# 期中复习之高精度—-乘法篇

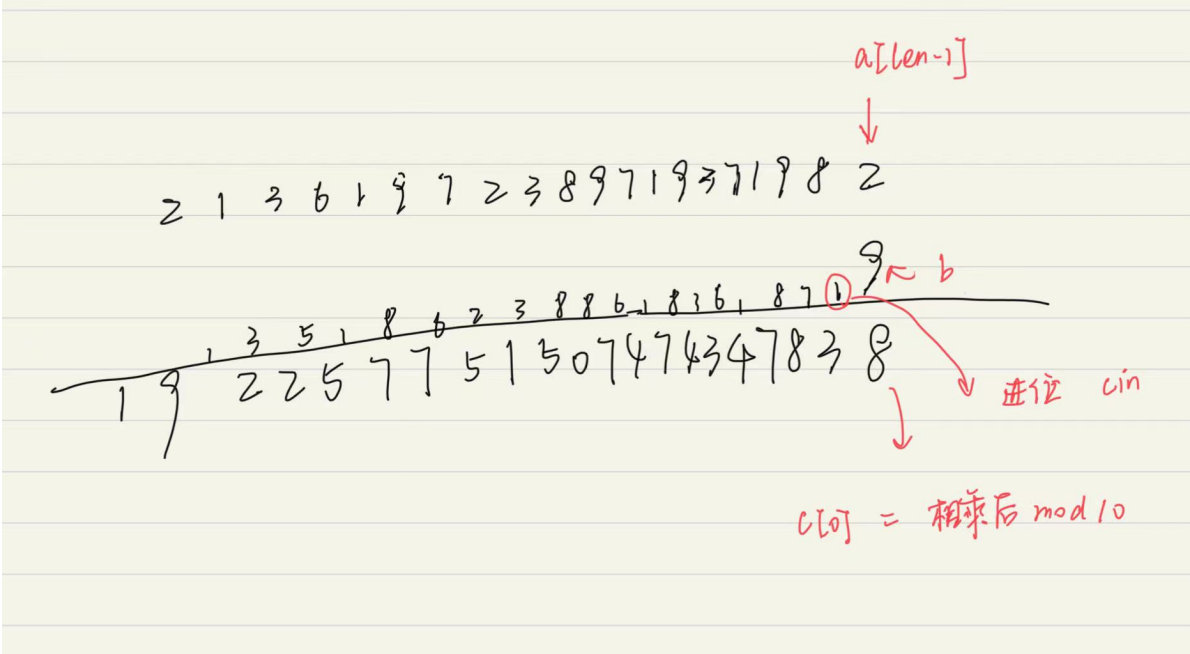
为什么只有乘法呢？因为我debug de了我好长时间（救命）

高精度乘法(A * B problem)

## Part1: 高精度的数据 乘 一位的数

例如： a = 21361972389719371982, b=9 这样的乘法

(和高精度加法的处理方法类似)




代码如下：

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

char * mul_O(char* a, char b);

int main()
{
    char a[20000];
    char b;
    char* c = (char*)malloc(sizeof(char)*12);

    scanf("%s", a);//以字符串的形式读入 a
    getchar();//去除掉缓冲区的'\n'
    scanf("%c",&b);//以字符的形式读入 b

    c = mul_O(a,b);// 计算乘法的函数

    printf("%s",c);
```

```
    return 0;
}

char * mul_0(char* a, char b)
{
    int i,j=0;
    int cin=0;
    int len;
    char* c = (char*)malloc(sizeof(char)*200000);
    char* d = (char*)malloc(sizeof(char)*200000);
    //创建了两个超长的字符串数组来存结果。
    len = strlen(a);

    for(i=len-1;i>=0;i--){
        c[j] = (cin + (a[i]-'0')*(b-'0')) % 10 + '0';
        //乘法
        cin = (cin + (a[i]-'0')*(b-'0')) /10;
        //处理进位
        j++;
    }

    if(cin!=0){
        //判断最高位是否进位(是不是和加法的代码很像)
        c[j]=cin+'0';
        for(i=0;i<=j;i++)
            d[i]=c[j-i];
        //算完之后c[0]是最低位，要把他倒过来输出
        d[i]='\0';
    }
    else{
        for(i=0;i<j;i++)
            d[i]=c[j-1-i];
        d[i]='\0';
    }
    return d;//结果存在d的字符串数组中
}
```

所以答案就是：192257751507474347838~

**注意适用范围：正整数乘法（高精度 * 1位数）并且不包含前导零且那个1位数不是0；**

# Part 2：高精度 * 高精度

就是part1 的plus版本，但思路差不多

# A*B Problem

## 题目描述

给出两个非负整数，求它们的乘积。

## 输入格式

输入共两行，每行一个非负整数。

## 输出格式

输出一个非负整数表示乘积。

## 样例 #1

### 样例输入 #1

```
1
2
```

### 样例输出 #1

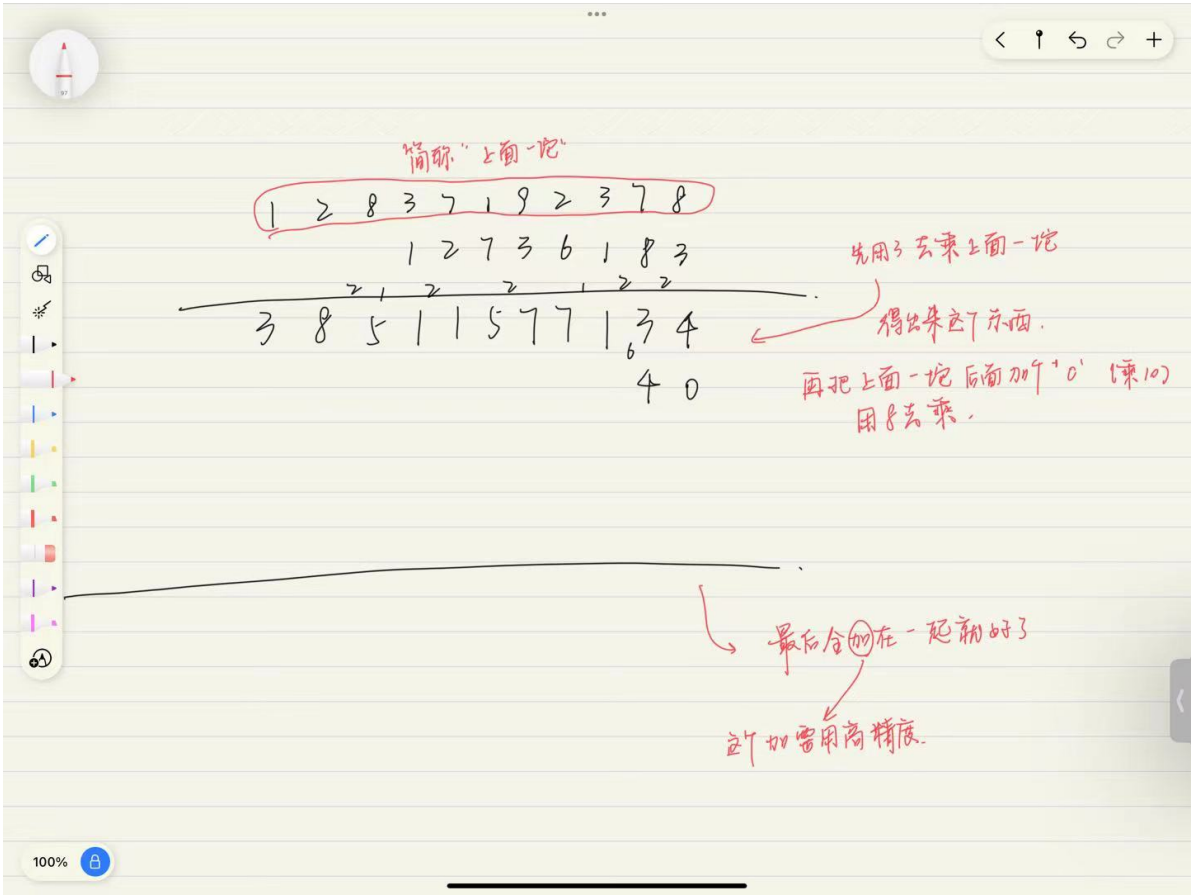
```
2
```

## 提示

每个非负整数不超过 $10^{2000}$。

PS：让大家体会一下用函数(结构化编程)的好处~。

比如要算： a = 12837192378，b = 12736183




代码：

```
#include <stdio.h>
```

```c
#include <stdlib.h>
#include <string.h>

char* mul(char* a, char* b);
// 高精度 * 高精度
char* add(char* a, char* b);
// 高精度加法
char * mul_0(char* a, char b);
// 高精度 * 1位数

int main()
{
    char a[5000];
    char b[5000];
    char* c =(char*)malloc(sizeof(char) * 5000);

    scanf("%s",a);
    scanf("%s",b);

    c = mul(a,b);

    printf("%s", c);

    return 0;
}

char* mul(char*a, char*b)
{
    int i;
    int len1 = strlen(a);
    char* c = (char*)malloc(sizeof(char) * 5000);
    char* d = (char*)malloc(sizeof(char) * 5000);

    if((strcmp(a,"0")==0)||(strcmp(b,"0")==0)){
        c[0] = '0';
        c[1] = '\0';
        return c;
    }
    //特判a和b中有一个为0的情况

    for(i=len1-1;i>=0;i--){
        if(i==len1-1){
            d = mul_0(b,a[i]);
        }
        else{
        c = mul_0(b,a[i]);
        d = add(c,d);
        }
        strcat(b,"0");
    }

    return d;
}

char * mul_0(char* a, char b)
```

```
    //高精度 * 一位 乘法
{
    int i,j=0;
    int cin=0;
    int len;
    char* c = (char*)malloc(sizeof(char)*5000);
    char* d = (char*)malloc(sizeof(char)*5000);
    len = strlen(a);

    for(i=len-1;i>=0;i--){
        c[j] = (cin + (a[i]-'0')*(b-'0')) % 10 + '0';
        cin = (cin + (a[i]-'0')*(b-'0')) /10;
        j++;
    }

    if(cin!=0){
        c[j]=cin+'0';
        for(i=0;i<=j;i++)
            d[i]=c[j-i];
        d[i]='\0';
    }
    else{
        for(i=0;i<j;i++)
            d[i]=c[j-1-i];
        d[i]='\0';
    }
    return d;
}

char* add(char*a, char*b)
    //高精度加法
{
    int i,j,k=0;
    int cin=0;
    char* c=(char*)malloc(sizeof(char)*5000);
    char* d=(char*)malloc(sizeof(char)*5000);
    i = strlen(a)-1;
    j = strlen(b)-1;

    while(i>=0||j>=0){
        a[i]=(i<0)?'0':a[i];
        b[j]=(j<0)?'0':b[j];
        c[k]=(a[i]-'0'+b[j]-'0'+cin)%10 + '0';
        cin =(a[i]-'0'+b[j]-'0'+cin)/10;
        if(i>=0)
            i--;
        if(j>=0)
            j--;
        k++;
    }

    if(cin==1){
        c[k]='1';
        for(i=0;i<=k;i++)
            d[i]=c[k-i];
```

```
        d[i]='\0';
    }
    else{
        for(i=0;i<=k-1;i++)
            d[i]=c[k-1-i];
        d[i]='\0';
    }
    return d;
}
```

结果是这个东西：163496831332413174

**注意适用范围：正整数乘法（高精度 * 高精度），且没有前导0**