

【C++语法】Lesson 2 讲义

【C++语法】Lesson 2 讲义

1. 程序流程结构

1.1. 顺序结构

1.2. 分支结构

1.2.1. if 语句

1.2.2. switch...case 语句

1.2.3. 三目运算符

1.3. 循环结构

1.3.1. while 循环

1.3.2. do...while 循环

1.3.3. for 循环

1.3.4. 嵌套循环

1.3.4. 【例题】输出九九乘法表

1.4. 跳转语句

1.4.1. break 语句

1.4.2. continue 语句

1.4.3. goto 语句

2. 拆位运算

2.1. 除运算与模运算的性质

2.2. 循环拆位

2.3. 【例题】水仙花数

3. 数组

3.1. 一维数组

3.1.1. 一维数组的定义方式

3.1.2. 一维数组数组名的用途

3.1.3. 【例题】找最小值

3.1.4. 【例题】数组元素逆置

3.2. 二维数组

3.2.1. 二维数组的定义

3.2.2. 二维数组数组名的用途

3.2.3. 【例题】成绩统计

4. 输入输出进阶

4.1. scanf函数、printf函数与cin对象、cout对象的区别和联系

4.1.1. printf与cout

4.1.2. scanf与cin

4.1.3. 什么时候用printf和scanf?

4.2. 取消流同步提高cin、cout速度


4.3. 【面试】数组的下标为何从0开始? 若从下标1开始存储数据有什么好处?

※ 字符数组与字符串

字符数组与字符串的定义

字符数组与字符串的输入输出

C++11加强for循环

本文档为程序设计算法协会C++语法课培训第二节课的讲义，制
作人：漂至 

1. 程序流程结构

1.1. 顺序结构

- 程序中的语句按从上到下的顺序逐一执行，如上面A+B Problem这道例题

1.2. 分支结构

- 也称选择结构，主要通过if语句和switch...case语句来实现

1.2.1. if 语句

1. 单行格式if语句

```
1  if(条件) {  
2      条件满足执行的语句;  
3      ...  
4  }  
5  // 如果条件满足时执行的语句只有一条，可以不加大括号  
6  if(条件) 条件满足执行的语句
```

- 输入一个人的年龄，若 <18岁 输出“您已成年”

```
1  int age;  
2  cin>>age;  
3  if(age<18) {  
4      cout<<"您未成年";  
5  }
```

2. 多行格式if语句

```
1  if(条件) {
2      条件满足执行的语句;
3      ...
4  }
5  else {
6      条件不满足执行的语句;
7      ...
8  }
```

- 输入一个人的年龄，若 <18岁 输出“您未成年”，反之输出“您已成年”

```
1  int age;
2  cin>>age;
3  if(age<18) {
4      cout<<"您未成年";
5  }
6  else {
7      cout<<"您已成年";
8  }
```

3. 多条件if语句

```
1  if(条件1) {
2      条件1满足执行的语句;
3      ...
4  }
5  else if(条件2) {
6      条件2满足执行的语句;
7      ...
8  }
9  else if(条件3) {
10     条件3满足执行的语句;
11     ...
12 }
```

```

12 }
13 ...
14 else {
15     所有条件都不满足执行的语句;
16     ...
17 }

```

- 输入一个人的年龄，若 <18岁 输出“少年”，若 >=18岁 且 <35岁 输出“青年”，若 >=35岁 且 <60岁 输出“中年”，若 >=60岁 输出“老年”

```

1  int age;
2  cin>>age;
3  if(age<18) {
4      cout<<"少年"<<endl;
5  }
6  else if(age>=18 && age<35) {
7      cout<<"青年"<<endl;
8  }
9  else if(age>=35 && age<60) {
10     cout<<"中年"<<endl;
11 }
12 else {
13     cout<<"老年"<<endl;
14 }

```

4. 嵌套if语句

```

1  if(条件1)
2  {
3      条件1满足执行的语句
4      if (条件1-1)
5      {
6          满足条件1并满足条件1-1;

```

```

7         ...
8     }
9     else
10    {
11        满足条件1不满足条件1-1;
12        ...
13    }
14 }
15 else if(条件2)
16 {
17     条件2满足执行的语句;
18     ...
19 }
20 ...
21 else
22 {
23     所有条件都不满足执行语句;
24     ...
25 }

```

- 举个🍊

```

1 #include<iostream>
2 using namespace std;
3 int main() {
4     int score=0;
5     cout<<"请输入考试分数:>";
6     cin>>score;
7     if(score>600) {
8         cout<<"我考上了一本大学"<<endl;
9         if(score>700) {
10             cout<<"我考上了北大"<<endl;
11         }
12         else if(score>650) {
13             cout<<"我考上了清华"<<endl;
14         }

```

```
15         else {
16             cout<<"我考上了人大"<<endl;
17         }
18     }
19     else if(score>500) {
20         cout<<"我考上了二本大学"<<endl;
21     }
22     else if(score>400) {
23         cout<<"我考上了三本大学"<<endl;
24     }
25     else {
26         cout<<"我未考上本科"<<endl;
27     }
28     return 0;
29 }
30
```

1.2.2. switch...case 语句

- 基本结构如下
- break语句用于结束当前语句，后续在循环结构中我们也会讲到

```
1  switch(控制表达式) {
2      case 常量1:
3          语句;
4          break;
5      case 常量2:
6          语句;
7          break;
8      case 常量3:
9          语句;
10         break;
11     case 常量4:
12         语句;
```

```
13         break;
14     default:
15         语句;
16         break;
17 }
```

- 设计一个程序模拟按键，当按下W时输出跳跃，按下S输出蹲下，按下A输出左移，按下D输出右移，按下J输出轻击，按下K输出重击，按下L输出闪避，按下其他键输出无操作

```
1 char ch; // 因为要输入的是字符型,所以用单字符型来承接
2 cin>>ch;
3 switch(ch) {
4     case 'w':
5         cout<<"跳跃"<<endl;
6         break;
7     case 's':
8         cout<<"蹲下"<<endl;
9         break;
10    case 'A':
11        cout<<"左移"<<endl;
12        break;
13    case 'D':
14        cout<<"右移"<<endl;
15        break;
16    case 'J':
17        cout<<"轻击"<<endl;
18        break;
19    case 'K':
20        cout<<"重击"<<endl;
21        break;
22    case 'L':
23        cout<<"闪避"<<endl;
24        break;
25    default:
26        cout<<"无操作"<<endl;
```

```
27         // break;
28     }
```

- 再来看个例子

```
1  #include<iostream>
2  using namespace std;
3
4  int main() {
5      // 请给电影评分
6      // 10~9: 经典
7      // 8~7: 非常好
8      // 6~5: 一般
9      // 5分以下 烂片
10     int score=0;
11     cout<<"请给电影打分:>";
12     cin>>score;
13     switch(score) {
14         case 10:
15         case 9:
16             cout<<"经典"<<endl;
17             break;
18         case 8:
19             cout<<"非常好"<<endl;
20             break;
21         case 7:
22         case 6:
23             cout<<"一般"<<endl;
24             break;
25         default:
26             cout<<"烂片"<<endl;
27             break;
28     }
29     return 0;
30 }
```


- 需注意：
 1. switch语句中表达式类型只能是整型或者字符型
 2. case里如果没有break，那么程序会一直向下执行
- 总结：与if语句比，对于多条件判断时，switch的结构清晰，执行效率高，缺点是switch不可以判断区间
- 在选择分支结构时，大多数情况都使用if语句

1.2.3. 三目运算符

1 | 语法：表达式1? 表达式2 : 表达式3

- 解释：
 - 如果表达式1的值为真，执行表达式2，并返回表达式2的结果；
 - 如果表达式1的值为假，执行表达式3，并返回表达式3的结果。
- 和if语句比较，三目运算符优点是短小整洁，缺点是如果用嵌套，结构不清晰

```
1 | int a=10,b=20,c;  
2 | c=a>b?a:b; // 如果a>b,则c=a,否则c=b
```

1.3. 循环结构

1.3.1. while 循环

- 作用：只要满足循环条件，就执行循环语句
- 注意：在执行循环语句时，程序必须提供跳出循环的出口，否则出现死循环

```
1 while(循环条件) {
2     循环语句;
3 }
```

```
1 int cnt=1; // cnt:count, 计数器
2 while(cnt<=10) {
3     cout<<cnt<<' ';
4     cnt++;
5 }
```

- 如果循环条件永真，则死循环

```
1 while(1) {
2     cout<<"haha"<<'\\n';
3 }
```

1.3.2. do...while 循环

- 与while的区别在于do...while会先执行一次循环语句，再判断循环条件

```
1 do {
2     循环语句;
3 }
4 while(循环条件)
```

```
1 int cnt=0;
2 do {
3     cout<<cnt<<endl;
4     cnt++;
5 }
6 while(cnt<=10);
```

- 通常情况下考虑到程序的规范性和可读性，更常使用while循环而非do...while

1.3.3. for 循环

- for循环是最常用的循环语句

```
1  for(起始表达式; 条件表达式; 末尾循环体) {  
2      循环语句;  
3  }
```

- 上面这段代码等价于下面这段，PS：任何for循环和while循环之间都可以相互转换

```
1  int i=1;  
2  while(i<=10) {  
3      cout<<i<<' '  
4      i++;  
5  }
```

- PS：当循环次数已知时，我们更常使用for循环，循环次数未知但循环结束条件已知，更常使用while循环

1.3.4. 嵌套循环

- 如果题目要求我们打印如下图案，我们可以在循环体中再嵌套一层循环

```
1  *  
2  * *  
3  * * *  
4  * * * *  
5  * * * * *
```

- 第1层打印一个1个*，第2层打印一个2个*，第i层打印一个i个*

```
1 for(int i=1;i<=5;i++) {  
2     for(int j=1;j<=i;j++) {  
3         cout<<"*<<' '  
4     }  
5     cout<<endl;  
6 }
```

1.3.4. 【例题】输出九九乘法表

```
1 // 从1×1 ~ 9×9  
2 for(int i=1;i<=9;i++) {  
3     for(int j=1;j<=i;j++) {  
4         cout<<j<<'*<<i<<"="<<i*j<<' '  
5     }  
6     cout<<endl;  
7 }
```

1.4. 跳转语句

1.4.1. break 语句

- 作用：用于跳出选择结构或者循环结构
- break使用的时机：
 1. 出现在switch条件语句中，作用是终止case并跳出switch（见§6.2.2.）
 2. 出现在循环语句中，作用是跳出当前的循环语句
 3. 出现在嵌套循环中，跳出最近的内层循环语句

```

1 #include<iostream>
2 using namespace std;
3
4 int main() {
5     int cnt=1;
6     while(1) {
7         if(cnt>10) break;
8         cout<<cnt<<' ';
9         cnt++;
10    }
11    return 0;
12 }

```

```

1 #include<iostream>
2 using namespace std;
3
4 int main() {
5     for(int i=1;i<=10;i++) {
6         for(int j=1;j<=10;j++) {
7             if(j>i) break;
8             else cout<<"*"<<' ';
9         }
10        cout<<endl;
11    }
12    return 0;
13 }

```

1.4.2. continue 语句

- 作用：在循环语句中，跳过本次循环体中余下尚未执行的语句，继续执行下一次循环
- 注意：continue并没有使整个循环终止，只是跳转到下一轮，而break会终止循环
- 如下面一段代码，用于输出1~100中的奇数

```

1  #include<iostream>
2  using namespace std;
3
4  int main() {
5      for(int i=1;i<=100;i++) {
6          // 如果i能被2整除,说明i是2的倍数,即偶数,则跳过,
           不输出
7          if(i%2==0) continue;
8          cout<<i<<' ';
9      }
10     return 0;
11 }

```

1.4.3. goto 语句

- 可以无条件跳转语句（强制跳转）
- 如果标记的名称存在，执行到goto语句时，会跳转到标记的位置
- 注意：goto语句被认为是有害的，1968年计算机科学家艾兹格·迪杰斯特拉（某些同学看到这个名字是不是DNA动了）在论文《Go To Statement Considered Harmful》中主张应该避免使用无条件的goto语句，因为它会导致流程变得难以追踪和维护，降低代码的可读性和可维护性，并提倡使用顺序、分支、循环等结构化的语句来代替goto，以提高程序的清晰度和逻辑性
- 这就是为什么在后面的高级语言如Java和Python中都见不到goto语句了

2. 拆位运算

- 拆位运算是循环与算术运算符的结合，目的是将多位数的每一位按照个、十、百、千...的顺序提取出来

2.1. 除运算与模运算的性质

- 注意：除可以用于整型之间、浮点型之间、整型与浮点型之间（结果自动转型为浮点型），而模运算只能用于整型之间
- 假设现在有一数 `num=157`，如何提取其个位数？十位数？百位数？

```
1 int num=157;
2 cout<<num/10<<'\n'; // 除10, 157/10=15, 丢掉最后一位
3 cout<<num/100<<'\n'; // 除100, 157/100=1, 丢掉最后两位
4 cout<<num/1000<<'\n'; // 除1000, 157/1000=0, 丢掉最后三位,没了
5
6 cout<<num%10<<'\n'; // 模10, 157/10=15...7, 得到最后一位7
7 cout<<num%100<<'\n'; // 模100, 157/100=1...57, 得到最后两位57
8 cout<<num%1000<<'\n'; // 模1000, 157/1000=0...157, 得到最后三位157
```

- 所以对于一个整型，我们很容易发现一个规律
 - 该数对 10^i 取模，得到最后 i 位数构成的整型
 - 该数除以 10^i ，表示丢掉最后 i 位数，取最后构成的整型

2.2. 循环拆位

- 有了如上规律，加上一个简单的循环，我们就能实现拆位
- 模10可以得到最后一位，除10可以丢掉最后一位，我们不断取出最后一位，再丢掉最后一位，就可以实现提取每一位的数字，当这个数字最后变成0的时候停止循环

```

1 // 当n==0时退出循环
2 while(n) {
3     int x=n%10; // 取最后一位
4     cout<<x<<' ';
5     n/=10; // x=x/10, 丢最后一位
6 }

```

2.3. 【例题】水仙花数

题目描述：水仙花数是指一个 3 位数，它的每个位上的数字的 3 次幂之和等于它本身

例如： $1^3 + 5^3 + 3^3 = 153$

请求出所有3位数中的水仙花数

- 水仙花数指一个三位数的各位数字的立方和等于这个数字本身，那我们把一个三位数的每一位分别提取出来，求其立方再加起来，看是否等于这个数本身
- 输出m~n范围内的所有水仙花数，则最后还要再最外面加一层循环从 $[m, n]$

```

1 #include<iostream>
2 using namespace std;
3
4 int main() {
5     // 三位数的范围是[100,900]
6     // 遍历m到n之间的所有数字
7     for(int i=100;i<=999;i++) {
8         // 注意!!!这里格外重要,用一个临时变量来代表num
9         int num=i;
10        int sum=0; // 用来记录每一个三位数的各个数位的
        数字的累加和
11        // 提取每一位

```



```

12         while(num) {
13             int x=num%10;
14             sum+=x*x*x;
15             num/=10;
16         }
17         // 是否为水仙花数
18         if(sum==i) cout<<i<<' ';
19     }
20     return 0;
21 }

```

- 那么我们的题单上还有一道题也是关于水仙花数的题，稍微有一些不一样，大家下来去做一下

O - 水仙花数

春天是鲜花季节，水仙花就是其中最迷人的代表，数学上有个水仙花数，他是这样定义的：“水仙花数”是指一个三位数，它的各位数字的立方和等于其本身，比如：153=1³+5³+3³。现在要求输出所有在m和n范围内的水仙花数。

Input

输入数据有多组，每组占一行，包括两个整数m和n（100≤m≤n≤999）。

Output

对于每个测试实例，要求输出所有在给定范围内的水仙花数，就是说，输出的水仙花数必须大于等于m,并且小于等于n，如果有多个，则要求从小到大排列在一行内输出，之间用一个空格隔开；如果给定的范围内不存在水仙花数，则输出no；每个测试实例的输出占一行。

Sample

Input	copy	Output	copy
100 120		no	
300 380		370 371	

3. 数组

本节讲义部分参考黑马程序员C++入门讲义

- 所谓数组，就是一个集合中存放了多个相同类型的数据元素

- 特点：

1. 数组中的每个数据元素都是相同的数据类型
2. 数组由连续的内存位置组成的
3. 数组中每个元素都有一个与之相对应的位置，这个位置被称为“下标”或“索引”，我们通常说下标。注意：数组的下标从0开始的

3	1	2	5	4	9	7	2
0	1	2	3	4	5	6	7

- 数组的命名规则同变量，不要与其他变量重名

3.1. 一维数组

3.1.1. 一维数组的定义方式

- 可以通过如下三种方式定义一维数组
- PS：数组在定义时就已经确定了其大小，后续无法修改，我们也不讲解变长数组，后续若有机会继续给大家讲解基于STL的数据结构专题，会讲到如何使用vector代替变长数组

- ```
1 数据类型 数组名[数组长度]; // 只是声明，不初始化
2 数据类型 数组名[数组长度] = { 值1, 值2 ... }; // 给定
 数组长度，可以不赋满
3 数据类型 数组名[] = { 值1, 值2 ... }; // 不给定数组长
 度，根据中括号中具体元素个数设置大小
```

- 我们来看下面这段示例

```
1 #include<iostream>
2 using namespace std;
3
4 int main() {
5 // 定义方式1
```

```
6 // 数据类型 数组名[元素个数];
7 int score[10];
8
9 // 利用下标赋值
10 score[0]=100;
11 score[1]=99;
12 score[2]=85;
13
14 // 利用下标输出
15 cout<<score[0]<<endl;
16 cout<<score[1]<<endl;
17 cout<<score[2]<<endl;
18 cout<<'\n';
19
20 // 第二种定义方式
21 // 数据类型 数组名[元素个数] = {值1, 值2, 值3
...};
22 // 如果{}内不足10个数据, 剩余数据用0补全
23 int score2[10]=
{100,90,80,70,60,50,40,30,20,10};
24
25 // 逐个输出
26 // cout<<score2[0]<<endl;
27 // cout<<score2[1]<<endl;
28
29 // 一个一个输出太麻烦, 因此可以利用循环进行输出
30 for(int i=0;i<10;i++) {
31 cout<<score2[i]<<endl;
32 }
33 cout<<'\n';
34
35 // 定义方式3
36 // 数据类型 数组名[] = {值1, 值2 , 值3 ...};
37 int score3[]=
{100,90,80,70,60,50,40,30,20,10};
38
```

```

39 for(int i=0;i<10;i++) {
40 cout<<score3[i]<<endl;
41 }
42 return 0;
43 }

```

### 3.1.2. 一维数组数组名的用途

1. 可以统计整个数组在内存中的长度

```

1 int arr[10]={1,2,3,4,5,6,7,8,9,10};
2 cout<<"整个数组所占内存空间为: "<<sizeof(arr)<<endl;
3 cout<<"每个元素所占内存空间为: "<<sizeof(arr[0])
 <<endl;
4 cout<<"数组的元素个数为: "
 <<sizeof(arr)/sizeof(arr[0])<<endl;

```

2. 数组名代表数组首元素地址（或者说是指向首元素的指针，指针是什么后面再讲）

```

1 int arr[10]={1,2,3,4,5,6,7,8,9,10};
2 cout<<arr<<endl; // 输出首元素地址,16进制表示
3 cout<<arr+1<<endl; // 第二个元素地址,可以看到字节数+了
 4,因为int占4字节
4 cout<<*(arr+1)<<endl; // 指针解引用操作,取到数组中第二
 个元素的值

```

### 3.1.3. 【例题】找最小值

- 题目描述：一个数组为  $\text{int arr}[5] = \{300, 350, 200, 400, 250\}$  请找出并打印其中的最小值

```

1 int a[5]={300,350,200,400,250};
2 int mmin=999; // mmin用于保存数组中的最小值,先初始化为一个极大值
3 // 5个元素,若下标从0开始存储,通常写作i=0;i<5
4 for(int i=0;i<5;i++) {
5 if(a[i]<mmin) mmin=a[i];
6 }
7 cout<<mmin;

```

### 3.1.4. 【例题】数组元素逆置

- 将一个数组中的元素存储顺序逆转
- 首先讲一下如何交换两个变量中的元素

```

1 #include<iostream>
2 using namespace std;
3
4 int main() {
5 int a=3,b=5,tmp; // tmp表示temporary,临时变量
6 cout<<a<<' '<<b<<'\n';
7 // 交换a和b中的元素
8 tmp=a; // a给tmp
9 a=b; // b给a
10 b=tmp; // tmp(原a)给b
11 // 检验是否交换了
12 cout<<a<<' '<<b<<'\n';
13 // 为什么不能a=b,b=a?自行思考哦
14 return 0;
15 }

```

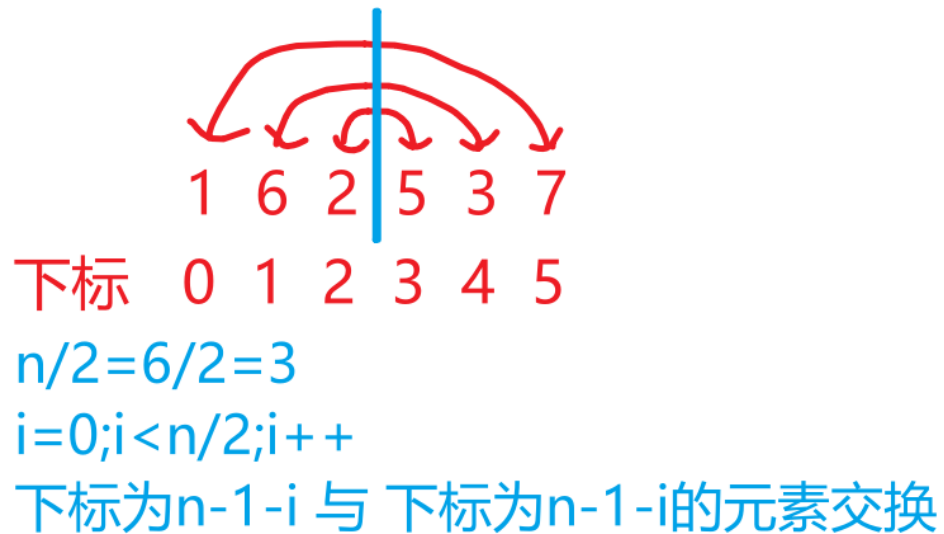
- 在python里，交换两个元素可以简写如下

```

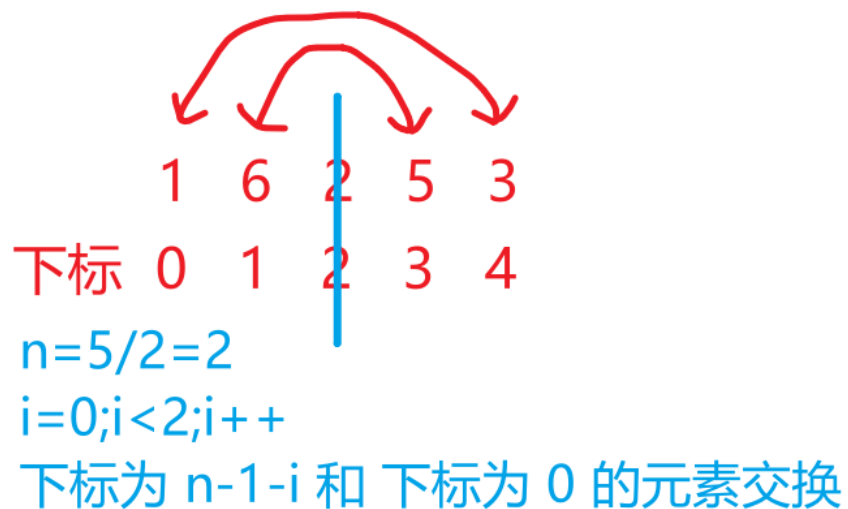
1 a,b=b,a

```

- 数组元素逆置，假如现在有一个数组为  
 $int\ arr = [x_1, x_2, \dots, x_n]$ ，我们希望将数组中的元素倒过来，  
 应输出  $arr = [x_n, x_{n-1}, \dots, x_1]$ 
  - 如果数组个数为偶数



- 如果数组个数为奇数



- 经过推导，你已经发现使用一段相同的代码就能代表这两种情况

```

1 #include<iostream>
2 using namespace std;
3
4 int main() {
5 int a1[]={1,6,2,5,3,7};
6 int a2[]={1,6,2,5,3};
7
```

```

8 // 获取arr1和arr2的长度
9 int n1=sizeof(a1)/sizeof(a1[0]);
10 int n2=sizeof(a2)/sizeof(a2[0]);
11
12 // 逆置
13 for(int i=0;i<n1/2;i++) {
14 // 交换 n1-1-i 和 i
15 int tmp=a1[i];
16 a1[i]=a1[n1-1-i];
17 a1[n1-1-i]=tmp;
18 }
19 for(int i=0;i<n2/2;i++) {
20 // 交换 n2-1-i 和 i
21 int tmp=a2[i];
22 a2[i]=a2[n2-1-i];
23 a2[n2-1-i]=tmp;
24 }
25
26 // 打印测试
27 for(int i=0;i<n1;i++) {
28 cout<<a1[i]<<' ';
29 }
30 cout<<'\n';
31 for(int i=0;i<n2;i++) {
32 cout<<a2[i]<<' ';
33 }
34 // 也可以提前声明i，在for循环里不写int i
35 return 0;
36 }

```

- 上面这种做法实际修改了数组中元素的顺序并顺序输出，有的同学不那么老实，直接倒着输出，也行，主要看题目需要输出什么哈

```

1 for(int i=n-1;i>=0;i--) {
2 cout<<a[i]<<' ';
3 }

```

## 3.2. 二维数组

|     |   |   |   |
|-----|---|---|---|
| 列下标 | 0 | 1 | 2 |
| 行下标 |   |   |   |
| 0   | 1 | 2 | 3 |
| 1   | 4 | 5 | 6 |

- 一维数组你可以理解成一行，二维数组就是既有行也有列，构成了一个矩阵，第一维是行，第二维是列
- 最简单的理解方法，每一行都是一个一维数组，有多行，构成一个二维数组
- 同样的只能存储相同类型的元素

### 3.2.1. 二维数组的定义

```

1 数据类型 数组名[行数][列数]; // 只声明，未赋初始值
2 数据类型 数组名[行数][列数] = { {数据1, 数据2 } ,
 {数据3, 数据4 } } // 一行一行赋值，最直观的赋值方法
3 数据类型 数组名[行数][列数] = { 数据1, 数据2, 数据
 3, 数据4}; // 一个一个赋值，自行计算每一个元素值应该在哪一行
4 数据类型 数组名[][列数] = { 数据1, 数据2, 数据3, 数
 据4}; // 二维数组可以不指派行数，但列数必须要确定

```



- 示例代码

```
1 #include<iostream>
2 using namespace std;
3
4 int main() {
5 // 方式1
6 // 数组类型 数组名 [行数][列数]
7 int arr[2][3];
8 arr[0][0]=1;
9 arr[0][1]=2;
10 arr[0][2]=3;
11 arr[1][0]=4;
12 arr[1][1]=5;
13 arr[1][2]=6;
14
15 // 矩阵用两重循环遍历输出
16 for(int i=0;i<2;i++) {
17 for (int j=0;j<3;j++) {
18 cout<<arr[i][j]<<" ";
19 }
20 cout<<endl;
21 }
22
23 // 方式2
24 // 数据类型 数组名[行数][列数]={数据1, 数据2} ,
25 // {数据3, 数据4}};
26 int arr2[2][3]={
27 {1,2,3},
28 {4,5,6}
29 };
30
31 // 方式3
32 // 数据类型 数组名[行数][列数]={数据1, 数据2, 数据
33 // 3, 数据4};
34 int arr3[2][3]={1,2,3,4,5,6};
```

```

33
34 // 方式4
35 // 数据类型 数组名[][列数]={数据1, 数据2, 数据3, 数
 据4};
36 int arr4[][3]={1,2,3,4,5,6};
37
38 system("pause"); // 暂停函数
39 return 0;
40 }

```

### 3.2.2. 二维数组数组名的用途

#### 1. 可以查看二维数组所占内存空间

```

1 // 二维数组数组名
2 int arr[2][3]={
3 {1,2,3},
4 {4,5,6}
5 };
6
7 cout<<"二维数组大小: "<<sizeof(arr)<<endl;
8 cout<<"二维数组一行大小: "<<sizeof(arr[0])<<endl;
9 cout<<"二维数组元素大小: "<<sizeof(arr[0][0])
 <<endl;
10
11 cout<<"二维数组行数: "<<sizeof(arr)/sizeof(arr[0])
 <<endl;
12 cout<<"二维数组列数: "
 <<sizeof(arr[0])/sizeof(arr[0][0])<<endl;

```

#### 2. 获取二维数组首地址

```

1 int arr[2][3]={
2 {1,2,3},
3 {4,5,6}
4 };
5 cout<<arr<<endl;
6 cout<<arr+1<<endl; // 地址增加了12个字节,因为每行有3列,
 一个元素4字节
7 cout<<*(arr[0])<<endl; // arr[0]表示第一行首元素地址
8 cout<<*(arr[1]+2)<<endl; // 输出第二行第三个元素

```

### 3.2.3. 【例题】成绩统计

- 案例描述：有三名同学（张三，李四，王五），在一次考试中的成绩分别如下表，请分别输出三名同学的总成绩

|    | 语文  | 数学  | 英语  |
|----|-----|-----|-----|
| 张三 | 100 | 100 | 100 |
| 李四 | 90  | 50  | 100 |
| 王五 | 60  | 70  | 80  |

```

1 int main() {
2 int scores[3][3]={
3 {100,100,100},
4 {90,50,100},
5 {60,70,80},
6 };
7 string names[3] = { "张三","李四","王五" };
8 // 遍历每个同学
9 for (int i=0;i<3;i++) {
10 int sum=0;
11 // 遍历每门课程
12 for (int j=0;j<3;j++) {

```

```
13 sum+=scores[i][j];
14 }
15 cout<<names[i]<<"同学总成绩为: "
16 <<sum<<endl;
17 }
18 return 0;
19 }
```

## 4. 输入输出进阶

- 数组在算法竞赛中是非常非常常用的，讲解了数组过后我们要重新审视一下基于数组的输入输出

### 4.1. scanf函数、printf函数与cin对象、cout对象的区别和联系

- `scanf()` 函数和 `printf()` 函数来自C语言，和 `cin`, `cout` 一样用作输入输出，接下来介绍其基本用法

#### 4.1.1. printf与cout

##### 1. 输出字面量

```
1 printf("Hello world!\n");
2 cout<<"Hello world!"<<endl;
```

- ##### 2. 当输出携带变量时，`printf()` 不可分段，使用一个或多个占位符表示变量在输出文本中的位置

```

1 int a=3,b=4,c=5;
2 printf("a=%d\n",a); // 一个变量
3 printf("a=%d, b=%d, c=%d\n",a,b,c); // 多个变量顺序
 承接
4 cout<<"a="<<a<<endl;
5 cout<<"a="<<a<<" , b="<<b<<" , c="<<c<<endl;

```

- 不同的变量类型使用不同的占位符，上述案例中 `int` 型的占位符是 `%d`，常见的占位符有以下几种

```

1 #include<iostream>
2 using namespace std;
3
4 int main() {
5 int a=1;
6 char b=2;
7 double c=3.141592;
8 char d[]={'H','e','l','l','o'}; // 字符数组
9 string e="Hello world";
10
11 printf("a=%d, b=%c, c=%f, d=%s\n",a,b,c,d);
12 // 整型 %d
13 // 字符型 %c
14 // 浮点型 %f
15 // 字符数组 %s
16 }

```

### 3. printf格式化输出

- 之前我们讲过使用 `cout` 做格式化输出，如设置固定宽度、占位字符等，如果交给 `printf` 写法非常简单
- 因此，输出结果需要格式化操作时，我们通常使用 `printf()`

```

1 #include <iostream>
2 #include <iomanip> // 包含头文件以使用格式化输出相关功能

```

```

3
4 using namespace std;
5
6 int main() {
7 // 1.指定输出宽度
8 cout << setw(10) << "Hello" << endl;
9 printf("%10s\n", "Hello"); // 输出宽度为 10
10
11 // 2.指定精度
12 cout << fixed << setprecision(2) << 3.14159
13 << endl; // 输出精度为 3
14
15 // 3.指定输出宽度的同时设置指定精度
16 cout << setw(10) << fixed << setprecision(2)
17 << 3.14159 << endl;
18 printf("%10.2f\n", 3.14159);
19
20 // 4.控制对齐方式
21 cout << left << setw(10) << "Hello" <<
22 endl; // 左对齐
23 printf("%-10s\n", "Hello"); // 左对齐
24 cout << right << setw(10) << "Hello" <<
25 endl; // 右对齐
26 printf("%10s\n", "Hello"); // 右对齐
27 cout << left << setw(10) << 12345 << endl;
28 printf("%10d\n", 12345);
29 printf("%010d\n", 12345); // 补充前导0到宽度为
30 10
31
32 // 4.设置填充字符
33 cout << setfill('-') << setw(10) << "Hello"
34 << endl; // 使用 '-' 填充
35 // printf不支持直接自定义设置填充字符
36
37 // 5.输出格式设置

```

```

33 cout << scientific << 3.14159 << endl; //
 科学计数法输出
34 printf("%e\n", 3.14159); // 科学计数法输出
35 cout << hex << 255 << endl; // 十六进制输出
36 printf("%x\n", 255); // 十六进制输出
37 cout << oct << 255 << endl; // 八进制输出
38 printf("%o\n", 255); // 八进制输出
39
40 return 0;
41 }

```

### 4.1.2. scanf与cin

- `scanf` 在输入变量时同样需要占位符指定输入类型，而 `cin` 可以直接输入多种类型的数据

```

1 #include <iostream>
2 using namespace std;
3
4 int main() {
5 // int→%d, long long→%lld, char→%c, float→%f,
 double→%lf, char[]→%s, 字符串不可用scanf直接输入
6 int a;
7 char b;
8 double c;
9 char d[10];
10 string e;
11 // 1. 输入单变量
12 cout<<"请输入一个整数:>";
13 scanf("%d",&a); // "&"叫做取地址符，即将%d输入到
 a这个变量所在的地址
14
15 // 注意,我们在表示输入结束的时候会输入回车到缓冲区,回
 车符是'\n'

```

```

16 // scanf("%c") 存储了这个多余的回车符,可以通过
 getchar()函数或者cin.ignore()来忽略该回车符
17 cin.ignore();
18
19 cout<<"请输入一个字符:>";
20 scanf("%c",&b);
21 cout<<"请输入一个浮点数:>";
22 scanf("%lf",&c);
23 cout<<"请输入一个字符串:>";
24 scanf("%s",d); // 字符数组名代表首元素地址,因此不
 需要取地址符
25
26 // 2. 输入多变量
27 int x,y;
28 cout<<"请输入两个整数(以空格或者回车分隔):>";
29 scanf("%d%d",&x,&y); // 以空格或者换行符为分隔以
 区分
30 cout<<x<<' '<<y<<'\n';
31 // cin>>a>>b>>c>>d>>e; // 字符数组可以直接cin哦,默
 认从下标0开始存储
32 // 3. 输入数值类型数组
33 int arr1[10];
34 printf("请输入10个数字(以空格或者回车分隔):>");
35 for(int i=0;i<10;i++) {
36 scanf("%d",&arr1[i]);
37 // cin>>arr1[i];
38 }
39 // 打印测试
40 for(int i=0;i<10;i++) {
41 printf("%d ",arr1[i]);
42 // cout<<arr1[i]<<' ';
43 }
44
45 return 0;
46 }

```



### 4.1.3. 什么时候用printf和scanf?

- `cin` 和 `cout` 处理输入输出的速度比 `scanf` 和 `printf` 更慢，当问题规模（数组大小） $n \leq 10^5$  时二者速度相当，但当问题规模（数组大小） $n > 10^5$  时 `scanf` 和 `printf` 处理输入输出的速度明显比 `cin` 和 `cout` 快，这是因为：
  - 同步机制：`cin` 和 `cout` 默认与C标准I/O库（即 `scanf` 和 `printf`）同步，这意味着C++标准库和C标准库的输入输出流是一起工作的，这样做是为了确保在同一个程序中混合使用 `cin/cout` 和 `scanf/printf` 时不会出现输出顺序混乱的问题，然而，这种同步机制会带来额外的开销，导致 `cin` 和 `cout` 的性能低于 `scanf` 和 `printf`
  - 流缓冲：`cin`和`cout`是基于C++流，需要更多的操作来处理类型安全和流缓冲
  - 格式化开销：`cin`和`cout`在处理复杂数据类型时，会进行更多格式化操作，这些额外的操作会带来性能上的损耗
- 有时候代码提交后会被反馈TLE，把所有`cin`和`cout`换成`scanf`和`printf`说不定就能AC（我是不是讲的很细啊啊啊，很多刷了很久题才知道的技巧全都毫无保留）

## 4.2. 取消流同步提高cin、cout速度

- 刚才说过`cin`和`cout`由于同步机制等速度不如`scanf`和`printf`，可以通过下面三句代码关闭这种同步提高`cin`和`cout`处理输入输出的速度，但仍不及`scanf`和`printf`

```
1 ios::sync_with_stdio(0); // 优化输入输出，取消
 cin/out和scanf/printf的关联
2 // 在需要频繁切换输入和输出的情况下，可以加上以下两条代码，
 表示取消cin和cout的绑定，允许输出操作在输入操作完成之前进
 行(以前是输入完了才能输出)
3 cin.tie(0);
4 cout.tie(0);
```

### 4.3. 【面试】数组的下标为何从0开始？若从下标1开始存储数据有什么好处？

- 数组的下标从0开始是C语言的设计选择，这个选择后来被C++、Java、C#等多种编程语言继承。这种设计的主要原因是：能够减少CPU指令运算
  - 下标从0开始，数组寻址的公式为：
$$arr[i] = base\_address + i \times type\_size$$
 $base\_address$ 是首元素地址， $i$ 是偏移量，首元素的偏移量为0， $type\_size$ 是该数据类型的字节数
  - 如果下标从1开始，数组寻址公式是：
$$arr[i] = base\_address + (i - 1) \times type\_size$$
如此一来每次寻址都需要多做一次  $i - 1$  的操作
  - 而数组这种基础数据结构，都是频繁间接或直接被使用的，所以要减少CPU的消耗
- 在算法竞赛中，我们经常看到一种写法，虽说数组下标是从0开始，但是很多人都会从下标1开始存储元素，举个例子

未赋初值，初值默认为0

实际存储和使用的数组，共5个元素

|      |   |   |   |   |   |   |
|------|---|---|---|---|---|---|
| 数组a: | 0 | 3 | 2 | 6 | 1 | 4 |
| 下标:  | 0 | 1 | 2 | 3 | 4 | 5 |

- 这样做的好处是：我们实际要操作的数据元素，在数组中的位置和其下标是对应的，写法简便易理解，操作数据也很方便，建议这样写，有时候题目要推公式，有时候从下标0开始存公式比从下标1好使，这个时候就要灵活使用存储方式了

## ※ 字符数组与字符串

- 字符数组也分一维和二维，只是这里数组的数据类型变为字符型 `char`
- 为什么这里单独把字符数组拿出来讲呢？主要是为了和字符串进行一个对比，它们二者之间有许多相似之处

### 字符数组与字符串的定义

- 字符数组有如下三种定义方法：
- 注意，字符数组的结尾一定是 `\0`，`\0` 这种写法就是我们之前讲过的转义字符，在C/C++中代表空字符，在计算机中约定俗成表示字符数组的结束，如果我们没有手动给 `\0`，C++也会默认帮我们加上末尾的结束符 `\0`

```
1 char str1[10]; // 如果不赋初值，那么所有值都为 '\0'
2 char str2[10]={'H','e','l','l','o','\0'}; // 如果
 给定大小，但用不完，那么没被赋值的区域都为 '\0'
3 char str3[10]={'H','e','l','l','o'}; // 不手动
 给'\0'，默认最后一位也是'\0'表示结束符
4 char str4[10]="Hello"; // 字符数组也可以直接用字符串的
 方式为其赋值
```

- 而字符串的定义就相当简单了

```
1 string str1;
2 string str2="Hello";
```

## 字符数组与字符串的输入输出

- 首先讲输入，字符数组可以用 `scanf` 也可以用 `cin`，也可以用 `for` 循环展开依次输入单个字符
- 字符串只能用 `cin` 输入

```
1 char str1[10];
2 cin>>str1;
3 scanf("%s",str1); // 直接输入整个字符串，不用加取地址
 符&，因为数组名已代表首元素地址
4 for(int i=0;i<10;i++) {
5 // cin>>str1[i];
6 scanf("%c",&str1[i]);
7 }
8
9 string str2;
10 cin>>str2;
```

- 再讲输出，字符数组可以用 `printf` 也可以用 `cout`，也可以用 `for` 循环展开依次输出单个字符
- 字符串只能用 `cout` 输出，或者 `for` 循环展开，其字符串长度可以用 `string` 类的函数获取，函数我们下节课讲

```
1 #include <iostream>
2 #include <iomanip> // 包含头文件以使用格式化输出相关功能
3 using namespace std;
4
5 int main() {
6 char str1[10]="Hello";
7 cout<<str1<<endl;
8 printf("%s\n",str1);
9 // 字符数组末尾是'\0',所以循环条件是str1[i]!='\0'
10 for(int i=0;str1[i]!='\0';i++) {
```

```
11 printf("%c",str1[i]);
12 }
13 cout<<'\\n';
14
15 string str2;
16 cin>>str2;
17 cout<<str2<<endl;
18 // str.size()是string自带的函数用于获取字符串长度，
 后面会讲哦
19 for(int i=0;i<str2.size();i++) {
20 // printf("%c",str2[i]);
21 cout<<str2[i];
22 }
23 return 0;
24 }
```

## C++11加强for循环

- C++的版本在不断迭代，早在2011年提出的C++11中，就引入了加强for循环这一新特性，我们简单介绍一下，因为后面讲数据结构的时候（不知道有没有这个机会了）会经常用到

以下是C++的一些主要版本以及它们的推出时间：

1. **C++98** (1998年) - 第一个标准化版本，正式名称为ISO/IEC 14882:1998。
2. **C++03** (2003年) - 小幅修订版，主要是对C++98的一些补充和修正。
3. **C++11** (2011年) - 引入了许多新特性，如自动类型推导 (`auto`)、范围for循环、智能指针、lambda表达式等。
4. **C++14** (2014年) - 是C++11的增量更新，进一步完善和优化了C++11的一些特性。
5. **C++17** (2017年) - 添加了更多的库特性和语言改进，例如结构化绑定、`if constexpr`、并行算法等。
6. **C++20** (2020年) - 带来了模块、协程、三向比较 (`spaceship operator`) 等重大改进，还包括范围库、概念等。
7. **C++23** (计划于2023年) - 预计会包括进一步的语言改进和新特性。

这些版本的发布标志着C++语言的不断发展和演进，每个版本都带来了新的功能和改进。你有在特定版本的C++中遇到过什么问题或者有兴趣深入了解哪一部分的新特性吗？

- 对数组使用加强for循环

```
1 int arr1[10]={1,2,3,4,5};
2 for(int i:arr1) cout<<i<<' '; // 表示将arr1中每个元素都用i代指, i为int型, 输出i即可遍历每个元素
3 cout<<'\n';
4
5 char str1[10]={'H','e','l','l','o'};
6 for(char ch:str1) cout<<ch<<' '; // 字符数组中每个元素都是char, 就用char代指咯
7 cout<<'\n';
8 // auto关键字可以直接识别数组中每个元素的类型, 以后经常用
9 for(auto ch:str1) cout<<ch<<' ';
10 cout<<'\n';
11
12 // 之前讲过字符串可以理解为变长字符数组, 所以也可以用加强for循环
13 string str2="Hello";
14 for(auto ch:str2) cout<<ch<<' ';
```