1. 数据结构

为了模块化编程，对大整数做一个结构封装，使用结构体MBigInt表示一个多精度的大整数结构：

struct MBigInt

{

int alloclen;//记录数组已经分配的总空间大小

int length;//记录大整数长度

int sign;//记录大整数的符号

un\_short \*pBigInt;//指向大整数数据的指针

};

1. 基本函数定义
2. 大整数初始化操作

函数initMBInt(MbigInt \*bint, int size )用于初始化一个大整数，默认长度为49字节。如果输入大小size小于49，仍会分配49字节的空间；否则在49的基础上以步长16为增量递加直至大于size。

1. 大整数销毁操作

函数deleteMBInt(MbigInt \*bint)用于释放大整数的堆内存，并将符号标记为非负。

1. 大整数的内存扩展

函数extendMBInt(MBigInt \*bi, int size)以16为增量在原来的空间大小基础上递加直到大于size，并分配相应大小内存。

1. 大整数的输入输出函数

函数read\_radix(MBigInt \*bi,char \*str)是将十进制表示的字符串读入并转换为大整数结构表示；函数write\_radix(MBigInt \*bi,char \*str)将大整数转换为十进制的字符串表示。

1. 加法运算

函数addMBInt1(MBigInt \* dst, MBigInt \*src1, MBigInt \*src2)实现两个大整数的加法操作。因为两个整数的符号相同，可以直接对两个整数数组对应位元素相加，并考虑进位问题。因为两个整数的数位存在以下三种情况（不妨设两个大整数分别为src1和src2）：src1->length == src2->length；src1->length > src2->length； src1->length < src2->length。所以在处理加法的时候，对数位较大的整数一分为二，第一部分是跟另一大整数的长度相等。可以先把相同长度部分先计算，然后再对多出的长度部分直接赋值，这样可以不用考虑谁大谁小问题，而且可以加快运算速度。如下所示：

表1 分段运算过程

|  |  |
| --- | --- |
| 1 2 3 | 4 5 6 7 8 9 |
| — — — | 4 5 6 7 8 9 |
| — — — | 9 1 3 5 7 8 |
| 1 2 3 | 9 1 3 5 7 8 |

1. 减法运算

函数addMBInt2(MBigInt \*dst, MBigInt \*src1, MBigInt \*src2)实现两个大整数的减法操作。先对两个大整数进行无符号比较，较大者作为被减数，然后进行减法操作。计算结果的符号由两个大整数的大小关系决定。

1. 乘法运算

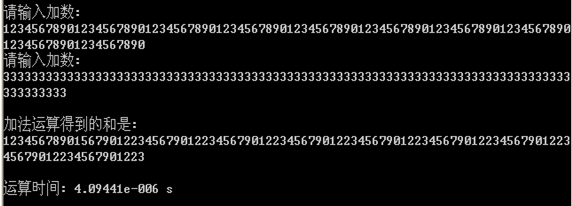
函数mulBasicMBInt(MBigInt \*product, MBigInt \*biA, MBigInt \*biB)实现两个大整数的乘法操作。本函数采用笔算乘法算法，算法复杂度为O()。用乘数的每一位乘以被乘数的每一位并加上上一列的进位而产生一行适当移位的中间结果，然后再将各行中间结果相加即得到乘法的最终结果。本算法伪代码如下：

|  |
| --- |
| 输入：分别含有n和t个B进制数位的大整数x、y  输出：B进制的乘积 |
| 1. i从0到n+t-1,置Wi = 0  2. i从0到t  2.1 c ＝ 0  2.2 j从0到n  计算  置  2.3  3. 返回 |

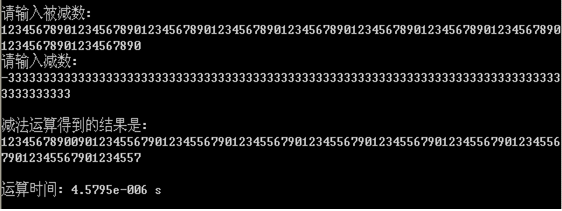
1. 除法运算

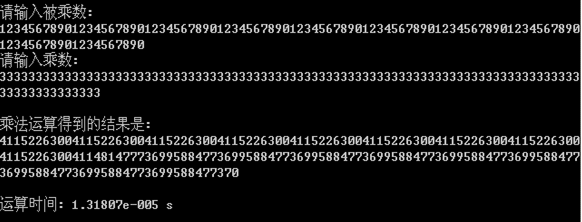
函数divMBInt(MBigInt \*a, MBigInt \*b, MBigInt \*c, MBigInt \*d)实现两个大整数的除法操作。本函数采用笔算除法的形式，从被除数的最高位开始，根据被除数和除数的最高几位来试商。使用二分查找法可提高试商的速度。

1. 结果

加法运算：  


减法运算：



乘法运算：  
 

除法运算：

最初版函数有一些bug，目前仍在解决中…………