

# 2025

# C 语言笔记

C-NOTE

编 辑 人：梁培东

修 订 号：NOTE-250215

# 目录

目录 .....	1
DAY_1 Hello,world! .....	1
DAY_2 GCC 编译 .....	2
DAY_3 C 语言的标准 C99 对 C89 的改变 .....	3
DAY_4 数值数据的表示 数码、基、位权、数值 .....	5
DAY_5 为什么引入数据类型 .....	7
DAY_6 整型-数据类型 .....	9
DAY_7 变量的作用和用法 .....	11
DAY_8 字符型-数据类型 .....	13
DAY_9 浮点型-数据类型 浮点型-存储 .....	15
DAY_10 bool、void 类型 .....	19
DAY_11 强制类型转换 .....	21
DAY_12 运算符 .....	24
DAY_13 使用 vi 编辑器 .....	37
DAY_14 输入输出 .....	39
DAY_15 控制语句-if .....	44
DAY_16 控制语句-switch .....	46
DAY_17 循环语句-goto .....	48
DAY_18 循环语句-while .....	50
DAY_19 循环语句-for .....	53
DAY_20 辅助控制语句 .....	56
DAY_21 一维数组 .....	59
DAY_22 二维数组 .....	63
DAY_23 字符数组和字符串 .....	67
DAY_24 字符串逆序输出 .....	71
DAY_25 字符串函数 .....	74
DAY_26 指针的基本用法 .....	83
DAY_27 指针的运算 .....	85
DAY_28 指针与一维数组 .....	89
DAY_29 指针与二维数组 .....	91
DAY_30 字符指针与字符串 .....	94
DAY_31 指针数组 .....	96
DAY_32 多级指针 .....	98
DAY_33 void 指针和 const 指针 .....	100

## DAY\_1 Hello,world!

### ●一、Hello,world!

➤代码实现参考 `hello,world.c`

```
//引入头文件,如果不写,后面的 printf,编译时会引起报警
//stdio.h 代表输入输出头文件,拓展名.h 的文件叫做头文件
#include <stdio.h>

//main 代表主函数,C 语言程序的入口,有且仅有 1 个,不能是其他名字
int main()
{
    //左括号和右括号是对应的,括号中叫函数体,里面可以写多条语句
    printf("Hello, world!\n");
    //printf 代表打印的意思,能在屏幕上输出要显示的内容,是个库函数可以直接调用
    //\n 代表换行,表示输出完之后换行显示
    printf("Welcome to Beijing.\n");
    printf("It's a beautiful day.\n");
    return 0; //返回了一个 0,对应了 main 函数前面的 int
}
```

## DAY\_2 GCC 编译

方法一:gcc first.c -Wall

①gcc 是 C 语言编译器

②first.c 是文件名,就是我们要编译的 C 程序

③-Wall 是 gcc 的选项,这样编译时尽可能的多显示警告信息

该命令会在当前目录下生成一个默认的执行文件

a.out 如果 a.out 已经存在了,会覆盖之前的文件

➤代码实现参考 first.c

```
//引入头文件,如果不写,后面的 printf,编译时会引起报警
//stdio.h 代表输入输出头文件,拓展名.h 的文件叫做头文件
#include <stdio.h>

//main 代表主函数,C 语言程序的入口,有且仅有 1 个,不能是其他名字
int main()
{
    //左括号和右括号是对应的,括号中叫函数体,里面可以写多条语句
    printf("Hello, world!\n");
    //printf 代表打印的意思,能在屏幕上输出要显示的内容,是个库函数可以直接调用
    //\n 代表换行,表示输出完之后换行显示
    printf("Welcome to Beijing.\n");
    printf("It's a beautiful day.\n");
    return 0; //返回了一个 0,对应了 main 函数前面的 int
}
```

方式二:gcc first.c -o first -Wall

①这种方式,可以自行指定执行文件的名字

②该示例中,执行文件的名字为 first

③这样做好处是多个 c 文件存在时,编译用默认名字

a.out,会覆盖之前的,实际执行的是最后一次编译生成的执行文件

# DAY\_3 C 语言的标准 C99 对 C89 的改变

## 一、for 语句内的变量声明

C89:不允许 for 语句内出现变量声明,须事先定义.如:for(int i = 0; i < 10; i++) 不允许

C99:允许 for 语句内出现变量声明.

注意编译选项: -std=c89      -std=c99

➤代码实现参考 for\_demo.c

```
#include <stdio.h>

int main()
{
    int len = 10;
    for (int i = 0; i < len; i++)
    {
        printf("%d", i);
    }
    printf("\n");
    return 0;
}
```

## 二、对数组的增强,可变长数组

C89:规定数组不能是变量,必须是固定的整数.

C99:规定数组可以是变量.

如: gcc array.c -Wall -std=c89 编译不报错

gcc array.c -Wall -pedantic-errors -std=c89 编译报错

-pedantic 使得 gcc 编译器拒绝所有的 GNU C 扩展,遇到扩展报警

-pedantic-errors 选项使得编译器遇到扩展时报编译失败的错误

➤代码实现参考 array.c

```
#include<stdio.h>

int main()
{
    int len = 10;
    int a[len];
    int i;
    a[0] = 100;
    for(i = 0;i < len; i++)
    {
        a[i] = i + 1;
```

```
    printf("%d", a[i]);
}
putchar('\n');
return 0;
}
```

### 三、查看当前环境的 C 标准

命令行输入命令: man gcc, 在帮助手册中搜索 std (-std 是代表 c 标准), n 是下一个

gnu18 是 ISO C17 的 GUN 定制版, 这是 gcc 默认的 C 标准, C17 标准, 2018 年 12 月发布的

\gcc array.c -Wall

\gcc array.c -Wall -std=gnu18

# DAY\_4 数值数据的表示 数码、基、位权、数值

## 一、数码

表示数的符号

十进制的 0 1 2 3 4 5 6 7 8 9

八进制的 0 1 2 3 4 5 6 7

二进制的 0 1

## 二、基

数码的个数

十进制数码个数为 10

八进制数码个数为 8

二进制数码个数为 2

## 三、位权(权)

每个数码所表示的数值等于该数码乘以一个与该数码所在位置相关的常数(这个常数叫位权)

$$123 = 100 + 2 * 10 + 3 * 1$$

$$= 1 * 10^2 + 2 * 10^1 + 3 * 10^0$$

一个数码处在不同位置上所代表的值不同,比如数字 3 在个位数位置上表示 3,十位数位置表示 30,在百位数上表示 300

位权的大小是以基数为底、数码所在位置的序号为指数的整数次幂

## 四、数制

数制就是计数方法(逢几进一,借一当几)

二进制(Binary) 八进制(Octonary)

十进制(Decimal) 十六进制(Hexadecimal)

常用的进制对照表

数制	十进制	二进制	八进制	十六进制
数码	0~9	0~1	0~7	0~9, A~F, a~f
基	10	2	8	16
权	$10^0, 10^1, 10^2, \dots$	$2^0, 2^1, 2^2, \dots$	$8^0, 8^1, 8^2, \dots$	$16^0, 16^1, 16^2, \dots$
特点	逢十进一	逢二进一	逢八进一	逢十六进一

十进制	二进制	八进制	十六进制
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7

# DAY\_5 为什么引入数据类型

## 一、为什么要引入数据类型

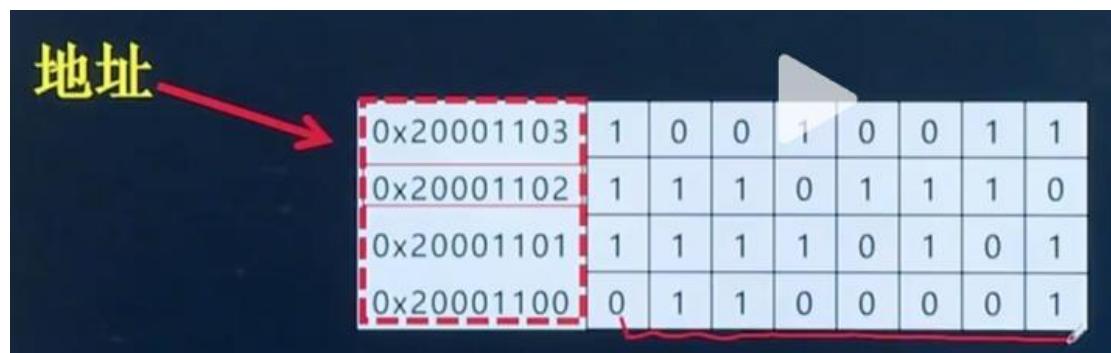
为了解决生活中的问题,需要用到数据类型:

比如:

年	龄(岁):	25	int
身高(厘米):	175.5	float	
性别(M,W):	M	char	

计算机中每个字节都有一个地址(类似门牌号)

CPU 通过地址来访问这个字节的空间



对于计算机系统,二进制的 0 和 1 数据没有任何意义

为了更接近现实生活,人为的规定了数据类型,便于有效组织管理这些数据

施加 int 类型后,代表数值 -290800982

施加 unsigned int 类型,代表数值 4004166314

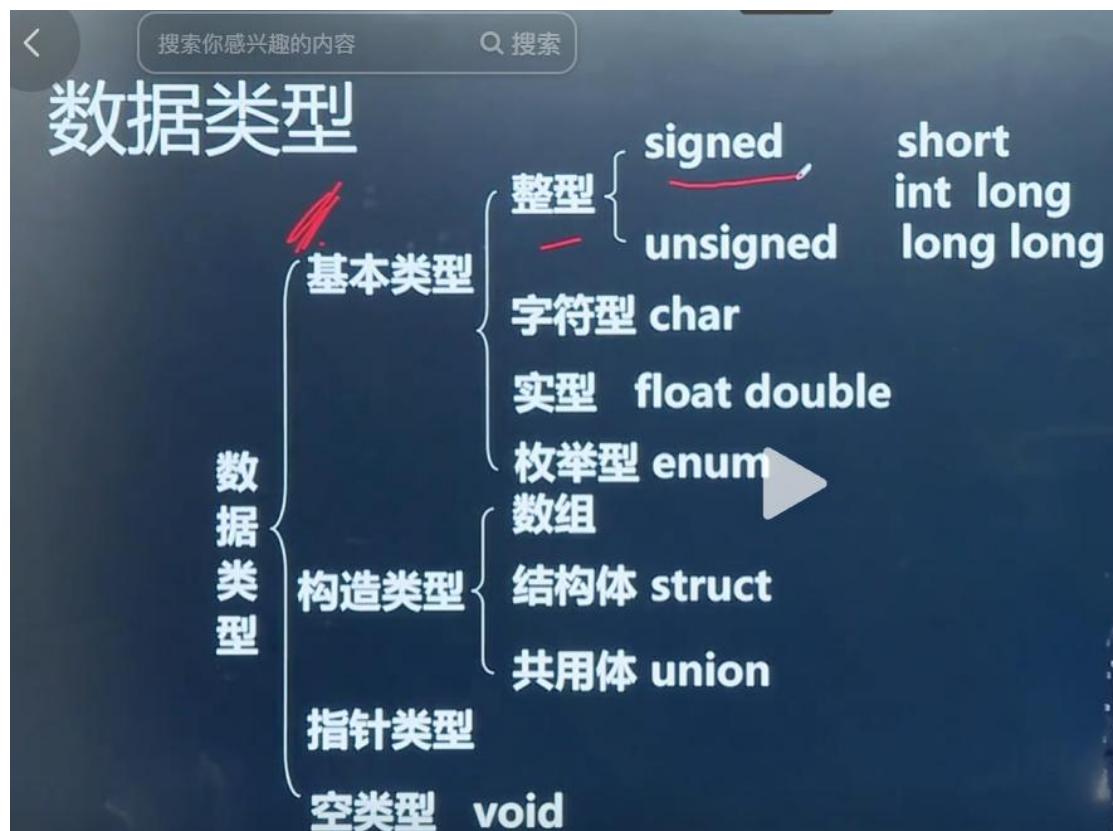
## 二、数据类型

➤代码实现参考 datatype.c

```
#include<stdio.h>

int main()
{
    int a = 0XEEAABAAA;
    printf("a = %#x %d\n", a, a);
    unsigned int b = 0XEEAABAAA;
    printf("b = %#x %u\n", b, b);
    return 0;
}
```

## 三、数据类型分类



# DAY\_6 整型-数据类型

一、请你列出至少 5 种数据类型,能存整数?

编号	数据类型	字节	取值范围
①	char	1	-128~127
②	unsigned char	1	0~255
③	short	2	-32768~32767
④	unsigned short	2	0~65535
⑤	int	4	-2147483648~2147483647
⑥	unsigned int	4	0~4294967295
⑦	long	4(32 位系统)、8(64 位系统)	
⑧	unsigned long	4(8)	
⑨	long long	8	-9223372036854775808~9223372036854775807
⑩	unsigned long long	8	0~18446744073709551615

二、在 C 语言中,如何编程得到每个整型的数据范围?整型的取值范围

➤代码实现参考 limit\_test.c

●在 linux 中, /usr/include/limits.h 文件中,有数据类型范围的定义

```
#include<stdio.h>
#include<limits.h> //linux 中的范围定义头文件
int main()
{
    printf("char:%d~%d\n",SCHAR_MIN,SCHAR_MAX);
    printf("short:%d~%d\n",SHRT_MIN,SHRT_MAX);
    printf("int:%d~%d\n",INT_MIN,INT_MAX);
    printf("long:%ld~%ld\n",LONG_MIN,LONG_MAX);      //%ld 表示 long 型变量输出
    printf("long long:%lld~%lld\n",LLONG_MIN,LLONG_MAX); //%lld 表示 long long
型变量输出
    return 0;
}
```

三、C 语言中,如何获取某个数据类型所占用的空间数?sizeof 运算符

●sizeof 是 C 语言中保留关键字,也是单目运算符.能获取某个数据类型所占用空间的字节数.

●使用形式:sizeof(变量名称)或者 sizeof 变量名或者 sizeof(数据类型)

●sizeof 返回值默认为无符号长整

```
int a;
sizeof(a) //√ 建议
sizeof a //√
sizeof(int) //√ 建议
sizeof int //×
```

➤代码实现参考 datatype2.c

```
#include<stdio.h>

int main()
{
    printf("char:%ld\n", sizeof(char)); //sizeof 返回值默认为无符号长整
    printf("short:%ld\n", sizeof(short)); //%ld 表示长整型
    printf("int:%ld\n", sizeof(int)); //当然也可以是%lu,表示无符号长整型
    printf("long:%ld\n", sizeof(long));
    printf("long long:%ld\n", sizeof(long long));
    return 0;
}
```

●注意:linux 下输入命令,查看系统架构命令:uname -m

●显示 x86\_64,为 64 位操作系统,则 long 与 long long 的占用的空间数为 8;若为 32 位系统,则为 4

# DAY\_7 变量的作用和用法

## 一、变量的作用

- 计算机中每一个字节的存储空间都有对应的地址,可以通过地址对其访问,进行读写操作.

0X2000000000

--	--	--	--	--	--	--	--

- 若需要内存中存储一个整数 58,怎么实现呢?

单片机逻辑开发,地址可以被直接赋值;但在 linux 下,由于不确定该地址是否被占用,不能直接赋值,存在段错误.

- C 语言设计了变量的概念,变量用来在程序中保存数据

比如:int val = 58;//声明一个 int 型变量 val, CPU 为 val 分配 4 个字节,可以存储数值 58

- C 语言提供了直接操作地址的功能,通过指针来访问某个地址空间,读写数据,后面再讲

## 二、变量的用法

- 变量的定义格式

✓关键字类型 变量名; int a;  
 ✓关键字类型 变量名 1, 变量名 2, ...; int a, b;

- 变量初始化赋值,要使用赋值运算符=,可以初始化时赋值,也可以初始化后单独赋值:

✓类型关键字 变量名 = 数值; int a = 20;  
 ✓类型关键字 变量名 1 = 数值, 变量名 2 = 数值, ...; int a = 10, b = 20;

- 表示将"=" 右边的值赋给左边的变量

## 三、变量的注意事项

- 变量名的要求:

1. 变量名以字母、数字、下划线组成;
2. 变量名不能以数字、下划线开头;
3. 变量名不能是关键字;
4. 变量名不能包括空格、标点符合、和类型说明符.

```
int num_1 = 5; //正确
int sum = 0; //正确
int 1_num = 5; //错误 不能数字开头
int %age = 18; //错误 不能有类型说明符(% & ! # @ $)
int while = 23; //错误 while 是 C 语言关键字
int name age = 10; //错误 不能有空格
```

- 变量必须先定义再使用

```
a = 100; //错误
int a;
```

- 变量在参与运算前,需要对其初始化:

```
int a;
int b;
b = a; //不会出现编译错误,但是存在运行风险,因为 a 的数值不确定
```

#### 四、变量的拓展

- 下面三点,后续课程会深入讲解

①变量的存储类型

```
auto  
extern  
register  
static
```

②局部变量和全局变量

③变量的作用范围

➤代码实现参考 DAY\_7\_test.c

```
#include<stdio.h>  
  
int main()  
{  
//    *(int *)0X20000400 = 58; // This line will cause a  
segmentation fault  
    int a = 58;  
    a = a + 1;  
    int b, c;  
    b = 10;  
    c = 20;  
    return 0;  
}
```

# DAY\_8 字符型-数据类型

## 一、计算机如何处理字符

- 计算机存储数据是 0,1 二进制数据
- 计算机存储字符数据,也需要把字符数据按照特定的编码格式转换成二进制数据进行存储

## 二、ASCII 码(0-255 共 256)

- 字符数据在机器内也被转换成二进制编码的格式
- 国际上普遍采用的一种编码是美国国家信息交换标准代码,简称为 ASCII 码  
在 linux 中,输入命令 `man ASCII`,查看对应的 ASCII 码

- ✓第一部分由 00H 到 1FH 共 32 个,一般用来通讯或作为控制之用,有些字符可显示于屏幕上,有些则无法显示在屏幕上
- ✓第二部分由 20H 到 7FH 共 96 个,这 96 个字符是用来表示阿拉伯数字、英文字母大小写和底线、括号等符号,都可以显示在屏幕上
- ✓第三部分由 80H 到 OFFH 共 128 个字符,一般称为[扩充字符],这 128 个扩充字符是由 IBM 制定的,并非标准的 ASCII 码.这些字符是用来表示框线、音标和其他欧洲非英语系的字母

## 三、字符型数据

数据类型	字节	取值范围
char	1	-128~127
unsigned char	1	0~255

### ●定义格式:

```
char c1 = 'A';
char c2 = 'a';
char c3 = 66;
注意字符常量要用单引号括起来,也可以用字符对应的 ASCII 码值进行赋值;
printf 对应的格式符是%c;
```

➤代码实现参考 `char1.c` `char2.c`

```
#include <stdio.h>
```

```
int main()
{
    char c1 = 'A';
    char c2 = 'B';
    char c3 = 'a';
    char c4 = 'b';
    char c5 = ' ';
    char c6 = '\n';
```

```

char c7 = '0';
char c8 = '1';
char c9 = 97;
printf("%c-%d\n", c1, c1);
printf("%c-%d\n", c2, c2);
printf("%c-%d\n", c3, c3);
printf("%c-%d\n", c4, c4);
printf("%c-%d\n", c5, c5);
printf("%c-%d\n", c6, c6);
printf("%c-%d\n", c7, c7);
printf("%c-%d\n", c8, c8);
printf("%c-%d\n", c9, c9);
return 0;
}

```

#### 四、大小写字母的转换

- 大写字母转小写字母 +32
- 小写字母转大写字母 -32

➤代码实现参考 char3.c

```

#include <stdio.h>

int main()
{
    char c1 = 'A';
    char c2 = 'a';
    printf("%d-%c\n", c1 + 32, c1 + 32); //大写转小写
    printf("%d-%c\n", c2 - 32, c2 - 32); //小写转大写
    return 0;
}

```

➤代码实现参考 char4.c

```

#include <stdio.h>

int main()
{
    int a = 3;
    char c1 = a + 48;
    char c2 = '2';
    int b = c2 - 48;
    printf("%d-%c\n", c1, c1);
    printf("%d\n", b);
    return 0;
}

```

# DAY\_9 浮点型-数据类型 浮点型-存储

## 一、浮点型

- 浮点型:用于存储小数,即数学中的实数

类型	关键字	长度	精确位数
单精度	float	4字节	6位
双精度	double	8字节	15-16位
长精度(C99新增)	long double	看编译器	精度不少于double的精度

➤代码实现参考 float\_demo.c

```
#include <stdio.h>

int main()
{
/*
*   float
*   double
*   long double
*/
    printf("float: %lu\n", sizeof(float));
    printf("double: %lu\n", sizeof(double));
    printf("long double: %lu\n", sizeof(long double));
    return 0;
}
```

## 二、浮点数的表示形式

- 浮点数 指数形式

0.008	8E-3
8000	8E+3

➤代码实现参考 float\_demo2.c

```
#include <stdio.h>
int main()
{
    float f1 = 0.008;
    float f2 = 8E-3;
    float f3 = 8000;
    float f4 = 8E+3;
    printf("f1 = %f\n f2 = %f\n f3 = %f\n f4 = %f\n", f1, f2, f3,
f4);
    return 0;
}
```

### 三、浮点型精度

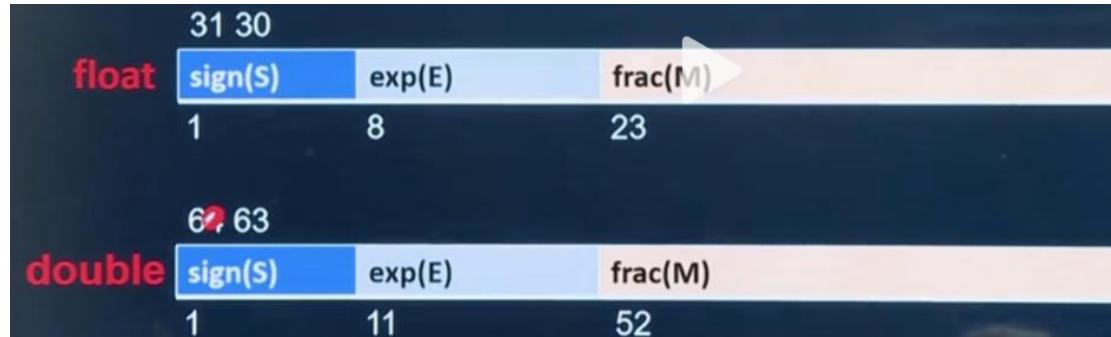
- %.30f 表示小数点后要凑够三十位,f 代表浮点型;若不够 30 位,按照默认单精度 6 位,后面的数据皆不准确
  - %lf 是 double 的格式说明符
  - %Lf 是 long double 的格式说明符
  - 代码实现参考
  - 绝大多数的浮点数是不能精确表示的
- 代码实现参考 float\_demo3.c

```
#include <stdio.h>

int main()
{
    float a = 1.12345678901234567890;
    double b = 1.12345678901234567890;
    long double c = 1.12345678901234567890;
    printf("float = [%.30f]\n", a);
    printf("double = [%.30lf]\n", b);
    printf("long double = [%.30Lf]\n", c);
    return 0;
}
```

### 四、IEEE754:计算机浮点数格式标准

- $f = (-1)^s \cdot M \cdot 2^E$
- S:符号(0 为正 1 为负) E:指数 M:尾数(有效位数)



## 五、IEEE-754 标准规定(略)

- 举例：浮点数 float 9.625 在内存中的存储

**举例：浮点数float 9.625 在内存中的存储**

1. 十进制转换为二进制

整数： $9 = (\underline{\text{1001}})B$	小数： $0.625 * 2 = \text{1.25}$ $0.25 * 2 = \text{0.5}$ $0.5 * 2 = \text{1.0}$ $0.625 = (\underline{\text{.101}})B$	<span style="color: red; border: 1px solid red; border-radius: 50%; padding: 2px 5px;">乘2取整</span> 小数部分为0时，运算结束，若不为0，比如0.1，只能保存近似值
--------------------------------------	--	---

9.625 = (1001.101)B

2. 确定 S: 符号 E: 指数 M: 尾数

$1001.101 = (-1)^0 * 1.001101 * 2^3$

S: 0      M: 1.001101      E: 3

实际存储 S: 0      M: 去掉整数1, 001101      E: 加上偏移127, 130

S	E	M
0	1 0 0 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	十六进制： 0x411A0000
31		



● <https://www.h-schmidt.net/FloatConverter/IEEE754.html>

● <https://baseconvert.com/ieee-754-floating-point>

用来对应转换

➤ 代码实现参考 float\_demo4.c

```
#include <stdio.h>

int main()
{
    float f = 9.625;
    printf("%#x\n", *(int *)&f); // %#x 以十六进制输出 十六进制格式
说明符
    return 0;
}
```

## 六、浮点型相关测试

```
#include <stdio.h>

int main()
{
    float f1 = 2.2;
    if (f1 == 2.2){
        printf("f1 == 2.2\n");
    }else{
        printf("f1 != 2.2\n");
    }

    float f2 = 2.2;      //代表 f2 是个浮点型的常数
    if (f2 == 2.2f){
        printf("f2 == 2.2f\n");
    }else{
        printf("f2 != 2.2f\n");
    }

    float f3 = 2.25;     //代表 f2 是个浮点型的常数
    if (f3 == 2.25){
        printf("f3 == 2.25\n");
    }else{
        printf("f3 != 2.25\n");
    }
    return 0;
}
```

●提示:根据 IEEE-754 标准思考是否相等

答案: f1 != 2.2      f2 == 2.2f      f3 == 2.25

●对于 f1:浮点数常数默认是 double 类型,2.2 是不能精确表示的,float 类型的 2.2 和 double 类型的 2.2 是不相等的,所以不等于 2.2

●对于 f2:float 类型添加常熟数值 f,所以 2.2f 是 float 类型,所以 f2 等于 2.2f

●对于 f3:2.25 是可以精确表示的 float,float 类型的 2.25 和 double 类型的 2.25 是相等的,所以 f3 等于 2.25

➤代码实现参考 demo\_compare.c(代码同上)

# DAY\_10 bool、void 类型

## 一、思考

C 语言有 bool 类型吗?

\_Bool 类型是什么?

## 二、\_Bool 类型-基础类型

●从 C99 标准开始,增加了关键字 \_Bool 用来表示布尔类型.

- ✓ 只能取值为 1 或 0
- ✓ 非零值为真(1),代表条件成立
- ✓ 零为假,代表条件不成立
- ✓ 占用的空间是 1 个字节,使用二进制的第 0 位来表示 0 和 1

➤代码实现参考 \_Bool.c

```
#include <stdio.h>

int main()
{
    _Bool a = -1;
    printf("%d\n", a);
    if (a) {
        printf("true:%d\n", a);
    }
    else {
        printf("false:%d\n", a);
    }
    printf("%ld\n", sizeof(_Bool));
    return 0;
}
```

●C99 中增加了一个头文件 stdbool.h 并在其中定义了 bool、ture 和 false

```
/usr/lib/gcc/x86_64-linux-gnu/11/include/stdbool.h
#define bool _Bool
#define ture 1
#define false 0
```

➤代码实现参考 bool.c

```
#include <stdio.h>
#include <stdbool.h>

int main()
{
    bool a = -1;
    printf("%d\n", a);
```

```

if (a) {
    printf("true:%d\n", a);
}
else {
    printf("false:%d\n", a);
}
printf("%ld\n", sizeof(bool));
return 0;
}

```

- bool 数据类型的数值只有 1 和 0 两种,对应 true(真)和 false(假)

转换规则:任何非零数值转换为 bool 类型都为 1

➤代码实现参考 bool\_test.c

```

#include <stdio.h>
#include <stdbool.h>

int main()
{
    bool f = false; // #define false 0
    printf("f = %d\n", f);
    f--; // a = a - 1 此时 f = -1, 输出为 1
    printf("f = %d\n", f);
    f--; // a = a - 1 此时 f = 0, 输出为 0
    printf("f = %d\n", f);
    f--; // a = a - 1 此时 f = -1, 输出为 1
    printf("f = %d\n", f);
    return 0;
}

```

### 三、void 类型(了解)

- 该类型也叫缺省值,代表没有类型.
- 它通常用作一种占位符,或用来表示“无返回值”的函数,或指向任一类型的指针等

### 四、void 类型的用法

- 定义不返回任何值的函数

```

void printfHelloWorld()
{
    printf("HelloWorld!");
}

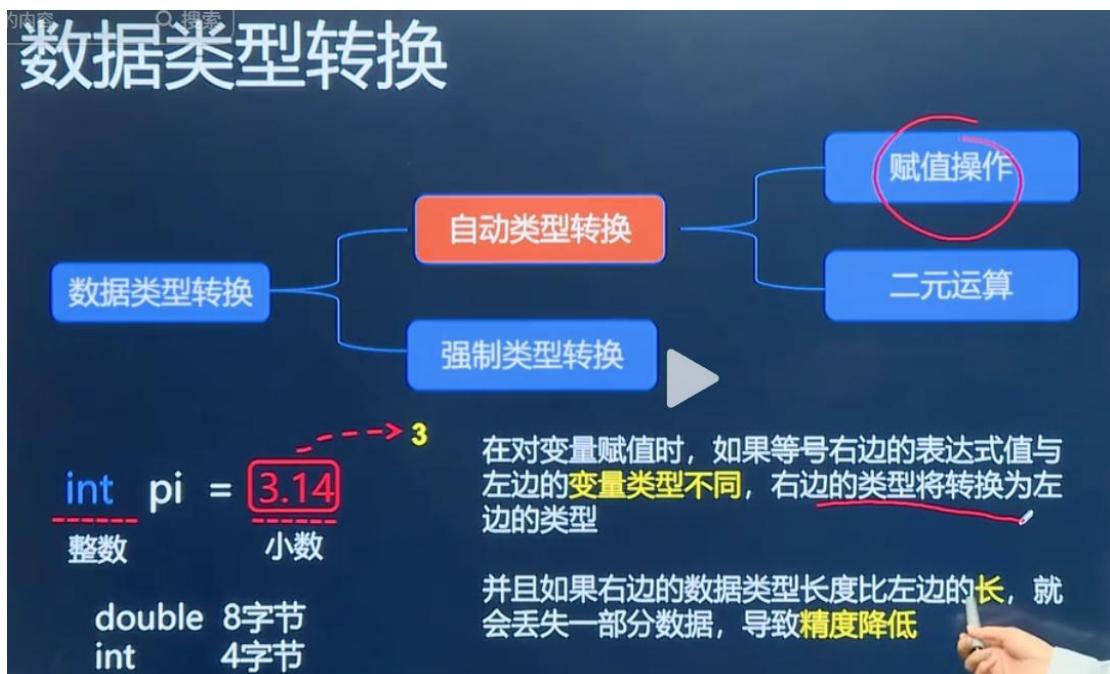
```

- 通用指针类型

void \*指针类型通常称为通用指针类型.这种指针可以指向任何类型的数据.

# DAY\_11 强制类型转换

## 一、数据类型转换



## 二、自动类型转换

- 赋值操作

➤ 代码实现参考 type\_conversion1.c

```
#include <stdio.h>

int main()
{
    int pi = 3.14;
    double a = 100;
    printf("%d\n", pi);
    printf("%lf\n", a);
    return 0;
}
```

a 的小数部分被舍弃了, b 没有精度损失

- 在不同的数据类型之间进行二元运算时，也会发生隐式的自动类型转换

- 如果参与运算的变量类型不同，会先转换成同一类型再进行运算

➤ 代码实现参考 type\_conversion2.c

```
#include <stdio.h>

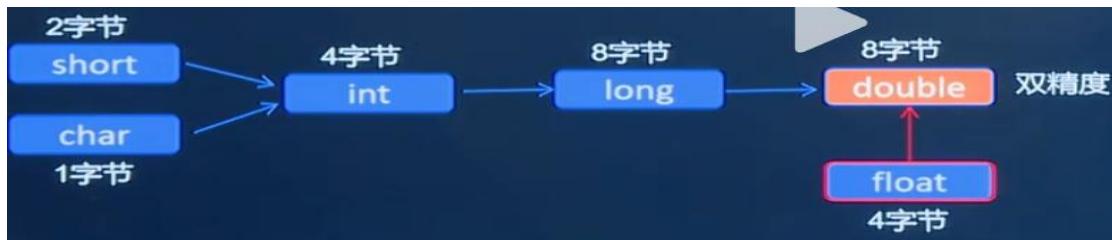
int main()
{
    char ch = 'A';
```

```

int a = 1000;
printf("%d\n", ch + a);
return 0;
}

```

- 如果运算时发生类型转换,就会按数据长度增加的方向进行,从而保证精度不降低
- 并且所有浮点数参与的运算都以双精度进行,即使表达式中只有 float,也会先转为 double 再进行计算



### 三、隐式自动类型转换

- 隐式自动类型转换是编译器根据代码上下文自行判断的结果

➤代码实现参考 type\_conertion3.c

```

#include <stdio.h>

int main()
{
    //char ch = 'A';
    int a = 1000;
    double f;
    f = a * 0.3;
    //printf("%d\n", ch + a);
    printf("f = %lf\n", f);
    return 0;
}

```

### 四、强制类型转换

- 为了确保类型转换按照开发者期望的方式进行,可以明确设置类型转换的方式,这就是强制类型转换
- 强制类型转换是一种运算符,在需要转换的表达式前添加小括号括起来的新类型名称

(type\_name)expression  
新类型名称 表达式

➤代码实现参考 type\_conertion4.c

```

#include <stdio.h>

int main()
{
    int a = 200;
    double b = 4.5;

```

```

    double c = 3.9;
    printf("%f\n", (float)a); //int to float
    printf("%d\n", (int)(b + c)); //double to int
    return 0;
}

```

- 有时,为了得到正确的结果,必须要使用强制类型转换
- 示例:要求变量 c 可以保留 2 个小数位

```

int a = 100;
int b = 3;
double c;
c = a / b;

```

➤代码实现参考 type\_conertion5.c

```

#include <stdio.h>

int main()
{
    int a = 100;
    int b = 3;
    double c;
    double d;
    c = a / b; //这里由于 a,b 为 int,因此相除结果取整数为 33,转换为
double 后输出结果为 33.00, false
    d = (double)a / b; //强制类型转换 a/b 的结果
    printf("%.2lf\n", c);
    printf("%.2lf\n", d); //输出结果为 33.33
    printf("%.2lf\n", (double)a / b); //输出结果为 33.33
    printf("%.2lf\n", (double)a / (double)b); //输出结果为 33.33
    printf("%.2lf\n", (double)(a / b)); //输出结果为 33.00,只
double 了 a
    return 0;
}

```

# DAY\_12 运算符

## 一、算术运算符

●C 语言提供算术运算符:`+, -, *, /, %, ++, --`, 如下:float,double 不能取余



运算符	功能说明	举 例
<code>+</code>	加, 一目取正	<code>a+b</code>
<code>-</code>	减, 一目取负	<code>a-b</code>
<code>*</code>	乘法	<code>a*b</code>
<code>/</code>	除法	<code>a/b</code>
<code>%</code>	取模或求余	<code>a%b</code>
<code>++</code>	增 1	<code>a++, ++b</code>
<code>--</code>	减 1	<code>a--, --b</code>

### ●整型变量算术运算

➤代码实现参考 `ope.c`

```
#include <stdio.h>

int main()
{
    int a = 15, b = 8, c;
    c = a + b;
    printf("c = %d\n", c);
    c = a - b;
    printf("c = %d\n", c);
    c = a * b;
    printf("c = %d\n", c);
    c = a / b;
    printf("c = %d\n", c);
    c = a % b;
    printf("c = %d\n", c);
    return 0;
}
```

## ●浮点型变量算术运算

➤代码实现参考 ope2.c

```
#include <stdio.h>

int main()
{
    float a = 15, b = 8, c;
    c = a + b;
    printf("c = %f\n", c);
    c = a - b;
    printf("c = %f\n", c);
    c = a * b;
    printf("c = %f\n", c);
    c = a / b;
    printf("c = %f\n", c);
//    c = a % b;
//    printf("c = %f\n", c);
    return 0;
}
```

●注意,对于浮点数不能取余

## 二、关系运算符

●C 语言的关系运算符如下:

运算符	功能说明	举 例
>	大于	a>b
>=	大于等于	a>=5
<	小于	3<x
<=	小于等于	x<=y+1
==	等于	x+1==0
!=	不等于	c != '\0'

例如

```
int a = 5, b = 6;
a > (b - 1)    结果值为 0
(a + 1) == b   结果值为 1
a >= (b - 2)   结果值为 1
a < 100        结果值为 1
(a + 3) <= b   结果值为 0
a != (b - 1)   结果值为 0
```

## 三、逻辑运算符

●c 语言的逻辑运算符如下:

运算符	功能说明	举 例
!	逻辑反	$!(x == 0)$
&&	逻辑与	$x > 0 \&\& x < 100$
	逻辑或	$y < 10    z < 10$

●逻辑非“!”运算符的运算律如下:

运算量	结果
1	0
0	1

例如,

```
int k = 8;
! (k == 0)      结果为 1
! ((k - 8) == 0)    结果为 0
! (k <= 0)      结果为 1
```

➤代码实现参考 ope3.c

```
#include <stdio.h>

int main()
{
    int a = 8, b = 7;
//    if (!(a - b) == 7){ //对(a - b)取反,依旧不满足, false
    if (!((a - b) == 7)){ //对(a - b) == 7 取反,满足, true
        printf("True\n");
    } else {
        printf("False\n");
    }
    return 0;
}
```

- 逻辑与 “`&&`” 运算符的运算规律如下:短路特性-逢 0 截止

左运算量	右运算量	结果
1	1	1
1	0	0
0	1	0
0	0	0

例如

```
int x = 5, y = 18;
(x >= 5) && (y < 20)      结果值为 1
((x + 1) >= 0) && (y < 17)  结果值为 0
((x - 8) >= 0) && (y == 18)  结果值为 0
((x - 5) > 0) && (y != 18)   结果值为 0
```

- 逻辑或 “`||`” 运算符的运算规律如下:短路特性-逢 1 截止

左运算量	右运算量	结果
1	1	1
1	0	1
0	1	1
0	0	0

例如

```
int x = 5, y = 18;
(x >= 5) || (y < 20)      结果值为 1
((x + 1) >= 0) || (y < 17)  结果值为 1
((x - 8) >= 0) || (y == 18)  结果值为 1
((x - 5) > 0) || (y != 18)   结果值为 0
```

## 四、位运算符

●C 语言的位运算符如下：

运算符	功能说明	举例
<code>~</code>	位逻辑反	<code>~a</code>
<code>&amp;</code>	位逻辑与	<code>a&amp;b</code>
<code> </code>	位逻辑或	<code>a b</code>
<code>^</code>	位逻辑异或	<code>a^b</code>
<code>&gt;&gt;</code>	右移位	<code>a&lt;&lt;1</code>
<code>&lt;&lt;</code>	左移位	<code>b&gt;&gt;4</code>

●位逻辑取反“`~`”运算符的运算规律如下：

例如：

```
unsigned char x = 0x17, y;
y = ~x;
printf("%#x\n", y);    // x 代表是 16 进制, #代表打印这个结果的时候自动加上 0x 前缀.
```

计算机内存中的 x 内容如下：
0 0 0 1 0 1 1 1
y 被赋值后计算机内存中的内容如下：
1 1 1 0 1 0 0 0

➤代码实现参考 bit1.c

```
#include <stdio.h>

int main()
{
    unsigned char x = 0x17, y;
    y = ~x;
    /* x 代表是 16 进制, #代表打印这个结果的时候自动加上 0x 前缀. */
    printf("%#x\n", y);
    return 0;
}
```

- 位逻辑与“&”运算符的运算规律如下:

左运算量的值	右运算量的值	位与结果值
0	0	0
0	1	0
1	0	0
1	1	1

例如:

```
unsigned char x = 0126, y = 0xac, z;
z = x & y;
printf("%#x\n", z);    x 代表是 16 进制, #代表打印这个结果的时候自动加上 0x 前缀.
```

➤代码实现参考 bit2.c

```
#include <stdio.h>

int main()
{
    unsigned char x = 0126, y = 0xac, z;
    z = x & y;
    /* x 代表是 16 进制, #代表打印这个结果的时候自动加上 0x 前缀. */
    printf("%#x\n", z);
    return 0;
}
```

- 位逻辑或“|”运算符的运算规律如下:

左运算量的值	右运算量的值	位或结果值
0	0	0
0	1	1
1	0	1
1	1	1

例如:

```
unsigned char x = 076, y = 0x89, z;
z = x | y;
printf("%#x\n", z);    x 代表是 16 进制, #代表打印这个结果的时候自动加上 0x 前缀.
```

$x   y$ 位逻辑或运算:								
左运算量	0	0	1	1	1	1	1	0
右运算量	1	0	0	0	1	0	0	1
结 果	1	0	1	1	1	1	1	1
$z$ 赋值结果为: 0xbff								

➤代码实现参考 bit3.c

```
#include <stdio.h>

int main()
{
    unsigned char x = 076, y = 0x89, z;
    z = x | y;
    /* x 代表是 16 进制, #代表打印这个结果的时候自动加上 0x 前缀. */
    printf("%#x\n", z);
    return 0;
}
```

●位逻辑异或“^”运算符的运算规律如下:

左运算量的位值	右运算量的位值	位异或结果值
0	0	0
0	1	1
1	0	1
1	1	0

例如:

```
unsigned char x = 75, y = 0173, z;
z = x ^ y;
printf("%#x\n", z);    x 代表是 16 进制, #代表打印这个结果的时候自动加上 0x 前缀.
```

$x \wedge y$  的位逻辑异或运算:

左运算量	0	1	0	0	1	0	1	1
右运算量	0	1	1	1	1	0	1	1
$\wedge$ 结果	0	0	1	1	0	0	0	0

$z$  赋值结果为: 060

➤代码实现参考 bit4.c

```
#include <stdio.h>

int main()
{
    unsigned char x = 75, y = 0173, z;
    z = x ^ y;
    /* x 代表是 16 进制, #代表打印这个结果的时候自动加上 0x 前缀. */
    printf("%#x\n", z);
    return 0;
}
```

●位移位运算的一般形式:

<运算量><运算符><表达式>其中:

<运算量>必须为整型结果数值;

<运算符>为左移位(<<) 或 右移位(>>)

<表达式>也必须为整型结果数值。

例如:

```
unsigned char a = 0xe4, b;
```

```
b = a << 3;
```

printf("%#x\n", b);    x 代表是 16 进制, #代表打印这个结果的时候自动加上 0x 前缀.

运算量 a 左移位 3 位存入 b 的情况:

左运算量	1	1	1	0	0	1	0	0
	$a \ll 3$							
$\ll$ 结果	0	0	1	0	0	0	0	0

➤代码实现参考 bit5.c

```
#include <stdio.h>
```

```
int main()
```

```
{
    unsigned char a = 0xe4, b;
    b = a << 3;
    /* x 代表是 16 进制, #代表打印这个结果的时候自动加上 0x 前缀. */
    printf("%#x\n", b);
    return 0;
}
```

●如何把一个无符号数的某位快速置 1?

例如:

$x = 0x4$   $y = 2$  第 2 位置 1,用或|运算, 0 或 0 还是 0 保持原来的数字, 0 或 1 实现置 1

000001**00**

000000**10**

00000110

①令 1 左移 1 位即可得到 000000**10**, 即  $1 \ll (y - 1)$

②得到的结果,使  $x = x | (1 \ll (y - 1))$  即可

➤代码实现参考 bit6.c

```
#include <stdio.h>

int main()
{
    unsigned char x = 0x4, y = 2;
    /* 要使得第 2 位快速置 1,用|运算置 1 */
    x = x | (1 << (y - 1));
    /* x 代表是 16 进制, #代表打印这个结果的时候自动加上 0x 前缀. */
    printf("%#x %#x\n", x, y);
    return 0;
}
```

●如何把一个无符号数的某位快速清零?

例如:

$x = 0x14$   $y = 3$  第 3 位清零,用或&运算, 0 与 1 为 0 保持原来数字, 1 与 0 实现清 0,

00010**100**

11111**011**

00010**000**

要获得 11111**011**, 不妨先获得取反数 00000**100**, 该数取反后就得到 11111**011**.

①令 1 左移 2 位即可得到 00000**100**, 即  $1 \ll (y - 1)$

②将左移数取反得到 11111**011**, 即  $(\sim(1 \ll (y - 1)))$

③得到的结果,使  $x = x | (\sim(1 \ll (y - 1)))$  即可

➤代码实现参考 bit7.c

```
#include <stdio.h>

int main()
{
    unsigned char x = 0x14, y = 3;
    /* 要使得第 3 位快速清零,&运算清 0 */
    x = x & (~ (1 << (y - 1)));
    /* x 代表是 16 进制, #代表打印这个结果的时候自动加上 0x 前缀. */
    printf("%#x %#x\n", x, y);
    return 0;
}
```

●如何利用位运算把一个十进制数转化成一个十制数?

➤代码实现参考 bit8.c

```
#include <stdio.h>

int main()
{
    int a = 65, b = 0;
    /* a 的 16 进制为 0x41 */
    printf("%#x %d\n", a, a);

    /*
     * b = a / 16
     * b 对 a 进行 16 位进制运算取整,商 4
     * a 左移 4,得到后四位的 16 进制为 4
     */
    b = a >> 4;
    printf("%#x %d\n", b, b);

    /*
     * b = a % 16;
     * b 对 a 进行 16 位进制运算求余,余 1
     * 得到的余数即为后 4 位,
     * 前四位清 0,后四位保留.
     * 用&运算,前四位 0000,后四位 1111,
     * 00001111 在 16 进制位为 f.
     */
    b = a & 0xf;
    printf("%#x %d\n", b, b);
    return 0;
}
```

## 五、赋值运算符

### ●赋值复合运算符

赋值运算符为“=”，其运算的一般形式如下：

<左值表达式> = <右值表达式>

赋值复合运算符其运算的一般形式如下：

<变量><操作符> = <表达式>

C 语言的赋值复合运算符如下：

运算符	功能说明	示例	
$+=$	加赋值复合运算符	$a+=b$	等价于 $a=a+b$
$-=$	减赋值复合运算符	$a-=b$	等价于 $a=a-b$
$*=$	乘法赋值复合运算符	$a*=b$	等价于 $a=a*b$
$/=$	除法赋值复合运算符	$a/=b$	等价于 $a=a/b$
$\%=$	求余赋值复合运算符	$a\%=b$	等价于 $a=a \% b$
$\&=$	位与赋值复合运算符	$a\&=b$	等价于 $a=a \& b$
$ =$	位或赋值复合运算符	$a =b$	等价于 $a=a b$
$\sim=$	位异或赋值复合运算符	$a\sim=b$	等价于 $a=a\sim b$
$>>=$	位右移赋值复合运算符	$a>>=b$	等价于 $a=a>>b$
$<<=$	位左移赋值复合运算符	$a<<=b$	等价于 $a=a<<b$

➤代码实现参考 assigndemo.c

```
#include <stdio.h>

int main()
{
    int n = 0, sum = 0;
    while (n++ < 100) {
        sum += n;
    }
    printf("sum = %d\n", sum);
    return 0;
}
```

## 六、特殊运算符

### ●条件运算符”?：“是三目运算符,其运算的一般形式是:

<表达式 1> ? <表达式 2> : <表达式 3>

类似于 if (<表达式 1>) {

```
    <表达式 2>;
} else {
    <表达式 3>;
}
```

例如：

```
int x= 82, y = 101;
x >= y ? x + 18 : y - 100 //运算结果为 1
x < (y - 11) ? x - 22 : y - 1 //运算结果为 0
```

➤代码实现如下

```
#include <stdio.h>

int main()
{
    int x = 70, y = 10;
    y = x++ > 70 ? x + y : 5; //此处先对 x 是否大于 70 做判断,然后 x
依旧要自增
    printf("x = %d y = %d\n", x, y); //结果为 x = 71 y = 5

    x = 70, y = 10;
    y = ++x > 70 ? x + y : 5;//此处先对 x 自增,然后判断是否大于 70
    printf("x = %d y = %d\n", x, y); //结果为 x = 71 y = 81
    return 0;
}
```

### ●逗号运算符

执行顺序为从左到右,最后的表达式的值会赋给最左边

```
float x = 10.5, y = 1.8, z = 0;
z = (x += 5, y = x+0.2); //x = 15.5, y = 15.7,最后得到 z = 15.7
z = (x = y = 5, x += 1);
z = (x = 5, y = 6, x + y);
z = (z = 8, x = 5, y = 3);
```

## 七、C 运算符的优先级

优先级	运算符及其含义	结合规律
1	<code>  </code> <code>()</code> <code>~&gt;</code> 后缀 <code>++</code> 后缀 <code>-</code>	从左向右
2	前缀 <code>++</code> 前缀 <code>-</code> <code>sizeof</code> <code>&amp;</code> <code>*</code> <code>+ -</code> <code>- -</code> <code>!</code>	从右向左
3	强制类型转换	从右向左
4	<code>*</code> <code>/</code> <code>%</code> (算术乘除)	从左向右
5	<code>+</code> <code>-</code> (算术加减)	从左向右
6	<code>&lt;&lt;</code> <code>&gt;&gt;</code> (位移位)	从左向右
7	<code>&lt;</code> <code>&lt;=</code> <code>&gt;</code> <code>&gt;=</code>	从左向右
8	<code>= =</code> <code>!=</code>	从左向右
9	<code>&amp;</code> (位逻辑与)	从左向右
10	<code>^</code> (位逻辑异或)	从左向右
11	<code> </code> (位逻辑或)	从左向右
12	<code>&amp;&amp;</code>	从左向右
13	<code>  </code>	从左向右
14	<code>? :</code>	从右向左
15	<code>=</code> <code>*=</code> <code>/=</code> <code>%=</code> <code>+=</code> <code>-=</code> <code>&amp;=</code> <code>^=</code> <code> =</code>	从右向左
16	<code>.</code>	从左向右

# DAY\_13 使用 vi 编辑器

●vi 是 Linux 中最基本的编辑器,但编辑器在系统管理,服务器配置工作中永远都是无可替代的。

## 一、vi 的三种模式

### ●命令行模式

用户在用 vi 编辑文件时, 最初进入的为该模式。可以进行复制、粘贴等操作。

### ●插入模式

进行文件编辑, 按 ESC 键可以回到命令行模式。

### ●底行模式

光标位于屏幕的底行。可以进行文件的保存、查找、替换、列出行号等。

## 二、vi 模式切换命令

命令	功能
i(Insert)	进入插入模式在 光标当前位置 之前 插入文本
a(Append)	进入插入模式在 光标当前位置 之后 插入文本
o(Open)	进入插入模式在 当前行的下一行 开始插入新文本
I	进入插入模式在 行首 插入文本
A	进入插入模式在 行尾 插入文本
O	进入插入模式在 当前行的上一行 开始插入新文本
:! Command	在 vi 中执行外部命令 Command,按回车键可回到 vi 继续工作

## 三、vi 保存和退出命令

命令	功能
:q	(Quit)退出没有修改的文件 (若文件被修改而没有保存, 则此命令无效)
:q!	强制退出, 且不保存修改过的部分
:w	(Write)保存文件, 但不退出 类似于随手保存 ctrl+s
:x	(Exit)保存文件并退出 类似于:wq
:x File	另存为 File 给出的文件名, 不退出
:r File	这个命令用于将指定文件的内容插入当前文件中

## 四、vi 拷贝与粘贴命令

命令	功能
[N]x	(Expurgate)删除从光标位置开始的连续 n 个字符 (并复制到编辑缓冲区)
[N]dd	(Delete)删除从光标位置开始的连续 n 行 (并复制到编辑缓冲区)
[N]yy	(Yank)复制从光标位置开始的连续 n 行到编辑缓冲区
p 或 P	(Put)从编辑缓冲区复制文本到当前光标 (即粘贴)
u	(Undo)取消上一次操作 (即撤销)

## 五、vi 光标命令

命令	功能
h	方向键，向左移动光标一个字符的位置，相当于键“←”
j	方向键，向下移动光标到下一行的位置，相当于键“↓”
k	方向键，向上移动光标到上一行的位置，相当于键“↑”
l	方向键，向右移动光标一个字符的位置，相当于键“→”
:N	移动光标到第 N 行 (N 待定)
1G	移动光标到文件的第 1 行
G	移动光标到文件的最后一行
:set number	设置显示行号
:set	取消显示行号

## 六、vi 的查找命令

- /string 查找字符串

n 继续查找

N 反向继续查找

支持正则表达式，比如： /the /end

## 七、vi 的替换命令

- 利用:s 命令可以实现字符串的替换。

/:s/str1/str2/

:s/str1/str2/g

.,\$ s/str1/str2/g

:1,\$ s/str1/str2/g

:%s/str1/str2/g

## 八、vi 复制和剪切命令

- y0 - 将光标至行首的字符拷入剪贴板
- y\$ - 将光标至行尾的字符拷入剪贴板
- d0 - 将光标至行首的字符剪切入剪贴板
- d\$ - 将光标至行尾的字符剪切入剪贴板
- range y - 块复制
- range d - 块剪切

# DAY\_14 输入输出

C 语言无 I/O 语句，I/O 操作由函数实现

```
#include <stdio.h>
```

一、字符输出函数

- 格式: putchar(c)

参数: c 为字符常量、变量或表达式

功能: 把字符 c 输出到显示器上

返值: 正常, 为显示的代码值

➤代码实现参考 putchar.c

```
#include <stdio.h>

int main()
{
    int a = 65;
    char ch = 'B';
    putchar(a);
    putchar('\n');
    putchar(ch);
    return 0;
}
```

二、格式输出函数

- 格式: printf("格式控制串", 输出表)

功能: 按指定格式向显示器输出数据

输出表: 要输出的数据

格式控制串: 包含两种信息

    格式说明: %[修饰符]格式字符, 用于指定输出格式

    普通字符: 原样输出

- 格式字符

格式符	作用
i, d	十进制整数
x,X	十六进制无符号整数
o	八进制无符号整数
u	无符号十进制整数
c	单一字符
s	字符串
e,E	指数形式浮点小数
f	小数形式浮点小数
g	e和f中较短一种
%%	百分号本身

修饰符	功能
m	输出数据域宽，数据长度<m，左补空格；否则按实际输出
.n	对实数，指定小数点后位数（四舍五入）
-	对字符串，指定实际输出位数
-	输出数据在域内左对齐（缺省右对齐）
+	指定在有符号数的正数前显示正号（+）
0	输出数值时指定左面不使用的空位置自动填0
#	在八进制和十六进制数前显示前导0，0x
l	在d, o, x, u前，指定输出精度为long型
l	在e, f, g前，指定输出精度为double型

### 三、字符输入函数

- 格式：getchar()

功能：从键盘读一字符

返回值：正常，返回读取的代码值；出错或结束键盘输入，返回-1(Ctrl+D)

➤代码实现参考 getchar.c

```
#include <stdio.h>

int main()
{
    int c;
    printf("Enter a character: ");
    c = getchar();
    printf("%c--%d->hex%x\n", c, c, c);
    return 0;
}
```

### 四、格式输入函数

- 格式：scanf(“格式控制串”，&地址表)

功能：按指定格式从键盘读入数据，存入地址表指定

返回值：正常，返回输入数据个数

地址表：变量的地址，常用取地址运算符&

➤代码实现参考 scanf1.c

```
#include <stdio.h>
int main()
{
    char ch;
    printf(">");
    scanf("%c", &ch);
    printf("%c\n", ch);
    return 0;
}
```

- 目前，`scanf` 支持的格式字符很多，如下：

格式符号	作用
i, d	十进制整数
x, X	十六进制无符号整数
o	八进制无符号整数
u	无符号十进制整数
c	单一字符
s	字符串
e	指数形式浮点小数
f	小数形式浮点小数

- `scanf` 函数支持的格式说明符可以带一下修饰符

修饰符	功能
h	用于d, o, x前，指定输入为short型整数
l	用于d, o, x前，指定输入为long型整数
	用于e, f前，指定输入为double型实数
m	指定输入数据宽度，遇空格或不可转换字符结束
*	抑制符，指定输入项读入后不赋给变量

➤ 代码实现参考 `scanf2.c`

```
#include <stdio.h>

int main()
{
    int ch, n;
    long m;
    printf(">");
    n = scanf("%x%ld", &ch, &m);
    printf("%d %d %ld\n", n, ch, m);
    return 0;
}
```

## 五、输入函数留下的“垃圾”

- 使用输入函数可能会留下垃圾

```
int x;
char ch;
scanf("%d", &x);
scanf("%c", &ch);
printf("x=%d, ch=%d\n", x, ch);
```

在实现函数时，敲下回车，`scanf` 会记录回车而不是输入的第二个字符，导致输出结果与预期不符

## ●方法 1：用 getchar() 清除

```
int x;
char ch;
scanf("%d", &x);
getchar();
ch = getchar();
printf("x=%d, ch=%c\n", x, ch);
```

## ●方法 2：用格式串中空格或 “%\*c” 来“吃掉”

```
int x;
char ch;
scanf("%d", &x);
scanf(" %c", &ch); 或 scanf("%*c%c", &ch);
```

## 六、字符串输入函数

## ●格式：char \* gets(char \*s)

功能：从键盘输入一以回车结束的字符串放入字符数组中，并自动加 ‘\0’

说明 1：输入串长度应小于字符数组维数

说明 2：与 scanf 函数不同，gets 函数并不以空格作为字符串输入结束的标志

➤代码实现参考 gets.c

```
#include <stdio.h>
#define N 5
int main()
{
    char s[5];

    gets(s);
    printf("%s\n", s);
    return 0;
}
```

注意，这样写的代码，最多只能输入 4 个字符串，N=5 是因为字符与字符串之间的结束保留‘\0’，若超出 4 个字符串，则会提示栈溢出。

## 七、字符串输出函数

## ●格式：int puts(const char \*s)

功能：向显示器输出字符串（输出完会自动换行）

说明：字符数组必须以 ‘\0’ 结束

➤代码实现参考 puts.c

```
#include <stdio.h>
#define N 5
int main()
{
    char s[5];

    gets(s);
    printf("s:%s", s);
    printf("***\n");
    puts("$$$");
    puts(s);
    return 0;
}
```

# DAY\_15 控制语句-if

## 一、if-else 语句

### ● if 语句概述

```
if (表达式)
    语句块 1
else
    语句块 2
```

### ● 常见形式

#### ① 简化形式

if (表达式) 语句块 例如: if ( $x > y$ ) printf("%d\n")

#### ② 阶梯形式

```
if(表达式 1) 语句块 1
else if(表达式 2) 语句块 2
else if(表达式 3) 语句块 3
else if(表达式 4) 语句块 4
...
else 语句块 n
```

#### ③ 嵌套形式

```
if 语句的嵌套
if ()
    if () 语句块 1
    else 语句块 2
else
    if () 语句块 3
    else 语句块 4
```

➤ 代码实现参考 `control_statement_if.c`

```
#include <stdio.h>

int main()
{
    float score;
    printf("please input your score:");
    scanf("%f", &score);

    if (score < 0 || score > 100) {
        printf("not in [0, 100]\n");
        return -1;
    } else if (score >= 90) {
        /*
         * 这里可以不用框定<= 100 的情况,
         * 因为依旧在上一个 if 的范围
         * 省略以精简代码
        */
    }
}
```

```
/*
printf("A-excellent\n");
} else if (score >= 70) { // [70, 90)
    printf("B-good\n");
} else if (score >= 60) { // [60, 70)
    printf("C-pass\n");
} else { // [0, 60)
    printf("D-not pass\n");
}
return 0;
}
```

●注意：

(1)语句块：当有若干条语句时，必须用{...}括起来。

(2)表达式：

a) 一般情况下为逻辑表达式或关系表达式

如：if (a == b && x == y) printf("a = b, x = y");

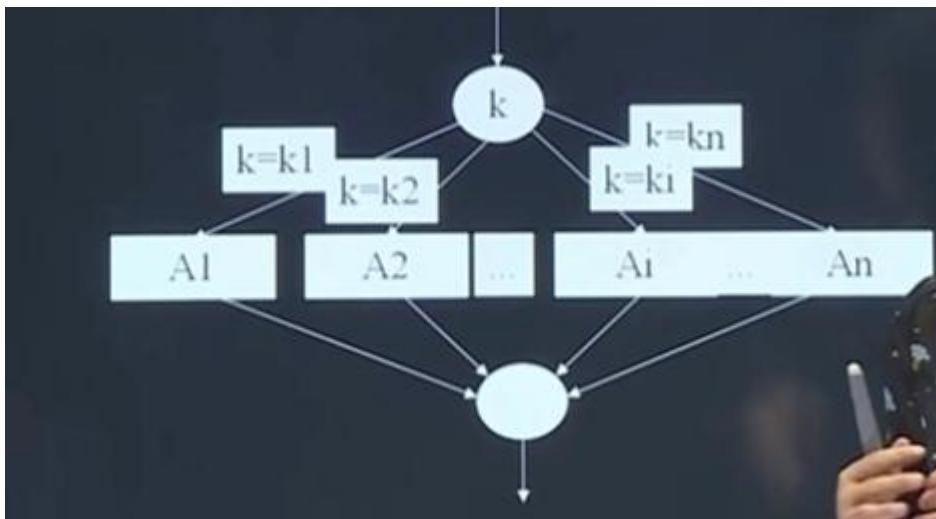
b) 也可以是任意类型(包括整型、实型、字符型、指针类型)

如：if ('a') printf("%d", 'a');

c) 请区分 if (x = 1) 与 if (x == 1) 的不同。

# DAY\_16 控制语句-switch

## 一、多分支语句



## 二、switch 语句的基本形式

- 表达式

```

switch (表达式) {
    case 常量表达式 1:
        语句块 1;
        break;
    case 常量表达式 2:
        语句块 2;
        break;
    .....
    case 常量表达式 n:
        语句块 n;
        break;
    default:
        语句块 n+1;
        break;
}
  
```

- switch 语句的使用：

1. 每个常量表达式的值必须各不相同，否则将会出现矛盾。
2. 当表达式的值与 case 后面的常量表达式值相等时，就执行此 case 后面的语句。
3. switch 中的表达式可以是整型、字符型表达式或枚举。
4. case 常量：只起语句标号的作用。

- break 语句用于强行跳出 switch 体，一般每个 case 后面应有一个 break 语句， default 分支后的 break 可以省略。

- 多个 case 可以执行一组语句。

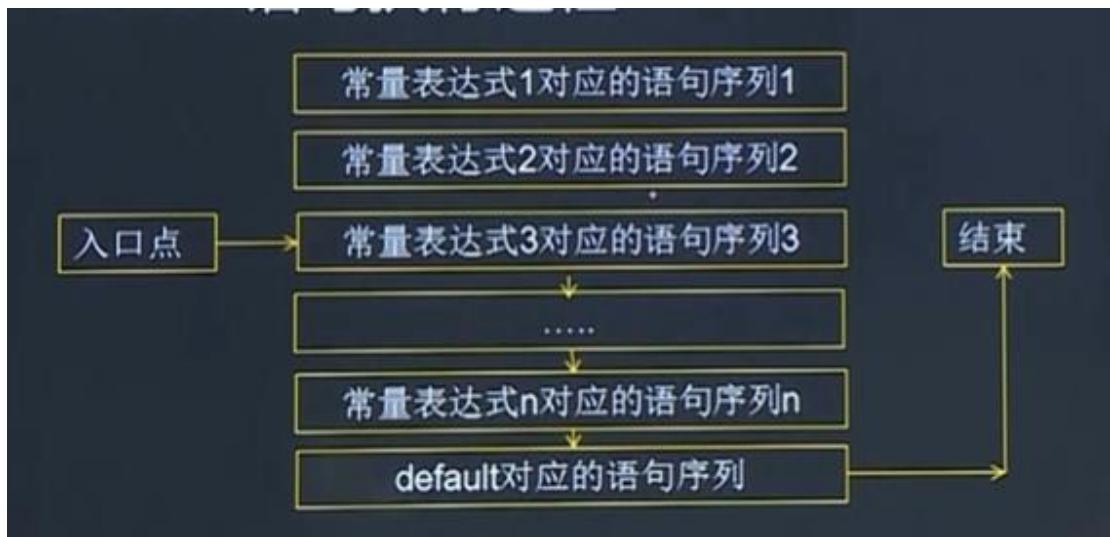
➤代码实现参考 control\_statement\_switch.c

```
#include <stdio.h>

int main()
{
    /* switch 语句块可以是整型、字符型表达式或枚举 */
    //int tmp = 1;
    char tmp = 'a';

    switch (tmp) {
        case 1:
            printf("tmp is 1\n");
            break;
        case 2:
            printf("tmp is 2\n");
            break;
        case 3:
            printf("tmp is 3\n");
            break;
        default:
            printf("tmp is not 1, 2, 3, 4\n");
            break;
    }
    return 0;
}
```

三、switch 语句的执行过程



# DAY\_17 循环语句-goto

## 一、goto语句构成循环

- 例子，求1到100的和

```
int main()
{
    int i = 1, sum = 0;
    if (i <= 100) {
        sum = sum + i;
        i++;
    }
    printf("%d", sum);
    return 0;
}
```

如何利用 goto语句构成循环？

- 代码实现参考 goto1.c

```
#include <stdio.h>

int main(int argc, const char * argv[])
{
    int i = 1;
    int sum = 0;

loop:
    if (i <= 100) {
        sum += i;
        i++;
        goto loop;
    }

    /* 注意此时 i 的值为 101， 打印时需 i - 1 满足题意 */
    printf("1 + 2 + ... + %d = %d\n", i - 1, sum);
    return 0;
}
```

## 二、集中于一处退出函数

- 循环结构程序

当函数有很多个出口，使用 goto把这些出口集中到一处是很方便的，特别是函数中有许多重复的清理工作的时候。理由是：

- 1.无条件跳转易于理解
- 2.可以减少嵌套
- 3.可以避免那种忘记更新某一个出口点的问题
- 4.算是帮助编译器做了代码优化

### 三、goto 语句用法

- 给定一个 main 函数

```
int main()
{
    /* 在函数调用前先留置标志位 loop, 名称可自定义*/
    loop:
        statatments;

    /* goto 跳转到 loop */
    goto loop;

    return 0;
}
```

# DAY\_18 循环语句-while

## 一、while 语句构成循环

- 基本形式

```
while (表达式) {
    statatments;
}
```

- 例子，求 1 到 100 的和

▷ 代码实现参考 while1.c

```
#include <stdio.h>

int main(int argc, const char * argv[])
{
    int i = 1;
    int sum = 0;

    while (i <= 100) {
        sum += i;
        i++;
    }

    /* 注意此时 i 的值为 101， 打印时需 i - 1 满足题意 */
    printf("1 + 2 + ... + %d = %d\n", i - 1, sum);
    return 0;
}
```

## 二、do-while 语句构成循环

- 基本形式

```
do {
    statatments;
} while(表达式);
```

- 例子，求 1 到 100 的和

▷ 代码实现参考 do-while1.c

```
#include <stdio.h>

int main(int argc, const char * argv[])
{
    int i = 1;
    int sum = 0;
```

```

do {
    sum += i;
    i++;
} while (i <= 100);

/* 注意此时 i 的值为 101， 打印时需 i - 1 满足题意 */
printf("1 + 2 + ... + %d = %d\n", i - 1, sum);
return 0;
}

```

### 三、程序举例

- 打印出所有的“水仙花”数。“水仙花”数是指一个3位数，其各个位数字立方和等于该数本身。

➤ 代码实现参考 `shuixianhua.c`

```

#include <stdio.h>
/*
* 记得 gcc 编译时链接 math 库
* gcc shuixianhua.c -Wall -lm
*/
#include <math.h>

int main(int argc, const char * argv[])
{
    int n = 100;
    /*
     * a 代表百位, b 代表十位, c 代表个位
     * a = n / 100 商的结果即为百位
     * b = n / 10 % 10 商后余数即为十位
     * c = n % 10 余数即为个位
     */
    int a, b, c;

    while (n <= 999) {
        a = n / 100;
        b = n / 10 % 10;
        c = n % 10;

        //      if (a*a*a + b*b*b + c*c*c == n) {
        if (pow(a, 3) + pow(b, 3) + pow(c, 3) == n) {
            printf("%d\n", n);
        }
        n++;
    }
}

```

```
    return 0;  
}
```

注：辅助语句-break; continue 后续再讲！

# DAY\_19 循环语句-for

## 一、一般形式

```
for (expression1; expression2; expression3) {
    statements;
}
```

### ●执行过程

- 1.先求解表达式 1;
- 2.求解表达式 2, 若为真, 则执行循环体, 然后执行步骤 3; 若为假, 则执行步骤 5;
- 3.求解表达式 3;
- 4.转回执行步骤 2;
- 5.执行 for 下面的语句。

## 二、for 语句构成循环

### ●例如:

```
for (i = 1; i <= 100; i++) {
    sum = sum + i;
}
```

- 表达式 1 可省略, 但循环之前应给循环变量赋初值。
- 表达式 2 可省略, 但将陷入死循环。
- 表达式 3 可省略, 但在循环体中增加使循环变量值改变的语句。

## 三、for 语句三种形式

➤代码实现参考 for1.c

```
#include <stdio.h>

int main(int argc, const char * argv[])
{
    int i, sum;
    sum = 0;

    for (i = 1; i <= 100; i++) {
        sum += i;
    }
    printf("sum = %d\n", sum);
    return 0;
}
```

➤代码实现参考 for2.c

```
#include <stdio.h>

int main(int argc, const char * argv[])
{
```

```

int i, sum;
sum = 0;
i = 1;

for (; i <= 100; i++) {
    sum += i;
}
printf("sum = %d\n", sum);
return 0;
}

```

➤代码实现参考 for3.c

```

#include <stdio.h>

int main(int argc, const char * argv[])
{
    int i, sum;
    sum = 0;
    i = 1;

    for (; i <= 100;) {
        sum += i;
        i++;
    }
    printf("sum = %d\n", sum);
    return 0;
}

```

四、for 语句实现九九乘法表

➤代码实现参考 chengfabiao.c

```

#include <stdio.h>

int main(int argc, const char * argv[])
{
    /* i 代表行数， j 代表列数 */
    int i, j;

    for (i = 1; i <= 9; i++) {
        for (j = 1; j <= i; j++) {
            printf("%d*%d=%d\t", j, i, i*j);
            /* 先是 i = 1, j = 1, 显示 1*1
             * 然后 i = 2, j = 1, 显示 1*2
             * 然后 i = 2, j = 2, 显示 2*2
             * 这样就显示了 1*2=2 2*2=4 的排版
        }
    }
}

```

```

        * 因此 j 与 i 要调换位置
    */
}

puts(" ");
}

return 0;
}

```

## 五、for 语句实现字母图案

- 思考：利用 for 循环打印下面的图案？（大小写）

```

F
_FE
__FED
___FEDC
____FEDCB
_____FEDCBA

```

➤代码实现参考 zimutuan.c

```
#include <stdio.h>

int main(int argc, const char * argv[])
{
    /* i 代表行数， j 代表列数 */
    int i, j;
    char ch;
    printf("Please input a letter:");
    scanf("%c", &ch);

    for (i = 0; i <= ch - 'A' ; i++) {
        for (j = 0; j < i; j++) {
            putchar('_');
        }
        for (j = 0; j <= i; j++) {
            putchar(ch - j);
        }
        puts(" ");
    }

    return 0;
}
```

# DAY\_20 辅助控制语句

## 一、break 语句

- 用于从循环体内跳出循环体，即提前结束循环。

① break 只能用在循环语句和 switch 语句中。

```
for (r = 1; r <= 10; r++) {
    area = pi * r * r;
    if (area > 100) {
        break;
    }
    printf("%f", area);
}
```

➤ 代码实现参考 break1.c

```
#include <stdio.h>
#define pi 3.14

int main()
{
    int r, area;
    for (r = 1; r <=10; r++) {
        area = pi * r * r;
        if (area > 100) {
            break;
        }
        printf("%d %d\n", r, area);
    }
    return 0;
}
```

## ● 素数

1. 在一个大于 1 的自然数中，除了 1 和此整数自身外，没法被其他自然数整除的数。

换句话说，只有两个正因数（1 和自己）的自然数即为素数。

1 大但不是素数的数称为合数。1 和 0 既非素数也非合数。

2. 对一个大于或等于 3 的正整数，判断它是不是一个素数

S1: 输入整数 n 的值

S2: i = 2

S3: n 被 i 除，得余数 r

S4: if r = 0, 则打印 n “不是素数”，算法结束 执行 S5

S5: i + 1 → i

S6: if i ≤ √n, 返回 S3; 否则打印 n “是素数”；然后算法结束

➤代码实现参考 break2.c

```
#include <stdio.h>
#include <math.h>

int main()
{
    int n;
    int i;

    printf("input n(>3):");
    scanf("%d", &n);

    i = 2;
    while (i <= sqrt(n)) {
        if (n % i == 0) {
            break;
        }
        i++;
    }
    if (i <= sqrt(n)){
        printf("%d is not a prime number.\n", n);
    } else {
        printf("%d is a prime number.\n", n);
    }
    return 0;
}
```

## 二、continue 语句

- 结束本次循环，接着判定下一次是否执行循环

- 注意 continue 与 break 的区别

- 1.continue 直接结束本次循环，而 break 终止本层循环

```
for (n = 100; n <= 200; n++) {
    if (n % 3 == 0) {
        continue; //比如到 102 时，他不满足条件，这次的循环将终止，打印不出；
                   //继续下一个循环，此时 103 满足条件，可以打印
    }
    printf( "%d", n);
}
```

➤代码实现参考 continue1.c

```
#include <stdio.h>

int main()
{
    int n;
    for (n = 100; n <= 200; n++) {
        if (n % 3 == 0) {
            continue;
        }
        printf("%d\n", n);
    }
    return 0;
}
```

# DAY\_21 一维数组

## ●数组概述

### 一、数组

①构造数据类型之一

②数组是具有一定顺序关系的若干个变量的集合，组成数组的各个变量称为数组的元素

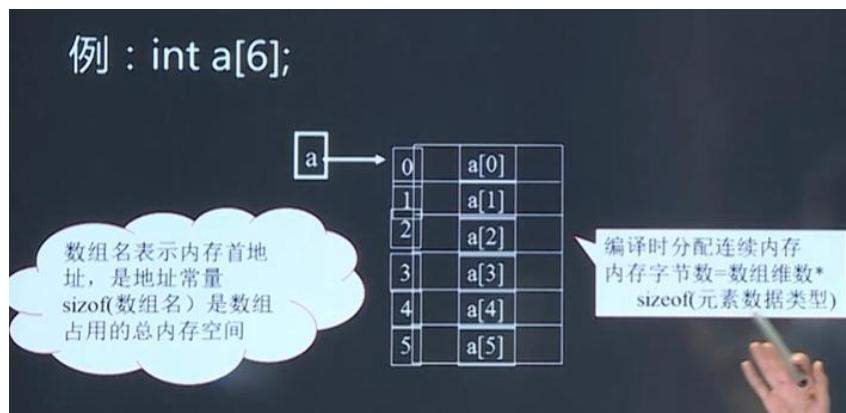
③数组中各元素的数据类型要求相同，用数组名和下标确定。数组可以是一维的，也可以是多维的

### 二、一维数组

①所谓一维数组是指只有一个下标的数组。它在计算机的内存中是连续存储的。

②C语言中，一维数组的说明一般形式如下：

<存储类型><数据类型><数组名>[<表达式>];



➤代码实现参考 array1.c

```
#include <stdio.h>

int main()
{
    int a[6], i;

    for (i = 0; i < 6; i++) {
        printf("%p\n", &a[i]);
        /*
         * 对于指针来说，地址是连续的
         * %p 主要用于 printf 等格式化输出函数中，
         * 用来以十六进制的形式输出指针变量所指向的内存地址
        */
    }
    //a = a + 1; //数组名代表着地址，不可以变动
    printf("%p\n", a);
    printf("%d\n", sizeof(a));
}
```

```

    return 0;
}

```

### 三、注意事项

①C 语言对数组不作越界检查，使用时要注意（不要依赖编译器解决所有问题！！）

```
int a[5]; a[5] = 10
```

②关于用变量定义数组维数

```
int i = 15; int a[i]
```

### ●一维数组的引用

一、数组必须先定义，后使用

二、只能逐个引用数组元素，不能一次引用整个数组

三、数组元素表示形式：数组名[下标]

其中：下标可以是常量或整型表达式

例 int a[10];

```
printf("%d", a); (X)
```

```
for(j=0;j<10;j++)
printf("%d\t", a[j]); (V)
```

### ●一维数组的初始化

①初始化方式：在定义数组时，为数组元素赋初值

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

②说明

1. 数组不初始化，其元素值为随机数

2. 对 static 数组元素不赋初值，系统会自动赋以 0 值

3. 只给部分数组元素赋初值

```
static int a[5];
```

等价于 : a[0]=0; a[1]=0; a[2]=0; a[3]=0; a[4]=0;

```
int a[5]={6,2,3};
```

等价于 : a[0]=6; a[1]=2; a[2]=3; a[3]=0; a[4]=0;

```
int a[3]={6,2,3,5,1}; (X)
```

int a[]={1,2,3,4,5,6};//编译系统根据初值个数确定

数组维数

➤代码实现参考 array2.c

```
#include <stdio.h>

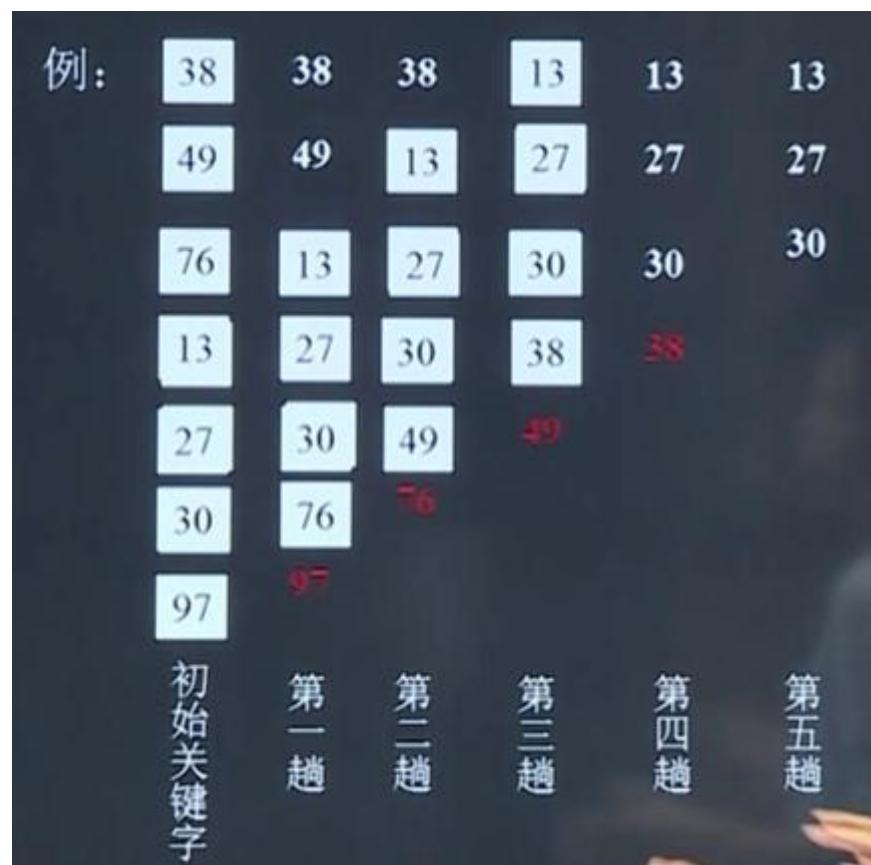
int main()
{
    int a[6] = {1, 4, 5, 4, 6, 7}, i, n;
    int b[] = {3, 4, 7, 8, 1, 0};

    n = sizeof(a) / sizeof(int); //计算数组元素个数
    for (i = 0; i < n; i++) {
        printf("%p %d\n", &a[i], a[i]);
        printf("%p %d\n", &b[i], b[i]);
    }
    //a = a + 1; //数组名代表着地址，不可以变动
    printf("%p\n", a);
    printf("%ld\n", sizeof(a));
    return 0;
}
```

●程序举例

一、冒泡排序

它重复地走访过要排序的数列，一次比较两个元素，如果顺序错误就交换。走访数列的工作是重复地进行直到没有再需要交换，也就是说该数列已经排序完成。



➤代码实现参考 maopao.c

```
#include <stdio.h>

int main(int argc, char const *argv[])
{
    int a[] = {49, 38, 97, 76, 13, 27, 30}, n, i, j, t;
    n = sizeof(a) / sizeof(int);

    for (i = 0; i < n-1; i++) { //控制比对轮数，最多比对n-1轮
        for (j = 0; j < n-1-i; j++) { //寻找j与i的关系，存在j+1<=n-1(-i)
            if (a[j] > a[j+1]) { //升序交换
                t = a[j]; //保留 a[j]
                a[j] = a[j+1]; //a[j]与 a[j+1]交换
                a[j+1] = t; //a[j+1]赋值为保留的 a[j]
            }
        }
    }

    for (i = 0; i < n; i++) {
        printf("%d ", a[i]);
    }
    puts("");
    return 0;
}
```

## DAY\_22 二维数组

### 一、二维数组的定义

1. 定义方式：（声明时列数不能省略，行数可以）

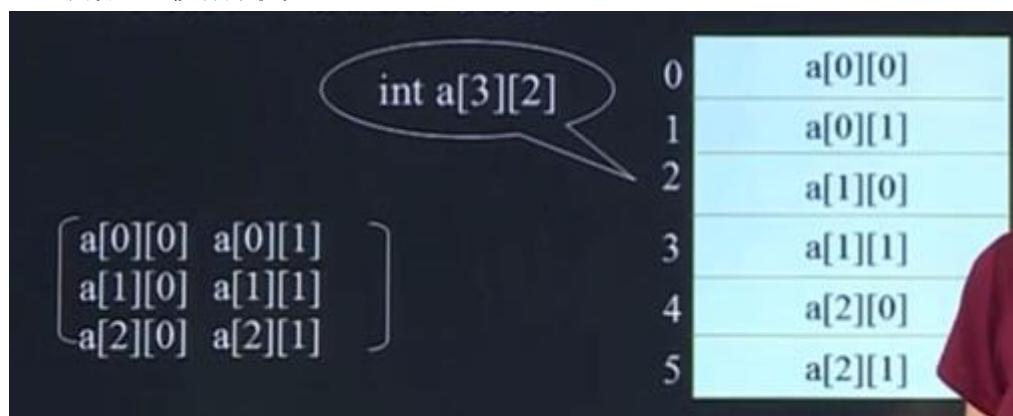
数据类型 数组名[常量表达式][常量表达式];

例：  
 int a[3][4];  
 float b[2][5];  
 int c[2][3][4];

### 二、数组元素的存放顺序

1. 原因：内存是一维的

2. 二维数组：按行序优先



➤ 代码实现参考 array3.c

```
#include<stdio.h>

int main(int argc, char *argv[])
{
    /* 验证二维数组的连续性 */
    int a[2][3];
    int i,j;

    for (i = 0; i < 2; i++) {
        for (j = 0; j < 3; j++) {
            printf("%p ", &a[i][j]);
        }
        putchar('\n');
    }
    return 0;
}
```

### 三、二维数组元素的引用

1.形式：数组名[下标][下标]

2.二维数组元素的初始化

①分行初始化

②按元素排列顺序初始化

**例** int a[2][3]={ {1,2,3}, {4,5,6} };

1	2	3	4	5	6
---	---	---	---	---	---

a[0][0] a[0][1] a[0][2] a[1][0] a[1][1] a[1][2]

➤代码实现参考 array4.c

```
#include<stdio.h>

int main(int argc, char *argv[])
{
    /* 元素全部初始化 */
    //int a[2][3] = {{1, 6, 9}, {2, 8, 5}};

    /* 元素部分初始化 */
    //int a[2][3] = {{1, 6}, {2, 8, 5}};

    /* 元素部分初始化 */
    int a[2][3] = {{1, 6}};
    int i,j;

    for (i = 0; i < 2; i++) {
        for (j = 0; j < 3; j++) {
            printf("%d ", a[i][j]);
        }
        putchar('\n');
    }
    return 0;
}
```

#### 四、程序举例

##### 1. 打印杨辉三角的前十行

```
1  
1 1  
1 2 1  
1 3 3 1  
1 4 6 4 1  
1 5 10 10 5 1  
...
```

➤代码实现参考 yanghuisanjiao.c

```
#include<stdio.h>  
  
int main(int argc, char *argv[])  
{  
    /* 杨辉三角前 10 行 */  
    int a[10][10] = {{0}};  
    int i, j;  
  
    for (i = 0; i < 10; i++) {  
        a[i][0] = 1;  
        for (j = 1; j <= i; j++) {  
            a[i][j] = a[i-1][j-1] + a[i-1][j];  
        }  
    }  
  
    for (i = 0; i < 10; i++) {  
        for (j = 0; j <= i; j++) {  
            printf("%-8d ", a[i][j]);  
        }  
        putchar('\n');  
    }  
  
    return 0;  
}
```

2.有一个 $3 \times 4$ 的矩阵，要求输出其中值最大的元素的值，以及它的行号和列号。这是一个关于矩阵运算的问题描述，主要任务是在给定的3行4列矩阵中找出数值最大的元素，并确定该元素所在的行号和列号。

➤代码实现参考 array5.c

```
#include<stdio.h>

int main(int argc, char *argv[])
{
    int a[2][3] = {{2, 5, 8}, {21, 56, 9}};
    int i, j, row, column;

    row = column = 0;

    /* 寻找最大值 */
    for (i = 0; i < 2; i++) {
        for (j = 0; j < 3; j++) {
            if (a[row][column] < a[i][j]) {
                row = i;
                column = j;
            }
        }
    }

    for (i = 0; i < 2; i++) {
        for (j = 0; j < 3; j++) {
            printf("%5d ", a[i][j]);
        }
        putchar('\n');
    }

    printf("max = %d %d %d\n", a[row][column], row, column);
    return 0;
}
```

# DAY\_23 字符数组和字符串

1. 掌握字符数组的用法

2. 字符数组

1) 字符数组的定义

- 本质：元素数据类型为字符类型的数组
- 声明形式：char c[10]（一维）或 char ch[3][4]（二维）
- 特点：由若干个字符型变量构成的有序集合

2) 字符数组的初始化

• 逐个字符赋值

◦ 方法：通过单引号逐个为数组元素赋值

◦ 注意事项：

- 编译器不检查数组越界，需自行控制
- 部分赋值时，未赋值的元素自动补零（'\0'）
- 示例 1：char ch[5] = {'H', 'e', 'l', 'l', 'o'}  
| ch[0] | ch[1] | ch[2] | ch[3] | ch[4] |  
| 'H' | 'e' | 'l' | 'l' | 'o' |

• 用字符串常量

◦ 方法：使用双引号括起的字符串常量初始化

◦ 特点：

- 字符串常量会自动添加'\0'作为结束符
- 等价于用字符序列初始化

◦ 特殊形式：

- char ch[6] = {"Hello"}
- char ch[6] = "Hello"
- char ch[] = "Hello"（自动计算长度）
- | ch[0] | ch[1] | ch[2] | ch[3] | ch[4] | ch[5] |  
| 'H' | 'e' | 'l' | 'l' | 'o' | '\0' |

3) 应用案例(字符串数组赋值访问)

➤ 代码实现参考 char\_array1.c

```
#include <stdio.h>
int main()
{
    char arr1[] = {'a', 'b', 'c'};
    char arr2[6] = {'a', 'b', 'c'};
    int i, n;

    n = sizeof(arr1) / sizeof(char);
    for (i = 0; i < n; i++) {
        putchar(arr1[i]);
    }
    putchar('\n');
```

```

n = sizeof(arr2) / sizeof(char);
for (i = 0; i < n; i++) {
    putchar(arr2[i]);
}

return 0;
}

```

## ◦ 关键发现:

- 使用%s输出字符数组时，必须确保数组以'\0'结尾
- 未以'\0'结尾的字符数组用%s输出会导致越界访问
- 相邻定义的字符数组可能在内存中连续存储

## ◦ 验证方法:

- 打印数组地址验证内存连续性
- 使用sizeof计算数组实际长度
- 对比逐个字符输出与%s输出的区别

## ◦ 安全建议:

- 推荐使用逐个字符访问的方式处理字符数组
- 使用%s输出时确保数组是合法的字符串（以'\0'结尾）
- 注意数组越界问题，编译器不会自动检查

## ➤ 代码实现

```

#include <stdio.h>
int main()
{
    char arr1[] = {'a', 'b', 'c'};
    char arr2[6] = {'a', 'b', 'c'};

    printf("arr1:%s %p\n", arr1, &arr1[2]);
    printf("arr2:%s %p\n", arr2, arr2);

#ifndef _MSC_VER
    int i, n;

    n = sizeof(arr1) / sizeof(char);
    for (i = 0; i < n; i++) {
        putchar(arr1[i]);
    }
    putchar('\n');

    n = sizeof(arr2) / sizeof(char);
    for (i = 0; i < n; i++) {
        putchar(arr2[i]);
    }
#endif
}

```

```

    putchar('\n');
#endif

    return 0;
}

```

○ 验证结果:

- 相邻数组确实在内存中连续存储
- 第一个数组未以'\0'结尾导致%s输出时读取到第二个数组内容

### 3.二维数组

#### 3.1 二维数组初始化

➤ 代码实现

二维字符数组的初始化示例: char arr[5][6] = {{".", "\*", "\0", "\0", "\0"}, {".", "\*", "\*", "\0", "\0"}, {"\0"}, {"\*", ".", "\*", "\0"}, {"\*", "\*", ".", "\*", {"\0", "\0", "\0", "\0", "\0"}}, {"\0", "\0", "\0", "\0", "\0"}};

#### 3.2 二维字符数组的菱形图案

二维字符数组可以用于创建各种图案,如菱形。通过双引号简化字符数组的赋值。

➤ 代码实现

菱形的创建示例: char arr[5][6] = {{".", "\*", "\0", "\0", "\0"}, {".", "\*", "\*", "\0", "\0"}, {"\0"}, {"\*", "\*", ".", "\*", {"\0", "\0", "\0", "\0", "\0"}}, {"\0", "\0", "\0", "\0", "\0"}};

**例** char fruit[5][7]={"Apple","Orange",
"Grape","Pear","Peach"};

fruit[0]	A	p	p	l	e	\0	\0
fruit[1]	O	r	a	n	g	e	\0
fruit[2]	G	r	a	p	e	\0	\0
fruit[3]	P	e	a	r	\0	\0	\0
fruit[4]	P	e	a	c	h	\0	\0

#### 3.3 二维数组计算大小

➤ 代码实现参考 array6.c

```

#include <stdio.h>

int main(int argc, char *argv[])
{
    char fruit[][11] = {"Banana", "Apple", "Strawberry",
"Watermelon"};
    int i, j, n, m;

    /* 计算二维数组行数: 整个数组大小除以每一行大小 */
    n = sizeof(fruit) / sizeof(fruit[0]);

```

```
/* 计算二维数组列数：每一行大小除以每一个元素大小 */
m = sizeof(fruit[0]) / sizeof(char);

/* 遍历二维数组方法一：putchar */
for (i = 0; i < n ; i++) {
    for (j = 0; j < m; j++){
        putchar(fruit[i][j]);
    }
    /* 输出完一列数组后换行 */
    putchar('\n');
}

/* 分隔 */
putchar('\n');

/* 遍历二维数组方法一：printf 与%s */
for (i = 0; i < n ; i++) {
    /*
        当数组元素定为 10 时，此方法会导致\n 无地方归纳
        使用方法一可避免
    */
    printf("%s\n", fruit[i]);
    /* 输出完一列数组后换行 */
}

return 0;
}
```

# DAY\_24 字符串逆序输出

通过分析多种思路，选择主流思路进行编程。第一种思路：get 获取字符串不变，逆序显示。第二种思路：get 获取字符串改变，原样输出。

一、get 获取字符串不变，逆序显示

➤ 代码实现参考 array7.c

```
#include <stdio.h>

#define N 20
int main(int argc, char *argv[])
{
    //char arr[] = {"Weclome"};
    char arr[N] = {0};
    /*
        写 0 的目的就是防止
        未超出数组部分出现乱码
    */
    int i, n;

    /* 从键盘输入一个字符串 */
    printf("请输入一个字符串: ");
    gets(arr);

    /* 从后往前遍历数组 */
    n = sizeof(arr) / sizeof(char);
    for (i = n - 1; i >= 0; i--) {
        putchar(arr[i]);
    }
    putchar('\n');

    return 0;
}
```

拓展：计算字符串长度 strlen 函数

在使用该函数前，需要引用头文件 string.h

➤ 代码实现参考 Calculate\_array.c

```
#include <stdio.h>
#include <string.h>
#define N 20
int main(int argc, char *argv[])
{
```

```

//char arr[] = {"Weclome"};
char arr[N];
/*
    使用 strlen 函数可以避免
    未超数组长度乱码问题
*/
int i, n;

/* 从键盘输入一个字符串 */
printf("请输入一个字符串: ");
gets(arr);

/* 从后往前遍历数组 */
n = strlen(arr);
for (i = n - 1; i >= 0; i--) {
    putchar(arr[i]);
}
putchar('\n');

return 0;
}

```

## 二、get 获取字符串改变，原样输出

➤ 代码实现参考 array8.c

```

#include <stdio.h>
#include <string.h>
#define N 20
int main(int argc, char *argv[])
{
    char arr[N];
    char ch;
    /*
        使用 strlen 函数可以避免
        未超数组长度乱码问题
    */
    int i, j, n;

    printf("请输入一个字符串: ");
    gets(arr);
    n = strlen(arr);

    /* i 记录第一个元素， j 记录最后一个元素 */
    i = 0;

```

```
j = n - 1;

/* i 与 j 未相遇 */
while (i < j) {
    /* i 与 j 进行交换 */
    ch = arr[i];
    arr[i] = arr[j];
    arr[j] = ch;
    /* i 与 j 都往中间走, i 增加, j 减少 */
    i++;
    j--; /* gets 默认在字符串末尾添加'\0', n = 6 */
}
printf("%s\n", arr);

return 0;
}
```

特别提醒：

**scanf 与 get 区别：**

1.gets 可以读取带空格的字符串， scanf 则不能。

2.scnf 使用%**s** 读取字符串时，空格作为结束符。

3.scnf 需要存储空格时，将 scnf 当作字符数组，逐字符输入，使用%**c** 输出。

# DAY\_25 字符串函数

## 一、常见字符串函数

C 库中实现了很多字符串处理函数

```
#include <string.h>
```

几个常见的字符串处理函数

- ①求字符串长度的函数 `strlen`
- ②字符串拷贝函数 `strcpy`
- ③字符串连接函数 `strcat`
- ④字符串比较函数 `strcmp`

## 二、字符串长度函数 `strlen`

格式:`strlen(字符数组)`

功能:计算字符串长度

返值:返回字符串实际长度, 不包括'\0'在内

`\xhh` 表示十六进制数代表的符号

`\ddd` 表示 8 进制的

例:对于以下字符串, `strlen(s)`的值为

```
char s[10] = { 'A', '\0', 'B', 'C', '\0', 'D' };
```

```
char s[] = "\t\v\\0will\n"
```

```
char s[] = "\x69\141\n";
```

答案: 1 3 3

➤ 代码实现参考 `strlen1.c`

```
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
    char s1[10] = {'A', '\0', 'B', 'C', '\0', 'D' };
    char s2[] = {'A', '\0', 'B', 'C', '\0', 'D' };
    char s3[] = "\tab\n\t\vd\\e";
    char s4[] = "\x69\141\n";
    int n;

    n = strlen(s1);
    printf("%d\n", n);
    printf("%d\n", strlen(s1));
    printf("%d\n", sizeof(s2) / sizeof(char));
    /* 字符串所占长度, 含'\0' */
    printf("%d\n", strlen(s3));
    printf("%d\n", strlen(s4));
```

```

    puts(s4); /* x69 十六进制对应字母 i, 141 八进制对应字母 a */

    return 0;
}

```

### 三、字符串拷贝函数 strcpy

格式: strcpy(字符串 1,字符串 2)

功能:将字符串 2, 拷贝到字符串 1 中去

返值:返回字符串 1 的首地址

说明:

- ①字符串数组 1 必须足够大
- ②拷贝时'\0'一同拷贝

➤ 代码实现参考 strcpy.c

```

#include <stdio.h>
#include <string.h>

#define N 30
int main(int argc, char *argv[])
{
    char src[] = "makeru";
    char dest[N];

    strcpy(dest, src);
    puts(src);
    puts(dest);

    int i, n;
    i = 0;
    n = strlen(src);
    while (i <= n) {
        dest[i] = src[i];
        i++;
    }

    puts(src);
    puts(dest);

    return 0;
}

```

#### 四、字符串指定拷贝函数 strncpy

格式:strncpy(字符数组 1,字符串 2,指定长度)

➤ 代码实现参考 strncpy.c

```
#include <stdio.h>
#include <string.h>

#define N 30
int main(int argc, char *argv[])
{
    char src[] = "makeru";
    char dest[N] = ".com.cn";

    strncpy(dest, src, 4);
    puts(dest);

    return 0;
}
```

#### 五、字符串连接函数 strcat

格式:strcat(字符数组 1,字符数组 2)

功能:字符数组 2 拷贝到字符数组 1 的后面去

返值:返回字符数组 1 的首地址

说明:

- ①字符数组 1 必须足够大
- ②拷贝时,字符数组 1 的'\0'删除, 拷贝完后默认追加'\0'

➤ 代码实现参考 strcat.c

```
#include <stdio.h>
#include <string.h>

#define N 30
int main(int argc, char *argv[])
{
    char ch[N] = "People's Republic of ";
    char ch2[] = "China";

    strcat(ch, ch2);
    puts(ch);

    return 0;
}
```

## 六、字符串指定连接函数 strncat

格式:strcat(字符数组 1,字符数组 2,拷贝长度)  
 功能:字符数组 2 指定长度拷贝到字符数组 1 的后面去  
 返值:返回字符数组 1 的首地址

➤ 代码实现参考 strncat.c

```
#include <stdio.h>
#include <string.h>

#define N 30
int main(int argc, char *argv[])
{
    char ch[N] = "People's Republic of ";
    char ch2[] = "China";

    strncat(ch, ch2, 4);
    puts(ch);

    return 0;
}
```

## 七、字符串比较函数 strcmp

格式: strcmp(字符串 1, 字符串 2)  
 功能: 比较两个字符串的大小  
 返值:  
     - 如果字符串 1< 字符串 2, 返回负整数  
     - 如果字符串 1= 字符串 2, 返回 0  
     - 如果字符串 1> 字符串 2, 返回正整数

说明:

- ① 比较规则: 按 ASCII 码值逐个字符比较, 直到出现不同的字符或遇到"\0"
- ② 字符串参数可以是字符数组名或字符串常量
- ③ 比较的是字符串的内容, 不是字符串的长度

➤ 代码实现参考 strcmp.c

```
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
    char str1[] = "Hello";
    char str2[] = "Hello";
    int result1 = strcmp(str1, str2);
    if (result1 == 0) {
        printf("结论: 两个字符串相等\n");
    }
}
```

```

char str3[] = "Apple";
char str4[] = "Banana";
int result2 = strcmp(str3, str4);
if (result2 < 0) {
    printf("结论: 字符串 1 小于字符串 2\n");
}

char str5[] = "Orange";
char str6[] = "Apple";
int result3 = strcmp(str5, str6);
if (result3 > 0) {
    printf("结论: 字符串 1 大于字符串 2\n");
}

return 0;
}

```

## 八、字符串指定长度比较函数 strncmp

格式: strcmp(字符串 1, 字符串 2, 指定长度)

➤ 代码实现参考 strncmp.c

```

#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
    char str1[] = "Hello";
    char str2[] = "Helloaaa";
    int result1 = strncmp(str1, str2, 6);
    if (result1 == 0) {
        printf("结论: 指定的两个字符串相等\n");
    } else {
        printf("结论: 指定的两个字符串不相等\n");
    }

    return 0;
}

```

## 九、忽略字符串大小写比较函数 strcasecmp

➤ 代码实现参考 strcasecmp.c

```

#include <stdio.h>
#include <string.h>

```

```

int main(int argc, char *argv[])
{
    char str1[] = "HELLO";
    char str2[] = "hello";

    printf("%d\n", strncmp(str1, str2, 5));
    printf("%d\n", strncasecmp(str1, str2, 5));

    return 0;
}

```

#### 十、查找字符函数 strchr 和 strrchr

**strchr:** 在字符串中查找指定的字符，返回其位置。

**strrchr:** 在字符串中查找指定的字符，返回其最后一次出现的位置。

使用时应确保字符串和字符的有效性。

➤ 代码实现参考 strchr.c

```

#include<stdio.h>
#include<string.h>

int main(int argc, char *argv[])
{
    char s1[] = "aba$s$fds";
    int ch;
    ch = '$';

    printf("%p %p \n", s1, strchr(s1, ch));
    printf("%p %p \n", s1, strrchr(s1, ch));

    printf("%ld\n", strchr(s1, ch) - s1);
    printf("%ld\n", strrchr(s1, ch) - s1);

    return 0;
}

```

#### 十一、查找字符串函数 strstr

**strstr:** 在长字符串中查找子字符串，返回其第一次出现的位置。

使用时应确保长字符串和子字符串的有效性。

➤ 代码实现参考 strstr.c

```

#include<stdio.h>
#include<string.h>

int main(int argc, char *argv[])

```

```

{
    char s[] = "how are you";
    char subs[] = "are";

    printf("%p\n", strstr(s, subs));
    printf("%ld\n", strstr(s, subs) - s);
    return 0;
}

```

## 十一、isalpha 函数、isupper 函数、islower 函数、isdigit 函数、isprint 函数

### 1.isalpha

函数原型: int isalpha(char c);

判断字符 c 是否为字母, 返回非零值或零。

注意, 需引用头文件#include <ctype.h>

### 2.isdigit

isdigit 函数用于检查字符是否为数字。

函数原型: int isdigit(char c);

判断字符 c 是否为数字 0 到 9, 返回非零值或零。

### 3.isprint 函数

isprint 函数用于检查字符是否为可打印图形。

函数原型: int isprint(char c);

判断字符 c 是否为可打印图形, 返回非零值或零。

### 4.isupper 函数

isupper 函数用于检查字符是否为大写字母。

函数原型: int isupper(char c);

判断字符 c 是否为大写字母, 返回非零值或零。

### 5.islower 函数

islower 函数用于检查字符是否为小写字母。

函数原型: int islower(char c);

判断字符 c 是否为小写字母, 返回非零值或零。

➤ 代码实现参考 isalpha.c

```

#include <stdio.h>
#include <string.h>
#include <ctype.h>

int main(int argc, char *argv[])
{
    int ch;

    while ((ch = getchar()) != EOF) { /* 从标准输入读取字符, 直到遇到文件结束符 EOF */
        /* 是否是字母 */
        if (isalpha(ch)) {

```

```

        if (isupper(ch)) { /* 是否是大小写字母 */
            printf("是大写字母: %c\n", ch);
        } else if (islower(ch)) {
            printf("是小写字母: %c\n", ch);
        }
    } else if (isdigit(ch)) { /* 是否是数字 */
        printf("是数字: %d\n", ch - '0');

        /*
         * 请注意, getchar() 函数读取的是字符, 而不是数字。
         * 当使用%d 格式符打印时, 会将字符的 ASCII 码值打印出来。
         * 若想打印数字本身, 需要将字符转换为整数。
         * 可以使用 ch - '0' 将字符转换为对应的整数, 或可以
         * 使用 ch - 48 将字符转换为整数。
        */
    }
}

return 0;
}

```

## 十二、大小写转换函数

`toupper` 和 `tolower` 函数用于将字符转换为大写或小写。

函数原型: `char toupper(char c);` 和 `char tolower(char c);`

转换后的值通过返回值返回, 如果转换不可能则返回原始字符。

➤ 代码实现参考 `toupper.c`

```

#include <stdio.h>
#include <string.h>
#include <ctype.h>

int main(int argc, char *argv[])
{
    int ch;

    while ((ch = getchar()) != EOF) { /* 从标准输入读取字符, 直到遇到文件结束符 EOF */
        if (isspace(ch)) {
            continue; /* 跳过空白字符, 直接进入下一次循环 */
        }
        /*
         * 思考:
         * 如果输入 A, 经转化后会提示不是字母.
         * 原因在于 getchar() 函数会读取包括空格在内的所有字符, 包括按下键盘后的换行符。
        */
    }
}

```

\* 因此，当输入 A 后，换行符会被读取，导致判断为不是字母。  
 \* 解决方法：  
 \* 可以在判断是否是字母之前，先判断是否是空白字符，如果是，直接跳过。

```
/*
 * 是否是字母 */
if (isalpha(ch)) {
    if (isupper(ch)) { /* 是否是大小写字母 */
        printf("是大写字母: %c\n", ch);
        ch = tolower(ch); /* 转换为小写字母 */
        printf("转换为小写字母: %c\n", ch);
    } else if (islower(ch)) {
        printf("是小写字母: %c\n", ch);
        ch = toupper(ch); /* 转换为大写字母 */
        printf("转换为大写字母: %c\n", ch);
    }
} else {
    printf("不是字母.\n");
    return 0;
}

return 0;
}
```

### 十三、判断传入的整数类型字符值函数 isspace

`isspace` 是 C 标准库<ctype.h>中的字符分类函数，接收 `int` 类型的字符值（通常为 `getchar()` 返回的字符或 `EOF`），核心作用是判断该字符是否为空白字符；若判定为空白字符则返回非 0 的整数（真值），否则返回 0（假值）。

该函数判定的空白字符包含多类“不可见但占据字符位置”的字符，其中' '（普通空格）是键盘空格键产生的常规空白；'\t' 为水平制表符，由 `Tab` 键触发；'\n' 是换行符，对应 `Enter` 键产生的换行操作；'\v' 为垂直制表符，用于垂直方向的字符对齐（实际开发中极少使用）；'\f' 是换页符，主要用于打印场景中触发换页动作（几乎不常用）；'\r' 是回车符，老式系统（如 `Windows`）中会与 '\n' 配合实现换行，单独使用仅让光标回到当前行的行首。

使用注意：函数参数必须为 `int` 类型（若用 `char` 类型，处理 `EOF` 或扩展 ASCII 字符时易出错），且使用前必须包含<ctype.h> 头文件；实际开发中常结合 `continue` 语句在循环中跳过这些空白字符，避免处理换行、空格等无业务意义的字符。

# DAY\_26 指针的基本用法

C 程序设计中使用指针可以使程序简洁、紧凑、高效，有效地表示复杂的数据结构，动态分配内存，得到多于一个的函数返回值。

## 一、地址和变量

在计算机内存中，每一个字节单元，都有一个编号，称为地址。在 C 语言中，内存单元的地址称为指针，专门用来存放地址的变量，称为指针变量。在不影响理解的情况下，有时对地址入当针和指针变量不区分，通称指针。

## 二、指针变量的说明

一般形式如下

<存储类型> <数据类型> \*<指针变量名>

例如, char \*pName;

指针的存储类型是指针变量本身的存储类型，指针说明时指定的数据类型不是指针变量本身的数据类型，而是指针目标的数据类型。简称为指针的数据类型。

指针在说明的同时，也可以被赋予初值，称为指针的初始化。

一般形式是

<存储类型><数据类型>\*<指针变量名>=<地址量>

例如:int a,\*pa=&a;

在上面语句中，把变量 a 的地址作为初值赋予了刚说明的 int 型指针 pa。

int a = 3; //int a; a= 3;

int \*pa = &a; //int \*pa; pa = &a;

指针指向的内存区域中的数据称为指针的目标。如果它指向的区域是程序中的一个变量的内存空间，则这个变量称为指针的目标变量。简称为指针的目标。

➤ 代码实现参考 pointer1.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int a = 10;
    int *p;
    p = &a;
    printf("%p %d\n", &p, sizeof(p)); /* p 所占的空间和地址 */
    printf("%p %p\n", p, &a); /* p 指向的地址和 a 的地址 */
    printf("%d %d\n", a, *p); /* p 指向的地址中的数据和 a 中的数据 */
}
return 0;
}
```

引入指针要注意程序中的 px、\*px 和 &px 三种表示方法的不同意义。设 px 为一个指针，则：

px -指针变量，它的内容是地址量

\*px - 指针所指向的对象，它的内容是数据  
 &px - 指针变量占用的存储区域的地址，是个常量

### 三、指针的赋值运算

指针的赋值运算指的是通过赋值运算符向指针变量送一个地址值  
 向一个指针变量赋值时，送的值必须是地址常量或指针变量，不能是普通的整数(除了赋零以外)

指针赋值运算常见的有以下几种形式：

1.把一个普通变量的地址赋给一个具有相同数据类型的指针

```
double x=15, *px;
px = &x;
```

2.把一个已有地址值的指针变量赋给具有相同数据类型的另一个指针变量.例如:

```
float a, *px, *py;
px = &a;
py = px;
```

➤ 代码实现参考 pointer2.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int a = 10;
    int *p, *q;
    p = &a;
    //q = &a;
    q = p;
    printf("%p %d\n", &p, sizeof(p)); /* p 所占的空间和地址 */
    printf("%p %d\n", &q, sizeof(q)); /* q 所占的空间和地址 */
    printf("%p %p\n", p, &a); /* p 指向的地址和 a 的地址 */
    printf("%p %p\n", q, &a); /* q 指向的地址和 a 的地址 */
    printf("%d %d\n", a, *p); /* p 指向的地址中的数据和 a 中的数据 */
}
printf("%d %d\n", a, *q); /* q 指向的地址中的数据和 a 中的数据 */
}
return 0;
}
```

3.把一个数组的地址赋给具有相同数据类型的指针。例如:

```
int a[20], *pa;
pa = a; //等价 pa = &a[0]
```

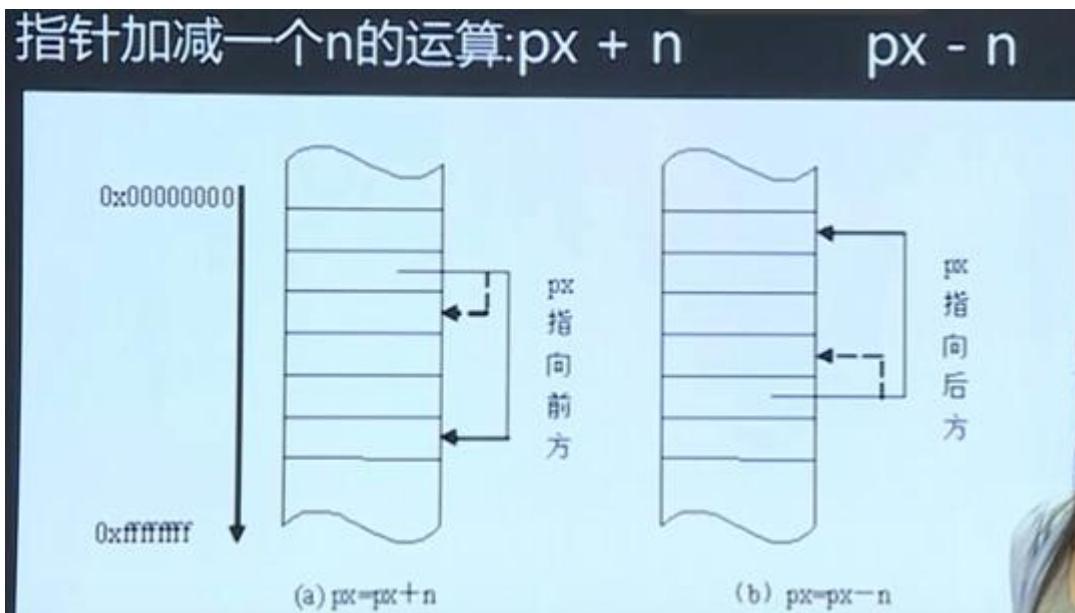
# DAY\_27 指针的运算

指针运算是以指针变量所存放的地址量作为运算量而进行的运算，指针运算的实质就是地址的计算，指针运算的种类是有限的，它只能进、赋值运算、算术运算和关系运算。

**指针的算术运算见下表：**

运算符	计算形式	意义
+	$px+n$	指针向地址大的方向移动 n 个数据
-	$px-n$	指针向地址小的方向移动 n 个数据
++	$px++$ 或 $px+1$	指针向地址大的方向移动 1 个数据
--	$px--$ 或 $px-1$	指针向地址小的方向移动 1 个数据
-	$px-py$	两个指针之间相隔数据元素的个数

## 一、指针加减



➤ 代码实现参考 pointer3.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int a = 10, *p;
    double b = 3, *q;
    p = &a;
    q = &b;
```

```

    printf("%p %p\n", p, p + 2);
    printf("%p %p\n", q, q + 2);
    return 0;
}

```

注意：

不同据类型的两个指针实行加减整数运算是无意义的  
 $px+n$  表示的实际位置的地址量是 $(px) + \text{sizeof}(px \text{ 的类型}) * n$   
 $px-n$  表示的实际位置的地址量是 $(px) - \text{sizeof}(px \text{ 的类型}) * n$

## 二、两指针的加减运算

$px - py$  运算的结果是两指针指向的地址位置之间相隔数据的个数。因此，两指针相减不是两指针持有的地址值相减的结果。两指针相减的结果值不是地址量，而是一个整数值，表示两指针之间相隔数据的个数。

➤ 代码实现参考 pointer4.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int a[5] = {4, 8, 1, 2, 7};
    int *p, *q, *m;
    p = a;
    q = &a[3];

    printf("%p %p\n", p, q);
    printf("%d %d\n", *p, *q);
    printf("%d\n", q - p);

    m = p++;
    printf("%p %d\n", p, *p);
    printf("%p %d\n", m, *m);
    return 0;
}
```

## 三、两指针的关系运算

表 4.2 指针的关系运算符

运算符	说明	例子
>	大于	px>py
<	小于	px<py
>=	大于等于	px>=py
<=	小于等于	px<=py
!=	不等于	px != py
==	等于	px == py

两指针之间的关系运算表示它们指向的地址位置之间的关系。指向地址大的指针大于指向地址小的指针。

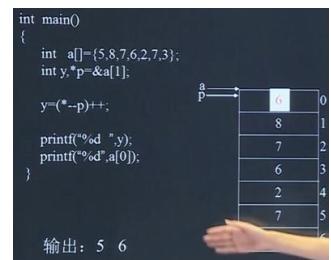
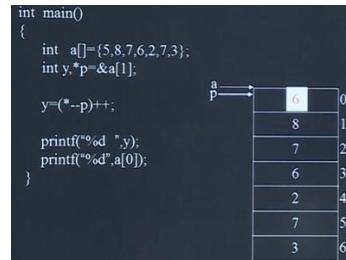
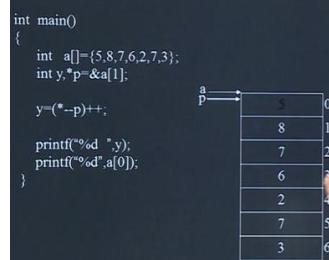
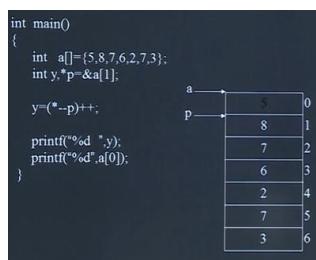
指针与一般整数变量之间的关系运算没有意义,但可以和零进行等于或不等于的关系运算,判断指针是否为空。

➤ 代码实现参考 pointer5.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int *p = NULL;

    printf("%p %d\n", p, p);
    return 0;
}
```



#### 四、指针当前的值

➤ 代码实现参考 pointer6.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int i, *p, a[7];
    p = a;
    for(i = 0; i < 7; i++) {
        scanf("%d", p++);
    }
    printf("\n");

    p = a;
    for(i = 0; i < 7; i++) {
        printf("%d", *p);
        printf("\n");
        p++;
    }
}
```

```
    }  
    return 0;  
}
```

结果：

```
5  
8  
7  
6  
2  
7  
3
```

```
5  
8  
7  
6  
2  
7  
3
```

注意优先级！！！

# DAY\_28 指针与一维数组

在 C 语言中，数组的指针是指数组在内存中的起始地址，数组元素的地址是指数组元素在内存中的起始地址

一维数组的数组名为一维数组的指针(起始地址)

例如：

```
double x[8];
```

因此，x 为 x 数组的起始地址

设指针变量 px 的地址值等于数组指针 x(即指针变量 px 指向数组的首元数)，则

x[i]、\*(px+i)、\*(x+i) 和 px[i] 具有完全相同的功能：访问数组第 i+1 个数组元素。

➤ 代码实现参考 pointer7.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int a[] = {1, 6, 9, 12, 61, 12, 21};
    int *p, i, n;

    p = a;
    n = sizeof(a) / sizeof(int);

    printf("%p %p %p\n", a, a+1, a+2);
    for (i = 0; i < n; i++) {
        printf("%d %d %d %d\n", a[i], *(p+i), *(a+i), p[i]);
    }
    puts("");
    return 0;
}
```

注意：

指针变量和数组在访问数组中元素时一定条件下其使用方法具有相同的形式，因为指针变量和数组名都是地址量

但指针变量和数组的指针(或叫数组名)在本质上不同，指针变量是地址变量，而数组的指针是地址常量

编写一个函数，将整型数组中的 N 个数按反序存放，需用到指针。

➤ 代码实现参考 pointer8.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int a[] = {1, 6, 9, 12, 61, 12, 21};
    int *p, *q, n, t;
```

```
n = sizeof(a) / sizeof(int);
p = a;
q = &a[n - 1]; /* 指向数组最后一个元素的地址 */

while (p < q) {
    t = *p;
    *p = *q;
    *q = t;
    p++;
    q--;
}

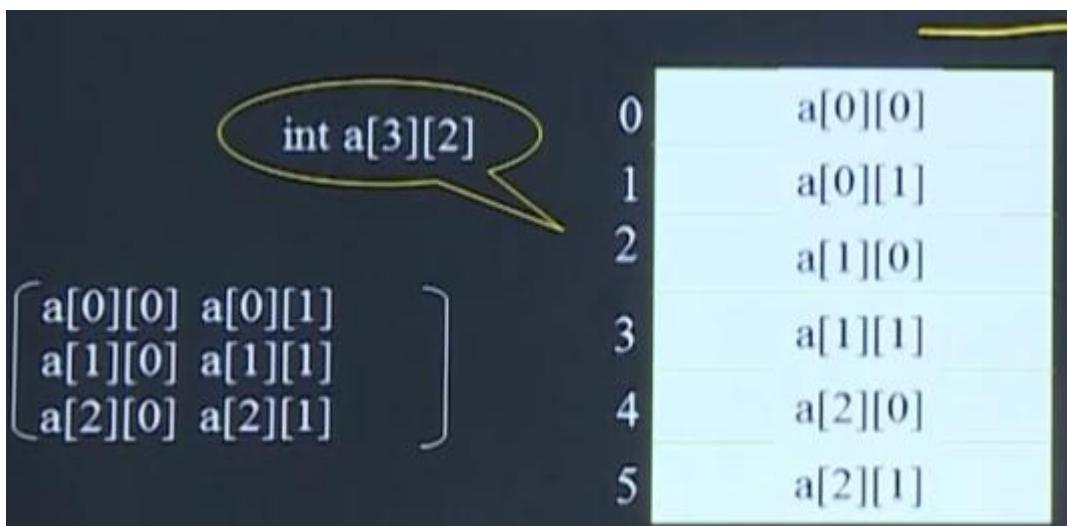
for (t = 0; t < n; t++) {
    printf("%d ", a[t]);
}
puts("");
return 0;
}
```

# DAY\_29 指针与二维数组

## 一、引入

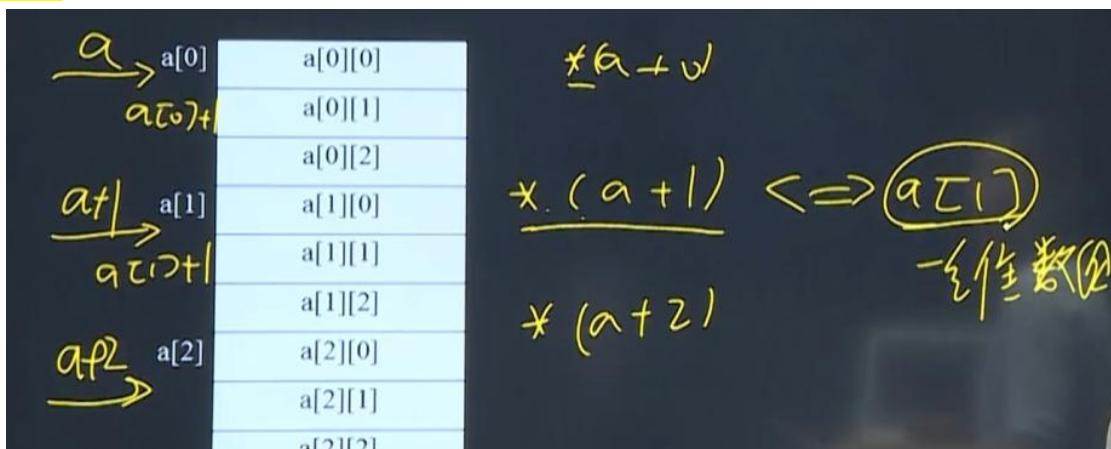
多维数组就是具有两个或两个以上下标的数组

在 C 语言中，二维数组的元素连续存储，按行优先存



编程实现，使用一级指针遍历二维数组。可把二维数组看作由多个一维数组组成。

比如 `int a[3][3]`, 含有三个元素: `a[0]`、`a[1]`、`a[2]`，元素 `a[0]`、`a[1]`、`a[2]` 都是一维数组名。二维数组名代表数组的起始地址，数组名加 1，是移动一行元素。因此，二维数组名常被称为行地址。



## 二、行指针

行指针(数组指针)存储行地址的指针变量，叫做行指针变量。形式如下：

<存储类型><数据类型>(\*<指针变量名>)[表达式]；

例如 `int a[2][3]; int (*p)[3];`

方括号中的常量表达式表示指针加 1，移动几个数据。

当用行指针操作二维数组时，表达式一般写成 1 行的元素个数，即列数。

➤ 代码实现参考 `pointer9.c`

```
#include <stdio.h>
```

```

int main(int argc, char *argv[])
{
    int a[3][2] = {{1, 6}, {9, 12}, {61, 12}};
    int (*p)[2]; /* 行指针,[2]为列数 */

    p = a;
    printf("%d %d %d\n", a[1][1], p[1][1], *(*(a + 1) + 1), *(*(p
+ 1) + 1));
    /*
     *   1 6
     *   9 12
     *   61 12
     * a[1][1]:先移动一行, 此时 a[1]为数组名, a[1][1]为 a[1]行 a[1]
     * 个元素, 即为 12. (第 a[1]行 a[0]为 9)
     *   p[1][1]:先移动一行, 此时 p[1]为数组名, p[1][1]为 p[1]行 p[1]
     * 个元素, 即为 12. (第 p[1]行 p[0]为 9)
     *   *(*(a + 1) + 1):先将 a 移动一行, 取*, 成为数组名, 再加 1, 取*
     * , 成为值 12.
     *   *(*(p + 1) + 1):先将 p 移动一行, 取*, 成为数组名, 再加 1, 取*
     * , 成为值 12.
    */

    return 0;
}

```

编程实现, 使用行指针表示二维数组 int a[2][4]的元素 a[1][1].

➤ 代码实现参考 pointer10.c

```

#include <stdio.h>

int main(int argc, char *argv[])
{
    int a[3][2] = {{1, 6}, {9, 12}, {61, 12}};
    int (*p)[2], i, j;

    p = a;
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 2; j++) {
            printf("%d ", p[i][j]); /* 不能写*p[i][j], 因为 p[i][j]
为值, 不是地址 */
        }
        puts("");
    }
}

```

```
    return 0;  
}
```

# DAY\_30 字符指针与字符串

## 一、引入

初始化字符指针是把内存中字符串的首地址赋予指针，并不是把该字符串复制到指针中

```
char str]= "Hello World";
char *p= str;
```

在 C 编程中，当一个字符指针指向一个字符串常量时，不能修改指针指向的对象的值

```
char *p = "Hello World";
*p = 'h'; // 错误，字符串常量不能修改
```

➤ 代码实现参考 pointer11.c

```
#include <stdio.h>
#include <ctype.h>

int main(int argc, char *argv[])
{
    char ch1[] = "Hello, world!";
    char ch2[] = "Hello, C!";

    char *p = ch1;

    /* 第二个元素是字母 */
    if (isalpha(*(p + 1))) {
        /* 先对数组的元素移动，再取* */
        if (isupper(*(p + 1))) {
            *(p + 1) = tolower(*(p + 1));
            printf("%c\n", *(p + 1)); /* 打印转换为小写后的元素 */
        }
        printf("%s\n", p); /* 打印转换后的字符数组 */
    } else {
        *(p + 1) = toupper(*(p + 1));
        printf("%c\n", *(p + 1));
        printf("%s\n", p);
    }
}

return 0;
}
```

不利用任何字符串函数，编程实现字符串连接函数的功能。

➤ 代码实现参考 pointer12.c

```
#include <stdio.h>
#include <ctype.h>
```

```
int main(int argc, char *argv[])
{
    char ch[100] = "welcome";
    char *p = ", hello world!";

    int i = 0;

    /* 先找到 ch 的结束位置'\0' */
    while (*(ch + i) != '\0') {
        i++;
    }

    /* 再将 p 的内容复制到 ch 的结束位置 */
    while (*p != '\0') {
        *(ch + i) = *p;
        i++;
        p++;
    }

    /* 最后在 ch 的结束位置添加'\0', 此时*p = '\0' */
    *(ch + i) = *p;

    puts(ch);

    return 0;
}
```

# DAY\_31 指针数组

## 一、引入

所谓指针数组是指由若干个具有相同存储类型和数据类型的指针变量构成的集合  
指针数组的一般说明形式

<存储类型><数据类型>\*<指针数组名>[<大小>]

指针数组名表示该指针数组的起始地址

声明一个指针数组

```
double *pa[2], a[2][3];
```

把一维数组 a[0] 和 a[1] 的首地址分别赋予指针变量数组的数组元素 pa[0] 和 pa[1]

```
pa[0]=a[0];// 等价 pa[0]= &a[0][0];
```

```
pa[1]=a[1];//等价 pa[1]= &a[1][0];
```

此时 pa[0] 指向了一维数组 a[0] 的第一个元素 a[0][0],  
而 pa[1] 指向了一维数组 a[1] 的第一个元素 a[1][0]

➤ 代码实现参考 pointer13.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int *p[3]; /* p[0] p[1] p[2] */
    int a[] = {3, 6, 1, 9, 10};

    p[0] = a;
    p[1] = a + 1;
    p[2] = a + 3;

    printf("%d %d %d\n", *p[0], *p[1], *p[2]);
    /* 不能写*(p + 1), p[1]对应的是数组的指针（地址） */
    printf("%d %d %d\n", *a, *(a + 1), *(a + 3));

    return 0;
}
```

➤ 代码实现参考 pointer14.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int a[2][3] = {{1, 4, 6}, {12, 9, 7}}; /* 数组名 a[0] a[1] */
    int *p[2];

    p[0] = a[0]; /*&a[0][0] */
```

```

p[1] = a[1]; /* &a[1][0] */

printf("%d\n", *(p[0] + 1));
printf("%d\n", p[0][1]);

return 0;
}

```

编程:利用指针数组处理一个二维数组, 要求求出二维数组所有元素的和。

➤ 代码实现参考 pointer15.c

```

#include <stdio.h>

int main(int argc, char *argv[])
{
    int a[2][3] = {{1, 4, 6}, {12, 9, 7}};
    int *p[2] = {a[0], a[1]};
    int i, j, sum = 0;

    /* 遍历数组 */
    for (i = 0; i < 2; i++) {
        for (j = 0; j < 3; j++) {
            printf("%d ", *(p[i] + j));
            //printf("%d ", p[i][j]);
            //printf("%d ", *((a + i) + j)); /* p[i] = *(a + i),
p[i]指针对应数组名 a[i] */
        }
        puts("");
    }

    /* 二维数组求和 */
    for (i = 0; i < 2; i++) {
        for (j = 0; j < 3; j++) {
            //sum += *((p + i) + j);
            sum += p[i][j];
        }
    }
    printf("sum = %d\n", sum);

    return 0;
}

```

# DAY\_32 多级指针

## 一、引入

把一个指向指针变量的指针变量，称为多级指针变量。对于指向处理数据的指针变量称为一级指针变量，简称一级指针。而把指向一级指针变量的指针变量称为二级指针变量简称二级指针。

二级指针变量的说明形式如下

<存储类型><数据类型> \*\*<指针名>

➤ 代码实现参考 pointer16.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int m = 10;
    int *p, **q;

    p = &m;
    q = &p;

    printf("%p %p %d\n", &m, p, *p);
    printf("%p %p %d\n", p, q, **q);

    return 0;
}
```

## 二、多级指针的运算

指针变量加 1，是向地址大的方向移动一个目标数据。类似的道理，多级指针运算也是以其目标变量为单位进行偏移：

比如，`int **p;p+1` 移动一个 `int*` 变量所占的内存空间。再比如 `int***p, p+1` 移动一个 `int **` 所占的内存空间。

## 三、多级指针和指针数组

指针数组也可以用另外一个指针来处理。

➤ 代码实现参考 pointer17.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int a[] = {3, 6, 9};
    int *m;
    int *p[2] = {&a[0], &a[1]}; /* 指针数组里，皆为指针 */
}
```

```

int **q;

m = &a[0]; /* m 地址 指向 a 地址 */
q = &p[0]; /* p[0]的地址, 即为 p */
q = p;      /* 等价 p[0]的地址 */

printf("%d %d\n", a[0], *(a + 1));
printf("%d %d\n", m[0], m[1]);
printf("%d %d\n", *p[0], *(*(p + 1)));
/*
 * 不能写*(p + 1)或 p[1], p[1]指的是 a[1]的地址, 而非值,
 * 但可以写*(*(p + 1)), 这是指向 a[1]中的值
 */
printf("%d %d\n", *q[0], *q[1]);
return 0;
}

```

➤ 代码实现参考 pointer18.c

```

#include <stdio.h>

int main(int argc, char *argv[])
{
    char *s[] = {"apple", "pear", "potato"};
    char **p;
    int i, n;

    p = &s[0]; //p = s;
    i = 0;
    /* 数组有三个字符串常量, 每个字符串对应一个指针, 除以指针所占空间
    数即为字符串个数 */
    n = sizeof(s) / sizeof(char *);

    while (i < n) {
        printf("%s\n", s[i]);
        printf("%s\n", p[i]);
        /* p[i]指向 s[i], s[i]又为字符指针, 指向字符串常量 */
        printf("%s\n", *p);
        /* p 为地址, 与%s 不匹配, *p 取值, 匹配 */
        i++;
    }
    return 0;
}

```

# DAY\_33 void 指针和 const 指针

## 一、void 指针

void 指针是一种不确定数据类型的指针变量，它可以通过强制类型转换让该变量指向任何数据类型的变量

一般形式为: void \*<指针变量名称>;

对于 void 指针，在没有强制类型转换之前，不能进行任何指针的算术运算。

➤ 代码实现参考 pointer19.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int m = 10;
    double n = 3.14;

    void *p, *q;
    //p = (void *)&m; /* 强制类型转换，将 int 型指针转换为 void 型指针 */
    p = &m; /* 隐式类型转换，将 int 型指针转换为 void 型指针 */
    printf("%d %d\n", m, *(int *)p); /* 强制类型转换，将 void 型指针转换为 int 型指针 */
    q = (void *)&n;
    printf("%0.2lf %.2lf\n", n, *(double *)q); /* 强制类型转换，将 void 型指针转换为 double 型指针 */
    /* 0.2lf 表示保留两位小数 */
    return 0;
}
```

使用 void 指针遍历一维数组。

➤ 代码实现参考 pointer20.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int a[] = {5, 9, 1, 6, 9, 10};
    int i, n;
    void *p;

    n = sizeof(a) / sizeof(int);
    p = a; /* a 本身就是地址，不用& */

    for (i = 0; i < n; i++) {
```

```
//printf("%d ", a[i]);
printf("%d ", *((int *)p + i));
//printf("%d ", ((int *)p)[i]);
/*
 * 切记不可以*((int *)p[i]),
 * []优先级比*高，会先 p[i]无强制类型转换而报错
 */
}
puts("");
return 0;
}
```