```cpp
#include "ifstream12.h"

#include <cassert>
#include <iostream>
#include <bit>

// Constructor
ifstream12::ifstream12(const char* aFileName, size_t aBufferSize)
    : fBuffer(new std::byte[aBufferSize]), fBufferSize(aBufferSize),
     fByteIndex(0), fBitIndex(7), fByteCount(0) {
        if (aFileName) {
            open(aFileName);
        }
}

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

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

// Fetch data
void ifstream12::fetch_data() {
    if (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 (fByteIndex >= fByteCount) {
           if (fIStream.eof()) {
               return std::nullopt;
           }
           fetch_data();
           if (fByteCount == 0) {
               return std::nullopt;
           }
       }
```

```
        if (fBitIndex < 0) {
            fByteIndex++;
            if (fByteCount <= fByteIndex) {
                if (fIStream.eof() || fByteCount == 0) {
                    return std::nullopt;
                }
                fetch_data();

            }
            fBitIndex = 7;
        }

        std::byte lByte = fBuffer[fByteIndex];
        size_t bit = static_cast<size_t>((lByte >> fBitIndex) &
         std::byte(1));
        fBitIndex--;
        return bit;
}

// Open file
void ifstream12::open(const char* aFileName) {
    assert(!isOpen());
        fIStream.open(aFileName, std::ifstream::binary);
        reset();
}

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

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

bool ifstream12::good() const {
    return fIStream.good() && (fByteIndex < fByteCount ||
     !fIStream.eof());
}

// EOF
bool ifstream12::eof() const {
    return (fByteIndex >= fByteCount && fIStream.eof()) ||
     (fByteIndex == fByteCount - 1 && fBitIndex < 0);
}

// Operator
```

```
ifstream12& ifstream12::operator>>(size_t& aValue) {
    aValue = 0;
        size_t bitsRead = 0;

        while (bitsRead < 12) {
            auto bit = readBit();
            if (!bit.has_value()) {
                break;
            }
            aValue |= (bit.value() << bitsRead++);
        }

        return *this;
}
```