Internship Program (Batch 2)

Task.#.3

Name: Mian Muhammad Awais

Section: C++ (Programming)

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.

✞ Code with explanation:

1. Includes:

```
1  #include <iostream>
2  #include <fstream>
3  #include <string>
4
5  using namespace std;
6
```

- **#include <iostream>:** This includes the standard input/output stream library, which allows the program to use cout and cerr for console output.
- **#include <fstream>:** This includes the file stream library, which allows the program to read from and write to files using ifstream (input file stream) and ofstream (output file stream).
- **#include <string>:** This includes the string library, allowing the program to use the string class, which is used to represent and manipulate strings.

2. Reading a File:

```
7  // Function to read the contents of a file into a string
8  string readFile(const string& filename) {
9      ifstream file(filename, ios::binary);
10     if (!file) {
11         cerr << "File could not be opened!" << endl;
12         return "";
13     }
14
15     string content((istreambuf_iterator<char>(file)), istreambuf_iterator<char>());
16     file.close();
17     return content;
18 }
19
```

- ifstream file(filename, ios::binary);: This line creates an input file stream object to read from the file specified by filename. The ios::binary flag ensures that the file is read in binary mode, which is important for reading non-text files or preserving exact content.
- if (!file): This checks if the file stream was successfully opened. If not, it prints an error message using cerr and returns an empty string.

- string content((istreambuf_iterator<char>(file)), istreambuf_iterator<char>());: This reads the entire file into a string. The istreambuf_iterator<char> is an iterator that reads characters from the file until the end.
- file.close();: Closes the file after reading.
- return content;: Returns the content of the file as a string.

3. **Writing to a File:**

```
19
20  // Function to write a string to a file
21  void writeFile(const string& filename, const string& content) {
22      ofstream file(filename, ios::binary);
23      if (!file) {
24          cerr << "File could not be created!" << endl;
25          return;
26      }
27
28      file << content;
29      file.close();
30  }
31
```

- ofstream file(filename, ios::binary);: This creates an output file stream object to write to the file specified by filename. The ios::binary flag is used to write data in binary mode.
- if (!file): Checks if the file was successfully created or opened for writing. If not, it prints an error message and exits the function.
- file << content;: Writes the string content to the file.
- file.close();: Closes the file after writing.

4. **Run-Length Encoding (RLE) Compression**:

```
31
32  // Function to compress data using RLE
33  string compressRLE(const string& data) {
34      string compressed = "";
35      int length = data.length();
36
37      for (int i = 0; i < length; i++) {
38          int count = 1;
39
40          // Count the number of consecutive characters
41          while (i < length - 1 && data[i] == data[i + 1]) {
42              count++;
43              i++;
44          }
45
46          // Append the character and its count to the compressed string
47          compressed += data[i];
48          compressed += to_string(count);
49      }
50
51      return compressed;
52  }
```

- string compressRLE(const string& data): This function compresses the input string using the RLE algorithm.
- for (int i = 0; i < length; i++): This loop iterates through each character in the string.
- int count = 1;: Initializes a counter to count consecutive characters.
- while (i < length - 1 && data[i] == data[i + 1]): This inner loop counts how many times the current character is repeated consecutively.
- compressed += data[i];: Adds the character to the compressed string.
- compressed += to_string(count);: Adds the count of consecutive characters to the compressed string.
- return compressed;: Returns the compressed string.

5. **Run-Length Encoding (RLE) Decompression:**

```
54  // Function to decompress RLE data
55  string decompressRLE(const string& data) {
56      string decompressed = "";
57      int length = data.length();
58
59      for (int i = 0; i < length; i++) {
60          char currentChar = data[i];
61          i++;
62
63          // The next characters should be digits, representing the count
64          string countStr = "";
65          while (i < length && isdigit(data[i])) {
66              countStr += data[i];
67              i++;
68          }
69
70          // Convert the count to an integer
71          int count = stoi(countStr);
72
73          // Append the character 'count' times to the decompressed string
74          decompressed.append(count, currentChar);
75
76          // Adjust index since the loop increments 'i' again
77          i--;
78      }
79
80      return decompressed;
81  }
82
```

- string decompressRLE(const string& data): This function decompresses the input string encoded with RLE.
- for (int i = 0; i < length; i++): This loop iterates through each character in the string.
- char currentChar = data[i];: Gets the current character to be decompressed.
- string countStr = "";: Initializes a string to store the count of repetitions.
- while (i < length && isdigit(data[i])): This inner loop reads digits following the character, which represent the count.
- int count = stoi(countStr);: Converts the count string to an integer.

- decompressed.append(count, currentChar);: Adds the character to the decompressed string count times.
- i--;: Adjusts the loop counter since it will be incremented in the next iteration.
- return decompressed;: Returns the decompressed string.

6. **Main Function:**

```
83  int main() {
84      string filename = "awais.txt";
85      string compressedFile = "compressed_awais.txt";
86      string decompressedFile = "decompressed_awais.txt";
87
88      // Read the original file
89      string data = readFile(filename);
90
91      if (data.empty()) {
92          cerr << "The file is empty or could not be read." << endl;
93          return 1;
94      }
95
96      // Compress the data
97      string compressedData = compressRLE(data);
98      writeFile(compressedFile, compressedData);
99
100     // Decompress the data
101     string decompressedData = decompressRLE(compressedData);
102     writeFile(decompressedFile, decompressedData);
103
104     cout << "Compression and decompression completed." << endl;
105
106     return 0;
107 }
```

- string filename = "awais.txt";: Sets the name of the file to be compressed.
- string compressedFile = "compressed_awais.txt";: Sets the name for the file where the compressed data will be stored.
- string decompressedFile = "decompressed_awais.txt";: Sets the name for the file where the decompressed data will be stored.
- string data = readFile(filename);: Reads the content of awais.txt into the data string.

- if (data.empty()): Checks if the file was empty or couldn't be read, and if so, exits with an error message.
- string compressedData = compressRLE(data);: Compresses the file content using RLE.
- writeFile(compressedFile, compressedData);: Writes the compressed data to compressed_awais.txt.
- string decompressedData = decompressRLE(compressedData);: Decompresses the data.
- writeFile(decompressedFile, decompressedData);: Writes the decompressed data to decompressed_awais.txt.
- cout << "Compression and decompression completed." << endl;: Outputs a completion message.
- return 0;: Ends the program successfully.

Output:

After running the program, you will have three files:

- **awais.txt (original file)**

```
Hitec University
```

- **compressed_awais.txt (compressed version)**

```
H1i1t1e1c1 1U1n1i1v1e1r1s1i1t1y1
```

- **decompressed_awais.txt (decompressed to match the original)**

```
Hitec University
```