Steganography Image Encoder and Decoder

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

void setLSB(char &byte, int bit) {
    byte = (byte & 0xFE) | (bit & 1);
}

int getLSB(char byte) {
    return byte & 1;
}

void encodeMessageInImage(const string &imageFile, const string &message,
const string &outputFile) {
    ifstream image(imageFile, ios::binary);
    ofstream output(outputFile, ios::binary);

    if (!image.is_open() || !output.is_open()) {
        cerr << "Error opening file!" << endl;
        return;
    }

    char header[54];
    image.read(header, 54);
    output.write(header, 54);

    string binaryMessage;
    for (char c : message) {
        binaryMessage += bitset<8>(c).to_string();
    }

    binaryMessage += "00000000";

    char pixel;
    size_t messageIndex = 0;

    while (image.get(pixel)) {
        if (messageIndex < binaryMessage.size()) {
            setLSB(pixel, binaryMessage[messageIndex] - '0');
            messageIndex++;
        }
        output.put(pixel);
    }

    image.close();
    output.close();
```

```
    cout << "Message encoded in " << outputFile << endl;
}

string decodeMessageFromImage(const string &imageFile) {
    ifstream image(imageFile, ios::binary);

    if (!image.is_open()) {
        cerr << "Error opening file!" << endl;
        return "";
    }

    image.seekg(54);

    string binaryMessage;
    char pixel;
    string message;

    while (image.get(pixel)) {
        binaryMessage += to_string(getLSB(pixel));

        if (binaryMessage.size() == 8) {
            char decodedChar = bitset<8>(binaryMessage).to_ulong();
            if (decodedChar == '\0') {
                break;
            }
            message += decodedChar;
            binaryMessage.clear();
        }
    }
    image.close();
    return message;
}

int main() {
    string imageFile = "input.bmp";
    string outputFile = "output.bmp";
    string message;
    int choice;
    do{
        printf("1. Encode\n2. Decode\n3. Exit\n");
        printf("Enter choice: ");
        cin >> choice;
        cin.ignore();
        switch(choice) {
            case 1:
                printf("Enter message: ");
                getline(cin,  message);
```

```
                encodeMessageInImage(imageFile, message, outputFile);
                break;
            case 2:
                cout << "The message in the image is: " <<
decodeMessageFromImage(outputFile) << endl;
                break;
            case 3:
                printf("Thank you!\n");
                break;
            default:
                cout << "Invalid choice" << endl;
        }
    } while(choice != 3);
    return 0;
}
```

The functions setLSB and getLSB are used to manipulate the least significant bit (LSB) of a byte. These functions are crucial for the steganography process implemented in your code.

setLSB

```
void setLSB(char &byte, int bit) {
    byte = (byte & 0xFE) | (bit & 1); // Set the LSB to the given bit
}
```

- **Purpose**: This function sets the least significant bit of a byte to a specified value (0 or 1).
- **Parameters**:
  - char &byte: A reference to the byte whose LSB is to be modified.
  - int bit: The bit value (0 or 1) to set as the LSB.
- **Operation**:
  - byte & 0xFE: Clears the LSB of the byte. 0xFE is 11111110 in binary, so the bitwise AND operation with byte results in a byte with the LSB set to 0.
  - bit & 1: Ensures that only the least significant bit of bit is considered.
  - | (bit & 1): Sets the LSB of byte to the value of bit.

getLSB

```
int getLSB(char byte) {
    return byte & 1;
}
```

- **Purpose**: This function retrieves the least significant bit of a byte.
- **Parameters**:

    - char byte: The byte from which the LSB is to be extracted.

- **Operation**:
    - byte & 1: Performs a bitwise AND operation with 1 (00000001 in binary), which isolates the LSB of byte.

Usage in Steganography

These functions are used in the encoding and decoding processes:

- **Encoding**: The setLSB function is used to embed each bit of the binary message into the LSBs of the image pixels.

```
setLSB(pixel, binaryMessage[messageIndex] - '0'); // Modify the LSB
```

- **Decoding**: The getLSB function is used to extract the LSBs from the image pixels to reconstruct the binary message.

```
binaryMessage += to_string(getLSB(pixel));
```

Example

Here is an example of how these functions are used in the context of your code:

```
// Encoding a message
char pixel = 0b10101010; // Example pixel byte
setLSB(pixel, 1); // Set LSB to 1
// pixel is now 0b10101011
// Decoding a message
char pixel = 0b10101011; // Example pixel byte
int lsb = getLSB(pixel); // lsb is 1
```

encodeMessageInImage

This function embeds a message into the least significant bits (LSBs) of the pixels in a BMP image.

Steps:

1. **Open Files**: Opens the input BMP image file and the output BMP file in binary mode.
2. **Check File Open Status**: Checks if both files are successfully opened. If not, it prints an error message and returns.
3. **Copy BMP Header**: Reads the first 54 bytes (BMP header) from the input image and writes it to the output file.
4. **Convert Message to Binary**: Converts each character of the message to its binary representation (8 bits) and appends it to binaryMessage.
5. **Add Null Terminator**: Appends "00000000" to binaryMessage to mark the end of the message.

6. **Embed Message**: Iterates through each pixel in the image, modifying the LSB of each pixel to embed the binary message.
7. **Write Modified Pixels**: Writes the modified pixels to the output file.
8. **Close Files**: Closes both the input and output files.
9. **Print Success Message**: Prints a message indicating that the encoding is complete.

decodeMessageFromImage

This function extracts a hidden message from the least significant bits (LSBs) of the pixels in a BMP image.

Steps:

1. **Open File**: Opens the BMP image file in binary mode.
2. **Check File Open Status**: Checks if the file is successfully opened. If not, it prints an error message and returns an empty string.
3. **Skip BMP Header**: Skips the first 54 bytes (BMP header) of the image file.
4. **Extract Binary Message**: Iterates through each pixel in the image, extracting the LSB of each pixel and appending it to binaryMessage.
5. **Convert Binary to Characters**: Converts each 8-bit binary string to its corresponding character.
6. **Check for Null Terminator**: If a null character ('\0') is found, it breaks the loop, indicating the end of the message.
7. **Append to Message**: Appends the decoded character to the message string and clears binaryMessage for the next character.
8. **Close File**: Closes the image file.
9. **Return Message**: Returns the decoded message.

These functions work together to hide and retrieve messages within BMP images by manipulating the least significant bits of the image pixels.