CQUPT – University at Albany

Computer Science – International College

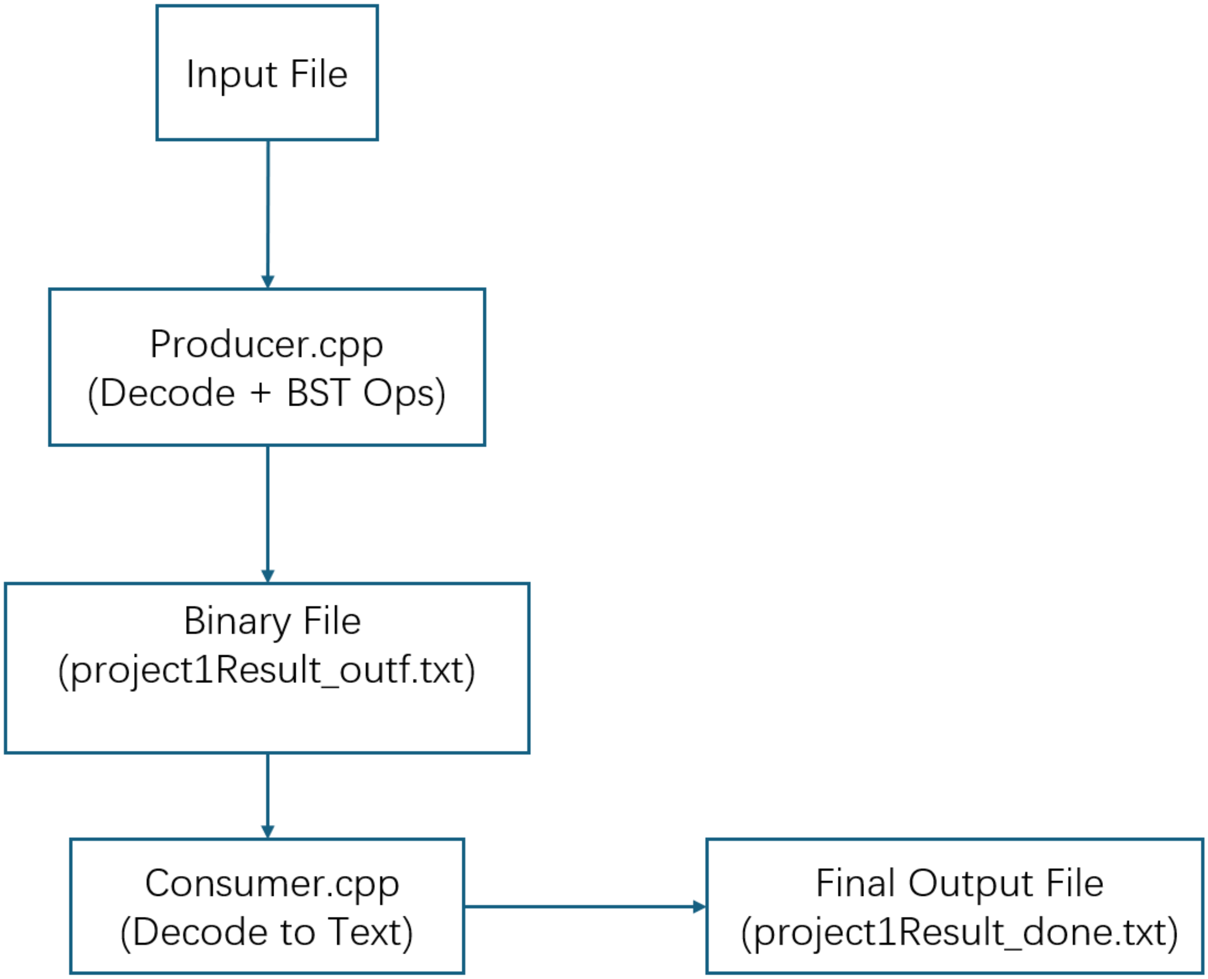
ICSI 403 --- Design and Analysis of Algorithms

Project 1 --- Spring 2025

Table of Contents

I. System documentation

i. A high-level data flow diagram for the system



ii. A list of routines and their brief descriptions

Routine	**Description**
encodeWithParity	Encodes a character with odd parity.
encodeInputFile	Encodes a text file into a binary file.
decodeWithParity	Decodes an 8-bit binary string with odd parity.
decodeResultFile	Decodes a binary file into a text file.
Symbols::performOperation	Performs arithmetic operations on a symbol's value.
Symbols::toString	Converts a symbol's identifier and value to a string.
BinarySearchTree::insert	Inserts a symbol into the BST.
BinarySearchTree::find	Finds a symbol in the BST by its identifier.
BinarySearchTree::inorder	Performs an in-order traversal of the BST to extract sorted symbols.
BinarySearchTree::getAllSymbolsSorted	Returns all symbols in the BST sorted lexicographically.
processFile	Processes a binary file and updates the BST.

iii. Binary Search Tree

a. The BST node structure is defined as follows:

```
struct TreeNode
{
    Symbols data;    // Stores the symbol (identifier and value)
    TreeNode *left;  // Left child node
    TreeNode *right; // Right child node

    TreeNode(Symbols sym) : data(sym), left(nullptr), right(nullptr) {}
};
```

- Each node stores a Symbols object, which contains an identifier (identifier) and a value (value).
- The left and right child nodes are used to maintain the BST structure.

b. BST Class

The BST class encapsulates operations such as insertion, search, and traversal:

```
class BinarySearchTree
{
private:
    TreeNode *root; // Root node of the BST

    // Helper function to insert a node
    TreeNode *insert(TreeNode *node, Symbols sym)
    {
        if (node == nullptr)
            return new TreeNode(sym); // If the node is null, create a new node

        // Insert into the left or right subtree based on the identifier
        if (sym.identifier < node->data.identifier)
            node->left = insert(node->left, sym);
        else if (sym.identifier > node->data.identifier)
            node->right = insert(node->right, sym);

        return node; // Return the updated node
    }

    // Helper function to find a node by identifier
    TreeNode *find(TreeNode *node, const std::string &id)
    {
        if (node == nullptr || node->data.identifier == id)
```

```
            return node; // If node is found or null, return it

        // Search in the left or right subtree based on the identifier
        if (id < node->data.identifier)
            return find(node->left, id);
        else
            return find(node->right, id);
    }

    // Helper function for in-order traversal (to extract sorted symbols)
    void inorder(TreeNode *node, std::vector<std::string> &symbolsList) const
    {
        if (node != nullptr)
        {
            inorder(node->left, symbolsList);             // Traverse left subtree
            symbolsList.push_back(node->data.toString()); // Add current node's data to
the list
            inorder(node->right, symbolsList);            // Traverse right subtree
        }
    }

public:
    BinarySearchTree() : root(nullptr) {} // Constructor initializes root to null

    // Public function to insert a symbol into the BST
    void insert(Symbols sym)
    {
        root = insert(root, sym); // Call the private insert function
    }

    // Public function to find a symbol by identifier
    Symbols *find(const std::string &id)
    {
        TreeNode *node = find(root, id); // Call the private find function
        if (node != nullptr)
            return &node->data; // Return the symbol if found
        return nullptr;         // Return null if not found
    }

    // Public function to get all symbols sorted lexicographically
    std::vector<std::string> getAllSymbolsSorted() const
    {
        std::vector<std::string> symbolsList;
        inorder(root, symbolsList);                        // Perform in-order
traversal to extract symbols
        std::sort(symbolsList.begin(), symbolsList.end()); // Sort the symbols
lexicographically
```

```
        return symbolsList;                                 // Return the sorted list
    }
};
```

b. How BST Used in the Producer

1. **Processing Algebraic Expressions**:
   - The processFile function reads the decoded algebraic expressions from the binary file.
   - For each expression, it either:
     1 Finds the symbol in the BST using find and updates its value using performOperation.
     2 If the symbol does not exist, it creates a new Symbols object, performs the operation, and inserts it into the BST using insert.

```
void processFile(const std::string &filename, BinarySearchTree &bst)
{
    // Decode the binary file into a temporary text file
    string outputFile = "temp1.txt";
    decodeResultFile(filename, outputFile);

    std::ifstream file(outputFile);
    if (!file.is_open())
    {
        std::cerr << "Failed to open file: " << filename << std::endl;
        return;
    }

    std::string line;
    while (std::getline(file, line)) // Read each line
    {
        std::istringstream iss(line);
        std::string identifier, operation;
        int value;

        iss >> identifier >> operation >> value; // Parse the line

        Symbols *sym = bst.find(identifier); // Find the symbol in the BST
        if (sym)
            sym->performOperation(operation, value); // Perform the operation if found
        else
        {
            Symbols newSym(identifier, 0);             // Create a new symbol with initial value 0
            newSym.performOperation(operation, value); // Perform the operation
            bst.insert(newSym);                        // Insert the new symbol into the BST
        }
    }

    file.close();
}
```

2. **Sorting and Output**:

1.After processing all expressions, the getAllSymbolsSorted function is called to retrieve all symbols in lexicographical order.

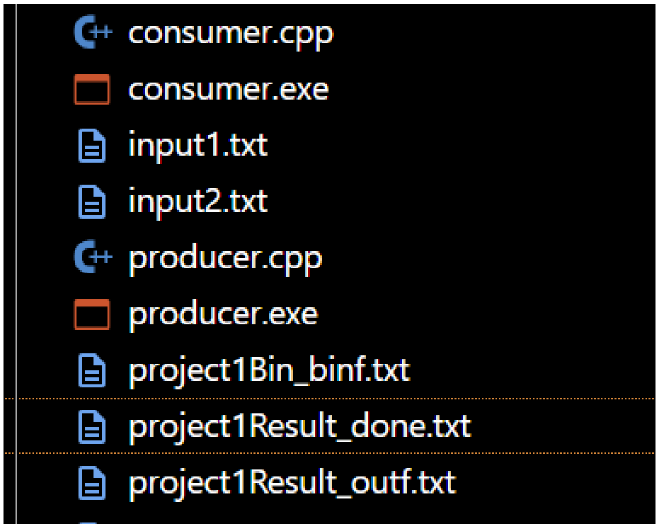
2.The sorted symbols are written to a file and encoded into the final binary file (project1Result.outf).

```
void writeSortedResultsToFile(const vector<string> &sortedSymbols)
{
    string outputFile = "temp2.txt";
    ofstream outFile(outputFile);

    if (!outFile.is_open())
    {
        cerr << "Failed to open file: " << outputFile << endl;
        return;
    }

    for (const auto &sym : sortedSymbols)
        outFile << sym << endl; // Write sorted symbols to file
    outFile.close();

    encodeInputFile(outputFile, "project1Result_outf.txt"); // Encode the results into a binary file
}
```

II Test documentation

i. How you tested your program

I execute my program in Vscode, the program structure as follows

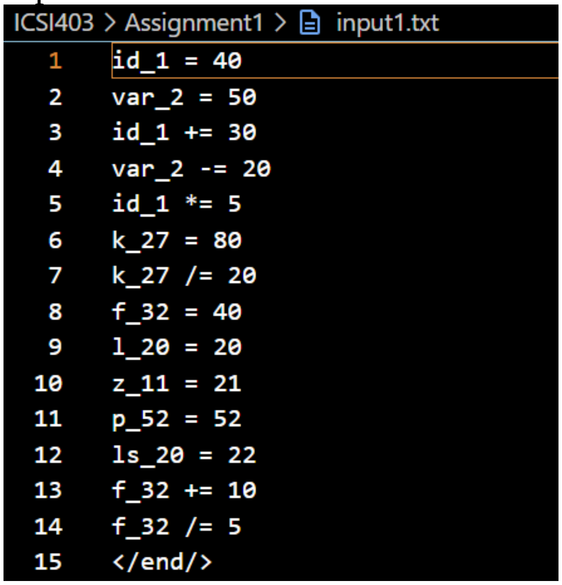


```
consumer.cpp
consumer.exe
input1.txt
input2.txt
producer.cpp
producer.exe
project1Bin_binf.txt
project1Result_done.txt
project1Result_outf.txt
```

I have four test files to test the program, after getting the results, I compare them with the answer which is calculated by myself.

ii. Testing outputs

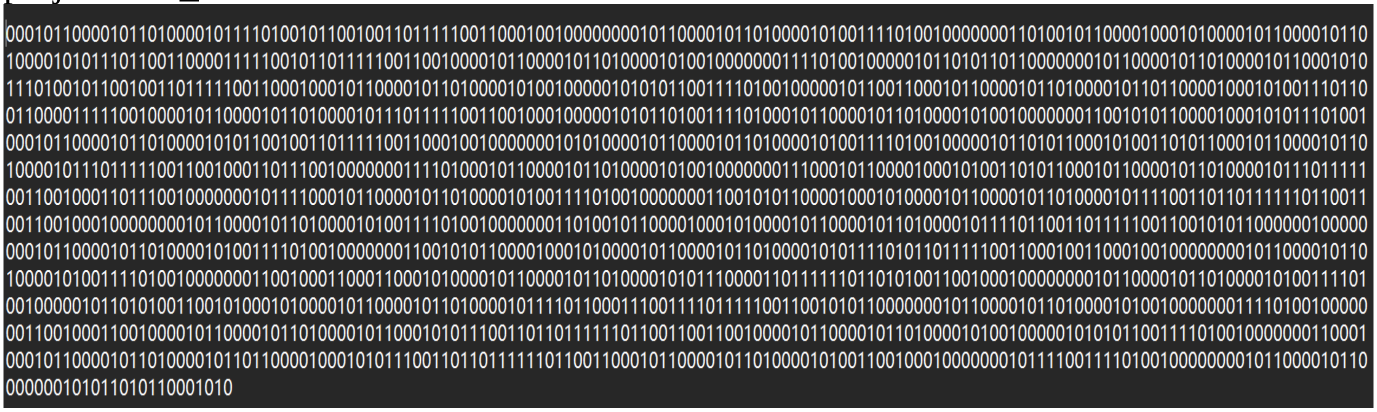
Input1.txt



ICSI403 > Assignment1 > input1.txt

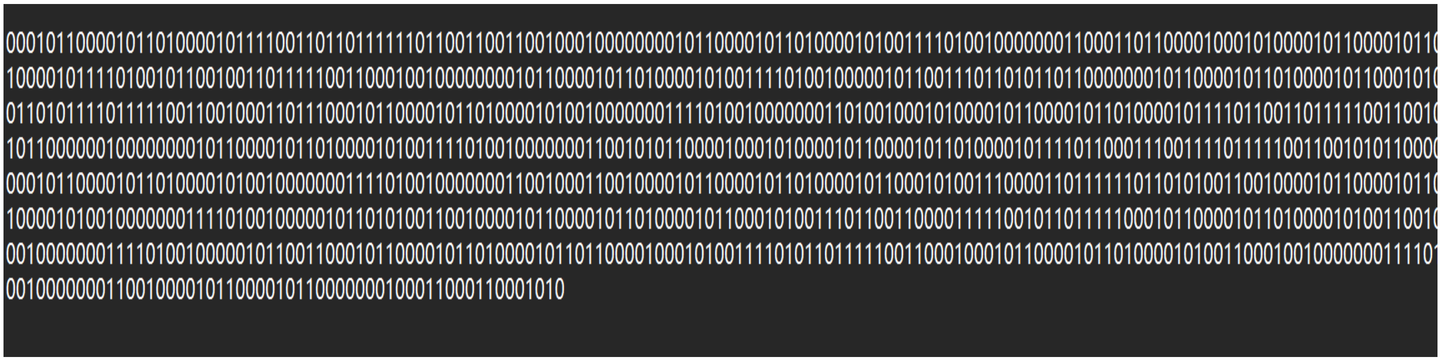
```
id_1 = 40
var_2 = 50
id_1 += 30
var_2 -= 20
id_1 *= 5
k_27 = 80
k_27 /= 20
f_32 = 40
l_20 = 20
z_11 = 21
p_52 = 52
ls_20 = 22
f_32 += 10
f_32 /= 5
</end/>
```

project1Bin_binf.txt



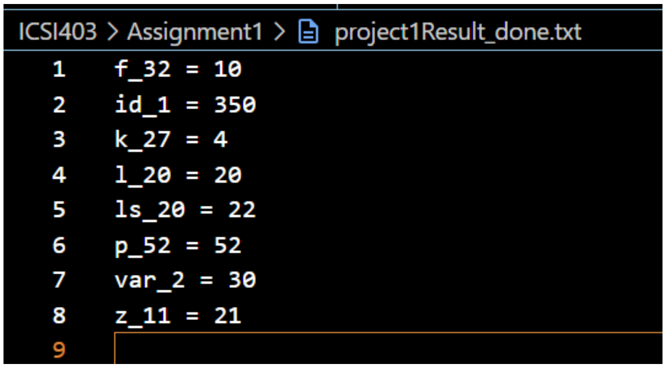
```
00010110000101101000010111101001011001001101111110011000100100000000010110000101101000010100111101001000000011010010110000100010100001011000010110
10000101011101100110000111110010110111111001100100001011000010110100001010010000000011110100100000101101011011000000010110000101101000010110001010
11101001011001001101111100110001000101100001011010000101001000001010101100111101001000001011001100010110000101101000010110110000100010100111011 0
01100001111100100001011000010110100001011101111110011001000100000101011010011110100010110000101101000010100100000000110010101100001000101011101001
00010110000101101000010101100100110111111001100010010000000010101000010110000101101000010100111101001000001011010110001010011010110001011000010110
100001011101111110011001000110111001000000001111010001011000010110100001010010000000011100010110000100010100110101100010110000101101000010111011111
0011001000110111001000000001011110001011000010110100001010011110100100000001100101011000010001010000101100001011010000101111001101101111110110011
001100100010000000001011000010110100001010011110100100000000110100101100001000101000010110000101101000010111101100110111111001100101011000000100000
000101100001011010000101001111010010000000011001010110000100010100001011000010110100001010111101011011111100110001001100010010000000001011000010110
10000101001111010010000000011001000110001100010100001011000010110100001010111000011011111101101010011001000100000000010110000101101000010100111101
00100000101101010011001010001010000101100001011010000101111011000111001111011111100110010101100000001011000010110100001010010000000011110100100000
001100100011001000010110000101101000010110001010111001101101111110110011001100100001011000010110100001010010000010101011001111010010000000110001
00010110000101101000010110110000100010101110011011011111101100110001011000010110100001010011001000100000001011110011110100100000000010110000101 10
000000101011010110001010
```

project1Result_outf.txt



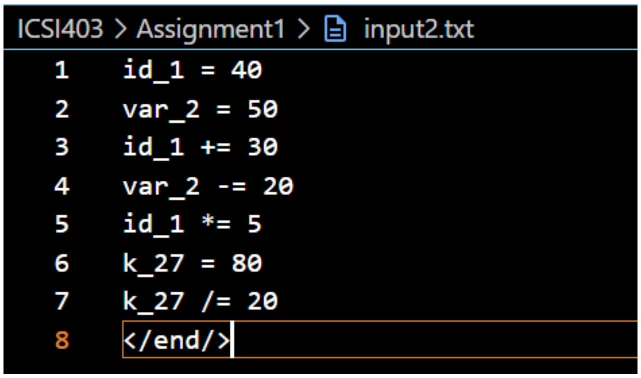
```
00010110000101101000010111100110110111111101100110011001000100000000010110000101101000010100111101001000000001100011011000010001010000101100001011 0
10000101111010010110010011011111100110001001000000000101100001011010000101001111010010000010110011101101011011000000010110000101101000010110001010
01101011110111111001100100011011100010110000101101000010100100000000111101001000000011010010001010000010110000101101000010111101100110111111001100 10
101100000001000000000101100001011010000101001111010010000000011001010110000100010100001011000010110100001011110110001110011110111111001100101011000 0
00010110000101101000010100100000000111101001000000011001000110010000101100001011010000101100010100111000011011111101101010011001000010110000101 10
10000101001000000001111010010000010110101001100100001011000010110100001011000101001110110011000011111001011011111000101100001011010000101001100 10
00100000001111010010000010110011000101100001011010000101101100001000101001111010110111111001100010001011000010110100001010011000100100000000111101
0010000000011001000010110000101100000000100011000110001010
```

Result



```
f_32 = 10
id_1 = 350
k_27 = 4
l_20 = 20
ls_20 = 22
p_52 = 52
var_2 = 30
z_11 = 21

```

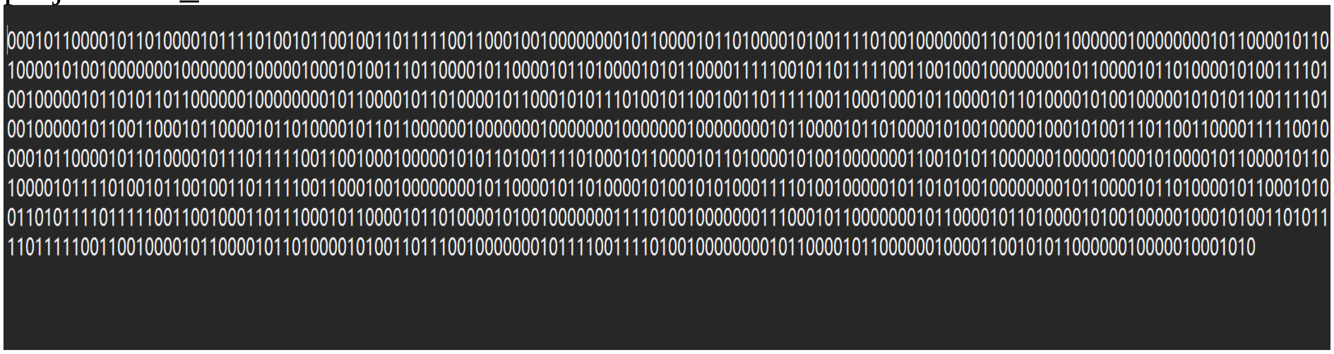
Input2.txt



ICSI403 > Assignment1 > input2.txt

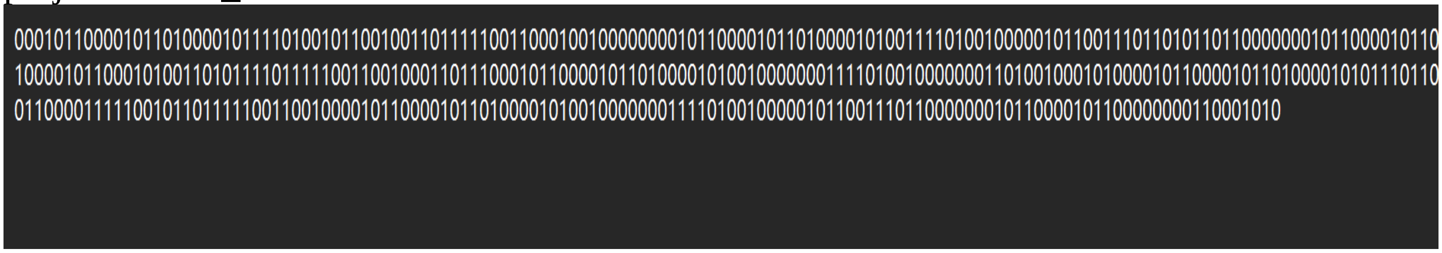
```
id_1 = 40
var_2 = 50
id_1 += 30
var_2 -= 20
id_1 *= 5
k_27 = 80
k_27 /= 20
</end/>
```

project1Bin_binf.txt



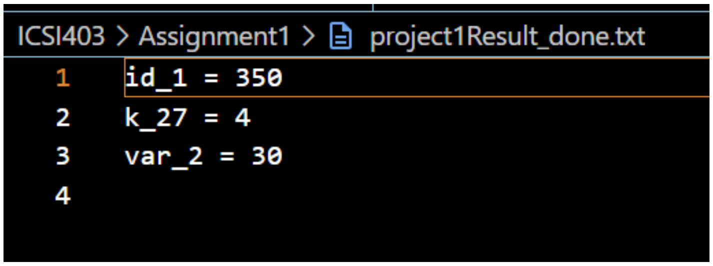
```
000101100001011010000101111010010110010011011111100110001001000000000101100001011010000101001111010010000000110100101100000010000000010110000101101
10000101001000000001000000010000010001010011101100001011000010110100001010110000111110010110111110011001000100000000010110000101101000010100111101
0010000001011010110110000000100000000010110000101101000010110001010111010010110010011011111100110001000101100001011010000101001000001010101100111101
001000000101100110001011000010110100001011011000000010000000100000001000000010000000001011000010110100001010010000010001010011101100110000111110010
000101100001011010000101110111111001100100010000010101101001111010001011000010110100001010010000000110010101100000010000010001010000101100001 0110
1000010111101001011001001101111110011000100100000000101100001011010000101001010100011110100100000101101010010000000001011000010110100001011000101 0
0110101111011111100110010001101110001011000010110100001010010000000111101001000000011100010110000000101100001011010000101001000001000101001101011
110111111001100100001011000010110100001010011011100100000000101111001111010010000000001011000010110000000100001100101011000000010000010001010
```

project1Result_outf.txt



```
00010110000101101000010111101001011001001101111110011000100100000000010110000101101000010100111101001000001011001110110101101100000001011000010110
100001011000101001101011110111110011001000110111000101100001011010000101001000000011110100100000001101001000101000010110000101101000010101110110
01100001111100101101111100110010000101100001011010000101001000000001111010010000010110011101100000000101100001011000000000110001010
```

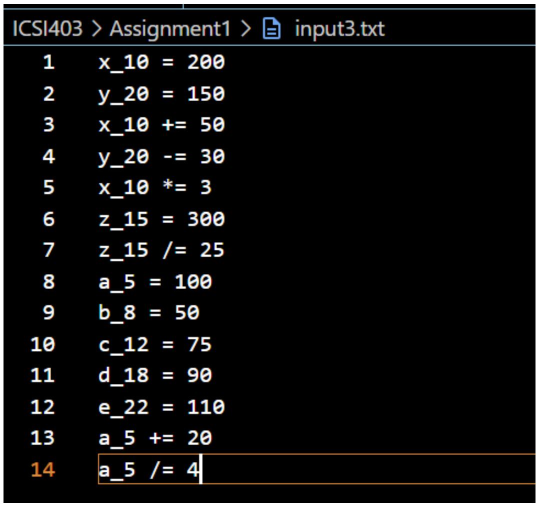
Result



ICSI403 > Assignment1 > project1Result_done.txt

```
id_1 = 350
k_27 = 4
var_2 = 30

```

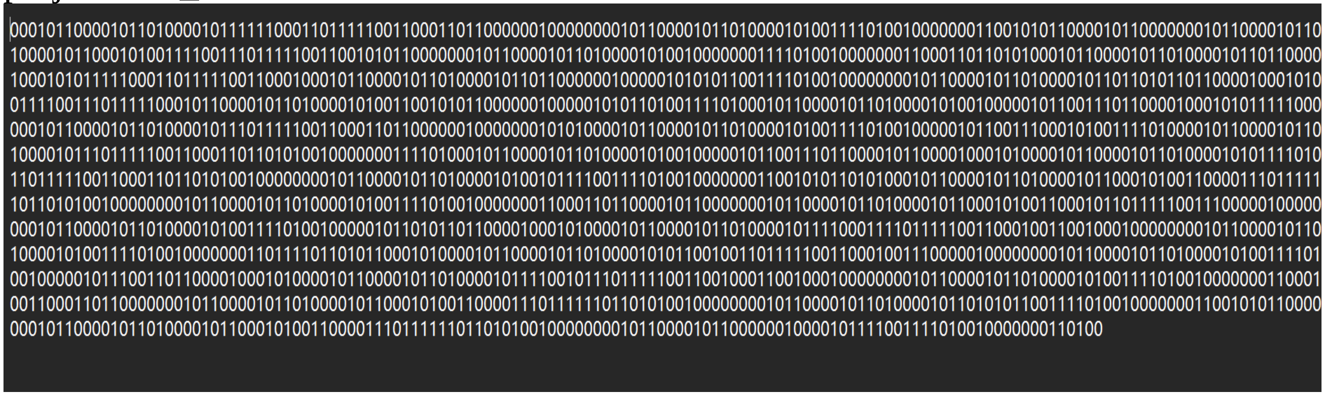
Input3.txt



ICSI403 > Assignment1 > input3.txt

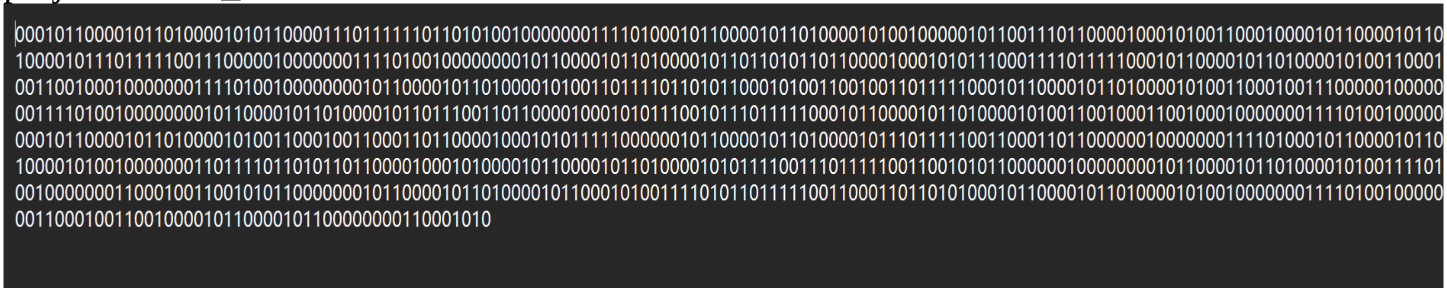
```
x_10 = 200
y_20 = 150
x_10 += 50
y_20 -= 30
x_10 *= 3
z_15 = 300
z_15 /= 25
a_5 = 100
b_8 = 50
c_12 = 75
d_18 = 90
e_22 = 110
a_5 += 20
a_5 /= 4
```

project1Bin_binf.txt



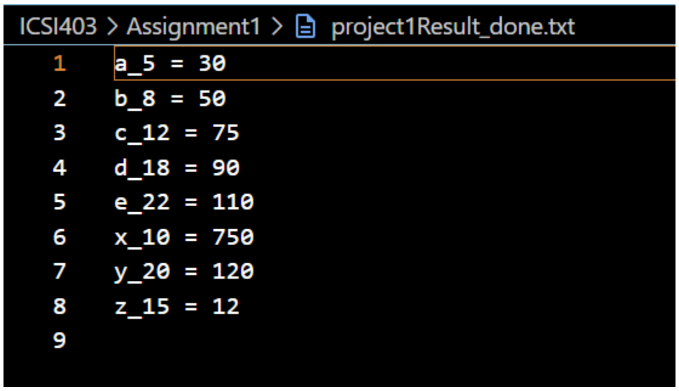
```
0001011000010110100001011111100011011111100110001101100000010000000010110000101101000010100111101001000000011001010110000101100000000101100001011 0
1000010110001010011110011101111100110010101100000001011000010110100001010010000000011110100100000001100011011010100010110000101101000010110110000 0
1000101011111000110111111001100010001011000010110100001011011000000100000101010110011110100100000000010110000101101000010110110101101100001000101 0
0111100111011111100010110000101101000010100110010101100000010000010101101001111010001011000010110100001010010000010110011101100001000101011111100 0
0001011000010110100001011101111100110001101100000010000000010101000010110000101101000010100111101001000000101100111000101001111010000101100001011 0
1000010111011111100110001101101010010000000011110100010110000101101000010100100000010110011101100001011000010001010000010110000101101000010101111010
1101111110011000110110101001000000000101100001011010000101001011110011110100100000000110010101101010001011000010110100001011000101001100001110111 11
1011010100100000000010110000101101000010100111101001000000011000110110000101100000000101100001011010000101100010100110001011011111100111000001000000
0001011000010110100001010011110100100000101101011011000010001010000101100001011010000101111000111101111110011000100110010001000000001011000010110
1000010100111101001000000011011110110101011000101000010110000101101000010101100100110111111001100010011100000100000000010110000101101000010100111101
0010000001011100110110000100010100001011000010110100001011110010111011111100110010001100100010000000010110000101101000010100111101001000000011000 1
0011000110110000000010110000101101000010110001010011000011101111110110101001000000001011000010110100001011010101100111101001000000011001010110000 0
00010110000101101000010110001010011000011101111111010101001000000000101100001011000000100001011110011110100100000000110100
```

project1Result_outf.txt



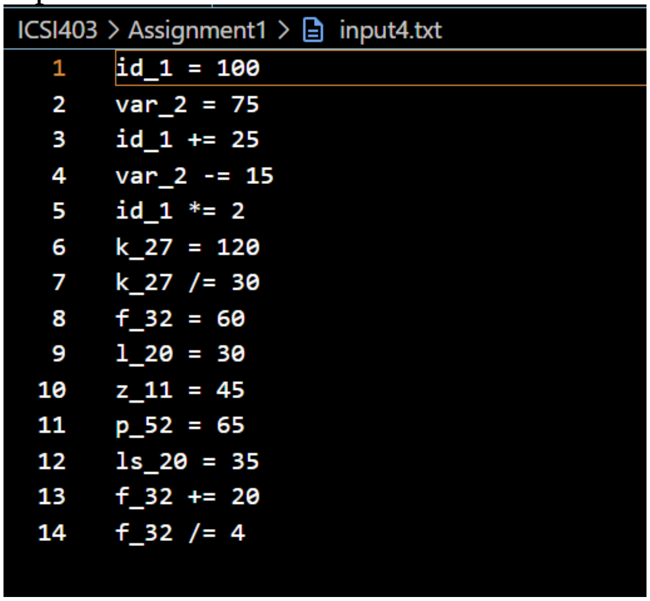
```
0001011000010110100001010110000111011111110110101001000000011110100010110000101101000010100100000101100111011000010001010011000100001011000010110
1000010111011111100111000001000000011110100100000000010110000101101000010110110101101100001000101011100011110111111000101100001011010000101001100 01
0011001000100000001111010010000000001011000010110100001010011011110110101100010100110010011011111100010110000101101000010100110001001110000010000 00
0011111010010000000001011000010110100001011011100110110000100010101110010111011111100010110000101101000010100110010001100100010000000111101001000000
0001011000010110100001010011000100110001101100001000101011111000000010110000101101000010111011111001100011011000000100000000111101000101100001011 0
100001010010000000110111101101011011000010001010000101100001011010000101011110011101111110011001010110000000100000000101100001011010000101001111101
0010000000110001001100101011000000001011000010110100001011000101001111010110111111001100011011010100010110000101101000010100100000000111101001000000
001100010011001000010110000101100000000110001010
```

Result



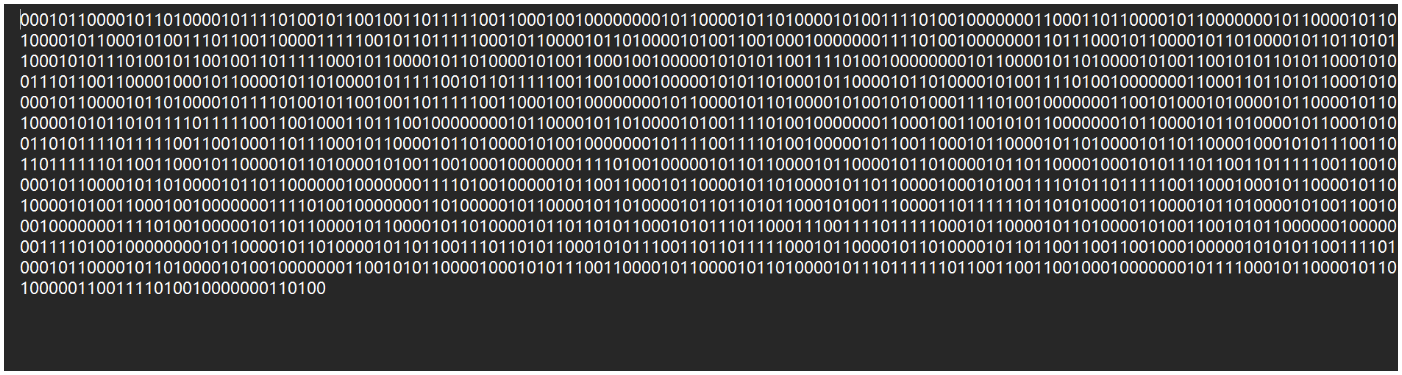
```
a_5 = 30
b_8 = 50
c_12 = 75
d_18 = 90
e_22 = 110
x_10 = 750
y_20 = 120
z_15 = 12

```

Input4.txt



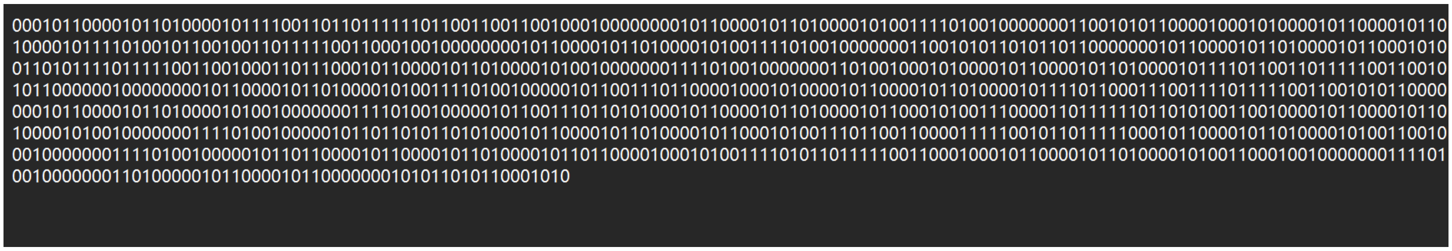
```
id_1 = 100
var_2 = 75
id_1 += 25
var_2 -= 15
id_1 *= 2
k_27 = 120
k_27 /= 30
f_32 = 60
l_20 = 30
z_11 = 45
p_52 = 65
ls_20 = 35
f_32 += 20
f_32 /= 4
```

project1Bin_binf.txt



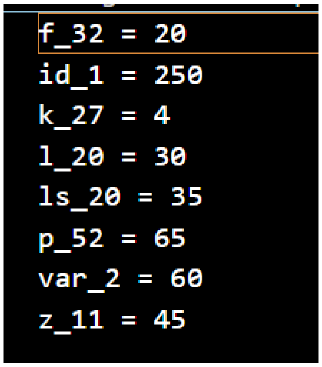
```
00010110000101101000010111101001011001001101111110011000100100000000101100001011010000101001111010010000000110001101100001011000000001011000010110
10000101100010100111011001100001111100101101111110001011000010110100001010011001000100000000111101001000000011011100010110000101101000010110110101
1000101011101001011001001101111100010110000101101000010100110001001000001010101100111101001000000000101100001011010000101001100101011010110001010
0111011001100001000101100001011010000101111100101101111110011001000100000101011010001011000010110100001010011110100100000001100011011010110001010
0001011000010110100001011110100101100100110111111001100010010000000010110000101101000010100101010001111010010000000110010100010100001011000010110
100001010110101111011111100110010001101110010000000010110000101101000010100111101001000000001100010011001010110000000010110000101101000010110001010
0110101111011111100110010001101110001011000010110100001010010000000101111001111010010000010110011000101100001011010000101101100001000101011100110
1101111110110011000101100001011010000101001100100010000000111101001000001011011000010110000101101000010110110000100010101110110011011111100110010
0001011000010110100001011011000000010000000111101001000001011001100010110000101101000010110110000100010100111101011011111001100010001011000010110
100001010011000100100000000111101001000000001101000001011000010110100001011011010110001010011100001101111111011010100010110000101101000010100110010
0010000000011110100100000101101100001011000010110100001011010101100010101110110001110011110111110001011000010110100001010011001010110000000100000
0011110100100000000101100001011010000101101100111010101100010101110011011011111100010110000101101000010110110011001100100010000001010101100111101
0001011000010110100001010010000000110010101100001000101011100110000101100001011010000101110111111101100110011001000100000000101111000101100001010110
1000001100111101001000000110100
```

project1Result_outf.txt



```
000101100001011010000101111001101101111111011001100110010001000000000101100001011010000101001111010010000000110010101100001000101000010110000101 10
1000010111101001011001001101111110011000100100000000101100001011010000101001111010010000000110010101101011011000000010110000101101000010110001010
011010111101111100110010001101110001011000010110100001010010000000111101001000000011010010001010000101100001011010000101111011001101111100110010
1011000000100000000010110000101101000010100111101001000000101100111011000010001010000101100001011010000101111011000111001111011111100110010101 10000
0001011000010110100001010010000000011110100100000101100111011010100010110000101101000010110001010011100001101111110110101001100100001011000010110
10000101001000000011110100100000101101101011010100010110000101101000010110001010011101100110000111110010110111111000101100001011010000101001 10010
00100000001111010010000001011011000010110000101101000010110110000100010100111101011011111100110001000101100001011010000101001100010010000000111101
0010000000110100000010110000101100000000101011010110001010
```

Result

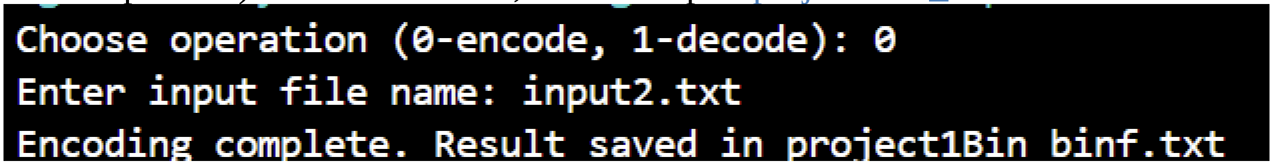


```
f_32 = 20
id_1 = 250
k_27 = 4
l_20 = 30
ls_20 = 35
p_52 = 65
var_2 = 60
z_11 = 45
```

III. User documentation

i. How to run your program

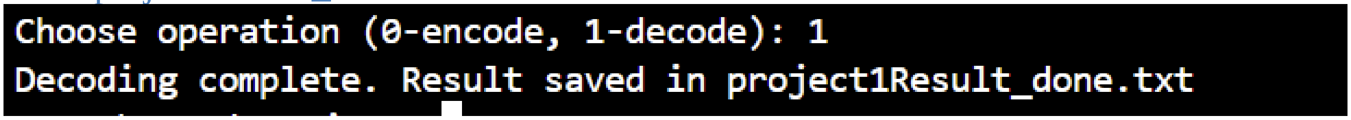
1. Run consumer which achieve input file (input1.txt , input2.txt, input3.txt, input4.txt) and encode them, it will output project1Bin_binf.txt



```
Choose operation (0-encode, 1-decode): 0
Enter input file name: input2.txt
Encoding complete. Result saved in project1Bin_binf.txt
```

2. Run producer (don't need input) which will receive project1Bin_binf.txt as input and decode them and do calculation, then it will encode the result and put it into project1Result_outf.txt

3. Run consumer again, it decode the project1Result_outf.txt and put the result into project1Result_done.txt



```
Choose operation (0-encode, 1-decode): 1
Decoding complete. Result saved in project1Result_done.txt
```

ii. Describe parameter (if any)

When running consumer, one can input 0 to choose encode file or 1 to decode file.

IV. Source Code

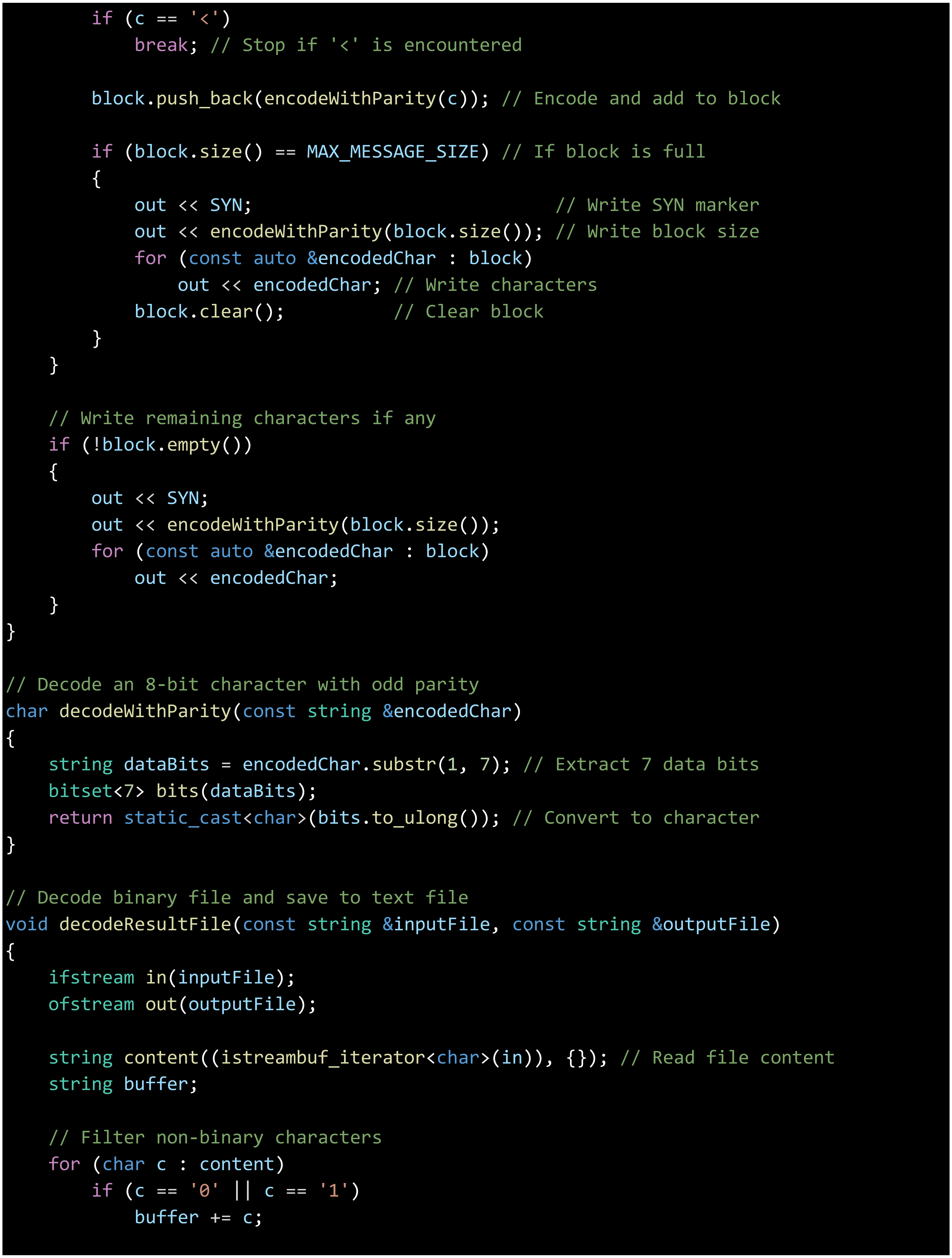
Correctness:

I execute my program to test the four example files, and the results are all correct Layering. Readability. Comments are showing follows

Layering Readability Comments Efficiency are showing as follows

Consumer

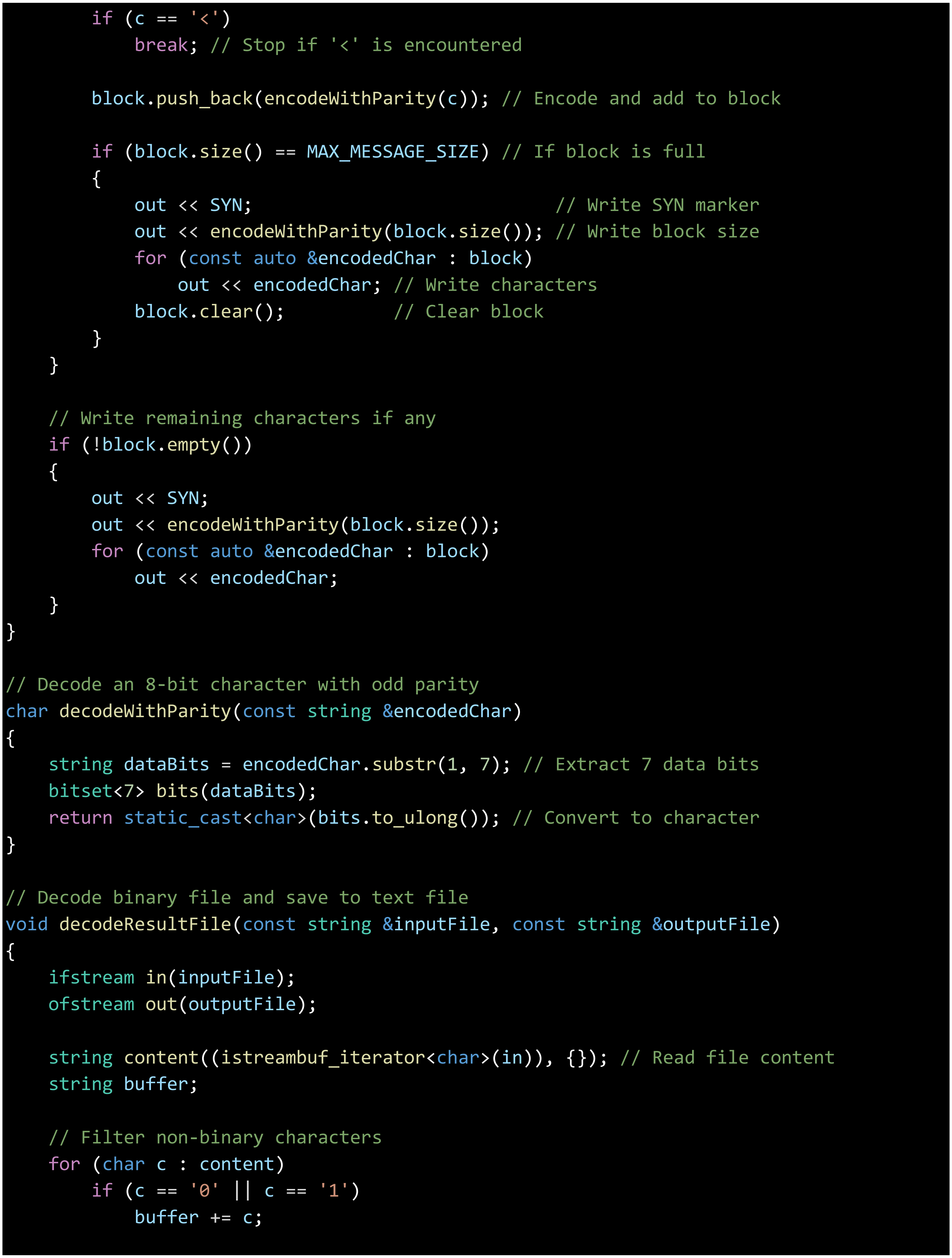
```
#include <iostream>
#include <fstream>
#include <bitset>
#include <string>
#include <vector>

using namespace std;

const int MAX_MESSAGE_SIZE = 5;        // Max characters per block
const string SYN = "0001011000010110"; // Odd parity encoding of ASCII 22 (SYN)

// Encode a character with odd parity
string encodeWithParity(char c)
{
    bitset<7> bits(c);
    int parity = bits.count() % 2 == 0 ? 1 : 0;      // Calculate parity bit
    return bitset<8>((parity << 7) | c).to_string(); // Combine parity and data
}

// Encode input file and save to binary file
void encodeInputFile(const string &inputFile, const string &outputFile)
{
    ifstream in(inputFile);
    ofstream out(outputFile);
    vector<string> block; // Store encoded characters
    char c;

    while (in.get(c))
    {
```



```
        if (c == '<')
            break; // Stop if '<' is encountered

        block.push_back(encodeWithParity(c)); // Encode and add to block

        if (block.size() == MAX_MESSAGE_SIZE) // If block is full
        {
            out << SYN;                            // Write SYN marker
            out << encodeWithParity(block.size()); // Write block size
            for (const auto &encodedChar : block)
                out << encodedChar; // Write characters
            block.clear();          // Clear block
        }
    }

    // Write remaining characters if any
    if (!block.empty())
    {
        out << SYN;
        out << encodeWithParity(block.size());
        for (const auto &encodedChar : block)
            out << encodedChar;
    }
}

// Decode an 8-bit character with odd parity
char decodeWithParity(const string &encodedChar)
{
    string dataBits = encodedChar.substr(1, 7); // Extract 7 data bits
    bitset<7> bits(dataBits);
    return static_cast<char>(bits.to_ulong()); // Convert to character
}

// Decode binary file and save to text file
void decodeResultFile(const string &inputFile, const string &outputFile)
{
    ifstream in(inputFile);
    ofstream out(outputFile);

    string content((istreambuf_iterator<char>(in)), {}); // Read file content
    string buffer;

    // Filter non-binary characters
    for (char c : content)
        if (c == '0' || c == '1')
            buffer += c;
    size_t pos = 0;
    while ((pos = buffer.find(SYN)) != string::npos) // Find SYN marker
    {
        buffer = buffer.substr(pos + 16); // Skip SYN

        if (buffer.size() < 8)
            break; // Check if enough bits remain

        string lengthEncoded = buffer.substr(0, 8);      // Extract block size
        int blockSize = decodeWithParity(lengthEncoded); // Decode block size
        buffer = buffer.substr(8);                       // Remove block size from buffer

        for (int i = 0; i < blockSize; i++) // Decode each character in block
        {
            if (buffer.size() < 8)
                break;

            string encodedChar = buffer.substr(0, 8); // Extract character
            buffer = buffer.substr(8);                // Remove character from buffer

            char decodedChar = decodeWithParity(encodedChar); // Decode character
            out.put(decodedChar);                             // Write to output file
        }
    }
}

int main()
{
    int choice;
    string inputFile, outputFile;

    cout << "Choose operation (0-encode, 1-decode): ";
    cin >> choice;

    if (choice == 0) // Encode
    {
        cout << "Enter input file name: ";
        cin >> inputFile;
        outputFile = "project1Bin_binf.txt"; // Output binary file
        encodeInputFile(inputFile, outputFile);
        cout << "Encoding complete. Result saved in " << outputFile << endl;
    }
    else if (choice == 1) // Decode
    {
        inputFile = "project1Result_outf.txt";  // Input binary file
        outputFile = "project1Result_done.txt"; // Output text file
        decodeResultFile(inputFile, outputFile);
        cout << "Decoding complete. Result saved in " << outputFile << endl;
    }
    else
    {
        cout << "Invalid choice!" << endl;
    }

    return 0;
}
```

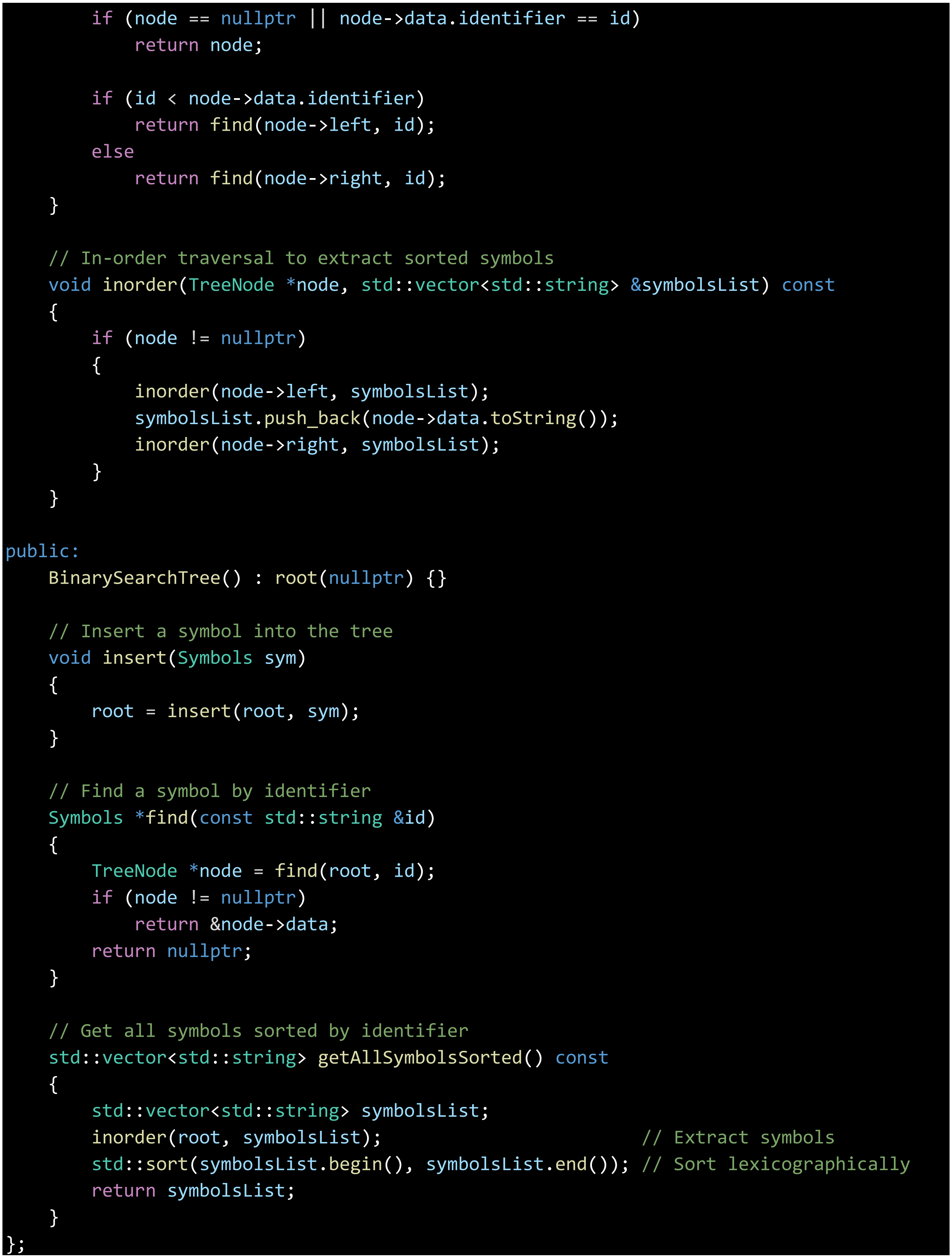
Producer

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <vector>
#include <algorithm>
#include <bitset>
using namespace std;

const int MAX_MESSAGE_SIZE = 5;        // Max characters per block
const string SYN = "0001011000010110"; // Odd parity encoding of ASCII 22 (SYN)

// Encode a character with odd parity
string encodeWithParity(char c)
{
    bitset<7> bits(c);
    int parity = bits.count() % 2 == 0 ? 1 : 0;      // Calculate parity bit
    return bitset<8>((parity << 7) | c).to_string(); // Combine parity and data
}

// Encode input file and save to binary file
void encodeInputFile(const string &inputFile, const string &outputFile)
{
    ifstream in(inputFile);
    ofstream out(outputFile);
    vector<string> block; // Store encoded characters
    char c;

    while (in.get(c))
    {
```

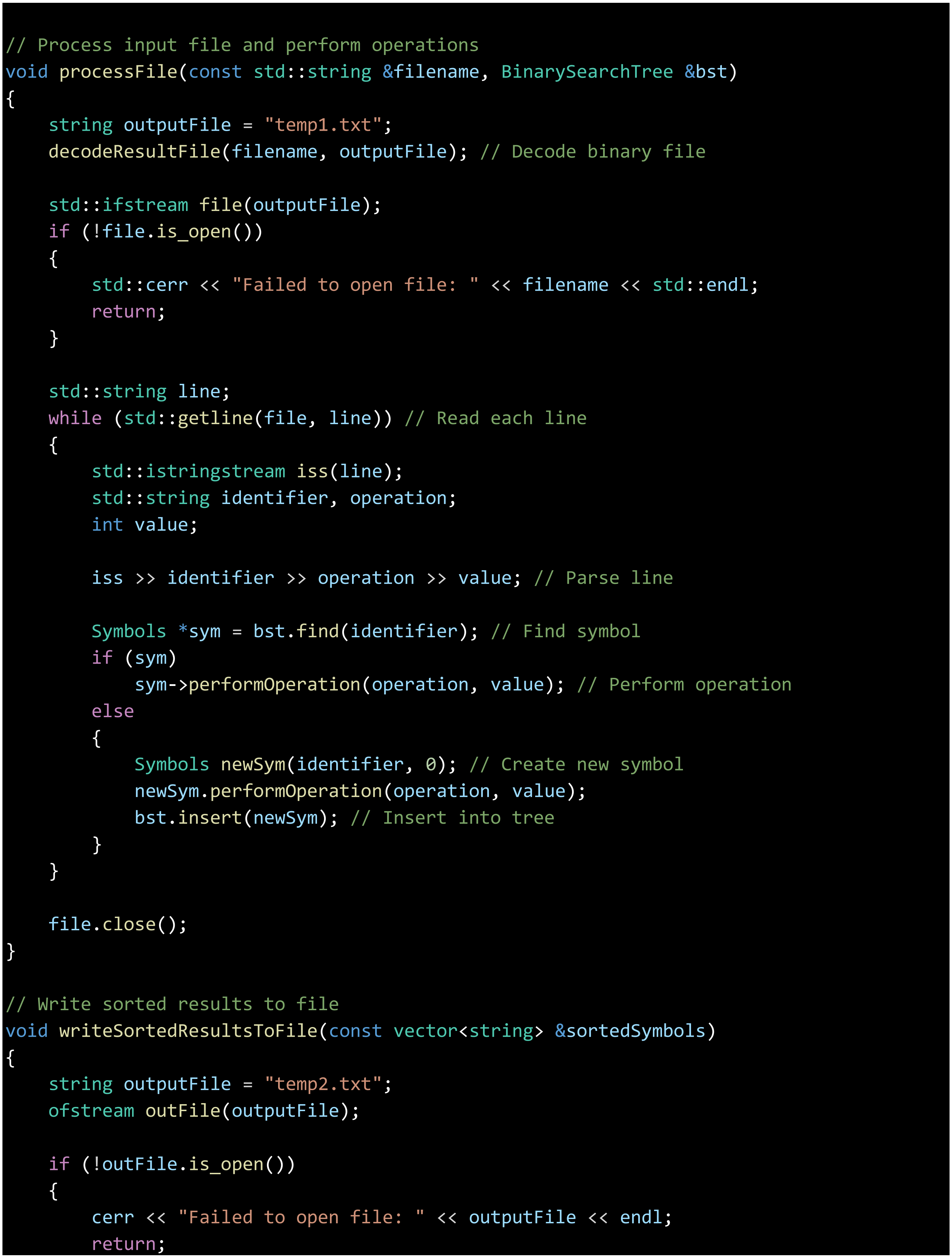


```
        if (c == '<')
            break; // Stop if '<' is encountered

        block.push_back(encodeWithParity(c)); // Encode and add to block

        if (block.size() == MAX_MESSAGE_SIZE) // If block is full
        {
            out << SYN;                            // Write SYN marker
            out << encodeWithParity(block.size()); // Write block size
            for (const auto &encodedChar : block)
                out << encodedChar; // Write characters
            block.clear();          // Clear block
        }
    }

    // Write remaining characters if any
    if (!block.empty())
    {
        out << SYN;
        out << encodeWithParity(block.size());
        for (const auto &encodedChar : block)
            out << encodedChar;
    }
}

// Decode an 8-bit character with odd parity
char decodeWithParity(const string &encodedChar)
{
    string dataBits = encodedChar.substr(1, 7); // Extract 7 data bits
    bitset<7> bits(dataBits);
    return static_cast<char>(bits.to_ulong()); // Convert to character
}

// Decode binary file and save to text file
void decodeResultFile(const string &inputFile, const string &outputFile)
{
    ifstream in(inputFile);
    ofstream out(outputFile);

    string content((istreambuf_iterator<char>(in)), {}); // Read file content
    string buffer;

    // Filter non-binary characters
    for (char c : content)
        if (c == '0' || c == '1')
            buffer += c;
    size_t pos = 0;
    while ((pos = buffer.find(SYN)) != string::npos) // Find SYN marker
    {
        buffer = buffer.substr(pos + 16); // Skip SYN

        if (buffer.size() < 8)
            break; // Check if enough bits remain

        string lengthEncoded = buffer.substr(0, 8);      // Extract block size
        int blockSize = decodeWithParity(lengthEncoded); // Decode block size
        buffer = buffer.substr(8);                       // Remove block size from
buffer

        for (int i = 0; i < blockSize; i++) // Decode each character in block
        {
            if (buffer.size() < 8)
                break;

            string encodedChar = buffer.substr(0, 8); // Extract character
            buffer = buffer.substr(8);                // Remove character from buffer

            char decodedChar = decodeWithParity(encodedChar); // Decode character
            out.put(decodedChar);                             // Write to output file
        }
    }
}

// Class to represent symbols (identifiers and their values)
class Symbols
{
public:
    std::string identifier;
    int value;

    // Constructor
    Symbols(std::string id, int val) : identifier(id), value(val) {}

    // Perform arithmetic operations
    void performOperation(std::string operation, int val)
    {
        if (operation == "+=")
            value += val;
        else if (operation == "-=")
            value -= val;
        else if (operation == "*=")
            value *= val;
        else if (operation == "/=")
```

```
            value /= val;
        else if (operation == "=")
            value = val; // Direct assignment
        else
            std::cerr << "Unknown operation: " << operation << std::endl;
    }

    // Convert symbol to string
    std::string toString() const
    {
        return identifier + " = " + std::to_string(value);
    }
};

// Binary Search Tree node
struct TreeNode
{
    Symbols data;
    TreeNode *left;
    TreeNode *right;

    TreeNode(Symbols sym) : data(sym), left(nullptr), right(nullptr) {}
};

// Binary Search Tree class
class BinarySearchTree
{
private:
    TreeNode *root;

    // Insert a node into the tree
    TreeNode *insert(TreeNode *node, Symbols sym)
    {
        if (node == nullptr)
            return new TreeNode(sym);

        if (sym.identifier < node->data.identifier)
            node->left = insert(node->left, sym);
        else if (sym.identifier > node->data.identifier)
            node->right = insert(node->right, sym);

        return node;
    }

    // Find a node by identifier
    TreeNode *find(TreeNode *node, const std::string &id)
    {
```



```
        if (node == nullptr || node->data.identifier == id)
            return node;

        if (id < node->data.identifier)
            return find(node->left, id);
        else
            return find(node->right, id);
    }

    // In-order traversal to extract sorted symbols
    void inorder(TreeNode *node, std::vector<std::string> &symbolsList) const
    {
        if (node != nullptr)
        {
            inorder(node->left, symbolsList);
            symbolsList.push_back(node->data.toString());
            inorder(node->right, symbolsList);
        }
    }

public:
    BinarySearchTree() : root(nullptr) {}

    // Insert a symbol into the tree
    void insert(Symbols sym)
    {
        root = insert(root, sym);
    }

    // Find a symbol by identifier
    Symbols *find(const std::string &id)
    {
        TreeNode *node = find(root, id);
        if (node != nullptr)
            return &node->data;
        return nullptr;
    }

    // Get all symbols sorted by identifier
    std::vector<std::string> getAllSymbolsSorted() const
    {
        std::vector<std::string> symbolsList;
        inorder(root, symbolsList);                        // Extract symbols
        std::sort(symbolsList.begin(), symbolsList.end()); // Sort lexicographically
        return symbolsList;
    }
};
```



```cpp
// Process input file and perform operations
void processFile(const std::string &filename, BinarySearchTree &bst)
{
    string outputFile = "temp1.txt";
    decodeResultFile(filename, outputFile); // Decode binary file

    std::ifstream file(outputFile);
    if (!file.is_open())
    {
        std::cerr << "Failed to open file: " << filename << std::endl;
        return;
    }

    std::string line;
    while (std::getline(file, line)) // Read each line
    {
        std::istringstream iss(line);
        std::string identifier, operation;
        int value;

        iss >> identifier >> operation >> value; // Parse line

        Symbols *sym = bst.find(identifier); // Find symbol
        if (sym)
            sym->performOperation(operation, value); // Perform operation
        else
        {
            Symbols newSym(identifier, 0); // Create new symbol
            newSym.performOperation(operation, value);
            bst.insert(newSym); // Insert into tree
        }
    }

    file.close();
}

// Write sorted results to file
void writeSortedResultsToFile(const vector<string> &sortedSymbols)
{
    string outputFile = "temp2.txt";
    ofstream outFile(outputFile);

    if (!outFile.is_open())
    {
        cerr << "Failed to open file: " << outputFile << endl;
        return;
    }

    for (const auto &sym : sortedSymbols)
        outFile << sym << endl; // Write sorted symbols
    outFile.close();

    encodeInputFile(outputFile, "project1Result_outf.txt"); // Encode results
}

int main()
{
    BinarySearchTree bst;

    processFile("project1Bin_binf.txt", bst); // Process input file

    std::vector<std::string> sortedSymbols = bst.getAllSymbolsSorted(); // Get sorted
symbols
    writeSortedResultsToFile(sortedSymbols);                            // Write
results to file

    return 0;
}
```