

# 目录

<b>第一章：程序设计和 C 语言</b>	<b>1</b>
计算机程序	1
计算机语言	1
高级语言的发展	1
C 语言的发展	1
C 语言的特点	2
最简单的 C 语言程序	2
C 语言程序的结构	3
运行 C 程序的步骤与方法	4
程序设计的任务	4
<b>第二章：算法——程序的灵魂</b>	<b>5</b>
程序=算法+数据结构	5
什么是算法	5
算法的特性	5
如何表示一个算法	5
结构化程序设计方法	6
<b>第三章：顺序程序设计</b>	<b>7</b>
数据的表现形式以及运算	7
数据类型	8
运算符和表达式	10
表达式	11
C 语句	11
数据的输入输出	12
<b>第四章：选择结构程序设计</b>	<b>16</b>
选择结构和条件判断	16
用 IF 语句实现选择结构	16
关系运算符和关系表达式	16
逻辑运算符和逻辑表达式	16
条件运算符和条件表达式	17
选择结构的嵌套	17
SWITCH 语句	17
<b>第五章：循环结构程序设计</b>	<b>19</b>
三大循环语句	19
循环：	20
BREAK 以及 CONTINUE 的区别	20
<b>第六章：利用数组处理批量数据</b>	<b>21</b>
怎样定义和引用一维数组	21
二维数组	21

字符数组 .....	22
<b>第七章：用函数实现模块化程序设计 .....</b>	<b>25</b>
为什么要用函数.....	25
函数调用时候的数据传递 .....	25
被调函数的声明和函数原型 .....	26
函数的递归调用.....	26
数组作为函数参数 .....	26
局部变量与全局变量 .....	27
变量的存储方式和生存期 .....	27
关于变量的声明与定义 .....	30
内部函数与外部函数 .....	30
<b>第八章：善于利用指针 .....</b>	<b>31</b>
指针是什么 .....	31
指针变量 .....	31
通过指针引用数组 .....	31
用数组名做函数参数 .....	32
通过指针引用多维数组 .....	32
指向多维数组元素的指针变量 .....	33
通过指针引用字符串 .....	33
指向函数的指针 .....	34
返回指针值的函数 .....	35
指针数组和多重指针 .....	35
动态内存分配与指向它的指针变量 .....	36
数组指针与指针数组区别 .....	38
函数指针和指针函数的区别 .....	38
<b>第九章：用户自己建立数据类型.....</b>	<b>39</b>
定义和使用结构体变量 .....	39
结构体指针 .....	43
用指针处理链表 .....	43
共同体类型 .....	43
枚举类型 .....	44
用 <code>typedef</code> 声明新类型名 .....	45
<b>第十章：对文件的输入输出 .....</b>	<b>46</b>
C 文件的有关基本知识 .....	46
与文件有关函数 .....	47
随机读写数据文件 .....	49
文件读写的出错检测 .....	49
文件函数的应用 .....	50
<b>C 语言第一章基本概念 .....</b>	<b>51</b>
<b>C 语言第二章基本概念 .....</b>	<b>52</b>

C 语言第三章基本概念 .....	52
C 语言第四章基本概念 .....	53
C 语言第五章基本概念 .....	54
C 语言第六章基本概念 .....	54
C 语言第七章基本概念 .....	54
C 语言第八章基本概念 .....	55
C 语言第九章基本概念 .....	57

# 第一章：程序设计和 C 语言

## 计算机程序

指令：可以被计算机理解并执行的基本操作命令。

程序：一组计算机能识别和执行的指令。一个特定的指令序列用来完成一定的功能。

软件：与计算机系统操作有关的计算机程序、规程、规则，以及可能有的文件、文档及数据。

## 计算机语言

机器语言：计算机能直接识别和接受的二进制代码称为**机器指令**。机器指令的集合就是该计算机的**机器语言**。

特点：难学，难记，难检查，难修改，难以推广使用。依赖具体机器难以移植。

汇编语言（符号语言）：机器语言的符号化。用英文字母和数字表示指令的**符号语言**。

特点：相比机器语言简单好记，但仍然难以普及。汇编指令需通过**汇编程序**转换为机器指令才能被计算机执行。依赖具体机器难以移植。

高级语言：高级语言更接近于人们习惯使用的自然语言和数学语言。

特点：功能强大，不依赖于具体机器。用高级语言编写的**源程序**需要通过**编译程序**转换为机器指令的**目标程序**。

## 高级语言的发展

1: 非结构的语言：初期的语言属于非结构化的语言，编程风格比较随意，只需要符合语言规则即可。例如：早期的 BASIC, FORTRAN, ALGOL 等；但是相对应的人们追求效率而采用了很多小技巧，程序难以阅读和维护。

2: 结构化的语言：规定程序必须由具有良好特性的基本结构(顺序结构、选择结构、循环结构)构成，程序中的流程不允许随意跳转，程序总是由上而下顺序执行各个基本结构。其特点是程序结构清晰，易于编写、阅读和维护。

3.面向对象的语言：面向对象语言（Object-Oriented Language）是一类以对象作为基本程序结构单位的程序设计语言，指用于描述的设计是以对象为核心，而对象是程序运行时刻的基本成分。

## C 语言的发展

1、1972—1973 年间，美国贝尔实验室的 D.M.Ritchie 在 B 语言的基础上设计出了 C 语言。

2、最初的 C 语言只是为描述和实现 UNIX 操作系统提供一种工作语言而设计的。

3、随着 UNIX 的日益广泛使用，C 语言也迅速得到推广。1978 年以后，C 语言先后移

植到大、中、小和微型计算机上。C 语言便很快风靡全世界，成为世界上应用最广泛的程序设计高级语言。

4、以 UNIX 第 7 版中的 C 语言编译程序为基础，1978 年，Brian W.Kernighan 和 Dennis M.Ritchie 合著了影响深远的名著 The C Programming Language，这本书中介绍的 C 语言成为后来广泛使用的 C 语言版本的基础，它是实际上第一个 C 语言标准。

5、1983 年，美国国家标准协会(ANSI)，根据 C 语言问世以来各种版本对 C 语言的发展和扩充，制定了第一个 C 语言标准草案('83 ANSI C)。

6、1989 年，ANSI 公布了一个完整的 C 语言标准——ANSI X3.159—1989(常称为 ANSI C 或 C 89)。

7、1990 年，国际标准化组织 ISO(International Standard Organization)接受 C 89 作为国际标准 ISO/IEC 9899: 1990，它和 ANSI 的 C 89 基本上是相同的。

8、1999 年，ISO 又对 C 语言标准进行了修订，在基本保留原来的 C 语言特征的基础上，针对应用的需要，增加了一些功能，尤其是 C++中的一些功能，并在 2001 年和 2004 年先后进行了两次技术修正，它被称为 C 99，C 99 是 C 89 的扩充。

## C 语言的特点

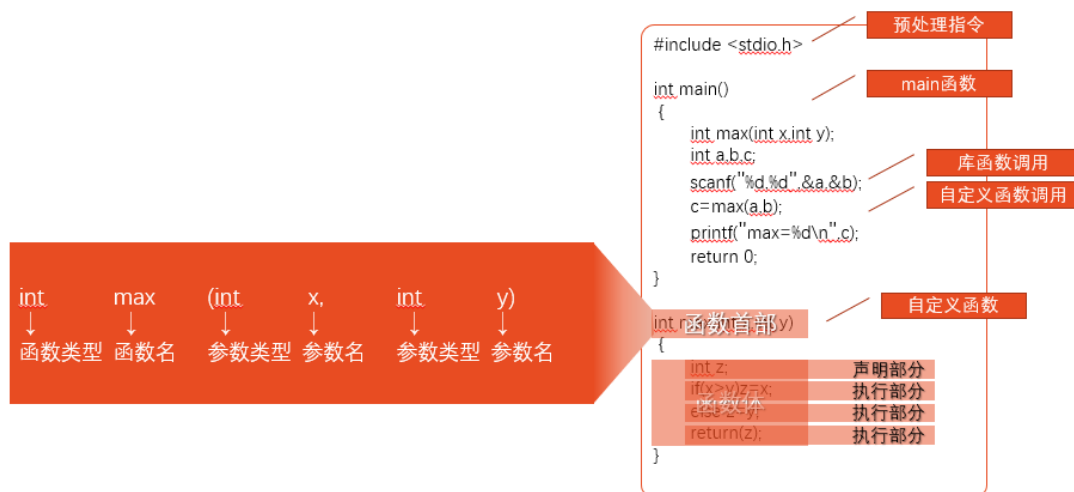
- 1、语言简洁、紧凑、使用方便、灵活。(关键字只有 37 个，而且程序书写自由灵活)
- 2、运算符丰富。(C 总计有 34 种运算符)
- 3、数据类型丰富。(int、char、float、double、\_bool…)
- 4、具有结构化的程序语句。(比如 if…else…，switch…case…，用函数作为程序的模块单位，便于实现程序的模块化。C 语言是完全模块化和结构化的语言)
- 5、语法限制不太严格，程序设计自由度大。(但是同样的，限制与灵活是一对矛盾。因为不太严格，所以需要程序编写者自己确保程序的正确，比如数组下标越界行为)
- 6、C 语言允许直接访问物理地址，能进行位 (bit) 操作，也可以实现汇编语言的大部分操作，可以直接对硬件进行操作。(也就是说，C 语言具有高级语言的功能，有可以实现低级语言的许多功能。不但是成功的系统描述语言，又是通用的程序设计语言)
- 7、C 语言编写的程序移植性好。(因为 C 的编译系统很简洁，所以很容易移植到新的系统上去。C 编译系统在新的系统上运行时，可以直接编译“标准链接库”中的大部分功能，不需要修改源代码，这是因为标准链接库就是用可移植的 C 语言写的。)
- 8、生成目标代码质量高，程序执行效率高。(C 语言的可移植性好，硬件控制能力强，表达和运算能力强)

## 最简单的 C 语言程序

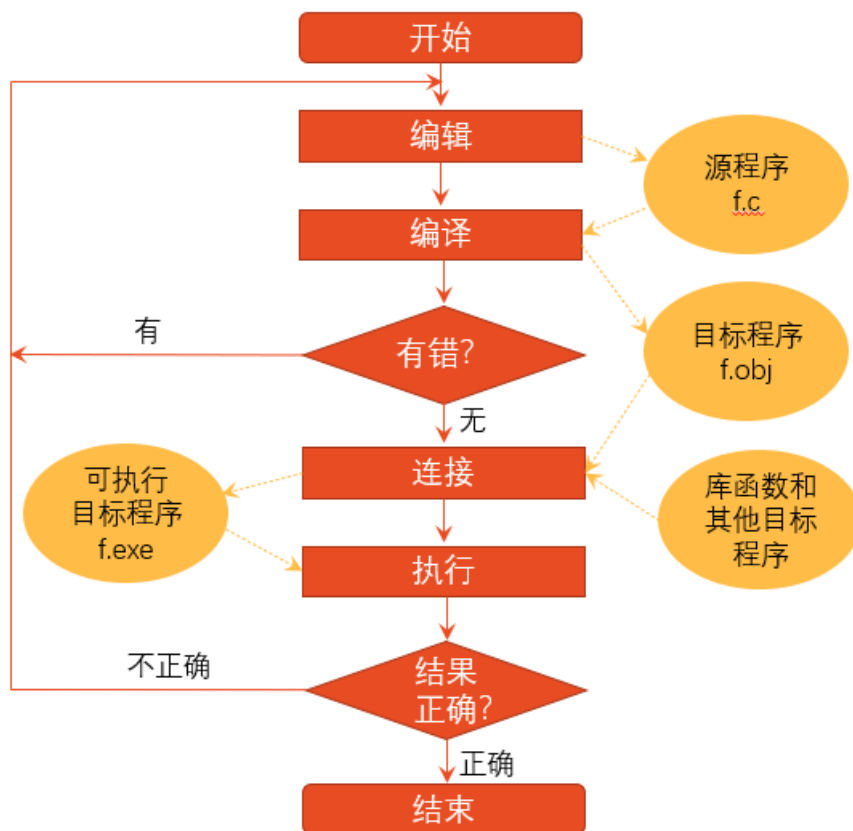
```
#include<stdio.h>    //预编译处理
int main()            //main 前面的 int 表示此函数的类型是 int 类型(整型)，即在执
                        //行主函数后会得到一个值(即函数值)，其值为整型。
{
    printf("\n");      //输出
    return 0;          //函数结束的标志
}
```

## C 语言程序的结构

- 1、一个程序由一个或多个源程序文件组成  
一个源程序文件可以包括 3 个部分：
  - ①预处理指令
  - ②全局声明
  - ③函数定义
- 2、函数是 C 程序的主要组成部分
- 3、一个函数包括两个部分
  - ①函数首部
  - ②函数体
  - 1) 声明部分
  - 2) 执行部分
- 4、程序总是从 main 函数开始执行的
- 5、程序中要求计算机的操作是由函数中的 C 语句完成的（如赋值语句等操作）
- 6、在每个数据声明和语句最后必须有一个分号
- 7、C 语言本身不提供输入输出语句
- 8、程序应当包含注释



## 运行 C 程序的步骤与方法



## 程序设计的任务

- 1、问题分析
- 2、设计算法
- 3、编写程序
- 4、对源程序进行编辑、编译和连接
- 5、运行程序、分析结果
- 6、编写程序文档

## 第二章：算法——程序的灵魂

### 程序=算法+数据结构

一个程序主要包括两个方面的信息：

(1) 对数据的描述：在程序中要指定用到哪些数据，以及这些数据的类型和数据的组织形式，这就是数据结构。

(2) 对操作的描述：要求计算机进行操作的步骤，也就是算法。

类比于做菜的时候，数据相当于配料，而数据结构相当于配料的组合。算法就相当于做菜步骤。

### 什么是算法

算法相当于解题的方法和步骤。每一个题有很多和解题的方法与步骤，但是不同的方法和步骤有优劣之分。选择算法的原则是：方法简单，步骤少。

计算机算法分类：

(1)：数值运算：数值运算的目的是求数值解。由于数值运算往往有现成的模型，可以运用数值分析方法，因此对数值运算的算法的研究比较深入，算法比较成熟；

(2)：非数值运算：计算机在非数值运算方面的应用远超在数值运算方面的应用。非数值运算的种类繁多，要求各异，需要使用者参考已有的类似算法，重新设计解决特定问题的专门算法；

### 算法的特性

- 1、有穷性（一个算法应该包括有限的操作步骤）
- 2、确定性（每一步的步骤应当是确定的）
- 3、有零个或多个输入（所谓输入是指在执行算法时需要从外界取得必要的信息）
- 4、有一个或多个输出（算法的目的是为了解，‘解’就是输出）
- 5、有效性（每一步都要有效的执行）

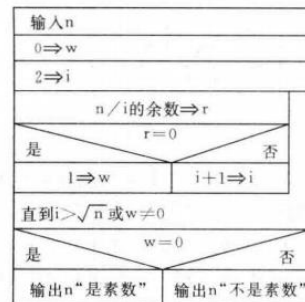
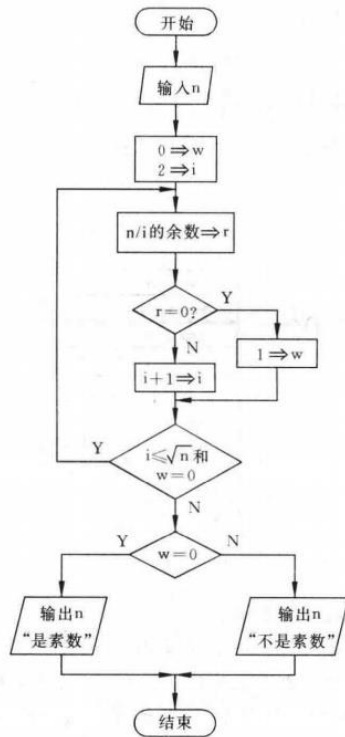
### 如何表示一个算法

- 1、自然语言
- 2、传统流程图
- 3、结构化流程图

4、伪代码（伪代码是用介于自然语言和计算机语言之间的文字和符号来描述算法。它如同一篇文章一样，自上而下地写下来。每一行(或几行)表示一个基本操作。它不用图形符号，因此书写方便，格式紧凑，修改方便，容易看懂，也便于向计算机语言算法(即程序)过渡）

- 5、机器语言





## 结构化程序设计方法

- 1、自顶向下
- 2、逐步细化
- 3、模块化设计
- 4、结构化编码

# 第三章：顺序程序设计

## 数据的表现形式以及运算

1、常量：在程序运行过程中，其值不能被改变的量为常量；

①整型常量

②实型常量

1) 十进制小数形式：123.145, ...

2) 指数形式：如 12.34E5 (表示  $12.34 \times 10^5$ )，12.34e5 (表示  $12.34 \times 10^5$ )

③字符常量

1) 普通字符：用单撇号括起来的一个字符，如'A','&'；字符常量在计算机中存储时，并不是存储字符本身，而是以其代码存储的，比如字符'a'的 ASCII 码值为 97，则其在存储单元中存放的是 97 (按照二进制形式存放)；

2) 转义字符：这是一种特殊形式的字符常量，就是以字符"\"开头，比如'\t'就相当于 Tab；这是控制字符，没有办法用一般形式的字符去表示；

3) 字符串常量：用双撇号把若干个字符括起来，如"jia"，字符串常量是双撇号中的全部字符但不包括双撇号本身；字符串常量末尾为自动添加'\0'，在实际运行过程中，对于 char ch[5] = "12345"，这是会报错的，但是如果是 char ch[s]={'1','2','3','4','5'}则正确，一定要区分开字符串常量和字符数组的区别；

4) 符号常量：#define PI 3.1415926，这是符号常量，在编译的时候，预处理器会先对 PI 进行处理，但是一定要明白，这只是简单的字符串替换而已。

使用符号常量，含义清楚，见名知意，同时，在需要改变在程序多处用到的同一个常量的时候，能做到“一改全改”；

2、变量

变量代表一个有名字的，具有特定属性的一个存储单元，它用来存放数据，也就是存放变量的值，在程序运行期间，变量的值是可以改变的；

变量必须先定义再使用。同时要明白变量名与变量值是不同的概念。变量名其实就是一个以名字为代表的存储地址，在程序编译的时候由编译系统给每个变量名分配对应的内存地址，而变量值则是存储带该地址中的具体的数值。

3、常变量

```
const int a=3;
```

其中定义了一个整型变量 a，其值为 3，在变量存在期间其值不能改变；

常变量与变量的异同是：常变量是具有名字的不变量，而常量是没有名字的不变量；有名字便于在程序中引用；

思考:常变量与符号常量有什么不同?

```
如:# define Pi 3.1415926//定义符号常量
```

```
const float pi=3.1415926;//定义常变量
```

符号常量 Pi 和常变量 pi 都代表 3.1415926,在程序中都能使用。但二者性质不同:定义符号常量用# define 指令,它是预编译指令,它只是用符号常量代表一个字符串,在预编译时仅进行字符替换,在预编译后,符号常量就不存在了(全替换成 3.1415926 了),对符号常量的名字是不分配存储单元的。而常变量要占用存储单元,有变量值,只是该值不改变而已。从使用的角度看,常变量具有符号常量的优点,而且使用更方便。有了常变量以后,可以不必多用符号常量。

#### 4、标识符

在计算机高级语言中,用来对变量、符号常量名,函数、数组、类型等命名的有效字符序列统称为标识符(identifier)。简单地说,标识符就是一个对象的名字。前面用到的变量名 pl,p2,c,f,符号常量名 PI,PRICE,函数名 printf 等都是标识符。

C 语言规定标识符只能由字母、数字和下画线 3 种字符组成,且第 1 个字符必须为字母或下画线。

## 数据类型

### 1、整型数据的分类

整型数据常见的存储空间和值的范围(Visual C++ 的安排)

类 型	字节数	取 值 范 围
int(基本整型)	4	$-2\,147\,483\,648 \sim 2\,147\,483\,647$ , 即 $-2^{31} \sim (2^{31}-1)$
unsigned int(无符号基本整型)	4	$0 \sim 4\,294\,967\,295$ , 即 $0 \sim (2^{32}-1)$
short(短整型)	2	$-32\,768 \sim 32\,767$ , 即 $-2^{15} \sim (2^{15}-1)$
unsigned short(无符号短整型)	2	$0 \sim 65\,535$ , 即 $0 \sim (2^{16}-1)$
long(长整型)	4	$-2\,147\,483\,648 \sim 2\,147\,483\,647$ , 即 $-2^{31} \sim (2^{31}-1)$
unsigned long(无符号长整型)	4	$0 \sim 4\,294\,967\,295$ , 即 $0 \sim (2^{32}-1)$
long long(双长型)	8	$-9\,223\,372\,036\,854\,775\,808 \sim 9\,223\,372\,036\,854\,775\,807$ 即 $-2^{63} \sim (2^{63}-1)$
unsigned long long (无符号双长整型)	8	$0 \sim 18\,446\,744\,073\,709\,551\,615$ , 即 $0 \sim (2^{64}-1)$

Note:

- ①其实还存在 signed 型, 但是默认认为 signed int 与 int 等价
- ②整形的值在存储单元中都是以补码的形式存在的;
- ③只有整形(包括字符型)数据可以添加 signed 或 unsigned, 实型数据不可以;
- ④对于无符号整形数据可以使用"%u"格式输出;

为什么对于 unsigned short 的数据赋值 -1 时会输出 65536 呢? 这是因为当对 -1 补码形式存放时, 就变成了全部二进制为 1, 此时因为 unsigned 类型, 所以无符号位, 此时按 %d 输出会得到 65535;

### 2、字符型数据

#### 1、字符与字符代码:

对于字符型数据, 程序只能识别 ASCII 码值中具有字符, 而其也有相对应的数值; 同时, 字符型的存放也是按照整型形式的, 但是并不是对应的补码, 而是其对应的 ASCII 代码存放。

例如: 'A' 的 ASCII 码值为十进制的 65, 则在地址中存放的为 1000001;

Note:

- ①整形 1 与字符'1'是不同的，整形  $1+1=2$ ，但是字符型并不是  $'1'+ '1' = '2'$ ；
- ②字符'1'只是代表了一个形状为'1'的符号；
- ③整形按照数值对应的补码存放。字符型按照对应的 ASCII 码值存放；

## 2、字符变量

对于字符变量，例如输入 `char ch='A'`，按照 `%c` 输出则为 'A'，按照 `%d` 输出则为 65；

表 3.3 字符型数据的存储空间和值的范围

类 型	字节数	取 值 范 围
signed char(有符号字符型)	1	$-128 \sim 127$ , 即 $-2^7 \sim (2^7 - 1)$
unsigned char(无符号字符型)	1	$0 \sim 255$ , 即 $0 \sim (2^8 - 1)$

char 型字符存进空间的时候:

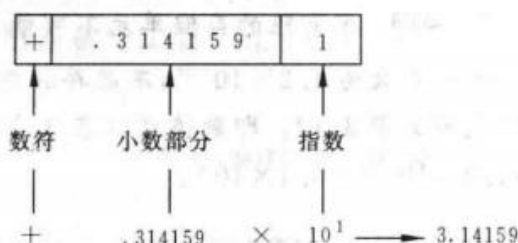
对应 0~127 相应的 ASCII 码值		对应相应的 ASCII 码值		此后每隔 256 进行一次循环，这是字符型的溢出处理结果
-256~-129	-128~-1	0~127	128~255	.....
无对应 ASCII 码值，为乱字符		无对应 ASCII 码值，为乱字符		

## 3、浮点型数据

之所以称为浮点型数据，是因为小数点的位置可以浮动，也就可以改变其对应的值；

### 1、float

float（单精度浮点型）：一般为其分配四个字节。在存储时候，系统会将实型数据分为小数部分以及指数部分，同时还有一个符号位。例如 3.14159 的存放：



该图其实是不准确的，因为实际的存储中，小数部分的存储是用二进制的，而指数部分使用的是幂指数存储。对于小数部分以及指数部分究竟用几位表示并未统一。如果小数部分占 bit 越多则精度越高，如果指数部分多的话则能表示的数值越大；

如何按照小数 24 位，指数部分（包括符号位）位 8 位，则其能表示的范围为：

$$-3.4 \times 10^{-38} \sim 3.4 \times 10^{38}$$

double 同理，而对于 long double 不同的编译器分配不同的空间；

实型数据的有关情况

类 型	字节数	有效数字	数值范围(绝对值)
float	4	6	0 以及 $1.2 \times 10^{-38} \sim 3.4 \times 10^{38}$
double	8	15	0 以及 $2.3 \times 10^{-308} \sim 1.7 \times 10^{308}$
long double	8	15	0 以及 $2.3 \times 10^{-308} \sim 1.7 \times 10^{308}$
	16	19	0 以及 $3.4 \times 10^{-4932} \sim 1.1 \times 10^{4932}$

Note:

①在实际运行中, float 都会自动转换为 double 型后进行计算, 这是为了提高精度;

②一定要记住, 有限的才能出单元是不可能准确的存储一个实数的, 例如 float 的范围其实并不是准确的  $-3.4 \times 10^{-38} \sim 3.4 \times 10^{-38}$ , 而应该是  $-3.4 \times 10^{-38} \sim -1.2 \times 10^{-38}$ , 0,  $1.2 \times 10^{-38} \sim 3.4 \times 10^{-38}$ ,

③只要是有小数点的值, 程序都会默认按照双精度型处理, 但是如果是 float 的话, 则会提醒精度不够, 但不会影响程序运行; (如下, 输入的是 float, 则存放的是 double)

```
int main()
{
    float a = 3.14159;
    printf("%f", a);
    return 0;
}
```

(double)(3.141589999999999883)

但是对于这种问题可以解决, 只需要如下即可, 强制分配 float:

```
int main()
{
    float a = 3.14159f;
    printf("%f", a);
    return 0;
}
```

(float)(3.141590118f)

## 运算符和表达式

### 1、优先级以及结合方向

Note:

①**%运算符只能使用在整型中**, 其结果也必须是整型, 除此之外的其他任意运算符的操作数都可以是任意类型;

②int/int 会只保留小数; 两个实数相除的话结果是双精度实数;

### 2、不同类型数据间的混合运算

(1) +、-、\*、/运算的两个数中有一个数为 float 或 double 型, 结果是 double 型, 因为系统将所有 float 型数据都先转换为 double 型, 然后进行运算。

(2) 如果 int 型与 float 或 double 型数据进行运算, 先把 int 型和 float 型数据转换为 double 型, 然后进行运算, 结果是 double 型。

(3) 字符(char)型数据与整型数据进行运算, 就是把字符的 ASCII 代码与整型数据进行运算。如:  $12 + 'A'$ , 由于字符 A 的 ASCII 代码是 65, 相当于  $12 + 65$ , 等于 77。如果字符型数据与实型数据进行运算, 则将字符的 ASCII 代码转换为 double 型数据, 然后进行运算。

### 3、强制类型转换运算符

(类型名) (表达式);

需要注意的是，这种转化并不会改变变量本身的数据类型，比如 `a = (int) (x+y)`，这相当于将 `x+y` 的值转换为 `int` 型赋值给 `a`，**并不会改变 `x+y` 本身**；

Note：

- ①C 语言中一共有两种强制转换，一种是系统主动转换，一种是用户强制转换；
- ②典型例题：float x；若要 x 对 3 取余，则需要 `(int) x%3`；或者 `((int) x) %3`；

## 表达式

表达式是由一系列运算符 (operators) 和操作数 (operands) 组成的。这既是表达式的定义，同时也指明了表达式的组成成分。运算符指明了要进行何种运算和操作，而操作数则是运算符操作的对象。表达式后面添加分号就成了语句。

## C 语句

一个函数包括两部分，声明部分和执行部分，其中执行部分就是由语句组成的，语句的作用就是向计算机系统发出操作的指令，要求执行相应的操作，当 C 语句编译之后可以产生若干条指令。需注意，声明部分不是语句，自然不会产生机器指令，它只是关于数据的声明；

### 1、C 语句的分类

#### (1) 控制语句

控制语句可以完成一定的控制功能：

常见的有：`if...else;for();while();do...while();continue;break;switch;return;goto`；

#### (2) 函数调用语句

函数调用语句由一个函数调用加一个分号构成；

例如：`printf ("\\n");`

#### (3) 表达式语句

表达式语句由一个表达式加一个分号构成；（一个表达式的最后添加一个分号就成了一个语句，从本质上来说，函数调用语句也属于表达式语句）

#### (4) 空语句

空语句只有一个分号，什么也不做；

#### (5) 复合语句

用 `{ }` 将一些语句和声明括起来，成为复合语句，也叫语句块；

### 2、赋值语句

例如：`x%=y+3` 就等于 `x%=(y+3)`；也就等于 `x=x%(y+3)`；

### 3、赋值过程中的类型转换

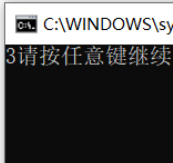
char 赋值为 int: char 转换为对应的 ASCII 码值赋值为 int;

int 赋值给 float: int 转换为小数形式按照浮点形式存放进存储空间; (疑问: 转换之后是按照双精度存储的吗? 猜测: 应该不会, 这就相当于强制转换, 已经声明了待转换的类型为 float)

int 赋值给 float: int 转换为小数形式按照双精度浮点形式存放进存储空间;

float 或 double 赋值给 int: 舍弃整个小数部分, 只保留整数赋值给 int; (疑问: 会四舍五入吗? 经过验证, 并不会四舍五入)

```
float f1 = 3.9;
int in;
in = f1;
printf("%d", in);
return 0;
```



double 赋值给 float: 先将精度转换为单精度, 即只保留 6~7 为有效数字, 但要注意, 原 double 型的大小不能超过 float 的数值范围;

float 赋值给 double: 数值不变, 不过在内存中会以 8 个字节存放, 有效位数会扩展;

int (4 字节) 赋值给 short (2 字节) 或 char (1 字节): 只会将其低字节部分数据送入, 而高字节出现“截断”;

```
int main()
{
    int in = 7946; //0001_1111_0000_1010
    char ch;
    ch = in;
    printf("%d", ch); //输出了10, 刚好为0000_1010, 说明只存放了低位进去
    return 0;
}
```

### 4、赋值表达式和赋值语句

赋值表达式和赋值语句是不同的, 赋值语句末尾必须由分号, 而赋值表达式没有; 同时, 一个表达式中可以包含多个表达式, 但不能包含赋值语句;

例如: `if((a=c)==0)` 一个表达式包括了 `c` 对 `a` 的赋值以及 `a` 等于 0 的判断两个表达式;

## 数据的输入输出

Note:

①输入输出都是以计算机为主机而言的;

②C 语言本身不提供输入输出语句。输入输出的操作都是由 C 标准函数库中的函数实现的;

# 1、输出格式字符

printf 函数中用到的格式字符	
格式字符	说 明
d,i	以带符号的十进制形式输出整数(正数不输出符号)
o	以八进制无符号形式输出整数(不输出前导符 0)
x, X	以十六进制无符号形式输出整数(不输出前导符 0x),用 x 则输出十六进制数的 a~f 时以小写形式输出,用 X 时,则以大写字母输出
u	以无符号十进制形式输出整数
c	以字符形式输出,只输出一个字符
s	输出字符串
f	以小数形式输出单、双精度数,隐含输出 6 位小数
e,E	以指数形式输出实数,用 e 时指数以“e”表示(如 1.2e+02),用 E 时指数以“E”表示(如 1.2E+02)
g,G	选用%f或%E格式中输出宽度较短的一种格式,不输出无意义的 0。用 G 时,若以指数形式输出,则指数以大写表示

printf 函数中用到的格式附加字符	
字 符	说 明
l	长整型整数,可加在格式符 d、o、x、u 前面
m(代表一个正整数)	数据最小宽度
n(代表一个正整数)	对实数,表示输出 n 位小数;对字符串,表示截取的字符个数
—	输出的数字或字符在域内向左靠

如果要输出%,则需要连续输入两个;



## 2、输入格式字符

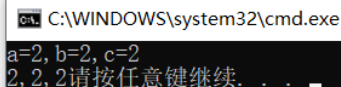
scanf 函数中用到的格式字符	
格式字符	说 明
d, i	输入有符号的十进制整数
u	输入无符号的十进制整数
o	输入无符号的八进制整数
x, X	输入无符号的十六进制整数(大小写作用相同)
c	输入单个字符
s	输入字符串, 将字符串送到一个字符数组中, 在输入时以非空白字符开始, 以第一个空白字符结束。字符串以串结束标志'\0'作为其最后一个字符
f	输入实数, 可以用小数形式或指数形式输入
e, E, g, G	与 f 作用相同, e 与 f, g 可以互相替换(大小写作用相同)

scanf 函数中用到的格式附加字符	
字符	说 明
l	输入长整型数据(可用 %ld, %lo, %lx, %lu)以及 double 型数据(用 %lf 或 %le)
h	输入短整型数据(可用 %hd, %ho, %hx)
域宽	指定输入数据所占宽度(列数), 域宽应为正整数
*	本输入项在读入后不赋给相应的变量

①scanf 别忘记了&取地址符;

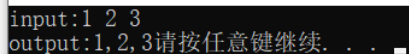
②格式对应原则:

```
int main()
{
    int a, b, c;
    scanf("a=%d, b=%d, c=%d", &a, &b, &c);
    printf("%d, %d, %d", a, b, c);
    return 0;
```



③int 型格式没分隔的时候, 输入之间必须由空格, 否则会当作一个数值:

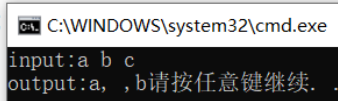
```
int main()
{
    int a, b, c;
    printf("input:");
    scanf("%d%d%d", &a, &b, &c);
    printf("output: %d, %d, %d", a, b, c);
    return 0;
```



④如果声明接收的为%c, 则不能有空格, 可以看到, 当由空格之后输入的字符没有接收

完全，这是因为字符型本身就是一位一位的读取存放的，空格也属于字符型，但是如果在 scanf 中读取的时候就是 scanf("%c %c %c", &a, &b, &c)的话，此时可以有空格（原则就是完全匹配）：

```
int main()
{
    char a, b, c;
    printf("input:");
    scanf("%c%c%c", &a, &b, &c);
    printf("output:%c,%c,%c", a, b, c);
    return 0;
}
```



⑤输入数值数据的时候，如果遇到了空格，回车或者 Tab，则会认为存放的当前位数据结束，进入读取下一位的存放，如果输入了字符的话，若是在末尾输入，则认为数据结束接收，若是在前面位或中间位则会报错；

### 3、getchar 以及 putchar

getchar 是读取一个字符，putchar 是输出一个字符；

当输入 getchar 的时候，并不会直接就读取，当按下 enter 之后才会得读取，这说明 getchar 输入之后其实是存放在缓冲区的。然后一个一个的赋值给相应的变量；（getchar 不仅会获取一个可显示的字符，还可以获得控制字符等无法显示的字符）

putchar (getchar()) 也是可以的；

输入完单独一个还有其他变量等待输入的时候不要按回车键，这个时候会把回车键当作一个字符存放在下一个变量中；

# 第四章：选择结构程序设计

## 选择结构和条件判断

C 语言有两种选择语句：(1) if 语句；(2) switch 语句；

### 用 if 语句实现选择结构

if 语句中的“表达式”可以是关系表达式、逻辑表达式、数值表达式。

else 子句不能单独使用，它必须和 if 一起使用，else 是 if 语句的一部分，也就是说，并不是 if 一结束 if 语句就完了，还需要看有没有 else；

if...else if...这个语句中结尾可以没有 else；

if 语句中，判断条件的结果只有两种，那就是条件为真和条件为假；

## 关系运算符和关系表达式

### 1、C 语言中共有六种关系运算符

小于 (<)、小于或等于(<=)、大于(>)、大于或等于(>=)；(优先级相同，且高)；

等于 (==)、不等于 (!=)；(优先级相同，但比上面四个低)；

**优先级：算术运算符>关系运算符>赋值运算符；(这些运算结合方向都是自左向右)**

### 2、关系表达式

使用关系运算符将两个数值或数值表达式连接起来的式子称为关系表达式；

## 逻辑运算符和逻辑表达式

用逻辑运算符将关系表达式或其他逻辑量连接起来的式子就是逻辑表达式；

如：与 (&&) 或 (||) 非 (!)；

Note:

&&	&	&			!	~
与	取地址	按位与	或	按位或	逻辑非	按位取反

当有多个逻辑运算符时: ! > && > ||;

优先级：非 (!) = 按位取反 (~) > 算术运算符 > 关系运算符 > &&、|| > 赋值运算符；

逻辑运算符也是自左向右；

逻辑运算值出来的最终结果只能是 1 或者 0，系统对于逻辑运算出来的值是通过 0 或者非 0 判断的；

例如：5 > 3 && 8 < 4 - !0

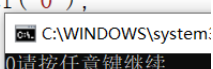
里面优先级最高的是!，先执行! 0，得 5 > 3 && 8 < 4 - 1

接下来优先级最高的是算术运算符，执行 4-1，得 5 > 3 && 8 < 3

接下来优先级最高的是关系运算符，执行 5>3，8<3，得 1&&0

则可以得到该运算后为得最终值为 0；

```
int main()
{
    if (5 > 3 && 8 < 4 - !0)
        printf("1");
    else
        printf("0");
    return 0;
}
```




## 条件运算符和条件表达式

表达式 1? 表达式 2: 表达式 3

例如：

```
int main()
{
    int a = 5, b = 3;
    (a > b) ? printf("max = %d", a) : printf("max = %d", b);
    return 0;
}
```



Note:

条件选择语句得执行部分只能放独立的表达式，不能放复合语句；

## 选择结构的嵌套

如果 if 语句中还包含了一个或多个 if 语句，则称之为语句嵌套；

需要注意的是，else 总是与离他最近的未配对的 if 配套；当然，如果想人为的配套，则可以使用花括号使需要配套的部分对应起来；

## switch 语句

switch 是多分支选择语句；

样式：

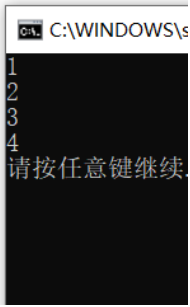
switch(表达式)

```
{
    case 常量 1: 语句 1    //语句就是表达式加分号
    case 常量 2: 语句 2
    ...
    default: 语句 n+1
}
```

Note:

- ①上述中的 switch 中表达式的值应为整型或字符型;
- ②每一个 case 中的常量必须是绝对的唯一, 不可以出现相同;
- ③case 相当于入口, 会根据 switch 判断的结果进入相应的入口;
- ④如果 case 进入之后后面无 break, 则按照顺序继续执行且**不需要判断**, 如果有, 则跳出当前的 switch 结构;

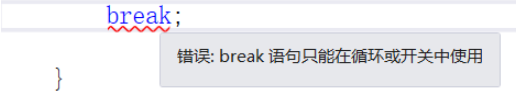
```
int main()
{
    int a = 1;
    switch (a)
    {
        case 1:printf("1\n");
        case 2:printf("2\n");
        case 3:printf("3\n");
        case 4:printf("4\n");
    }
    return 0;
}
```



## if 与 switch 的异同 (这个认知是错误的, 及时的更改)

对于 if...else if...else if...此类语句, 判断 if 之后, 即使进了 if, 仍然会在执行完 if 之后继续判断下一个, 而对于 switch, 当有 break 的时候, 会直接跳出, 不会继续执行判断; 而且需要注意, **if 中使用不了 break**; (这是错误的, 事实上, if...else if...这种语句只会进入其中一个, 并不会连续判断)

```
int main()
{
    int a = 1;
    if (a > 1)
        printf(">1");
    else
    {
        printf("test");
        break;
    }
    return 0;
}
```



# 第五章：循环结构程序设计

## 三大循环语句

### 1、while

一般形式：while（表达式） 语句

while 是先判断在循环，只要当循环条件表达式为真，就执行循环体语句；

```
int main()
{
    int add=0; //此处为0相当于0+1+2+...+100，为1才为1+2+...+100，结果一样，但是逻辑不同
    int num=0;
    while (add <= 100)
    {
        num = num + add;
        add++;
    }
    printf("%d", num);
    return 0;
}
```

### 2、do...while

其与 while 不同的是，while 先判断，但是 do 是先执行；其余一样；

### 3、for

for(表达式 1; 表达式 2; 表达式 3)

表达式 1：设置初始条件；

表达式 2：循环条件表达式；

表达式 3：循环修正，即循环的调整；

**for 语句的执行顺序：**

先执行初始化，然后进行判断语句，判断成功后执行循环体的语句，执行完毕之后进行修正；

### 4、for 与 while

1、for(表达式 1; 表达式 2; 表达式 3) 语句

无条件等价于

表达式 1;

while (表达式 2)

{

语句

表达式 3;

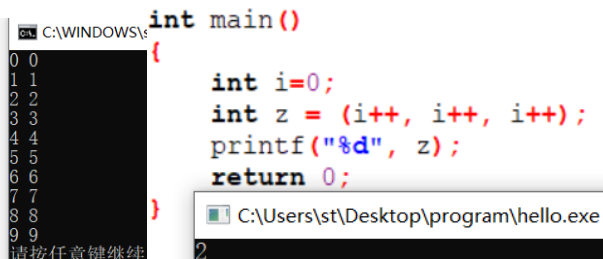
}

2、for 中表达式 1、2、3 都是可以省略的，但是其分号不能省略；

3、表达式 1 不一定非是赋初值，也可以是其他值；

4、表达式 1、3 中也可以用逗号表达式，逗号表达式一定要用括号括起来，如 (i=0, j=0;;)，这也是可以的。同时经过验证，表达式 2 也可以使用逗号表达式；（特别需要注意的是，逗号表达式运算后的结果为最右边的表达式的值，在下题中，num<5 根本没有使用上，更重要的是，取得是最右边的值，但是从左往右执行的时候前面的表达式仍会执行）

```
int main()
{
    int add=0;
    int num=0;
    for (; num < 5, add<10; num++, add++)
        printf("%d %d\n", num, add);
    return 0;
}
```



```
int main()
{
    int i=0;
    int z = (i++, i++, i++);
    printf("%d", z);
    return 0;
}
```

## 循环：


循环中嵌套循环就是循环的嵌套；

这三种循环都可以使用 break 提前退出，也可以使用 continue 提前退出本次循环；

break 是不能单独使用的，只能用在 switch 以及循环中；

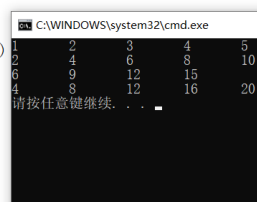
## break 以及 continue 的区别

```
int main()
{
    int i, j;
    for (i = 1; i < 5; i++)
    {
        for (j = 1; j < 6; j++)
        {
            if (i == 3 && j == 1)
                break;
            printf("%d\t", i*j);
        }
        printf("\n");
    }
    return 0;
}
```



```
1 2 3 4 5
2 4 6 8 10
3 6 9 12 15
4 8 12 16 20
5 10 15 20 25
```

```
int main()
{
    int i, j;
    for (i = 1; i < 5; i++)
    {
        for (j = 1; j < 6; j++)
        {
            if (i == 3 && j == 1)
                continue;
            printf("%d\t", i*j);
        }
        printf("\n");
    }
    return 0;
}
```



```
1 2 3 4 5
2 4 6 8 10
3 6 9 12 15
4 8 12 16 20
5 10 15 20 25
```

分析：当遇到 break 时，会直接跳出 j 层的循环，直接换行后 i+1，所以中间空了出来；但是遇到 continue 只不过退出了 j 层中当前的循环，会继续 j=2 的地方继续进行；

## 第六章：利用数组处理批量数据

数组：

- ①数组是一组有序数据的集合；
- ②用一个数组名和下标唯一的确定数组中的元素；
- ③数组中的每一个元素都属于同一个数据类型；

### 怎样定义和引用一维数组

定义：类型符 数组名[常量表达式]；（因为是常量表达式，所以 `a[3+5]`也是允许的）

引用：数组名[下标]

定义和引用时候的形式相同，但是含义不同，定义的时候 `int a[10]`含义为定义了一个包含 10 个元素的 `int` 型的数组；而 `t=a[6]`相当于引用 `a[6]`中的值赋值给 `t`；

初始化：

- ①在定义数组的时候可以初始化：  
`int a[5] = {0,1,2,3,4};`
- ②可以只给数组中的一部分赋值：  
`int a[5] = {0,1};`
- ③给数组中每一个都赋值为 0：  
`int a[5] = {0};`

如果在初始化的过程中，指定了长度但是并没有完全的赋值，则对于 `int` 行数组而言其它位默认为 0，如果是 `char` 型数组则默认为 `'\0'`，如果是指针型数组则默认为 `NULL`（空指针）；

### 二维数组

定义：类型说明符 数组名[常量表达式][常量表达式]；

在内存中，二维数组是按照行的顺序存放的，先存放第一行，然后以此存放；也就是说，二维数组其实也是按照线性存放的，不过人为的按照矩阵的形式表示而已；三维数组同理；

引用：数组名 [下标][下标]，类似于一维数组；

初始化：

- ①在定义数组的时候可以初始化：  
`int a[3][4] = {{1,1,1,1}, {1,1,1,1}, {1,1,1,1}};`  
`int a[3][4] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};`
- ②可以只给数组中的一部分赋值：  
`int a[3][4] = {{1}, {}, {0, 1}};`

如果对全部元素都赋值的话是可以省略第一维的数据的；

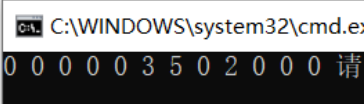
同理：`int a[][4] = {{1}, {}, {0, 1}};`也是可以的，相当于一维默认为 3 了；



```

int main()
{
    int a[][4] = { {}, {0, 3, 5}, {2} };
    int *p = a[0];
    for (int i = 0; i <= 11; i++)
    {
        printf("%d ", *(p++));
    }
    return 0;
}

```



## 字符数组

### 1、字符数组的基础内容

C 语言中没有字符串类型，也没有字符串变量，字符串都是存放在字符型数组中的；

定义字符数组和整型数组类似，当然，其实也可以用整型数组去存放字符型数组，但是完全没必要，因为太浪费空间了；

各项初始化与整型数组类似，不多赘述；二维字符数组的定义以及引用类似于二维整型数组；

**Note:** 字符数组与字符串常量其实并无太大差别，区别在于当用字符数组存放字符串常量的时候会自动加一个'\0'，也就是说，如果定义了一个十位空间的字符数组，存放字符串常量的时候只能存放 9 个元素，因为需要空一位存放'\0'；如果存放了 10 个则会报错；

在 ASCII 码值中，'\0'不会起任何作用，仅可以作为一个标志；

等价性：

char c[6] = "ABCDE";

等价于：char c[6] = {'A', 'B', 'C', 'D', 'E', '\0'};

不等价于 char c[6] = {'A', 'B', 'C', 'D', 'E'};

字符数组是否添加'\0'完全取决于自身的需求；

### 2、字符数组的输入输出

**如果按照%s 输出，那么遇到\0 就会自动停止了；**

```

int main()
{
    char a[10] = {'C', 'H', 'I', 'N', 'E', '\0', 'A', 'B', 'C'};
    printf("%s", a);
    return 0;
}

```



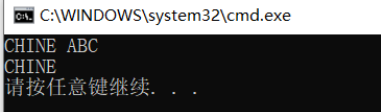
字符数组的输入输出有两种：%c 与%s；

**Note:**

①用%c 的话输入的是按照数组存放，但如果是按照%s 输入的话，那就是按照字符串存放，也就是会自动在末尾加\0；

②按照字符串输出则会停止在\0 之前，但如果是字符输出则会全部输出：

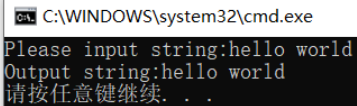
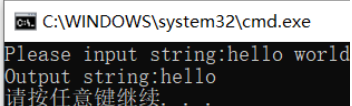
```
int main()
{
    char a[10] = {'C','H','I','N','E','\0','A','B','C'};
    for (int i = 0; i <= 9; i++)
    {
        printf("%c", a[i]);
    }
    printf("\n%s\n", a);
    return 0;
}
```



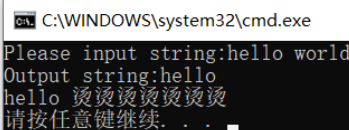
③输出字符中不会包括\0;

④如果需要用 scanf 读取%s 存放进字符串数组的话，千万别忘记需要给\0 留一位；同时，如果要连续存放多个%s，则需要用空格隔开；

⑤为什么会出现下面右图那样的状况呢？这是因为当输入 hello 之后空格的话，系统会默认的将 hello 作为一个完整的字符串，这个时候就会在后面添加\0，那么后面的会存放进去吗？由最下面的图可知，当\0 之后后面的值不会存放进去，也就是仍然是不清楚的值，为什么是不清楚的值呢？这是因为没有赋初值，如果赋初值的话那么就不会是烫烫烫了；

<pre>int main() {     char a[10],b[10];     printf("Please input string:");     scanf("%s%s", a,b);     printf("Output string:");     printf("%s %s\n", a, b);     return 0; }</pre>		<pre>int main() {     char a[10];     printf("Please input string:");     scanf("%s", a);     printf("Output string:");     printf("%s\n", a);     return 0; }</pre>	
--	---	--	--

```
int main()
{
    char a[20];
    printf("Please input string:");
    scanf("%s", a);
    printf("Output string:");
    printf("%s\n", a);
    for (int i = 0; i <= 19; i++)
    {
        printf("%c", a[i]);
    }
    printf("\n");
    return 0;
}
```



⑥如果用%s 输入的话，那么是不需要加&的，因为数组名本身就是地址；

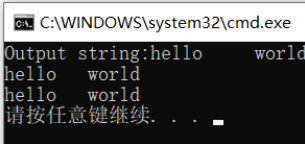
### 3、使用字符串处理函数

puts 函数：puts 可以输出字符串中的转义字符，printf 也是可以的；

```

int main()
{
    char a[20] = "hello\tworld";
    printf("Output string:");
    printf("%s\n", a);
    for (int i = 0; i <= 19; i++)
    {
        printf("%c", a[i]);
    }
    printf("\n");
    puts(a);
    return 0;
}

```



gets 函数: gets 函数会读取缓冲区中的字符串存放在数组中, **而且对于同一个数组, 可以存放空格, 并不像 scanf 一样遇见空格会默认认为字符串接收完毕, 别忘记字符串都会在后面添加\0; 所以数组可存放大小一定要大于需存放的数据的最大值;**

strcat (a, b): 将字符串 a 与 b 连接起来;

连接之前都有\0, 在连接之后只在新字符串后添加;

strcpy (a, b) :将字符串 b 的值赋给 a, a 只能是数组名, 但是 b 可以是数组名也可以是字符串; **cpy 的时候会将\0 也一起复制过去;**

strncpy (a, b, int size) :将字符串 b 的值前 n 个数据赋给 a; **这个 cpy 的时候不会默认复制\0, 但是如果 cpy 的时候 size 刚好在\0 的话, 就会复制过去了;**

strcmp (a, b): 比较 a 和 b 是否相同, 相同返回 0, 不相同的时候, 如果 a>b, 则函数返回一个正整数, 反之返回一个负整数; (返回的值是按照相应对照字符之间的 ASCII 码值的差对比的);

strlen: 测量字符串长度的函数, 不包括\0, 也就是说, 它的值实际比实际存储的时候小 1, 但是输出的是字符串的实际长度;

strlwr: 字符串中的大写转换为小写;

strupr: 字符串中的小写转换为大写;

# 第七章：用函数实现模块化程序设计

## 为什么要用函数

函数的声明必须放在**主函数前面或者主函数的开头部分**；

Note：

①一个 C 程序一般由一个或多个程序模块组成，每一个程序模块作为一个源程序文件，一个源程序文件可以为多个 C 程序共用；

②一个源程序文件由一个或多个函数以及其他有关内容（如指令、数据声明与定义等）组成；

③函数可以互相调用，但是不能函数嵌套定义，但是可以嵌套调用，也可以嵌套声明，其他函数也不能调用主函数；

④从用户使用的角度上看：函数分为库函数以及用户自定义的函数；

⑤从函数的形式看：函数分为无参函数以及有参函数；

空函数的作用：**空函数可以用来在程序编写初始阶段占一个位置，有助于增加函数的可读性，结构也会更加清晰，以后扩展新功能也方便**；

## 函数调用时候的数据传递

### 1、形参与实参

形式参数与实际参数：在定义函数时函数名后面括号中的变量名称为“形参”，而在主函数中调用一个函数的时候，函数名后面括号中的参数称为“实参”。实参可以是常量、变量、表达式，但要求它们要有确定的值；而且实参与形参的类型应该相同或者赋值兼容；调用函数的时候，系统会把实参的值传递给被调用函数的形参。也可以说是形参从实参处得到一个值；

### 2、函数调用的过程

1、在函数中指定形参之后，当函数未被调用的时候，这些形参是不会占用内存中的存储单元的，只有发生调用之后才会被临时分配内存单元；

2、调用的时候首先会将实参的值传递给形参；

3、在进入被调用函数之后，此时形参中已经有值了，就可以利用形参进行相关的操作；

4、当被调用函数执行完毕之后，会利用 return 语句返回相应的返回值（如果有返回的话），返回值的类型必须跟函数类型一致；

Note：数据的传递只能是单向传递，即只能由实参传递到形参；

## 被调函数的声明和函数原型

函数的首行称为函数原型，在加一个分号就成了函数的声明；

为什么选择函数的首部作为函数的声明呢？因为函数首部的包括了函数是否合法的基本信息，包括函数名、函数值类型、参数个数、参数类型和参数的顺序，在检查函数调用的时候要求这些信息必须与函数声明一致，这样就可以对函数调用的合法性进行检查；

函数的声明与定义也是不同的，**什么是定义**呢？是对函数基本信息的确定，包括指定函数名、函数值类型、形参以及函数体等等、这是一个完成、独立的函数单位。那**什么是声明**呢？声明是把定义的信息通知编译器的，以便在调用该函数的时候确定是否调用正确，函数的声明不包括函数体；

## 函数的递归调用

在调用函数的过程中如果又出现直接或间接的调用了该函数本身，就称为函数的递归调用；

递归调用不能无限制的调用下去，可以用 if 语句去控制使其执行有限次；

递归相当于梦中梦中…梦，每次退出之后其实只是退出了当前的梦，但还是在上一个梦中，所以退出时需要一级一级的往回退；在实际过程中，如果反推执行执行之后就会正向回溯，这个时候就会一级一级的回溯，每回溯一级就会将值做出实际的更改，这就相当于由结果与条件的关系以及初始值去反推第一步与所求值之间的关系，然后又反过来正向求出实际的结果；对于汉诺塔的详细分析见小知识总结；

## 数组作为函数参数

数组名可以当作函数参数，此时传递的是数组第一个元素的地址；如果是数组元素的话仅与变量相同；

用数组元素做函数实参的时候，是把实参的值传给形参，也是“值传递”的方式，数据传递的方向是从实参传到形参，单向传递；

如果传递的是一维的数组名的话，其形参可以为 `int *p`，也可以是 `int p[10]`，这是等价的，这是为什么呢？因为再数组传递的过程中，其本质都是再传递数组的首元素的地址，也就是本质还是在传递指针；（假设有一个 `int` 型的十位元素数组）；如传递的是二维数组的数组名，**形参就只能为 `int p[3][4]`，调用的时候只需要 `function (a)` 就可以，不需要 `function (a[0])`；**

如果要用指针指向二维数组，那**就必须初始化的时候为 `int *p = a[0]`**，必须把首行第一个的地址传递给指针，不需要说明是第一个元素；

一维数组：形参可以为 `int *p`，也可以是 `int p[10]` 或 `int p[ ]`，实参是 `p`，要传递的也是地址；

二维数组：形参 `int p[3][4]` 或 `int p[ ][4]`，实参是 `p` 就可以；

## 局部变量与全局变量

定义变量的情况：

- ①函数的内部的开头定义
- ②函数的外部定义
- ③函数内部的复合语句中定义

如果是函数内部开头，则变量在本函数内有效，如果是外部，则是全局变量，如果是函数内部的复合语句中的话，则只在本复合语句中有效；


形参也是局部变量，仅在被调用的函数内部有效；不同函数中是可以使用相同变量名的函数的；

如果定义了全局变量 a，但在函数中也定义了 a，那么当进入函数的时候，使用的 a 是函数内部定义的局部变量，而不是那个全局变量 a；

全局变量的作用域其实可以说是在定义之后开始的，并不一定是开头，比如在两个函数体中间定义的话，它的作用域就不会包括上面的函数体；

```
int a = 100;
int fun(int a)
{
    return a+1;
}

int main()
{
    int a = 20;
    int b = 2;
    b = fun(b);
    printf("%d\n", b);
    return 0;
}
```



如果要通过一个函数要得到好几个值得话，那就考虑全局变量；

为什么不建议在不必要得时候使用全局变量：

- ①全局变量在程序全部执行过程中都需要占用存储单元；
- ②全局变量使得函数得通用性降低，因为在使用过程中如果使用全局变量的话，那么执行情况就会受到有关的外部变量的影响，降低了程序的可靠性与通用性，同时全局变量会导致函数的“内聚性”变弱，与其他模块的“耦合性”太强，与其他模块的相互影响有点多；
- ③使用了过多的全局变量会降低程序的清晰性；

## 变量的存储方式和生存期

从变量值的存在的时间，即生存期来观察，可以将变量的存储分为两种不同的方式：静态存储方式和动态存储方式；

静态存储方式：在程序运行期间由系统分配固定的存储空间的方式；

动态存储方式：在程序运行期间根据需要进行动态的分配存储空间的方式；

在内存中供用户使用的存储空间可以分为 3 部分：

- ①程序区
- ②静态存储区
- ③动态存储区

程序的数据分别存放在静态存储区和动态存储区；

全局变量分配在静态存储区；

动态存储区存放：函数形参，函数中没有用 static 声明的变量，函数调用时的现场保护和返回地址；

C 语言中的每个变量与函数都有两个属性：数据类型以及数据的存储类别；（其中数据的存储类别指的就是数据在内存中存储的方式）

C 的存储类别包括四种：auto 自动的，statics 静态的，register 寄存器的，extern 外部的；

## 1、局部变量的存储类别

自动变量（auto）：如果在函数中的局部变量不专门声明为 statics 的话，那么就会动态的分配存储空间，数据存储 in 动态存储区中，这类变量在调用时会分配存储空间，调用结束时会自动释放，所以这类局部变量也被称为自动变量；在一般意义上，auto 可以省略，不声明 auto 的话就会隐含的指定为自动变量；（自动变量时一个不确定的值，如果不赋初值直接引用的话会报错）

静态局部变量（static 局部变量）：该类变量在函数调用结束后不会释放内存，会继续存在着，保留原值，但是不论什么时候，虽然在函数调用结束之后该变量仍存在，但是该变量仍然是局部变量，其他函数不能引用；（静态局部变量在编译时候会赋初值且只会赋值一次，如果不赋初值的话就会自动赋初值为 0 或 '\0'）

寄存器变量（register）：一般情况下，变量都是存放在内存中的，但是如果有一些变量频繁的被使用的话，那么存取变量的值就需要花费太多的时间。为了提高效率，所以将局部变量的值放在 CPU 中的寄存器中，需要的时候会直接从寄存器中取值，提高执行效率；（如今计算机一般都会自动的将使用频繁的变量放进寄存器中）

上述三种存放的位置不同：auto 存放在动态存储区，statics 存放在静态存储区，register 存放在寄存器中；

## 2、全局变量的存储类别

一般来说，全局变量的作用域是从变量的定义处开始，到本程序文件的末尾；如果想要扩展该变量的作用域，可以用以下方法：

1、extern：外部变量声明，表明把该外部变量的作用域与扩展到当前所定义的位置；

通常将外部变量的定义会放到引用它的所有函数的前面，这样就可以避免在函数中多添加一个 extern；如过声明该值是外部变量的时候，例如 extern int a；这个时候 int 是可以不需要的，因为 a 在外部已经是定义好的，此处只不过是声明以下而已；

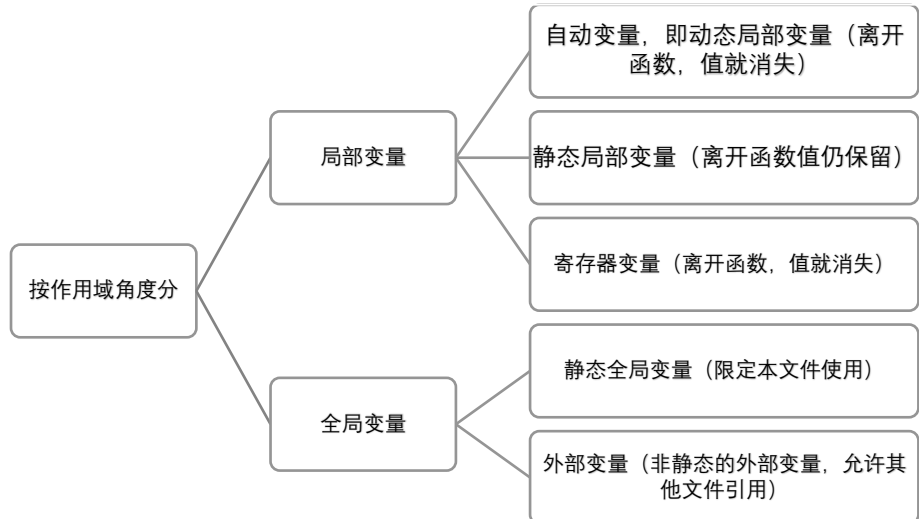
2、将外部变量的作用域扩展到其他文件也可以使用 extern；

3、将外部变量的作用域限制在被本文件中使用的是 static，此时是静态外部变量（注意，这与静态局部变量是不同的），例如声明了一个全局变量 static int a，此时加上 extern 也只能在本文件使用，其他文件是使用不了的，这就相当于将范围限制在了本文件中；

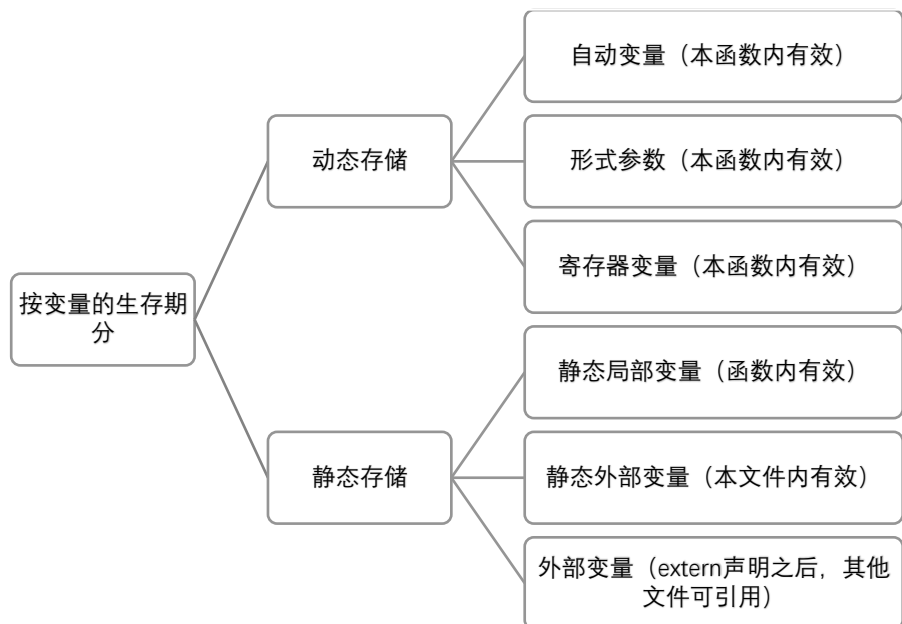
不要认为全局变量声明了 static 之后才是静态，这是错误的，全局变量从申请初始就会存放在静态区域；

### 3、存储类别小结

1、从作用域角度划分：（局部变量与全局变量）

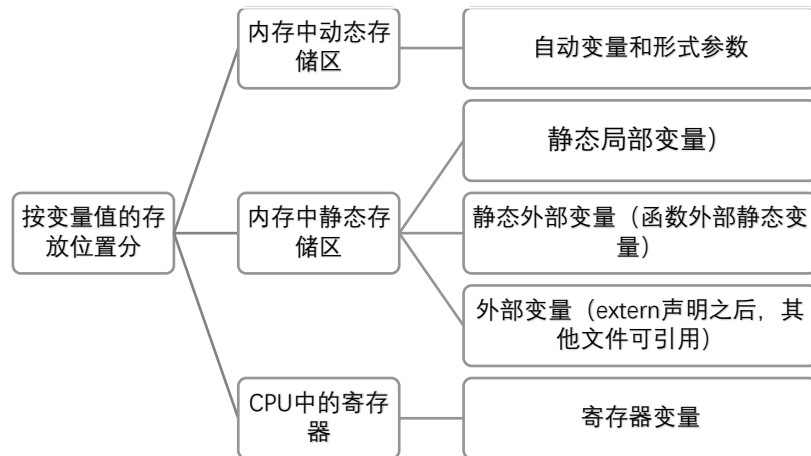


2、从变量存在的时间（生存期）区分：（动态存储与静态存储）



3、从变量值的存放位置来区分：





4、如果变量在某个文件或函数范围内使有效的，则称该范围使该变量的作用域，也称作变量在此作用域中“可见”，这种性质也被称为可见性；如果一个变量在某一时刻使存在的，则认为这一时刻属于该变量的生存期，也就该变量在此时刻“存在”；

存在看的是占据了空间是否没释放，作用域看的是变量是否能被引用；

## 关于变量的声明与定义

声明：不需要建立存储空间的声明就是声明；（例如 `extern a`；这是直接从外部引用，不需要申请空间）

定义：需要建立存储空间的声明；（例如 `int a`；需要申请对应的空间区存放）

在一个函数中出现的对变量的声明，除外部变量之外其余的都是定义。在函数中对其他函数的声明不是函数的定义；

## 内部函数与外部函数

如果函数只能被本文件中其他函数所调用的话，那它就是内部函数，这个时候需要在定义函数的时候在函数名以及函数类型前加上 `static`；（例如 `static int fun(int a)`）

如果在定义函数的时候在函数名以及函数类型前没添加任何关键字或者加上 `extern` 的话，则称为外部函数；（例如 `extern int fun(int a)`或 `int fun(int a)`）

# 第八章：善于利用指针

## 指针是什么

通过地址可以找到所需的变量单元，所以可以说地址指向该变量单元。将地址形象化的称为“指针”；

如果只是单传的地址信息是没有用的，因为不知道指向的数据的类型，也就不知道是几位，所以 C 语言中的地址需要包括位置信息以及所指向的数据的类型信息（地址也被称为带类型的地址）；

在实际工作中，对变量的访问都是通过地址进行的；

如果是直接按变量名进行的访问，就被称为“**直接访问**”，而如果是将变量 i 的地址存放在另一个变量 p 中，然后通过 p 找到 i 的地址从而访问 i 变量的话，就被称为“**间接访问**”；

变量名并不会占用内存空间，其本身就是一个标识符；

如果变量专门用来存放另一个变量的地址，则其称为“**指针变量**”（指针与指针变量是不同的，指针指的就是地址，而指针变量指的是存放另一个变量地址信息的变量）；

## 指针变量

一个变量的指针的含义包括两个方面，一个是以存储单元编号表示的纯地址，另一个则是它指向的存储单元的数据类型；

指针变量中只能存放地址数据，不能赋值；

printf(“%o”, p);这个语句可以输出 p 指向的变量的地址数据；

不能通过值传递改变实际变量的值，同时也不能通过改变指针形参的值去使指针实参的值改变（如下，这种传递只是改变了形参的值并不会改变实参的值）；

（当实参传递过来被形参接受之后，后续所有的交换只不过都是在交换形参而已，但是形参又会随着函数的调用结束而被释放，故在被调用函数中的值的形参本身的交换是不会对之前的地址中的数据起到作用的，除过是直接交换了形参指向空间的值）

```
void swap(int *pa, int *pb)
{
    int *p;
    p = pa;
    pa = pb;
    pb = p;
}
```

## 通过指针引用数组

数组元素的指针就是数组元素的地址；

对于数组 int a[10]；其中 a 是数组名，&a[0]是数组首地址，在实际运行中，a 的值就是&a[0]；

对于指向 int 型变量（假设 int 每位占 4 字节）的指针 p，当进行 p++的时候，就相当于对 p 中保存的地址信息加 4，而如果是 char 的话就相当于加 1；（加多少完全是由指针初

始化定义时候的类型决定的)

在运用中, \* (p+i) 或者\* (a+i) 指向的都是 a[i]中的数据;(在计算机实际的运行中, 其实对于 a[i]本质上就是按照\*(a+i)计算的)

在两个指针指向同一个数组的时候, 例如对于指向 float 型的 p1, p2 指针, 若 p1 指向 a[5], p2 指向 a[8], 此时两者做差的值等于两者实际差除以数组元素的长度, 通过计算就可以知道两者之间的相对位置;

```
int main()
{
    int x, y;
    float arr[10] = {2,1,5,7,0,3,8,9,6,10};
    float *pa = &arr[8];
    float *pb = &arr[5];
    printf("%d", pa-pb);
    return 0;
}
```

数组名和数组指针不同的地方在于、**数组指针的值可以改变**, 即可以指向不同的位置, **但是数组名本质上是一个指针型常量**, 它并不是指针变量, 不可以改变;

如果需要指针在位移之后重新指向数组起始位置, 只需要**重新赋值**就可以了, 在指针位移发生之后重新 p=a 赋值就可;

当 p 指向的是数组起始位置的时候, 此时可以使用 p[i], 其本质相当于 a[i], 也相当于 \* (p+2);

对于\*p++, 其相当于\* (p++), 因为++与\*的优先级相同, 此时会先++地址, 然后取值;  
当进行++ (\*p) 的时候, 这就相当于对 p 指向的变量的值进行增加, 例如之前 a[i]是 3, 在执行完此语句后就会变成 4;

## 用数组名做函数参数

如果 swap(a[0], a[2]), 此时只相当于将 a[0]和 a[2]的值给进行了传递, 并不是地址;

以变量名和数组名作为函数参数的比较		
实参类型	变量名	数组名
要求形参的类型	变量名	数组名或指针变量
传递的信息	变量的值	实参数组首元素的地址
通过函数调用能否改变实参的值	不能改变实参变量的值	能改变实参数组的值

实参数组名代表一个固定的地址, 也就是一个指针常量, 而形参数组名代表的并不是一个固定的地址, 而是按照指针变量处理(所以可以发现, 实参中的数组名不可以被再次赋值, 而形参中的数组名实可以被重新赋值);

## 通过指针引用多维数组

二维数组 a 的有关指针

表示形式	含 义	值
a	二维数组名,指向一维数组 a[0],即 0 行起始地址	2000
a[0], *(a+0), *a	0 行 0 列元素地址	2000
a+1, &a[1]	1 行起始地址	2016
a[1], *(a+1)	1 行 0 列元素 a[1][0]的地址	2016
a[1]+2, *(a+1)+2, &a[1][2]	1 行 2 列元素 a[1][2]的地址	2024
*(a[1]+2), *((a+1)+2), a[1][2]	1 行 2 列元素 a[1][2]的值	是元素值,为 13

一维数组中的 a[1]即\*(a+1), 其直接指向的是其中的元素值, 但是对于二维数组来说, 二维数组中的 a[1]即\*(a+1), (a+1)本身就不是元素值, 而是第一行元素的起始地址信息;

也就是说\*\*\*(a+1)才等于 a[1][0], 也相当于\*a[1];

二维数组中 a[i]、&a[i]、a+i、&(a+i)、&a[i][0]的值是相等的, 但是他们指向的对象类型不同; (例如 a, \*a, 两者值是相同的, 到那时 a 的意思是 0 行起始地址, 而\*a 意思是 0 行 0 列元素地址)

二维数组名是指向行 (一维数组的), 当给二维数组名加 1 的时候, 会直接加上一行的全部元素的字节, 直接指向下一行的起始地址;

\*(a+1)+1 相当于(&a[1][0])+1;

## 指向多维数组元素的指针变量

如果定义了 `int *p`, 则 p 会将二维数组看成一维数组去运作, 例如对于数组 p 指向数组 a[3][4]的起始位置, 则 p++就会从 a[i][j]指向 a[i][j+1], 当 j+1 等于 3 之后, 即 i 行读取完之后, 下次会自动指向 a[i+1][0];

如果定义的是 `int (*p)[4]`, 当 p 指向数组 a[3][4]的起始位置之后, 则 p++就会从 a[i][0]指向 a[i+1][0], 如果其为形参的话, 那么实参需要给传递 a, 而不能给传递\*a, 因为 a 代表的是 0 行的起始地址, 而\*a 传递的是 0 行 0 列元素的地址, 这两者传递的基类型是不同的;

注意: \* ((p+1)+1) 与\* ((p+1)+1) 是不同的, 虽然\*(p+1)与(p+1)的值是相同的, 但是其指向的类型是不同的, \* ((p+1)+1) 表示的是地址, 而\* (\*(p+1)+1) 表示的是值;

**标志位思想;**

## 通过指针引用字符串

C 语言中, 字符会存放在字符数组中, 可以用一下两种方法引用字符串:

- 1: 用字符数组存放一个字符串, 可以用下标引用, 也可以用%s 输出;
- 2: 用字符指针变量指向一个字符串常量 (C 中只有字符变量, 没有字符串变量), 通过指针引用;

引用方法 1: 先定义字符数组 buf, 随后指针指向 buf:

```
char buf[20] = "This is Hello world!";
char *p = buf;
```

引用方法 2: 直接定义字符指针变量:

```
char *p = "This is Hello world!";
```

对于引用方法 2，在实际运行过程中，C 语言对字符串常量是按照字符串数组处理的，当初始化之后就会在内存中开辟一个字符数组来存放字符串常量，但是这个字符数组是没有名字的，所以不能使用数字名来引用，只能通过指针变量来引用；

对于上述两种方法，第一种里面的值是可以改变的，但是对于第二种引用方法，是无法修改值的，因为第二种中 p 指针指向的是常量，常量是无法被修改的，第一种中 p 指向的是变量 buf；

**在指针引用的时候最关键的是一定要知道指针当前的位置在何处；**

数组可以在定义的时候统一赋值，但是不能用赋值语句对字符数组的全部元素则很难个体赋值，只能单个单个的赋值，例如：

```
char a[20];  
a = "Hello world!";           //非法的行为;  
a[0] = 'H';                   //合法的行为;
```

## 字符数组与字符指针变量的比较

- 1，字符数组由若干个元素组成，而字符指针中存放的是地址；
- 2，可以对字符指针变量赋值，但不能对数组名赋值；
- 3，编译的时候机器会为字符数组分配若干个存储单元，但对于字符指针变量指挥分配一个存储单元；
- 4，指针变量的值是可以改变的，而字符数组名代表的是一个固定的值（数组首元素的地址），不能改变；
- 5，只要使用了字符指针，那么就一定要及时的指定它的指向，否则很有可能造成系统损坏；

例如：`char *a;`  
`scanf("%s", a);` //错误，此时 a 还没有具体的指向；  
但对于：`char *a, str[10];`  
`a = str;`  
`scanf("%s", a);` //正确，此时 a 已经指向了 str；

- 6，字符数组中的值可以被改变，但是指针变量指向的字符串常量是不能被改变的；
- 7，下列这种操作也是符合的：

```
char *format = "a = %d, b = %d\n";  
printf(format, a, b);  
这种 printf 函数称为可变格式输出函数；
```

## 指向函数的指针

函数名带代表函数的起始地址，函数名也就是函数的指针；

```

int max(int x, int y) //max函数的定义
{
    if(y > x)
    {
        x = y;
    }
    return x;
}

int main()
{
    int x, y;
    int (*p)(int x, int y); //这行只能放函数里面
    p = max; //将p指针指向max函数的入口地址
    scanf("%d%d", &x, &y);
    printf("%d\n", (*p)(x, y)); //其中的(*p)(x,y)就相当于max(x,y)
    return 0;
}

```

指向函数的指针变量的一般形式：

类型名 (\*指针变量名) (函数参数列表);

Note:

- 1、定义指向函数的指针变量之后，如果要指向函数，也只能指向与其格式相同的函数；
- 2、使用指针调用函数之前，一定要使指针变量指向该函数（例如：p=max），在指向的时候不需要给出参数之类的信息，这只代表一个指向；
- 3、调用函数指针的时候只需要用(\*p)替换源函数名即可（例如：源：max(x, y)，调用：(\*p)(x, y)）；
- 4、通过指针变量调用函数比较灵活，可以根据情况的不同调用不通的函数；

## 指向函数的指针做函数参数

指向函数的指针的一个重要用途是把函数的入口地址作为参数传递到其他函数；

在调用函数 fun 的时候，如果传递的是另一个函数 min 的入口地址，那么在接下来的 fun 函数中就可以调用 min 函数了；

## 返回指针值的函数

形式：类型名 \*函数名 (参数列表)；

例如：int \*a (int x, int y)；

其中 a 是函数的函数名，int x, y 是其中的参数，而其最大的不同是返回值是一个 int 型的指针；

## 指针数组和多重指针

### 1、指针数组

一个数组，如果其元素都是指针类型数据的话，这就是**指针数组**，其本质是数组中的每一个元素都存放一个地址，相当于一个指针变量；

例如：int \*p[4];

因为[]比\*优先级高，所以 p 先于[4]结合，形成 p[4]形式，所以决定了其为数组形式，

随着该数组与前面的\*结合，表示该数组是指针类型的，每个数组元素都可指向一个整型变量；

注意不要写成 `int (*p)[4]`，这是指向一维数组的指针变量；  
一位指针数组的一般形式为：类型名 \*数组名[数组长度]；

## 2、多重指针

指向指针的指针就是二重指针；

## 3、指针数组做 main 函数的形参（不重视）

平常情况下 main 函数没有参数，但是在某些情况下，main 函数可以有参数；

`int main(int argc, char *argv[ ])`；

main 函数的实参只能由操作系统给出，在操作命令的状态下，在命令行中回包括命令名以及需要传给 main 函数的参数；

## 动态内存分配与指向它的指针变量

非静态的局部变量（包括形参）都是分配在内存中的动态存储区，这个区域被称为“栈”，除此之外，C 语言还允许建立内存动态分配区域，以存放一些临时的数据，这些数据不需要在程序的声明部分定义，也不需要等到函数结束时才释放，可以随时开辟，随时释放，他们存储在一个特别的自由存储区，称为“堆”，这个区域只能通过指针来引用；

## 1、建立内存的动态分配

主要有 malloc, calloc, free, realloc 这 4 个函数，包含在头文件 `#include <stdlib.h>` 中；

1、malloc 函数开辟动态存储区

函数原型：`void *malloc(unsigned int size)`；

作用：在内存的动态存储区中分配一个长度为 size 的连续空间，其中 size 不能为负数，其返回值为所分配区域的第一个字节的地址，也可以说这是一个返回值为指针型的函数，返回值会指向所分配区域的第一个字节；（例如：`p=malloc (100)`）；

2、calloc 函数开辟动态存储区

函数原型：`void *calloc(unsigned n, unsigned size)`；

作用：在内存的动态存储区中分配 n 个长度为 size 的连续空间，可以给数组分配；分配成功之后返回值为指向所分配区域的第一个字节的地址，若失败则返回 NULL；（例如：`p=calloc(50, 4)`）

3、realloc 函数重新分配动态存储区

函数原型：`void *realloc(void *p, unsigned int size)`；

作用：当已经通过 malloc 以及 calloc 获得了动态空间之后，如果想要改变其大小，则可以用 realloc 重新分配；（例如：`realloc (p, 50)`）

4、用 free 函数释放动态存储区



函数原型：void free (void \*p);

作用：释放指针变量 p 所指向的动态空间，使得这部分空间能重新被其他变量使用；  
(例如：free(p))

## 2、void 指针类型

void 类型并不是指向任何类型的数据，而是“指向空类型”或“不指向确定的类型”；

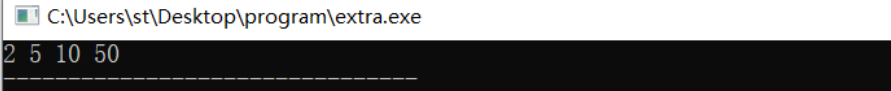
也就是说，在使用的时候，当将该指针的值赋值给另外指针变量的时候系统会对它进行类型转换，使之适合于被赋值的变量的类型；它只是一个单纯的地址，不含有特定类型的信息；说本质一些，void \*p 中的 p 指针只是过渡性的，它不可能存储信息；

指针变量的类型及含义

变量定义	类型表示	含 义
int i;	int	定义整型变量 i
int * p;	int *	定义 p 为指向整型数据的指针变量
int a[5]	int [5]	定义整型数组 a,它有 5 个元素
int * p[4];	int * [4]	定义指针数组 p,它由 4 个指向整型数据的指针元素组成
int (* p)[4];	int (*) [4]	p 为指向包含 4 个元素的一维数组的指针变量
int f();	int ()	f 为返回整型函数值的函数
int * p();	int * ()	p 为返回一个指针的函数,该指针指向整型数据
int (* p)();	int (*) ()	p 为指向函数的指针,该函数返回一个整型值
int **p;	int **	p 是一个指针变量,它指向一个指向整型数据的指针变量
void * p;	void *	p 是一个指针变量,基类型为 void(空类型),不指向具体的对象

```
int main()
{
    int *p[4]; //指针的集合, p[0]本身依然是指针, 指向的是buf1的首地址
    int buf1[5] = {1,2,3,4,5};
    int buf2[5] = {5,4,3,2,1};
    int buf3[5] = {10,20,30,40,50};
    int buf4[5] = {50,40,30,20,10};
    p[0] = buf1;
    p[1] = buf2;
    p[2] = buf3;
    p[3] = buf4;
    printf("%d %d %d %d", *(p[0]+1), *p[1], *p[2], *p[3]);

    return 0;
}
```





```

int main()
{
    int *pi;
    int a[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
    int (*p)[3]; //这儿的p本身代表的是每一行的首地址，每加一次会直接加到下一行的首地址
    p = &a[0]; //p=a也可以，但是p=a[0],p=&a[0][0]也不行
    printf("%d\n", *p[0]);
    printf("%d\n", *p[2]);
    pi = p[2];
    printf("%d %d %d", pi[0], pi[1], pi[2]);
    return 0;
}

```



## 数组指针与指针数组区别

数组指针是指向数组的指针：(\*p)[10];

指针数组是存放指针的数组：\*p[10];

对于数组指针的应用：

数组指针(\*p)[M]与 a[N][M]有联系，当把 a 的地址赋值给 p 之后，此时：p[0]的指向就是 a[0][0]的地址，但是不同的是 p 每加一次 a 就会多加一行；也就是说，p 的基类型其实是数组，指向是每行的首地址，如果要引用非首行的值，就需要\*(p[i]+j)，这就可以引用 i 行 j 列的值了；

对于指针数组的应用：

指针数组本质上是数组，不过里面存放的是指向某种类型的指针。比如：申请了 a[N][M]，则就可以申请\*p[N]，此时将 p[0]=a[0]，……p[N]=a[N]之后，就相当于 p[0]这个指针本身指向的是 a[0][0]的地址。但同样的其基类型是数组，并不是某个基本类型，因此，p[i]就指向的是 a[i]的首地址，p 每加一次 a 就会多加一行。同理，引用的时候也需要\*(p[i]+j)；

在调用函数传递的过程中，如果形参是(\*p)[M]，则实参需要为 a 或者&a[10];

如果形参是\*p[10]的话，则需要在主函数中先申请一个指针数组，将其指向对用的二维数组之后，将指针数组的数组名作为实参，例如：先将 p[i]=a[i]，之后实参就是 p 即可；

## 函数指针和指针函数的区别

指针函数是返回值为指针的函数：int \*fu(int m);

函数指针是指向函数的指针：返回值类型 (\*指针变量名)(形参列表);

函数指针：

```

例如： int fun(int x);      //声明一个函数
        int (*p)(int x);    //声明一个函数指针
        p = fun;            //将 fun 函数的首地址赋值给指针

```

# 第九章：用户自己建立数据类型

## 定义和使用结构体变量

例如：

```
struct Student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
}; //这儿的分号不能省略
```

上述结构体中，struct 说明这是一个结构体类型，其中包括 num、name、sex……等成员信息；

结构体的一般类型形式：

**struct 结构体名**  
**{成员表列};**

花括号中的是该结构体包括的子项，也称为结构体的成员；

其中结构体的成员还可以属于另外一个结构体类型：

```
struct Date
{
    int month;
    int day;
};

struct Student
{
    int num;
    char sex;
    struct Date bitrh; //成员bitrh属于Date结构体，其包含两个子项
};
```

### 1、定义结构体变量

先进行结构体的声明，随后定义；

```
struct Student
{
```

```

    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
}; // 声明

```

此时已经进行了声明结构体类型 struct Student，接下来可以用来定义变量；

**struct Student student1, student2;**

此时定义结束之后系统就为他们分配空间；

理论上分配空间为  $4+20+1+4+4+30=63$  字节，但实际在存储过程中对于一个字节往往也会分配 4 个字节，以保证存储对齐；

对于 name[20]，刚好是 4 的倍数，所以是 20 个字节；

对于 sex，会分配 4 个字节；

对于 addr[30]，为保证对齐，会分配 32 个字节；

所以总计  $4+20+4+4+4+32=68$  个字节；

## 在声明类型的同时定义变量；

```

struct Student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
} student1, student2;
这种也可以定义，方便但是不适合大型程序；

```

## 不指定类型名直接定义结构体类型变量

```

struct
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
} student1, student2;
这种不建议使用；

```

## 2、结构体变量的初始化与引用

### 1、在定义的时候可以进行定义以及赋初值;

```
struct Student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
    char addr[30];
}stu1 = {101, "wangming", 'M', 20, 80, "shanxi"};
```

### 2、可以对某一成员进行初始化

数值型默认为 0，字符型默认为'\0'，指针型默认为 NULL;

```
struct Student stu2={.name="zhang"}
```

### 3、对结构体变量成员的值的引用

引用方式为：结构体变量名.成员名;

也可以用这种方式对其赋值：student1.num = 100;

### 4、结构体中对包含的结构体的引用

```
struct Date
{
    int month;
    int year;
};
struct Student
{
    int num;
    struct Date birth;
}stu1 = {101, 9, 2001};

int main()
{
    printf("%d", stu1.birth.year);
    return 0;
}
```

若要引用 Date 种的 day，则：stu1.birth.year;

只能一级一级的找，不能一蹴而就;

## 5、结构体变量也可以进行各种运算

```
int num = stu1.num + stu2.num;  
  
stu1.num++; //优先级最高, 所以会将stu1.num先捆绑
```

## 6、同类型的结构体变量可以相互赋值

```
stu1 = stu2;
```

## 7、可以引用变量成员的地址, 也可以引用结构体变量的地址

```
scanf("%d", &stu1.num);  
printf("%o", &stu1); //输出stu1的起始地址
```

## 3、使用结构体数组

在定义的同时对数组进行初始化

```
struct Vote  
{  
    char name[10];  
    int cnt;;  
}candidate[3] = {"n" , 0 , "m" , 0 , "z" , 0};
```

定义结构之后在主函数中进行结构体数组初始化

可以整体初始化或者每个成员中的变量单独的初始化, 不能对成员整体初始化

```
struct Vote candidate[3] = {"n" , 0 , "m" , 0 , "z" , 0}; //合法  
candidate[0] = {"N1" , 0}; //非法
```

## 定义结构体数组

方法 1:

```

struct Vote
{
    char name[10];
    int cnt;;
}candidate[3];

```

方法 2:

```

struct Vote candidate[3];

```

## 结构体指针

struct Stu1 \*pt; //pt 可以指向 struct Stu1 类型的变量或数组元素;

**p = &Stu1, 同时\*p=Stu1, (\*p).num=Stu1.num;**

对于指针也可以使用箭头, 例如: p->num 就等于 Stu1.num; 但是 Stu1 不可以使用箭头;

### 1、指向结构体数组的指针

如果已经定义了 Stu[3], 那么可以在主函数中定义 **struct Stu1 \*pt;**

此时默认 p 指向的是 Stu[0]的首地址, 当 p 加 1 时, 会指向 p[1]的首地址, 依此类推;

对于指向结构体的指针, p 的增值时结构体的长度;

### 2、用结构体变量和结构体变量的指针做函数参数

结构体的值传递给另一个函数, 有 3 个方法:

1、用结构体变量的成员做参数, 此时属于值传递;

2、用结构体变量做实参, 此时传递的也是值, 但同样值传递后做出的修改并不能返回到主函数中, 所以不太使用;

3、用指向结构体变量的指针做实参, 此时将结构体变量的地址传给形参;

## 用指针处理链表

链表本质上就是通过结构体定义的一种数据结构;

## 共同体类型

union 共同体名

{

成员表列

}变量表列;

共同体所占的内存就是其所含成员的最大字节的成员的内存; 例如:

union 共同体名

{

```

char ch;
int num;
}a,b;

```

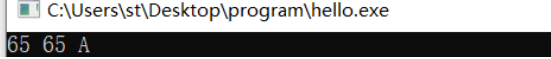
那么其中这个联合体的变量 a, b 各占 4 个字节, 其中每个单纯的变量只占 4 位;  
引用和赋值的时候和结构体类似: a.ch;  
共同体最大的不同就是在同一时刻只能存放同一个值;

```

union Student
{
    int num;
    int sorce;
    char surname;
}a;

int main()
{
    a.num = 65;
    printf("%d %d %c", a.num, a.sorce, a.surname);
    return 0;
}

```

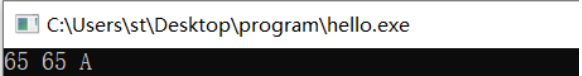


对共同体初始化的时候, 真正起作用的只有最后一个值:

```

a.num = 97;
a.sorce = 56;
a.surname = 'A';
printf("%d %d %c", a.num, a.sorce, a.surname);
return 0;

```



与结构体类似, 同类型的共同体变量是可以相互赋值的;

允许用共同体变量作为函数参数, 也可以用指针;

共同体类型可以定义在结构体的定义中, 也可以定义共用体数组, 同样的, 结构体也可以出现在共用体的定义中, 数组也可以作为共用体的成员;

## 枚举类型

如果一个变量只有几个可能的值, 那么就可以定义位枚举类型, “枚举”就是将可能的值一一的列举出来, 变量的值只限于列举出来的值的范围;

枚举类型的一般形式: **enum [枚举名] {枚举元素列表};**

例如: **enum Weekday{sum, mom, tue, wed, thu, fri, sat};**

其中, enum 说明是枚举类型, 枚举名为 Weekday, 枚举常量包括花括号中的值, 这些值被称为枚举常量;

那么枚举变量在哪儿呢:

自主定义: **enum Weekday workday, weekday;** 其中, workday 以及 weekday 被称为枚举变量, 每个枚举变量都包含了 7 个枚举常量;

**枚举常量是不能被赋值的, 但是可以引用和输出, 也可以赋值给其他变量;**

每一个枚举常量都是一个整数, 如果没有认为定义, 那么会按照在括号中的顺序从 0 开始以此排序;

例如: int a = sum; 这是可以的;

如果不想默认的话，那么可以显式的给枚举定义值：

```
enum Weekday{sum, mom=3, tue, wed, thu, fri, sat};
```

其输出的值：sum 就是 0，mom 就是 3，在 tue 就是 4，以此类推；

枚举元素可以用来比较；

## 用 typedef 声明新类型名

1、简单的用一个新的类型名代替原有的类型名：

```
typedef int Integer;    //指定用 Integer 为类型名，与 int 作用相同；
```

**int j 等价于 integer j；**

2、命名一个简单的类型名代替复杂的类型表示方法：

①命名一个新的类型名代表结构体类型：

对于原有的：

```
struct Stuent
{
    int num;
    float sorce;
};
```

如果要申请新变量：

```
struct Stuent Stu1;
```

但是如果 typedef 的话：

```
typedef struct Stuent
{
    int num;
    float sorce;
}Data;
Data Stu1;
```

其中用 Data 表示一个结构体类型名，指代的是上面的一个结构体类类型；

②命名一个新的类型名代表数组类型

```
typedef int Num[100];    //声明 Num 为整型数组类型名
```

```
Num a;                  //定义 a 为整型数组名，其含有 100 个元素
```

③命名一个新的类型名代表指针类型

```
typedef char * String;  //声明 String 为字符指针类型
```

```
String p, s[10];        //定义 p 为字符指针变量，s 为字符指针数组
```

④命名一个新的类型名代表指向函数的指针类型

```
typedef int (*Pointer)(); //声明 Pointer 为指向函数的指针类型，函数
```

返回整型值；

```
Pointer p1,p2;           //p1, p2 为 Pointer 类型的指针变量；
```

3、声明新类型的方法：

① 先按定义变量的方法写出定义体(如：int i;)。

② 将变量名换成新类型名(例如：将 i 换成 Count)。

③ 在最前面加 typedef(例如：typedef int Count)。

④ 然后可以用新类型名去定义变量。



4、#define 与 typedef 的区别：

#define 是在预编译时处理的，只是简单的字符串替换，而 typedef 是在编译阶段处理的，本身并不是简单的字符串替换，而是生成一个类型名；

5、typedef 有利于程序的移植；

## 第十章：对文件的输入输出

### C 文件的有关基本知识

在程序设计中，主要用到两种文件：

1、**程序文件**：包括源文件.c，目标文件.obj，可执行文件.exe 等，这种文件的内容是程序代码；

2、**数据文件**：文件的内容不再是程序，而是供程序运行时的读写的数据；

数据文件是研究的对象；

在实际运行过程中，经常需要将一些数据输出到磁盘上保存起来，需要的时候又从磁盘中输入到计算机内存，这就用到了**磁盘文件**；

**操作系统**为了简化用户对输入输出设备的操作，将各种设备都统一作为文件来处理；

文件一般指的是**存储在外部介质上数据的集合**；

输入输出是数据传送的过程，常将输入输出形象的称为**流**，即**数据流**。流表示了信息从源到目的端的流动；

文件是由操作系统统一管理的；

C 语言将文件看作一个字符或者字节的序列；

C 的数据文件是由一连串的字符或字节组成，并不会考虑行的界限，对文件的存取是以字符或者字节为单位的，输入输出流的开始以及结束仅受到程序控制而不受物理符号（如回车控制符）控制，这种文件也被称为**流式文件**；

**文件名的基本样式**：E:\dev\Dev-Cpp\devcpp.exe

**文件的分类**：数据根据组织形式的不同，可以分为 ASCII 文件以及二进制文件；因为数据在内存中是按照二进制存储的，如果不加转换的存储仅在外存中，就形成了二进制文件，也称为**映像文件**；如果在存储的过程中进行了 ASCII 的转换，那么就生成了 ASCII 文件，也称为**文本文件**；

**文件缓冲区**：ANSI C 采用的是“**缓冲文件系统**”处理数据，也就是说系统自动的在内存区为程序每一个正在使用文件开辟**文件缓冲区**，每一个文件不论是输入还是输出都是在同一个缓冲区；

**文件类型指针**：简称文件指针，对于相应的文件会在内存中开辟相应的文件信息区，用来存放有关文件的信息（文件名字，文件状态以及当前位置等）；

**对于文件的调用**：**FILE fp**，此时是申请了一个 FILE 结构体类型的变量，但是实际运行中，一般都是直接调用 FILE \*fp，直接用指针指向文件，通过该指针就可以直接找到与它相关联的文件；

（指向文件的指针变量并不是指在数据文件的开头，而是指在了内存中存放文件信息区的开头）

# 与文件有关函数

## 1、fopen（文件名， 使用文件方式）

```
例如： FILE *fp;  
        fp = fopen("a1", "r");
```

使用文件方式		
文件使用方式	含 义	如果指定的文件不存在
r(只读)	为了输入数据,打开一个已存在的文本文件	出错
w(只写)	为了输出数据,打开一个文本文件	建立新文件
a(追加)	向文本文件尾添加数据	出错
rb(只读)	为了输入数据,打开一个二进制文件	出错
wb(只写)	为了输出数据,打开一个二进制文件	建立新文件
ab(追加)	向二进制文件尾添加数据	出错
"r+"(读写)	为了读和写,打开一个文本文件	出错
"w+"(读写)	为了读和写,建立一个新的文本文件	建立新文件
"a+"(读写)	为了读和写,打开一个文本文件	出错
"rb+"(读写)	为了读和写,打开一个二进制文件	出错
"wb+"(读写)	为了读和写,建立一个新的二进制文件	建立新文件
"ab+"(读写)	为读写打开一个二进制文件	出错

## 2、fclose（文件指针）

```
关闭文件;
```

## 3、fgetc, fputc 的两种写法

在实际运行中, char ch = fgetc (fp); 这种写法是可以的;  
(注意 fgetc 与 fgets 的区别)

函数名	调用形式	功 能	返 回 值
fgetc	fgetc(fp)	从 fp 指向的文件读入一个字符	读成功,带回所读的字符,失败则返回文件结束标志 EOF(即-1)
fputc	fputc(ch,fp)	把字符 ch 写到文件指针变量 fp 所指向的文件中	输出成功,返回值就是输出的字符;输出失败,则返回 EOF(即-1)

函数名	调用形式	功 能	返 回 值
fgets	fgets(str,n,fp)	从 fp 指向的文件读入一个长度为(n-1)的字符串,存放到字符数组 str 中。	读成功,返回地址 str,失败则返回 NULL。
fputs	fputs(str,fp)	把 str 所指向的字符串写到文件指针变量 fp 所指向的文件中	输出成功,返回 0;否则返回非 0 值

为什么 fgets 只会读取 n-1 个字符呢? 因为它读取之后是要存放进字符串的,而字符串必须在最后一位留出一个空间取存放'\0', 所以, 如果要读取 m 个字符的话, 那么就要输入 m+1 作为传送长度; fgets 在读到 n-1 个字符之前如果遇到了'\n'或者文件结束符 EOF, 读入即结束;

## 4、feof 函数

可以监测文件尾标志是否已经被读取过, 当文件读取结束时, feof 函数值为真(1 表示), 否则为假(0 表示), 当然这只是返回值, 监测标志是还是要判断文件标识符本身是否与 feof 的值(-1) 相同, 一个是返回值, 一个是标识值, 两者不可混淆;

## 5、用格式化的方式读写文本文件

fprintf (文件指针, 格式字符串, 输出表列);

fscanf (文件指针, 格式字符串, 输入表列);

例如: fprintf ( fp, "%d, %f", i, f); //作用是将 i 和 f 按照相应的格式存到文件中

fscanf ( fp, "%d, %d", &i, &f); //作用是将文件的数据按照相应的格式读进 i 和 f 中

这种读写方式不方便, 在实践保存输入的时候要将文件中的 ASCII 码值转换为二进制形式才能保存进内存变量中, 耗费时间, 因此不太使用;

## 6、用二进制方式向文件读写数据

(这里使能使用 wb 或 rb 或 ab 方式打开文件, 否则会出现问题)

fread (buf, size, count, fp);

fwrite (buf, size, count, fp);

buf 是一个地址, 在 fread 中负责存放从文件中读取的数据, 对于 fwrite 来说, 它负责将自身中的字符传送到文件中;

size: 读写的每个数据项的长度;

count: 要读写的数据项的数量;

fp: 文件指针;

6、区分概念

①数据的存储方式: 分为文本方式(以 ASCII 码值存放进文件)与二进制方式(以二进制方式原封不动的存放);

②文件的分类: 分为文本文件(文件中全部为 ASCII 码值)以及二进制文件(按二进制方式把内存中的数据复制进文件, 也称为映像文件);

③文件的打开方式: 根据有没有 b 分为文本方式与二进制方式;

④文件读写函数：分为文本读写函数（fgetc、fgets、fputc、fputs、fscanf、fprintf 等等）与二进制读写函数（getw、putw、fread、fwrite）；

## 随机读写数据文件

读写数据方式由顺序读写与随机读写，顺序读写就是按照顺序进行读写，而随机读写并不是随机的指向数据，而是可以人为的指向自己希望获得的数据；

对于流式文件既可以顺序读写，也可以随机读写；

### 1、rewind 函数

可以是文件位置标记指向文件开头，此时如果已经读到了文件末尾的话，会默认的将 feof 函数重新置为 0（假）；

### 2、fseek 函数

一般形式：fseek（文件指针，位移量，起始点）；

C 标准指定的名字

起 始 点	名 字	用数字代表
文件开始位置	SEEK_SET	0
文件当前位置	SEEK_CUR	1
文件末尾位置	SEEK_END	2

fseek ( fp, 100L, 0); //表示将文件位置标记向前移动到离文件开头 100 个字节处

fseek ( fp, -50L, 1); //表示将文件位置标记向后移动到离当前位置 50 个字节处

### 3、ftell 函数

该函数可以得到流式文件中文件位置标记的当前位置；

i = ftell (fp);

有了这些位置的定义之后就可以实现输出文件中指定的信息了；

## 文件读写的出错检测

### 1、ferror 函数

ferror (fp);

在调用各种输入输出函数的时候（fputs、fgets、fread 等等）都可以使用该函数检测/如果该函数的返回值为 0 则表示未出错，否则表示出错，在执行 fopen 的时候，该函数的初始值会自动置为 0，以待检测 open 是否成功；

## 2、clearerr 函数

如果出现了错误的话，那么可以使用 clearerr 函数将文件出错标志和文件结束标志置为 0；

因为在 ferror 函数一旦检测到了错误，那么该标志就会移植保留，直到对同一文件电泳 clearerr 函数或者 rewind 函数；

## 3、feof 函数

feof(fp)为真的条件是：读完最后一个字符后在读一次文件，即要读到文件尾标识符 (EOF) ,此时才会为真

while(!feof(fp))

等价于：while(ch!=EOF)或者 while(ch!=-1)

## 文件函数的应用

函数名	定义	正常返回	失败返回
fopen()	FILE *fopen(const char *filename, const char *mode)	FILE 类型的指针	NULL
fclose()	fclose(文件指针)	0	非 0
fread()	unsigned int fread(void *buffer, unsigned int size, unsigned int count, FILE *fp);	函数返回的是实际读到的数据块个数	
fwrite()	unsigned int fwrite (const void *buffer, unsigned int size, unsigned int count, FILE *fp);	函数返回的是实际写入的数据块个数	
fscanf()	fscanf(文件指针, 格式字符, 输入列表);	读取的数据个数	EOF
fprintf()	fprintf(文件指针, 格式字符, 输出列表);	写入文件的字节数	负数
fgets()	char *fgets (char *s, int n, FILE *fp)	指针 s 的值	NULL
fputs()	fputs(_In_z_ const char * _Str, _Inout_ FILE * _File)	非负整数	EOF
fgetc()	int fgetc (FILE *fp);	该字符	EOF 即(-1)
fputc()	int fputc (int c, FILE *fp)	该字符	EOF
fseek()	int fseek (FILE *fp, long offset, int fromwhere)	0	非 0
rewind()	void rewind (FILE *fp)	无	
ftell()	long ftell (FILE *fp)	当前位置	-1L

# C 语言第一章基本概念

1. 什么是程序：一组计算机能识别和执行的指令
2. 什么是指令：可以被计算机理解并执行的操作命令
3. 什么是软件：与计算机系统操作有关的计算机程序、规程、规则，以及可能有的文件、文档及数据
4. C 语言有 37 个关键字，34 种运算符
5. 一个程序由一个或多个源程序文件组成
6. 一个源程序文件包括三个部分：预处理指令、全局声明、函数定义
7. 函数是用来完成一定的功能而形成的一系列 C 语句的集合，是一个模块
8. 函数包括函数首部和函数体，函数体包括声明部分和执行部分（允许即无声明也无执行部分）
9. 源程序：用 C 语言编写的程序
10. 目标程序：为源程序经编译可直接被计算机运行的机器码集合，在计算机文件上以.obj 作扩展名
11. 可执行程序：将所有编译后得到的目标模块连接装配起来，在与函数库相连接成为一个整体，生成一个可供计算机执行的目标程序，成为可执行程序
12. 程序编辑：上机输入或者编辑源程序。
13. 程序编译：先用 C 提供的“预处理器”，对程序中的预处理指令进行编译预处理对源程序进行语法检查，判断是否有语法错误，直到没有语法错误未知，编译程序自动把源程序转换为二进制形式的目标程序
14. 程序连接：将所有编译后得到的目标模块连接装配起来，在与函数库相连接成为一个整体的过程称之为程序连接
15. 程序：一组计算机能识别和执行的指令，运行于电子计算机上，满足人们某种需求的信息化工具
16. 程序模块：可由汇编程序、编译程序、装入程序或翻译程序作为一个整体来处理的一级独立的、可识别的程序指令
17. 程序文件：程序的文件称为程序文件，程序文件存储的是程序，包括源程序和可执行程序
18. 函数：将一段经常需要使用的代码封装起来，在需要使用时可以直接调用，来完成一定功
19. 主函数：又称 main 函数，是程序执行的起点
20. 被调用函数：由一个函数调用另一个函数，则称第二个函数为被调用函数
21. 库函数：一般是指编译器提供的可在 c 源程序中调用的函数。可分为两类，一类是 c 语言标准规定的库函数，一类是编译器特定的库函数
22. 程序调试：是将编制的程序投入实际运行前，用手工或编译程序等方法进行测试，修正语法错误和逻辑错误的过程
23. 程序测试：是指对一个完成了全部或部分功能、模块的计算机程序在正式使用前的检测，以确保该程序能按预定的方式正确地运行
24. 编写代码时尽可能的保持用函数写，对于共用的变量要多加小心，如果时共用变量 i 判断 while 循环的时候，一定要多关注 i 的初始值，i 的修正值。对于数组或者变量的初始化要多加判断，时候要初始化为 0，时候要为 1，要多小心

## C 语言第二章基本概念

1. 算法：要求计算机进行操作的步骤
2. 数据（结构）：在程序中指定要用到哪些数据，以及这些数据的类型和数据的组织形式
3. 算法的特性：有穷性，确定性，有效性，有零个或多个输入，有一个或多个输出
4. 表示算法的方式：自然语言、传统流程图、结构化流程图（N-S 流程图）、伪代码等
5. 三种基本结构：顺序结构、循环结构（当型循环结构，直到型循环结构）、选择结构
6. 三中基本结构共同特点：只有一个出口，只有一个入口，每一部分都有意义，不存在死循环
7. 结构化程序设计：自顶向下，逐步细化，模块化设计，结构化编码
8. 结构化算法：由一些顺序、选择、循环等基本结构按照顺序组成，流程的转移只存在于一个基本的范围之内
9. 结构化算法便于编写，可读性高，修改和维护起来简单，可以减少程序出错的机会，提高了程序的可靠性，保证了程序的质量，因此提倡结构化的算法
10. 顺序结构：顺序结构是一种线性、有序的结构，它依次执行各语句模块
11. 选择结构：选择结构是根据条件成立与否选择程序执行的通路。
12. 循环结构：循环结构是重复执行一个或几个模块，直到满足某一条件位置

## C 语言第三章基本概念

1. 计算机中的浮点型数据都是按照双精度存储的
2. 数据有两种表现形式：常量和变量  
常量：整型常量、实型常量、字符常量（普通字符、转义字符）、字符串常量、符号常量（注意符号常量与变量的区别，符号常量不占用内存，预编译之后这个符号会全部用其对应的值替换）  
变量  
常变量  
标识符
3. 数据类型：基本类型（整形，浮点）、枚举类型（enum）、空类型（void）、派生类型（指针、数组、结构体、共用体、函数）
4. 只有整形数据或字符型可以添加 signed 或 unsigned 修饰符，输出输入无符号整形的时候用 %u
5. 在存储浮点型时，数值以规范化的二进制数指数形式存放在存储单元中，此时系统会将实型数据分成小数部分和指数部分两个部分
6. 有限的存储单元不可能完全精确的存储一个实数，所以，如果要判断浮点型数据是否为 0 的时候，不能使其与 0 相比较，必须限定一个无限趋近于 0 的界限，例如十的负六次方之类
7. 只有 % 要求参与运算的对象是整数，结果也是整数，其他运算符可以是任何算求类型
8. sizeof 是关键字，也是运算符
9. 优先级：非算关，与或条赋逗
10. 关系运算符中，> ≥ < ≤ 的优先级高于 == 和 !=

11. 条件运算符、赋值运算符、单目运算符是从右往左的结合方向，其余都是从左往右
12. 函数的声明部分不是语句
13. 表达式：用 C 语言运算符将运算对象连接起来的式子，就叫表达式
14. 语句：语句用于向计算机系统发出操作指令
15. C 语句分为 5 类
16. 控制语句：用于完成一定的控制功能
17. 函数调用语句：用于调用函数，由函数调用与分号构成
18. 表达式语句：由一个表达式加一个分号构成
19. 空语句：只有一个分号的语句，可以用来作为流程的转向点，也可以用来作为循环语句中的循环体
20. 复合语句：用 {} 括起来的多个语句
21. 赋值语句：将一个数据赋值给一个变量
22.  $x*=2+3 \Leftrightarrow x=x*(2+3)$
23. 赋值表达式末尾没有分号，赋值语句末尾必须有分号。一个表达式中可以包含一个或多个赋值表达式，但是绝对不能包含赋值语句
24. 定义非静态变量的时候如果不给赋初值，那该变量的值是任意值，定义静态变量的时候如果不给赋初值，那该变量的值默认是 0
25. 从计算机向输出设备输出数据称为输出，从输入设备向计算机输入数据称为输入

## C 语言第四章基本概念

1. 用 scanf 读取双精度的时候一定要用 %lf，否则会出错
2. else 不能单独使用，它是 if 语句的一部分，总是与它上面尚未配对的最近的 if 配对
3. 逻辑表达式：用逻辑运算符将关系表达式或其他逻辑量连接起来的式子
4. 关系表达式：用关系运算符将两个数值或数值表达式连接起来的式子（关系表达式的值是一个逻辑值，即“真”或“假”，在 C 的逻辑运算中，以“1”代表真，以“0”代表假，但是在判断一个量是否为“真”的时候，以 0 代表“假”，非 0 代表真）
5. 逻辑运算中，如果从左往右的判断中，前面已经判断为假了，则不会再执行后面的判断语句了
6. 算术运算即“四则运算”，是加法、减法、乘法、除法、乘方、开方等几种运算的统称。其中加减为一级运算，乘除为二级运算，乘方、开方为三级运算。在一道算式中，如果有多级运算存在，则应先进行高级运算，再进行低一级的运算。C 语言中的算术运算符包括：+、-、\*、/、++、--、% 等种类。如果只存在同级运算；则从左至右的顺序进行；如果算式中有括号，则应先算括号里边，再按上述规则进行计算。
7. 关系的基本运算有两类：一类是传统的集合运算（并、差、交等），另一类是专门的关系运算（选择、投影、连接、除法、外连接等），而在 C 语言中，关系运算通常被认为是比较运算，将两个数值进行比较，判断比较结果是否符合给定的条件。常见的关系运算符包括：<、<=、>、>=、==、!= 等种类。其中，前 4 种关系运算符(<、<=、>、>=)的优先级别相同，后 2 种(==、!=)也相同。而前 4 种高于后 2 种。例如，> 优先于 ==。而 > 与 < 优先级相同。并且，关系运算符的优先级低于算术运算符，关系运算符的优先级高于赋值运算符
8. 在逻辑代数中，有与、或、非三种基本逻辑运算。表示逻辑运算的方法有多种，如语句描述、逻辑代数式、真值表、卡诺图等。而在 C 语言中，逻辑运算通常用于使用逻



辑运算符将关系表达式或其它逻辑量连接起来组成逻辑表达式用来测试真假值。常见的逻辑运算符包括：&&、||、! 等种类

9. 在 C 语言中逻辑常量只有两个，即 0 和 1，用来表示两个对立的逻辑状态，其中 0 表示假，1 表示真。逻辑变量与普通代数一样，也可以用字母、符号、数字及其组合成为的逻辑表达式表示。对于系统来说，判断一个逻辑量的值时，系统会以 0 作为假，以非 0 作为真。例如 3&&5 的值为真，系统给出 3&&5 的值为 1

## C 语言第五章基本概念

1. while 循环中要多检查终止条件是否添加
2. for 循环的顺序：先执行表达式 1，然后判断表达式 2，判断为真则执行循环体中的语句（为假则退出循环），循环体语句执行完毕之后，进行表达式 3 的执行，随后继续判断表达式 2...以此循环，直到不满足表达式 2
3. for 语句循环中的表达式可以是逗号表达式，逗号表达式中的每个语句都会执行，但是最后一个语句的值是循环体表达式的实际值
4. break 可以使流程跳出 switch 结构，继续执行 switch 语句下面的一个语句
5. break 可以用来从循环体内跳出当前所在的循环体，即提前结束循环，执行循环下面的语句
6. continue 语句提前结束本次循环，接着执行下次循环
7. 如果使用循环的话尽量使用 for 循环

## C 语言第六章基本概念

1. 数组是一组有序数据的集合，其中每一个元素都属于同一个数据类型
2. 用 %d 接收 0，转化成 %c 就相当于 '0'，要是用 %c 接收 0，转化成 %d 就相当于 48
3. 字符数组的值只能用 strcpy 之类的去赋值，不能直接用赋值语句赋值
4. strlwr 转换为小写
- 5.strupr 转换为大写
6. strncpy(字符数组，字符串，n)

## C 语言第七章基本概念

1. 从本质意义上来说，函数是用来完成一定功能的，函数就是功能，每一个函数用来实现特定的功能
2. 一个 C 程序由一个或多个程序模块组成，每一个程序模块作为一个源程序文件。
3. 一个源程序文件由一个或多个函数以及其他有关内容组成
4. 一个源程序文件就是一个编译单位，程序编译时是以源程序文件为单位进行编译的
5. C 语言的执行以及结束都在 main 函数中

6. 函数从用户使用角度来分有两种
7. 库函数：由系统提供，用户不需要自主定义，可以直接使用它们。不同的编译器的库函数的数量和功能会有所不同
8. 自定义函数：用户自己定义的函数，用来解决用户专门需求的函数
9. 函数从函数形式来分：无参函数以及有参函数
10. 形参：在定义函数时函数名后面括号中的变量名
11. 实参：在主调函数中调用一个函数时，函数名后面括号中的参数
12. 实参与形参的类型应相同或赋值兼容
13. 函数的返回值是通过函数中的 return 语句获得的
14. 函数的类型在定义函数时就需要指定，return 语句中的表达式类型需要与该类型相同，如果不同，则由函数类型决定
15. 函数的定义是指对函数功能的确立，包括指定函数名，函数值类型，形参及其类型以及函数体等，它是一个完整的，独立的高数单位
16. 函数的声明的作用是把函数的命题，函数类型以及形参的类型，个数和顺序通知编译系统，便于在调用该函数的时候系统按此进行对照检查，声明中不包括函数体
17. 递归调用：在调用一个函数的过程中又出现直接或间接地调用该函数本身
18. 递归调用需要的是大局观，不能拘泥于细节把握，要高屋建瓴，第 n 步到第 n-1 步，第 n-1 到第 n-2 步…第 2 步到第 1 步的操作都相同，第 1 步的时候结束，只要把 n 到 n-1 的步骤写出来，后面再将 n-1 的值传递过去，将 n-1 作为整体
19. 变量的声明与定义：建立存储空间的声明称为定义，不需要建立储存空间的声明称为声明，简言之，在函数中出现的对变量的声明（除了用 extern 声明的以外）都是定义，在函数中对其他函数的声明不是函数的定义
20. 根据函数能否被其他源文件调用，将函数区分为内部函数和外部函数

## C 语言第八章基本概念

1. 地址指向该变量单元，因此将地址形象化的称为“指针”（地址就是指针，指针就是地址）
2. 直接访问：直接对变量名进行访问
3. 间接访问：将变量 i 的地址存放在另一个变量中，然后通过该变量来找到变量 i 的地址，进而访问 i 变量本身
4. 地址本身就是指针，而存放某个变量的地址的变量即是指针变量，指针变量的值就是地址（也就是指针）
5. 在定义指针变量时必须指定基类型
6. 一个变量的指针包含两个含义，一个是以存储单元编号表示的纯地址，另一个是它指向的存储单元的数据类型
7. 数组元素的指针就是数组元素的地址
8. 引用数组元素可以用下标法，也可以用指针法
9. 数组名代表的不是整个数组，而是数组首元素的地址
10. 两个地址不能相加，那是无意义的
11. 指针指向数组的时候，注意不能忘记当前的指向，如果指到了数组尾，下面还需要从头开始的话，一定要指回去
12. 实参数组名代表的是一个固定的地址，可以说是一个常量，改变不了，但是形参的数

组名并不是固定的地址，而是一个指针变量

13. C 语言不管是变量名作为传参还是地址作为传参，本质上都是“值传递”
14. 不论是一维数组还是二维数组， $*(a+i) = a[i]$
15. 指针与二维数组的关系的本质在于一定要搞清楚指向的基类型是什么
16. 指针数组与数组指针的区别在于看指针最先和谁接触，那么本质就属于谁，对于  $(*p)[5]$ ，本质就是指针，指向的是数组类型的元素
17. 引用字符串的时候，可以用字符数组存放一个字符串，也可以直接通过字符指针变量输出一个字符串常量
18. 字符指针变量输出字符串常量的格式

```
char *string="I love";
```

实质上是在内存中开辟了一个字符数组存放字符串常量，但是字符数组没有名字，不能通过数组名去引用，只能通过指针变量去引用，在初始化的时候，实际上就是把字符串第一个元素的地址赋给指针变量 string，此时 string 指向的是字符串中的第一个字符

19. 需要注意，C 语言中只有字符变量，没有字符串变量

20. 

```
char *string="I love";
```

等价于 

```
char *string;
```

```
string="I love";
```

不等于 

```
char *string;
```

```
*string="I love";
```

注意上面的区别

21. 字符指针变量与字符数组的本质区别：字符数组是数组，字符指针变量是指针，存放的是地址，指向的是字符串首地址
22. 字符数组中各元素的值可以改变，但是字符指针变量指向的字符串常量的值不可以改变（毕竟是常量，常量是固定的不变的）
23. 指向函数的指针  

```
int (*p) (int, int)
```

```
p=max;
```

此时会流转到 max 函数的入口地址  
用指向函数的指针做函数参数，重点例题
24. 多重指针
25. void\*变量并不是指向任意类型的数据，而是指向空类型，也可以说不指向确定的类型
26. 指针变量的类型以及含义

```
int i;
```

```
int *p;
```

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

```
int *p[4];
```

```
int (*p)[4];
```

```
int f();
```

```
int *p();
```

```
int (*p)();
```

```
int **p;
```

```
void *p;
```

## C 语言第九章基本概念

1. 用户以及建立由不同类型数据组成的组合型的数据结构，就称为结构体
2. 结构体声明的时候一定要在后面加上“;”
3. scanf 向结构体变量输入的时候，对于数组类型的变量不需要添加取地址符，但是对于其他普通类型必须取地址
4.  $\text{stu.nun}$   
 $\Leftrightarrow (*p).\text{num}$   
 $\Leftrightarrow p \rightarrow \text{num}$
5. 共用体的例题
6. 枚举类型的例题
7. typedef

## C 语言第十章基本概念

1. 缓冲文件系统中，关键的概念是“文件类型指针”，简称“文件指针”。每个被使用的文件都在内存中开辟一个相应的文件信息区，用来存放文件的有关信息(如文件的名称、文件状态及文件当前位置等)。这些信息是保存在一个结构体变量中的。该结构体类型是由系统声明的，取名为 FILE。
2. 通过文件指针访问文件的好处是：可以随机访问文件，有效表示数据结构，动态分配内存，方便使用字符串，有效使用数组。
3. 对文件的打开是指为文件建立相应的信息区(用来存放有关文件的信息)和文件缓冲区(用来暂时存放输入输出的数据)。“关闭”是指撤销文件信息区和文件缓冲区，使文件指针变量不再指向该文件，显然就无法进行对文件的读写了。