# EC 504 – Spring 2023 – Homework Zero

**In class put the code on the CCS `/projectnb/ec504rb/students/yourname/HW0` You should put the code in another director such as `/projectnb/ec504rc/students/yourname/HW0_worki` For fun you can change some of the parameters. No grade but it will show me that you have the system down. In the mean time you should start to read the CLRS Chapters1, 2, and 3 Handouts on GitHub.**

## 1 $P^3$: Practice Power Program

The exercise is to go to the GitHub and get the prototype code in the file `EC504_2023/HW0_codes`. The code is a main program that calls 3 function to compute the power $x^N$ for large integer $N$.

The naive approach is to multiply N-times the value $x$

$$x^N = x * x * x * \cdots * x \tag{1}$$

Clearly it is take $O(N)$ multiples. Better way is a basic **Divide and Conquer** idea. Think of the binary representation of the integer $N$ with b+1 bits for any number $N \leq 2^{b+1} - 1$

$$N = n_0 + 2n_1 + 4n_2 + 8n_3 + \cdots + 2^b n_b \tag{2}$$

so that

$$x^N = (x)^{n_0+2n_1+4n_2+8n_3+\cdots+2^b n_b} = x^{n_0} * (x^2)^{n_1} * (x^4)^{n_2} * (x^8)^{n_3} * \cdots * (x^{2^b})^{n_b} \tag{3}$$

We can cleverly see there only $\log_2(N)$ factors are needed of the form $1, x, x^2, x^4, \cdots, x^b$ depending on if $n_k = 1$ or $n_k = 0$. The fast ones all use this idea. The first function is the standard C routine. (It actually has to convert the integer power to a floating point. Argh so silly but still uses this trick!). The next is the slow one. The final one the fast multiple bit idea – but I left out a line for you to complete. *That is the exercise – One line.*

```
double cPower(double x, long int N)
{
  return pow(x, (double)N);
}

double slowPower(double x, long int N)
{
  double pow = 1.0;
  int i;
  for( i = 0;  i < N; i++)
```

```
    {
  pow = x*pow;
     }
  // if(i < N)   cout <<"Slow Failed with iteration stop at i = " << i << endl;
  return pow ;
}

double  fastPower(double x, long int N)
{
  double factor = x; // holds x , x^2 , x^4, x^8, x^16,  etc
  double pow = 1.0;
  while(N > 0)
    {
      if(N%2) pow = factor*pow ; // Update pow
      N = N/2;
      // Update the binary factor by squaring to give the correct factors of x's.
    }
  return pow;
}
```

There is also a slick bit manipulation code, which I'm sure C must use for floating point exponents.

```
double veryfastPower(double x, long int N)
{
    double pow = 1.0;
    for (;;)
    {
      if (N & 1)  //Copies a bit to the result if it exists in both operands.
            pow *= x;
      N >>= 1;  //Binary Right Shift Operator.
      if (!N)  // Check for zero
            break;
        x = x*x;
    }

    return pow;
}
```

The program is compiled automatically by putting `myPower.cpp` in a directory with `makefile` and typing `make -k` on the command line. Then it will run by typing `.\power`

Ok at this point you make a directory `/projectnb/alg504/yourname/HW0` and move it there. Do this right away and we can see if everything is set to go. Then you can fix up the fastPower and see how much faster it is.

## Small things to do.

You should read and understand the code as is. Many of this C features will be used in coding exercises. (By the way C and C++ are really same language – C++ just a few, not always better, extra `bells and whistles`) All compiler are now C++ compilers.

Also there is one missing code line in

`double fastPower(double x, long int N)`.

I have put the correct output file and a figure in `HW0_codes` to show what the correct output is when you fix it. If you want you can change parameters and graph the data in whatever way you want. I have also give a figure done with `gnuplot` to compare performance scaling. Gnuplot is nice unix utility which we will use later but you may have other graphing programs you like. Anyway this is **no grade practice problem**. **Full credit if we can snarf it from your `HW0` directory in your SCC account.** Don't worry about details now but play with the code so you understand the C coding style.

Cheers,

Rich