Step 1: Problem Identification and Statement:

The objective is to design a "hamming code" software that detects and corrects errors while the data is being transmitted over unreliable networks. The software asks the user to choose the function desired: encoding, decoding or exiting the program. Then, it asks the user to input the name of the file that contains the messages. Then, if the user chose to encode, the program would encode the 4-bit messages provided from the file by embedding 3 parity bits into specific positions. The program would then print the encoded messages and generate a file containing them. Otherwise, if the user chose to decode messages, the program would identify whether each message has an error, correct it, provide the position of the bit that was incorrect and print the original decoded message. Finally, it would print that information to the user and write it into a file.

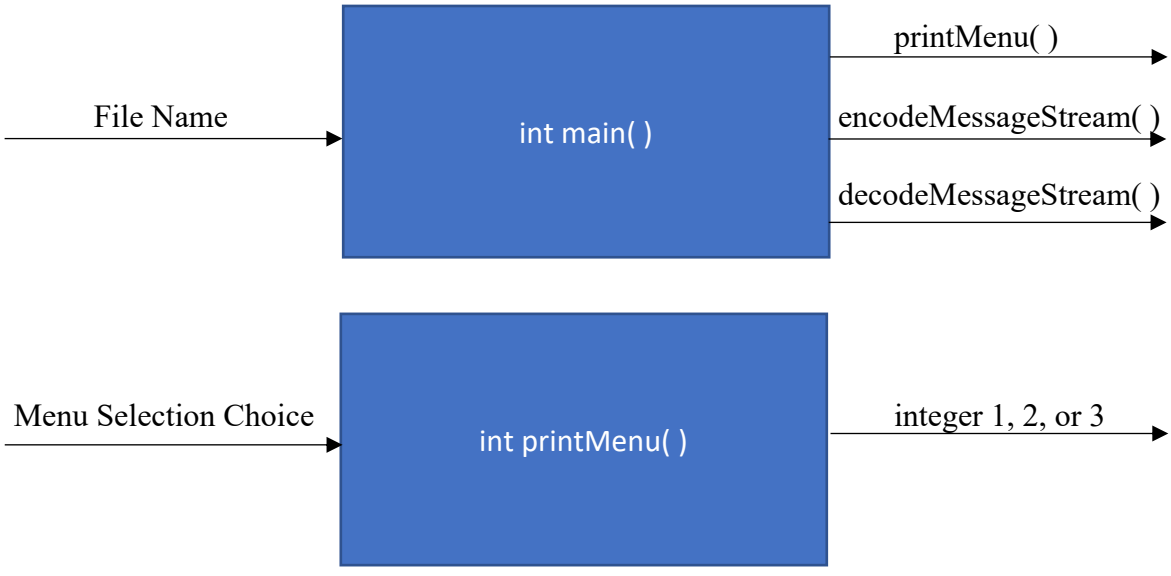
Step 2: Gathering of Information and Input/Output Description:

Relevant Information:
Define symbolic constant for the Length of the message array (a fixed length of 7 for the 4 fixed bits and the 3 parity bits for a message to be encoded or decoded).
Files should always be closed after finishing writing or reading from them.
Dimension of the array should be passed as an argument to a function.
Calculating the parity bits values is done by utilizing the XOR (written as ^) operation. For instance, to calculate p1, this line of code was used:
P1= message[0]^message[2]^message[4]^message[6]
This operation counts the number of 1s within the values of the given array indices. If the number of 1s is odd, then the parity is 1. If they number is odd, then the parity is 0.
After the three parities have been calculated, the binary number generated by p4,p2,p1 is converted to decimal number by using the following equation[1]:

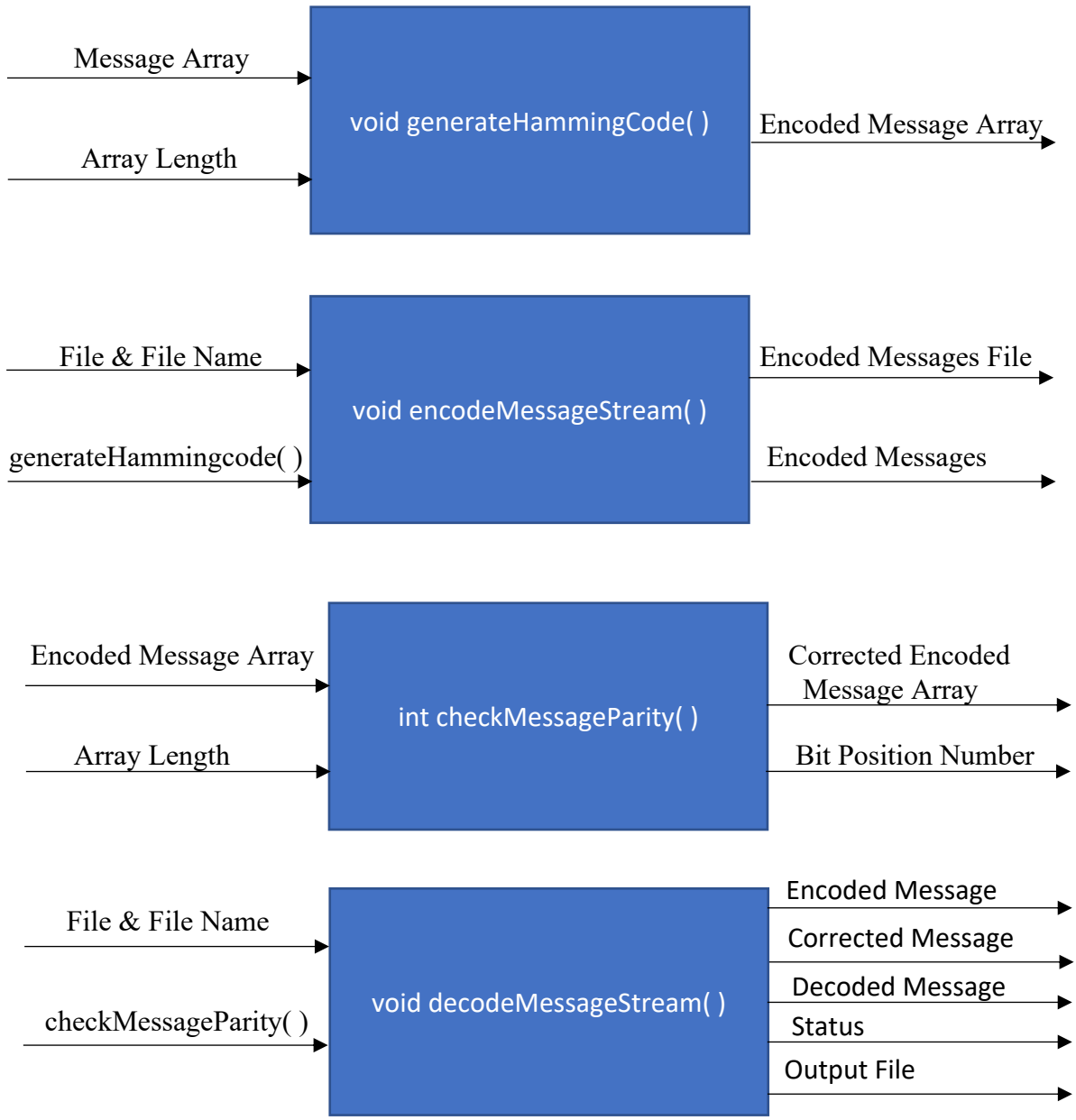
$$p_4 \times 2^2 + p_2 \times 2^1 + p_1 \times 2^0$$

The main function utilizes various functions which are further dependent on other functions; all of those functions will be located in a separate library that is included in the main program.
The files provided and generated for the messages strictly only contain Boolean values and no other characters other than end of line which separates each message.

Input/Output Description:



1"Program for Binary To Decimal Conversion." GeeksforGeeks, www.geeksforgeeks.org/program-binary-decimal-conversion/.



This is presented to the user for input:

```
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
```

If the user chose option 1 or 2, the user is then asked to input the file name:

For choice 1)

```
Please input the name of the file with the messages to encode with the
file extension (e.g. messages.txt):
```

For choice 2)

```
Please input the name of the file with the messages to decoded with the
file extension (e.g. messages.txt):
```

The program outputs the following:

For choice 1)

```
Encoded Message: 0101101
Encoded Message: 1100001
Encoded Message: 1111000
Encoded Message: 1100110
Encoded Message: 1111111
Encoded Message: 1111000
Encoded Message: 1001100
Encoded Message: 0000111
The messages have been successfully encoded; check the
'encodedMessages.txt' file.
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
```

For choice 2)

```
Encoded|Corrected|Decoded|STATUS
0101101|0101101|0101|NO_ERROR
1100001|1100001|1100|NO_ERROR
1111000|1111000|1110|NO_ERROR
1100110|1100110|1101|NO_ERROR
1110111|1111111|1111|ERROR_4
1111000|1111000|1110|NO_ERROR
1001100|1001100|1001|NO_ERROR
0000111|0000111|0001|NO_ERROR
0000111|0000111|0001|NO_ERROR
The messages have been successfully decoded; check the
'decodedMessages.txt' file.
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
```

Finally, the user can exit the program by pressing 3:

```
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
3

(exits program)
```

Step 3: Design of the algorithm and hand-solved problems

Test Case 1: (Menu Selection)

a) "Encode a message" Choice

```
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
1
Please input the name of the file with the messages to encode with the
file extension (e.g. messages.txt):
```

b) "Decode a message" Choice

```
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
2
Please input the name of the file with the messages to decoded with the
file extension (e.g. messages.txt):
```

c) Exiting Program

```
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
3
(exits program)
```

Test Case 2: (Encoding Messages)

a) If input file fails to open

```
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
1
Please input the name of the file with the messages to encode with
the file extension (e.g. messages.txt):
test.txt
Could not open the input file test.txt

(exits program)
```

b) If output file fails to open

```
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
1
Please input the name of the file with the messages to encode with
the file extension (e.g. messages.txt):
SampleMessages.txt
Could not open the output file encodedMessages.txt

(exits program)
```

c) Program is executed successfully

```
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
1
Please input the name of the file with the messages to encode with the
file extension (e.g. messages.txt):
SampleMessages.txt
Encoded Message: 0101101
Encoded Message: 1100001
Encoded Message: 1111000
Encoded Message: 1100110
Encoded Message: 1111111
Encoded Message: 1111000
Encoded Message: 1001100
Encoded Message: 0000111
The messages have been successfully encoded; check the
'encodedMessages.txt' file.
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
```

The generated file should include the following:
0101101
1100001
1111000
1100110
1111111
1111000
1001100
0000111

Test Case 3: (Decoding Messages)

a) If input file fails to open

```
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
2
Please input the name of the file with the messages to decoded with the
file extension (e.g. messages.txt):
test.txt
Could not open the input file test.txt

(exits program)
```

b) If output file fails to open

```
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
```

```
2
Please input the name of the file with the messages to decoded with the
file extension (e.g. messages.txt):
NoErrors.txt
Could not open the output file decodedMessages.txt

(exits program)
```

c) Program is successfully executed with a file with **no** errors in messages

```
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
2
Please input the name of the file with the messages to decoded with the
file extension (e.g. messages.txt):
NoErrors.txt
Encoded|Corrected|Decoded|STATUS
0101101|0101101|0101|NO_ERROR
1100001|1100001|1100|NO_ERROR
1111000|1111000|1110|NO_ERROR
1100110|1100110|1101|NO_ERROR
1111111|1111111|1111|NO_ERROR
1111000|1111000|1110|NO_ERROR
1001100|1001100|1001|NO_ERROR
0000111|0000111|0001|NO_ERROR
The messages have been successfully decoded; check the
'decodedMessages.txt' file.
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
```

The file generated should include the following:

0101101|0101101|0101|NO_ERROR
1100001|1100001|1100|NO_ERROR
1111000|1111000|1110|NO_ERROR
1100110|1100110|1101|NO_ERROR
1111111|1111111|1111|NO_ERROR
1111000|1111000|1110|NO_ERROR
1001100|1001100|1001|NO_ERROR
0000111|0000111|0001|NO_ERROR

d) Program is successfully executed with a file with **one** error in messages

```
1) Encode a message
2) Decode a message
3) Exit
2
Please input the name of the file with the messages to decoded with the
file extension (e.g. messages.txt):
OneError.txt
Encoded|Corrected|Decoded|STATUS
0101101|0101101|0101|NO_ERROR
1100001|1100001|1100|NO_ERROR
```

```
1111000|1111000|1110|NO_ERROR
1100110|1100110|1101|NO_ERROR
1110111|1111111|1111|ERROR_4
1111000|1111000|1110|NO_ERROR
1001100|1001100|1001|NO_ERROR
0000111|0000111|0001|NO_ERROR
0000111|0000111|0001|NO_ERROR
The messages have been successfully decoded; check the
'decodedMessages.txt' file.
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
```

The file generated should include the following:
0101101|0101101|0101|NO_ERROR
1100001|1100001|1100|NO_ERROR
1111000|1111000|1110|NO_ERROR
1100110|1100110|1101|NO_ERROR
1110111|1111111|1111|ERROR_4
1111000|1111000|1110|NO_ERROR
1001100|1001100|1001|NO_ERROR
0000111|0000111|0001|NO_ERROR
0000111|0000111|0001|NO_ERROR

e) Program is successfully executed with a file with errors in **all** messages

```
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
2
Please input the name of the file with the messages to decoded with the
file extension (e.g. messages.txt):
AllErrors.txt
Encoded|Corrected|Decoded|STATUS
1101101|1101101|1101|ERROR_7
1000001|1100001|1100|ERROR_6
1011000|1111000|1110|ERROR_6
1110110|1110110|1111|ERROR_5
1011111|1111111|1111|ERROR_6
1111100|1111100|1111|ERROR_3
1001110|1001110|1001|ERROR_2
0000110|0000111|0001|ERROR_1
The messages have been successfully decoded; check the
'decodedMessages.txt' file.
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
```

The file generated should include the following:
1101101|1101101|1101|ERROR_7
1000001|1100001|1100|ERROR_6
1011000|1111000|1110|ERROR_6
1110110|1110110|1111|ERROR_5
1011111|1111111|1111|ERROR_6

1111100|1111100|1111|ERROR_3
1001110|1001110|1001|ERROR_2
0000110|0000111|0001|ERROR_1

Algorithm Design:

For this software, a library called “hamming.h” is created which includes the code of all the functions used within the main function.

Main:

import hamming.h library

Main() Function

Assign 0 to y

Repeat
{
Call printMenu function and assign output value to y

(if y = 1)
{
Print “Please input the name of the file with the messages to encode with the file extension (e.g. messages.txt):”, newline
Read value into fname
Call encodeMessageStream() function with the argument fname
Print “The messages have been successfully encoded; check the 'encodedMessages.txt' file.”, newline
}

(if y = 2)
{
Print “Please input the name of the file with the messages to decoded with the file extension (e.g. messages.txt):”, newline
Read value into fname
Call decodeMessageStream() function with the argument fname
Print “The messages have been successfully decoded; check the 'decodedMessages.txt' file.”, newline
}
}
WHILE y is not equal to 3

(exits program)

Hamming.h Library

Assign 7 to constant LENGTH

printMenu() Function

Assign 0 to n
Assign 0 to x

Repeat
{
(if user made an invalid choice), *this can be done by assigning zero to x in the first do/while loop, making the condition of the if statement (x==1) and assigning 1 to x at the end of* ***this*** *do/while loop*)
{
Print “Invalid Choice.”, newline, “Please input a valid selection from the menu.”, newline
}
Print “Please select an option from the following menu:”, newline
Print “1) Encode a message”, newline
Print “2) Decode a message”, newline
Print “3) Exit”, newline
Read to n
}
WHILE n is less than 1 or greater than 3

Return n

generateHammingCode() Function with first parameter [boolean array “message”] and second parameter [length of array]

Assign 0 to i

While (i < LENGTH)
{
(if i=6)
i’th index of message array is 1 if the number of 1s in 0, 2 and 4 indices is odd and 0 if the number of 1s is even
(if i=5)
i’th index of message array is 1 if the number of 1s in 0, 1 and 4 indices is odd and 0 if the number of 1s is even
(if i=3)
i’th index of message array is 1 if the number of 1s in 0, 1 and 2 indices is odd and 0 if the number of 1s is even

increment i by 1
}

encodeMessageStream() Function with one parameter [string "file"]

```
Declare a Boolean array with size equal to LENGTH

Open file "file" for reading as fin
(if cannot open file "file")
{
        Print Error "Could not open the input file 'file'.", newline
        Exit program
}

Open file "encodedMessages.txt" for writing as fout
(if cannot open file "encodedMessages.txt")
{
        Print Error "Could not open the input file encodedMessages.txt.", newline
        Exit program
}

Assign 1 to i
While ( i is less than or equal to LENGTH )
{
        (if end of file is reached)
                Break loop
        (if i is equal to 3, 5 or 6)
                Start next iteration of the loop and increment i by 1

        Read character from the file "file" and assign it to ch
        (if ch is '0')
                Assign 0 to the i'th index of "message" array
        (else if ch is '1')
                Assign 1 to the i'th index of "message" array
        Else
        {
                Read character from the file "file"
                Assign -1 to i
                Call the generateHammingCode() function with the first argument [
                "message" array ] and second argument [ LENGTH ]
                Print "Encoded message: "

                Assign 0 to x
                While (x<LENGTH)
                {
                        Print x'th index of "message" array
                        Write x'th index of "message" array to encodedMessages.txt
                }
                Print newline
                Write newline to the file encodedMessages.txt
        }

        Close the file "file"
```

Close the file “encodedMessages.txt”
Increment i by 1
}

checkMessageParity() Function with first parameter [Boolean array “encodedmessage”] and second parameter [length of array]

Assign 1 to p1 if the number of 1s in the indices 0, 2, 4 and 6 of the array “encodedmessage” is odd and assign it to 0 if it’s even
Assign 1 to p2 if the number of 1s in the indices 0, 1, 4 and 5 of the array “encodedmessage” is odd and assign it to 0 if it’s even
Assign 1 to p4 if the number of 1s in the indices 0, 1, 2 and 3 of the array “encodedmessage” is odd and assign it to 0 if it’s even

(if p1=0 and p2=0 and p4=0)
Assign -1 to bit_number

Else
{
Assign “p4*2^2 + p2*2 + p1*1” to bit_number
Assign “LENGTH – bit_number” to i

(if the i’th index of “encodedmessage” array is equal to 1, then assign 0 to it instead)
(if the i’th index of “encodedmessage” array is equal to 0, then assign 1 to it instead)
}

Return bit_number

decodeMessageStream() Function with one parameter [string “file”]

Open file “file” for reading as fin
(if cannot open file “file”)
{
Print Error “Could not open the input file ‘file’.”, newline
Exit program
}

Open file “decodedMessages.txt” for writing as fout
(if cannot open file “encodedMessages.txt”)
{
Print Error “Could not open the input file decodedMessages.txt.”, newline
Exit program
}

Print “Encoded|Corrected|Decoded|STATUS”, newline

```
While (end of file is not reached)
{
        Declare a Boolean array “encodedmessage” with size LENGTH

        Assign 0 to x
        While (x is less than or equal to LENGTH)
        {
                Read character from the file “file” and assign it to ch
                (if ch is ‘0’)
                        Assign 0 to x’th index of “encodedmessage” array
                (if ch is ‘1’)
                        Assign 1 to x’th index of “encodedmessage” array
                (if ch is end of line character)
                        Break loop

                Increment x by 1
        }

        Assign 0 to y
        While (y < LENGTH)
        {
                Print y’th index of “encodedmessage” array
                Write y’th index of “encodedmessage” array to the file
                “decodedMessages.txt”
                Increment y by 1
        }

        Print “|”
        Write “|” to the file “decodedMessages.txt”
        Call checkMessageParity() Function with the first argument
        [“encodedmessage”] array and second argument [ LENGTH of array ] and
        assign the output value to bit_number

        Assign 0 to z
        While (z<LENGTH)
        {
                Print y’th index of “encodedmessage” array
                Write y’th index of “encodedmessage” array to the file
                “decodedMessages.txt”
                Increment z by 1
        }

        Print “|”
        Write “|” to the file “decodedMessages.txt”

        Assign 0 to a
        While (a<LENGTH)
        {
                (if a=6 or a=5 or a=3)
                        Start next iteration of the loop and increment a by 1
```

```
                Print a'th index of "encodedmessage" array
                White a'th index of "encodedmessage" array to the file
                "decodedMessages.txt"
                Increment a by 1
        }

        (if bit_number is equal to -1)
        {
                Print "|NO_ERROR", newline
                Write "|NO_ERROR", newline to the file "decodedMessages.txt"
        }
        Else
        {
                Print "|ERROR_", bit_number, newline
                Write "|ERROR_", bit_number, newline to the file
                "decodedMessages.txt"
        }
Close input file "file"
Close output file "decodedMessages.txt"
}
```

Step 4: Implementation

Main Function

```
//
//  main.cpp
//  Assignment 2
//
//  Description: Computer Engineering Case Study
//  Created by Muaath Asali on 10/18/19.
//  Copyright © 2019 Muaath Asali. All rights reserved.
//

#include <iostream>
#include <fstream>
#include "hamming.h"
using namespace std;



int main() {
    int y=0;
    string fname;

    do
    {

    y=printMenu();

    if (y==1)
    {
              cout << "Please input the name of the file with the messages to
               encode with the file extension (e.g. messages.txt):" << endl;
        cin >> fname;
        encodeMessageStream(fname);
        cout << "The messages have been successfully encoded; check the
         'encodedMessages.txt' file." << endl;
    }

    if (y==2)
    {
              cout << "Please input the name of the file with the messages to
               decoded with the file extension (e.g. messages.txt):" << endl;
        cin >> fname;
        decodeMessageStream(fname);
        cout << "The messages have been successfully decoded; check the
         'decodedMessages.txt' file." << endl;
    }

    }
    while (y!=3);

    return 0;
}
```

Library "hamming.h"

```
#ifndef hamming_h
#define hamming_h
#include <iostream>
#include <fstream>
#define LENGTH 7
using namespace std;


int printMenu() {
    int n=0, x=0;
    do
    {
        if (x==1) // this makes it so that the invalid choice message is
         only displayed after incorrect try
        {
            cout << "Invalid Choice." << endl;
            cout << "Please input a valid selection from the menu." << endl;
        }

        cout << "Please select an option from the following menu:" << endl;
        cout << "1) Encode a message" << endl;
        cout << "2) Decode a message" << endl;
        cout << "3) Exit" << endl; cin >> n;

        x=1;
    }
    while (n<1 || n>3);

    return (n);
}


void generateHammingCode(bool message[], int n) {
    for (int i=0; i<=n; i++)
    {
        if(i==6)
            message[i] = message[4]^message[2]^message[0];
        if(i==5)
            message[i] = message[4]^message[1]^message[0];
        if(i==3)
            message[i] = message[2]^message[1]^message[0];
    }
}


void encodeMessageStream(string file) {
    bool message[LENGTH];
    char ch;

    ifstream fin(file);

    if(fin.fail())
    {
        cerr << "Could not open the input file " << file << endl;
```

```
        exit(1);
    }

    ofstream fout("encodedMessages.txt");

    if(fout.fail())
    {
    cerr << "Could not open the output file encodedMessages.txt" << endl;
    exit(1);
    }

    for (int i=0; i<=LENGTH; i++)
    {

        if(fin.eof())
            break;

        if (i==6 || i==5 || i==3) // those indices are left empty for the
         parity values to occupy them later
            continue;

        ch = fin.get();

        if (ch=='0')
            message[i] = 0;
        else if (ch=='1')
            message[i] = 1;
        else // After the program reads end of line, it does all
         processing necessary for 1 line and resets the loop
        {
            fin.get();
            i = -1; // Causes loop to restart with i=0
            generateHammingCode(message, LENGTH);
            cout << "Encoded Message: ";

            for (int x=0; x<LENGTH; x++)
            {
                cout << message[x];
                fout << message[x];
            }
            cout << endl;
            fout << endl;
        }

    }

    fin.close();
    fout.close();

}


int checkMessageParity(bool encodedmessage[], int n) {
    int p1, p2, p4, bit_num, i;
```

```
    p1 =
     encodedmessage[0]^encodedmessage[2]^encodedmessage[4]^encodedmessage
     [6];
    p2 =
     encodedmessage[0]^encodedmessage[1]^encodedmessage[4]^encodedmessage
     [5];
    p4 =
     encodedmessage[0]^encodedmessage[1]^encodedmessage[2]^encodedmessage
     [3];

    if (p1==0 && p2==0 && p4==0)
        bit_num = -1; // indicates no error found

    else
    {
    bit_num = (p4*2*2 + p2*2 + p1*1);
    i = n - bit_num;

    if (encodedmessage[i]==1)
        encodedmessage[i] = 0;
    if (encodedmessage[i]==0)
        encodedmessage[i] = 1;
    }

    return bit_num;
}


void decodeMessageStream(string file) {

    ifstream fin(file);

    if(fin.fail())
    {
        cerr << "Could not open the input file " << file << endl;
        exit(1);
    }

    ofstream fout("decodedMessages.txt");

       if(fout.fail())
       {
       cerr << "Could not open the output file decodedMessages.txt" << endl;
       exit(1);
       }

    cout << "Encoded|Corrected|Decoded|STATUS" << endl;

    while(!fin.eof())
    {
        bool encodedmessage[LENGTH];
        int bit_num;

        for (int x=0; x<=LENGTH; x++) //reads the original encoded message
        {
```

```
            char ch;

            ch = fin.get();

            if (ch=='0')
                encodedmessage[x] = 0;
            if (ch=='1')
                encodedmessage[x] = 1;
            if (ch=='\n')
                break;
        }

        for (int y=0; y<LENGTH; y++) // prints the original encoded message
        {
            cout << encodedmessage[y];
            fout << encodedmessage[y];
        }

        cout << "|";
        fout << "|";
        bit_num = checkMessageParity(encodedmessage, LENGTH);

        for (int z=0; z<LENGTH; z++) // prings the corrected encoded message
        {
            cout << encodedmessage[z];
            fout << encodedmessage[z];
        }

        cout << "|";
        fout << "|";

        for (int a=0; a<LENGTH; a++) // prints the 4-bit message
        {
            if (a==6 || a==5 || a==3)
                continue;
            cout << encodedmessage[a];
            fout << encodedmessage[a];
        }
        if (bit_num==-1)
        {
            cout << "|NO_ERROR" << endl;
            fout << "|NO_ERROR" << endl;
        }
        else
        {
            cout << "|ERROR_" << bit_num << endl;
            fout << "|ERROR_" << bit_num << endl;
        }
    }
    fin.close();
    fout.close();
}

#endif /* hamming_h *
```

Step 5: Software testing and verification

Test Case 1: (Menu Selection)

a) "Encode a message" Choice

```
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
1
Please input the name of the file with the messages to encode with the file
    extension (e.g. messages.txt):
```

b) "Decode a message" Choice

```
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
2
Please input the name of the file with the messages to decoded with the file
    extension (e.g. messages.txt):
```

c) Exiting Program

```
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
3
Program ended with exit code: 0
```

Test Case 2: (Encoding Messages)

a) If input file fails to open

```
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
1
Please input the name of the file with the messages to encode with the file
    extension (e.g. messages.txt):
test.txt
Could not open the input file test.txt
Program ended with exit code: 1
```

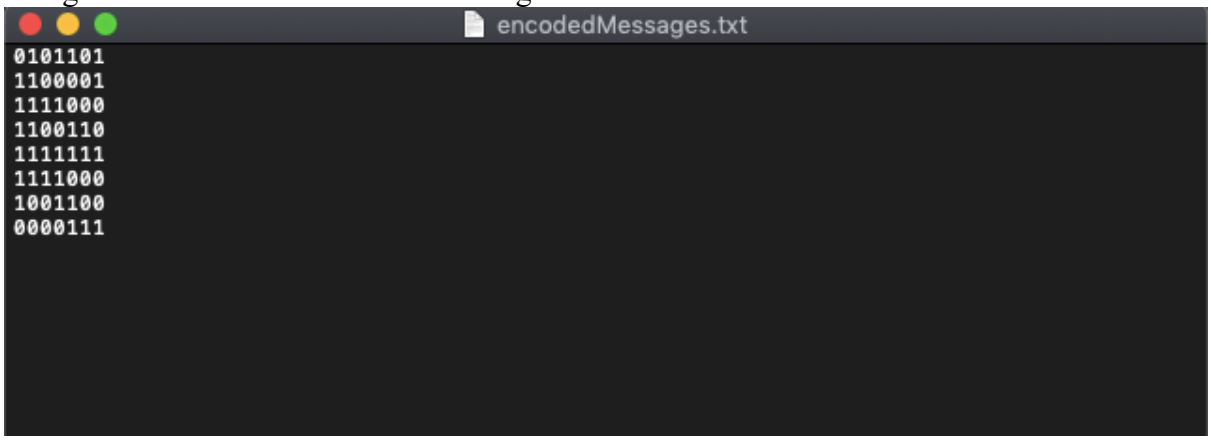
b) If output file fails to open

I couldn't physically prove this test case because this code comes after the input file code, which means that the input file code should be successful in order to be able to test the output file code. However, since both output and input share same file directory, if the input file code works, the output file code will work too.
Both "if fail statements" of input and output file codes are the same. Therefore, it is safe to assume that this test case has been proven too by proving the previous one shown.

c) Program is executed successfully

```
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
1
Please input the name of the file with the messages to encode with the file
    extension (e.g. messages.txt):
SampleMessages.txt
Encoded Message: 0101101
Encoded Message: 1100001
Encoded Message: 1111000
Encoded Message: 1100110
Encoded Message: 1111111
Encoded Message: 1111000
Encoded Message: 1001100
Encoded Message: 0000111
The messages have been successfully encoded; check the 'encodedMessages.txt'
    file.
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
```

The generated file includes the following:



Test Case 3: (Decoding Messages)

a) If input file fails to open

```
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
2
Please input the name of the file with the messages to decoded with the file
    extension (e.g. messages.txt):
test.txt
Could not open the input file test.txt
Program ended with exit code: 1
```

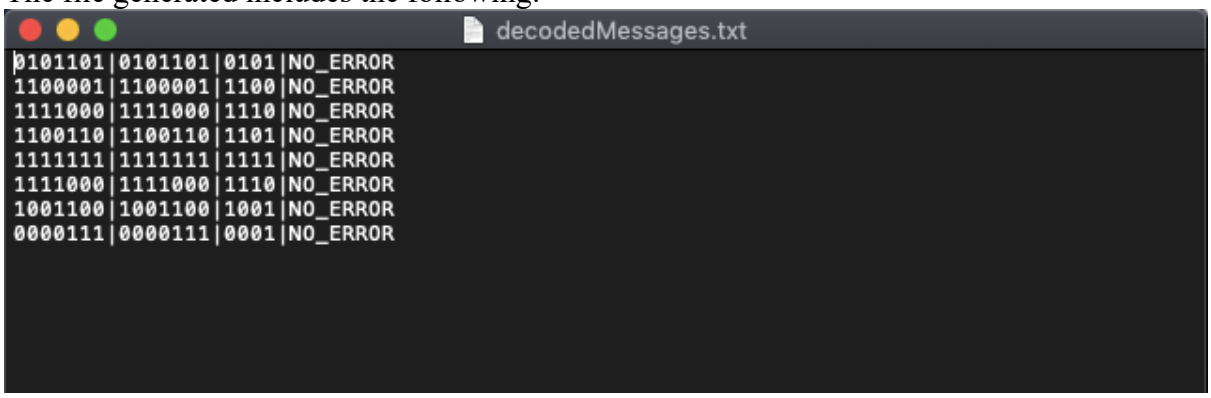
b) If output file fails to open

I couldn't physically prove this test case because this code comes after the input file code, which means that the input file code should be successful in order to be able to test the output file code. However, since both output and input share same file directory, if the input file code works, the output file code will work too.
Both "if fail statements" of input and output file codes are the same. Therefore, it is safe to assume that this test case has been proven too by proving the previous one shown.

c) Program is successfully executed with a file with **no** errors in messages

```
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
2
Please input the name of the file with the messages to decoded with the file
    extension (e.g. messages.txt):
NoErrors.txt
Encoded|Corrected|Decoded|STATUS
0101101|0101101|0101|NO_ERROR
1100001|1100001|1100|NO_ERROR
1111000|1111000|1110|NO_ERROR
1100110|1100110|1101|NO_ERROR
1111111|1111111|1111|NO_ERROR
1111000|1111000|1110|NO_ERROR
1001100|1001100|1001|NO_ERROR
0000111|0000111|0001|NO_ERROR
The messages have been successfully decoded; check the 'decodedMessages.txt'
    file.
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
```

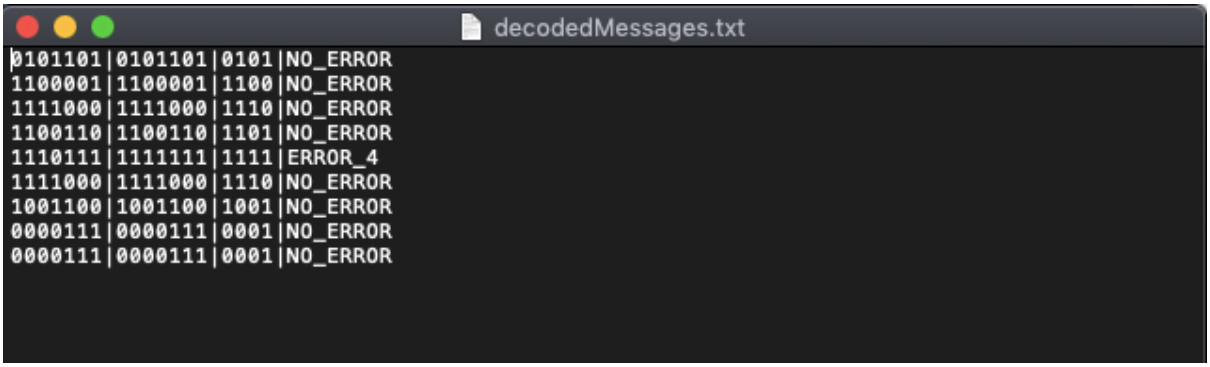
The file generated includes the following:



d) Program is successfully executed with a file with **one** error in messages

```
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
2
Please input the name of the file with the messages to decoded with the file
    extension (e.g. messages.txt):
OneError.txt
Encoded|Corrected|Decoded|STATUS
0101101|0101101|0101|NO_ERROR
1100001|1100001|1100|NO_ERROR
1111000|1111000|1110|NO_ERROR
1100110|1100110|1101|NO_ERROR
1110111|1111111|1111|ERROR_4
1111000|1111000|1110|NO_ERROR
1001100|1001100|1001|NO_ERROR
0000111|0000111|0001|NO_ERROR
0000111|0000111|0001|NO_ERROR
The messages have been successfully decoded; check the 'decodedMessages.txt'
    file.
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
```

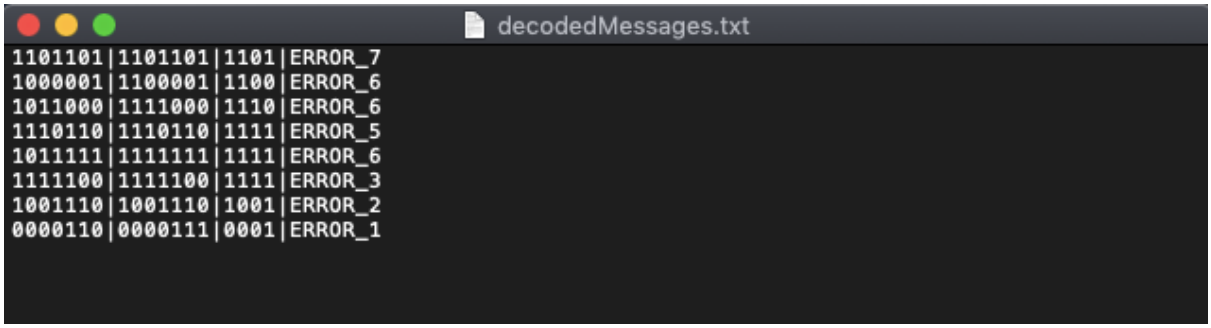
The file generated included the following:



e) Program is successfully executed with a file with errors in **all** messages

```
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
2
Please input the name of the file with the messages to decoded with the file
    extension (e.g. messages.txt):
AllErrors.txt
Encoded|Corrected|Decoded|STATUS
1101101|1101101|1101|ERROR_7
1000001|1100001|1100|ERROR_6
1011000|1111000|1110|ERROR_6
1110110|1110110|1111|ERROR_5
1011111|1111111|1111|ERROR_6
1111100|1111100|1111|ERROR_3
1001110|1001110|1001|ERROR_2
0000110|0000111|0001|ERROR_1
The messages have been successfully decoded; check the 'decodedMessages.txt'
    file.
Please select an option from the following menu:
1) Encode a message
2) Decode a message
3) Exit
```

The file generated included the following:



User Guide:

This software serves as a hamming code that encodes messages to be sent over unreliable networks where some data might be lost and decodes messages in addition to correcting them if some data was lost.

The software presents the user with a set selection menu to either encode a message, decode a message or exit the program, and asks the user to input their selection. Then, it asks the user to input the name of the file with the messages.

If the user chose to encode, the software embeds parity bits into the message and generates a file containing the encoded messages.

If the user chose to decode, the software identifies each message, tells the user whether the message had an error, corrects the error, provides the bit position number the error was located in and prints the original decoded message too. Additionally, it outputs all of that information into a generated file.

Finally, the user can choose to run the software again to encode or decode, or could simply exit the program by pressing 3.

Bibliography:

"Program for Binary To Decimal Conversion." GeeksforGeeks, www.geeksforgeeks.org/program-binary-decimal-conversion/.