

计算机程序设计基础

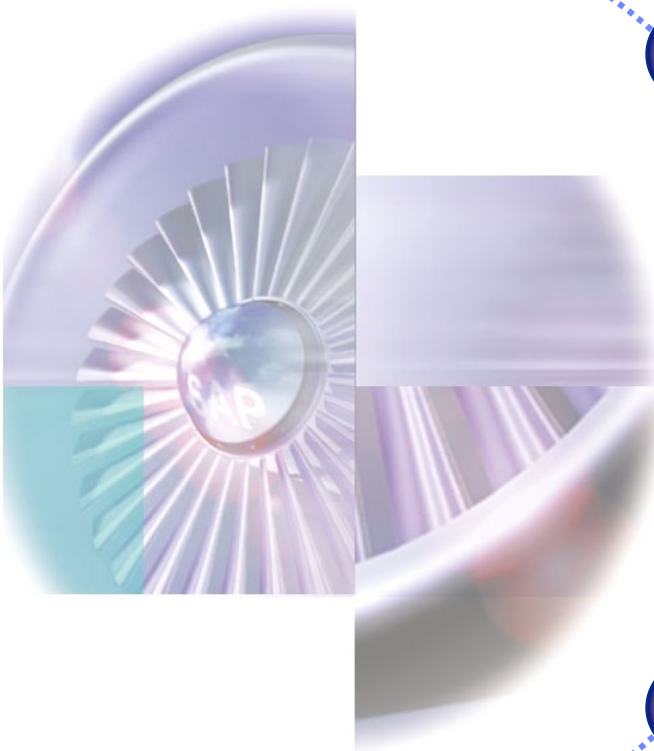
Programming Fundamentals

韩文弢

清华大学计算机系



第四章 数组

- 
- 1 一维数组的定义和引用
 - 2 二维数组的定义和引用
 - 3 字符数组
 - 4 程序举例

4.1 一维数组的定义和引用

```
// 随机产生一个数字不重复的四位数
srand((unsigned)time( NULL ));
while(1)
{
    d = rand() / (RAND_MAX);
    Target = 1000 + (d * 8999); //1000~9999

    // 将它拆分为四位数字
    T1 = Target / 1000;
    T2 = (Target / 100) % 10;
    T3 = (Target / 10) % 10;
    T4 = Target % 10;
    if((T1 != T2) && (T1 != T3) && (T1 != T4) &&
        (T2 != T3) && (T2 != T4) && (T3 != T4))
        break;
}
```

什么是数组？

定义：一组有序、有名、具有相同数据类型的变量。

- 有名：数组有一个名字，如 `a`, `vector`, ...;
- 有序：数组元素连续存放，`0`, `1`, `2`, ...;
- 同类型：长度相同，如 `int`, `double`, `char`;
- 变量：数组的每个元素的值可变。

一维数组的定义方式

类型说明符 数组名 [常量表达式];

指明了数组元素的个数

例如: `char RollerCoaster[5];`

表示数组名为RollerCoaster, 它有5个字符型元素。

RollerCoaster

'X'	'M'	'L'	'F'	'?'
-----	-----	-----	-----	-----

数组元素的访问

数组名[数组下标]

说明：数组下标从 **0** 开始

例如：RollerCoaster[4] = 'H';

RollerCoaster

'X'	'M'	'L'	'F'	'H'
0	1	2	3	4

数组下标的边界

- ☺ 边界：指数组下标的最小值和最大值
- ☺ 若数组长度为N，则数组下标的范围为 **0~N-1**
- ☺ 编译器**不会**进行数组下标的越界检查，越界是未定义行为，可以发生任何事情，一般来说会覆盖相邻的变量或者导致程序崩溃。

数组的存储

- ☺ 数组名：数组的起始地址
- ☺ 数组的存储：在内存中逐个元素连续存放
- ☺ 例如：char RollerCoaster[5];

RollerCoaster



0x001AF768 0x001AF769 0x001AF76A 0x001AF76B 0x001AF76C

前三名

问题描述：

学校要选拔三名同学参加北京市数学竞赛，为在比赛中取得好成绩，学校决定在所有学生中，挑选平时成绩排在前三名的同学组队参赛。请编写一个程序，输入所有学生的成绩，然后找出其中的前三名（学生成绩为整数，人数不确定，但不会超过200人，输入-1时结束）。

问题分析	编程模式
1. 数据个数不确定， 当输入-1时结束	循环条件永真
2. 寻找第1名	最大值/最小值
3. 寻找前3名	最大值+调整

求最大值

```
Max = 0;  
while(1)  
{  
    .....  
    if(Scores[i] > Max)  
        Max = Scores[i];  
    .....  
}
```

Why 0?

i	Scores[i]	Max
初始化	—	0
0	100	100
1	60	100

寻找前3名



First



Second



Third



Scores[i]

```
void main( )
{
    int i, First, Second, Third, Scores[200];
    First = 0;
    Second = 0;
    Third = 0;
    i = 0;
    while(true)
    {
        cout <<"输入第" << i+1 <<"个学生成绩:";
        cin >> Scores[i];
        if(Scores[i] == -1) break;
        if(Scores[i] > First)
        {
            Third = Second;
            Second = First;
            First = Scores[i];
        }
    }
}
```

```
    else if(Scores[i] > Second)
    {
        Third = Second;
        Second = Scores[i];
    }
    else if(Scores[i] > Third)
        Third = Scores[i];
    i++;
}
cout << "前三名成绩分别为: " << First
      << " " << Second << " " << Third;
}
```

能否不用数组？

运行结果:

请输入第 1 个学生的成绩: 100

请输入第 2 个学生的成绩: 60

请输入第 3 个学生的成绩: 90

请输入第 4 个学生的成绩: 80

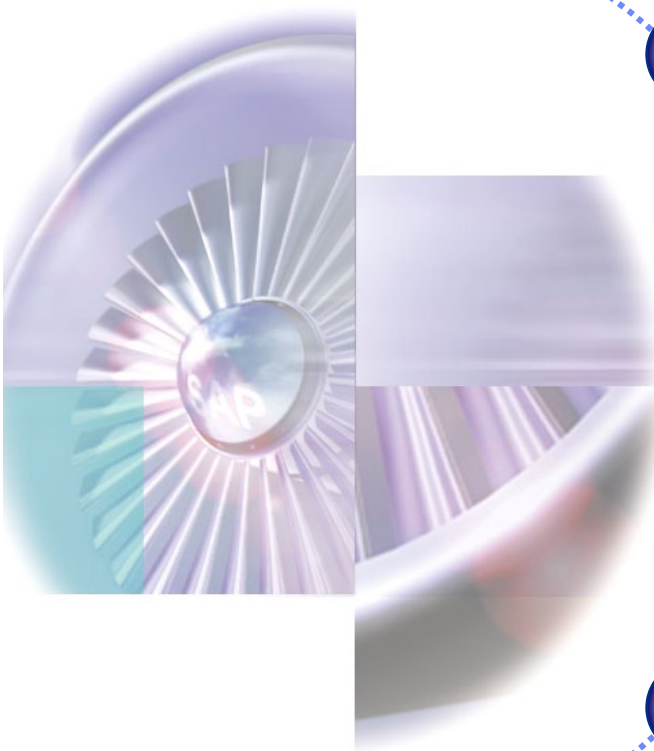
请输入第 5 个学生的成绩: 95

请输入第 6 个学生的成绩: 70

请输入第 7 个学生的成绩: -1

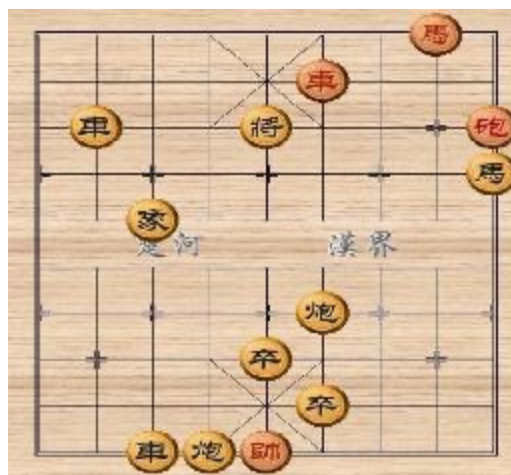
前三名成绩分别为: 100, 95, 90

第四章 数组

- 
- 1 一维数组的定义和引用
 - 2 二维数组的定义和引用
 - 3 字符数组
 - 4 程序举例

Why 二维数组？

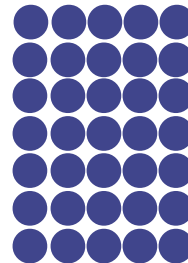
- 一维数组：一字长蛇阵
- 队列、棋盘、二维码...



一维数组



二维数组



二维数组的定义方式

类型说明符 **数组名[常量表达式][常量表达式];**

 ↑ ↑
 行数 列数

例如: `int scores[150][16];`

例如：现有七个学生的四次平时作业成绩

作业0 作业1 作业2 作业3

学生0	22	15	25	25
学生1	12	12	25	20
学生2	5	17	25	24
学生3	15	19	25	13
学生4	2	0	25	25
学生5	25	22	24	21
学生6	8	4	25	12

```
int score[7][4];  
score[0][0] = 22  
score[6][3] = 12  
2*score[3][0]=30
```

二维数组的一些术语：

```
int a[3][4];
```

- **a**是一个**int**类型的二维数组，大小为 3×4 ；
- **a[0][0]**, **a[0][1]**, **a[0][2]**, ..., **a[2][3]**都是数组**a**的元素，每个元素都是一个**int**类型的变量；
- 该数组第 **i** 行、第 **j** 列的元素为**a[i][j]**，其中 **i** 称为数组的行下标，**j** 称为数组的列下标。
- 边界指的是数组下标的最小值和最大值（在这里行下标的边界是**0**和**2**，列下标的边界是**0**和**3**）

另一种观点：

二维数组可以看作是一种特殊的一维数组：即它的元素又是一个一维数组。例如，对于 `int a[3][4]`，它有三个元素：`a[0]`、`a[1]`和`a[2]`，每个元素又是一个包含4个元素的一维数组。

a { `a[0]` ----- `a[0][0]`, `a[0][1]`, `a[0][2]`, `a[0][3]`
`a[1]` ----- `a[1][0]`, `a[1][1]`, `a[1][2]`, `a[1][3]`
`a[2]` ----- `a[2][0]`, `a[2][1]`, `a[2][2]`, `a[2][3]`

在定义二维数组 `int a[3][4]` 之后，内存的分布情况如下：

内存分布：

按行存放，

先顺序存放

第一行的所

有元素，再

存放第二行

的元素。

0x0012FF7C

0x0012FF78

0x0012FF74

0x0012FF70

0x0012FF6C

0x0012FF68

0x0012FF64

0x0012FF60

0x0012FF5C

0x0012FF58

0x0012FF54

0x0012FF50

a[2][3]

a[2][2]

a[2][1]

a[2][0]

a[1][3]

a[1][2]

a[1][1]

a[1][0]

a[0][3]

a[0][2]

a[0][1]

a[0][0]

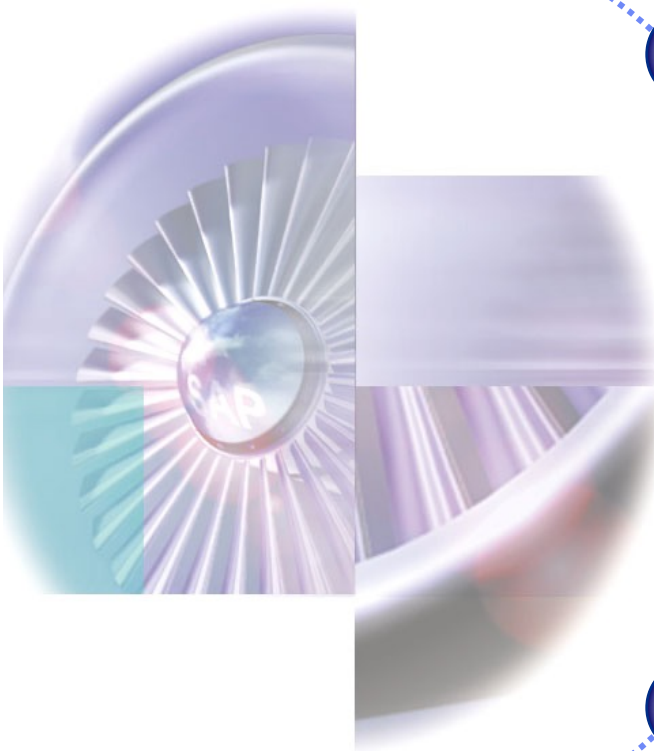
a[2]

a[1]

a[0]

同一列能否当一个数组？

第四章 数组

- 
- 1 一维数组的定义和引用
 - 2 二维数组的定义和引用
 - 3 字符数组
 - 4 程序举例

字符数组： 用来存放字符数据的数组。

例如：

```
char c[10] = {'I', ' ', 'a', 'm', ' ', 'h', 'a', 'p', 'p', 'y'};
```

字符数组的定义、初始化和引用与通常的数组都是一样的。

字符串和字符串结束标志:

在 C/C++ 语言当中, 将字符串作为字符数组来处理, 并以字符'\0'来表示字符串的结束标志。

如: "How do you do."

H	o	w	_	d	o	_	y	o	u	_	do	.	\0
---	---	---	---	---	---	---	---	---	---	---	----	---	----

若有多个 '\0'呢?

字符串处理函数cstring

(一) puts函数（字符串输出）

功能：将一个字符串输出到屏幕。

格式：int puts(const char *string);

例子：char str[] = {"Hello"};
puts(str);
puts("Hello");

(二) gets函数（字符串输入）

功能：从键盘输入一个字符串到字符数组。

格式：char *gets(char *buffer);

例子：

```
char str[10];  
gets(str);
```

自动补零！

（三）strcat函数（字符串连接）

格式: `char *strcat(char *destination,
const char *source);`

功能：连接两个字符数组中的字符串，把 **source** 连接到 **destination** 的后面。

例子: `char src[] = "world";`
`char dest[30] = "hello ";`
`strcat(dest, src); // "hello world"`

（四）strcpy函数（字符串复制）

格式： `char *strcpy(char *destination,
 const char *source);`

功能： 将字符串source复制到字符数组destination中去。

例子： `char dest[30];
char src[] = "hello";
strcpy(dest, src);`

(五) strcmp函数（字符串比较）

格式: `int strcmp(const char *string1,
const char *string2);`

功能: 比较字符串string1和string2。

说明: 如果string1 < string2, 返回一个负整数(-1);
如果string1 = string2, 返回0;
如果string1 > string2, 返回一个正整数(1);

例子: `strcmp(str1, str2);`
`strcmp("America", "China");`
`strcmp("Love", "Life");`
`strcmp("ab", "abc");`

strlen函数： 返回字符串的实际长度（不包括末尾的 ‘\0’字符）。

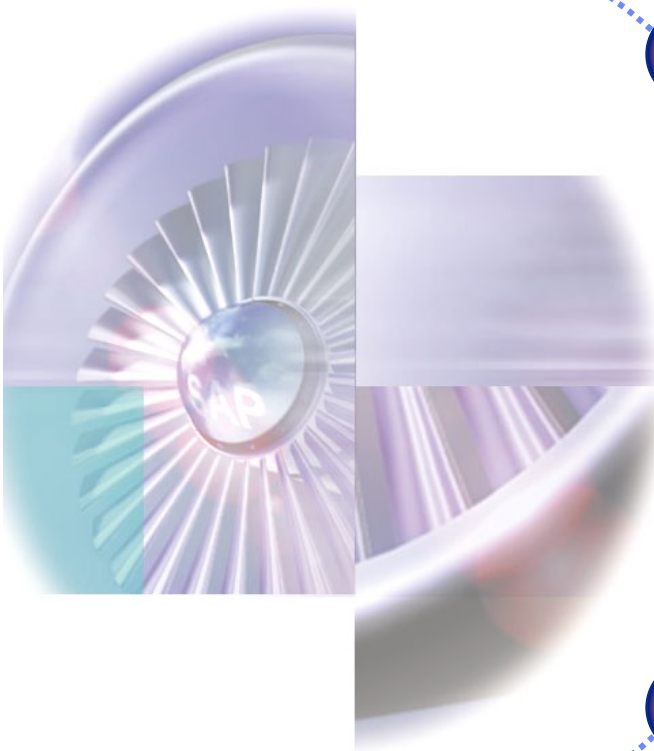
.....

填空

/* 程序功能是将无符号八进制数字构成的字符串转换为十进制整数。如:若输入字符串为556, 则输出十进制整数366 */

```
#include <cstring>
void main( )
{
    char s[10];
    int i, n = 0;
    gets(s);
    i = 0;
    while( s[i] != '\0' )
    {
        n = n*8 + s[i] - '0';
        i++;
    }
    cout << n << endl;
}
```

第四章 数组

- 
- 1 一维数组的定义和引用
 - 2 二维数组的定义和引用
 - 3 字符数组
 - 4 程序举例

4.4.1 字符串拼接

问题描述：

接受键盘输入的两个字符串，并将其首尾相接后输出。每个字符串内部不含空格，两个字符串之间以空白符分隔。

问题分析：

(1) 数据结构

字符串的存储需要用字符数组

(2) 算法要点

字符串拼接方法：先找到第一个字符串的末尾，然后将第二个串的字符逐个添加到末尾。

注意，要去掉第一个串的结束符 ‘\0’ 。

```
int main()
{
    char str1[50], str2[20];
    int i, j;
    cout << "输入字符串1: ";
    cin >> str1;
    cout << "输入字符串2: ";
    cin >> str2;
    i = 0;
    while(str1[i] != '\0')    i++;
    str1[i++] = ' ';
    for(j = 0; str2[j] != '\0'; i++, j++)
        str1[i] = str2[j];
    cout << "拼接结果: " << str1;
    return (0);
}
```

运行结果:

输入字符串1: **hello**

输入字符串2: **world!**

拼接结果: **hello world!**

字符串的输入方法

- `cin >> str`: 输入一个字符串，直到碰上第一个空白字符(空格、Tab或换行);
- `cin.getline(str, len)`: 输入一行字符，直到碰上第一个换行;
- 前者输入的字符串不能包含空格和Tab，而后者输入的字符串可以。若分别使用这两种不同的输入方法，则当用户敲入 “aa bb cc”时，得到的输入字符串分别是: “aa” 和 “aa bb cc”。

4.4.2 回文问题

问题描述：

回文是指具有如下特性的一个短语：该短语顺过来读和反过来读所得到的字母序列是完全相同的。例如：“**level**”就是一个回文。编写一个程序，输入一个短语，然后判断它是否是回文。

算法思路

回文判断用首尾比较算法，将字符串的第一个字符与最后一个字符进行比较，第二个字符和倒数第二个字符进行比较，...。若出现不相等的情形，则立即退出。

"level"

l	e	v	e	l	'\0'
---	---	---	---	---	------

str 0 1 2 3 4 5

1. `str[0]` ↔ `str[4]`: 'l' ↔ 'l'; 若字符串长度
2. `str[1]` ↔ `str[3]`: 'e' ↔ 'e'; 为N，则最多
3. `str[2]` ↔ `str[2]` 需要 $N/2$ 次
4. 完成! 比较即可。

```
#include <iostream>
#include <cstring>
using namespace std;
void main( ){
    char str[80], flag;
    int i, j;
    cout << "输入一个字符串: ";
    cin >> str;
    i = 0; j = strlen(str)-1;
    flag = 'Y';
    for(; i < j; i++, j--)
    {
        if(str[i] != str[j])
        {
            flag = 'N';
            break;
        }
    }
    if(flag == 'Y') cout << str << "是一个回文\n";
    else cout << str << "不是一个回文\n";
}
```

如何测试本程序？

测试数据:

输入一个字符串: a

a是一个回文

输入一个字符串: aa

aa是一个回文

输入一个字符串: ab

ab不是一个回文

输入一个字符串: aba

aba是一个回文

输入一个字符串: abb

abb不是一个回文

输入一个字符串: abccba

abccba是一个回文

输入一个字符串: abcdefghijkl

abcdefghijkl不是一个回文

汉字回文?

4.4.3 冒泡排序

问题描述：

输入一组数据，
然后把它们按照从小到大
或从大到小排列。最后
输出排序后的结果。



a[0]	8	8	8	8	8	8	1
a[1]	9	9	9	9	9	1	8
a[2]	2	2	2	2	1	9	9
a[3]	4	4	4	1	2	2	2
a[4]	3	3	1	4	4	4	4
a[5]	1	1	3	3	3	3	3

初始值 1,3互换 1,4互换 1,2互换 1,9互换 1,8互换 一遍扫描完成

a[0]	1	1	1	1	1	1
a[1]	8	8	8	8	8	2
a[2]	9	9	9	9	2	8
a[3]	2	2	2	2	9	9
a[4]	4	4	3	3	3	3
a[5]	3	3	4	4	4	4

上次结果 3,4互换 2,3不动 2,9互换 2,8互换 二遍扫描完成

a[0]	1	1	1
a[1]	2	2	2
a[2]	3	3	3
a[3]	4	4	4
a[4]	8	8	8
a[5]	9	9	9

上次结果 8,9不动 五遍扫描完成

算法结束



1
2
3
4
8
9

问题分析

1. 需要多少轮循环？每轮循环得到的最小值保存在何处？
2. 在每轮循环中，需要比较多少次？每次比较哪两个元素？

1、循环次数

从图中可以看出最小的一个数经过第一遍扫描就交换到a[0]中。如果将a[5]视为水底，a[0]视为水面：

最轻的(最小的)一个数 1 最先浮到水面，交换到a[0]；
次轻的 2 经过第二遍扫描后交换到a[1]；
再轻的 3 经过第三遍扫描后交换到a[2]；

...

依此类推，有6个数，前5个数到位需要5遍扫描，而最大的数自然落在a[5]中。因此，6个数只需5遍扫描，若数组长度为n，每遍扫描得到的最小值，被交换到a[j]当中，则j的取值范围为： **$j = 0, 1, 2, n-2$**

2、比较次数

核心问题是 i 和 j 的范围！

再来看在每遍扫描中，相邻两个数组元素的比较次数。

(1) 当 $j = 0$ 时，需将 $a[5]$ 与 $a[4]$ 比较、 $a[4]$ 与 $a[3]$ 比、 $a[3]$ 与 $a[2]$ 比、 $a[2]$ 与 $a[1]$ 比、 $a[1]$ 与 $a[0]$ 比。在比较这5次后，最小数到达 $a[0]$ 。如果抽象为“ $a[i]$ 与 $a[i-1]$ 比”，则 $i = 5, 4, 3, 2, 1$ ；

(2) 当 $j = 1$ 时，即第二遍搜索，需比较4次，之后次小的一个数到达了 $a[1]$ 。这时 $a[1]$ 就不必再与 $a[0]$ 进行比较了。因此 $i = 5, 4, 3, 2$ ；

(3) 当 $j = 2$ 时，即第三遍搜索， $i = 5, 4, 3$ ；当 $j = 3$ 时， $i = 5, 4$ ；当 $j = 4$ 时， $i = 5$ 。

若数组长度 n ，则 i 的取值范围： $i = n-1, n-2, \dots, j+1$ 51

冒泡排序算法设计：

为了表述方便，定义以下 3 个变量：

n —— 待排序的数的个数，这里 **n = 6**

j —— 扫描遍数， **$j = 0, 1, 2, \dots, n-2$**

i —— 第 **j** 遍扫描中待比较元素的下标，即
 $a[i]$ 与 $a[i-1]$ 比，其中 $i = n-1, n-2, \dots, j+1$

```

void main( )
{
    int i, j, n, temp, a[6];
    n = 6;
    for (i = 0; i < n; i = i+1)
    {
        cout << "请输入待排序的数a[" << i << "]=";
        cin >> a[i];
    }
    for(j = 0; j <= n-2; j++)          // 冒泡排序，外层循环
        for(i = n-1; i >= j+1; i--)    // 内层循环
        {
            if(a[i] < a[i-1])           // 把小的数往上冒
            {
                temp = a[i];
                a[i] = a[i-1];
                a[i-1] = temp;
            }
        }
    for(i = 0; i < n; i++)  cout << a[i] << " ";
}

```

4.4.4 找数

问题描述:

在下列数据网格中寻找一个整数，方法是：
(1)在某一行从左往右或从右往左找；(2)在某一系列从上往下或从下往上找。如整数3451在该网格中出现了两次，即(3, 1, 下)和(3, 3, 右)。需考虑回绕的情形，如5412出现在(1, 5, 右)。

1	2	3	4	5	4
2	5	4	3	2	1
3	5	3	4	5	1
4	4	3	4	5	2
5	3	4	3	4	3
1	4	5	2	3	4

问题分析

1. 数据网格如何存放？
2. 该整数有多少位？
3. 假设有 M 位，从某个位置出发，如何得到从左往右、从右往左、从上往下和从下往上的 M 个数字？
4. 在得到了 M 个数字之后，如何把它合成为一个整数？

假设有M位，数组为a，当前位置为 (i, j)

- 从左往右: $(j+k) \% 6$ ($k = 0, 1, \dots, M-1$)
- 从右往左: $(j-k+6) \% 6$ ($k = 0, 1, \dots, M-1$)
- 从上往下: $(i+k) \% 6$ ($k = 0, 1, \dots, M-1$)
- 从下往上: $(i-k+6) \% 6$ ($k = 0, 1, \dots, M-1$)

1	2	3	4	5	4
2	5	4	3	2	1
3	5	3	4	5	1
4	4	3	4	5	2
5	3	4	3	4	3
1	4	5	2	3	4


```

void main()
{
    int a[6][6] = { 1, 2, 3, 4, 5, 4,
                    2, 5, 4, 3, 2, 1,
                    3, 5, 3, 4, 5, 1,
                    4, 4, 3, 4, 5, 2,
                    5, 3, 4, 3, 4, 3,
                    1, 4, 5, 2, 3, 4};

    int i, j, k;
    int value, temp, N=6, M;

    cin >> value;
    temp = value;
    M = 0;
    while(temp != 0){
        M++;
        temp /= 10;
    }
}

```

```

for(i = 0; i < N; i++)
for(j = 0; j < N; j++)
{
    temp = 0;
    for(k = 0; k < M; k++)
        temp = temp * 10 + a[i][(j+k)%N];
    if(temp == value) cout <<i+1<<" "<<j+1<<"右\n";
    temp = 0;
    for(k = 0; k < M; k++)
        temp = temp * 10 + a[i][(j-k+N)%N];
    if(temp == value) cout <<i+1<<" "<<j+1<<"左\n";
    temp = 0;
    for(k = 0; k < M; k++)
        temp = temp * 10 + a[(i-k+N)%N][j];
    if(temp == value) cout << i+1<<" "<<j+1<<"上\n";
    temp = 0;
    for(k = 0; k < M; k++)
        temp = temp * 10 + a[(i+k)%N][j];
    if(temp == value) cout << i+1<<" "<<j+1<<"下\n";
}
}

```

测试数据:

411

1 6 下

3454

1 3 右

3451

3 1 下

3 3 右

154

3 6 左

6 1 上

5435

3 5 左

4 5 下

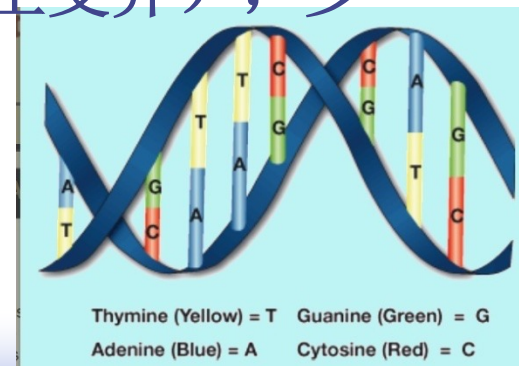
1	2	3	4	5	4
2	5	4	3	2	1
3	5	3	4	5	1
4	4	3	4	5	2
5	3	4	3	4	3
1	4	5	2	3	4

4.4.5 DNA序列

问题背景：

人类基因组计划的第一阶段于2000年6月26日胜利结束，我国科研工作者圆满地完成了其中的1%的测序工作。

众所周知，组成DNA的碱基有四种：腺嘌呤(A)、鸟嘌呤(G)、胞嘧啶(C)、胸腺嘧啶(T)。对任意两个人，其染色体上的DNA序列大部分相同，但总会有少数碱基对不同。这种不同是由基因的变异引起，但每个人的变异位置不尽相同。这样，对于大部分位点来说，很可能的情形是：大部分人在该位点上的碱基是一致的（没有发生变异），少数人具有不同的碱基（发生了变异）。



这就给科学家一个启示：在测序过程中，若仅仅使用一个人的样本，在很多位点上测出的结果就不具有代表性；而如果能测出多个人的序列，则有可能“整合”出一段具有人类共性的序列出来，这样更有利于研究。

如，若要测人的某一段DNA序列，得到4人的样本：

AAAGGCCT

AGAGCTCT

AAGGATCT

AAACTTCT AAAGATCT

整合规则：(1) 取出在每个位置上出现次数最多的碱基作为整合后该位置上的碱基；(2) 若某个位置出现次数最多的碱基不止一种，则优先选择A，其次C、G和T。

问题描述：

对一组DNA序列进行整合。

输入：N行($2 \leq N \leq 10$)，每一行是一个DNA序列，具有相同的长度(2到100之间)；

输出：一个字符串，即整合后的序列。

问题分析：

一、数据结构

- (1) 这组DNA序列如何存放？
- (2) 如何记录在每个位置上各种碱基的出现次数？

二、算法

- (1) 如何实现字符到数组下标的映射？
- (2) 如何统计各位置上每种碱基的出现次数
- (3) 如何找到出现次数最多的碱基？
- (4) 若次数相同，如何实现ACGT优先顺序

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    char DNA[10][101], Result[101];
    char Nb[4] = {'A', 'C', 'G', 'T'};
    double Num[4], Max;
    int i, j, N, len;

    cin >> N;
    for(i=0; i<N; i++) cin >> DNA[i];
    len = strlen(DNA[0]);
```



```
for(j = 0; j < len; j++)
{
    Num[0] = 0.3;    Num[1] = 0.2;
    Num[2] = 0.1;    Num[3] = 0;
    for(i = 0; i < N; i++)
    {
        switch(DNA[i][j])
        {
            case 'A':    Num[0] += 1;    break;
            case 'C':    Num[1] += 1;    break;
            case 'G':    Num[2] += 1;    break;
            case 'T':    Num[3] += 1;    break;
        }
    }
}
```

```
    Max = 0;
    for(i = 0; i < 4; i++)
    {
        if(Num[i] > Max)
        {
            Max = Num[i];
            Result[j] = Nb[i];
        }
    }
}
Result[j] = 0;
cout << "整合结果: " << Result << endl;
return 0;
}
```

运行结果:

4

AAAGGCCT

AGAGCTCT

AAGGATCT

AAACTTCT

整合结果: AAAGATCT

C++ 标准库的字符串和向量

◆ C 的数组和字符串的缺陷

- 长度固定，空间需要程序员事先分配
- 对字符串，有大量越界的隐患

◆ C++ 语言标准库提供

- `std::string` 类型（在 `<string>` 中）
- `std::vector<T>` 类型（在 `<vector>` 中）
- 长度可按需伸缩，自动分配空间

本讲小结

- ◆ 一维数组
- ◆ 二维数组
- ◆ 用字符数组表示的字符串