

SZU-ACM周训2.0

循环结构、数组、结构体、函数

Kaslen_

目录

➤ 循环

➤ 数组

➤ 结构体

➤ 函数

➤ 时间复杂度简单分析

循环结构

for 循环

- 格式：for(循环初始表达式;循环条件;改变量表达式){代码块}



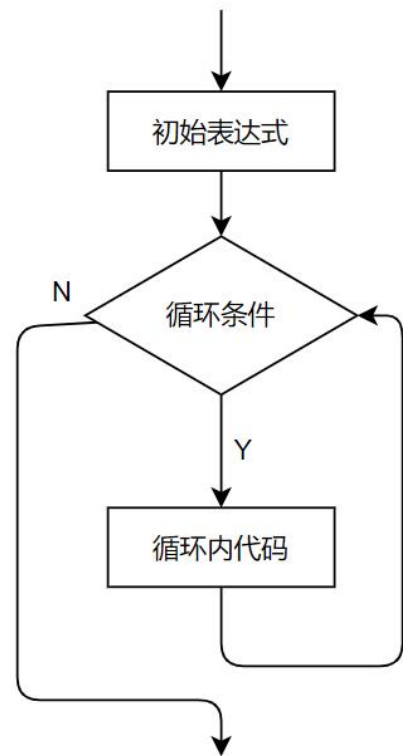
```
1  for (int i=1;i<=n;i++){  
2      代码块  
3  }
```



```
1  for (;;){  
2      代码块  
3  }
```


for 循环

- 使用最广泛的一种循环；
- 所有的循环都能写成 for 循环形式；
- 循环内部定义的变量以及初始表达式处定义的变量的生存周期为整个循环过程；
- 循环的流程图如右图。




while 循环

- 格式: `while(循环条件){代码块}`



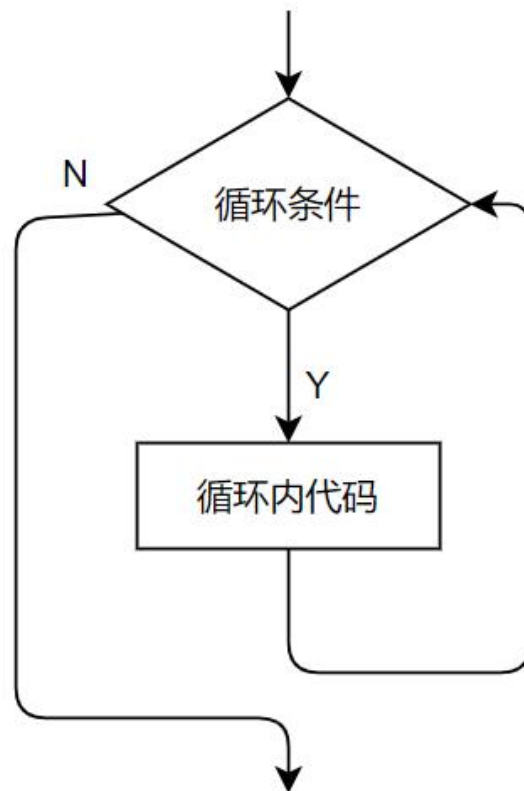
```
1  while (i<=10){  
2      cout<<i<<'\\n';  
3      i++;  
4  }
```



```
1  while (cin>>x){  
2      cout<<x<<'\\n';  
3  }
```

while 循环

- 同 for 循环，循环内部定义的变量生存周期为整个循环过程；
- 循环的流程图如右图。



do-while 循环

- 格式:

do{

 代码块

}while(循环条件);



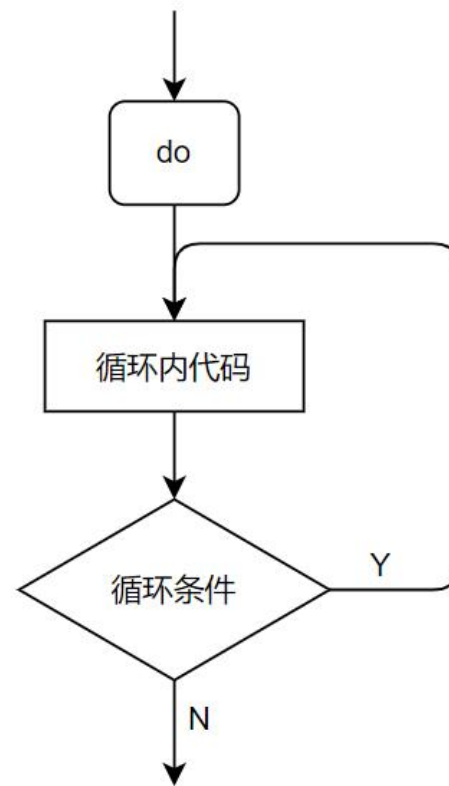
```
1  int i=1;
2  do{
3      cout<<i<<'\\n';
4      i++;
5  }while(i<=10);
```



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


do-while 循环

- 与 for、while 循环的差异在于无论是否满足循环条件，至少会运行一次循环内代码；
- 循环的流程图如右图；



不同循环相互转换



```
1  for (int i=1;i<=10;i++){  
2      cout<<i<<'\\n';  
3  }
```



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

- 以上两种写法等价
- 一般不用 do-while 循环

例题

- 输入一个正整数 $n(1 \leq n \leq 1000)$ ，求： $\sum_{i=1}^n \frac{1}{i}$ ，输出保留10位小数。



```
1 void solve(){
2     int n;
3     cin>>n;
4     double ans=0;
5     for (int i=1;i<=n;i++){
6         ans+=(double)1/i;
7     }
8     cout<<fixed<<setprecision(10)<<ans<<'\n';
9 }
```

数组

数组

- 班上有 n ($1 \leq n \leq 10000$) 个同学，现在输入 n 以及 n 位同学的分数，求全班同学分数的平均值，并求分数大于平均值同学的人数。

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  signed main(){
5      int n;
6      cin>>n;
7      double sc[10010];
8      double sum;
9      for (int i=1;i<=n;i++){
10         cin>>sc[i];
11         sum+=sc[i];
12     }
13     double ave=sum/n;
14     cout<<ave<<"\n";
15     int ans=0;
16     for (int i=1;i<=n;i++)
17         if (sc[i]>ave) ans++;
18     cout<<ans<<"\n";
19     return 0;
20 }
```

数组

- 数组是相同类型变量的数列；
- 定义：<变量类型> 数组名[大小] //一维数组
- 例如：int a[100010]；
- 同理可定义二维、三位等多维向量；
- 例如：int a[110][110]；
- 数组可在定义时初始化；
- 例如：int a[3]={1,2,3}； int a[100010]={0}；
- 数组可通过下标直接访问
- 例如：int a[3]={1,2,3}； cout<<a[2]<<'\n'；

数组

- 注意：

- 数组的有效下标为 $0 \sim n-1$ ；
- 如： `int a[100]`；有效下标为 $0 \sim 99$ ，不包括100；
- 多维数组同理；
- 多维数组虽然是多维，但在计算机内存角度还是一维存储的；
- 如 `int a[3][3]`；在计算机内存角度是这样的：

<code>a[0][0]</code>	<code>a[0][1]</code>	<code>a[0][2]</code>	<code>a[1][0]</code>	<code>a[1][1]</code>	<code>a[1][2]</code>	<code>a[2][0]</code>	<code>a[2][1]</code>	<code>a[2][2]</code>
----------------------	----------------------	----------------------	----------------------	----------------------	----------------------	----------------------	----------------------	----------------------

- 多维数组也可通过类似方式初始化，如右图。

```
int a[3][3]={
    1,2,3,
    4,5,6,
    7,8,9
};
```

数组的本质

- 数组的本质其实是指针，在计算机内部为一段连续的地址空间；
- 如运行以下程序，输出的结果为：0x1086098 表示该数组存储开始位置的地址（不同电脑输出结果会有差异）。
- 若改成：cout<<*a<<'\n';输出为？
- 若改成：cout<<*(a+1)<<'\n';输出为？
- 故可通过 *(数组名+k) 的方式访问下标为k的元素。



```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  signed main(){
5      int a[10]={1,2,3,4,5,6,7,8,9};
6      cout<<a<<'\n';
7      return 0;
8  }
```


字符数组（字符串）

- C语言风格的字符串（字符数组）：`char s[10010];`
- C++引入的 `string` 类型：`string s;`
- 字符串一般用 “” 双引号引起来；
- `char` 本质是大小为 1 字节的整数，即 ASCII 码；

字符串：char 数组

- 用 char 类型的数组+**结尾** ‘\0’ 特殊字符表示字符串；
- ‘\0’表示字符串结尾，若没有则不会被识别为字符串，输出会出现问题；
- 可用 scanf(“%s”,s) 或 cin>>s；读入整个字符串；
- 若输入的为 “CzxLLL_25”，则存储为：

0	1	2	3	4	5	6	7	8	9
‘C’	‘z’	‘x’	‘L’	‘L’	‘L’	‘_’	‘2’	‘5’	‘\0’

- 同样可直接输出整个字符数组，结尾判定为 ‘\0’。

字符串：char 数组

- 注意：

- 一般数组大小可根据数据范围适当开大若干位，防止越界；
- char 数组当字符串使用时**赋初值或进行修改操作**时需要在结尾处加上 `'\0'`；
- 若使用 `scanf()` 读入，可通过指针操作选择字符串开始存储的下标，`cin` 不支持；
- 例如：`scanf("%s", s+1);` 表示该字符串从下标为 1 处开始存储；
- `scanf()` 和 `cin` 均会遇到空格和回车终止读入，若需要读完一整行则须使用 `getline(cin, s);`


字符串：string

- 需引入 `string.h` 或 `cstring` 头文件（若使用万能头则无需另外引入）；
- `string` 与 `char` 数组类似，同样可通过下标访问，但是结尾没有 `'\0'`；
- `string` 类变量无需给定大小；
- `string` 支持 `+` 运算，如 `"114"+"514"="114514"`，`"abc"+"d"="abcd"`；
- 本质是变量类型而非指针，故不能像 `char` 数组一样通过指针改变起始存储下标，但可通过在前面加空格的方式调整：`s=" "+s`；


字符串：string

- 注意：

- string 类型做 + 运算时尽可能使用 += 运算符而非 + 运算符，如按顺序将 'a'~'z' 添加到字符串 s 的末尾，左下写法效率高于右下写法；



```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  signed main(){
5      string s="";
6      for (char c='a';c<='z';c++)
7          s+=c;
8      return 0;
9  }
```



```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  signed main(){
5      string s="";
6      for (char c='a';c<='z';c++)
7          s=s+c;
8      return 0;
9  }
```


例题

- 输入一行字符串，字符串不包括回车，统计其中出现的数字个数。
- 样例输入：Today is 2023-10-15
- 样例输出：8
- 题目链接：<https://www.luogu.com.cn/problem/B2109>

函数

函数

- 可以说C++程序都是由函数构成的；
- 程序都必须有主函数main
- 函数可以有返回值，也可以没有返回值；
- 在函数中可以调用其他函数，也可以调用自身（递归）；
- 一个函数大概长成右图这样。



```
1 (返回值,没有为void) (参数) {  
2     ...  
3     return 返回值 (没有不  
4     用)  
5 }
```


1. 参数传递：传值与传引用

我们先看两个简单的函数



```
1 void F(int x) {  
2     x = 1;  
3     //返回类型为空时 return可以省略  
4 }  
5 int a = 0;  
6 F(a);  
7 cout << a << "\n";  
8 //结果输出0  
9
```



```
1 void F(int &x) {  
2     x = 1;  
3     //返回类型为空时 return可以省略  
4 }  
5 int a = 0;  
6 F(a);  
7 cout << a << "\n";  
8 //结果输出1  
9
```

怎么理解二者的差异？

在第一份代码中，a只是作为一个值传给了x，在函数中执行了 $x = 1$ ；但函数结束后x变量就销毁了，和a没有关系，所以a还是0。

在第二份代码中，x是作为a的一个引用，可以理解为此时的x就是a，在函数中执行了 $x = 1$ ；所以相当于 $a = 1$ 。

- 我们可以用传引用来实现一个两数交换的swap函数：

```
1 void Swap(int &x, int &y) {  
2     int t = x; //引入一个中间变量  
3     x = y;  
4     y = t;  
5 }  
6 void solve() {  
7     int a = 1, b = 2;  
8     Swap(a, b);  
9     cout << a << " " << b << "\n";  
10    //结果为 2 1  
11 }
```

- 再来看几个有返回值的函数：

```
1 double f(double x) {  
2     return x * x + x - 1; //相当于二次函数  $y = x^2 + x - 1$   
3 }
```

此函数的作用是返回二次函数x所对应的y

在传入数组时注意传入的是头指针，二者作用相同：

```
1 void f(int (&b)[]) {  
2     b[0]++;  
3 }  
4 void solve() {  
5     a[0] = 0;  
6     f(a);  
7     cout << a[0] << "\n";  
8 }
```

```
1 void f(int b[]) {  
2     b[0]++;  
3 }  
4 void solve() {  
5     a[0] = 0;  
6     f(a);  
7     cout << a[0] << "\n";  
8 }
```


2. 结构体中定义函数

- 在结构体中也可以定义函数，用 .函数名(参数) 就能调用。
- 结构体内的函数是被每个结构体变量单独享用的。

```
1 struct Info {  
2     int a, b, c;  
3     Info (int _a, int _b, int _c) {a = _a, b = _b, c = _c;}  
4     int f() {  
5         return a * b + c;  
6     }  
7 };  
8 Info A = Info(1, 2, 3), B = Info(0, 1, 2);  
9 cout << A.f() << " " << B.f() << "\n";  
10 //结果为5 2
```

3. 函数的自身调用

- 本节课只介绍最简单的递归，后面的课程会有专门的讲解。



```
1 int fac(int x) {  
2     if (x == 1) return 1;  
3     else return x * fac(x - 1);  
4     //作用是返回x的阶乘  
5 }
```

结构体

结构体

- 结构体(struct)可将若干变量整合成一个整体，形成一个新的数据类型，与类(class)相似；
- 比如说存储平面上的点 (double x, double y) 可利用结构体整合成一个 point 类型；
- 声明：

```
struct 结构体名{  
    类型 变量;  
};
```
- 使用：结构体名 变量名； 与定义普通变量相同；

结构体

- 如上述 point 类型结构体声明：

```
struct point{  
    double x,double y;  
}
```

- 定义：point p;
- 访问：p.x;p.y;即变量名后加 . 加内部数据变量名；

结构体

- 结构体内部可定义函数：

```
struct point{  
    double x,double y;  
    void inc(double a,double b){  
        x+=a,y+=b;  
    }  
};
```

- 调用： `point p;p.inc(a,b);` 与访问结构体内变量同理。

结构体

- 一个例子;
- 结构体类似其他数据类型, 可作为函数参数或者函数类型;

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  struct Vec{
5      double x,y;
6      void add(Vec a){
7          x+=a.x;
8          y+=a.y;
9      }
10     void show(){
11         cout<<'('<<x<<','<<y<<')'<<'\n';
12     }
13 };
14
15 double dot(Vec a,Vec b){
16     return a.x*b.x+a.y*b.y;
17 }
18
19 signed main(){
20     double x,y;
21     cin>>x>>y;
22     Vec p={x,y};           //按顺序给每个结构体内参数赋值
23     cin>>x>>y;
24     Vec q={x,y};
25     p.show(),q.show();
26     p.add(q),p.show();
27     double sum=dot(p,q);
28     cout<<sum<<'\n';
29     return 0;
30 }
```

时间复杂度简单分析

时间复杂度简单分析

- 时间复杂度，用于衡量算法的效率，算法竞赛中用于估计程序的运行时限预防超时(TLE)；
- 一般使用大 O 表示法来表示时间复杂度，初学者可简单地认为程序运行的次数，竞赛中一般考虑时间复杂度的上界；
- 例如右边这段代码，如何分析时间复杂度？
- 观察到两个循环分别循环 n 次和 m 次，显然这是该程序耗时的主要部分；

```
int n,m,sum;
int a[1010][1010];
for (int i=1;i<=n;i++)
    for (int j=1;j<=m;j++){
        cin>>a[i][j];
        sum+=a[i][j];
    }
```

时间复杂度简单分析

- 循环内部一共两条语句，粗略估计循环嵌套内运行次数约为 $2nm$ 次；
- 我们将前面的系数省略，则将该程序时间复杂度记作 $O(nm)$ ；
- 注意到我们将一些运算赋值以及定义语句当成一个单位“1”来分析；
- 同时我们只关注了运行次数较多的一段代码，忽略其他循环外的定义语句，这是分析时间复杂度常用的方法之一。

```
int n,m,sum;
int a[1010][1010];
for (int i=1;i<=n;i++)
    for (int j=1;j<=m;j++){
        cin>>a[i][j];
        sum+=a[i][j];
    }
```

时间复杂度简单分析

- 另一个例子;
- 观察到右侧有三个循环体;
- 三个循环体时间复杂度分别为 $O(nm)$, $O(k)$, $O(nmk\log k)$, 其中 `sort()` 函数时间复杂度为 $O(k\log k)$;
- 总时间复杂度为 $O(nm + k + nmk\log k)$, 一般我们取最高的时间复杂度, 则最终时间复杂度为 $O(nmk\log k)$;

```
int n,m,k,sum;
int a[1010][1010];
int b[100010];
for (int i=1;i<=n;i++)
    for (int j=1;j<=m;j++){
        cin>>a[i][j];
        sum+=a[i][j];
    }
for (int i=1;i<=k;i++) cin>>b[i];
for (int i=1;i<=n;i++)
    for (int j=1;j<=m;j++){
        sort(b,b+k);
    }
```

时间复杂度简单分析

- 常见的复杂度形式一般有：

$$O(1) < O(n) < O(\log n) < O(n \log n) < O(n^2) < O(2^n) < O(n!) < O(n^n)$$

- 一般认为前四个为比较优秀的时间复杂度，最后三个一般在 $n \leq 20$ 的时候出现；
- 根据具体的数据范围和时间限制选择合适的算法；
- 一般认为计算机 1s 的运行次数为 $2e8 \sim 3e8$ 次左右；
- 一般来说正确的算法所需时间复杂度会远小于给定时限；
- 多做题多读题解。

空间复杂度

- 同理对应的存在空间复杂度；
- 每个变量都需要一定大小的内存，一般题目为了防止提交者开过大的数组，会设置空间限制；
- 一般在空间限制为 256MB 时，认为 $2e7$ 大小的 *long long* 数组是可行的；
- 同样根据数据范围选择适当的数组大小。

例题

- <https://codeforces.com/problemset/problem/1821/A>
- 给定 $q (1 \leq q \leq 2 \cdot 10^4)$ 个询问，每个询问给定一个仅由数字和 ‘?’ 组成的字符串 $s (1 \leq |s| \leq 5)$ ，‘?’ 处可表示任意数字，问有多少个数字 x 能与该字符串匹配，要求 x 严格大于 0 且不含前导 0.
- 样例输入输出：

input	output
8	90
??	9
?	0
0	1
9	0
03	100
1??7	90
?5?	100
9??99	

例题

- 若 $s[0] = '0'$ 则方案数一定为 0;
- 不考虑前导 0, $'?'$ 贡献为 10 (0~9 每个数字都能取);
- 若 $s[0] = '?'$ 则该 $'?'$ 的贡献为 9(1~9);
- 故答案为 $10^k \times (s[0] == '?' ? 9 : 1)$;
- 时间复杂度为 $O(\sum |s|)$ 。

参考程序

- `#define int long long` 虽然方便写代码，但可能导致常数过大，谨慎使用；
- `ios::sync_with_stdio(false);` 解绑流输入输出，可以让 `cin/cout` 读入速度与 `scanf()/printf()` 相当；
- (主)函数内部定义的变量记得赋初值；

```
1  #include <bits/stdc++.h>
2  #define int long long
3
4  using namespace std;
5
6  signed main(){
7      ios::sync_with_stdio(false);
8      int T=1;
9      cin>>T;
10     while (T--){
11         string s;
12         cin>>s;
13         if (s[0]=='0'){
14             cout<<"0\n";
15             continue;
16         }
17         int ans=1;
18         if (s[0]=='?') ans=9;
19         for (int i=1;i<s.size();i++){
20             if (s[i]=='?') ans=ans*10;
21         }
22         cout<<ans<<'\\n';
23     }
24     return 0;
25 }
```