

《计算机语言与程序设计》

第4周 循环结构、数组

清华大学 自动化系
范 静 涛

1. 三种循环语句: **while** 循环

一般形式:

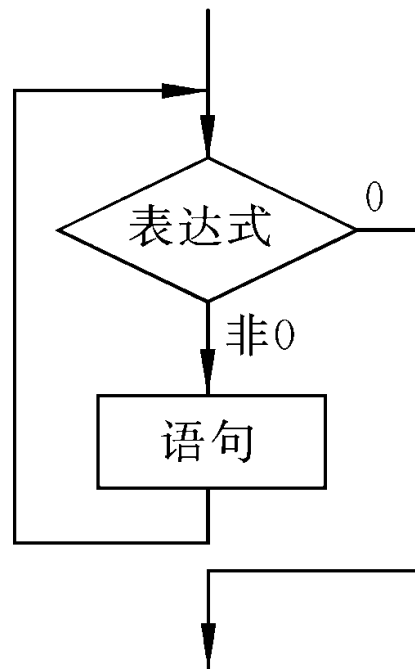
```
while (condition) {  
    statements  
}
```

根据循环条件执行循环语句:

表达式condition求值, 并转为_Bool值;

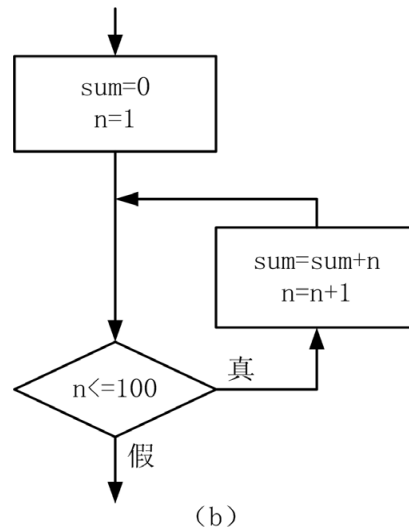
condition为真, 则执行statements, 重复步骤①;

condition为假, 执行后续语句;



1.三种循环语句: **while**循环

```
#include <stdio.h>
int main(int argc, char* argv[])
{
    int n = 1;
    int sum = 0;
    //循环直到n大于100
    while (n <= 100)
    {
        sum += n; //累加和
        n++;
    }
    printf("sum is %d\n", sum);
    return 0;
}
```



注意:

- (表达式)后边没有分号!
- 不管循环体包含几条语句,均应该用花括弧括起来,以复合语句形式出现。
- 在循环体中应有使循环趋向于结束的语句。如果无此语句,则n的值始终不改变,循环永不结束——死循环

1.三种循环语句: **while**循环

```
#include <stdio.h>
int main(int argc, char* argv[])
{
    int n = 1;
    int sum = 0;
    //循环直到n大于100
    while (n <= 100)
    {
        sum += n; //累加和
        n++;
    }
    printf("sum is %d\n", sum);
    return 0;
}
```

循环初始:
发生在**循环之前**, 使得循环“就绪”

循环条件:
循环得以继续或终止的判定;

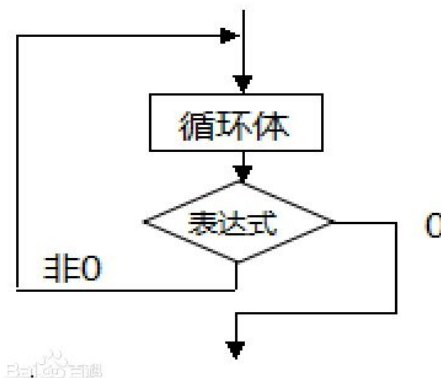
循环控制:
在**循环内部**控制循环条件的关键过程。

循环结构需精确设计**三要素**: **循环初始**、**循环条件** 和 **循环控制**。

1.三种循环语句:do-while循环

一般形式:

```
do {  
    statements  
} while (condition);
```



根据循环条件执行循环语句:

执行statements

表达式condition求值, 并转为_Bool值;

condition为真, 则执行statements, 重复步骤①;

condition为假, 执行后续语句;

1.三种循环语句: do-while循环

```
#include <stdio.h>
int main(int argc, char* argv[])
{
    int x;
    scanf("%d", &x);
    int digit;
    int ret = 0;

    if (x < 0) {
        printf("please input an positive number.\n");
    }
    else {
        do {
            digit = x % 10;
            ret = ret * 10 + digit;
            x /= 10;
        } while (x > 0);
        printf("%d", ret);
    }
    return 0;
}
```

注意此处:
while()后必有分号

1. 三种循环语句: **while**循环 vs. **do-while**循环

在**一般情况下**, 用**while**语句和用**do-while**语句处理同一问题时, 若二者的循环体部分是一样的结果一样么? (初始值相同? 循环次数相同?)

1. 三种循环语句: **for**循环

一般形式:

```
for (initialization; condition; increment) {  
    statements  
}
```

求解initialization。

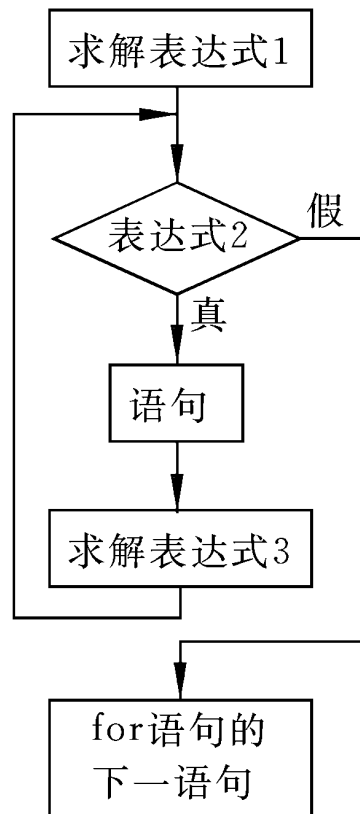
求解condition, 若其值为真, 则执行statements
, 然后执行下面第③步。若为假, 则结束循环, 转到第⑤步。

求解increment。

转回上面第②步骤继续执行。

循环结束, 执行for语句下面的一个语句

C语言中的for循环语句使用最为灵活, 可以取代while和do-while



1. 三种循环语句: **for**循环

for语句最简单的形式:

for(循环变量赋初值; 循环条件; 循环变量增值)

```
int Sum = 0;
for (int i = 1; i <= 100; i++) {
    Sum += i;
}
```

```
int Sum = 0;
int i = 1;
for(;;) {
    sum += i;
    i++;
}
```

说明:

(1) 3

个表达式都可省略（但不建议），即：不设初值，不判断条件(默认为真)，循环变量不增值。无终止地执行循环体。

```
for (i = 0, Sum = 0; i <= 100; i++, Sum +=i) {
    ;
}
```

```
for (i = 1, Sum = 0; i <= 100; Sum +=i, i++) {
    ;
}
```

(2) 表达式1和3

可以是与循环变量无关的其他表达式，也可以是逗号表达式

最好不要把与循环控制无关的内容放到for语句中

```
for (i = 0, j = 100, Sum = 0; i < j; i++, j--) {
    Sum += (i + j);
}
if (i == j) {
    Sum += i;
}
```

1. 三种循环语句：几种循环的比较

三种循环都可以用来处理同一问题，一般情况下它们可以互相代替。

for循环紧凑简练、功能强大，凡用while循环能完成的，用for循环都能实现。

关于循环中止条件

- ❓ 在while循环和do-while循环中，只在while后面的括号内指定循环条件，因此为了使循环能正常结束，应在循环体中包含使循环趋于结束的语句(如i++等)。
- ❓ for循环可以在表达式3中包含使循环趋于结束的操作。

1. 三种循环语句：几种循环的比较

	while	do-while	for
设置循环初始值	语句前	语句前	语句前或表达式1
结束条件的判断	先判断后执行	先执行后判断	先判断后执行
循环控制的计数	循环体内	循环体内	表达式3或循环体内

关于初始化

- ❓ 用**while**和**do-while**循环时，循环变量初始化的操作应在**while**和**do-while**语句之前完成。
- ❓ **for**语句可以在表达式1中实现循环变量的初始化。

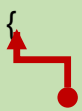
关于循环中止

- ❓ **while**循环、**do-while**循环和**for**循环，均可以用**break**语句跳出循环，用**continue**语句结束本次循环


1. 循环的加速与中止: **continue** vs. **break**

continue加速，直接进入下次循环

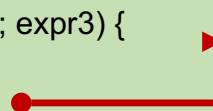
```
while (expr1) {  
    ...  
    continue;  
    ...  
}
```



```
do {  
    ...  
    continue;  
    ...  
} while(expr1);
```




```
for (expr1; expr2; expr3) {  
    ...  
    continue;  
    ...  
}
```




break终止，执行循环结构的后继语句


```
while (expr1) {  
    ...  
    break;  
    ...  
}  
...
```



```
do {  
    ...  
    break;  
    ...  
} while(expr1);  
...
```



```
for (expr1; expr2; expr3) {  
    ...  
    break;  
    ...  
}  
...
```



1. 循环的加速与中止: **continue** vs. **break**

```
#include <stdio.h>
int main(int argc, char* argv[]) {
    for (int n = 100; n <= 200; n++){
        if (n % 3 != 0) {
            continue;
        }
        printf("%d\n", n);
    }
    return 0;
}
```

加速，直接进入下次循环

1. 循环的终止: **break**语句

```
#include <stdio.h>
int main(int argc, char* argv[]) {
    int m;
    scanf("%d", &m);

    //从2到m-1之间逐一检查是否被m整除
    for (int i = 2; i <= m - 1; i++) {
        //如果整除则结束检查
        if (m % i == 0) {
            break;
        }
    }
    //根据循环结束位置判断是否素数
    if (i == m) {
        printf("Yes\n");
    }
    else {
        printf("No\n");
    }
    return 0;
}
```

提前结束循环

OR

```
#include <stdio.h>
int main(int argc, char* argv[]) {
    int m;
    scanf("%d", &m);
    int i = 0;
    //从2到m-1之间逐一检查是否被m整除
    for (i = 2; i <= m - 1; i++) {
        //如果整除则结束检查
        if (m % i == 0) {
            break;
        }
    }
    //根据循环结束位置判断是否素数
    if (i == m) {
        printf("Yes\n");
    }
    else {
        printf("No\n");
    }
    return 0;
}
```

在**for**里定义循环变量，在循环结束后，对其访问，右侧方法是通用的

1. 循环的嵌套

一个循环体内可以包含另一个完整的循环结构，形成嵌套循环

C语言中循环语句（while、do-while、for）可以互相嵌套，形成多重循环，循环嵌套的层数没有限制

```
while()
{
    ...
    while()
    {
        ...
    }
    ...
}
```

```
do
{
    ...
    do
    {
        ...
    } while();
    ...
} while();
```

```
for(;;)
{
    ...
    for(;;)
    {
        ...
    }
    ...
}
```

```
for(;;)
{
    while()
    {
        ...
    }
    ...
    for(;;)
    {
        do
        {
            ...
        } while();
        ...
    }
    ...
}
```

1.循环的嵌套

```
#include <stdio.h>
int main(int argc, char* argv[]) {
    for (int Row = 1; Row <= 10; Row++) {
        //每行先输出若干空格：与行相关
        for (int Space = 1; Space <= 10 - Row; Space++) {
            printf(" ");
        }
        //控制每行的*的个数：与行相关
        for (int asterisk = 1; asterisk <= 2 * Row - 1; asterisk++) {
            printf("*");
        }
        //每行末尾输出1个换行
        printf("\n");
    }
    return 0;
}
```

```

      *
    ***
  *****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****

```


1. 循环程序举例

整数逆序

```
#include <stdio.h>
int main(int argc, char* argv[]) {
    int x = 0;
    int digit = 0;
    int ret = 0;
    scanf("%d", &x);
    while(x > 0) {
        digit = x % 10;
        printf("%d", digit);
        ret = ret * 10 + digit;
        //printf("x=%d,digit=%d,ret=%d\n", x, digit, ret);
        x /= 10;
    }
    // printf("\n%d", ret);
    return 0;
}
```

- 个位数正确么？
- **x=0**结果正确么？
- **x = 10, 100**正确么？
- **x**是负数？

调试技巧也要掌握和积累

1. 循环程序举例

求100以内的全部素数，每行打印5个数字。

```
#include <stdio.h>
#include <math.h>
int main(int argc, char* argv[]) {
    int Counter = 0;
    for (int Num = 3; Num < 100; Num += 2) {
        int SqrtOfNum = sqrt(Num);
        int i;
        for (i = 3; i <= SqrtOfNum; i++) {
            if (Num % i == 0) {
                break;
            }
        }
        if (i >= SqrtOfNum + 1) {
            printf("%d ", Num);
            Counter++;
            if (Counter % 5 == 0) {
                printf("\n");
            }
        }
    }
    printf("\n");
    return 0;
}
```

Num += 2, 跳过偶数, 减少循环次数

SqrtOfNum = sqrt(Num)

放在循环外, 减少循环次数

1. 循环程序举例

如何同时用1角、2角和5角硬币凑出10元以下整数金额？

嵌套顺序？

```
#include <stdio.h>
#include <math.h>
int main(int argc, char* argv[]) {
    unsigned int x = 0;
    _Bool IsExit = 0;
    scanf("%u", &x);
    for (unsigned int CountOfOne = 1; CountOfOne < x * 10; CountOfOne++) {
        for (unsigned int CountOfTwo = 1; CountOfTwo < x * 10 / 2; CountOfTwo++) {
            for (unsigned int CountOfFive = 1; CountOfFive < x * 10 / 5; CountOfFive++) {
                if (CountOfOne + CountOfTwo * 2 + CountOfFive * 5 == x * 10) {
                    printf("%d元: %d个1角, %d个2角, %d个5角\n", x, CountOfOne, CountOfTwo, CountOfFive);
                    IsExit = 1;
                    break;
                }
            }
        }
        if (IsExit == 1) {
            break;
        }
    }
    if (IsExit == 1) {
        break;
    }
    return 0;
}
```

1. 循环程序举例

求Fibonacci数列前40个数。这个数列有如下特点：

- ❑ 第1,2两个数为1,1。
- ❑ 从第3个数开始，该数是其前面两个数之和。即：

$$F(1)=1 \quad (n=1)$$

$$F(2)=1 \quad (n=2)$$

$$F(n)=F(n-1)+F(n-2) \quad (n \geq 3)$$

请每行打印四个数。

1. 循环程序举例

```
#include <stdio.h>
int main(int argc, char* argv[]) {
    unsigned long f1 = 1;
    unsigned long f2 = 2;
    for (int i = 1; i <= 20; i++) {
        printf("%12lu%12lu", f1, f2);
        if (i % 2 == 0) {
            printf("\n");
        }
        f1 += f2;
        f2 += f1;
    }
    return 0;
}
```

2.什么是数组?

一个简单的问题，用程序实现如下功能：连续输入100个数，然后反序输出。

- 如何定义变量?
- 如何使用变量?

为了处理批量数据，C语言把若干具有**相同数据类型**的数据**有序组织**起来，形成一种组合数据类型，称为数组。

这些数据可以用一个名字来表示，并通过循环逐个访问和处理。

数组可以是一维、二维、甚至多维。

2.1 一维数组：声明

一维数组的声明格式为：

类型说明符 数组名 [整形表达式];

例如：

```
const int N = 20;  
int Array1[10];  
int Array2[5-1];  
int Array3[N]; //正确，长度是整型常量、符号常量或常量表达式  
int Array5[59.5]; //错误，长度非整型  
int m = 10;  
int Array6[m]; //正确，但变长数组(VLA, 仅C99支持)，只能声明在函数内部。
```

说明：

- 数组名定名规则和变量名相同，遵循标识符定名规则。
- 方括号中的**表达式**用来表示数组长度，必须是**正整数**。
- 数组长度必须在定义时指定，且一经定义长度不可改变。
- 常量表达式中可以包括常量和符号常量，但不能包含变量（因为编译时需要分配合适的内存）。

2.1 一维数组：存储

C语言规定数组元素在内存中连续存放

- ❓ 数组占用得内存空间是所有数据元素占用的字节数的总和
- ❓ 每个数据元素占用空间就是基类型的字节数

例如: `unsigned short int score[77];`

低地址

86
90
71
52
...
84

高地址

说明：

可以将一维数组看作是内存中一个"很大的变量"，简称块（block），数组名就是这个块的名字。

2.1 一维数组: 引用

C语言是通过数组名加相对偏移来索引元素，一般形式为

数组名[下标]

这里方括号 ([]) 为下标引用运算符

例如: $a[0] = a[5] + a[7] - a[2 * 3];$

说明:

- 下标必须是**整型**
- 下标**从0开始, 不能越界** (编译器不做检查)。
- 只能引用单个数组元素, 不可以批量引用。
- 定义数组时用到的方括号与引用数组元素时用到的方括号含义不同。
- 整个数组不允许进行赋值运算、算术运算等操作, 只有元素才可以。

2.1 一维数组: 引用

输入10个整数，逆序输出

```
#include <stdio.h>
int main(int argc, char* argv[]) {
    const int N = 10;
    int i = 0;
    int Array[N];
    for (i = 0; i < N; i++) {
        scanf("%d", &Array[i]);
    }
    for(i = N - 1; i >= 0; i--) {
        printf("%d ", Array[i]);
    }
    printf("\n");
    return 0;
}
```

2.1 一维数组: 定义/初始化

在定义数组时对数组元素赋以初值（VLA不能初始化）。

给定元素个数，部分/全部初始化元素

```
int Array[5] = {1, 2, 3};
```

1	2	3	0	0
---	---	---	---	---

```
int Array[5] = {1, 2, 3, 4, 5};
```

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

- 初值列表的大括号" {} "是必需的;
- 初值按一维数组内存形式中的元素**排列顺序一一对应初始化**
- 初值列表提供的元素个数**不能超过数组长度**
，但可以小于数组长度。如果初值个数小于数组长度，则只初始化前面的数组元素，**剩余元素初始化为0**。

2.1 一维数组: 定义/初始化

在定义数组时对数组元素赋以初值（VLA不能初始化）。

不给定元素个数，全部初始化元素

```
int Array[] = {1, 2, 3};
```

1	2	3
---	---	---

```
int Array[] = {1, 2, 3, 4, 5};
```

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

- 初值列表的大括号"{}"是必需的;
- 在提供了初值列表的前提下，数组定义时不用指定数组长度，编译器会根据初值个数自动确定数组的长度。

2.1 一维数组：定义/初始化

在定义数组时对数组元素赋以初值（VLA不能初始化）。

全部元素初始化为0的特殊操作

```
int Array[5] = {0};
```

0	0	0	0	0
---	---	---	---	---

不初始化数组，则不能确定元素数值，不一定是0

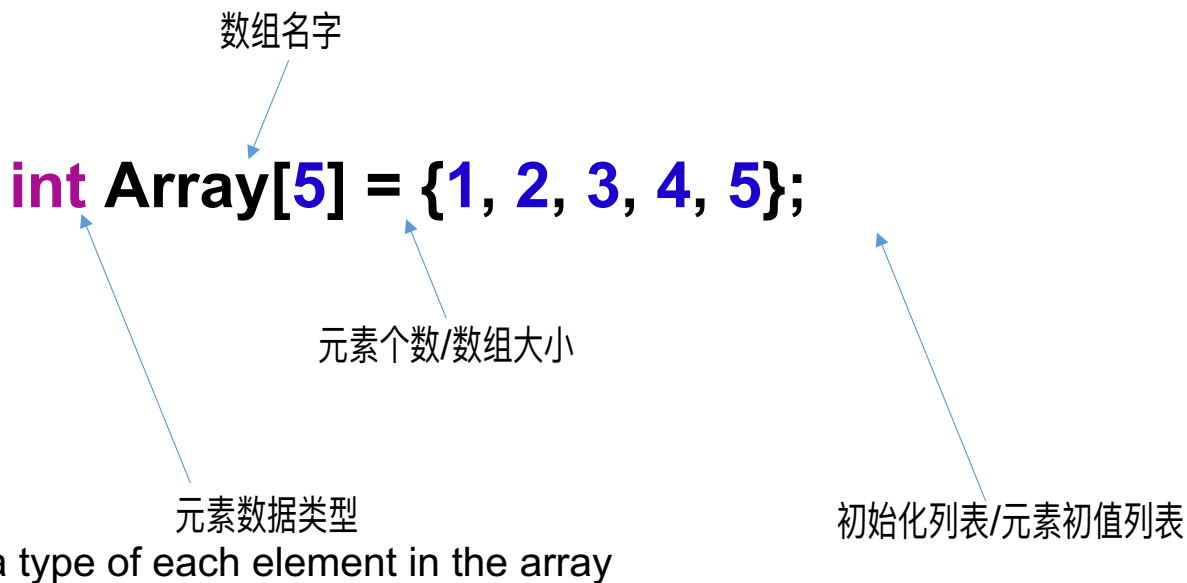
```
int Array[5];
```

?	?	?	?	?
---	---	---	---	---

- 不使用列表进行初始化时，**元素初始值为脏数据**。
- 具体初始值与**定义所在位置**和**存储类型**有关，待续

2.1 一维数组: 定义/初始化规律总结

数组，是不是数据类型？必须不是



数组是连续存放的若干相同类型的变量/常量的集合

2.1 一维数组：定义/初始化规律总结

```
int Array[5] = {1, 2, 3, 4, 5};
```

方括号内指定N个元素

初始化列表M个元素

N和M	元素实际个数	元素初值
无 / M	M	初始化列表指定全部初值
N / 无	N	脏数据
无 / 无		错误
N == M	N	初始化列表指定全部初值
N > M	N	初始化列表指定前M个初值，其余为0

2.1 一维数组：程序实例

用起泡法对5个同学的身高排序(由高到更高)。



2.1 一维数组：程序实例

用起泡法对5个同学的身高排序(由高到更高)。



2.1 一维数组：程序实例

用起泡法对5个同学的身高排序(由高到更高)。



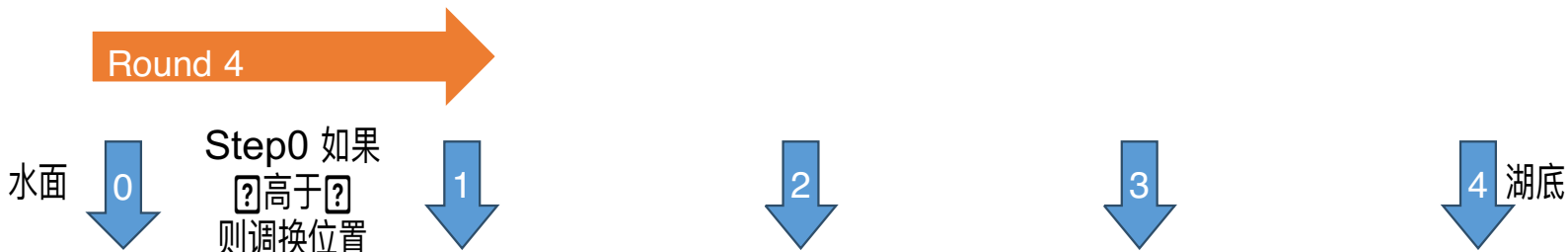
2.1 一维数组：程序实例

用起泡法对5个同学的身高排序(由高到更高)。



2.1 一维数组：程序实例

用起泡法对5个同学的身高排序(由高到更高)。



2.1 一维数组：程序实例

```
#include <stdio.h>
int main(int argc, char* argv[]) {
    const unsigned int Count = 5u;
    unsigned int Height[Count] = {0u};
    unsigned int Round = 0u;
    unsigned int Step = 0u;
    unsigned int Temp = 0u;
    //input-loop
    for (Step = 0u; Step < Count; Step++){
        scanf("%u", &Height[Step]);
    }//end of input-loop

    for (Round = 1u; Round < Count; Round++) {
        for (Step = 0u; Step < Count - Round; Step++) {
            if (Height[Step] > Height[Step + 1u]) {
                Temp = Height[Step];
                Height[Step] = Height[Step + 1u];
                Height[Step + 1u] = Temp;
            }
        }//end of Step-loop
    }//end of Round-loop
    return 0;
}
```

2.1 一维数组：程序实例

用起泡法对Count个数值进行排序

需要多少Round?

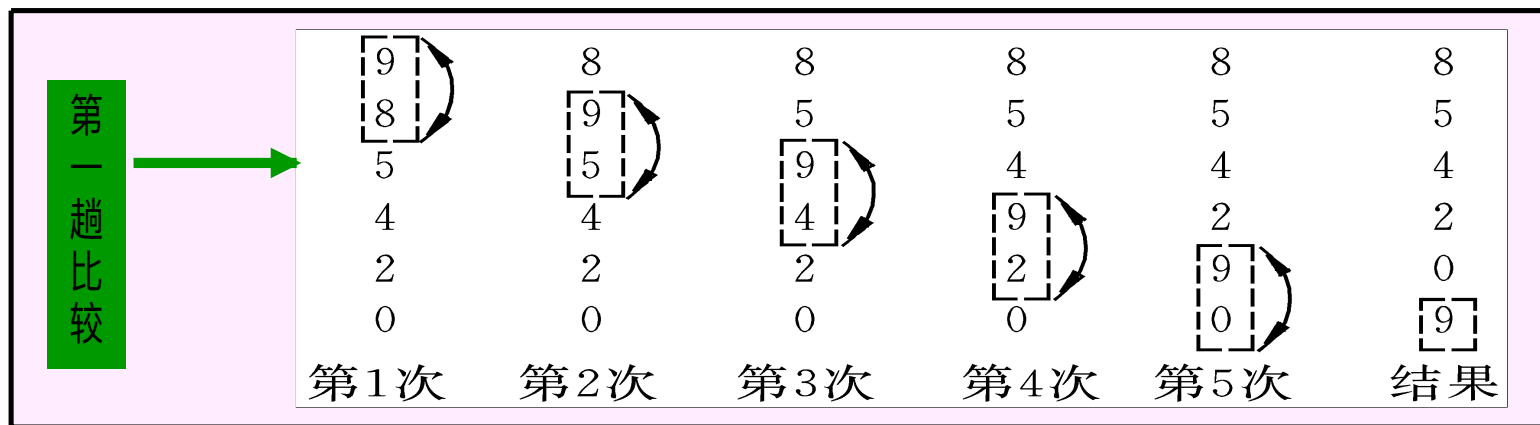
Count – 1个Round

每个Round需要多少次比较 (Step) ?

**每个Round, 需要Count – Round 次比较
(Round从1开始)**

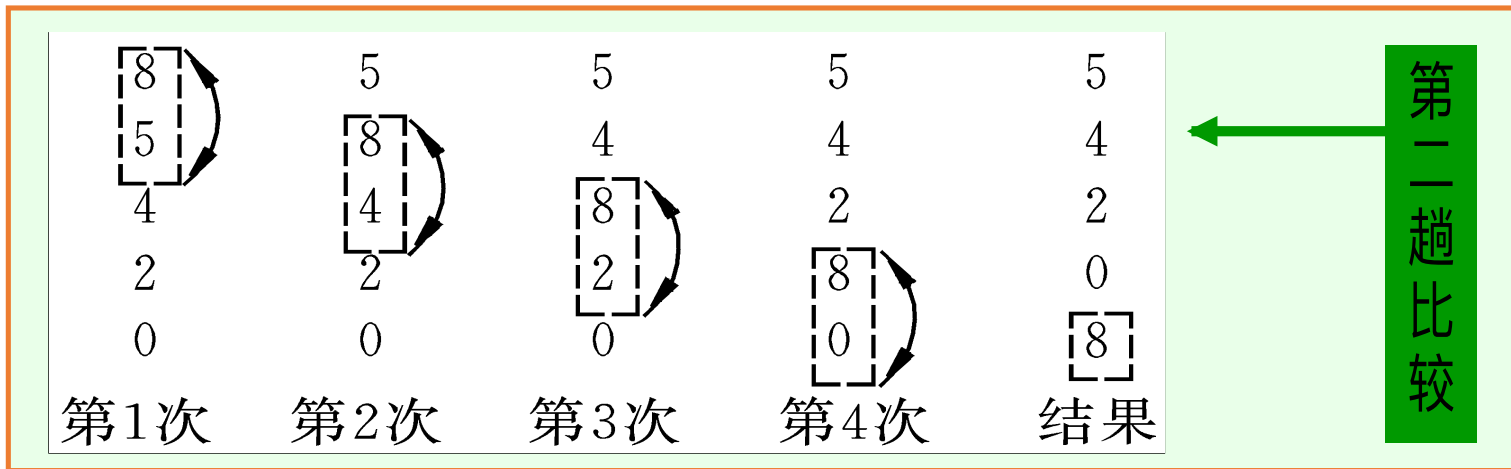
2.1 一维数组：程序实例

用起泡法对10个数排序(由小到大)。



经过第一趟(共5次比较与交换)后, 最大的数9已"沉底"。然后进行对余下的前面5个数第二趟比较,

2.1 一维数组：程序实例



2.1 一维数组：程序实例

```
#include <stdio.h>
int main(int argc, char* argv[]) {
    const unsigned int Count = 5u;
    unsigned int Height[Count] = {0u};
    unsigned int Round = 0u;
    unsigned int Step = 0u;
    unsigned int Temp = 0u;
    //input-loop
    for (Step = 0u; Step < Count; Step++){
        scanf("%u", &Height[Step]);
    }//end of input-loop

    for (Round = 1u; Round < Count; Round++) {
        for (Step = 0u; Step < Count - Round; Step++) {
            if (Height[Step] > Height[Step + 1u]) {
                Temp = Height[Step];
                Height[Step] = Height[Step + 1u];
                Height[Step + 1u] = Temp;
            }
        }//end of Step-loop
    }//end of Round-loop
    return 0;
}
```

Count-1 ↑Round

**每个Round, 需要Count-Round次比较
(Round从1开始)**

2.2 二维数组：声明

二维数组的声明格式为：

类型说明符 数组名[表达式][表达式];

例如： `int Array1[3][4];` //不可以写成`Array1[3,4]`
`int Array2[2][9];` //不可以写成`Array[2, 9]`

说明：

- 二维数组的元素类型、数组名和常量表达式的含义和要求完全与一维数组类似。
- 这里用方括号 ([]) 对应了维数，约定多维数组越往左称为"高维"，越往右称为"低维"。

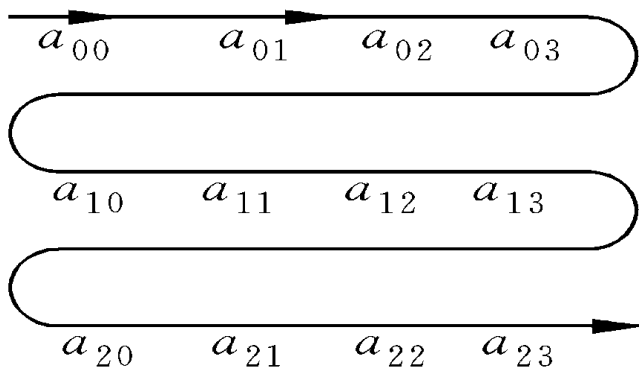
[]数量 等于 维数

所有表达式乘积 等于 元素数

2.2 二维数组：存放

二维数组在内存中线性排列，排列顺序是**按行存放**，即先顺序存放第一行的元素，再存放第二行的元素……

例如， $a[3][4]$ 数组的存放的顺序



2.2 二维数组：存放

我们可以把二维数组看作是一种特殊的一维数组：它的元素又是一个一维数组。

例如：可以把 a 看作是一个包含3个元素的一维数组，元素为 $a[0]$, $a[1]$, $a[2]$ ，每个元素是一个包含4个元素的一维数组，比如 $a[0]$ 包含 $a[0][0]$, $a[0][1]$, $a[0][2]$, $a[0][3]$ 。

$$a \left[\begin{array}{l} a[0] \text{ ----- } a_{00} \quad a_{01} \quad a_{02} \quad a_{03} \\ a[1] \text{ ----- } a_{10} \quad a_{11} \quad a_{12} \quad a_{13} \\ a[2] \text{ ----- } a_{20} \quad a_{21} \quad a_{22} \quad a_{23} \end{array} \right.$$

注意：

$a[0]$ 的下一个是 $a[1]$, $a[1]$ 的下一个是 $a[2]$ ，其余依次类推；

$a[0][0]$ 的下一个 $a[0][1]$, $a[0][3]$ 的下一个是 $a[1][0]$ ，其余依次类推

后续指针
运算会用到!

2.2 二维数组：存放

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

初始化稍后讲

地址	值	数组元素
3000H	1	a[0][0]
3004H	2	a[0][1]
3008H	3	a[0][2]
300CH	4	a[1][0]
3010H	5	a[1][1]
3014H	6	a[1][2]
3018H	7	a[2][0]
301CH	8	a[2][1]
3020H	9	a[2][2]

2.2 二维数组: 引用

二维数组元素的表示形式为:

数组名[下标1][下标2]

说明:

- 下标可以是整型常量、变量或表达式, 如 $a[2 - 1][2 * 2 - 1]$ 。
- 两个下标各自独立索引对应维的元素
- 数组元素可以出现在表达式中, 也可以被赋值

例如: $b[1][2] = a[1][2] * a[2][3];$

- 在使用数组元素时, 应该注意下标值应在已定义的数组大小的范围内。

```
int a[3][4]; /* 定义a为3×4的数组 */
```

```
...
```

```
a[3][4] = 3;
```



2.2 二维数组：定义/初始化

按照多维数据形式给二维数组赋初值

给定行数，部分/全部行初始化（行内也可以是部分）

```
int Array[3][4] = {  
    {1, 2, 3, 4},  
    {5, 6, 7, 8},  
    {9, 10, 11, 12}  
};
```

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

未定给的行或行内为给定的初值的元素，初值均为0

2.2 二维数组：定义/初始化

按照多维数据形式给二维数组赋初值

未给定行数，全部行初始化（行内也可以是部分）

```
int Array[][4] = {  
    {1, 2, 3, 4},  
    {},  
    {9, 10, 11},  
};
```

行数自动确定为3
列数不能省略

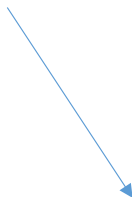
2.2 二维数组：定义/初始化

按照一维数据形式给二维数组赋初值

行数无所谓，全部/部分元素初始化

```
int Array[][4] = {1, 2, 3, 4, 5};
```

行数自动确定为2
列数不能省略

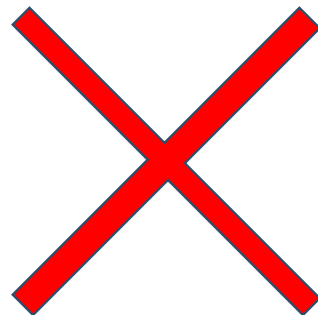


给定的元素数 % 列数 == 0 ? 给定的元素数 / 列数 : 给定的元素数 / 列数 + 1

2.2 二维数组：程序举例

将一个方阵二维数组行和列元素互换

```
int main(int argc, char* argv[]) {  
    int Array[4][4] = {  
        {1, 2, 3, 4},  
        {5, 6, 7, 8},  
        {9, 10, 11, 12},  
        {13, 14, 15, 16}  
    };  
    int Temp = 0;  
    for (int Row = 0; Row < 4; Row++) {  
        for (int Col = 0; Col < 4; Col++) {  
            Temp = Array[Row][Col];  
            Array[Row][Col] = Array[Col][Row];  
            Array[Col][Row] = Temp;  
        } //end of Col-loop  
    } //end of Row-loop  
    return 0;  
}
```



2.2 二维数组：程序举例

将一个方阵二维数组行和列元素互换

```
int main(int argc, char* argv[]) {  
    int Array[4][4] = {  
        {1, 2, 3, 4},  
        {5, 6, 7, 8},  
        {9, 10, 11, 12},  
        {13, 14, 15, 16}  
    };  
    int Temp;  
    for (int Row = 0; Row < 4; Row++) {  
        for (int Col = Row + 1; Col < 4; Col++) {  
            Temp = Array[Row][Col];  
            Array[Row][Col] = Array[Col][Row];  
            Array[Col][Row] = Temp;  
        } //end of Col-loop  
    } //end of Row-loop  
    return 0;  
}
```



2.2 二维数组：程序举例

有一个 3×4

的矩阵，要求编程序求出其中值最大的那个元素的值，以及其所在的行号和列

```
int main(int argc, char* argv[]) {  
    int Array[3][4] = {  
        {1, 2, 3, 4},  
        {5, 6, 7, 8},  
        {9, 10, 11, 12}  
    };  
    int Max = Array[0][0];  
    int Row = 0;  
    int Col = 0;  
    for (int i = 0; i < 3; i++) {  
        for (int j = 0; j < 4; j++) {  
            if (Array[i][j] > Max) {  
                Max = Array[i][j];  
                Row = i;  
                Col = j;  
            }  
        }  
    }  
    //end of j-loop  
    //end of i-loop  
    return 0;  
}
```

2.2 二维数组：程序举例

计算4*4方阵右上三角元素的和

```
int main(int argc, char* argv[]) {  
    const int Size = 4;  
    int Array[Size][Size] = {  
        {1, 2, 3, 4},  
        {5, 6, 7, 8},  
        {9, 10, 11, 12},  
        {13, 14, 15, 16},  
    };  
    int Sum = 0;  
    for (int Row = 0; Row < Size; Row++) {  
        for (int Col = Row; Col < Size; Col++) {  
            Sum += Array[Row][Col];  
        } //end of Col-loop  
    } //end of Row-loop  
    return 0;  
}
```

定义常量，移植性、扩展性、可读性增强

注意：大量的数据输入如何调试？用好txt和Ctrl+V。

2.3 数组：降维等价性

数组内存寻址方式：基址变址

一维数组： $DataType\ Array[N]$, 则 $Array[Index]$ 的地址: 为 $Array$ 中第一个元素地址 $+ Index * sizeof(DataType)$;

二维数组： $DataType\ Array[N][M]$, 则 $Array[Idx1][Idx2]$ 的地址为: $Array$ 中第一个元素地址 $+ Idx1 * M * sizeof(DataType) + Idx2 * sizeof(DataType)$

数组的首地址为基址.

元素类型和索引值带权累加和为变址，各维度元素个数就是权。

2.3 数组：降维等价性

如果将一个二维数组**Array2D[N][M]**当做一维数组**Array1D**看，则**Array1D**应有 **$[N * M]$** 个元素

Array2D[Idx1][Idx2]代表着**Array1D**中的 **$[Idx1 * M + Idx2]$**

Array1D中的**[Idx]**代表着**Array2D[Idx / M][Idx % M]**

我们物理上可以用一维数组的实际存储来表示二维数组，甚至更高维的数组

2.4 字符数组: 定义

数组的基类型可以是字符，即字符数组

char 字符数组名[常量表达式]

文本信息用途非常广，如姓名、通信地址、邮箱、学号等。字符数组及衍生出来的"字符串"使得程序能方便地表示文本信息。

例如: **char** c[10];

c[0]='I';

c[1]=' ';

c[2]='a';

c[3]='m';

c[4]=' ';

c[5]='h';

c[6]='a';

c[7]='p';

c[8]='p';

c[9]='y';

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]	c[9]
I		a	m		h	a	p	p	y

2.4 字符数组：初始化

对字符数组初始化，可逐个字符赋给数组中各元素。

例如：char c[10]={ 'l', 'a', 'm', 'h', 'a', 'p', 'p', 'y' }

如果初值个数小于数组长度，则只将这些字符赋给数组中前面那些元素，其余的元素自动定为空字符。

例如：char c[10]={ 'c', ' ', 'p', 'r', 'o', 'g', 'r', 'a', 'm' };

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]	c[9]
c		p	r	o	g	r	a	m	\0

注意：是空字符，不是空格字符，也不是‘0’!!!

2.4 字符数组：初始化

由于初值列表的字符通常很多，因此经常不给长度值，字符数组的长度由编译器自动确定

例如，`char s[]={'H','e','l','l','o',' ','W','o','r','l','d'};`

s

H	e	l	l	o	␣	W	o	r	l	d
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]

实际上数据应是字符的ASCII值，形式如下：

s

72	101	108	108	111	32	87	111	114	108	100
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]

2.4 字符数组：初始化

字符数组在使用时，同样只能逐个引用字符元素的值而不能一次引用整个字符数组对象，如不能进行赋值、算术运算等

例如：`char s1[5]={'B','a','s','i','c'} , s2[5];`
`s2 = s1;` //错误，数组不能赋值
`s2[0] = s1[0];` //正确，数组元素赋值

2.4 字符数组：二维字符数组

定义和初始化一个二维字符数组

```
char diamond[5][5]=  
{  
    { ' ' ' ' ' ' '*'},  
    { ' ' '* ' ' '*'},  
    { '* ' ' ' ' '*'},  
    { ' ' '* ' ' '*'},  
    { ' ' ' ' '* ' ' }  
}
```

```
      *  
    *  *  
  *    *  
 *      *  
  *    *  
    *  *  
      *
```

```
char A[3][20]={{"C"}, {"C++"}, {"Data Structure"}};
```

```
char A[3][20]={"C", "C++", "Data Structure"};
```

```
char A[][20]={"C", "C++", "Data Structure"};
```

2.4 字符数组: 引用

输入不多于100个字符，输出期中不是*的字符

```
#include <stdio.h>
int main(int argc, char* argv[]) {
    char s[100];
    int Count = 0;
    //连续输入多个字符，直到回车'\n'为止
    while ((s[Count] = getchar()) != '\n') {
        Count++;
    }
    for (int idx = 0; idx < Count; idx++) {
        if (s[idx] != '*') { //过滤'*'字符
            printf("%c", s[i]);
        }
    }
    return 0;
}
```

注意:

字符数组存储的实际字符个数未必总是数组长度，其余空间存储值 不确定，因此就要始终记录实际个数。

2.4 字符串：结束标志

语言规定字符串是以'\0' (ASCII值为0) 字符作为结束符的字符数组

通过判断数组元素是否为空字符来判断字符串是否结束

，无需记录字符实际长度

❓ 字符数组并不要求它的最后一个字符为'\0'，甚至可以不包含'\0'，

❓ 例如：`char c[5]={'C','h','i','n','a'};`
为了与字符串处理方法一致，在字符数组中也常人为地加上一个'\0'，方便在字符数组中对于字符串的引用

例如：`char c[6]={'C','h','i','n','a','\0'};`

2.4 字符串：结束标志

字符串长度是指在第1个空字符之前的字符个数（不包括空字符）

字符串常量是以**一对双引号括起来的字符序列**。C

语言在编译时为字符串常量自动在其后增加一个空字符

例如："Hello"的存储形式为：

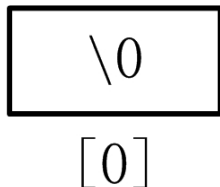
H	e	l	l	o	\0
[0]	[1]	[2]	[3]	[4]	[5]

数组长度是6，字符串长度是5

2.4 字符串：结束标志

字符串实际存放在字符数组中，因此**定义字符数组时**
数组长度至少为字符串长度加**1**（空字符占位），**避免越界**

空字符串尽管字符串长度是0，但它依然要占据字符数组空间，空字符串""
的存储形式为：



\0

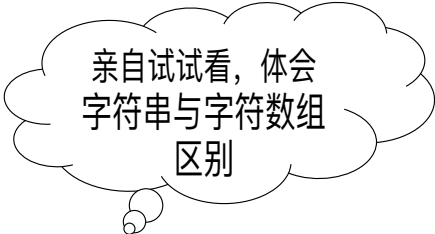
[0]

如果一个字符数组中**没有空字符而把它当作字符串使用**
，程序往往因为没有结束条件而**数组越界**。

2.4 字符串：结束标志

比较以下数组初始化方法的异同？？？

- `char c[10];`
`c[0]='I';c[1]=' '; c[2]='a'; c[3]='m'; c[4]=' ';c[5]='h'; c[6]='a';c[7]='p'`
`;c[8]='p';c[9]='y';`
- `char c[10]={ 'I',' ','a','m',' ','h','a','p','p','y' }`
- `char c[] = { 'I',' ','a','m',' ','h','a','p','p','y' }`
- `char c[] = "I am happy"`
- `char c[] = "I am happy"`



亲自试试看，体会
字符串与字符数组
区别

2.4 字符串: 定义

字符串不是C

语言的内置数据类型，但应用程序通常都将它当作基本类型来用，称为C风格字符串（C-style string）。

由于数组的整体操作是有限制的，例如不能赋值、运算、输入输出，所以**C语言标准库函数中专门针对字符串定义了许多函数**，可以方便地处理字符串。

2.4 字符串: 定义

字符串定义与字符数组完全相同，形式为：

```
char 字符串名[表达式], .....;
```

C语言允许使用字符串常量初始化字符数组

```
char s[12]={ "Hello World" }; //数组长度为字符串长度加1
```

```
char s[12]= "Hello World"; //字符串初始化
```

```
char s[]="Hello World"; //字符串初始化，编译器自动计算字符串长度
```

```
char s[11]="Hello World"; //不是字符串，因为它没有空字符
```

2.4 字符串：输入输出

方法1：用格式化输入（scanf）和输出（printf）函数

- ❑ 逐个字符输入输出。用格式符"%c"输入或输出一个字符。
- ❑ 将整个字符串一次输入或输出。用"%s"格式符，意思是对字符串的输入输出。

方法2: gets和puts

```
例如： char c[6];  
        for (int i = 0; i < 6-1; i++)  
            scanf("%d", &c[i]);  
        c[6] = 0;  
  
        for (int i = 0; i < 6-1; i++)  
            printf("%d", c[i]);
```

2.4 字符数组：输入输出

用scanf函数输入一个字符串

[?] 如果利用一个scanf函数输入多个字符串，则在输入时以空格分隔。

例如：`char str1[5], str2[5], str3[5];`
`scanf("%s %s %s", str1, str2, str3);`

输入数据：

How are you?

数组中未被赋值的元素的值自动置'\0'。

H	o	w	\0	\0
a	r	e	\0	\0
y	o	u	?	\0

注意：

- scanf函数中的输入项**如果字符数组名，不要再加地址符&**，因为在C语言中数组名代表该数组的起始地址。
- 由于scanf函数将空格、TAB、回车作为输入项的间隔，**所以输入字符串时遇到这三个字符就结束。**
- scanf输入完成后，**在字符串末尾添加空字符，未赋值单元置零。**

2.4 字符数组：输入输出

用printf输出一个字符串

例如, `char str[] = "China";`
`printf("%s", str);`

说明:

1. 用"%s"格式符输出字符串时, printf函数中的输出项是**字符数组名**, 而不是数组元素名。
2. 如果数组长度大于字符串实际长度, 也只输出到遇'\0'结束。
3. 输出字符不包括结束符'\0'。
4. **如果字符串没有空字符, printf就会引起数组越界。**
5. 如果一个字符数组中包含一个以上'\0', 则遇**第一个'\0'**时输出就结束。

2.4 字符数组：输入输出

gets函数

❓ 其一般形式为：

```
char *gets(char *s);
```

❓ 输入一个字符串到字符数组s中。s是字符数组或指向字符数组的指针，输入后自动添加'\0'

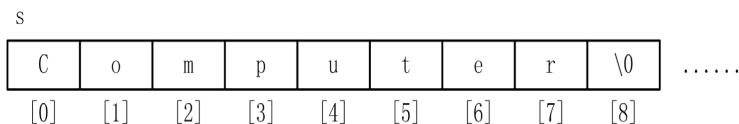
❓ **s是字符数组**或指向字符数组的指针(不是数组元素!!!)

❓ 返回值是字符数组的起始地址。

例如：char str[80];

 gets(str);

输入 Computer



注意：

- 输入的字符个数应**小于字符串定义的数组长度**，否则将导致数组越界。
- gets函数**可以输入空格和TAB**，但不能输入回车。

2.4 字符数组：输入输出

puts函数

❓ 其一般形式为：

```
int puts (char *s)
```

❓ 其作用是将一个字符串(以'\0'结束的字符序列)输出到终端，输完后再输出一个换行 ('\n')。

❓ **s是字符数组**或指向字符数组的指针，返回值表示输出字符的个数。

```
例如： char str[]={"China\nBeijing"};  
        puts(str);
```

输出： China
Beijing

注意：

- 用puts函数输出的字符串中**可以包含转义字符**
- 输出**第一个\0即结束**

2.4 字符串：处理函数

strcat函数

❓ 一般形式为

```
char *strcat(char *s1, const char
```

❓ strcat将s2字符串连接到s1的后面，包括空字符。

❓ s1是字符数组或指向字符数组的指针，其长度应该足够大，以便能容纳连接的字符串；

❓ s2可以是字符串常量、字符数组或指向字符数组的指针。

strncat函数

❓ strcat连接字符串时可能会由于一个字符串比另一个字符串长而导致溢出，使用strncat可以避免。n指定追加的字符数。

```
char *strncat(char *s1, const char *s2, size_t n);
```

2.4 字符串：处理函数

strcat函数

例如：

```
char str1[30]={"People's Republic of "};
```

```
char str2[]={"China"};
```

```
print("%s", strcat(str1, str2));
```

输出：

People's Republic of China

str1:	P	e	o	p	l	e	'	s		R	e	p	u	b	l	i	c		o	f		\0	\0	\0	\0	\0	\0	\0	\0	\0	
str2:	C	h	i	n	a	\0																									
str3:	P	e	o	p	l	e	'	s		R	e	p	u	b	l	i	c		o	f		C	h	i	n	a	\0	\0	\0	\0	\0

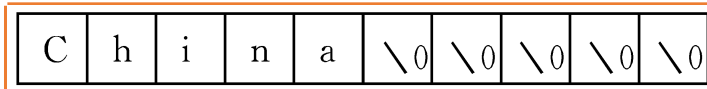
2.4 字符串：处理函数

strcpy函数

- ❓ 一般形式为：char *strcpy(char *s1, const char *s2);
- ❓ strcpy是"字符串复制函数"，将字符串2复制到字符数组1中去。

例如：

```
char str1[10], str2[]={ "China" };  
strcpy(str1, str2);
```



- ❓ 可以用strcpy函数将字符串2中前面若干字符复制到字符数组1中。

例如：

strcpy(str1, str2, 2); 作用是将str2中前面2个字符复制到str1中去，然后再加一个'\0'

2.4 字符串：处理函数

strcpy函数

说明：

- 字符数组1的长度不应小于字符串2的长度。（strncpy采用截断拷贝，可以避免溢出）
- "字符数组1"必须写成数组名形式(如str1), "字符串2"可以是字符数组名，
也可以是一个字符串常量。如strcpy(str1, "China")。
- 复制时连同字符串后面的'\0'一起复制到字符数组1中。
- 不能用赋值语句将一个字符串常量或字符数组直接给一个字符数组。

例如：

```
str1="China"; str1=str2;    //不合法
```

用strcpy函数只能将一个字符串复制到另一个字符数组中去。
用赋值语句只能将一个字符赋给一个字符型变量或字符数组的元素。

2.4 字符串：处理函数

strcmp函数

❓ 其一般形式为：

```
int strcmp(const char *s1, const char
```

❓ 作用是比较s1和s2

❓ s1和s2可以是字符串常量、字符数组或指向字符数组的指针。

❓ 函数值是比较的结果

- 如果字符串1=字符串2，函数值为0。
- 如果字符串1>字符串2，函数值为一正整数。
- 如果字符串1<字符串2，函数值为一负整数。

字符串大小比较的规则
"a"与"Big Foot"的大小？

2.4 字符串：处理函数

strcmp函数

- ❓ 字符串比较的规则是对两个字符串自左向右依次比较字符的ASCII数值，直到出现不同的字符或遇到空字符为止。
- ❓ 若全部字符相同，则认为字符串相等。
- ❓ 若出现不同的字符则以第一个不相同的字符的比较结果为准。

一般地，数字字符小于字母、大写字母
小于小写字母、英文小于汉字等

```
if (str1>str2) ..... // 不是字符串比较的含义  
if (strcmp(str1,str2)==0) ..... //比较字符串相等  
if (strcmp(str1,str2)!=0) ..... //比较字符串不相等  
if (strcmp(str1,str2)>0) ..... //比较str1大于str2  
if (strcmp(str1,str2)<0) ..... //比较str1小于str2
```

2.4 字符串：处理函数

strlen函数

❓ 其一般形式为：

```
size_t strlen(const char  
*s);
```

❓ strlen是测试字符串长度的函数，s
可以是字符串常量、字符数组或指向字符数组的指针。

❓ 函数的值为字符串中的实际长度(不包括'\0'在内)。

例如，

```
char str[20]="Visual Basic";  
n=strlen("Language"); //n=8  
n=strlen(str); //n=12  
n=sizeof str; //n=20
```