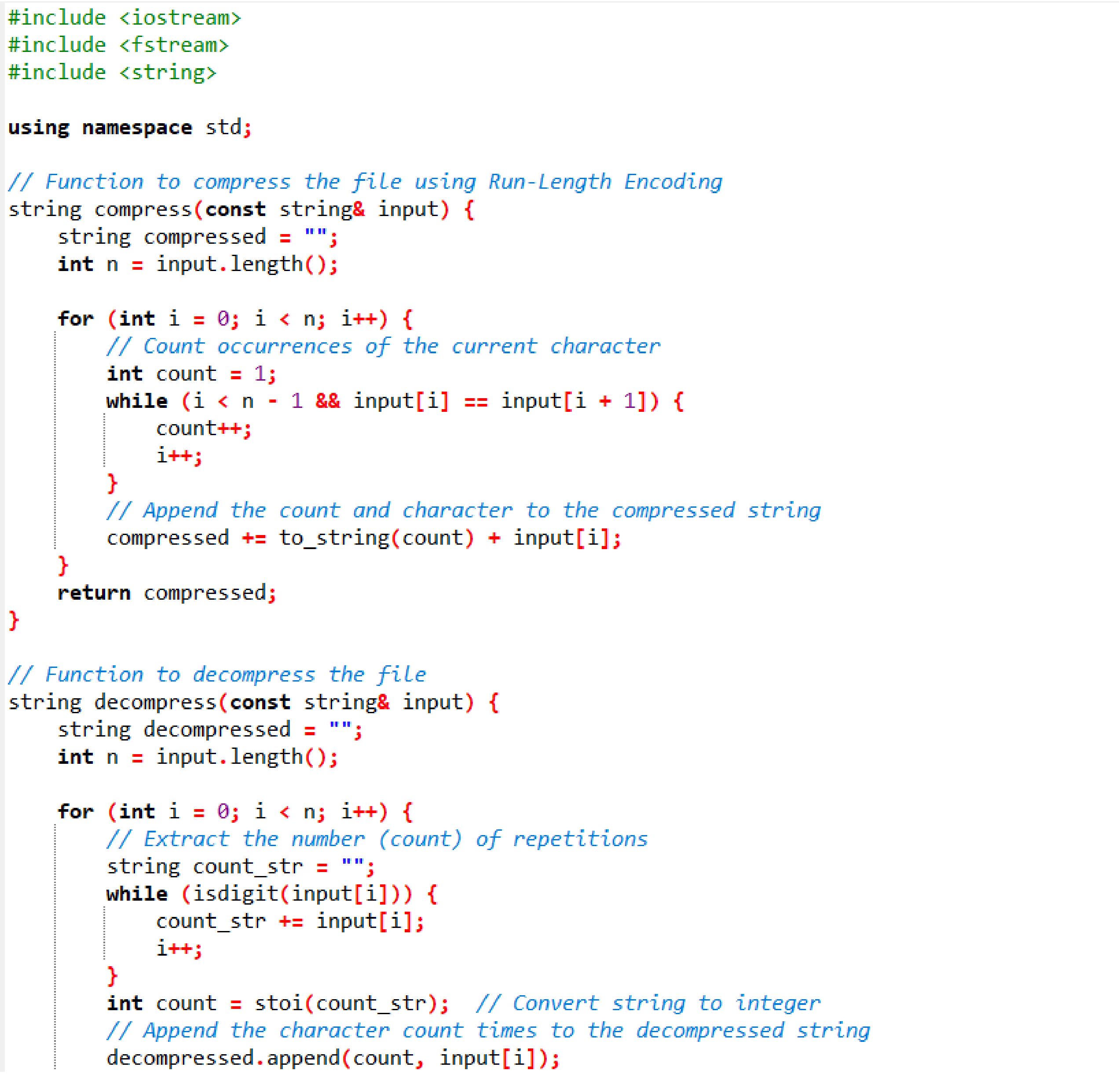
- Objective: Develop a basic file compression and decompression tool.
- Description: Create a C++ program that reads a file, compresses its content using a simple algorithm (e.g., Run-Length Encoding), and writes the compressed data to a new file. Also, implement decompression.
- Key Steps:

o Reading and writing files
o Implementing the Run-Length Encoding algorithm
o Handling edge cases (e.g., different file types, empty files)
o Creating functions for both compression and decompression

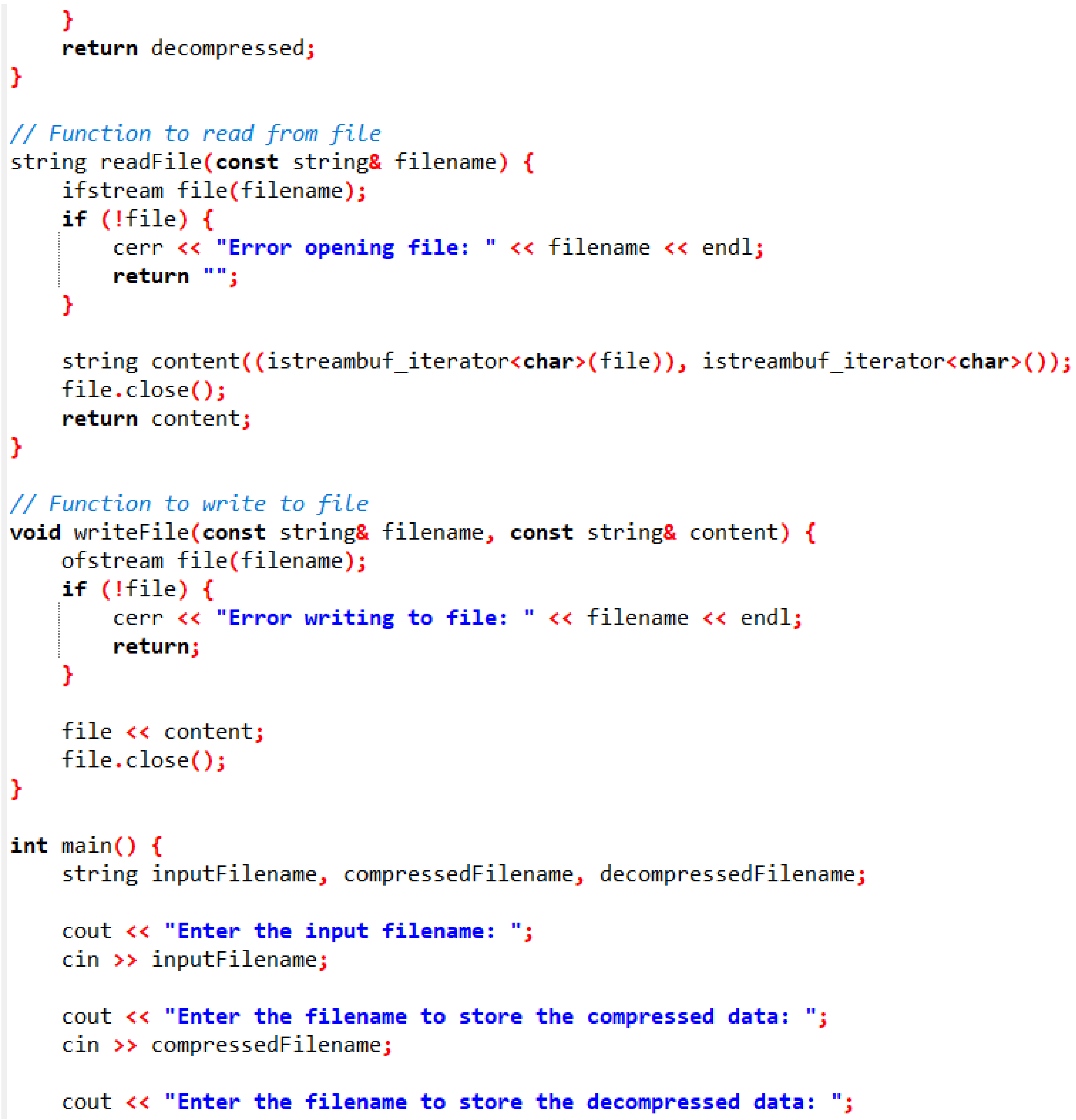
Solution:

Code:

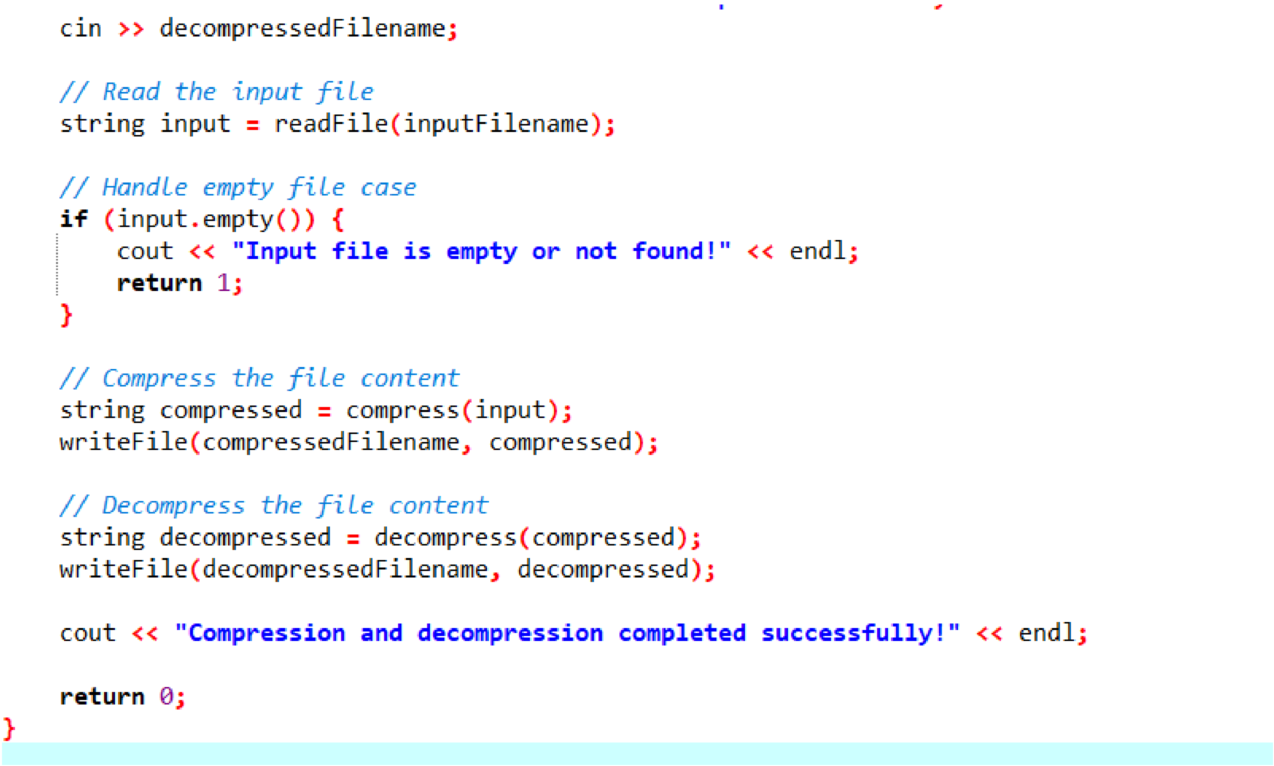
3



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

using namespace std;

// Function to compress the file using Run-Length Encoding
string compress(const string& input) {
    string compressed = "";
    int n = input.length();

    for (int i = 0; i < n; i++) {
        // Count occurrences of the current character
        int count = 1;
        while (i < n - 1 && input[i] == input[i + 1]) {
            count++;
            i++;
        }
        // Append the count and character to the compressed string
        compressed += to_string(count) + input[i];
    }
    return compressed;
}

// Function to decompress the file
string decompress(const string& input) {
    string decompressed = "";
    int n = input.length();

    for (int i = 0; i < n; i++) {
        // Extract the number (count) of repetitions
        string count_str = "";
        while (isdigit(input[i])) {
            count_str += input[i];
            i++;
        }
        int count = stoi(count_str);  // Convert string to integer
        // Append the character count times to the decompressed string
        decompressed.append(count, input[i]);
```



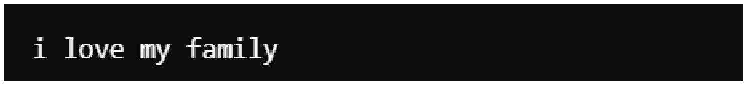
```
    }
    return decompressed;
}

// Function to read from file
string readFile(const string& filename) {
    ifstream file(filename);
    if (!file) {
        cerr << "Error opening file: " << filename << endl;
        return "";
    }

    string content((istreambuf_iterator<char>(file)), istreambuf_iterator<char>());
    file.close();
    return content;
}

// Function to write to file
void writeFile(const string& filename, const string& content) {
    ofstream file(filename);
    if (!file) {
        cerr << "Error writing to file: " << filename << endl;
        return;
    }

    file << content;
    file.close();
}

int main() {
    string inputFilename, compressedFilename, decompressedFilename;

    cout << "Enter the input filename: ";
    cin >> inputFilename;

    cout << "Enter the filename to store the compressed data: ";
    cin >> compressedFilename;

    cout << "Enter the filename to store the decompressed data: ";
```



```
    cin >> decompressedFilename;

    // Read the input file
    string input = readFile(inputFilename);

    // Handle empty file case
    if (input.empty()) {
        cout << "Input file is empty or not found!" << endl;
        return 1;
    }

    // Compress the file content
    string compressed = compress(input);
    writeFile(compressedFilename, compressed);

    // Decompress the file content
    string decompressed = decompress(compressed);
    writeFile(decompressedFilename, decompressed);

    cout << "Compression and decompression completed successfully!" << endl;

    return 0;
}
```

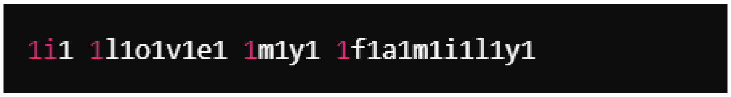
Output:

input file (laiba's file):



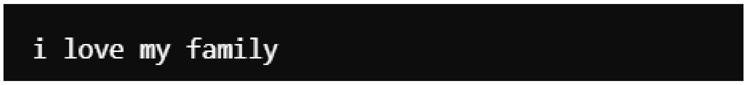
```
i love my family
```

Compressed file (compressed.txt):



```
1i1 1l1o1v1e1 1m1y1 1f1a1m1i1l1y1
```

Decompressed file (decompressed.txt):



```
i love my family
```

Explanation:

This C++ program performs file compression and decompression using the **Run-Length Encoding (RLE)** algorithm. Here's a detailed explanation of its key components and how they work:

---

1. Run-Length Encoding (RLE) Algorithm:

- **Purpose**: The RLE algorithm reduces the size of data by encoding consecutive repeating characters. Each sequence of repeating characters is replaced by the number of repetitions followed by the character.
- **Example**: The string `"AAABBBCC"` would be encoded as `"3A3B2C"`. This reduces storage space, especially when there are many consecutive identical characters.
- **Limitation**: If the string does not contain many consecutive repetitions (like in normal text), the compression may not significantly reduce the file size.

---

2. File Compression:

- The program reads the content of an input file and compresses it using RLE.
- **Compression Logic**:
  - It examines each character in the file.
  - If a character is repeated consecutively, the program counts the repetitions and stores the count followed by the character.
  - For characters that do not repeat, the count is simply `1`, followed by the character.

Example: For the text `"i love my family"`, there are no consecutive repetitions, so each character is compressed with a count of `1`, producing a compressed output like `"1i1 1l1o1v1e1 1m1y1 1f1a1m1i1l1y1 "`.

---

3. File Decompression:

- The compressed data can be decompressed back to its original form. The decompression function reads the compressed file, extracts the number of repetitions, and then restores the original characters.
- **Decompression Logic**:
  - It scans the compressed data for the numbers representing the counts and the characters that follow them.
  - For each number, it appends the corresponding character to a new string the specified number of times, recreating the original uncompressed text.

Example: The compressed output `"1i1 1l1o1v1e1 1m1y1 1f1a1m1i1l1y1  "` will be decompressed to `"i love my family "`.

---

4. File Input and Output:

- **File Reading**: The program reads the entire content of the input file (which could be any type of text file) into a string. If the file cannot be opened, an error message is displayed, and the program exits.
- **File Writing**: After compression or decompression, the results are written to new files. If any issue occurs while writing to the file, the program informs the user of the error.

5. User Interaction:

- When the program is executed, it prompts the user to provide:
  - The **input filename**: The name of the file containing the original (uncompressed) data.
  - The **compressed filename**: The name of the file where the compressed data will be stored.
  - The **decompressed filename**: The name of the file where the decompressed data will be written.

Once the filenames are provided, the program:

- Reads the input file, compresses its content, and writes the compressed data to the specified file.
- Decompresses the compressed data and writes the restored content to another file.

---

6. Edge Case Handling:

- **Empty File**: If the input file is empty or does not exist, the program will print an error message and stop, ensuring the user knows there's a problem.
- **Non-Text Files**: The program handles basic text files. For more complex file formats (like binary files), the logic would need modification, as the program currently processes the input as a string.

---

7. Compression and Decompression Process:

- **Compression**:
  - Each character and its consecutive repetitions are tracked.
  - This results in a compressed file where each character is preceded by the number of its occurrences.
- **Decompression**:
  - The program interprets the compressed data by reading the counts and recreating the original sequence of characters, exactly as they were in the input file.

Final Output:

After running the program, the user will have three files:

1. **Original file** (input file with the original text).
2. **Compressed file** (containing the compressed version of the text).
3. **Decompressed file** (containing the restored original text).

The program concludes by displaying a success message, confirming that the compression and decompression processes have been completed successfully.Bottom of Form