### 大数计算项目报告

### 陈泰然 PB14210070

在这个项目中，我用如下结构体来存储大整数：

输入时number中被输入数字所对应字符，然后transform\_numchar\_to\_number函数将其转化为数字，然后在计算中使用。

typedef struct number{

int positive; //这个数的正负

char digit[TLEN]; //这个数的各个数位（高位在高下标）

int len; //这个数的有效长度

int format; //digit中存储的是数字还是数字所对应的字符

}Number;

在这个项目中我将关于大数的运算分为内部运算和外部运算两类，内部运算（如internal\_add函数）仅适用于外部运算函数（如add函数）的调用。在main函数中调用operate函数，operate函数根据不同运算符调用不同的外部运算函数，外部运算函数再来调用对应的内部运算函数来完成具体的运算。

在做乘法运算时，有两个内部运算函数可供选择：

internal\_mul\_fft使用FFT算法，在位数较大时运算速度优势十分明显；

internal\_mul\_normal使用普通算法，在小位数运算时占优势。

其中FFT算法计算大数乘法的思路如下：

任意两个整数相乘可以看作这样的一个过程（以123456\*987654为例）：

123456\*987654=121931812224

123456\*987654

=(6\*10^0+5\*10^1+4\*10^2+3\*10^3+2\*10^4+1\*10^5)\*

(4\*10^0+5\*10^1+6\*10^2+7\*10^3+8\*10^4+9\*10^5)

=(6\*4)\*10^0+(5\*4+6\*5)\*10^1+(4\*4+5\*5+6\*6)\*10^2+...+(2\*9+1\*8)\*10^9+(1\*9)\*10^10

如果将10的指数看作一个数组的下标的话，则上式可以写成：

[6,5,4,3,2,1]\*[4,5,6,7,8,9]=[24,50,77,104,130,154,115,80,50,26,9]

容易看出，这不是别的，正是序列[6,5,4,3,2,1]和序列[4,5,6,7,8,9]的卷积。而卷积可以利用FFT计算，速度是非常快的。

另外在乘法普通算法中，我没有直接计算乘法，而是将九九乘法表中积的个位和十位分开存放于两个二维数组中，这样就不用先计算乘法再将积的个位和十位分开了，减少了近20%的运算时间。

至于数的正负号，以及某些特殊值的运算，都在外部运算函数中进行了专门处理，从而保证了传入内部运算函数的参数都是正确无误的，使程序更易维护。