Ifstream12.cpp

```
#include "ifstream12.h"
#include <iostream>
#include <bit>
// Constructor
ifstream12::ifstream12(const char* aFileName, size_t aBufferSize)
    : fBuffer(new std::byte[aBufferSize]), fBufferSize(aBufferSize),
    fByteCount(0), fByteIndex(0), fBitIndex(7) {
    if (aFileName) {
        open(aFileName);
    }
    reset();
}

// Destructor
ifstream12::~ifstream12() {
    close();
    delete[] fBuffer;
}

// Reset buffer
void ifstream12::reset() {
    fByteIndex = 0;
    fBitIndex = 7;
    fByteCount = 0;
}

// Fetch data from file
void ifstream12::fetch_data() {
    if (!fIStream.eof() && fIStream.good()) {
        fIStream.read(reinterpret_cast<char*>(fBuffer), fBufferSize);
        fByteCount = fIStream.gcount();
        fByteIndex = 0;
        fBitIndex = 7;
    }
}

// Read next bit
std::optional<size_t> ifstream12::readBit() {
    if (fByteCount == 0 && !eof()) {
        fetch_data();
        if (fByteCount == 0) {
            return std::nullopt; // No more data to read
        }
```

```
    }

    if (fByteCount == 0) {
        return std::nullopt;
    }

    std::byte lByte = fBuffer[fByteIndex] & (std::byte{ 1 } << fBitIndex);
    size_t bitValue = std::to_integer<size_t>(lByte) ? 1 : 0;

    if (--fBitIndex < 0) {
        fBitIndex = 7;
        fByteIndex++;
        if (fByteIndex >= fByteCount) {
            fByteIndex = 0;
            fByteCount = 0;
            fetch_data();
            if (fByteCount == 0) {
                return std::nullopt; // No more data after refetch
            }
        }
    }

    return bitValue;
}

// Open file
void ifstream12::open(const char* aFileName) {
    close(); // Close any open file first
    fIStream.open(aFileName, std::ios::binary);
    reset();
}

// Close file
void ifstream12::close() {
    if (fIStream.is_open()) {
        fIStream.close();
    }
    reset();
}

// Check if file is open
bool ifstream12::isOpen() const {
    return fIStream.is_open();
}

// Check if stream is good
bool ifstream12::good() const {
    return fIStream.good();
```

```
}

// Check for EOF
bool ifstream12::eof() const {
    return fIStream.eof() && fByteCount == 0;
}

// Read 12-bit value
ifstream12& ifstream12::operator>>(size_t& aValue) {
    aValue = 0;
    for (int i = 0; i < 12; i++) {
        auto bit = readBit();
        if (!bit) {
            break; // EOF reached
        }
        if (bit.value() == 1) {
            aValue |= (1 << i);
        }
    }
    return *this;
}
```