



C++编程



目录

- 函数
- 递归
- 不定长输入
- 无穷大
- 排序和去重
- 埃氏筛
- 线性筛
- 二分查找
- 在线和离线
- 二维数组
- 字符数组
- 字符串
- 高精度运算
- 结构体
- 文件操作
- 快速读入
- 位运算

挑战

- 给定正整数 n ($n \leq 10^{18}$)，判定该数是几位数

Sample input	Sample output
2147483647	10

秒了它

```
#include<bits/stdc++.h>
using namespace std;
long long n;
int ans;
int main()
{
    scanf("%lld", &n);
    while (n > 0)
        n /= 10, ans ++;
    printf("%d", ans);
    return 0;
}
```

继续挑战

- 给定正整数 n ($n \leq 10^{10000}$)，判定该数是几位数

Sample input	Sample output
21474836479223372036854775807	29

分析

- 10^{10000} 这么大的数，没有任何数据类型可以支持

```
? n;  
scanf("% ? ", &n);
```

- 这时候就必须使用“字符”数据类型



一个一万位的数
文件
9.76 KB

997424673654617379941856381417666661371930976934499594343421467623372724694551
303939121801967855661260711079156970330636560803003267688053153104899858862153
548794630679137531996259106401177816759557750466679880791503835569135381832926
607920374515317910144591552312251582919280069641047256463926102044875510648560
686755501899162751931347354507050559082792164685161822392296245097381599524415
881436732680193206170111253337930606727496662424445369751218795753203438617656
729803383745219500758242867804217666943422626050741465481063610944331943543724
487931158427015879770379963667992932799224455209044794408457082773685130921520
113807851491439082109481245360224745266663635146590584631386567845996932499684
084965424077529414262840277447624092366477424197105529366531755613361853115408
990124559891355929740716234424536708783610130163971555926080024682214332284143
098372363006375436806002327346843136971491661101010560944216096653147567935408
225356870552314335226156817538343982866784218798192844788639608521436021936553
749119503432633579384544037533649923492236815870855146639377236679615364696284
285085565232983377131036826584802509981935320010386969387806769203967168072559
540625525112932618271577069004634546615037629222499047429179323052005636451602
997124794765312348259870040184116842005683546607120242191507607504884344249426
319196724757224367076297833978302964079816779592937943897576958575919298318096
882764213830106990430671424926214514208630324678925850381037286395463764849764
779707981893695056851547066531079623549543389741061260182341064571770790067290
809961806835327539092060356967760658423364725117182063702286174559928665094447
791451566502905970437358020484791384307101840512650651485247146133976970373085
493356103311967227727205871187259472695979413405756525903698614905224784100738
003307900074098793666862850960746549540298080596121530124706659180491880962520
385244983202910676008527109275265096590991535808910719974159492507964200153304
354107376602781903747479004911340591423706195106841709996108376081910971950732

字符

- 不能进行算术运算的字母、符号、数字等，都属于字符范畴
- 比如: `'a', 'b', 'c', '*', '#', '1', '2';`

```
char n;  
scanf("%c", &n);
```

- 但是单个的字符解决不了我们的问题，我们需要10,000个字符

字符数组

- 于是我们定义一个长度10,000的字符数组，它的每个数组元素都是一个字符

```
char n[10010];  
scanf("%s", n);
```

特别注意：

1. 字符数组的占位符是：%s，单个字符则是：%c
2. 读入字符数组不需要读地址符 &

字符数组

- 还有一个重要的问题没有解决： 我们不知道读入的字符有多少个！

\0

- 标题的这个“\0”，是系统悄悄给我们加上的（你甚至无法输出查看它），用于标记（字符数组中的）字符输入完毕
- \0后面又是什么呢？如果我们开全局数组，系统默认初始化为“空字符”（我们知道 `int` 数组默认初始化为数字0）

n[0]	n[1]	n[2]	n[3]	n[4]	n[5]	n[6]	n[7]	n[8]	n[9]	n[10]	n[11]	n[12]
'2'	'1'	'4'	'7'	'4'	'8'	'3'	'6'	'4'	'7'	'\0'	"	"

我们不妨做个小实验

- 实验的内容很简单：输入k个字符数组，然后全范围输出
- 空格的地方是什么？正是 \0

```
3
abcdefghi
abcdefghi
opqrst
opqrst hi
uvw
uvw st hi
```

```
#include<bits/stdc++.h>
using namespace std;
char a[10];
int main()
{
    int k; scanf("%d", &k);
    while (k --)
    {
        scanf("%s", a);
        for (int i = 0; i < 10; i ++)
            printf("%c", a[i]);
        printf("\n");
    }
    return 0;
}
```

参考代码

- 验证了\0的存在，那我们可以这样写：

```
#include<bits/stdc++.h>
using namespace std;
char n[10010];
int i;
int main()
{
    scanf("%s", n);
    while (n[i] != '\0') i ++;
    printf("%d", i);
    return 0;
}
```

```
21474836479223372036854775807
29
-----
Process exited after 33.4 seconds with
return value 0
请按任意键继续. . .
```

strlen 函数

- 系统提供了一个函数strlen，用于返回字符数组的长度（不包括\0）
- 它的函数返回值是 int 型

```
#include<bits/stdc++.h>
using namespace std;
char n[10010];
int main()
{
    scanf("%s", n);
    printf("%d", strlen(n));
    return 0;
}
```

```
21474836479223372036854775807
29
-----
Process exited after 33.4 seconds with return value 0
请按任意键继续. . .
```

strlen 函数

- strlen 函数相当于这样

```
#include<bits/stdc++.h>
using namespace std;
char n[10010];
int main()
{
    scanf("%s", n);
    printf("%d", strlen(n));
    return 0;
}
```

这种把整个数组作为形式参数传入函数的做法，非不得已不推荐使用



```
#include<bits/stdc++.h>
using namespace std;
char n[10010];
int mystrlen(char x[])
{
    int i = 0;
    while (x[i] != '\0') i ++;
    return i;
}

int main()
{
    scanf("%s", n);
    printf("%d", mystrlen(n));
    return 0;
}
```

字符

- 计算机科学中，字符的一大意义在于突破了数据范围的限制，因此大整数、高精度等都需要把数字转成字符处理，然后再转回数字

课堂练习

Tips: 字符
可以直接取模
运算



- 输入任意 n ($n \leq 100$) 个整数 a_i ($a_i \leq 10^{60}$), 分别判断其奇偶性

Sample input	Sample output
2 //n	even
1024 //a _i , 下同	odd
5931	

参考代码

为什么这里完全不用考虑先后输入的不同数字，位数不一的问题？



```
#include<bits/stdc++.h>
using namespace std;
char a[70];
int n;
int main()
{
    scanf("%d", &n);
    while (n --)
    {
        scanf("%s", a);
        int len = strlen(a);
        if (a[len-1] % 2 == 0) printf("even\n");
        else printf("odd\n");
    }
    return 0;
}
```

解析

- 所以，为什么字符可以取模运算？
(不是说好的字符不能参与四则运算吗)

```
48 49 50 51 52 53 54 55 56 57
-----
Process exited after 0.1063 seconds w
ith return value 0
请按任意键继续. . .
```

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    char a[10] = {'0','1','2','3','4','5','6','7','8','9'};
    for (int i = 0; i < 10; i++)
        printf("%d ", a[i]);
    return 0;
}
```

ASCII码

```
48 49 50 51 52 53 54 55 56 57
-----
Process exited after 0.1063 seconds w
ith return value 0
请按任意键继续. . .
```

- 美国信息交换标准代码。所有的（英文）字符（包括数字和标点符号，以及回车、换行、空格等等）在计算机中都是以一个字节长度的二进制形式存储的，共**128**种
- **00110000**（十进制数就是**48**）代表字符：‘0’，其他依次类推
- 所以刚才的程序，仅仅是字符‘0’~‘9’的**ASCII**码（十进制）刚好和其数字本身的奇偶性保持一致，而并非是字符（数字）本身参与取模运算的结果（但这应该也是**ASCII**码设计者的初衷之一）

ASCII码

- ‘a’~‘z’对应ASCII码（十进制）：97~122

```
97 98 99 100 101 102 103 104 105 106 107 108 109
110 111 112 113 114 115 116 117 118 119 120 121
122
```

```
-----
Process exited after 0.1568 seconds with return
value 0
请按任意键继续. . .
```

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    for (int i = 'a'; i <= 'z'; i++)
        printf("%d ", i);
    return 0;
}
```

ASCII码

- ‘A’~‘Z’对应ASCII码（十进制）： 65~90

```
65 66 67 68 69 70 71 72 73 74 75 76 77
78 79 80 81 82 83 84 85 86 87 88 89 90
```

```
-----
Process exited after 0.1792 seconds with
return value 0
请按任意键继续. . .
```

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    for (int i = 'A'; i <= 'Z'; i++)
        printf("%d ", i);
    return 0;
}
```

字符

- 但这并不违背“字符不能参与四则运算”
- 目测下面程序的运行结果：

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    char a = '1', b = '2';
    printf("%d", a + b);
    return 0;
}
```

99

Process exited after 0.0129 seconds w
ith return value 0
请按任意键继续. . .

挑战

- 一个英文单词（最长189819个）夹杂了大小写，要求把它们转换为全小写

Sample input	Sample output
PneumonoultramicroScopicsilicovolcANoconiosis	pneumonoultramicroscopicsilicovolcanoconiosis

参考代码

小写字母的
ASCII码比对应的
大写字母大32



```
#include<bits/stdc++.h>
using namespace std;
char word[200000];
int main()
{
    scanf("%s", word);
    int len = strlen(word);
    for (int i = 0; i < len; i ++)
        if (word[i] >= 'A' && word[i] <= 'Z') printf("%c", word[i] + 32);
        else printf("%c", word[i]);
    return 0;
}
```


继续挑战

- 一段英文（包含空格、标点符号等），要求把它们转换为全小写

Sample input	Sample output
To be or Not to be, Is a Question!	to be or not to be, is a question!

应战

```
To be or not to be
to
-----
Process exited after 10.47 seconds with return value 0
请按任意键继续. . .
```

- 结果我们发现：尴尬地只输入了一个单词
- 原来 `scanf` 输入字符时有个局限：空格是多个数据之间的间隔符！
- `cin` 在这点上和 `scanf` 没有区别，也不行
- 输入字符必须有新手段

gets 读行函数

- gets()用于完整读入一行（遇到换行符停下）
- 换行符：\n，ASCII码（十进制）10

```
#include<bits/stdc++.h>
using namespace std;
char word[200000];
int main()
{
    gets(word);
    int len = strlen(word);
    for (int i = 0; i < len; i++)
        if (word[i] >= 'A' && word[i] <= 'Z') printf("%c", word[i] + 32);
        else printf("%c", word[i]);
    return 0;
}
```

```
To be or Not to be, Is a Question!
to be or not to be, is a question!
-----
Process exited after 2.474 seconds wi
th return value 0
请按任意键继续. . .
```

gets 读行函数

```
William Shakespeare (1564-1616), the foremost writer, prominent dramatist and poet in the European Renaissance. He created a large number of popular literary works, occupies a special position in the history of European literature, has been hailed as "Olympus Zeus in human literature." He is also known as the four great tragedies of ancient Greece, Aeschylus, Sophocles and Euripides. _
```

- 注意我的表述是： `gets()`遇到换行符才会停下
- 因此我们看到的这段英文，是一行！用`gets()`可以全吃下！

- 计算机科学是严谨的



tolower 函数/toupper 函数

- 直接用于把大写字母转成小写（tolower函数）
- 小写字母转成大写（toupper函数）
- 但是注意它们的自变量是单个字符，而不是整个字符数组

```
To be or Not to be, Is a Question!  
to be or not to be, is a question!  
-----  
Process exited after 2.999 seconds wi  
th return value 0  
请按任意键继续. . .
```

```
#include<bits/stdc++.h>  
using namespace std;  
char word[200000];  
int main()  
{  
    gets(word);  
    int len = strlen(word);  
    for (int i = 0; i < len; i ++)  
        printf("%c", tolower(word[i]));  
    return 0;  
}
```

tolower 函数/toupper 函数

toupper函数用法一样，就不再赘述啦



- tolower 函数相当于这样

```
#include<bits/stdc++.h>
using namespace std;
char word[200000];
int main()
{
    gets(word);
    int len = strlen(word);
    for (int i = 0; i < len; i++)
        printf("%c", tolower(word[i]));
    return 0;
}
```

```
#include<bits/stdc++.h>
using namespace std;
char word[200000];
char mytolower(char x)
{
    if (x >= 'A' && x <= 'Z') return x + 32;
    return x;
}

int main()
{
    gets(word);
    int len = strlen(word);
    for (int i = 0; i < len; i++)
        printf("%c", mytolower(word[i]));
    return 0;
}
```

课堂练习

- 给定正整数 n ($n \leq 10^{10000}$)，计算 n 各位上的数字之和

Sample input	Sample output
108	9

回文串



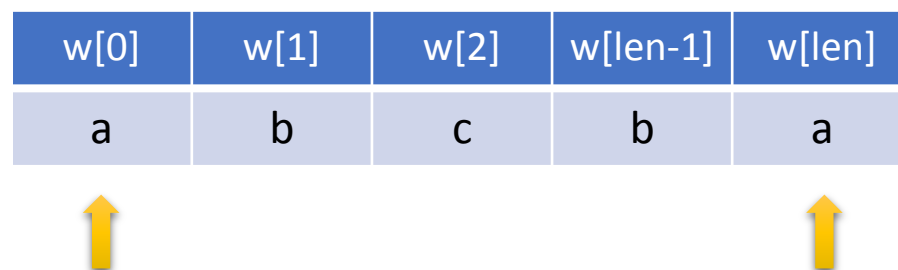
- 形如 “abba”、“abcba” 这样的字串，被称为回文串，特别注意 “a” 也算回文串
- 你的程序要能识别某给定字串（至多10000个小写字母组成），是否为回文串（输出“yes”或“no”）

Sample input	Sample output
abcba	yes

分析

- 对于abcba这样的字符串：
- 只需要从两头向中间扫一遍，逐个判等即可
- 正中间的位置是无需判断的

w[0]	w[1]	w[2]	w[len-1]	w[len]
a	b	c	b	a



继续挑战一下

- 形如“abba”、“abcba”这样的字串，被称为回文串，特别注意“a”也算回文串
- 你的程序要能识别某给定k（ $1 \leq k \leq 100$ ）条字串（每条字串由至多10000个小写字母组成），是否为回文串（输出“yes”或“no”）

Sample input	Sample output
3 //k	yes
abcba	no
bbc	no
nba	

最小周期串

- 如果一个字符串 s 能以另一个字符串 s' 重复多次得到，就称字符串 s' 为 s 的周期串，周期串 s' 中长度最短的即是最小周期串，比如 `abccabc` 的最小周期串为 `abc`。
注意 `aaaa` 的最小周期串是 `a` 而不是 `aa`，`abc` 的最小周期串为 `abc`
- 你的程序要能输出给定的 k ($1 \leq k \leq 100$) 条字符串的最小周期串

Sample input	Sample output
3 //k abccabc aaaa nba	abc a nba

分析

- 因为周期串的长度未知，所以直接寻找很费事
- 但无论周期串的长度如何，肯定是从[0]号字符开始的，而不可能从中间位置开始
- 那么我们只要确定了长度，就确定了最小周期串

Sample input	Sample output
3 //k abcabc aaaa nba	abc a nba

分析

- 最小周期串的长度？范围 $\in [1, \text{len}]$ ，我们无法预知，所以只能逐个尝试，且1比2优，2比3优.....
- 那我们的思路就是先确定周期串的长度，然后得到的第一个周期串就必然是最小周期串（因为是从小往大循环）

分析

- 而且，周期串的长度有什么特点？
- 只可能是整个字符串长度的因子数

```
for (i:1~len)
{
    if (len % i == 0)
    {
        判断长度 i 是否是周期串的长度
    }
}
```

分析

- 我们模拟一遍匹配的过程

word[0]	word[1]	word[2]	word[3]	word[len]	word[len]
a	b	c	a	b	c

↑ ↑ ↑ ↑

i: 1, 匹配失败, break

i: 2, 匹配失败, break

i: 3, 匹配暂时成功, 继续

分析

- 我们模拟一遍匹配的过程

word[0]	word[1]	word[2]	word[3]	word[len]	word[len]
a	b	c	a	b	c

↑ ↑

- 所以我們还需要一个 j 来扫一遍

i: 1, 匹配失败, break

i: 2, 匹配失败, break

i: 3, 匹配暂时成功, 继续

i: 3, 匹配暂时成功, 继续

i: 3, 匹配完全成功, 输出

分析

```
for (j:i~len)
    if (word[j] != word[j%i]) 匹配失败
```

word[0]	word[1]	word[2]	word[3]	word[4]	word[5]
a	b	c	a	b	c



i: 1, 匹配失败, break

i: 2, 匹配失败, break

i: 3, 匹配暂时成功, 继续

i: 3, 匹配暂时成功, 继续

i: 3, 匹配完全成功, 输出

分析

- 程序框架

```
for (i: 1~len)
{
    if (len % i == 0) //判断周期串长度 i是否可行
    {
        标记true;
        for (j: i~len)
        {
            if (word[j]) != word[j%i]
                标记false;
                匹配失败, 退出;
        }
        检查标记(如果保持true不变, 证明一直匹配成功直到字符串结束)
        {
            长度为 i的字符串即为最小周期串;
            从0号位开始输出;
            退出, 不再寻找更长的周期串;
        }
    }
}
```

字符串

- 何为字符串？即一串连续的字符
- 比如：

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    string s = "Hello C++!";
    cout << s;
    return 0;
}
```

- 那这不就是字符数组嘛！

```
Hello C++!
-----
Process exited after 0.1336 seconds w
ith return value 0
请按任意键继续. . .
```

字符串

- 字符串的用途的确和字符数组几乎没区别
- 实际上，字符串也是用字符数组来实现的，不过封装好了类似于函数可以直接调用
- 但在用法上还是有区别的

字符串

- 字符串是一种数据类型，而不是数组，所以它没有下标
- 那能支持多少个字符？弹性长度

特别注意：

1. 字符串不需要定义长度
2. 字符串只能用cin/cout，不支持scanf/printf

字符串的输入输出

```
3
abcdefg
abcdefg
Hello
Hello
NBA
NBA
```

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int k; scanf("%d", &k);
    string s;
    while (k --)
    {
        cin >> s;
        cout << s << endl;
    }
    return 0;
}
```

字符串的长度

- 字符数组获取长度是 `strlen(x)`
- 字符串获取长度则是 `x.size()` 或者 `x.length()`

15

```
-----  
Process exited after 0.2822 seconds w  
ith return value 0  
请按任意键继续. . .
```

```
#include<bits/stdc++.h>  
using namespace std;  
int main()  
{  
    string s = "Happy new year!";  
    cout << s.size() << endl;  
    return 0;  
}
```

getline 函数

- cin 读入字符串也会遇到空格停下
- 字符数组解决这个问题是用gets(x)函数读入整行
- 字符串则是用**getline(cin, x)**函数读入整行

```
Happy new year!  
15  
  
-----  
Process exited after 10.59 seconds wi  
th return value 0  
请按任意键继续. . .
```

```
#include<bits/stdc++.h>  
using namespace std;  
int main()  
{  
    string s;  
    getline(cin, s);  
    cout << s.size() << endl;  
    return 0;  
}
```


标题统计

- 凯凯刚写了一篇美妙的作文，请问这篇作文的标题中有多少个字符？注意：标题中可能包含大、小写英文字母、数字字符、空格和换行符。统计标题字符数时，空格和换行符不计算在内
- 输入文件只有一行

Sample input	Sample output
Ca 45	4

秒掉

- 题目都说了输入只有一行，所以只有空格不计数

```
#include<bits/stdc++.h>
using namespace std;
int ans;
string s;
int main()
{
    getline(cin, s);
    for (int i = 0; i < s.size(); i++)
        if (s[i] != ' ') ans++;
    cout << ans;
    return 0;
}
```

字符串的用法

- 字符串虽然是一个数据类型，但它也是用字符数组实现的
- 所以也可以对字符串中的单个字符操作

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    string s;
    getline(cin, s);
    for (int i = 0; i < s.size(); i ++)
        cout << s[i] << endl;
    return 0;
}
```

- 从这点上看，字符数组能做到的，字符串基本也能做到

字符串的用法

- 字符串联结运算“+”

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    string s, s2;
    cin >> s;
    while (cin >> s2)
        s += s2; // 字符串联结运算
    cout << s;
    return 0;
}
```

```
2018
ha pp y new y ear !
^Z
2018happynewyear!
-----
Process exited after 13.71 seconds wi
th return value 0
请按任意键继续. . .
```

消除多余空格

- 某英文文档（由大小写英文及标点符号构成，文档长度未知）因为排版疏忽，导致部分单词间出现了多余的空格，你的程序要能消除这些多余的空格

Sample input	Sample output
This is a great program	This is a great program

参考代码

- 我们利用字符串联结运算来消除多余的空格

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    string s, s2;
    cin >> s;
    while (cin >> s2)
        s += ' ' + s2;
    cout << s;
    return 0;
}
```

更多黑科技：reverse 函数

- reverse函数用于字符串翻转

```
CCF
FCC
-----
Process exited after 6.26 seconds with
return value 0
请按任意键继续. . .
```

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    string s;
    cin >> s;
    reverse(s.begin(), s.end());
    cout << s;
    return 0;
}
```

更多黑科技：reverse 函数

- 还可以只翻转字符串的一部分
- 表示从1号位开始翻转

```
CCF
CFC
-----
Process exited after 2.61 seconds with
return value 0
请按任意键继续. . .
```

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    string s;
    cin >> s;
    reverse(s.begin() + 1, s.end());
    cout << s;
    return 0;
}
```


判断回文串

- 给定 k 条字符串，分别判断其是否为回文串

Sample input	Sample output
3 //k	no
bbc	yes
noipion	no
Aa	

字符串的用法

- 从这个例子看出:
 1. 字符串可以直接赋值
 2. 字符串可以直接判相等
- 体现出比字符数组更好的整体性

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int k; scanf("%d", &k);
    string s, s2;
    while (k --)
    {
        cin >> s;
        s2 = s;
        reverse(s.begin(), s.end());
        if (s == s2) printf("yes\n");
        else printf("no\n");
    }
    return 0;
}
```

更多黑科技： erase 函数

- erase 函数用于删除字符串指定部分
- **x.erase(n, m)**从字符串x的n号位置开始，连续删除m个字符

```
2019CCF
2019
-----
Process exited after 3.454 seconds with return value 0
请按任意键继续. . .
```

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    string s;
    cin >> s;
    s.erase(4, 3);
    cout << s;
    return 0;
}
```

清理高位零

- 用字符串读入某给定 k 个大整数 n ($n \leq 10^{10000}$), 但是发现该数前面有多余的高位零, 要求清除这些高位零

Sample input	Sample output
3 //k	2019
00002019	10086
10086	4399
04399	

延伸一下

- 如果这些数不是这么大，而是在int或者long long范围内呢？
- 那我们需要（从字符数组读入后转为）开int数组存入每个数的各位数字，然后对每个数组：



```
int i = 0;
while (a[i] == 0) i ++; // i即是第一个非零数字的位置
for ( ; i <= 数的位数; i ++ )
    printf("%d", a[i]);
```

正确做法

- int 型读入本来就会自动过滤高位零的

```
3
00002019
2019
10086
10086
03499
3499
```

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int k, n; scanf("%d", &k);
    while (k --)
    {
        scanf("%d", &n);
        printf("%d\n", n);
    }
    return 0;
}
```

不忘初心，方得始终

类似的初心

- 给定一个只包含“+”、“-”，和数字的算术表达式，求其值

Sample input	Sample output
137+502+51-24	666

不忘初心，方得始终

类似的初心

- 如果我们忘了初心，意思就是：学了后面忘前面
- 我们会要用字符数组接收这条读入的算术表达式，然后在字符数组中扫一遍，根据读到的“+”、“-”运算符分别处理

不忘初心，方得始终

类似的初心

```
137+502+51-24
^Z
666
-----
Process exited after 13.91 seconds with return value 0
请按任意键继续. . .
```

- 那么初心是什么呢？我们试试cin/scanf的功能有多强大

```
#include<bits/stdc++.h>
using namespace std;
int ans, a;
int main()
{
    cin >> ans; //第一个数单独处理
    while (cin >> a) ans += a;
    cout << ans;
    return 0;
}
```

不忘初心，方得始终

更多黑科技：find 函数

- find 函数用于字符串内部定位
- **x.find(y)**: 在字符串x中查找字符串y第一次出现的位置
- 如果没找到，返回-1

```
NOIP
OI
1
-----
Process exited after 13.39 seconds wi
th return value 0
请按任意键继续. . .
```

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    string s, s2;
    cin >> s >> s2;
    cout << s.find(s2);
    return 0;
}
```

更多黑科技：find 函数

- find 函数用于字符串内部定位
- **x.find(y)**: 在字符串x中查找字符串y第一次出现的位置
- 如果没找到，返回-1
- 空格也计算位置

```
Happy new year!  
ye  
10  
-----  
Process exited after 24.6 seconds with return value 0  
请按任意键继续. . .
```

```
#include<bits/stdc++.h>  
using namespace std;  
int main()  
{  
    string s, s2;  
    getline(cin, s);  
    getline(cin, s2);  
    cout << s.find(s2);  
    return 0;  
}
```

更多黑科技：find 函数

- find 函数用于字符串内部定位
- **x.find(y)**: 在字符串x中查找字符串y第一次出现的位置
- 如果没找到，返回-1
- 空格也计算位置
- 还可以从x的指定位置开始查找
- **x.find(y, m)**: 从字符串x的m号位置开始查找字符串y第一次出现的位置

```
Happy new year!  
new  
-1  
-----  
Process exited after 6.363 seconds wi  
th return value 0  
请按任意键继续. . .
```

```
#include<bits/stdc++.h>  
using namespace std;  
int main()  
{  
    string s, s2;  
    getline(cin, s);  
    getline(cin, s2);  
    cout << (int)s.find(s2, 10);  
    return 0;  
}
```

单词统计

- 一般的文本编辑器都有查找单词的功能，该功能可以快速定位特定单词在文章中的位置，有的还能统计出特定单词在文章中出现的次数
- 现在，请你编程实现这一功能，具体要求是：给定一个单词，请你输出它在给定的文章中出现的次数和第一次出现的位置，没有出现则输出-1。注意：匹配单词时，不区分大小写，但要求完全匹配

Sample input	Sample output
to To be or not to be is a question	2 0

分析

- 需要解决几个问题：

1. 大小写不一致

这个可以直接利用`tolower`函数全转换为小写

2. `to` 和 `toto` 是不匹配的，但`find`函数会认为它们匹配

空格	to	空格
空格	toto	空格

```
word = ' ' + word + ' ' ;  
text = ' ' + text + ' ' ;
```

分析

- 需要解决几个问题：

3. find函数很容易找到第一个位置

```
first = text.find(word);
```

4. 而在这之后我们需要继续往后找

To be or not to be is a question



分析

- 需要解决几个问题:

4. 为此我们记录当前找的位置pos，并作为下次查找的起点

```
pos = text.find(word, pos)
```

5. 在往后跳之前，需要把当前找到的word给跳过，不然会重复

```
pos += word.size() - 1;
```

为什么-1?

To be or not to be is a question



分析

- 需要解决几个问题：

6. 最后一个问题，这种查找什么时候结束？

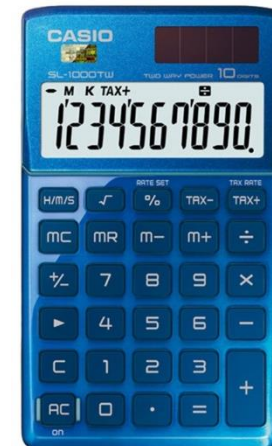
因为当find函数没有找到时候，会返回-1，我们就利用这点

```
while ((pos = text.find(word, pos)) >= 0)
{
    .....
}
```

高精度加法

- A+B problem

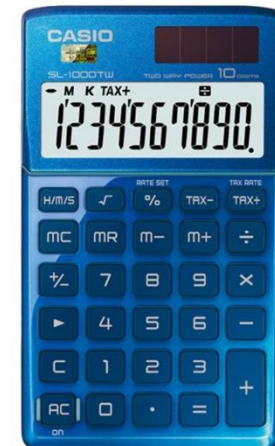
$$0 \leq A, B \leq 10^{10000}$$



Sample input	Sample output
2147483647 9223372034707292160	9223372036854775807

高精度加法

- 其实高精度加法，就是模拟一遍我们小学就会的竖式加法



$$\begin{array}{r} 456 \\ + 789 \\ \hline 1245 \end{array}$$

The diagram illustrates the process of adding two numbers (456 and 789) using a grid-based method. The numbers are placed in a 2x3 grid. A horizontal line is drawn below the grid. Below the line, the digits 1, 2, 4, and 5 are placed in a row of six boxes, representing the result of the addition. The first two boxes are empty, and the last four boxes contain the digits 1, 2, 4, and 5 respectively. The carry values 1, 1, and 1 are shown above the last three columns of the grid.

高精度加法

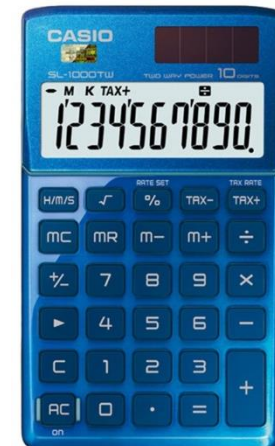
- 但是也有几个问题要解决：

1. 读入数据

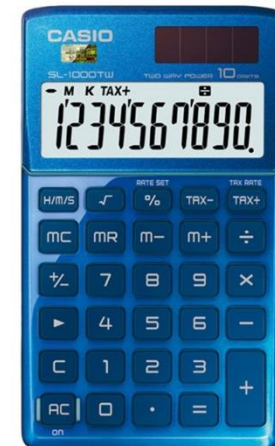
这么大的数据，用字符串读入是最佳选择

2. 字符串虽然读入方便，但后续的运算比较麻烦
把数字一个个存入另外开设的int数组，再进行运算

```
A → a[]; B → b[];  
a[] + b[] → c[];
```

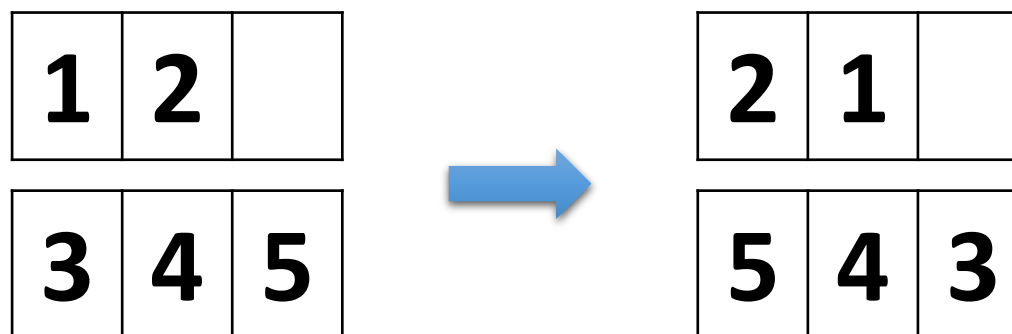


高精度加法



- 但是也有几个问题要解决：

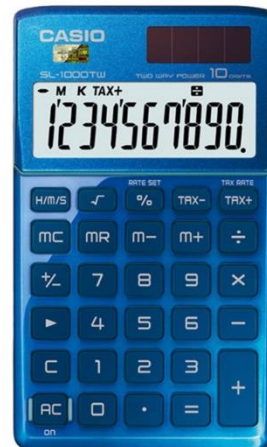
3. 而且，我们会遇到数位对齐的问题：明明要和2竖式对齐的是5
我们把数字从字符串对应转移到int数组时，需要逆序，输出最后的和时，再逆回来



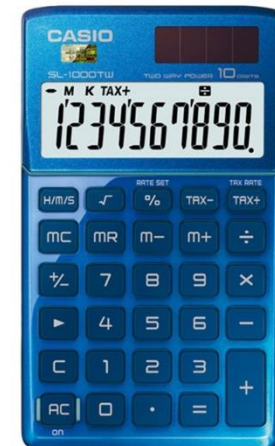
高精度加法

- 程序框架

```
cin >> A >> B;  
逆序A → a[];  
逆序B → b[];  
for (i: 0~max(A.size(),B.size()))  
{  
    模拟竖式加法;  
    和 → c[];  
}  
逆序输出c[];
```



分析



- 我们之前遗留了一个问题：A+B，得到的和的位数，最大有可能达到：

```
max(A.size(), B.size()) + 1
```

- 所以我们有必要对c[]中的最高位（逆序输出前是最后一位）判断是否产生了最高位进位