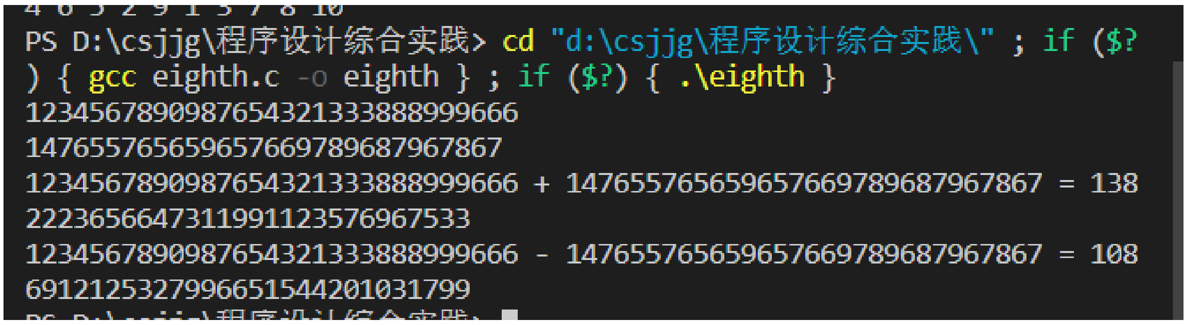
```c
#include <stdio.h>
#include <malloc.h>
#include <assert.h>

struct  Node
{
    int digit;
    struct Node *next, *prev;
};
struct UBigNumber
{
    int     digitCount;
    struct Node *pHead, *pTail;
};

struct UBigNumber InputUBN ();
void PrintUBN (struct UBigNumber ubn);
struct UBigNumber AddUBN (struct UBigNumber *pA, struct UBigNumber *pB);
void DestoryUBN (struct UBigNumber *pA);
void _InitUBN (struct UBigNumber *pUBN);
void _AppendDigit (struct UBigNumber *pUBN, int digit);
void _AppendFrontDigit (struct UBigNumber *pUBN, int digit);
void _Normalize (struct UBigNumber *pUBN);
struct Node *_NewNode ();
struct UBigNumber MyMinus(struct UBigNumber *pA, struct UBigNumber *pB);

int main ()
{   struct UBigNumber A, B, C,D;
    A = InputUBN ();
    B = InputUBN ();
    C = AddUBN (&A, &B);
    D = MyMinus(&A, &B);

    PrintUBN (A);
    printf (" + ");
    PrintUBN (B);
    printf (" = ");
    PrintUBN (C);
    printf("\n");
    PrintUBN (A);
    printf (" - ");
    PrintUBN (B);
    printf (" = ");
    PrintUBN (D);

    DestoryUBN (&A);
    DestoryUBN (&B);
    DestoryUBN (&C);
    return 0;
}


struct UBigNumber InputUBN ()
{
    struct UBigNumber result;
    _InitUBN(&result);

    char ch;
    do
        ch = getchar ();
    while (ch < '0' || ch > '9');
    while (ch >= '0' && ch <= '9')
    {
        _AppendDigit (&result, ch - '0');
        ch = getchar ();
    }
    _Normalize(&result);
    return result;
}
void PrintUBN (struct UBigNumber ubn)
{
    assert (ubn.digitCount > 0 && ubn.pHead->next != NULL);
    struct Node *la = ubn.pHead->next;
    while (la)
    {
        printf ("%d", la->digit);
        la = la->next;
    }
}
struct UBigNumber MyMinus (struct UBigNumber *pA, struct UBigNumber *pB)
{
    struct UBigNumber result, *pResult = &result;
    int flag=0;
    _InitUBN(pResult);
    struct Node *p1, *p2;
    p1 = pA->pTail;
    p2 = pB->pTail;
    while (p1 != pA->pHead && p2 != pB->pHead)
    {
        int digit = p1->digit - p2->digit +flag;
        flag=0;
        if(digit>=0){
        _AppendFrontDigit (pResult, digit);
        p1 = p1->prev;
        p2 = p2->prev;
        }else{
        _AppendFrontDigit (pResult, digit+10);
        p1 = p1->prev;
        p2 = p2->prev;
        flag=-1;
        }
    }
    while (p1 != pA->pHead->next && p1 != pA->pHead)
    {
        int digit = (p1->digit) + flag;
        flag=0;
        if(digit<0){
        _AppendFrontDigit (pResult, digit+10);
        p1 = p1->prev;
        flag=-1;;
        }else{
        _AppendFrontDigit (pResult, digit);
        p1 = p1->prev;
        }
    }
    if((p1->digit +flag)>0 && p1 == pA->pHead->next){
        _AppendFrontDigit (pResult, p1->digit+flag);
    }else{

    }

    return result;
}
struct UBigNumber AddUBN (struct UBigNumber *pA, struct UBigNumber *pB)
{
    struct UBigNumber result, *pResult = &result;
    _InitUBN(pResult);
    int iCarry = 0;
    struct Node *p1, *p2;
    p1 = pA->pTail;
    p2 = pB->pTail;
    while (p1 != pA->pHead && p2 != pB->pHead)
    {
        int digit = p1->digit + p2->digit + iCarry;
        iCarry = digit / 10;
        digit %= 10;
        _AppendFrontDigit (pResult, digit);
        p1 = p1->prev;
        p2 = p2->prev;
    }
    while (p1 != pA->pHead)
    {
        int digit = p1->digit + iCarry;
        iCarry = digit / 10;
        digit %= 10;
        _AppendFrontDigit (pResult, digit);
        p1 = p1->prev;
    }
        while (p2 != pB->pHead)
    {
        int digit = p2->digit + iCarry;
        iCarry = digit / 10;
        digit %= 10;
        _AppendFrontDigit (pResult, digit);
        p2 = p2->prev;
    }
    if (iCarry != 0)
        _AppendFrontDigit (pResult, iCarry);
    return result;
}
void DestoryUBN (struct UBigNumber *pUBN)
{
    while (pUBN->pHead != NULL)
    {
        struct Node *p =  pUBN->pHead;
        pUBN->pHead = p->next;
        free (p);
    }
}
void _InitUBN (struct UBigNumber *pUBN)
{
```

```c
    struct Node *p = _NewNode ();
    pUBN->pHead = pUBN->pTail = p;
    p->next = p->prev = NULL;
    pUBN->digitCount = 0;
}
void _AppendDigit (struct UBigNumber *pUBN, int digit)
{
    if (pUBN->digitCount == 1 && pUBN->pTail->digit == 0)
    {//直到出现非0数字才可以结束
        pUBN->pTail->digit = digit;
        return;
    }
    struct Node *p = _NewNode ();  //数字链表添加一个结点
    p->digit = digit;
    p->next = NULL;
    p->prev = pUBN->pTail;
    pUBN->pTail->next = p;
    pUBN->pTail = p;
    ++pUBN->digitCount;
}
void _AppendFrontDigit (struct UBigNumber *pUBN, int digit)
{
    struct Node *p = _NewNode ();
    p->digit = digit;
    p->next = pUBN->pHead->next;
    if (p->next != NULL)
        p->next->prev = p;
    p->prev = pUBN->pHead;
    pUBN->pHead->next = p;
    if (pUBN->pTail == pUBN->pHead)
        pUBN->pTail = p;
    ++pUBN->digitCount;
}
void _Normalize (struct UBigNumber *pUBN)
{
    if (pUBN->digitCount == 0)
        _AppendDigit (pUBN, 0);
    while (pUBN->digitCount > 1 && pUBN->pHead->next->digit == 0)
    {//过滤0
        struct Node *p;
        p = pUBN->pHead->next;
        pUBN->pHead->next = p->next;
        p->next->prev = pUBN->pHead;
        free (p);
        --pUBN->digitCount;
    }
}

struct Node *_NewNode ()
{
    struct Node *p;
    p = (struct Node *) malloc (sizeof (struct Node));
    if (p == NULL)
    {
        printf ("Error : out of memory\n");
        exit (-1);
    }
    return p;
```

```
}

```



```
4 6 3 2 9 1 5 7 8 10
PS D:\csjjg\程序设计综合实践> cd "d:\csjjg\程序设计综合实践\" ; if ($?
) { gcc eighth.c -o eighth } ; if ($?) { .\eighth }
12345678909876543213333888999666
147655765659657669789687967867
12345678909876543213333888999666 + 147655765659657669789687967867 = 138
2223656647311991123576967533
12345678909876543213333888999666 - 147655765659657669789687967867 = 108
6912125327996651544201031799
PS D:\csjjg\程序设计综合实践>
```