P5 – Digital Objects

Final

Reusing the “readBit” and “printBit” I had created in the lab P2, I was able to test all of my various gates and adders.

```
int main() {
    HalfAdder ha;

    cout << "A: ";
    ha.a = readBit();

    cout << "B: ";
    ha.b = readBit();

    ha.calculate();

    cout << "\n\nSum Bit: ";
    printBit(ha.s);

    cout << "\nCarry Bit: ";
    printBit(ha.c);

    return 0;
}
```

```
A: 0
B: 1

Sum Bit: 1
Carry Bit: 0
```

```
A: 1
B: 1

Sum Bit: 0
Carry Bit: 1
```

```
int main() {
    FullAdder fa;

    cout << "A: ";
    fa.a = readBit();

    cout << "B: ";
    fa.b = readBit();

    cout << "C: ";
    fa.cin = readBit();

    fa.calculate();

    cout << "\nSum Bit: ";
    printBit(fa.s);

    cout << "\nCarry Bit: ";
    printBit(fa.cout);

    return 0;
}
```

```
A: 1
B: 1
C: 1

Sum Bit: 1
Carry Bit: 1
```

```
A: 1
B: 0
C: 1

Sum Bit: 0
Carry Bit: 1
```

With this completed I then moved on to creating the multi-bit adder and subtracter. For the adder to become a subtractor we must flip all the bits (for b inputs) and set the input carry equal to 1.

```
int main() {
    vector<bool> a(N);
    vector<bool> b(N);
    vector<bool> c(N+1);
    vector<bool> s(N);

    bool isSub;         //sub or add => 1 if sub

    FullAdder fa;       //main addition full adder
    XOR sXOR;           //AND gate which flips bits of b if subtract

    cout << "Number A: ";
    a = readNum();

    cout << "Binary A: ";
    printBin(a);
    cout << "\n\n";


    cout << "Number B: ";
    b = readNum();

    cout << "Binary B: ";
    printBin(b);
    cout << "\n\n";


    cout << "Addition or Subtraction: ";
    isSub = readSymbol();

    cout << ((isSub) ? "Subtracting" : "Adding") << endl;



    c[0] = isSub;         //pre-load first carry in with nSub
    sXOR.a = isSub;       //load and with nSub to flip bits if nSub is false

    for(int i=0; i<N; i++) {
        sXOR.b = b[i];

        fa.a = a[i];
        fa.b = sXOR.calculate();        //flips bit if isSub is false
        fa.cin = c[i];
        fa.calculate();

        c[i+1] = fa.cout;
        s[i] = fa.s;

    }

    cout << "\nAddition Result: ";
    printBin(s);

    cout << "\nCarry Bit: ";
    printBit(c[N]);

    return 0;
}
```

```
Number A: 48
Binary A: 00110000

Number B: 27
Binary B: 00011011

Addition or Subtraction: +
Adding

Addition Result: 01001011
```

```
Number A: 12
Binary A: 00001100

Number B: 8
Binary B: 00001000

Addition or Subtraction: +
Adding

Addition Result: 00010100
```

```
Number A: 12
Binary A: 00001100

Number B: 8
Binary B: 00001000

Addition or Subtraction: -
Subtracting

Addition Result: 00000100
```

```
Number A: 121
Binary A: 01111001

Number B: 87
Binary B: 01010111

Addition or Subtraction: -
Subtracting

Addition Result: 00100010
```

Additional Work

```
// Initialize to human, 10 strength, 10 hit points
Creature::Creature() {
    type = 0;           //human type
    strength = 10;
    hitpoints = 10;
};

// Initialize creature to new type, strength, hit points
Creature::Creature(int newType, int newStrength, int newHit) {
    type = newType;
    strength = newStrength;
    hitpoints = newHit;
};


std::string Human::getSpecies() {
    return "Human";
}

std::string Elf::getSpecies() {
    return "Elf";
}

std::string Cyberdemon::getSpecies() {
    return "Cyberdemon";
}

std::string Balrog::getSpecies() {
    return "Balrog";
}
```

```
int Human::getDamage() {
    return getDamage();
}

int Elf::getDamage() {
    return getDamage();
}

int Cyberdemon::getDamage() {
    return getDamage();
}

int Balrog::getDamage() {
    return getDamage() * 2;
}
```

```
class Human : public Creature {

public:
    Human() : Creature() {};        //default type is human

    int getDamage();
    std::string getSpecies();
};


class Elf : public Creature {

public:
    Elf() : Creature(3, 10, 10) {};     //initialise elf

    int getDamage();
    std::string getSpecies();
};


class Demon : public Creature {

public:
    Demon(int newType) : Creature(newType, 10, 10) {};

    int getDamage();
};




class Cyberdemon : public Demon {

public:
    Cyberdemon() : Demon(1) {};

    int getDamage();
    std::string getSpecies();
};

class Balrog : public Demon {

public:
    Balrog() : Demon(2) {};

    int getDamage();
    std::string getSpecies();
};
```

```cpp
int Creature::getStrength() {
    return strength;
}

void Creature::setStrength(int newStrength) {
    strength = newStrength;
}


int Creature::getHitpoints() {
    return hitpoints;
}

void Creature::setHitpoints(int newHit) {
    hitpoints = newHit;
}
```

```cpp
class Creature {
private:
    int type; // 0 human, 1 cyberdemon, 2 balrog, 3 elf
    int strength; // How much damage we can inflict
    int hitpoints; // How much damage we can sustain
    std::string getSpecies(); // Returns type of species

public:
    // Initialize to human, 10 strength, 10 hit points
    Creature();

    // Initialize creature to new type, strength, hit points
    Creature(int newType, int newStrength, int newHit);

    // Also add appropriate accessor and mutator functions
    // for type, strength, and hit points
    int getDamage();

    int getStrength();
    void setStrength(int newStrength);

    int getHitpoints();
    void setHitpoints(int newHit);

};
```

Complete LogicGates

```
class Gate {

public:
    bool a = false;
    bool b = false;

    Gate() {};
    Gate(bool A, bool B) {
        a = A;
        b = B;
    };
    ~Gate() {delete &a; delete &b;};

};

class AND : public Gate {

public:
    AND() : Gate() {};
    ~AND() {};

    bool calculate() {
        return a && b;
    }
};


class OR : public Gate {

public:
    OR() : Gate() {};
    ~OR() {};

    bool calculate() {
        return a || b;
    }
};

class XOR : public Gate {

public:
    XOR() : Gate() {};
    ~XOR() {};

    bool calculate() {
        return (a != b);
    }
```

```
};



class HalfAdder : public XOR, AND  {

public:
    bool a = false;
    bool b = false;
    bool s = false;
    bool c = false;

    HalfAdder() : lXOR(), lAND() {};
    ~HalfAdder() {delete &lXOR; delete &lAND;};

    void calculate() {
        lAND.a = a;
        lAND.b = b;

        lXOR.a = a;
        lXOR.b = b;

        c = lAND.calculate();
        s = lXOR.calculate();
    }

private:
    XOR lXOR;
    AND lAND;
};



class FullAdder : HalfAdder, OR {

public:
    bool cin = false;
    bool a = false;
    bool b = false;
    bool s = false;
    bool cout = false;

    FullAdder() : addr(), lOR() {};
    ~FullAdder() {delete &addr; delete &lOR;};

    void calculate() {
        //using the same HalfAdder twice rather than two seperate
```

```
        addr.a = a;
        addr.b = b;
        addr.calculate();

        bool cTemp = addr.c;

        addr.a = cin;
        addr.b = addr.s;
        addr.calculate();

        lOR.a = addr.c;
        lOR.b = cTemp;

        s = addr.s;
        cout = lOR.calculate();
    }

private:
    HalfAdder addr;
    OR lOR;
};
```