# 链 表 专 题 实 践 报 告

## ——超高精度大数的运算

*Jammy Zeta*

# 目录

# 一. 摘要

运算是每一台计算机必备的技能，也是日常人们最普遍的需求。但往往由于编程语言体系中数字类型的存储空间有限，不能进行大数的存储与运算，数据的精确度也大大受限。C 语言作为一种经典的编程语言也存在着这样棘手的问题，其中的 int 型整数最大只能取到 $2^{31}$-1，这只是一个十位的整数，即便是 long long int 最大取值也未能超过 20 位数；而 double 型浮点数更是只能保留十五位小数，在一些领域上还难以达到高精确度的要求。因此本报告旨在运用 C 语言中的链表来实现超高精度有符号大（实）数的比较、加、减以及乘法运算。选择链表而非数组的原因在于创建大数时相较于数组更加灵活，也能够更充分地利用存储空间。

# 二. 问题分析

1. 由于加、减、乘运算均是从小位开始，而大小比较则是从大位开始，因此单向链表难以一次性实现，故采用双向链表
2. 由于参与运算的对象是有符号实数，因此必须记录每个大数的符号以及小数点的位置，而这些信息可以存放在大数的头节点当中
3. 三种运算中加和减很显然是等价的，而减的运算结果则可用以判断大数大小。然而事实上问题并没有这么简单，因为在异号大数的加法运算中我们需要判断两个大数的绝对值大小才能进行操作。因此这时候只能从其中一方切入来打破这个闭环。显而易见的是，比较大小最易实现
4. 对于比较大小，可以结合符号位与两者的整数位数分情况讨论
5. 对于加法运算，我们必须将两者对应的位数相加。然而两个大数的位数不一定相等，故为了简化后续操作，在加法运算之前，先将两个大数的前端与后端补上相应的 0(`zeroBigNumber`)使两者的整数位与小数位均相等，再进行从小位开始的逐位操作，可分为同号相加与异号相加两种情况
6. 对于减法运算事实上已经在加法运算中实现了，只需要将减数的符号位取反之后再和第一个大数参与加法运算即可
7. 对于乘法运算事实上等价于整数的乘法，无非还需要记录小数点的位置。根据严格的数学证明我们可以知道以下事实：

   一个$\alpha$位整数与一个$\beta$位整数相乘的结果只会是一个$(\alpha+\beta)$位或$(\alpha+\beta-1)$位整数.

   因此在乘法运算之前我们可以先读出两个大数的整数位数和小数位数，全部相加后的值作为结果大数的位数创建相应链表，将链表中每一个结构体的存储数值初始化为 0，然后根据列竖式的原理进行相乘再相加
8. 考虑到每一次运算之后可能会产生无效的前端 0 与后端 0，因此每次运算结束后均要对结果大数进行去 0 化处理（`nmlBigNumber`）

# 三． 函数说明

| 函数声明 | 说明 |
| --- | --- |
| sgl# *scanBigNumber() | 读入大数，构建链表，<br>返回读入大数的头节点 |
| void zeroBigNumber<br>(sgl *heada, sgl *headb) | 用补零的方式将两个大数的小数位数<br>与整数位数补成相等的 |
| void nmlBigNumber(sgl *heada) | 与 zeroBigNumber 相反，去掉大数<br>末尾与开头的 0 |
| void printBigNumber(sgl *head) | 输入头节点，输出大数 |
| int cmpBigNumber<br>(sgl *heada, sgl *headb) | 比较两个大数之间的大小，<br>返回整数表示判断结果 |
| int jdgBigNumber<br>(sgl *heada, sgl *headb) | 基于上一个函数比较两个大数绝对值<br>的大小，同样返回整数表示判断结果 |
| sgl *addBigNumber<br>(sgl *heada, sgl *headb) | 将两个大数相加返回结果<br>大数（和）的头节点 |
| sgl *sbtBigNumber<br>(sgl *heada, sgl *headb) | 基于上一个函数将两个大数相减返回<br>结果大数（差）的头节点 |
| sgl *mtpBigNumber<br>(sgl *heada, sgl *headb) | 将两个大数相乘返回结果<br>大数（积）的头节点 |
| void freeBigNumber(sgl *head) | 释放存储大数链表的空间 |
| void printERROR() | 读入非法字符（非数字与取负符号）<br>时报错并退出程序 |

# sgl 为存储大数链表的单个结构体名称

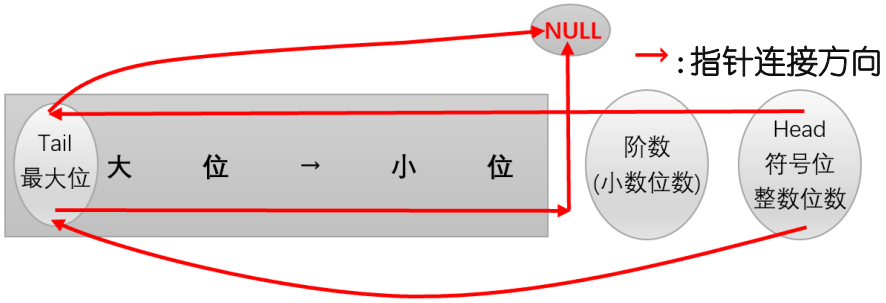
# 四． 算法说明

1. 存储大数链表中的单个结构体定义如下：

```
typedef struct single
{   struct single *pre;
    int n;
    struct single *next;} sgl;
```

其中 pre 存储的是位数大一位的结构体指针，next 则是位数小一位的指针

2. 存储大数链表的基本结构与一般双链表不同，具体示意图及说明如下：




① Head 为一个大数的头节点，其中存储符号位*整数位数的数值。指针从 pre 方向可依次读到阶数(大数的小数位数)以及从低位到高位的每位数值；而头节点的 next 指针则连接着大数的最高位(Tail)，可以从高位到低位依次读取大数各位数

② Head->pre 也是一个用于存储大数信息的节点，其用于存储大数的阶数，即小数位数。运算时读取相应大数前可以通过该节点获知小数点的位置，而后再进行相应的运算

③ 这里只有两个指针指向了 NULL，分别是最高位->pre 与最小位->next。最高位->pre=NULL 可以在运算时表示大数已读完;最小位->next=NULL 可以在比较时表示大数已读完，而且后面两个节点不可能从大数的最低位去读取，因此可以指向 NULL

④ 可见头节点是最重要的，通过一个大数的头节点即可方便地获取大数的相关信息并能够从不同的方向读取大数

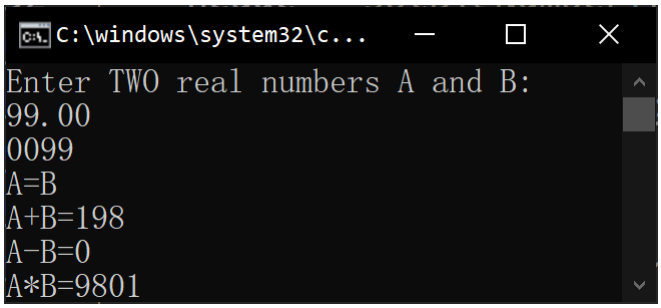
3. 读入大数时一边构建链表一边计数，读到小数点时再开始另外计数。先判断大数的首字符是否为'-',以此标记符号位，然后继续读取直到回车出现即表示大数读取完成，而后再创建两个节点将之前信息存入，完善链表结构
4. 由于读取时 scanf 的是字符,因此程序可能会把非数字字符读入并将其 ASCII 码存入结构体后进行运算。为避免该情况出现，设立报错函数 printERROR()，其作用是一旦检测到非法字符，报错并退出程序
5. 考虑到一开始输入的大数中可能会包含不影响数值的前端 0 与后端 0，而这些 0 可能会影响后续操作并增加不必要的处理时间，因此在读入两个大数之后，先对大数进行去 0 化处理（nmlBigNumber）
6. 比较运算中用整型变量 judge 的值（1：＞，0：=，-1：小于）代表比较结果，主要分以下三种情形讨论：
   a) 若正数与负数相比较，可直接得出正数＞负数的结果
   b) 如若同号，则先记录符号位（以 1 与-1 代表正负），再比较两者的整数位数，整数位数更大者对应绝对值必然更大，再乘以符号位即可
   c) 如若同号且整数位数相同，则从大位开始向小位逐一比较，一旦遇到两者对应位数的数值不相同或（仅）其中一个大数已到达最后一位时，跳出循环并乘以符号位，返回结果
7. 加法运算中的两种情况处理方式如下：
   a) 同号相加：记录符号位，设置一个记录进位数的整型变量,从小位开始逐一相加，在相加的过程中构建链表，最后返回结果大数的头节点

b) 异号相加：事实上即为减法运算。先比较两者的绝对值大小，记录绝对值更大者的符号位，然后用绝对值较大值减去绝对值较小值，思路与加法类似，其中进位变量记录的则是借位数，在下一位运算时减去即可

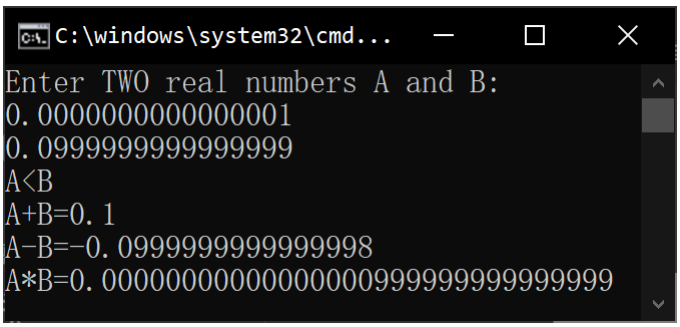
8. 减法运算中为在改变符号时不改变原大数本身的符号位，在函数中创建一个临时结构体变量使其等于第二个大数(减数)头节点所指的结构体，改变符号位后引用加法函数得出结果
9. 乘法运算中先创建一个数值均为 0 且节点数为两大数位数之和的链表，然后进行双重循环，使 a 的每一位数与 b 的每一位数相乘，每一次内循环结束之后相加的起点均要在链表中后移一位(仿照列竖式的方式每次乘出来的结果均往左移一位)，因此需要一个指针变量来控制
10. zeroBigNumber 与 nmlBigNumber 均是对原大数直接进行操作，包括大数头节点中的相关信息

# 五. 部分样例展示

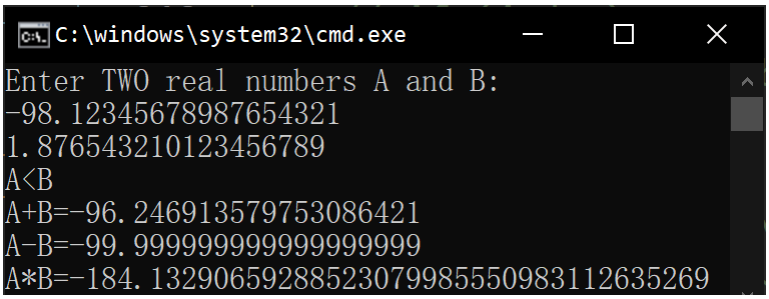
## Sample 1



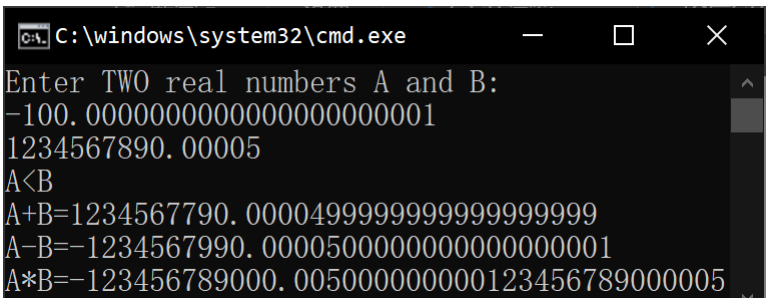

## Sample 2



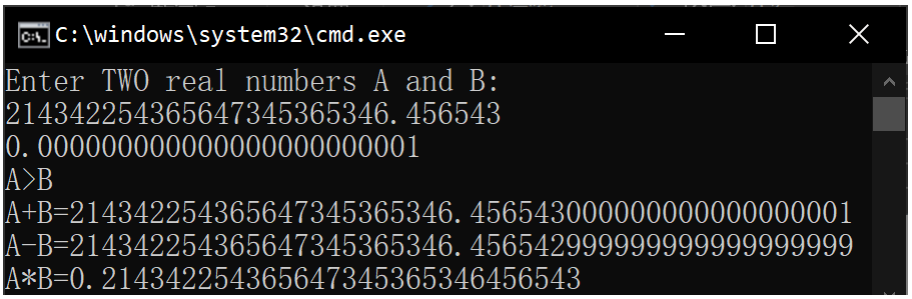
## Sample 3



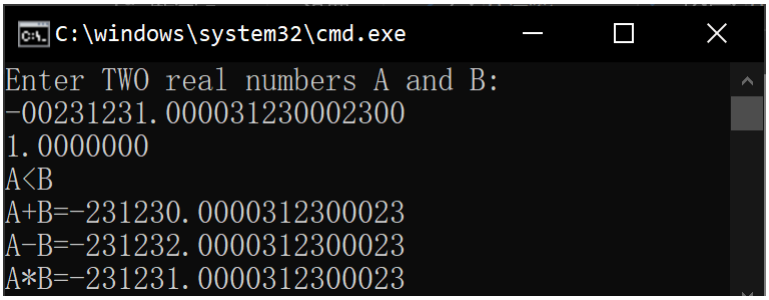

## Sample 4



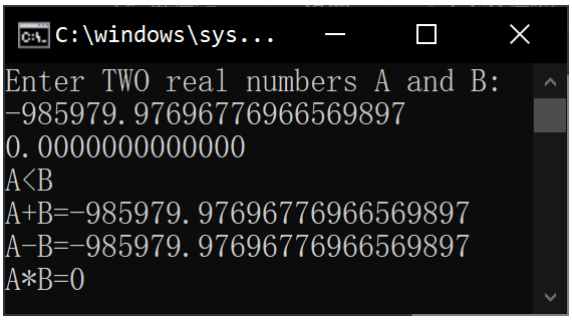

## Sample 5



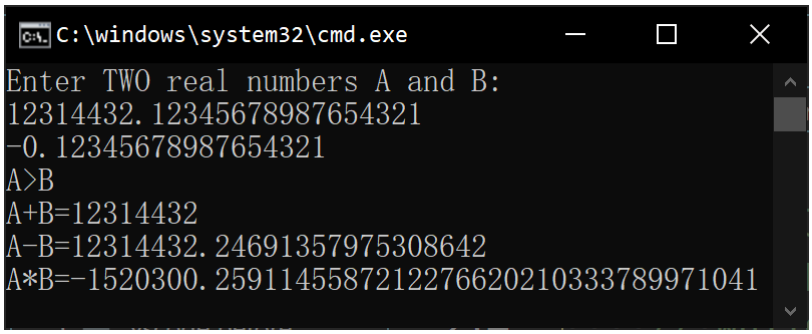

## Sample 6



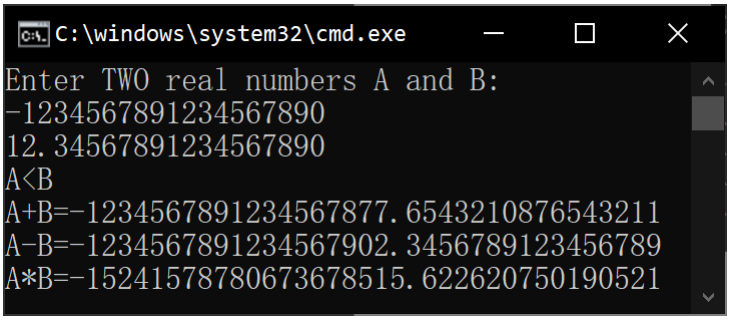
## Sample 7



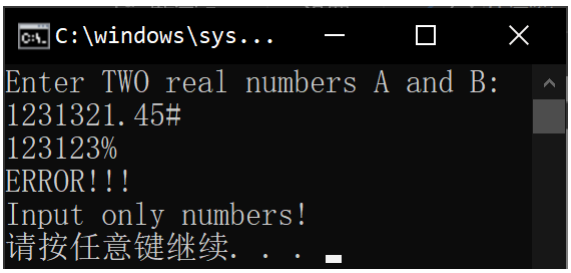

## Sample 8




## Sample 9

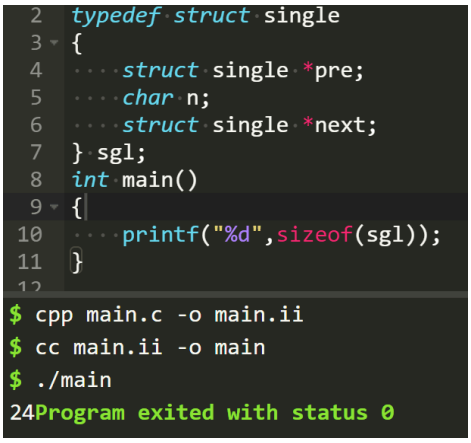


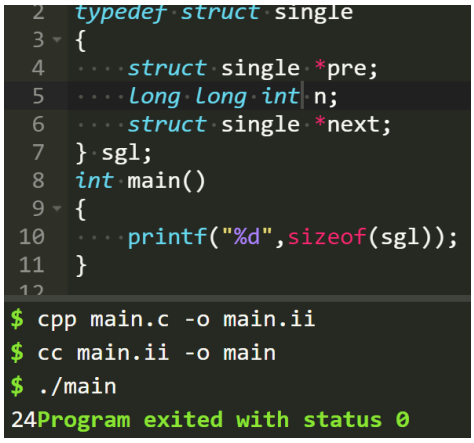

## Sample 10



# 六. 不足与反思

1. 结构体中存储数据的类型是 int，但我们只用其存放了一位数，然而却占用了四个字节，这是极大的空间浪费！尤其是当数据非常庞大的时候。起初我原本打算在结构体中用 char 代替 int,但事实证明这是一种徒劳,详见下图：




这才依稀想起曾经有同学讲过结构体中的存储格式比较特殊，不同类型的变量在内存中会按所占空间最大的变量对齐，因此由于该结构体中含有结构体指针（占 8 字节）导致不论是 int 还是 char 这个结构体的所占空间都为 3*8=24 字节。如此看来，想要节省空间便不是一件简单的事情了。

2. 链表的结构或许还可以优化，两个头节点存储大数信息还是偏多了，而且链表整体结构偏复杂，应考虑只用一个头节点存储大数信息并优化链表结构，可以为后续算法带来很多益处

3. 乘法算法可以考虑 Karatsuba 大数乘法降低时间复杂度

# 七. 附录（源代码）

```
#include <stdio.h>
#include <stdlib.h>
#define sgn(x) (2 * (x->n > 0) - 1)
#define max(x, y) ((x > y) ? x : y)
#define dgt(x) (abs(x->n) + x->pre->n)
typedef struct single
{
    struct single *pre;
    int n;
    struct single *next;
} sgl;
sgl *scanBigNumber();
void zeroBigNumber(sgl *heada, sgl *headb);
void nmlBigNumber(sgl *heada);
void printBigNumber(sgl *head);
int cmpBigNumber(sgl *heada, sgl *headb);
int jdgBigNumber(sgl *heada, sgl *headb);
sgl *addBigNumber(sgl *heada, sgl *headb);
sgl *sbtBigNumber(sgl *heada, sgl *headb);
sgl *mtpBigNumber(sgl *heada, sgl *headb);
void freeBigNumber(sgl *head);
void printERROR();
int main()
{
    printf("Enter TWO real numbers A and B:\n");
    sgl *heada, *headb, *headc;
    heada = scanBigNumber();
    headb = scanBigNumber();
    nmlBigNumber(heada);
    nmlBigNumber(headb);
    switch (cmpBigNumber(heada, headb))
    {
```

```
    case 1:
        printf("A>B");
        break;
    case 0:
        printf("A=B");
        break;
    case -1:
        printf("A<B");
        break;
    }
    printf("\n");
    headc = addBigNumber(heada, headb);
    printf("A+B=");
    printBigNumber(headc);
    freeBigNumber(headc);
    headc = sbtBigNumber(heada, headb);
    printf("A-B=");
    printBigNumber(headc);
    freeBigNumber(headc);
    headc = mtpBigNumber(heada, headb);
    printf("A*B=");
    printBigNumber(headc);
    freeBigNumber(headc);

    freeBigNumber(heada);
    freeBigNumber(headb);
}
sgl *scanBigNumber()
{
    char s;
    int sgn, rcnt = 0, lcnt = 1;
    sgl *phead, *tmp, *tail;
    scanf("%c", &s);
    if (s == '-')
```

```
    {
        sgn = -1;
        tail = phead = (sgl *)malloc(sizeof(sgl));
        phead->n = getchar() - '0';
        phead->next = phead->pre = NULL;
    }
    else if (s >= '0' && s <= '9')
    {
        sgn = 1;
        tail = phead = (sgl *)malloc(sizeof(sgl));
        phead->n = s - '0';
        phead->next = phead->pre = NULL;
    }
    else
        printERROR();
    while ((s = getchar()) != '\n')
    {
        if ((s < '0' || s > '9') && s != '.')
            printERROR();
        if (s != '.')
        {
            tmp = (sgl *)malloc(sizeof(sgl));
            tmp->pre = phead;
            tmp->n = s - '0';
            tmp->next = NULL;
            phead->next = tmp;
            phead = tmp;
            ++lcnt;
        }
        else
        {
            while ((s = getchar()) != '\n')
            {
                tmp = (sgl *)malloc(sizeof(sgl));
```

```
                tmp->pre = phead;
                tmp->n = s - '0';
                tmp->next = NULL;
                phead->next = tmp;
                phead = tmp;
                ++rcnt;
            }
            break;
        }
    }
    tmp = (sgl *)malloc(sizeof(sgl));
    tmp->pre = phead;
    tmp->n = rcnt;
    tmp->next = NULL;
    phead = (sgl *)malloc(sizeof(sgl));
    phead->pre = tmp;
    phead->n = sgn * lcnt;
    phead->next = tail;
    return phead;
}
void zeroBigNumber(sgl *heada, sgl *headb)
{
    if (abs(heada->n) == abs(headb->n) && heada->pre->n == hea
db->pre->n)
        return;
    sgl *zerofront = (abs(heada->n) > abs(headb->n)) ? headb :
 heada,
        *zerorear = (heada->pre->n > headb->pre->n) ? headb->p
re : heada->pre,
        *tmp = zerofront->next;
    int fdelta = abs(abs(heada->n) - abs(headb->n)),
        rdelta = abs((heada->pre->n - headb->pre->n));
    zerofront->n = sgn(zerofront) * max(abs(heada->n), abs(hea
db->n));
```

```
    zerorear->n = max(heada->pre->n, headb->pre->n);
    sgl *target = zerofront;
    zerofront = zerofront->next;
    while (fdelta--)
    {
        tmp = (sgl *)malloc(sizeof(sgl));
        tmp->n = 0;
        tmp->next = zerofront;
        zerofront = zerofront->pre = tmp;
    }
    tmp->pre = NULL;
    target->next = tmp;
    tmp = zerorear->pre;
    target = zerorear;
    zerorear = zerorear->pre;
    while (rdelta--)
    {
        tmp = (sgl *)malloc(sizeof(sgl));
        tmp->n = 0;
        tmp->pre = zerorear;
        zerorear = zerorear->next = tmp;
    }
    tmp->next = NULL;
    target->pre = tmp;
}
void nmlBigNumber(sgl *head)
{
    if (head->next->n != 0 && head->pre->pre->n != 0)
        return;
    int sgn = sgn(head);
    sgl *tmp = head->next, *cup;
    int *lcnt = &head->n, *rcnt = &head->pre->n;
    head->n = abs(head->n);
    while (tmp->n == 0 && *lcnt > 1)
```

```
    {
        --*lcnt;
        cup = tmp->next;
        free(tmp);
        tmp = cup;
    }
    tmp->pre = NULL;
    head->next = tmp;
    head->n *= sgn;
    tmp = head->pre->pre;
    while (tmp->n == 0 && *rcnt > 0)
    {
        --*rcnt;
        cup = tmp->pre;
        free(tmp);
        tmp = cup;
    }
    tmp->next = NULL;
    head->pre->pre = tmp;
    if (!*rcnt && head->next->n == 0)
        head->n = abs(head->n);
}
void printBigNumber(sgl *head)
{
    int lcnt = head->n,
        rcnt = head->pre->n;
    if (lcnt < 0)
    {
        printf("-");
        lcnt *= -1;
    }
    head = head->next;
    while (lcnt--)
    {
```

```
        printf("%d", head->n);
        head = head->next;
    }
    if (rcnt)
        printf(".");
    while (rcnt--)
    {
        printf("%d", head->n);
        head = head->next;
    }
    printf("\n");
}
int cmpBigNumber(sgl *heada, sgl *headb)
{
    int judge = 0;
    if (heada->n > headb->n)
        judge = 1;
    else if (heada->n < headb->n)
        judge = -1;
    else
    {
        int sgn = sgn(heada);
        heada = heada->next;
        headb = headb->next;
        while (heada && headb)
        {
            if (heada->n > headb->n)
            {
                judge = 1;
                break;
            }
            else if (heada->n < headb->n)
            {
                judge = -1;
```

```
                break;
            }
            else
            {
                heada = heada->next;
                headb = headb->next;
            }
        }
        if (heada && !headb)
            judge = 1;
        else if (!heada && headb)
            judge = -1;
        judge *= sgn;
    }
    return judge;
}
int jdgBigNumber(sgl *heada, sgl *headb)
{
    int judge;
    sgl a = *heada, b = *headb;
    a.n = abs(a.n);
    b.n = abs(b.n);
    if (cmpBigNumber(&a, &b) == 1)
        judge = 0;
    else
        judge = 1;
    return judge;
}
sgl *addBigNumber(sgl *heada, sgl *headb)
{
    int sgn = jdgBigNumber(heada, headb);
    zeroBigNumber(heada, headb);
    sgl *headc = (sgl *)malloc(sizeof(sgl)),
        *former = (sgl *)malloc(sizeof(sgl)),
```

```
        *starta, *startb, *tmp;
    headc->n = abs(heada->n);
    former->n = heada->pre->n;
    former->next = NULL;
    headc->pre = former;
    int add = 0, carry = 0;
    if (sgn(heada) * sgn(headb) > 0)
    {
        starta = heada->pre->pre;
        startb = headb->pre->pre;
        sgn = sgn(heada);
        while (starta)
        {
            add = starta->n + startb->n + carry;
            carry = add / 10;
            add = add % 10;
            tmp = (sgl *)malloc(sizeof(sgl));
            tmp->n = add;
            tmp->next = former;
            former = former->pre = tmp;
            starta = starta->pre;
            startb = startb->pre;
        }
        if (carry)
        {
            tmp = (sgl *)malloc(sizeof(sgl));
            tmp->n = carry;
            tmp->next = former;
            former->pre = tmp;
            headc->n += 1;
        }
        headc->n *= sgn;
    }
    else
```

```
    {
        starta = sgn ? headb : heada;
        startb = sgn ? heada : headb;
        headc->n *= sgn(starta);
        starta = starta->pre->pre;
        startb = startb->pre->pre;
        while (starta)
        {
            add = starta->n - startb->n - carry;
            if (add < 0)
            {
                carry = 1;
                add += 10;
            }
            else
                carry = 0;
            tmp = (sgl *)malloc(sizeof(sgl));
            tmp->n = add;
            tmp->next = former;
            former = former->pre = tmp;
            starta = starta->pre;
            startb = startb->pre;
        }
    }
    tmp->pre = NULL;
    headc->next = tmp;
    headc->pre->pre->next = NULL;
    nmlBigNumber(heada);
    nmlBigNumber(headb);
    nmlBigNumber(headc);
    return headc;
}
sgl *sbtBigNumber(sgl *heada, sgl *headb)
{
```

```
    sgl sbt = *headb;
    sbt.n *= -1;
    return addBigNumber(heada, &sbt);
}
sgl *mtpBigNumber(sgl *heada, sgl *headb)
{
    sgl *headc = (sgl *)malloc(sizeof(sgl)),
        *former = (sgl *)malloc(sizeof(sgl)),
        *tmp;
    former->n = heada->pre->n + headb->pre->n;
    former->next = NULL;
    int digit = dgt(heada) + dgt(headb);
    headc->n = (digit - former->n) * sgn(heada) * sgn(headb);
    headc->pre = former;
    for (int i = 0; i < digit; ++i)
    {
        tmp = (sgl *)malloc(sizeof(sgl));
        tmp->n = 0;
        tmp->next = former;
        former = former->pre = tmp;
    }
    tmp->pre = NULL;
    headc->pre->pre->next = NULL;
    headc->next = tmp;
    sgl *starta = heada->pre->pre,
        *startb = headb->pre->pre,
        *start = headc->pre->pre;
    tmp = headc->pre->pre;
    int mtp = 0, carry = 0;
    while (starta)
    {
        startb = headb->pre->pre;
        while (startb)
        {
```

```
            mtp = startb->n * starta->n + carry;
            tmp->n += mtp;
            carry = tmp->n / 10;
            tmp->n = tmp->n % 10;
            tmp = tmp->pre;
            startb = startb->pre;
        }
        if (carry)
        {
            tmp->n += carry;
            carry = 0;
        }
        starta = starta->pre;
        tmp = start = start->pre;
    }
    nmlBigNumber(headc);
    return headc;
}
void freeBigNumber(sgl *head)
{
    sgl *tmp = head;
    while (tmp)
    {
        tmp = head->pre;
        free(head);
        head = tmp;
    }
}
void printERROR()
{
    printf("ERROR!!!\nInput only numbers!");
    exit(0);
}
```