Byte Program

```cpp
#include <iostream>
using namespace std;

#ifndef __BYTE
#define __BYTE

class Byte {
public:

	enum { UNDER, OVER, OK, BYTESIZE = 8 };
	Byte() { error = false; }
	Byte(char *);
	int read(istream & = cin);
	Byte add(Byte);

	// switch high-order bit only
	Byte biasTo2sComplement();

	// apply 2s complement conversion for magnitude evaluation
	Byte to2sComplement();

	friend ostream & operator << (ostream &, const Byte &);
	int toInteger();
	const char * getErrorMessage() const;
	bool hasError() const;

private:
	char byte[BYTESIZE];
	bool error;
	const char * errorMessage;
	int magnitude();
};

#endif
```

Byte Program

```
// ByteTester.cpp : Driver Program

#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cstring>
#include "Byte.h"

using namespace std;

int _tmain (int argc, _TCHAR* argv[])
{

Byte b1, b2, b3;
char * comp = "2\'s complement ";
char * bias = "biased notation";

ofstream out("sum.out", ios::out);
ifstream in("byte.in", ios::in);

while (b1.read(in) && b2.read(in))
{
      b3 = b1.add(b2.biasTo2sComplement());
      out << "    " << b1 << "\t" << comp << "\t" << std::right
          << std::setw(8) << b1.toInteger() << endl;

      out << "+   " << b3 << "\t" << bias << "\t" << std::right
          << std::setw(8) << b2.biasTo2sComplement().toInteger() << endl;

      out << "-----------\t\t\t" << std::right << std::setw(8) << "----"
          << endl;

      out << "    " << b3 << "\t" << comp << "\t" << std::right
          << std::setw(8) << b3.toInteger() << endl << endl;
}

out.close();
in.close();

}
```

Byte Program

Selected Byte Class Methods

```
#include "stdafx.h"
#include <iostream>
#include "Byte.h"

const char * comp = "2\'s complement";
const char * bias = "biased notation";
const char * over = "overflow";
const char * under = "underflow";


Byte::Byte(char * str)
{
	for (int i = 0; i < BYTESIZE; i++)
		byte[i] = str[i]; // *(str + i);
	error = false;
}



int Byte::read(istream & in)
{
	for (int i = 0; i < Byte::BYTESIZE; i++)
		in >> byte[i];

	if (in.fail())
		return false;
	else
		return true;
}
```

Byte Program

```
Byte Byte::biasTo2sComplement()
{
	Byte temp;
	temp.byte[0] = byte[0] == '0' ? '1' : '0';
	for (int i = 1; i < Byte::BYTESIZE; i++)
		temp.byte[i] = byte[i];
	return temp;
}



int Byte::toInteger()
{
	Byte temp;
	int sign = 1;

	if (byte[0] == '1')
	{
		temp = to2sComplement();
		sign = -1;
	}
	else
		temp = *this;

	return temp.magnitude() * sign;
}
```

Byte Program

```
Byte Byte::add(Byte b)
{

Byte result;



return result;
}
```