Swinburne University Of Technology

Faculty of Information and Communication Technologies

ASSIGNMENT COVER SHEET

Subject Code:	COS30008
Subject Title:	Data Structures & Patterns
Assignment number and title:	3 – Design Patterns and 12 Bit I/O
Due date:	May 13, 2024, 10:30
Lecturer:	Dr. Markus Lumpe

Your name: Xuan Tuan Minh Nguyen **Your student id:** 103819212

Marker's comments:

Problem	Marks	Obtained
1	138	
Total	138	

Extension certification:

This assignment has been given an extension and is now due on ________________

Signature of Convener:______________________

```cpp
/**
 * @headerfile ifstream12.h
 * @file ifstream12.cpp
 * @author Xuan Tuan Minh Nguyen - 103819212
 * @date May 10th, 2024
 * @brief This is the implementation of the ifstream12 class based on the 
   ifstream12.h for Problem Set 3
 */

// Include header file with cassert to handle assertions
#include "ifstream12.h"
#include <cassert>

// Reset the byte index, byte count, and bit index.
// However, the content of the buffer is not reset.
void ifstream12::reset()
{
    fByteCount = 0;
    fByteIndex = 0;
    fBitIndex = 7;
}

// Fetch the data from the stream to the buffer
void ifstream12::fetch_data()
{
    // Reset the buffer indexes
    reset();
    // If the stream is good
    if (fIStream.good())
    {
        // Read the data to the buffer
        fIStream.read(reinterpret_cast<char *>(fBuffer), fBufferSize);
        fByteCount = fIStream.gcount();
    }
}

// Function that handle the mapping process of read a bit from the buffer
std::optional<size_t> ifstream12::readBit()
{
    // If the byte count is 0, fetch the data
    if (fByteCount == 0)
        fetch_data();
    // If the end of file is reached, return null
    if (eof())
        return std::nullopt;
    // Conversion of bit patterns to single value
    std::byte byte = fBuffer[fByteIndex] & (std::byte{1} << fBitIndex);
    // Bit-mask for the bit at fBitIndex. If value > 0 then return 1, else 
       return 0
    size_t bitRet = std::to_integer<size_t>(byte) > 0 ? 1 : 0;
    // Jump to the next bit
    fBitIndex--;
    // If no more bit to jump, jump to next byte
    if (fBitIndex < 0)
    {
        fByteCount--;
        fByteIndex++;
        fBitIndex = 7;
    }
    // Return the bit value
    return bitRet;
}

// Constructor of ifstream12, taking in the file name and buffer size
ifstream12::ifstream12(const char *aFileName, size_t aBufferSize) :
  fBuffer(new std::byte[aBufferSize]), fBufferSize(aBufferSize),
  fByteCount(0), fByteIndex(0), fBitIndex(7)
{
    // Reset the buffer
    reset();
    // Open the file
    open(aFileName);
}

// Destructor of ifstream12, close the file and delete the buffer
ifstream12::~ifstream12()
{
    close();
    delete[] fBuffer;
}

// Function that handles file opening
void ifstream12::open(const char *aFileName)
{
    // Make sure that the file is not open
    assert(!isOpen());
    // If the file name is not nullptr
    if (aFileName)
        // Open the file in binary mode
        fIStream.open(aFileName, std::ios::binary);
}

void ifstream12::close()
{
    // Make sure that the file is open
    assert(isOpen());
    // Close the file
    fIStream.close();
```

```cpp
}

// Function that calls to ifstream.is_open()
bool ifstream12::isOpen() const
{
    return fIStream.is_open();
}

// Function that calls to ifstream.good()
bool ifstream12::good() const
{
    return fIStream.good();
}

// Function that return true if the available input bytes are 0
bool ifstream12::eof() const
{
    return fByteCount == 0;
}

// Overload the operator >> to read 12 bits from the stream
ifstream12 &ifstream12::operator>>(size_t &a12BitValue)
{
    // Make sure that the file is open
    assert(isOpen());
    // Reset the 12 bit value
    a12BitValue = 0;
    // Read 12 bits from the stream
    for (size_t i = 0; i < 12; i++)
    {
        // Read the bit
        std::optional<size_t> bit = readBit();
        // If the bit is null, break the loop
        if (!bit.has_value())
            break;
        // If the bit is 1, set the bit at i to 1
        if (bit.value() == 1)
            a12BitValue += (1 << i);
    }
    // Return the updated stream
    return *this;
}

```