

CS 7A – Spring '12 – Assignment #2 - Bin2Dec

Study the following as part of this assignment

1. **Algorithm 1.2, Conversion from Binary to Decimal**, and **Algorithm 1.3, Conversion from Binary to Decimal by Horner's Method** on page 5 of FCC++.
2. §2.9 *The Standard Library*, on pages 32 and 33 of FCC++, including the `<cmath>` file.
3. The `for` statement introduced in §3.8 on page 51 of FCC++.
4. §5.1 and §5.2 on pages 88 and 89 of FCC++, describing how to define, traverse and initialize an array. (Note that EXAMPLE 5.2 contains a syntax error. It should be `sqrt(10.0*i)`, with rounded parentheses, not square brackets. This is a function in the `<cmath>` file.)

Now, consider the C++ code below, which purports to implement Algorithm 1.2 in C++.

```
int main()
{
    double dec = 0.;
    // b will hold the binary number 10111011
    //(note that it's backwards with the least significant digit first)
    double b[8]={1.,1.,0.,1.,1.,1.,0.,1.};
    cout << "\ndec is initially = " << dec;
    for(int i=0;i<8;i++)
    {
        dec += b[i]*pow(2.,i); // inefficient!
        cout << endl << "dec is increased by b[" << i << "]*"
            << pow(2.,i) << " = " << b[i] << "*" << pow(2.,i)
            << " to " << dec;
    }
    cout << "\n\nSo the binary number ";
    for (int i=7;i>=0;i--) cout << b[i];
    cout << " is " << dec << " in decimal " << endl;
}
```

This program has some debugging code built into it so you can see more of what is happening while it's happening. The output of this code is

```
dec is initially = 0
dec is increased by b[0]*1 = 1*1 to 1
dec is increased by b[1]*2 = 1*2 to 3
dec is increased by b[2]*4 = 0*4 to 3
dec is increased by b[3]*8 = 1*8 to 11
dec is increased by b[4]*16 = 1*16 to 27
dec is increased by b[5]*32 = 1*32 to 59
dec is increased by b[6]*64 = 0*64 to 59
dec is increased by b[7]*128 = 1*128 to 187
```

So the binary number 10111011 is 187 in decimal

Copy this code into your compiler and get it to work. Note that you'll need both the `<iostream>` and `<cmath>` files from the Standard Library. Try modifying parts of it to help understand how it works.

Next, consider the C++ code below, which is significantly improved, but not quite Horner's method.

```
int main()
{
    int dec = 0.;
    // b will hold the binary number 10111011
    //(note that it's backwards with the least significant digit first)
    int b[8]={1,1,0,1,1,1,0,1};
    int power_of_2 = 1;
    cout << "\ndec is initially = " << dec;
    for(int i=0;i<8;i++)
    {
        dec += b[i]*power_of_2; // first multiplication
        cout << endl << "dec is increased by b[" << i << "]"*
            << power_of_2 << " = " << b[i] << "*"
            << power_of_2 << " to " << dec;
        power_of_2 *= 2; // second multiplication
    }
    cout << "\n\nSo the binary number ";
    for (int i=7;i>=0;i--) cout << b[i];
    cout << " is " << dec << endl;
}
```

The output of this code is identical to that of the previous program (check this by modifying your program) but the program does not require the `<cmath>` file and doesn't use the `pow()` function which won't take the first argument as an int. Thus the program itself is much shorter (no `<cmath>` header, requires less memory (int types instead of double) and runs faster (multiplication is quicker than exponentiation).

Horner's method (algorithm 1.3) significantly improves on this by doing about half as many multiplications.

```
int main()
{
    // b will hold the binary number 10111011
    //(note that it's backwards with the least significant digit first)
    int b[8]={1,1,0,1,1,1,0,1};
    int dec = b[7];
    cout << "\ndec is initially = " << dec;
    for(int i=6;i>=0;i--)
    {
        cout << "\n2*" << dec << " + " << b[i] << " = ";
        dec = 2*dec+b[i]; //Horner's method
        cout << dec;
    }
    cout << "\n\nSo the binary number ";
    for (int i=7;i>=0;i--) cout << b[i];
    cout << " is " << dec << endl;
}
```

The output of this code is

```
dec is initially = 1
2*1 + 0 = 2
2*2 + 1 = 5
2*5 + 1 = 11
2*11 + 1 = 23
2*23 + 0 = 46
2*46 + 1 = 93
2*93 + 1 = 187
```

So the binary number 10111011 is 187 in decimal form

Your assignment, should you choose to accept it, is to write a C++ program that will prompt the user for a number less than 256, convert that number to a binary number which is stored in an array of 8 digits, display that number and then use Horner's method to convert it back into decimal. There is a [WikiHow](#) on converting a decimal to binary which may help. The output of your program should look something like this:

```
Please enter a non-negative integer less than 256:
187

The decimal equivalent of 187 is 10111011

Horner's method converts this back to 187
```

As discussed in class, Horner's algorithm extends to evaluating more general polynomial forms. To see this, note that Horner's form of the binary representation of 187:

$$\begin{aligned} 10111011_2 &= 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2 + 1 \\ &= (1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2 + 1) 2 + 1 \\ &= ((1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2 + 0) 2 + 1) 2 + 1 \\ &= (((1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2 + 1) 2 + 0) 2 + 1) 2 + 1 \\ &= (((((1 \times 2^3 + 0 \times 2^2 + 1 \times 2 + 1) 2 + 1) 2 + 0) 2 + 1) 2 + 1) 2 + 1 \\ &= ((((((1 \times 2^2 + 0 \times 2 + 1) 2 + 1) 2 + 1) 2 + 0) 2 + 1) 2 + 1) 2 + 1) 2 + 1 \\ &= (((((((1 \times 2 + 0) 2 + 1) 2 + 1) 2 + 1) 2 + 0) 2 + 1) 2 + 1) 2 + 1 \end{aligned}$$

Can be generalized to an arbitrary seventh degree polynomial:

$$p(x) = a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$$
$$= \left(\left(\left(\left(\left(\left(\left(\left(a_7x + a_6 \right) x + a_5 \right) x + a_4 \right) x + a_3 \right) x + a_2 \right) x + a_1 \right) x + a_0 \right. \right. \right.$$

So the algorithm suggested by this is to start with the leading term a_n and then repeatedly

(1) multiply by x and (2) add the next a_{n-1} .

To reverse this, (1) put the remainder after division by x in the next significant digit and (2) divide by x .