



# C++编程



# 扫雷游戏（模拟）

- 数据范围可以支持 $n^2$ ，那么直接把地雷分布用二维数组存起来，然后扫一遍即可
- 无需开字符数组，直接开 `int` 数组存储0/1，更方便统计

Sample Input	Sample Output
3 3 *?? ??? ?*?	*10 221 1*1
2 3 ?*? *??	2*1 *21

# 扫雷游戏

```
char x;  
for (int i = 1; i <= n; ++ i)  
    for (int j = 1; j <= m; ++ j)  
    {  
        cin >> x;  
        if (x == '*') a[i][j] = 1;  
    }  
for (int i = 1; i <= n; ++ i)  
    for (int j = 1; j <= m; ++ j)  
        ans[i][j] = a[i-1][j-1] + a[i-1][j] + a[i-1][j+1] + a[i][j-1] + a[i][j+1] + a[i+1][j-1] + a[i+1][j] + a[i+1][j+1];  
//把八个方向的地雷数加起来
```

# 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  
54062552511293261827157706900463454661503762922249904742917932305200563451602  
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 <sub>i</sub> , 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	pneumonoultramicroscopicssilicovolcanoconiosis



# 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 wi
th 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



# 参考代码

```
#include<bits/stdc++.h>
using namespace std;
char a[10010];
int ans;
int main()
{
    scanf("%s", a);
    int len = strlen(a);
    for (int i = 0; i < len; i ++)
        ans += a[i] - '0';
    printf("%d", ans);
    return 0;
}
```

# 回文串



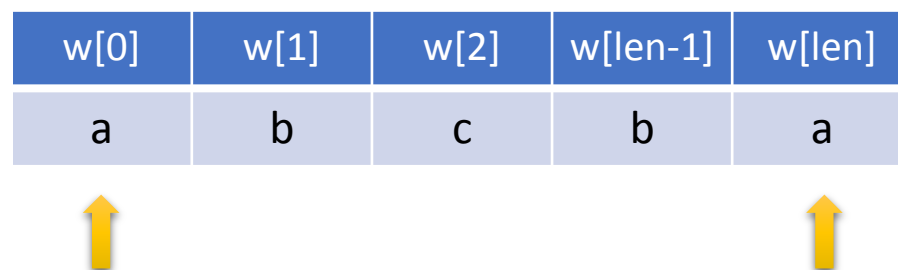
- 形如 “abba”、“abcba” 这样的字串，被称为回文串，特别注意 “a” 也算回文串
- 你的程序要能识别某给定字串（至多10000个小写字母组成），是否为回文串（输出“yes”或“no”）

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



# 参考代码

这里不用考虑  
len的奇偶性



```
#include<bits/stdc++.h>
using namespace std;
char word[10010];
bool judge()
{
    int len = strlen(word);
    for (int i = 0; i < len / 2; i ++){
        if (word[i] != word[len-1-i]) return false;
    }
    return true;
}

int main()
{
    scanf("%s", word);
    if (judge()) printf("yes");
    else printf("no");
    return 0;
}
```

# 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	

# 应战

这是在线算法



```
#include<bits/stdc++.h>
using namespace std;
char word[10010];
bool judge()
{
    int len = strlen(word);
    for (int i = 0; i < len / 2; i ++)
        if (word[i] != word[len-1-i]) return false;
    return true;
}

int main()
{
    int k; scanf("%d", &k);
    while (k --)
    {
        scanf("%s", word);
        if (judge()) printf("yes\n");
        else printf("no\n");
    }
    return 0;
}
```

```
3
abcba
yes
bbc
no
nba
no
```

# 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[4]	word[5]
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;
        }
    }
}
```

# 课外加练

- Luogu uva 455