Homework 2.1

作业是大数的加减乘除取余运算，直接用C++代码来执行操作。首先判断输入是字符还是数字，如果是字符，肯定是不能计算的，那就直接输出表达式，如果是数字，再进行下面的操作。

因为超大数肯定是不能用`int`,`long long`等类型来读入的，我的思路是用链表来存储超大数，利用头插法，逆序遍历字符串插入（即输入的数是123的话，链表的存储为3->2->1），这个过程体现在`ListNode *keepnumber(string a)`函数里。

为了主要的加减乘除取余的函数操作，我还写了`void Print(ListNode *L)`，`ListNode *Reverse(ListNode *l1)`，`int Getlen(ListNode *l1)`，`bool Compare(ListNode *l1, ListNode *l2)`几个辅助函数，顾名思义功能分别是把链表转成字符串输出，逆置链表，获取链表长度，比较两个链表中存储的数字大小。

在主要函数中，加法是比较常规的，即从低位开始相加，唯一需要注意的就是进位，代码中使用`carry`来保存进位。减法对比加法，额外需要注意的就是数字中的先导0需要去除。由于先导0在字符串中体现在链表后方，处理非常不便，这里采用逆置链表，去除先导0，再逆置的方法，个人认为效果不错。

乘法是按位进行的，即类比于我们小学学习的竖式计算，每次只计算一个个位数乘一个大数，获得两个结果后进行相加，需要注意的只有前面记得要补0，这个算法相比把第一个数加第二个数那么多遍节省了很多时间。除法是用减法实现的，不过每次都会翻倍，这样也减少了很多时间，特别是第一个数比第二个数大很多倍的情况下。取余直接套用了除法的代码，只在返回值上做了修改。

需要注意的是，我会自动比较两个输入的数大小，如果小数提前输入，不会对结果有任何影响。但是没有处理负数，所以输入负数会按照表达式而不是大数来处理。

代码如下，没有使用任何大数计算的库，可以直接copy运行：

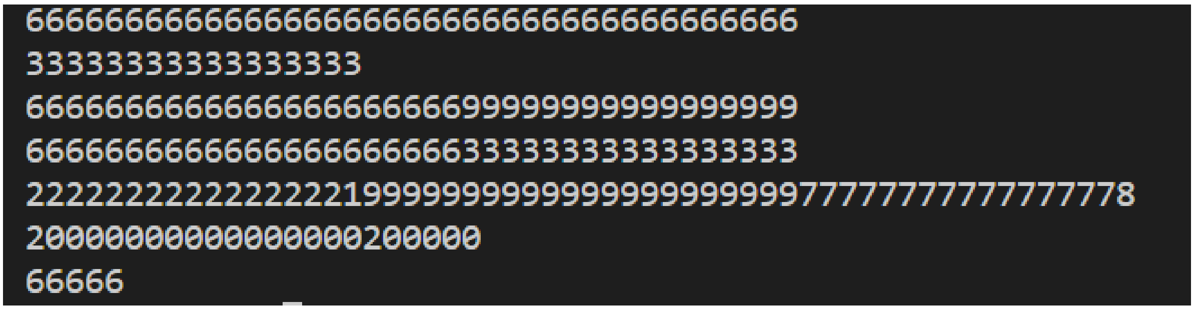
```
#include <bits/stdc++.h>
using namespace std;

typedef struct ListNode
{
    int val;
    struct ListNode *next;
} ListNode;

void Print(ListNode *L)
{
    vector<int> stack;
    while (L != nullptr)
    {
        // cout << L->val;
        stack.push_back(L->val);
        L = L->next;
    }
    for (int i = stack.size() - 1; i >= 0; i--)
    {
        cout << stack[i];
    }
    cout << endl;
    return;
}
```

```
ListNode *keepnumber(string a)
{
    int len = a.size();
    ListNode *L;
    ListNode *head = nullptr;
    ListNode *tail = nullptr;
    for (int i = 0; i < len; i++)
    {
        ListNode *L = new ListNode;
        L->next = nullptr;
        L->val = int(a[len - 1 - i] - '0');
        if (head == nullptr)
        {
            head = L;
            tail = L;
        }
        else
        {
            tail->next = L;
            tail = L;
        }
    }
    return head;
}

ListNode *Reverse(ListNode *l1)
{
    ListNode *newhead = NULL;
    ListNode *cur = l1;

    while (cur)
    {
        ListNode *next = cur->next;
        cur->next = newhead;
        newhead = cur;
        cur = next;
    }
    return newhead;
}

int Getlen(ListNode *l1)
{
    int cnt = 0;
    while (l1 != nullptr)
    {
        cnt++;
        l1 = l1->next;
    }
    return cnt;
}

bool Compare(ListNode *l1, ListNode *l2)
{
    if (Getlen(l1) > Getlen(l2))
```

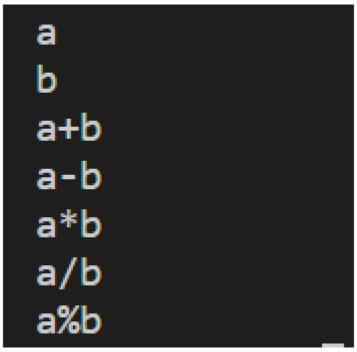
```
        return true;
    if (Getlen(l1) < Getlen(l2))
        return false;
    vector<int> keep1;
    while (l1 != nullptr)
    {
        keep1.push_back(l1->val);
        l1 = l1->next;
    }
    vector<int> keep2;
    while (l2 != nullptr)
    {
        keep2.push_back(l2->val);
        l2 = l2->next;
    }
    for (int i = keep2.size() - 1; i > -1; i--)
    {
        if (keep1[i] > keep2[i])
            return true;
        if (keep1[i] < keep2[i])
            return false;
    }
    return true;
}

ListNode *addTwoNumbers(ListNode *l1, ListNode *l2)
{
    ListNode *head = nullptr, *tail = nullptr;
    int carry = 0;
    while (l1 || l2)
    {
        int n1 = l1 ? l1->val : 0;
        int n2 = l2 ? l2->val : 0;
        int sum = n1 + n2 + carry;
        if (head == nullptr)
        {
            head = (ListNode *)malloc(sizeof(ListNode));
            head->val = sum % 10;
            head->next = nullptr;
            tail = head;
        }
        else
        {
            tail->next = (ListNode *)malloc(sizeof(ListNode));
            tail->next->val = sum % 10;
            tail->next->next = nullptr;
            tail = tail->next;
        }
        carry = sum / 10;
        if (l1)
        {
            l1 = l1->next;
        }
        if (l2)
        {
            l2 = l2->next;
        }
    }
    if (carry > 0)
    {
        tail->next = (ListNode *)malloc(sizeof(ListNode));
        tail->next->val = carry;
        tail->next->next = nullptr;
    }
    return head;
}

ListNode *subTwoNumbers(ListNode *l1, ListNode *l2)
{
    ListNode *head = nullptr, *tail = nullptr;
    int carry = 0;
    while (l1 || l2)
    {
        int n1 = l1 ? l1->val : 0;
        int n2 = l2 ? l2->val : 0;
        int sum = n1 - n2 - carry;
        if (head == nullptr)
        {
            head = (ListNode *)malloc(sizeof(ListNode));
            head->val = (sum + 10) % 10;
            head->next = nullptr;
            tail = head;
        }
        else
        {
            tail->next = (ListNode *)malloc(sizeof(ListNode));
            tail->next->val = (sum + 10) % 10;
            tail->next->next = nullptr;
            tail = tail->next;
        }
        carry = sum >= 0 ? 0 : 1;
        if (l1)
        {
            l1 = l1->next;
        }
        if (l2)
        {
            l2 = l2->next;
        }
    }
    if (carry > 0)
    {
        tail->next = (ListNode *)malloc(sizeof(ListNode));
        tail->next->val = carry;
        tail->next->next = nullptr;
    }
    ListNode *keep = Reverse(head);                 // 这一串代码是专门清楚相减后的先导0
    while (keep->val == 0 && keep->next != nullptr) // 如果只剩1个0，保留0
    {
        keep = keep->next;
    }
    head = Reverse(keep);
    return head;
}

ListNode *mulTwoNumbers(ListNode *l1, ListNode *l2)
{
    ListNode *sum1 = new ListNode;
    sum1->val = 0;
    sum1->next = nullptr;
    ListNode *sum2 = new ListNode;
    sum2->val = 0;
    sum2->next = nullptr;
    while (l2 != nullptr)
    {
        for (int i = 0; i < l2->val; i++)
        {
            sum1 = addTwoNumbers(l1, sum1);
        }
        sum2 = addTwoNumbers(sum1, sum2);
        ListNode *head = new ListNode;
        head->val = 0;
        head->next = l1;
        l1 = head;
        l2 = l2->next;
        sum1->val = 0;
        sum1->next = nullptr;
    }
    return sum2;
}

ListNode *divTwoNumbers(ListNode *l1, ListNode *l2)
{
    ListNode *keep = new ListNode;
    keep->val = 1;
    keep->next = nullptr;
    ListNode *sum = new ListNode;
    sum->val = 0;
    sum->next = nullptr;
    ListNode *l0 = l2;
    while (Compare(l1, l2))
    {
        while (Compare(l1, addTwoNumbers(l0, l0)))
        {
            keep = addTwoNumbers(keep, keep);
            l0 = addTwoNumbers(l0, l0);
        }
        l1 = subTwoNumbers(l1, l0);
        l0 = l2;
        sum = addTwoNumbers(keep, sum);
        keep->val = 1;
        keep->next = nullptr;
    }
    return sum;
```

```
}

ListNode *perTwoNumbers(ListNode *l1, ListNode *l2)
{
    ListNode *keep = new ListNode;
    keep->val = 1;
    keep->next = nullptr;
    ListNode *sum = new ListNode;
    sum->val = 0;
    sum->next = nullptr;
    ListNode *l0 = l2;
    while (Compare(l1, l2))
    {
        while (Compare(l1, addTwoNumbers(l0, l0)))
        {
            keep = addTwoNumbers(keep, keep);
            l0 = addTwoNumbers(l0, l0);
        }
        l1 = subTwoNumbers(l1, l0);
        l0 = l2;
        keep->val = 1;
        keep->next = nullptr;
    }
    return l1;
}

int main()
{
    string a, b;
    cin >> a >> b;
    for (int i = 0; i < a.size(); i++)
    {
        if (!isdigit(a[i]))
        {
            cout << a << "+" << b << endl;
            cout << a << "-" << b << endl;
            cout << a << "*" << b << endl;
            cout << a << "/" << b << endl;
            cout << a << "%" << b << endl;
            return 0;
        }
    }
    for (int i = 0; i < b.size(); i++)
    {
        if (!isdigit(b[i]))
        {
            cout << a << "+" << b << endl;
            cout << a << "-" << b << endl;
            cout << a << "*" << b << endl;
            cout << a << "/" << b << endl;
            cout << a << "%" << b << endl;
            return 0;
        }
    }
    if (a.size() < b.size() || (a.size() == b.size() && a < b))
    {
        string temp = a;
        a = b;
        b = temp;
    }
    ListNode *L1 = keepnumber(a);
    ListNode *L2 = keepnumber(b);
    ListNode *L3 = addTwoNumbers(L1, L2);
    ListNode *L4 = subTwoNumbers(L1, L2);
    ListNode *L5 = mulTwoNumbers(L1, L2);
    ListNode *L6 = divTwoNumbers(L1, L2);
    ListNode *L7 = perTwoNumbers(L1, L2);
    Print(L3);
    Print(L4);
    Print(L5);
    Print(L6);
    Print(L7);
    return 0;
}
```

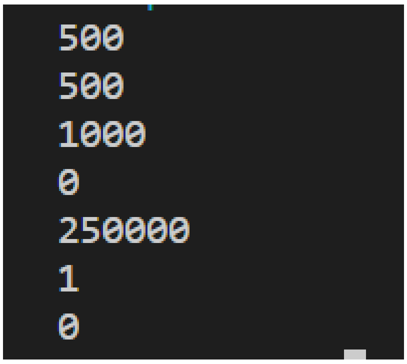
以下是测试结果，可见代码的正确性~



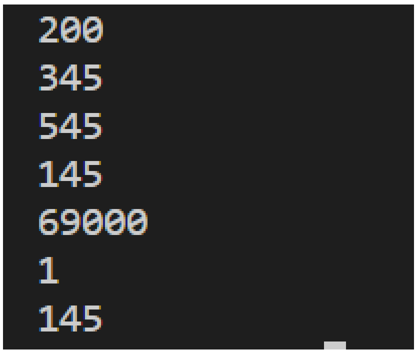
```
666666666666666666666666666666666666666
33333333333333333
666666666666666666666699999999999999999
666666666666666666666633333333333333333
22222222222222221999999999999999999999977777777777777778
20000000000000000200000
66666
```



```
a
b
a+b
a-b
a*b
a/b
a%b
```



```
500
500
1000
0
250000
1
0
```



```
200
345
545
145
69000
1
145
```

Homework 2.2

作业是实现DH算法，就是用离散对数来交换密钥，最后两方都获得一个协商密钥。

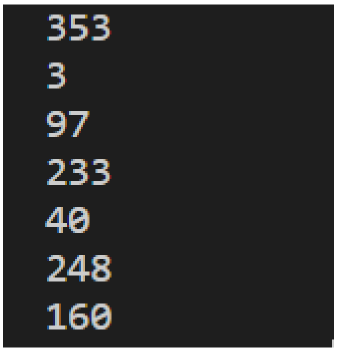
我先贴上代码再进行解释。

```
#include <bits/stdc++.h>
#include "IS2.cpp"
using namespace std;
bool Judge0(ListNode *l0)
{
    if (l0->val == 0 && l0->next == nullptr)
        return true;
    return false;
}

int main()
{
    string p, g, a, b;
    cin >> p >> g >> a >> b;
    ListNode *pl = keepnumber(p);
    ListNode *gl = keepnumber(g);
    ListNode *al = keepnumber(a);
    ListNode *bl = keepnumber(b);
    ListNode *ahelp = al;
    ListNode *help = new ListNode;
    help->val = 1;
    help->next = nullptr;
    ListNode *Al = help;
    ListNode *Bl = help;
    while (!Judge0(ahelp))
    {
        Al = mulTwoNumbers(Al, gl);
        ahelp = subTwoNumbers(ahelp, help);
    }
    Al = perTwoNumbers(Al, pl);
    Print(Al);
    while (!Judge0(bl))
    {
        Bl = mulTwoNumbers(Bl, gl);
        bl = subTwoNumbers(bl, help);
    }
    Bl = perTwoNumbers(Bl, pl);
    Print(Bl);
    ListNode *key = help;
    while (!Judge0(al))
    {
        key = mulTwoNumbers(key, Bl);
        al = subTwoNumbers(al, help);
    }
    key = perTwoNumbers(key, pl);
    Print(key);
    return 0;
}
```

在代码中，`IS2.cpp`这个文件就包含了Homework 2.1代码中除了main以外的所有函数，其中这个文件中用到的主要就是存数的函数，减法，乘法，取余的运算和输出函数。

在输入中，pl，gl即为共有信息p，g，al即为Alice随机选择的私钥a，并利用$A=(g^a)modp$计算出公钥Al，bl即为Bob随机选择的私钥b，并利用$B=(g^b)modp$计算出公钥Bl，两人交换公钥。Alice利用Bob传来的Bl，通过$key=(B^a)modp$计算共享密钥key，Bob利用Alice传来的Al，通过$key=(A^b)modp$计算共享密钥key。而如果第三方从中拦截，他们只能拦截A和B，而通过A，B来计算a，b是很难做到的，没有a，b就算不出密钥key。这样就成功传递了信息。

测试结果如下，前四排是输入，后三排是输出。跑得还稍微有点慢，大概是好几秒计算出一个结果，主要原因除了计算太复杂可能是计算乘方的时候没有优化。不过我的乘法也是计算一位数字的最快，所以优化的话性能提升不大。不过可以表明计算结果是对的。



```
353
3
97
233
40
248
160
```