Swinburne University Of Technology

Faculty of Information and Communication Technologies

ASSIGNMENT COVER SHEET

Subject Code: HIT3303/8303
Subject Title: Data Structures & Patterns
Assignment number and title: 3 – Design Patterns and 12 Bit I/O
Due date: **April 6, 2011, 10:30 a.m., on paper**
Lecturer: Dr. Markus Lumpe

Your name:_____

Marker's comments:

Problem	Marks	Obtained
1	49	
Total	49	

Extension certification:

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

Signature of Convener:_____

OStream12Bits.h

```
#ifndef OSTREAM12BITS_H_
#define OSTREAM12BITS_H_

typedef unsigned char byte; //8 bit positive number i.e. a byte
#include <fstream>

//This class is an adapter for a ofstream object (that works with 12 bits rather than 8 at a time)
class OStream12Bits
{
	std::ofstream fOStream;
	byte fBuffer[32]; //read 32 bits before actually writing to the file
	int fByteIndex;
	int fBitIndex; //how many bits to the left to shift the output

	void init(); //initialise data members
	void finishWriteBit(); //complete write to file
	void writeBit0(); //write 0
	void writeBit1(); //write 1

public:
	OStream12Bits();
	OStream12Bits(const char* aFileName);

	void open(const char* aFileName);
	void close();
	bool fail();
	void flush(); //flush the buffer (i.e. write it to the file) and reset byteIndex/bitIndex
	OStream12Bits& operator<<(int aCode);
};

#endif /* OSTREAM12BITS_H_ */
```

OStream12Bits.cpp

```
#include "OStream12Bits.h"

using namespace std;

OStream12Bits::OStream12Bits()
{
	init();
}

OStream12Bits::OStream12Bits(const char* aFileName)
{
	init();
	open(aFileName);
}

void OStream12Bits::init()
{
	for(int i = 0; i < 32; i++){
		fBuffer[i] = 0;
	}
	fByteIndex = 0;
	fBitIndex = 8;
}

void OStream12Bits::writeBit0()
{
	//no need to explicitly write 0, because it already is
	fBitIndex--;
	finishWriteBit();
}

void OStream12Bits::writeBit1()
{
	fBuffer[fByteIndex] += 1 << (fBitIndex - 1); //write 1 bit to the next MSB
	fBitIndex--;
	finishWriteBit();
}

//Checks if the bit index needs to be reset
//resets if need be, and increments the byte index for the next byte in the buffer
//if the byte index is 32 (end of buffer - last byteIndex is buffer[31]), flush the buffer and reset
void OStream12Bits::finishWriteBit()
{
	if(fBitIndex == 0)
	{
		if(fByteIndex == 31){
			fByteIndex++;
			flush(); //write full buffer to stream
		}
		else{
			fByteIndex++;
			fBitIndex = 8;
		}
	}
}

void OStream12Bits::open(const char* aFileName)
{
	fOStream.open(aFileName, ofstream::binary);
}

bool OStream12Bits::fail()
{
```

```
	return fOStream.fail();
}

void OStream12Bits::close()
{
	fOStream.close();
}

//Writes the buffer to the stream and resets the variables
void OStream12Bits::flush()
{
	fOStream.write((char*)fBuffer, fByteIndex + (fBitIndex % 8 ? 1 : 0)); //addition is to write another byte if there is a partial
                                                                          byte (i.e. bitIndex is not 8) (only used for the end)
	init();
}

OStream12Bits& OStream12Bits::operator<<(int aCode)
{
	aCode = aCode & 0x0fff; //mask lower 12 bits - we process 12 bits at a time
	for(int i = 0; i < 12; i++){
		if(aCode & 0x01){
			writeBit1();
		}
		else{
			writeBit0();
		}
		aCode >>= 1; //shift right: aCode = aCode / 2
	}
	return *this;
}
```

IStream12Bits.h

```
#ifndef ISTREAM12BITS_H_
#define ISTREAM12BITS_H_

typedef unsigned char byte;
#include <fstream>

class IStream12Bits
{
private:
	std::ifstream fIStream;
	byte fBuffer[32];
	int fByteCount; //number of bytes up to in the buffer
	int fByteIndex;
	int fBitIndex;

	void reload();
	int fetchCode();

public:
	IStream12Bits();
	IStream12Bits(const char* aFileName);

	void open(const char* aFileName);
	void close();
	bool fail();
	bool eof();
	IStream12Bits& operator>>(int& aCode);
};

#endif /* ISTREAM12BITS_H_ */
```

IStream12Bits.cpp

```
#include "IStream12Bits.h"
#include <iostream>

using namespace std;

IStream12Bits::IStream12Bits()
{
	fByteCount = 0; //start at first entry in the buffer
}

IStream12Bits::IStream12Bits(const char* aFileName)
{
	fByteCount = 0;
	open(aFileName);
}

void IStream12Bits::open(const char* aFileName)
{
	fIStream.open(aFileName, ifstream::binary);
	reload();
}

//Fills input buffer with the next bytes to be read
//resets variables for reading
void IStream12Bits::reload()
{
	fByteCount = 0;
	if(fIStream.is_open() && !eof() && !fail()){
		for(int i = 0; i < 32; i++){
			if(!IStream12Bits::eof()){
				fBuffer[i] = fIStream.get();
				fByteCount++;
			}
		}
	}

	fByteIndex = 0;
	fBitIndex = 8;
}

//Transforms bits stored in little-endian format yielding an integer of 12 bits
int IStream12Bits::fetchCode()
{
	int result = 0;

	for(int i = 0; i < 12; i++){
		if(fBuffer[fByteIndex] & (1 << (fBitIndex - 1))){ //if the MSB is 1, write it to the result
			result += (1 << i);
		}
		fBitIndex--;

		//if at the end of the byte, move on to the next
		if(fBitIndex == 0){
			fByteIndex++;
			// Check for reload
			if (fByteIndex == fByteCount)
			{
				reload();
				if(fByteCount == 0){
					return -1;
				}
			}
			fBitIndex = 8;
```

```
                        }
                }

                return result;
}

void IStream12Bits::close()
{
                fIStream.close();
}

bool IStream12Bits::fail()
{
                return fIStream.fail();
}

bool IStream12Bits::eof()
{
                return fIStream.eof();
}

IStream12Bits& IStream12Bits::operator>>(int& aCode)
{
                aCode = fetchCode();
                return *this;
}
```

main.cpp

```
#include "OStream12Bits.h"
#include "IStream12Bits.h"
#include <iostream>
#include <stdlib.h>

using namespace std;

void write4096()
{
	cout << "Write 4096 codes" << endl;

	OStream12Bits lWriter;

	lWriter.open("sample.lzw");
	if(lWriter.fail()){
		cerr << "Error: Unable to open output file!" << endl;
		exit(1);
	}

	for(int i = 4095; i >= 0; i--){
		lWriter << i;
	}

	lWriter.close();
}

void read4096()
{
	cout << "Read 4096 codes" << endl;

	IStream12Bits lInput;

	lInput.open("sample.lzw");
	if(lInput.fail()){
		cerr << "Error: Unable to open input file!" << endl;
		exit(2);
	}

	for(int i = 4095; i >= 0; i--){
		int code;
		lInput >> code;

		if(code != i){
			cerr << "Error: Code mismatch: " << code << " != " << i << endl;
			exit(3);
		}
	}

	if(!lInput.eof()){
		cerr << "Error: Input stream not exhausted!" << endl;
	}

	lInput.close();
}

int main()
{
	write4096();
	read4096();
	cout << "SUCCESS" << endl;
	return 0;
}
```