

# 从零开始学习**C**语言

本书的主要特点如下：

内容全面详细。本章一共分为**18**章，涵盖了**C**语言中的所有知识，并将**C**语言各个知识点做一个章节进行讲解。并举出大量实例。

结构清晰明了。本章**18**章中，每章都分若干个小节，每个小节一个小知识点。结构层次清晰可见。

讲解由浅入深。向读者介绍**C**语言的基本理论知识、数据结构和基本的编程规则，让读者对**C**语言的基本知识以及结构化程序设计思想有一个初步的认识；接着对**C**语言一些复杂的数结构类型如数组、函数、指针操作、结构体与共用体、文件等进行详细的讲解。

实例丰富多样。本书所讲的每一个知识点都运用充分的实例进行讲解说明，便于读者掌握。

# 第1章 C语言入门基础

C语言作为国际上流行的计算机高级语言，能实现多种功能。为使读者能够对C语言有一个全面的认识，本章在介绍C语言之前，还简单的介绍了很多其他的相关知识。

计算机语言的演变；

数制、数制转换与存储；

程序设计思想—算法；

C语言的发展简史和特点；

认识C语言程序；

**Turbo C V2.0**的运行环境及基本操作。

## 1.1 计算机语言的演变

- 机器语言
- 汇编语言到
- 高级语言
- 面向对象或面向问题的高级语言

# 1.1.1 机器语言

机器语言是第一代计算机语言。计算机所使用的是由“0”和“1”组成的二进制数，二进制是计算机的语言的基础，所以也称为二进制语言。机器语言指用机器码书写程序，不易被人们识别和读写，所以使用机器语言是十分痛苦的，特别是在程序有错需要修改时，更是如此。而且，由于每台计算机的指令系统往往各不相同，所以在一台计算机上执行的程序，要想在另一台计算机上执行，必须另编程序，造成了重复工作。但由于计算机能够直接识别程序中的指令，故而运算效率是所有语言中最高的，这种用二进制编写的程序也叫“目标程序”。

## 1.1.2 汇编语言

汇编语言又称符号语言，对机器指令进行简单的符号化，它也是利用计算机所有硬件特性并能直接控制硬件语言。人们为了减轻使用机器语言编程的痛苦，对机器语言进行了一种有益的改进：用一些简洁的英文字母、符号串来替代一个特定的指令的二进制串，比如，用“**ADD**”表示加法，“**MOV**”表示数据传递等等，因此，人们就能理解程序所进行的操作，方便用户对程序进行纠错及维护。

### 1.1.3 高级语言

用高级语言编写的程序称为“源程序”，源程序不能在计算机上直接运行，必须将其翻译成二进制程序后才能执行。翻译有两种方式：解释程序和编译程序。解释程序是将一次只读一行源程序，并执行该行语言指定的操作，每次运行用户程序时，必须要用解释程序。在程序的开发过程中，运用解释的方式执行程序，便于程序员对程序进行调试。编译程序是将源程序全部翻译成目标代码即二进制程序后再执行，只读取一次，节省了大量的时间。

## 1.1.4 面向对象或面向问题的高级语言

第四代语言是使用第二代第三代语言编制而成的。面向对象的语言是在面向过程的计算机语言的基础上发展而来的，如C++语言就是由C语言发展而来的。所谓面向对象，就是基于对象的概念，以对象为中心，类和继承为构造机制，认识了解刻画客观世界以及开发出相应的软件系统。它是把构成问题事务分解成各个对象，建立对象的目的不是为了完成一个步骤，而是为了描述某个事物在整个解决问题的步骤中的行为。比较典型代表的面向对象语言有C++、Virtual Basic、Delphi等。



## 1.2 数制、数制转换与存储

- 数制
- 数制转换
- 计算机中数据的存储

## 1.2.1 数制

### 1. 二进制数

二进制数由两个基本数字0、1组成，二进制数的运算规律是逢二进一。

例如：

100101可以写成 $(100101)_2$ 或写成100101B。

二进制数的加法和乘法运算如下：

$0+0=0$     $0+1=1+0=1$     $1+1=10$     $0*0=0$     $0*1=1*0=0$

$1 \times 1 = 1$

$$\begin{array}{r} 1000110 \\ + 0010101 \\ \hline 1011011 \end{array}$$

## 1.2.1 数制

### 2. 八进制数

八进制是由0~7八个数字组成，运算规则是逢8进一。

例如：

八进制261写成  $(261)_8$ 、 $(261)_0$ 。

## 1.2.1 数制

### 3. 十进制数

十进制数是我们常用的数据表示方法，由0~9十个数字组成，运算规则是逢10进一。

例如：

十进制126可表示为  $(126)_{10}$ 、126D、126。

## 1.2.1 数制

### 4. 十六进制数

十六进制数由0~9以及A~F十六个数字组成，A~F分别表示十进制数10~15，运算规则是逢16进一。通常在表示进用例如：

(1FA)<sub>16</sub>、(1FA)<sub>H</sub>

# 注意

在C语言程序中

十六进制需要以0x开头

八进制需要以0开头，

例如：

0123表示八进制的123

0x123表示十六进制的123

## 1.2.2 数制的转换

### 1. 二进制、八进制、十六进制转换成十进制

规则：数码乘以各自的权的累加

【例1-1】其他进制转换成十进制。

$$(10001)_B = 2^4 + 2^0 = 16 + 1 = 17$$

$$(101.01)_B = 2^2 + 2^0 + 2^{-2} = 4 + 1 + 0.25 = 5.25$$

$$(011)_O = 8^1 + 8^0 = 8 + 1 = 9$$

$$(72)_O = 7 * 8^1 + 2 * 8^0 = 7 * 8 + 2 * 1 = 58$$

$$(112A)_H = 1 * 16^3 + 1 * 16^2 + 2 * 16^1 + 10 * 16^0 = 4394$$

## 1.2.2 数制的转换

### 2. 十进制转换成二进制、八进制、十六进制

规则：

整数部分：除以进制取余数，直到商为0，余数从下到上排列。

小数部分：乘以进制取整数，得到的整数从上到下排列。



## 【例1-2】十进制转换成其他进制。

### (1) 十进制20.345转换成二进制

整数部分：

$$20/2=10 \quad \text{----余}0$$

$$10/2=5 \quad \text{----余}0$$

$$5/2=2 \quad \text{----余}1$$

$$2/2=1 \quad \text{----余}0$$

$$1/2=0 \quad \text{----余}1$$



小数部分：

$$0.345*2=0.69 \quad \text{----取整数}0$$

$$0.69*2=1.38 \quad \text{----取整数}1$$

$$0.38*2=0.76 \quad \text{----取整数}0$$

$$0.76*2=1.52 \quad \text{----取整数}1$$

$$0.52*2=1.04 \quad \text{----取整数}1$$



$$20.345D=10100.01011B$$

## 【例1-2】十进制转换成其他进制。

(2) 十进制100转换成八进制、十六进制

$$100/8=12 \text{ ----余}4$$

$$12/8=1 \text{ ----余}4$$

$$1/8=0 \text{ ----余}1$$

$$100D=144O$$

$$100/16=6 \text{ ----余}4$$

$$6/16=0 \text{ ----余}6$$

$$100D=64H$$

## 1.2.2 数制的转换

### 3. 二进制转换八进制

规则：

整数部分：从右向左按三位进行分组，不足补零。

小数部分：从左向右按三位进行分组，不足补零。

【例1-3】将二进制数  $(1101101110.110101)_2$  转换成八进制数。

$$\begin{array}{cccccc} \underline{001} & \underline{101} & \underline{101} & \underline{110} & \underline{110} & \underline{101} \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 5 & 5 & 6 & 6 & 5 \end{array}$$

$$(1101101110.110101)_2 = (1555.65)_8$$

## 1.2.2 数制的转换

### 4. 二进制转换成十六进制

规则：

- 整数部分：从右向左按四位进行分组，不足补零。
- 小数部分：从左向右按四位进行分组，不足补零。

【例1-4】将二进制数  $(001101101110.110101)_2$  转换成十六进制数。

$$\begin{array}{ccccccc} & \underline{0011} & \underline{0110} & \underline{1110} & \underline{.1101} & \underline{0100} & \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\ & 3 & 6 & E & D & 4 & \end{array}$$

$$(001101101110.110101)_2 = (36.ED4)_{16}$$

## 1.2.2 数制的转换

### 5. 八进制、十六进制转换成二进制

规则：

- 一位八进制对应三位二进制。
- 一位十六进制对应四位二进制。

**【例1-5】** 八进制、十六进制转换成二进制。

$$(136)_8 = (\underbrace{001}_{\downarrow 1} \underbrace{011}_{\downarrow 3} \underbrace{110}_{\downarrow 6})_2$$

$$(17A)_{16} = (\underbrace{0001}_{\downarrow 1} \underbrace{0111}_{\downarrow 7} \underbrace{1010}_{\downarrow A})_2$$

## 1.2.3 计算机中数据的存储

如表示：比较一下1与-1的原码、反码和补码。

表·数据存储表

整数	原码	反码	补码
1	0000 0001	0000 0001	0000 0001
-1	1000 0001	1111 1110	1111 1111

## 1.3 程序设计思想—算法

在我们遇到问题的时候，我们首先在大脑中形成一种解题思路，然后再根据可行的思路运用具体的步骤解决问题。在程序设计中，也需要有一种编程思路，这就是算法。

## 1.3.1 算法的概念

在程序设计中，算法应该能够离散成具体的若干个操作步骤，而且每一个步骤都是能够用程序设计语言提供的语句或者语句串来完成的。

例如，求二个整数中的最大的数。解决这个问题的算法如下：

第1步 开始。

第2步 输入二个整数 $a$ 、 $b$ 。

第3步 比较 $a$ 、 $b$ 的大小，如果 $a > b$ 时，输出 $a$ ，否则输出 $b$ 。

第4步 结束。

需要注意的是，程序是有开始和结束的，所以算法必须有“开始”和“结束”这两个步骤。



## 1.3.2 算法的特点

算法具有以下五个重要的特征：

1. 有穷性
2. 确定性
3. 有效性
4. 输入
5. 输出

### 1.3.3 算法的表示方法

#### 1. 自然语言




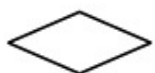

【例1-6】用自然语言描述100以内正整数的和。

1. 设S代表总和，N代表正整数；
2.  $S=0$ ， $N=1$ ；
3.  $S=S+N$ ，原来的和加上一个正整数；
4.  $N=N+1$ ，把下一个正整数赋给N；
5. 判断N是否小于100，如果是跳转到步骤3，否则跳转到步骤6；
6. 输出S的值。

# 1.3.3 算法的表示方法

## 2. 流程图

### 常用流程图构件

符号	名称	用途
	开始、结束符	用来表示程序的开始和结束。一个算法只能有一个开始处，但可以有多个结束处
	输入、输出框	表示数据的输入和计算数据的输出
	处理框	表示需要处理的内容，只有一个入口和一个出口
	判断框	用来表示分支情况，菱形框的四个顶点中，通常用上方的顶点表示入口，视需要用其余两个顶点来表示出口
	流程线	指出流程控制的方向，即动作的次序

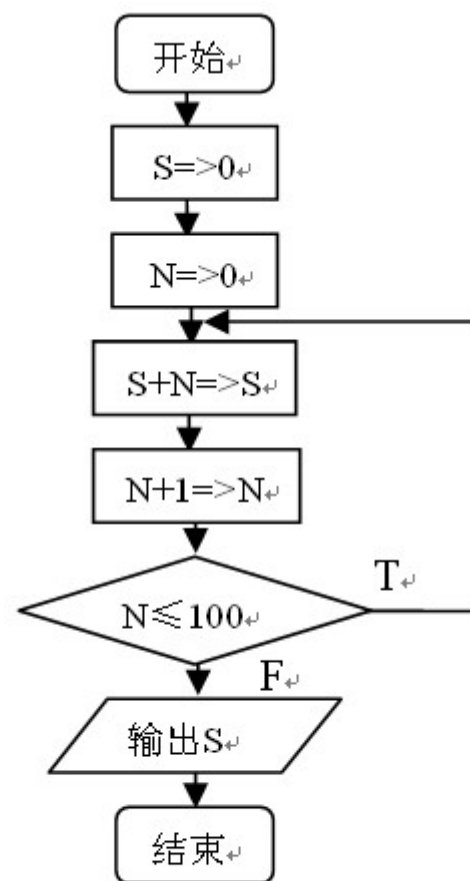


图 1-1 100 以内正整数的流程图表示

## 1.3.3 算法的表示方法

### 3. 伪代码

将上述【例1-6】用伪代码描述表示如下：

①  $N \leftarrow 1;$

②  $S \leftarrow 0;$

③ do while  $N \leq 100$

④ {  $S \leftarrow S + N;$

$N \leftarrow N + 1;$

}

⑤ print S

# 1.3.3 算法的表示方法

## 4. N-S图

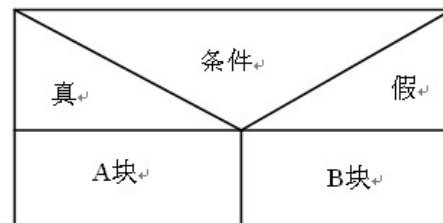
(1) 顺序结构N-S图

(2) 选择结构N-S图

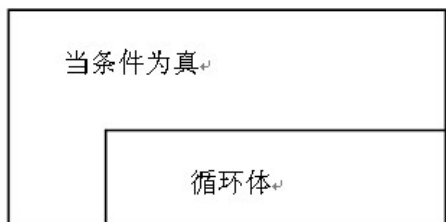
(3) 循环结构N-S图



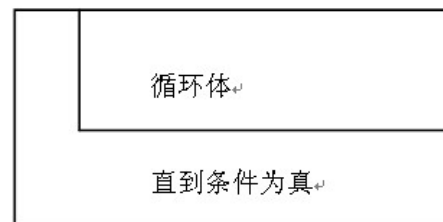
顺序结构



选择结构



当型循环



直到型循环

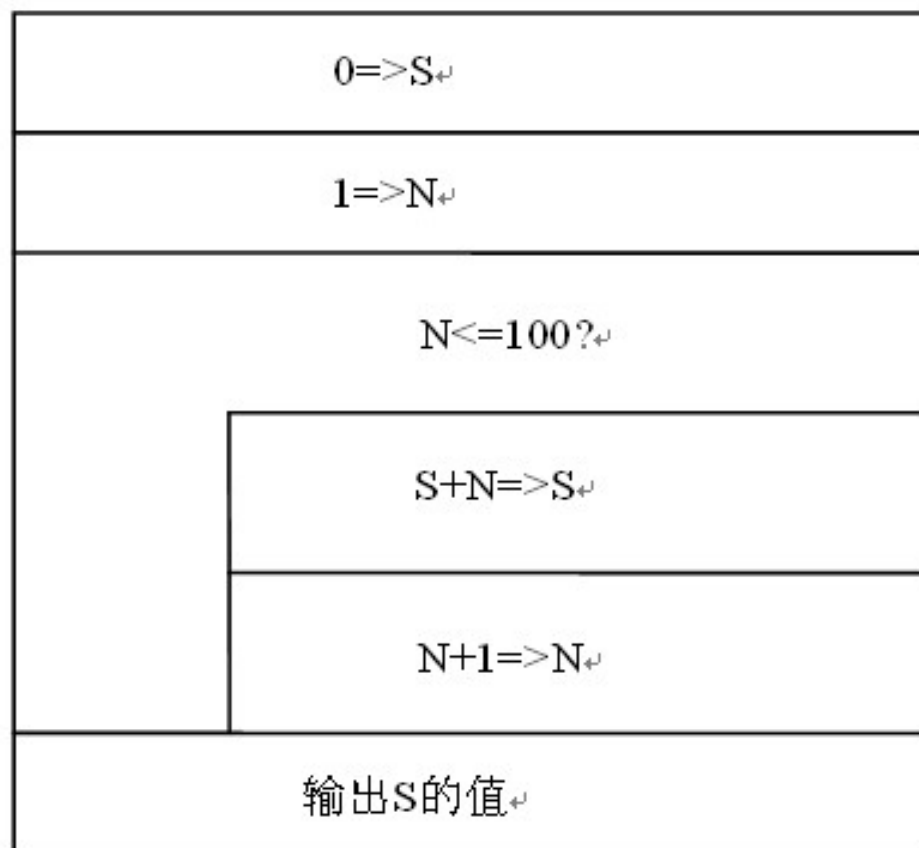


图 1-3 100 以内正整数的 N-S 图表示

## 1.3.4 算法分析

什么样的算法才是一个好的算法呢？通常从下列几个方面衡量算法的优劣：

1. 正确性
2. 可读性
3. 健壮性
4. 时间复杂度与空间复杂度

$$T(n)=O(f(n))$$

$$S(n)=O(f(n))$$



## 1.4 C语言的发展简史和特点

C语言能够快速发展成为最受欢迎的语言之一，主要是因为它具有强大的功能。它既有高级语言的特点，又具有汇编语言的特点，它可以作为工作系统设计语言，编写系统应用程序，也可以作为应用程序设计语言，编写不依赖计算机硬件的应用程序。

## 1.4.1 C语言的诞生与发展

1972年，贝尔实验室D. M. Ritchie设计出C语言，当时Ken Thompson刚刚使用汇编语言和B语言开发出UNIX操作系统，但用汇编语言开发系统非常烦琐，于是D. M. Ritchie用C语言改写UNIX系统的内核。

1977年Dennis M. Ritchie 发表了不依赖于具体机器系统的C语言编译文本《可移植的C语言编译程序》。

1978年由美国电话电报公司(AT&T) 贝尔实验室正式发布C语言。同由B. W. Kernighan和D. M. Ritchie共同完成的了著名的《The C Programming Language》一书，通常简称为《K&R》

1983年，美国国家标准化协会（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ANSI C 标准。

1990年，国际标准化组织ISO（International Organization for Standards）接受了89 ANSI C 为ISO C 的标准（ISO9899-1990）。

目前c语言在世界范围内都是相当流行的高级语言。C语言最初是为了描述和实现unix系统的，但随着c语言的发展，它适用任何平台，C可以用来编写应用软件，也可以用来编写系统软件。许多著名的系统软件，如DBASE IV都是由C语言编写的。用C语言加上一些汇编语言子程序，就更能显示C语言的优势了，像PC- DOS、WORDSTAR等就是用这种方法编写的。

## 1.4.2 C语言的特点

C语言具有以下几个基本特点。

1. 紧凑简洁、灵活方便
2. 运算符丰富多样
3. 数据结构多样性
4. 程序语言模块化
5. 控制语句结构化
6. 接近硬件与系统
7. 运行效率高
8. 可移植性好

## 第2章 认识C语言程序

C语言程序的结构特征；

C语言程序的书写风格；

C语言程序的开发过程；

熟悉**Visual C++ 6.0**集成开发环境；

用**Visual C++ 6.0**运行一个C程序。

## 2.1 C语言程序的结构特征

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

①文件包含

```
void main()
```

②主函数

```
{
```

③程序开始

```
int a ,b ,sum;
```

④变量定义

```
scanf( "%d", &a );    /*输入a*/
```

⑤格式输入函数与注释

```
scanf( "%d", &b );
```

⑥格式输入函数

```
sum=a+b;    /*对a、 b求和*/
```

⑦求和与注释

```
printf("sum=%d \n",sum);
```

⑧格式输出函数

```
}
```

⑨程序结束

## 2.1 C语言程序的结构特征

### 1. 文件包含

通用的格式是

**# include<文件名>或# include“文件名”**它

属于预处理命令中的一种。文件包含的作用是将该程序编译时所需要的文件复制到本文件，再对合并后的文件进行编译。**stdio.h**是基本输入输出的头文件，在上例中，我们用到输入输出函数**printf()**、**scanf()**，因此需要在源程序的开头写上**# include<stdio.h>**。

## 2.1 C语言程序的结构特征

### 2. 主函数

main()表示主函数，这是系统提供的特殊函数，每一个C语言程序有且只有一个main()函数。函数的内部用一对大括弧括起来，括起来的部分称为函数体。

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

①文件包含

```
void main()
```

②主函数

```
{
```

③程序开始

```
int  a ,b ,sum;
```

④变量定义

```
scanf( "%d", &a );
```

/\*输入a\*/

⑤格式输入函数与注释

```
scanf( "%d", &b );
```

⑥格式输入函数

```
sum=a+b;
```

/\*对a、b求和\*/

⑦求和与注释

```
printf("sum=%d \n", sum);
```

⑧格式输出函数

```
}
```

⑨程序结束

## 2.1 C语言程序的结构特征

### 3. 变量的定义

一个变量在内存中占据一定的存储单元，在该存储单元中存放变量的值。本行定义了三个变量**a**、**b**、**sum**，分别用来存储等待输入的两个整型数和他们的和，便于以后的操作。

C语言中，变量的定义必须符合标识符的命名规则，即标识符只能字母（大小可均可）、数字和下划线3种字符组成，第1个字母不能是数字。

C语言对大小写严格区分，变量一般用小写。变量遵循先定义后使用的原则，定义变量有利于系统分配存储空间，定义变量其实就是在内存中开辟存储单元。



## 2.1 C语言程序的结构特征

### 4. 格式输入与输出函数

输入函数的作用是将输入设备（如键盘）按指定的格式输入一组数据，赋到指定的变量存储单元，作为变量的值。

输出函数的作用是向系统指定的输出设备（如显示器）输出若干个任意类型的数据。

```
# include<stdio.h>           ①文件包含
void main()                   ②主函数
{                               ③程序开始
    int  a ,b ,sum;           ④变量定义
    scanf( "%d", &a );        /*输入a*/      ⑤格式输入函数与注释
    scanf( "%d", &b );        ⑥格式输入函数
    sum=a+b;                  /*对a、b求和*/ ⑦求和与注释
    printf("sum=%d \n", sum);  ⑧格式输出函数
}
```

## 2.1 C语言程序的结构特征

### 5. 注释部分

例子中的⑤⑦行“/\*”开头到“\*/”结尾之间的内容表示注释，它可以在一行书写或分多行书写，可写在程序的任何位置。

⑤⑦中的注释部分是对所要进行的操作的说明，⑤是一个输出语句，输入变量a的值，⑦是一个赋值语句，将a、b的相加的和赋给sum。

## 2.2 C语言程序的书写风格

为了增强程序的可读性，便于人们理解和查错，建议使用良好的书写格式。

```
#include <stdio.h>
void main( )
{
int k=0; char c='A'; /*定义一个整型变量，一个字符变量，并赋值*/
do {                /*直到型循环*/
    switch (c++)
    {                /* switch 多分支语句*/
        case 'A' : k++; break;
        case 'B' : k--;
        case 'C' : k+=2; break;
        case 'D' : k=k%2; break;
        case 'E' : k=k*10; break;
        default: k=k/3;
    }
    k++;
} while(c<'G');
printf("k=%d\n", k);
}
```

## 2.2 C语言程序的书写风格

C语言的书写格式，具体如下：

- 1、C语言程序使用英文小写字母书写。大写字母一般符号常量或特殊用途使用。C语言区分字母大小写，如 **student** 和 **STUDENT** 是两不同的标识符。
- 2、标识符是用于标识某个量的符号，可由程序员任意定义，但为了增加程序的可读性，命名应尽量有相应的意义，以便阅读理解以及程序员之间的交流。

## 2.2 C语言程序的书写风格

3、不使用行号，通常按语句的顺序执行。前面的例子中我们使用编号是为了讲解的方便，在正常的源程序中，不能使用。

4、所有语句都必须以分号“;”结束，作为语句之间的分隔符。

5、C程序中一个语句可以占多行，一行也可以有多个语句，但要用分号分隔开。

## 2.2 C语言程序的书写风格

6、不强制规定语句在一行中的起始位置，但同一结构层次的语句应左对齐。低一层次的语句或说明可比高一层次的语句或说明缩进若干格后书写，以便看起来更加清晰，增加程序的可读性。属于同一模块时要用“{ }”括起来，如上例中的**do-while**语句和**switch**语句。

7、为了使程序更加清晰，可以使用空行，空行不影响程序的执行，但不要在一个语句内加空行。

## 2.2 C语言程序的书写风格

8、C语言中有的符号必须配对使用。如注释符号“/\* \*/”，模块起止符号“{ }”，圆括号“（ ）”等。在输入的时为了避免忘记，可连续输入这些起止符号，然后再在其中进行插入来完成内容的编辑。

9、在源程序中，凡是用“/\*”和“\*/”括起来的文字都是注释。可以在程序的任何一处插入注释。注释是对程序或其局部的说明，不参加编译也不在目标程序中出现。建议多使用注释信息，可以增加程序的可读性。



## 2.3 C语言程序的开发过程

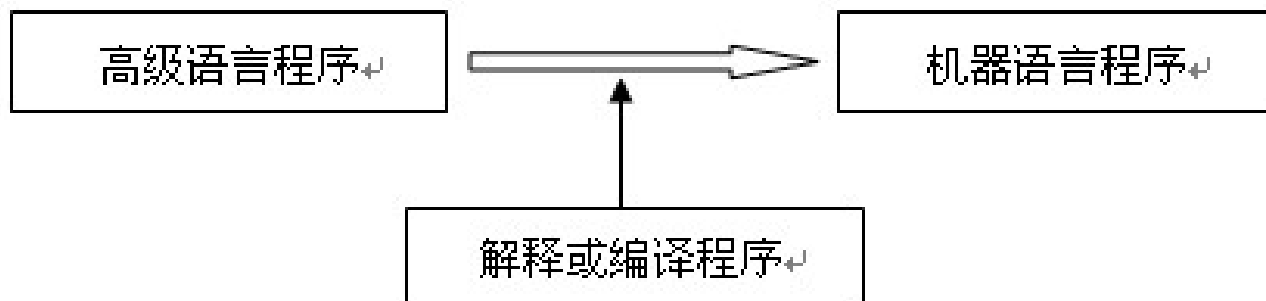


图 2-1 C 语言程序的“翻译”过程

## 2.3 C语言程序的开发过程

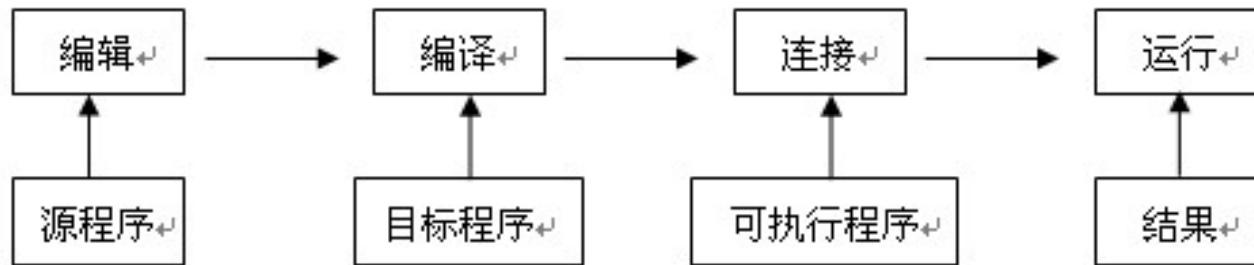


图 2-2 C 语言程序的开发过程

## 2.4 Visual C++集成开发环境

**Visual C++ 6.0**是一个基于**Windows**操作系统的可视化集成开发环境（**integrated development environment, IDE**），已成为专业程序员进行软件开发的首选工具，是目前非常盛行的一种C编译系统，功能十分强大，操作方便，视图界面友好。

## 2.4.1 熟悉Visual C++ 6.0集成开发环境

### 1. 安装Visual C++ 6.0

运行安装文件中的**setup.exe**程序，然后按照安装程序的提示信息进行操作，可以指定系统文件存放的路径，但一般不必自己另行指定，采用系统提示的默认方案即可完成安装过程。

## 2.4.1 熟悉Visual C++ 6.0集成开发环境

### 2. 启动Visual C++ 6.0

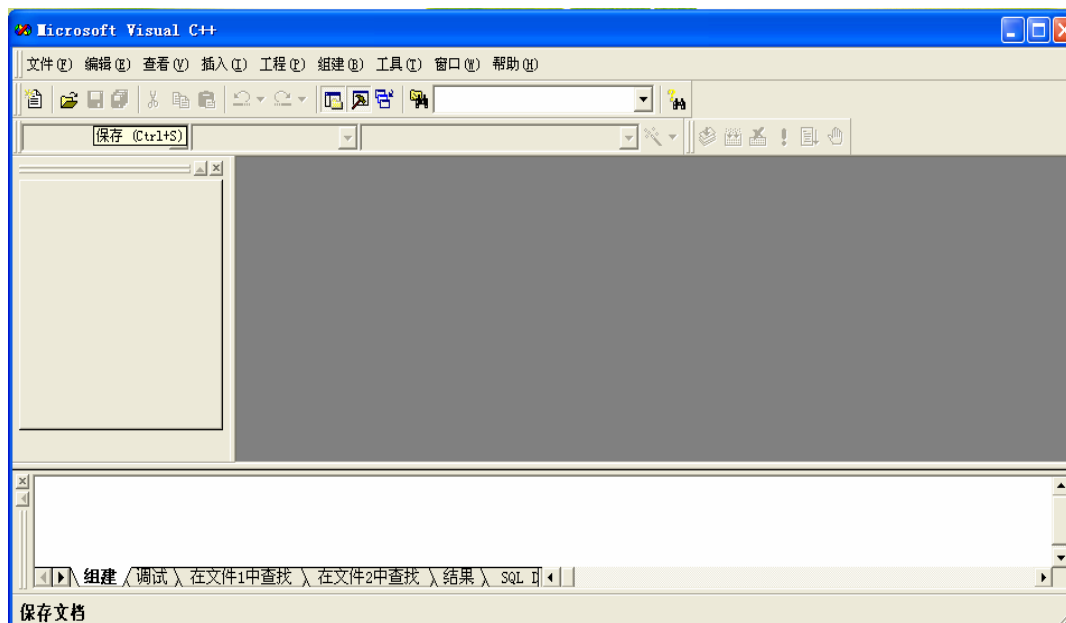


图 Visual C++ 6.0主窗口

## 2.4.2 C语言在Visual C++ 6.0的开发过程

如图2-4所示刚开始进入**Visual C++ 6.0**的界面时，里面的项目工作区和文本编辑区是空的，要开始一个新程序的开发时，需要通过应用程序向导建立新的工程项目，并在项目中添加文件，然后再进行其它的开发操作。

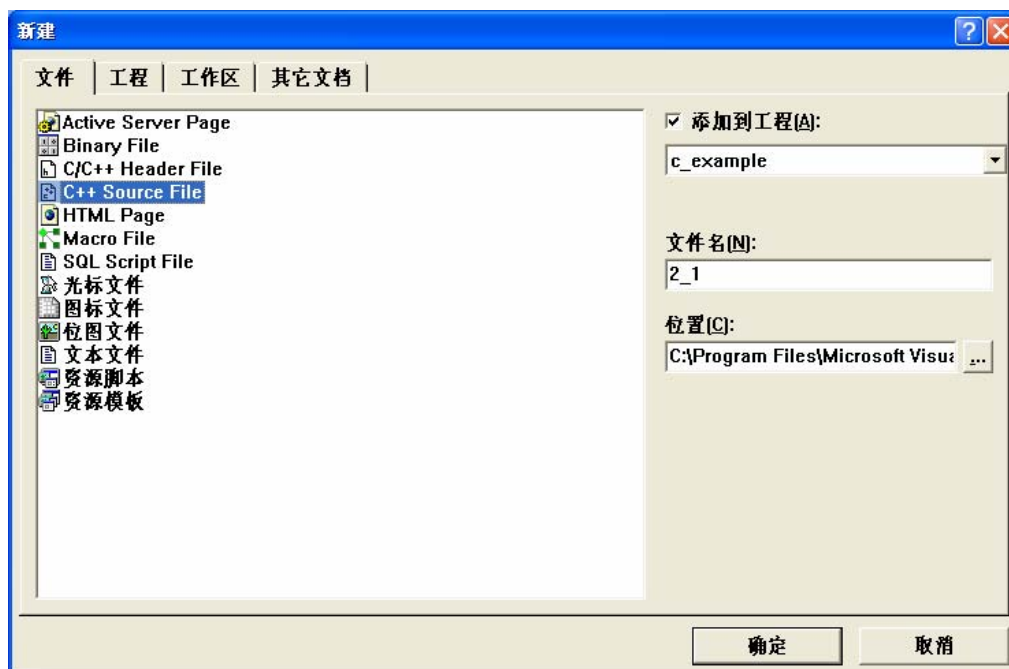
## 2.4.2 C语言在Visual C++ 6.0的开发过程

### 1. 新建工程项目



## 2.4.2 C语言在Visual C++ 6.0的开发过程

### 2. 建立项目中的文件





## 2.5 用Visual C++运行一个C程序

在上一节我们熟悉了**Visual C++ 6.0**集成开发环境，以及在里面的程序开发过程，现在我们来编辑并运行一个简单的C语言程序，来熟悉一下**Visual C++ 6.0**中的整个上机过程。

1. 编辑源程序
2. 编译连接源程序
3. 运行程序

```
#include<stdio.h>
void main( )
{
    int  a,b,sum;
    printf("输入第一个数a: ");
    scanf( "%d", &a );           /*输入a*/
    printf("输入第一个数b: ");   /*输入b*/
    scanf( "%d", &b );
    sum=a+b;                     /*对a、b求和*/
    printf("%d和%d的和是%d \n", a, b, sum);
}
```

## 第3章 常量、变量与标识符

C语言中的数据包括常量和变量，作为操作对象的数据都是以某种特定的形式存在的，可以用C语言中的标识符来表示一个常量或者一个变量。

标识符；

常量；

变量；

变量的初始化。

## 3.1 标识符

我们已经知道在C语言中，数据在计算机内存中存储的，程序设计中用到的数据，要到计算机的内存中读取，因此需要用到一个符号来代表它，这里就是我们所要讲的标识符。

标识符是指用来标识常量名、变量名、函数名、数组等对象，按照一定的命名规则定义的字符序列，即一个代号。

### 3.1.1 标识的命名

标识符的命名规则如下：

标识符由字母（包括大写字母和小写字母）、数字及下划线组成，且第一个字符必须是字母或者下划线。

在C语言中，大写字母和小写字母是有区别的，即作为不同的字母来看待，应引起注意。

## 3.1.2 保留字

保留字也称关键字，是指在高级语言中，那些已经定义过的标识符，用户不能再将这些字作为变量名、常量名、函数名、数组名等。

C语言共有32个关键字，具体可分为4类：

数据类型关键字（12个）：**char、double、enum、float、int、long、short、signed、struct、union、unsigned、void。**

控制语句关键字（12个）：**break、case、continue、default、do、else、for、goto、if、return、switch、while。**

存储类型关键字（4个）：**auto、extern、register、static。**

## 3.1.2 保留字

其他关键字（4个）：**const**、**sizeof**、**typedef**、**volatile**

。

C语言中除了上述的保留字外，还使用一些具有特定含义的标识符，称为特定字。如**include**、**define**、**ifdef**、**ifndef**、**endif**、**line**。这些特定标识符主要用在C语言的编译预处理命令中。

## 3.1.2 保留字

在C语言中，标识符的命名除了遵守命名规则、不使用关键字以外还要注意以下几点。

在C语言中，大写字母和小写字母是有区别的，即作为不同的字母来看待，因此**Teacher**、**TEACHER**是两个不同的标识符。

在起名时，应注意做到“见名知义”。比如表示姓名：比较好的标识符：**Name**、**name**、**xing\_ming**、**Xingming**、**xm**等；比较差的标识符：**x**、**y**、**abc**等。

尽量不用单个的“**l**”和“**o**”作标识符。这个与数字中的“**1**”和“**0**”很相像，程序设计过程中容易混淆。

代数计算时可以采用习惯的名字。如：圆的半径和面积：**r**、**s**；立方体的长、宽、高和体积：**a**、**b**、**h**、**v**。



## 3.2 常量

常量是指在程序运行过程中其值不随程序的运行而改变的量。常量在程序中不需要进行任何说明就可以直接使用，常量本身就隐含了它的类型。常量区分为不同的类型，分为直接常量和符号常量。

## 3.2.1 直接常量

直接常量是直接写出来的，直接常量的书写形式决定了它的类型。直接常量包括整型常量、实型常量、字符型常量和字符串常量。例如：

整型常量：15、-8、0。

实型常量：3.7、-8.2、58.12E-2。

字符常量：‘a’、‘A’、‘+’、‘5’。

字符串常量：“this is a boy.”、“a”、“123”。

## 3.2.2 符号常量

符号常量是指用一个标识符代表一个常量。如商场内某一产品的价格中发生了变化，如果我们在一个程序中多次用到了这种商品的价格，需要逐修改非常麻烦，这样可以定义一个符号常量，在文件的开头写这么一行命令：

```
#define PRICE 50
```

这里用**#define**命令行定义**PRICE**代表常量**50**，后面的程序中用到这种商品的价格时，直接用**PRICE**，可以和常量一样进行运算，如果常量的值需要发生变化，只需要在**#define**命令行进行修改，达到一改全改的目的。

## 3.2.2 符号常量

这里需要说明以下几点：

符号常量名习惯上用大写，以便与变量名相区分。

一个**#define**对应一个常量，占一行；**n**个常量时需**n**个**define**与之对应，占**n**行。（这将在第7章的预编译部分进行详细的讲解）。

符号常不同于变量，它的值在其作用域内不能改变，也不能再被赋值。

在程序中使用符号常量具有可读性好，修改方便的优点。

## 3.3 变量

变量是指在程序运行过程中其值可以改变的量。程序中使用的变量名是用户根据需要而取名，变量名必须符号标识符的命名规则。

在C语言中，由于程序的多样性的需要，对变量也有各种各样的要求，比如：变量的生命期，变量的初始状态，变量的有效区域，变量的开辟地和变量的开辟区域的大小等等，为了满足这些要求，C语言设置了以下变量：不同数据类型的变量、全局变量、局部变量、静态变量（静态全局变量和静态局部变量）、寄存器变量、外部变量等。这里我们只要先讲解不同数据类型的变量，在第6章我们将逐一对其他种类的变量进行讲解。

### 3.3.1 变量的定义

变量的定义需要注意以下几点。

每个变量定义语句都必须以分号结尾。

变量定义语句可以出现在变量使用之前的任何位置。  
程序设计时只要不违背“先定义，后使用”的原则即可。

变量一经定义，每一个变量就有一种确定的类型，在编译时就能为其分配相应的存储单元。

一个变量在内存中占据一定的存储单元，用变量名来标识在内存中所分配的存储单元，在该存储单元中存放变量的值。

### 3.3.2 变量的初始化

变量的初始化就是对变量赋初值。初始化变量并不是必须的，但是在c语言中未初始化的变量是其数据类型允许范围内的任意值（静态变量除外），为了防止运算中出错，一般建议定义变量后，立即初始化。变量的初始化有种方法：一种是定义初始化，即定义变量的同时对其赋予初始值。另一种方法是先定义变量，然后再进行赋值或是等到需要赋值的时候再赋值。

## 3.4 变量的初始化

对于变量的初始化，我们可以归纳以下几点：

- (1) 初始化实际上是一个赋值语句。
- (2) 在定义变量的时候，可以只给部分变量赋值。
- (3) 如果同时对几个变量赋相同的初值，应该注意书写格式。



## 3.4 变量的初始化

在C语言中，使用变量时，如果它出现在表达式中，事先必须有一个初始值，否则其值将是一个不确定的值。变量获取初始值有以下几种方法：

赋值语句：“=”在C语言中是赋值符号，运用赋值符号可以对变量进行赋值。

读取语句：在有些程序的值是不确定的需要用户自己输入，因此需要用读取语句从外部的输入。

## 第4章 数据类型

C语言具有丰富的数据类型。C语言中的数据类型分为四大类，即基本数据类型、构造类型、指针类型和空类型。本章重点介绍了基本基本数据类型。

C语言中的数据类型；

整型数据；

实数型数据；

字符型数据；

数值型数据间的混合运算。

## 4.1 C语言中的数据类型

所谓数据类型是按被说明量的性质、表示形式、占据的存储空间的多少，构造特点来划分的。在C语言中，数据类型可分为：基本数据类型、构造数据类型、指针类型、空类型四大类。

数据的类型不同，它们取值范围、运算属性以及存储方式等会不相同，C语言程序中所用到的数据都必须指明一定的数据类型后才能对数据进行各种操作。

## 4.1.1 基本数据类型

基本数据类型是语言系统定义的数据类型，只能有单一的值，在程序定义变量时可以直接引用。C语言中常用的基本数据类型有整型、实型、字符型。如在填写人的年龄时，使用的整型数据；学生的分数要用实型类型；学生姓名是由多个字符组成的。

## 4.1.2 构造数据类型

构造数据类型是由基本数据类型按一定的规则组合而成的，因此也称为导出类型数据。数组是由相同类型的数据组合而成的，如一班学生的数学成绩组合在一起，就是一个实数型数组。结构体是由不同类型的数据组合而成的，比如统记一个学生的信息包括学号（长整型）、学生姓名（字符型）、性别（字符型）、年龄（整型）等，所有的数据组合在一起就成了构造体。如果若干个数据不同时使用时，为了节省内存空间，我们就可以让它们占用相同的内存区域，这些数据组合起来就是共用体，它可以是同类型的数据，也可以不同类型的数据。

## 4.1.3 指针数据类型

指针是一种特殊的数据类型，是C语言的核心，也是C语言点所在，同时又是具有重要作用的数据类型，其值用来表示某个量在内存储器中的地址。在本书的第14章我们将会重点给予讲解。

## 4.1.4 空类型

空类型是从语法完整性的角度给出的一处数据类型，表示不需要具全的数据值，因此也就没有数据类型。空类型在调用函数值时，通常应向调用者返回一个函数值，这个返回的函数值是具有一定的数据类型的，应在函数定义及函数说明中给以说明。但是，也有一类函数，调用后并不需要向调用者返回函数值，这种函数可以定义为“空类型”，其类型说明符为**void**。

## 4.2 整型数据

整型数据分为一般整型、短整型和长整型，并且每一种类型又分为带符号和无符号两种类型。

表 4-1 整型数据

数据类型	说明	所占字节	取值范围
short [int]	短整型	2	-32768~32767
signed short [int]	带符号短整型	2	-32768~2767
unsigned short [int]	无符号短整型	2	0~65535
int	整型	4	-2147483648~2147483647
signed [int]	带符号整型	4	-2147483648~2147483647
unsigned [int]	无符号整型	4	0~4294967295
long [int]	长整型	4	-2147483648~2147483647
signed long [int]	带符号长整型	4	-2147483648~2147483647
unsigned long [int]	无符号长整型	4	0~4294967295



## 4.2.1 整型常量

整型常量的数据类型是整数，包括正整数、负整数和零。在C语言中，整型常量有以下三种不同的数制表示形式：

十进制整数常量：这种表示方法就是我们平时所熟悉的表示方法，以数字0~9构成，最高位也就是就左边第一位不能为0。例如，-39，0，171等

八进制整型常量：以数字0开头，其后再写上要表示的八进制数。八进制数各位由0~7这八个数字之一组成。例如0134，0471，-072。

十六进制整型常量：以0X或0x，其后再写上要表示的十六进制数。十六进制各位由数字0~9或字母a~f或A~F构成。如0x17，0XCF,-0X1f等。

## 4.2.2 整型变量

整型变量是指其值为整型数据的变量。整型数据有三种即整型(int)、短整型(short int)和长整型(long int)。为了方便书写,我们将short int和long int后面的int省略,分别用short和long来表示短整型和长整型。

### 1. 整型变量的定义

整型变量分为整型变量、短整型变量、长整型变量。

```
int a;           /*定义一个整型变量a*/
```

```
short d=16;      /*定义一个短整型变量d*/
```

```
long s;          /*定义一个长整型变量s*/
```

```
[signed] int a;   /*定义一个带符整型变量a*/
```

```
unsigned [int] num; /*定义一个无符号整型变量num*/
```

**【例4-1】**整型数据简单运算。

# 数据的溢出

【例4-2】整型数据的溢出。

```
#include <stdio.h>
void main()
{
    short a, b, c, d;
    a=32767;
    b=a+1;
    c=-32768;
    d=c-1;
    printf("%d\n%d\n%d\n%d\n", a, b, c, d);
}
```

## 4.3 实数型数据

实型数据表示的实际上就是带小数的数值，又称为浮点型数据。实型数据在单精度实型（**float**）、双精度实型（**double**）和长双精度实型三种，长双精度实型数据一般情况下很少用到。它们表示数值的方法是一样的，区别在于数据的精度、取值范围以及在内存中占用的存储空间有所不同。如表所示：

表 4.3 实型数据

数据类型	说明	所占字节	取值范围	有效数字
float	单精度实型	4	$-3.4 \times 10^{38} \sim 3.4 \times 10^{38}$	6~7
double	双精度实型	8	$-1.7 \times 10^{308} \sim 1.7 \times 10^{308}$	15~16
long double	长双精度实型	16	$-1.2 \times 10^{4932} \sim 1.2 \times 10^{4932}$	18~19

## 4.3.1 实数型常量

实型常量有两种表示形式：

### 1. 小数表示法

C语言中实数只能使用十进制小数表示，不能用八进制或十六进制表示。这种形式由符号、整数部分、小数部分和小数部分级成，其格式如下：

±整数部分. 小数部分

## 4.3.1 实数型常量

### 2. 指数表示法

用指数形式表示特别大或特别小的数值。指数形式的实数由尾数部分、字母E或e和指数部分组成。其格式如下：

±尾数部分E(e) ±指数部分

指数形式的表示方法实际等价于：

±尾数部分 $\times 10^{\pm \text{指数部分}}$

因此，12.3e3等价于 $12.3 \times 10^3$ ，0.12E+5等价于 $0.12 \times 10^5$ 。

10023.45可以表示为0.1002345e+5、1.002345e+4、10.02345e+3等，其中只有1.002345e+4才是规范化的指数形式。

## 4.3.1 实数型常量

需要说明以下几点：

实型常量的类型都是双精度浮点型。

实数在计算机中只能近似表示，运算中也会产生误差

。

小数部分和指数部分具体有多少位,没有具体的标准,不同的编译系统有不同的规定。小数部分越多，精确度越高；指数部分越多，数值的范围就越大。

## 4.3.2 实数型变量

在程序运行过程中可以改变其值的实型量被称为实型变量。实型变量分为单精度(float)、双精度(double)和长双精度三种类型。在定义实型变量时用以下方式：

```
float x;           /*定义float型变量x*/  
double y;          /*定义double型变量y*/  
long double z,     /*定义long double型变量z*/
```

**【例4-3】** 测试单精度实型的有效位数。



## 4.3 字符型数据

字符型数据指的是由字母、符号和不用于算术操作的数字组成，又称为非数值型数据。字符型数据分为字符型（**char**）、带符号字符型(**signed char**)和无符号字符（**unsigned char**）。

表 4.3 字符型数据

数据类型	说明	所占字节	取值范围
char	字符型	1	-128~127
signed char	带符号字符型	1	-128~127
unsigned char	无符号字符型	1	0~255

## 4.3.1 字符型常量

字符型常量包括由一对单引号括起来的一个字符构成的一般字符常量和由反斜杠（\）开头的特定的字符序列构成的转义字符。

### 1. 一般字符常量

字符型常量是由一对单引号括起来的一个字符。这个字符是ASCII字符集中的字符，字符常量的值为该字符的ASCII值。例如：

**'A'、'x'、'D'、'?'、'3'、'X'**

## 2. 转义字符

转义字符是指由反斜杠（\）开头的特定的字符序列。C语言允许使用这种特殊形式的字符常量，因为在程序设计过程中，有一些字符如回车符、退格符、制表符等控制符号，不能在屏幕上显示，也不能从键盘上输入，只有用转义字符来表示。

表 3-4 转义字符

转义字符	含义	转义字符	含义
\n	换行（相当于Enter键）	\t	水平制表符（相当于Tab键）
\v	垂直制表符	\b	退格（相当于Backspace键）
\r	回车	\f	换页
\\	输出一个反斜杠\	\'	输出一个单引号
\"	输出一个双引号	\0	表示空
\ddd	1至3位八进制数所代表的字符	\xhh	1到2位十六进制数所代表的字符

【例4-4】转义字符应用举例。

## 4.3.2 字符型变量

字符型变量就是用一个标识符表示字符型数据，并且该标识符的值可以发生变化。字符变量只能存放一个字符。

### 1. 字符型变量的定义与存储

字符型变量就是值为字符常量的变量。字符变量只能存放一个字符。

字符型变量的定义与整型变量、实型变量的定义相同，如下：

```
char c1, ch1;
```

例如：

```
char ch;
```

```
ch='a';
```

【例4-5】字符型数据的输出

【例4-6】字符型数据的运算。

## 4.4 数值型数据间的混合运算

C语言中，一般情况下相同类型的数据可直接进行运算，运算的结果就是这种类型。例如：

**5.0/2.0：**参加运算的两个数都是实型，结果为实型**2.5**

。

**5/2：**参加运算的两个数都是整型，结果为整型**2**。

## 4.4.1 自动类型转换

自动类型转换是由系统自动完成的，又称为隐式转换。不同类型的数值进行运算时，系统会自动将级别低的类型转换成级别高的类型，然后再进行运算，运算结果与其中级别高的操作数的类型相同。

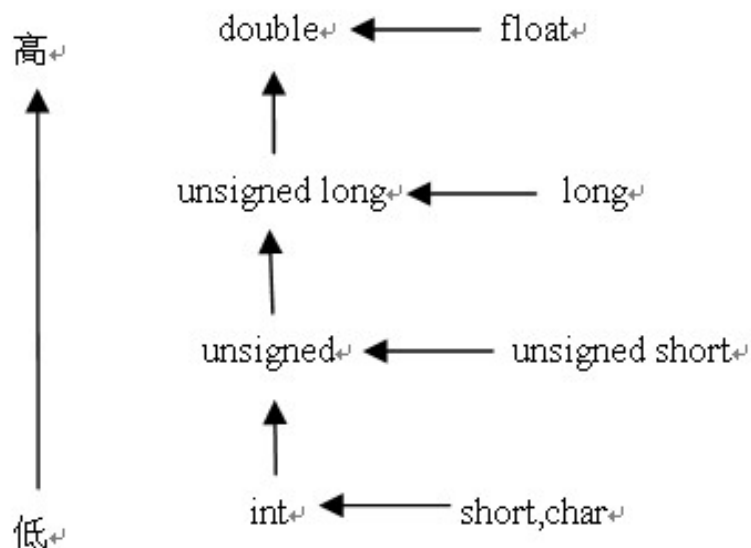


图 3-5 自动类型转换规则

## 4.4.2 强制类型转换

强制类型转换是利用强制类型转换运算符将安然将数据类型转换成所需要的类型。强制类型转换符是由一对圆括号将某个类型名括起来构成的。

强制类型转换的语法格式为：

(类型名) 表达式

(double)a     /\*将变量a转换成double型\*/

(int)(x+y)    /\*将x与y的和转换成整型\*/

(int)x+y    /\*先将x转换成整型，然后再与y求和\*/

【例4-7】 求一个浮点数的个位数字。

【例4-8】 强制类型转换示例

# 数据类型之间的转换

- (1) 实型之间的转换
- (2) 整型与实型之间的转换
- (3) 字符型与实型之间的转换
- (4) 整型之间的转换
- (5) 有符号数向无符号数转换时



## 第5章 运算符及其表达式

运算符是指用来对运算对象进行各种运算的操作符号。表达式是指由多个运算对象和运算符组合在一起的合法算式。其中运算对象包括常数、常量、变量和函数。本章内容如下：

- 算术运算符及算术表达式；
- 赋值运算符及赋值表达式；
- 关系运算符及关系表达式；
- 逻辑运算符及逻辑表达式；
- 条件运算符及条件表达式；
- 逗号运算符及逗号表达式；
- 位运算符。

## 5.1 算术运算符及算术表达式

算术运算符包括基本算术运算符和自增、自减运算符。基本算术运算符是对数值型也包括字符型数据进行加、减、乘、除的四则运算。

表 5-1 算术运算符

运算符	名称	应用举例	实现功能
+	加法运算符	$a+b$	求a与b的和
	正值运算符	+5	表示正数5
-	减法运算符	$a-b$	求a与b的差
	负值运算符	-8	表示负8
*	乘法运算符	$a*b$	求a与b的乘积
/	除法运算符	$a/b$	求a除以b的商
%	求余运算符	$a\%b$	求a除以b的余数
++	自增运算符	$++a$	变量a的值加1，等价于 $a+1$
--	自减运算符	$--b$	变量b的值减1，等价于 $b-1$

## 5.1.1 算术运算符

算术运算符的具体运用原则：

+（正）、-（负）运算符是属于同一级别的单目运算符，结合方向是自右向左。

+（加）、-（减）运算符是属于同一级别的双目运算符，结合方向是自左向右。例如 $a+b-c+d$

\*, /, %是同一级别的双目运算符，结合方向是自左向右。例如： $a+b*c$ ，运算顺序是先计算 $b$ 与 $c$ 的乘积，然后再与 $a$ 求和，即 $a+(b*c)$ 。

## 算术运算符注意以下几点:

/（除法运算符）的除数不能为0，即不能用一个数去除以0。

\*（乘号运算符）在式子中不能省略，也不能写成是代数式子中的乘号“×”或“·”。例如：求长方体的体积公式为 $abc$ ，在编程时要写成： $a*b*c$ 。

如果两个整型数相除，得到整型结果。如果两个实数相除或其中有一个是实数，那么得到结果为实型。例如：

$5/3=1$ ， $2/4=0$ ， $5/-3=1$ ， $5./3=1.666667$ ， $5.0/3.0=1.666667$

%求余运算符（或称求模运算），只适合于整型数据和字符型数据。求余运算的结果符号与被除数相同，其值等于两数相除后的余数。

$5\%3$             /\* 值为2 \*/

$-7\%-3$         /\* 值为-1 \*/

## 算术运算符注意以下几点：

++、--（自增、自减运算符）属于同一级别的单目运算符，结合方向是自右向左。自增、自减运算符只能与变量结合使用，放在变量的前面或者是后面。有以下4种形式：

++a：a的值先增加1后，再参与其他运算。

a++：a的值先参与其他运算，再使a的值增加1。

--a：a的值先减小1后，再参与其他运算。

a--：a的值先参与其他运算，再使a的值减小1。

例如：m=3;m1=m++; 等价于m=3; m++; m1=m;

**【例5-1】**“++”和“--”运算符在的使用。

对于自增、自减运算符，做以下几点说明：

自增或自减函数只能用于变量，不能用于常量或表达式。例如：  
(a+b)++这样的表示方法是错误的。

在一个表达式中对一个变量自增或自减多次，可能造成困惑。  
a=3;k=(++a)+(++a);这种程序很容易出错，在编程的过程中要避免使用这样的程序，而且也没有必要使用如此难懂的程序，完全可以使用另一种方法来表示，增加程序的可读性。

++、--运算符的结合方向是自右向左，如：-i++等价于-(i++)。

++、--运算符的优先级大于乘、除、求余的优先级。

++、--运算符运算的操作对象只能为整型变量、字符型变量和指针变量，而不能是其他类型的变量。

++、--运算符运算常用于循环变量中，是循环变量自动加1或减1；也可用于指针变量，是指针指向前一个或后一个地址。

## 5.1.2 算术表达式

用算术运算符将运算对象即运算量或操作数连接起来，构成符号C语言语法规则的式子，称为算术表达式。算术表达中，运算对象包括常量、变量和函数。

例如： **$x+y*a/x-5\%3$** ， **$3.5+56\%10+3.14$** ， **$a++*1/3$** 。

## 5.1.2 算术表达式

关于算术表达式有以下几点说明：

算术表达式的求值顺序按算术运算的优先级别高低次序进行，先执行优先级别高的，再执行优先级别低的。

以表达式 $8\%3+9/2$ 为例， $\%$ 、 $/$ 运算符的优先级高于 $+$ 运算的优先级，因此在运算的过程中先算求余和除法， $8\%3=2$ ， $9/2=4$ ，然后再求和 $2+4=6$ ，因此最后的结果为6。

在算术表达式中，运算对象有常量，也有变量。

例如：

`'a'+5*2`

`(double)(8%3)`

**【例5-2】**算术表达式的应用举例。



## 5.2 赋值运算符及赋值表达式

C语言的赋值运算符包括简单的赋值运算符和复合赋值运算符，本小节主要讲解简单的赋值运算符，复合赋值运算符将在位运算符一节中进行详细的说明。

## 5.2.1 赋值运算符

赋值运算符与代数里面的等号相同，即“=”。赋值运算符的作用是把运算符右边的表达式的值赋给其左边的变量，其结合性是从右向左。例如：

**a=5;**

表 5-2 赋值运算符

运算符	名称	应用举例	实现功能
=	赋值	$a=b$	将b的值赋给a
+=	加赋值	$a+=b$ 等价于 $a=a+b$	将a加b的和赋给a
-=	减赋值	$a-=b$ 等价于 $a=a-b$	将a减b的差赋给a
*=	乘赋值	$a*=b$ 等价于 $a=a*b$	将a乘b的积赋给a
/=	除赋值	$a/=b$ 等价于 $a=a/b$	将a除以b的商赋给a
%=	求余赋值	$a\%=b$ 等价于 $a=a\%b$	将a除以b的余数赋给a
&=	位与赋值	$a\&=b$ 等价于 $a=a\&b$	将a与b按位与后赋给a
=	位或赋值	$a =b$ 等价于 $a=a b$	将a与b按位或后赋给a
^=	按位异或赋值	$a\^=b$ 等价于 $a=a\^b$	将a与b按位异或后赋给a
<<=	位左移赋值	$a<<=2$ 等价于 $a=a<<2$	将a左移2位后赋值给a
>>=	位右移赋值	$a>>=2$ 等价于 $a=a>>2$	将a右移2位后赋值给a

## 5.2.1 赋值运算符

对赋值运算符我们有下列几点认识：

(1) 赋值运算符“=”左边必须是变量，右边可以常量、变量，也可以是函数调用或表达式。

例如：  $s = a * b / c - 12.34$

`int a; a = 'b';`

(2) 赋值与运算符“=”与数学中的等号“=”看起来相同，但是它们的含义、作用完全不同。

`a = a + 2;`

(3) 复合赋值运算符是由其他运算符与基本的赋值运算符组合而成的

。

例如：

`a += 3;` 相当于：`a = a + 3;`，

`b *= a;` 相当于 `b = b * a;`。

`h /= x + y;` 不能理解为： $h = h / x + y$ ，应该理解为  $h = h / (x + y)$

## 5.2.2 赋值表达式

由赋值运算符将一个变量和一个表达式连接起来的式子称为赋值表达式。一般的书写形式如下：

变量 赋值运算符 表达式

赋值表达式：

$a=10$              $b=c+d$              $a/=d+2$

赋值语句：

$a=10;$              $b=c+d;$              $a/=d+2;$

## 5.2.2 赋值表达式

对于赋值表达式，需要说明以下几点：

(1) 赋值运算符的左边必须为变量，而赋值表达式的左边可以是变量，也可以是赋值表达式。当赋值表达式的左边是赋值表达式的时候，应该带上括号。

$(a=3*4)=4*6$  （正确）

$a=3*4=4*6$  （错误）

(2) 赋值表达式的右边的表达式可以是一个算术表达式、关系表达式、逻辑表达式等等，也可以是一个赋值表达式。例如： $c2=c1=5$ 相当于 $c2=(c1=5)$

(3) 赋值表达式里面可以包含复合赋值运算符。

例如： $c2=c1+=1$ 相当于 $c2=(c1+=1)$

(4) 在C语言中，赋值操作不仅出现在赋值语句中，而且可以以表达式形式出现在其他语句中。

```
printf( "%d", a=b=3 );
```

## 5.3 关系运算符及关系表达式

C语言中关系运算常用于选择结构、循环结构的条件判断。由关系运算符连接的式子称为关系表达式，用于条件的判断。

## 5.3.1 关系运算符

表 5-3 关系运算符

关系运算符	名称	应用举例	实现功能
<	小于	$a < b$	a小于b
<=	小于或等于	$a \leq b$	a小于或等于b
>	大于	$a > b$	a大于b
>=	大于或等于	$a \geq b$	a大于或等于b
=	等于	$a = b$	a等于b
!=	不等于	$a \neq b$	a不等于b

关系运算符是用来比较两个运算量大小的运算符，实际上就是一种“比较运算”，运算的结果只能是“1”或“0”。当两者的比较关系成立的时候，结果为“1”；当两者的比较关系不成立的时候，结果为“0”，因此关系运算符的结果类型为整型。

## 5.3.1 关系运算符

对关系运算符进行以下几点说明：

(1) 关系运算符的优先级别比算术运算符的级别低，但比赋值运算符的级别高。

例如： $a=2*2<8$                       顺序为 $a=((2*2)<8)$

(2) 关系运算符用于比较的两个运算量的类型为整型、字符型等，也可以连接两个表达式，比较的结果是一个逻辑量，即“真”或“假”，在C语言中没有逻辑型数值，分别用整数1和0表示。

(3) 关系运算符的结合方向是从左向右，因此当一个表达式中出现优先级相等的关系运算符时，从左向右开始运算。

(4) 在关系运算符用“ $=$ ”表示等于，用“ $!=$ ”表示不等于，这与数学中的表示方法完全不同，因此在编程中要特别注意，以免写错关系运算符而导致错误的结果。

$a==b$                $a=b$



## 5.3.2 关系表达式

用关系运算符将两个表达式连接起来构成的式子称为关系表达式。一般的书写形式如下：

表达式 关系运算符 表达式

例如：

`'A' > (c='a')`

该关系表达式中的表达式为赋值表达式，将字符'a'赋值给变量c，即`'A' > 'a'`，'A'的ASCII值为65，'a'的ASCII值为97，即`65 > 97`，关系不成立，关系运算的结果为0。

## 5.4 逻辑运算符及逻辑表达式

逻辑运算符与关系运算符经常放在一起使用。关系运算是指数与值之间的关系，逻辑运算是将真值和假值连接在一起的方式。由于关系运算符产生了真或假的结果，所以关系运算表达式中常常使用逻辑运算符。

## 5.4.1 逻辑运算符

逻辑运算符是对两个含有关系运算符的表达式或逻辑值进行的运算符号，运算的结果为逻辑值。

表 5-4 逻辑运算符

逻辑运算符	名称	结合性	应用举例	功能说明
&&	逻辑与	自左向右	a&&b	a与b相与
	逻辑或	自左向右	a  b	a与b相或
!	逻辑非	自右向左	!a	非a，逻辑值与a相反

## 对逻辑运算符作出几点说明：

(1) “&&”和“||”是双目运算，需要两个操作数，如  $a \&\&b$ ， $a||b$ 。而“!”是单目运算符，只需要一个操作数，如  $!a$ 。

(2) 逻辑非的优先级高于逻辑与的优先级，而逻辑与的优先级又高于逻辑或的优先级。

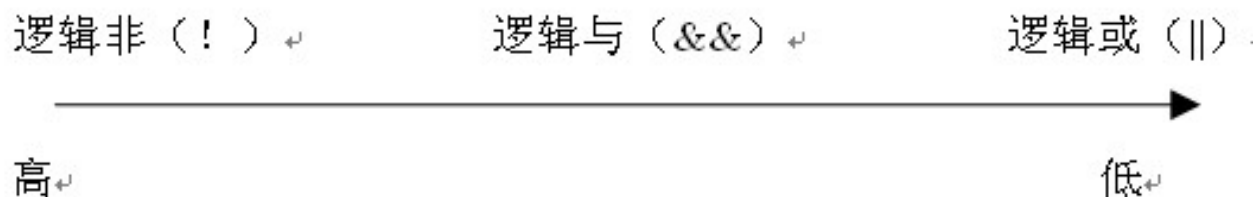


图 5-2 三个逻辑运算符的优先顺序

(3) 运算符之间的运算优先顺序是逻辑非（！）运算符优先级最高，算术运算符优先级高于关系运算符，关系运算符又高于逻辑与（&&）和逻辑或（||），而赋值运算符优先级最低。

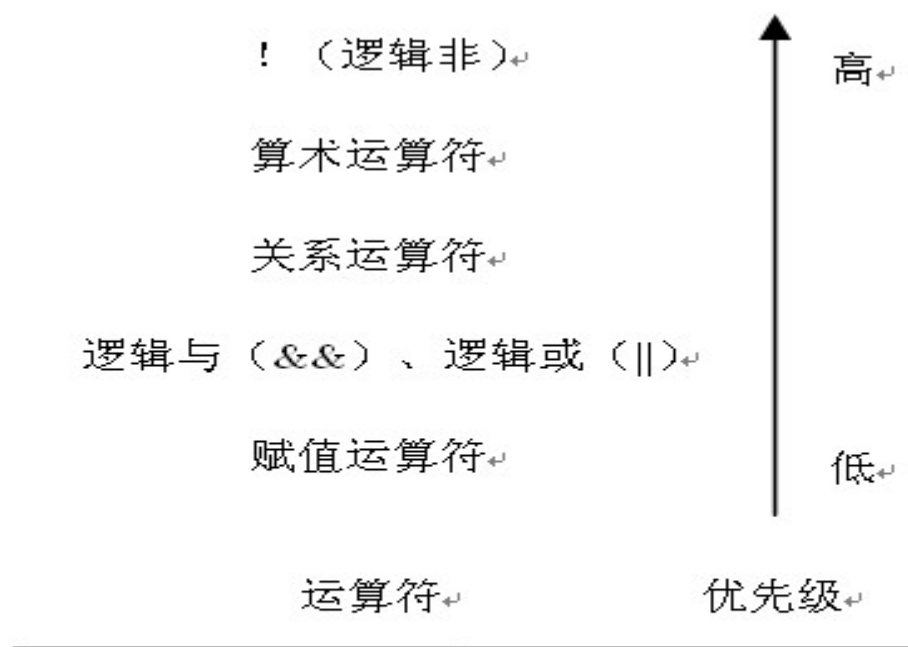


图 5-3 逻辑运算符的优先级

## 5.4.2 逻辑运算规则

表 5-5 逻辑运算规则

对象 a	对象 b	$a \& b$	$a \parallel b$	$!a$
0	0	0	0	1
0	非 0	0	1	1
非 0	0	0	1	0
非 0	非 0	1	1	0

逻辑运算的运算规则可简单归纳为：

逻辑与同真为真

**$a \& b \& c$**

逻辑或同假为假

**$a \parallel b \parallel c$**

逻辑非遇假变真

**$!a$**

## 5.4.3 逻辑表达式

由逻辑运算符连接起来构成的表达式称为逻辑表达式。逻辑运算的对象通常是关系表达式逻辑表达式，也可以是算术表达式、赋值表达式等其他表达式。

例如：

**a>10 && a<15**            /\*逻辑表达式的运算对象是关系表达式\*/

**!(a<=10) && !(a>=15)**   /\*逻辑表达式的运算对象是逻辑表达式\*/

**(m=a>b)&&(n=c>d)**       /\*逻辑运算符的运算对象是赋值表达式\*/

**t=++x||++y&&++z**       /\*逻辑运算符的运算对象是算术表达式\*/



逻辑运算有时比较复杂，需要注意以下几点：

(1) 在一个逻辑表达式中可以包含多个逻辑运算和其他各种运算符，首先注意数值哪个是数值运算，哪些是关系运算，哪些是逻辑运算，搞清各个运算符之间的关系，然后按它们的优先级进行运算。

(2) 逻辑表达式在进行求值的过程中，不一定必须将表达式求值到底，这是逻辑运算的特殊性所在，称为短路运算。例如：a&&b&&c

**【例5-6】**输入若干个字符，分别统计数字字符的个数、英文字母的个数，当输入换行符时输出统计结果，运行结束。

## 5.5 条件运算符及条件表达式

条件运算符是C语言中唯一的三目运算符，它根据一个表达式的结果等于**true**还是**false**，执行两个表达式中的一个。由于涉及到三个操作数——一个用于判断的表达式和另外两个表达式，因此这个运算符也称为三元运算符。

## 5.5.1 条件运算符

条件运算符由符号“?”和“:”组合而成的。条件运算符有三个运算对象，三个运算对象都表达式。第一个运算对象可以是任何类型的表达式，如算术表达式、关系表达式、赋值表达式和逻辑表达式等等，后面两个表达式是类型相同的任何表达式。

表 5-6 条件运算符

运算符	名称	结合性	应用举例	功能说明
? :	条件运算符	自右向左	a?b:c	如果a的逻辑值为真，计算b，生成该操作的结果；如果a的逻辑值为假，计算c，生成该操作的结果

## 条件运算符的优先级

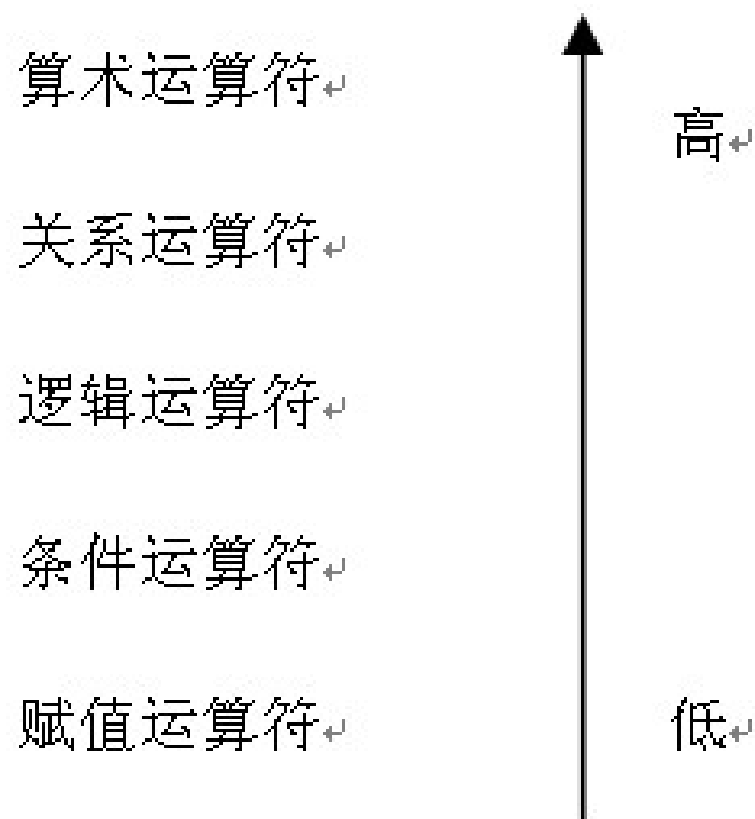


图 5-4 条件运算符的优先级

## 5.5.2 条件表达式

由条件运算符连接而构成的表达式称为条件表达式。  
一般的表达形式为：

表达式1 ? 表达式2 : 表达式3

## 5.5.2 条件表达式

关于条件表达式做以下几点说明：

(1) 条件表达式中含有三个操作对象，它们都是表达式，可以是各种类型的表达式。通常情况下，表达式1是关系表达式或逻辑表达式，用于描述条件表达式中的条件，根据条件的真假来判断是进行表达式2的运算还是进行表达式3的运算。表达式2和表达式3可以是常量、变量或表达式如算术表达式、关系表达式、赋值表达式和逻辑表达式等。

## 5.5.2 条件表达式

(2) 条件表达式的求解过程:

第一步: 求解表达式1的值。

第二步: 如果表达式1的值为真即为非0, 求解“表达式2”的值作为整个条件表达式的值。

第三步: 如果表达式1的值为假即等于0, 求解“表达式3”的值作为整个条件表达式的值。

## 5.5.2 条件表达式

(3) 条件表达式允许嵌套使用，即允许条件表达式中的表达式2和表达式3又是一个条件表达式。

(4) 一般情况下，条件表达式与结构程序设计中的if语句可以进行相互替换。

(5) 条件表达式中，表达式1的类型可以与表达式2、表达式3的类型不同，表达式2与表达式3的类型也可以不同，此时表达式值的类型为两者较高类型。



## 5.6 逗号运算符及逗号表达式

逗号在C语言中可以作为一种运算符使用，称为逗号运算符。

## 5.6.1 逗号运算符

表 5-7 逗号运算符

运算符	名称	结合方向	应用举例	功能说明
,	逗号运算符	自左向右	$a+b, c+d$	先计算表达式 $a+b$ 的值，再计算 $c+d$ 的值，将 $c+d$ 的值作为运算的结果。

(1) 逗号运算符是双目运算符，运算的对象可以是任何类型的表达式，运算的结果值是最后一个表达式的值。

(2) 逗号运算符是所有运算符中优先级最低的。

(3) 逗号运算符的结合方向是自左向右。逗号运算符将表达式连接起来，运算的时候按连接的顺序依次进行运算，所以又称顺序求值运算符。

(4) 并不是任何地方出现的逗都是作为逗号运算符，有的时候逗号是用于各个对象之间的间隔。

```
printf("%d, %d, %d", a, b, c);
```

## 5.6.2 逗号表达式

用逗号运算符将表达式连接起来构成的表达式就称为逗号表达式。逗号表达式的语法格式为：

表达式1， 表达式2

## 5.6.2 逗号表达式

对于逗号表达式作以下几点说明：

(1) 逗号表达式的求解过程是：先求解表达式1，再求解表达式2。整个逗号表达式的值是表达式2的值。

(2) 逗号表达式在求解的过程中要注意各个运算符之间的优先级，逗号运算符的优先级最低。

(3) 逗号表达式可以进行嵌套，即表达式1和表达式2本身也可以是逗号表达式。

(4) 逗号表达式无非是把若干个表达式“串联”起来，按表达式出现的顺序依次求值，表达式“(a=12, a\*4), a+5”等价于“a=12, a\*4, a+5”，即表达式进行顺序求值。

逗号表达式的一般形式可以扩展为：

表达式1，表达式2，表达式3.....表达式n

## 5.7 位运算符

位运算是**C**语言的一种特殊运算功能，它是以二进制位为单位进行运算的，即进行数的二进制位的运算。

位运算符分为位逻辑运算符、位移运算符和位自反赋值运算符三种。

位运算对象只能是整型（**int,short,unsigned,long**）或字符型（**char**）数据。

## 5.7.1 位逻辑运算符

位运算是指对二进制数按位进行运算，其操作对象是一个二进制位集合，每个二进制位只能存放0或1。位逻辑运算符是将数据中的每个二进制位上的“0”或“1”看成逻辑值，逐位进行逻辑运算的位运算符。

### 1. 位逻辑运算符

表 5-8 位逻辑运算符

运算符	名称	应用举例	功能说明
$\sim$	按位非	$\sim a$	将a按位取反
$\&$	按位与	$a \& b$	将a和b按位相与
$ $	按位或	$a   b$	将a和b按位相或
$\wedge$	按位异或	$a \wedge b$	将a和b按位相异或

表 5-9 位逻辑运算符的运算规则

对象a	对象b	$a \& b$	$a   b$	$a \wedge b$	$\sim a$
0	0	0	0	1	1
0	1	0	1	0	1
1	0	0	1	0	0
1	1	1	1	1	0

## 2. 位逻辑运算符的优先级

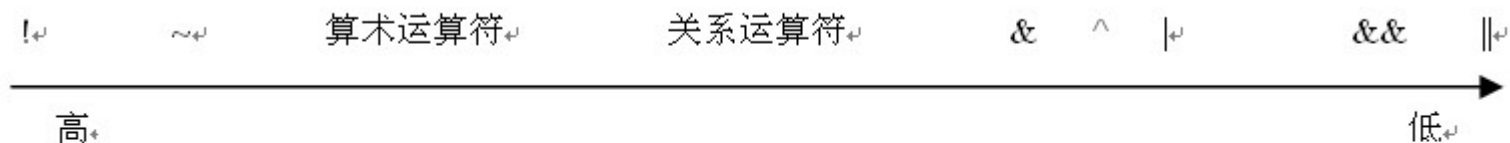


图 5-9 位逻辑运算符

## 5.7.2 移位运算符

移位运算是将数据看成二进制数，对其进行向左或右移动若干位的运算。移位运算包括左移位运算和右移位运算。

表 5-10 移位运算符

运算符	名称	应用举例	功能说明
<<	左移运算符	$a << b$	将a向左移b位
>>	右移运算符	$a >> b$	将a向右移b位



## 5.7.3 位自反赋值运算符

复合赋值运算符是由某个规定的运算符和基本赋值运算符组合而成的，当组成复合赋值运算符中的“某个规定的运算符”是位运算符时，复合赋值运算符就称为位自反赋值运算符。

表 1 位自反赋值运算符

赋值运算符	名称	应用举例	实现功能
$\&=$	位与赋值	$a\&b$ 等价于 $a=a\&b$	将a与b按位与后赋给a
$ =$	位或赋值	$a b$ 等价于 $a=a b$	将a与b按位或后赋给a
$\wedge=$	按位异或赋值	$a\wedge b$ 等价于 $a=a\wedge b$	将a与b按位异或后赋给a
$\ll=$	位左移赋值	$a\ll 2$ 等价于 $a=a\ll 2$	将a左移2位后赋值给a
$\gg=$	位右移赋值	$a\gg 2$ 等价于 $a=a\gg 2$	将a右移2位后赋值给a

## 5.8 长度运算符

长度运算符是由一个关键字sizeof表示的，它用于计算一种数据类型所占用的字节数。

表示格式为：

sizeof(类型说明符或变量)

表 5-12 长度运算符

运算符	名称	应用举例	功能说明
sizeof	长度运算符	sizeof(int)	求整型数据类型的内存字节数
		sizeof(a)	求变量a在内存中所占的空间

对长度运算符做以下几点说明：

(1) 长度运算符 (sizeof) 是一个单目运算符，它的功能是返回给定类型的运算对象所占内存字节的个数，因此长度运算的结果是一个整型数。

(2) 长度运算符的运算对象除了数据类型符说明符以外，还可以数组名或表达式等。如果运算对象是一个表达式（如常量、变量、数组名、结构体变量、共用体变量等），则 sizeof() 不会对表达式求值，只给出该表达式所占用的字节数。

(3) 长度运算符的优先级与其他单目运算符如~、!、++、--是同级别的，结合性是自右向左。

## 第6章 输入与输出

数据的输入与输出是一个算法所具有的特点。任何高级语言必须有数据的输入和输出功能，**C**语言本身不提供输入和输出语句，输入与输出操作是由函数来实现的。本章将介绍**C**语言中的各种语句以及数据输入与输出。本章内容如下：

**C**语句概述；

输入与输出函数；

整型数据的输入与输出；

浮点型数据的输入与输出。

## 6.1 C语句概述

一般的**C**语言语句包括数据描述和数据操作两部分。数据描述由声明部分完成，如变量定义等，没有操作；数据操作由语句来实现的，即程序的执行部分，程序的功能也是由执行语句实现的。

## 6.1.1 流程控制语句

**C**语言提供了多种语句来实现这些流程的结构。流程控制语句用于控制程序的流程，用于实现程序的各种结构方式，它们由特定的语句定义符组成。

表 6-1 流程控制语句

if.....else	条件语句
for( )	for循环语句
while( )	while循环语句
do-while	do-while循环语句
continue	结束本次循环，继续下一轮
break	终止执行switch或循环语句
switch	多分支选择语句
goto	转向语句
return	返回语句

## 6.1.1 流程控制语句

### 1. 条件判断语句

**if...else**语句

**switch...case**语句

### 2. 循环执行语句

**for(表达式1;表达式2;表达式3)**语句

**while(条件表达式)**语句

**do{}while(条件表达式)**语句

### 3. 转向语句

**break**语句、**continue**语句、**return**语句、**goto**语句。

## 6.1.2 函数调用语句

函数调用语句是由函数名、实际参数列表组成，以分号结尾。其一般形式为：

函数名(实际参数表);

执行函数语句就是调用函数体并把实际参数赋予函数定义中的形式参数，然后执行被调函数体中的语句，求取函数值。例如：

**printf("how are you!");**

调用C语言中标准输出函数，用来输出一个字符串。

**getchar();**

调用C语言系统标准库函数，字符输入函数，用来输入一个字符。

**max(a,b);**



## 6.1.3 表达式语句

表达式后面加上一个分号就构成了一个表达式语句。在前面第5章中我们学习了算术表达式、关系表达式、逻辑表达式、赋值表达式等，任何表达式加上一个分号都可以构成表达式语句，执行表达式语句就是计算表达式的值。

**a=4;                   /\*赋值表达式语句\*/**

**i++;                   /\*自增运算表达式语句\*/**

**a+=b+c;               /\*复合赋值表达式语句\*/**

**~ i | j>>3;           /\*位运算表达式语句\*/**

**'x'+1>c , -a-5\*b<=d+1;   /\*逗号表达式语句\*/**

**C语言中的语句大部分都是表达式语句。**

## 6.1.4 空语句

空语句又称空操作语句，只有一个分号，在程序执行过程中不作任何操作。空语句的作用一是在循环语句中使用空语句提供一个不执行操作的空循环体；二是为有关语句提供标号，用以说明程序执行的位置。

```
while(getchar()!='\n') ;      /*空语句*/
```

这里的循环体由空语句构成。该语句的功能是，只要从键盘输入的字符不是回车换行则重新输入。空语句的存在只是语法完整性的需要，其本身并不代表任何动作。

## 6.1.5 复合语句

把多个语句用括号“{ }”括起来组成的一个语句称复合语句。一般语句格式为：

```
{  
语句1;  
语句2;  
...  
语句n;  
}
```

## 6.1.5 复合语句

复合语句从形式上看是多个语句的组合，但在语法意义上它是一个整体，相当于一条语句，所以凡是可以用简单语句的地方都可以用复合语句来实现。在程序设计中复合语句被看成是一条语句，而不是多条语句。例如：

**for**循环语句的循环体使用复合语句。

```
for(i=1;i<=10;i++)
```

```
{
```

```
z=x+y;
```

```
t=z/100;
```

```
printf("%d",t);
```

```
}
```

## 6.1.5 复合语句

复合语句内的各条语句都必须以分号“;”结尾，在括号“{”外不需加分号。组成复合语句的语句数量不限，一个语句可以占多行，一行也可以有多个语句。在复合语句中不仅有执行语句，还可以说明变量。例如：

```
{  
    int c;    /*定义变量*/  
    c=getchar();  
    putchar(c);  
}
```

## 6.2 输入与输出函数

**C**语言程序是由函数构成的，输入与输出操作都是由函数来实现的。**C**语言中没有提供输入的语句，数据的输入与输出是利用系统函数来完成的。在**C**语言标准函数库中有一些输入输出函数，可以在程序中直接调用，其中包括：**scanf()**（格式输入函数）、**printf()**（格式输出函数）、**getchar()**（字符输入函数）、**putchar()**（字符输出函数）、**gets()**（字符串输入函数）、**puts()**（字符串输出函数）。

**#include< stdio.h >或#include “ stdio.h ”**

## 6.2.1 格式输出函数

`printf()` 函数是格式控制输出函数。`printf()` 函数使用的一般格式为：

```
printf("格式控制字符", 输出项列表);
```

例如：

```
printf("a=%d, b=%f, c=%c\n", a, b, c);
```

## 1. printf( )函数的格式控制字符

“格式控制字符”是用双引号括起的一串字符，包括格式说明、普通字符和转义字符3种。格式控制字符的功能是指定输出数据的格式和类型。

。

格式说明符由%和格式字符组成。

### (1)格式字符

**d**格式字符，输出十制整数

**x**（或**X**）格式字符，输出十六进制整数。

**o**（或**O**）格式字符，输出八进制整数。

**U**格式字符，输出不带符号的十进制整数。

**c**格式字符，输出单个字符。

**s**格式字符，输出字符串。

**f**格式字符，输出十进制单、双精度数。

**e**（或**E**）格式字符，输出指数形式的实型数。

**g**（或**G**）格式字符，输出指数形式的实型数不带无效0的浮点数。

**%**格式字符，输出一个百分号。



**(2)转义字符**

**(3)普通字符**

例如：

```
int x=3, y=6, z=0;
```

```
printf("x=%d,y=%d,z=%d",x,y,z);
```

输出的结果为： **x=3, y=6, z=0**

# 输出项列表有以下几点说明：

输出列表：

(1) 输出项列表的数据可以有一个或多个，也可以没有。**printf( )**函数允许没有输出项列表部分，这表示输出一个字符串。

(2) 输出项列表可以是多个输出项，各个输出项之间用逗号“,”分隔。

(3) **printf( )**函数中格式控制部分的“格式说明符”和“输出项列表”在个数和类型上必须一一对应。

(4) 输出项列表可以由变量、常量或表达式组成。如果是常量，直接常量代替“格式控制字符”里面的“格式字符”；如果是变量，则用变量里面存储的值来取代“格式控制字符”里面的“格式字符”；如果是表达式，则先对表达式进行运算，然后用它的运算结果取代“格式控制字符”里面的“格式字符”。

## 6.2.2 格式输入函数

**scanf( )**函数是格式控制输入函数，**scanf( )**函数使用的一般格式为：

**scanf("格式控制字符",输入项列表);**

**scanf( )**函数有“格式控制字符”，“输入项列表”两个参数，**scanf( )**函数的功能是按照“格式控制字符”中规定的输入格式，从键盘上读取若干个数据，按“输入项列表”中变量从左向右的顺序，依次存入对应的变量中。

**scanf( )**函数的格式控制字符基本与**printf( )**函数的格式控制相似，都是以 % 开始，以一个格式字符结束，格式控制的功能是规定输入数据的格式。

## 6.2.2 格式输入函数

关于**scanf( )**函数，作如下几点说明：

(1) 在**scanf( )**函数中一个格式说明要求输入一个数据，就必须在地址列表中有一个变量的地址与之对应，并且类型要一致。

**scanf("%d%f",&a,&b);**

(2) **scanf( )**函数的输入项必须是一个“地址”，它可以是一个变量的地址，也可以是数组的首地址，但不能是变量名。

**&a、&b**分别表示变量**a、b**存储单元的地址。

## 6.2.2 格式输入函数

(3) **scanf( )**函数格式说明符与**printf( )**函数相似，格式说明符与输入项列表一一对应。常用到的格式说明符有以下几种：

- D**: 输入有符号的十进制整数；
- o**: 输入无符号的八进制整数；
- u**: 输入无符号的十进制整数；
- x**或**X**: 输入无符号的十六进制整数；
- c**: 输入单个字符；
- s**: 输入字符串；
- f**: 输入单精度或双精度数。

## 6.2.2 格式输入函数

(4) 格式说明符中有普通字符或转义字符时，数据输入时，必须按它们的原样输入。

(5) “格式控制字符”中如果没有任何普通字符，输入数据时，各个数据之间可以用空格、跳格键(**Tab**)或回车键作为间隔符。

(6) 可以指定输入数据所占的列数，系统自动按指定的列数截取所需的数据。

**【例6-5】**指定输入数据列数。

(7) 输入数据时不能规定精度。

## 6.2.2 格式输入函数

(8) scanf() 函数在进行输入的时候是没有提示的。

### 【例6-7】

(9) 如果输入的数据多于scanf() 函数所要求的个数，余下的数据可以为下一个scanf() 函数接着使用。

### 【例6-8】

(10) 在scanf() 函数中某格式字符读入数据时，遇以下情况时认为该数据结束：

- \*遇“数据分隔符”如，空格、回车、tab或指定字符。

- \*遇宽度结束，如“%3d”，只取3列后读取结束。

- \*遇非法输入。

### 【例6-9】

## 6.2.2 格式输入函数

(11) 需要注意的是，在使用格式说明符%c输入一个字符时，凡是从键盘输入的字符，包括空格、回车等都被作为有效字符接收。

### 【例6-10】

(12) 如果在%后面有个“\*” 附加说明符，表示跳过它指定的列数。

### 【例6-11】



## 6.2.3 字符输入与字符输出函数

当用**scanf( )**函数和**printf( )**函数对字符型数据进行输入与输出时，格式说明符要使用“%c”，在C语言中对字符型数据还专门提供了字符输入函数（**getchar( )**）与字符输出函数（**putchar( )**）。

### 1. **getchar( )**函数

**getchar( )**函数是字符输入函数，它的功能是从键盘上输入一个字符。一般形式为：

**getchar( )**

### 2. **putchar( )**函数

**putchar( )**函数为字符输出函数，它的作用是在显示器上输出一个字符。其一般形式为：

**putchar(参数);**

## 6.3 整型数据的输入与输出

整型数据的输入与输出通常使用**printf( )**、**scanf( )**函数，函数参数中格式控制符使用整型数据的格式说明符，输入或输出列表使用整型常量、整型变量或整型表达式。

## 6.3.1 整型数据的输出

整型数据格式说明符有以下几种：

**d**：有符号的十进制整数；

**o**：无符号的八进制整数；

**u**：无符号的十进制整数；

**x**或**X**：无符号的十六进制整数。

## 6.3.1 整型数据的输出

(1) **d**格式符，表示十进制整数(有符号数)

用法: **%d**、**%ld**、**%md**、**%lmd**

(2) **o**格式符，表示八进制无符号整数(**o**必须小写)，没有表示八进制的前导**0**。

用法: **%o**、**%lo**、**%mo**、**%lmo**

说明: **%lo**表示八进制长整型，**m**指定输出字符的宽度。

(3) **X|x**格式符，表示十六进制整数，没有表示十六进制的前导**0x**。

用法: **%x**、**%X**、**%lx**

(4) **u**格式符，表示无符号的十进制整数

用法: **%u**

## 6.3.2 整型数据的输入

整型数据的输入使用**scanf( )**函数，函数参数使用整型数据格式说明符。如在**d**、**o**、**x**等。但通常情况下，整型数据格式说明符一般使用**d**，便于从键盘上面输入。

“**%d%d%d**”表示按十进制整数格式输入数据，两个数据之间可以用一个或多个空格、回车键**Enter**或跳格键**Tab**分隔都是合法的，但不能用“**;**”，或其它不符合其格式的间隔符。只有当格式为“**%d,%d,%d**”时，才能用“**,**”作为间隔，即要求输入数据的格式必须与“格式控制”的格式完全一致。

## 6.4 浮点型数据的输入与输出

浮点型数据的输入与输出同样用格式输入输出函数 **scanf( )**、**printf( )**。

浮点型数据的输出格式控符有以下几种：

(1) **f**格式符，表示实型（单精度、双精度），即以小数形式输出。

用法：**%f**、**%mf**、**%.nf**、**%m.nf**

## 6.4 浮点型数据的输入与输出

(2) **e** (或**E**) 格式符, 使用指数格式表示实型 (单精度、双精度) 数据。

用法: **%e**、**%me**、**%m.ne**、**%.ne**

(3) **g** 格式符, 用来输出实型 (单精度、双精度), 根据数值的大小自动选择占用宽度最小的一种, 不输出无意的零。

用法: **%g**

## 第7章 顺序结构与选择结构

**C**语言程序设计总体上包括两个方面的内容：数据定义和数据操作，数据定义是指程序中的数据描述语句，用来定义一系列数据的类型，完成数据的初始化等；数据操作是指程序中的操作控制语句，用来控制程序的执行过程，一般程序的执行结构包括三种：顺序结构、选择结构和循环结构。前几章我们学习了数据定义方面的有关内容，本章将重点介绍**C**语言程序设计的数据操作。本章内容：

顺序结构程序设计；

选择结构程序设计；

应用举例。



## 7.1 顺序结构程序设计

顺序结构程序是最简单，最基本的程序设计，它由简单的语句组成，程序的执行是按照程序员书写的顺序进行的，没有分支、转移、循环，且每条语句都将被执行。顺序结构的程序是从上到下依次执行的，其执行流程如图所示。



图 顺序结构执行流程

## 7.2 选择结构程序设计

由于顺序结构程序是顺序执行的，无分支、无转移、无循环，因此它不可能处理复杂的问题，而在数据处理过程中，通常需要根据不同的条件进行判断，然后选择程序进行处理，由此可见，顺序结构无法满足要求，而选择结构就是为了解决这类问题而设定的。

一般而言，**C**语言中选择语句包括两种：**if**语句和**switch**语句。所谓选择语句就是通过判断条件来选择执行哪一条语句，进而达到编程目的。

## 7.2.1 if语句

**if**语句又称为条件语句，可以实现多路分支。**C**语言中，**if**语句一般格式如下：

```
if(条件 1)↵  
{↵  
    → 语句 1↵  
}↵  
else if(条件 2)↵  
{↵  
    → 语句 2↵  
}↵  
else if(条件 3)↵  
{↵  
    → 语句 3↵  
}↵  
...↵  
else if(条件 m)↵  
{↵  
    → 语句 m↵  
}↵  
else↵  
{↵  
    → 语句 m+1↵  
}↵
```

## 7.2.1 if语句

### 1. 常用的if语句格式

通常在运用的过程中，**if**分支语句有几种常用的格式：

格式一：

**if**语句最简单的格式是没有**else**，只有**if**关键字。格式如下：

**if** <条件> 语句

格式二：

程序中应用最多的**if**语句是两路分支，它的基本格式如下：

```
if(条件 1){  
    → 语句 1  
}  
else  
{  
    → 语句 2  
}
```

## 7.2.1 if语句

### 2. if语句的嵌套

在if语句中出现的执行语句既可以是一条语句也可以是复合语句，在复合语句再次出现if语句就构成了if语句的嵌套。格式如下：

```
if(条件 1)↵  
{↵  
→ if(条件 2)↵  
→ {↵  
→ → 语句 1↵  
→ }↵  
→ else↵  
→ {↵  
→ → 语句 2↵  
→ }↵  
}↵  
else↵  
{↵  
→ if(条件 3)↵  
→ {↵  
→ → 语句 3↵  
→ }↵  
→ else↵  
→ {↵  
→ → 语句 4↵  
→ }↵  
}↵
```

## 7.2.2 switch语句

**if**语句一般用于处理一个或两个分支的选择结构，如果分支较多时，就需要使用**if**语句的嵌套，但嵌套的**if**语句层数越多，程序越复杂，可读性就越差。**C**语句提供的**switch**语句能同时处理多个分支选择结构。其语法格式为：

```
switch(表达式)
{
case 常量1: 语句组1
case 常量2: 语句组2
...
case 常量n: 语句组n
default: 语句组n+1
}
```

## 7.2.2 switch语句

这里要说明的是：

(1) switch后面括号内的“表达式”可以是任何类型的数据。可以是整形表达式、字符型表达式，也可以枚举类型数据。

(2) 每个case的常量表达式的值必须互不相同，否则会产生错误的选择。

(3) 各个case和default的出现次序不影响执行的结果。

(4) 在执行switch语句时，根据switch后面表示式的值找到匹配的入口标号，执行完该case语句后，继续执行下一个case语句，不再进行标号判断。case常量表达式只起到入口标示的作用。

**【例7-8】**

## 7.3 应用举例

本章学习结构化程序设计中的顺序结构和选择结构，在讲解的同时也给出了一些实例，为了加深读者对本章内容的认识，本节将给出一些实例，请读者结合前面所学的知识进行分析。

**【例7-10】** 给一个不多于5位的正整数，求它是几位数，并逆序打印出各位数字。

**【例7-11】** 从键盘上输入若干个学生的成绩，统计并输出最高成绩和最低成绩，当输入负数时结束输入。



## 7.3 应用举例

**【例7-13】**企业发放的奖金根据利润提成。利润(I)低于或等于10万元时，奖金可提10%；利润高于10万元，低于20万元时，低于10万元的部分按10%提成，高于10万元的部分，可提成7.5%；20万到40万之间时，高于20万元的部分，可提成5%；40万到60万之间时高于40万元的部分，可提成3%；60万到100万之间时，高于60万元的部分，可提成1.5%，高于100万元时，超过100万元的部分按1%提成，从键盘输入当月利润I，求应发放奖金总数。

## 第8章 循环结构程序设计

许多问题都会遇到规律性的重复操作。例如，求和问题、求有一定规律的问题和一些迭代等问题都会用到循环结构。循环结构是结构化程序设计的基本结构之一，它与顺序结构、选择结构共同构成了作为各种复杂程序的基本构造单元。本章重点讲解3种循环语句：**while**语句、**do...while**语句和**for**语句。本章内容如下：

**while**语句；

**do...while**语句；

**for**语句；

总结应用。

## 8.1 while语句

**while**语句的一般形式如下：

**while**(表达式)

{

循环语句

}

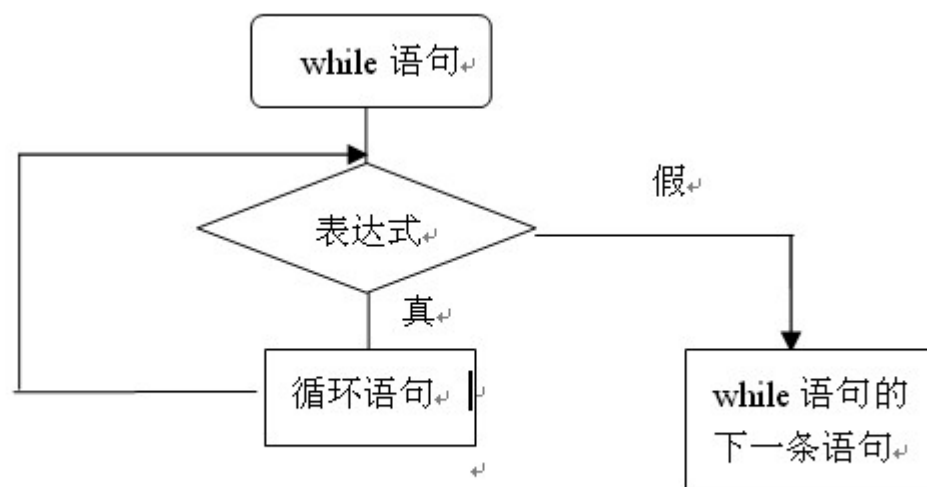


图 8-1 while 语句执行流程图

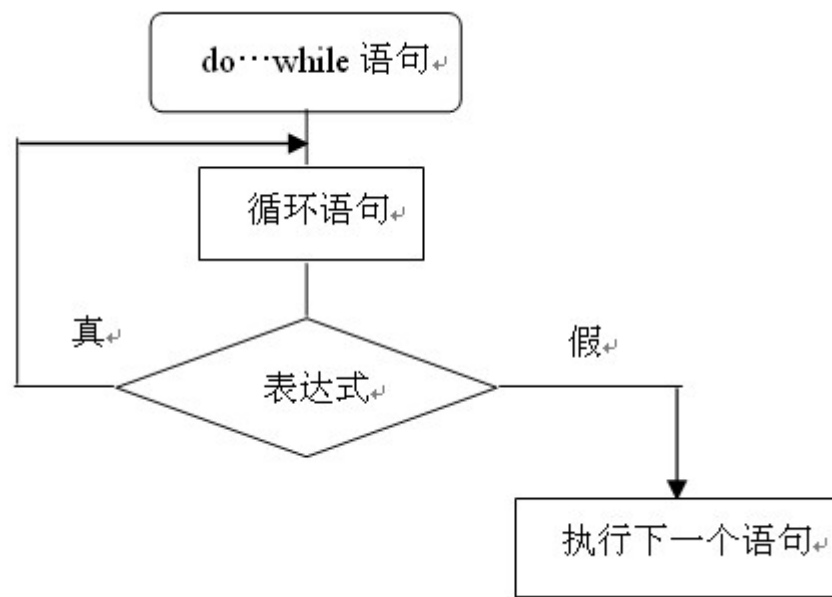
## 8.2 do...while语句

**do...while**语句是另一种循环结构，它和**while**语句的不同是，**do...while**语句先执行循环，然后判断循环条件是否成立。其一般形式如下：

**do**

循环语句

**while** （表达式）；



## 8.2 do...while语句

对于**do...while**语句作以下几点说明：

在**C**语言中**do...while**语句被称为直到型循环，它的执行特点是：先执行，后判断。

**do...while**语句执行过程：（1）先执行一次循环语句，然后判断表达式的值。（2）若表达式的值为非**0**，再返回（1）。如果表达式的值为**0**，则直接退出循环语句，执行下一条语句。

【例8-5】从键盘输入一个正整数**n**，计算该数的各位数之和并输出。例如，输入数是**4569**，则计算：**4+5+6+9=24**并输出。

## While 与do...while比较

```
#include <stdio.h>
void main()
{
    int i;
    float sum;
    sum=0;
    scanf("%d",&i);
    while(i<=20)
    {
        sum+=1/(float)i;
        i+=1;
    }
    printf("sum=%f",sum);
}
```

```
#include <stdio.h>
void main()
{
    int i;
    float sum;
    sum=0;
    scanf("%d",&i);
    do
    {
        sum+=1/(float)i;
        i+=1;
    }while(i<=20);
    printf("sum=%f",sum);
}
```

## 8.3 for语句

**for**语句是控制重复的指令，在表达式为真时，重复执行语句。**for**语句是C语言中应用最频繁的语句，**for**循环语句，可以用于循环次数已经确定的情况，也可以用哦关于循环次数不确定的情况。可以完全代替**while**以及**do...while**语句。

## 8.3.1 for循环结构

**for**语句的一般形式为：

**for**（表达式1;表达式2;表达式3）  
语句

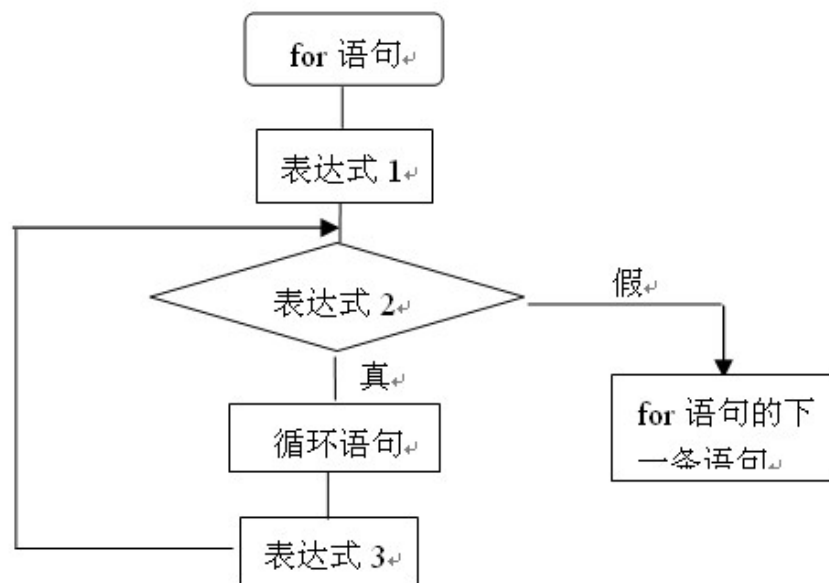


图 8-7 for 语句执行流程图



for语句的作用可以用while语句描述如下

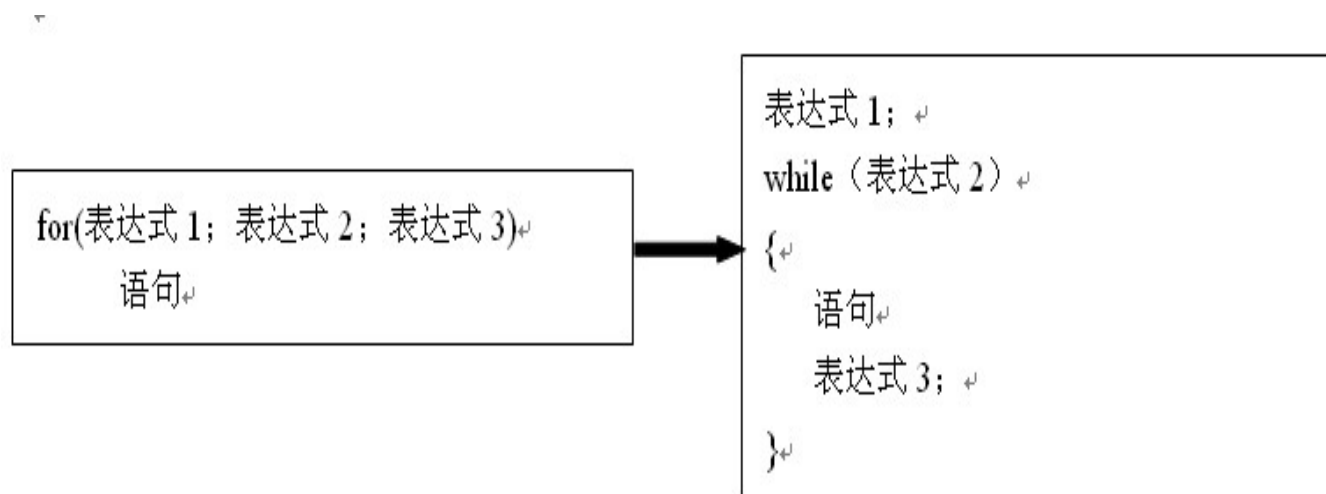


图 8-8 for 语句和 while 语句的转换

## 8.4 总结应用

从前面几节上中我们了解的循环结构程序设计的三种重要的语句结构，三种结构形式不同，判断条件不同，在解决时也有差异。

## 8.4.1 几种循环的比较

(1) 4种循环都可以处理同一问题，一般情况下可以互换，但是尽量不用**goto**型。

(2) **while** 和 **do-while** 循环，在**while**后面只指定循环的条件，而使循环趋于结束的语句包含在循环体中。而**for** 循环可以在“表达式3”中包含使循环趋于结束的操作。甚至可将循环体放到“表达式3”中。

## 8.4.1 几种循环的比较

用**while**和**do-while**循环时，循环变量初始化应在**while** 和 **do-while**之前；而**for** 循环可以在“表达式1”中实现循环变量的初始化。

**while**循环：先判断后执行；**do-while**循环：先执行后判断；**for**循环：先判断后执行。

对 **while**循环、**do-while**循环、**for**循环，用 **break** 语句跳出循环，可以用**continue**语句结束本次循环。对于**goto**语句构成的循环,不能如此操作。

## 8.4.2 循环语句的嵌套

(1) **while**与**do-while**嵌套。格式为：

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

## 8.4.2 循环语句的嵌套

(2) **while**与**for**嵌套。格式为：

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

## 8.4.2 循环语句的嵌套

(3) **do-while**与**for**嵌套。格式为：

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

对于循环嵌套需要说明以下几点：

(1) 使用嵌套时，一个循环结构应完整地嵌套在另一个循环体内，循环体间不能交叉。

(2) 嵌套的外循环与内循环的循环控制变量不能同名，并列的内、外循环控制变量可以同名。

```
for (i=1; i<=9; i++)  
    for (j=0; j<=9; j++)  
        for (k=0; k<=9; k++)  
        {  
            a=i*100+j*10+k ;  
            if(a==i*i*i+j*j*j+k*k*k)  
                printf("%d\n", a) ;  
        }
```



## 8.4.3 应用举例

本章学习了三种循环结构语句，并对它们作了一定的区分，以及讲解三个循环结构自身的嵌套和它们之间的相互嵌套，下面的一些例子巩固对本章知识的理解。

【例8-16】判断一个数是否是素数。

【例8-18】用“辗转相除法”对数入的两个正整数m和n求其最大公约数和最小公倍数。

【例8-19】猴子吃桃子问题。猴子第一天摘下若干个桃子，当即吃了一半，还不过瘾，又多吃了一个。第二天早上又将剩下的桃子吃掉一半，又多吃了一个。以后每天早上都吃了昨天的一半零一个。到第10天早上一看，只剩下一个桃子了。求第一天共摘下多少个桃子。

程序分析：

设今天的桃子数为y，昨天的桃子数为x

则有： $y = x - (x/2 + 1)$

$x = 2 * (y + 1)$

从第10天 $y=1$ 起求出x，把x又当成今天（ $y=x$ ）求昨天（x），这样向前推9天，即为第一天的桃子数。

## 第9章 结构语句的转移

结构化程序设计包括顺序结构、选择结构和循环结构，通过跳转语句与结构语句的结构使用，能够用于解决复杂的问题。

**break**语句

**continue**语句

**goto**语句

## 9.1 break语句

**break**语句的作用就是跳出循环语句，然后执行这一循环语句的下一语句。前面介绍的循环语句都是在执行循环语句时，通过对一个表达式的测定来决定是否终止对循环体的执行。

**break**语句的格式如下：

**break;**

## 9.1 跳出switch结构

在前面的选择结构中，我们知道**switch-case**语句的执行流程是：首先计算**switch**后面圆括号中表达式的值，然后用此值依次与各个**case**的常量表达式比较，依次的执行下去。如果这样的话就得不到我们想要的结果。我们想要的是若圆括号中表达式的值与某个**case**后面的常量表达式相等，就执行此**case**后面的语句，然后就跳**switch-case**语句，不执行其它**case**后面的常量表达式，因此这里需要用到**break**语句

## 9.2 跳出循环结构

**break**语句可以用来从循环体内跳出，**break**语句用于强制结束执行的程序。

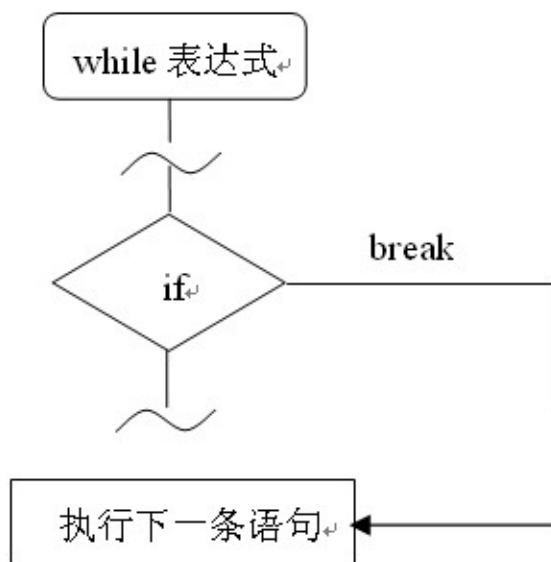


图 9-1 while 语句中 break 跳转流程图

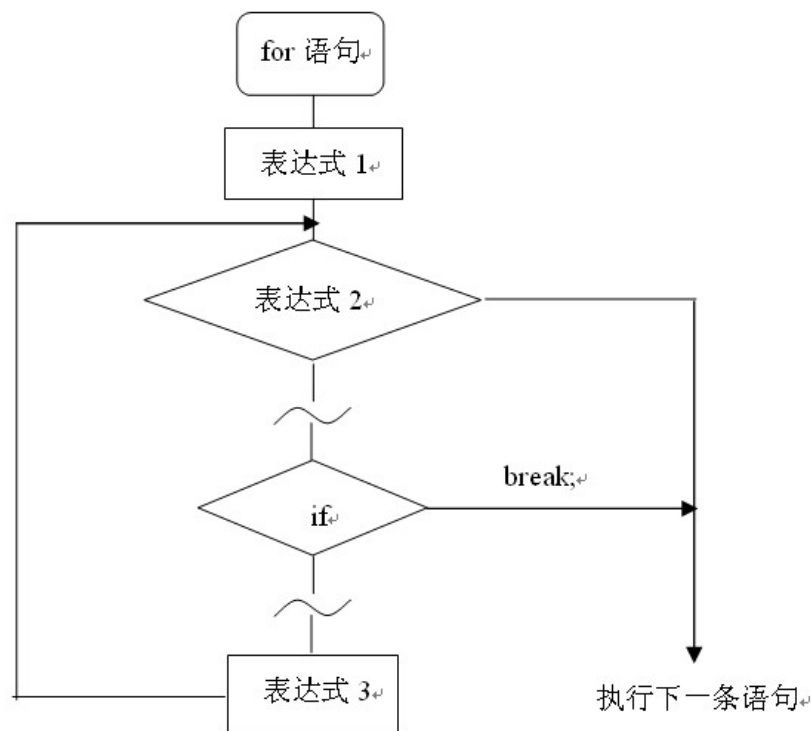


图 9-2 for 语句中 break 跳转流程图

## 9. 2. 1 问题1

```
while(1)
{
...
if(...)
{
break;
}
...
}
```

## 9.2.2 问题2

```
for(...)
{
  for(...)
  {
    ...
    break;
  }
  ... <----- ①
}
```

## 9.2 continue语句

**continue**语句也是用于循环控制的语句，**break**语句是中断循环并从循环体跳出，而**continue**语句则是中断循环体后返回循环的开头。即跳过循环体中**continue**下面的语句，重新执行循环体。**continue**语句的一般形式如下：

**continue;**

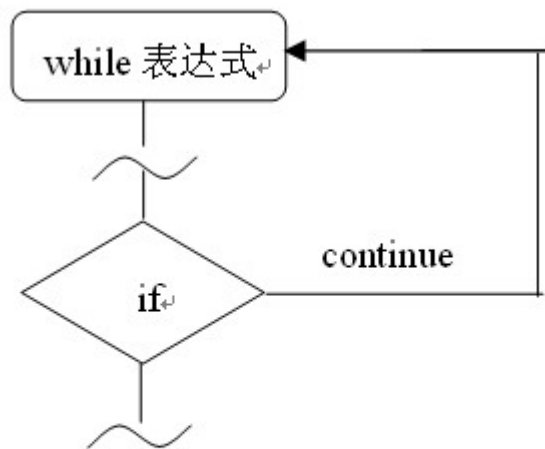


图 9-4 while 语句中 continue 跳转流程图

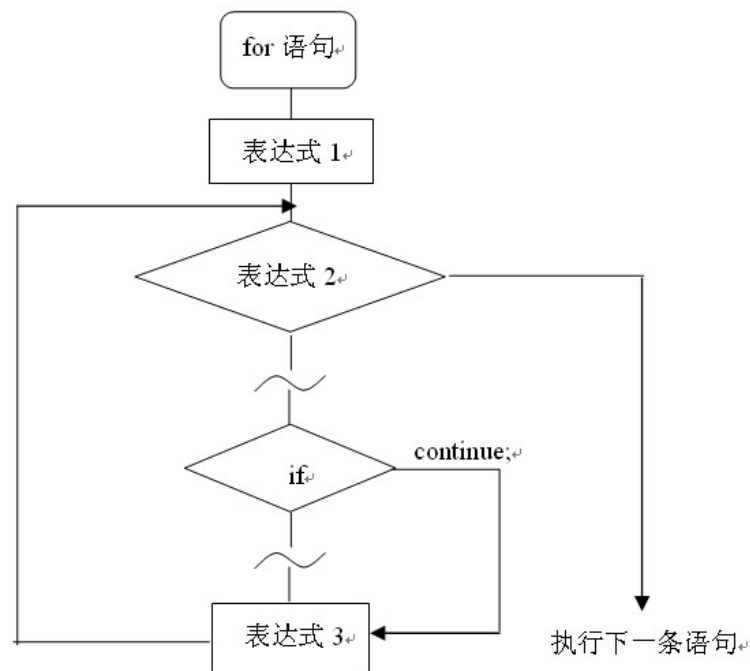


图 9-5 f 语句中 continue 跳转流程图



## 9.3 goto语句

在**basic**等语言中有无条件转移**goto**语句，**C**语言中也有无条件转移指令**goto**语句。**goto**语句只在一个函数内有效。**goto**语句最好不要在循环语句中用来跳出循环。

**goto**语句的格式如下：

**goto** 标号；

...

标号：语句

# 第10章 数组

数组是在一个变量名之下存放的多个数据的存储区的说明，是具有相同类型的数的集合按照一定的顺序组成的数据。

本章内容：

数组的概述

一维数组

二维数组

## 10.1 数组的概述

数列和矩阵是用来描述一批数据之间的关系的。如：  
表示x数列时通常写成：

$x_1, x_2, x_3, \dots, x_n$

表示一个 $2 \times 3$ 矩阵y可以写成：

$y_{11} \ y_{12} \ y_{13}$

$y_{21} \ y_{22} \ y_{23}$

分析数列和矩阵不难发现它们有三个特点：

有一批数据；

这些数据之间有一定的内在联系；

这些数据的类型相同。

## 10.2.1 一维数组的定义

一维数组的定义方式:

类型说明符 数组名 [常量表达式];

例如:

```
int age[10];
```

## 10.2.2 一维数组的初始化

在定义数组时给元素赋初值。一般语法为：  
类型符 数组名[元素个数]={常量表};例如：

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

说明：

(1) 若给所有元素赋初值，“元素个数”可以省略。如：int a[] = {1, 2, 3, 4, 5};

(2) 可以只给一部分元素赋初值，但元素个数不能省略。未被赋初值的元素则为0。如：int a[5] = {1, 2, 3};。此时a[0]=1，a[1]=2，a[2]=3，a[3]=0，a[4]=0。

(3) 若使全部元素都为0，可以将其定义为“全局变量”或“静态变量”，也可以写成：int a[5] = {0}; a[5] = {0, 0, 0, 0, 0}

(4) 若数组在定义时未进行初始化，则各元素的值是随机的。如：int a[5];

(5) 上述数组初始化的方法都是在定义时进行的，用赋值语句是错的。

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

```
a[5] = {1, 2, 3, 4, 5}    /*错*/
```

(6) 若初始化时，赋值的数据多于数组的个数。例如：

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

## 10.2.2 一维数组的初始化

对一维数组做以下几点说明：

(1) 在一条定义语句中，可以定义多个数组或变量，但存储类型和数据类型应相同。

(2) 在一条定义语句中，可以给全部数组元素赋初值，也可以只给其中某些数组元素赋初值，没有赋初值的元素均自动获得初值，这个初值为空值，即数值0，或字符串结束标志符‘\0’。

(3) 如果数组中所有元素均赋初值，数组长度可以省略，此时数组的长度等于初值表中初值的数目。如果只给部分数组元素赋初值，此时数组长度不能省略。

## 10.2.3 一维数组的引用

数组必须先定义后引用，在定义了数组变量后，我们就可以引用其中的每个元素了。一维数组的引用格式如下：

数组名[下标表达式]

在引用数组元素时，常用的形式是：a[i]

若i=0，a[i]——>a[0]

若i=1，a[i]——>a[1]

引用数组所有的元素成为遍历，遍历数组是通过循环来改变下表进行的。例如：

读入：for(i=0;i<5;i++)     scanf("%d",&a[i]);



数组元素的地址

&数组名[i]或数组名+i

数组首地址

数组名或&数组名[0]

例如：

int x[5];      /\*定义一个整型数组x，含有5个数组元素\*/

x              /\*表示数组的首地址\*/

&x[0]            /\*表示数组的首地址\*/

&x[i]            /\*表示数组元素x[i]地址即第i+1元素的地址\*/

x+i              /\*表示数组元素x[i]地址即第i+1元素的地址\*/

## 10.2.4 一维数组的程序举例

**【例10-5】** 输入10个整数存入一维数组，再按逆序重新存放后再输出。

**【例10-6】** 用选择法对10个整数排序（从小到大）。

**【例10-8】** 用筛选法求100以内的素数。

## 10.3 二维数组

二维数组其实就是一维数组的一种转换形式，我们有时候为了做题的方便会把二维数组看成一维数组来处理。

## 10.3.1 二维数组的定义

二维数组定义的一般形式为：

类型符 数组名[行数][列数];

例如：

`int b[2][3];` 定义了一个 $2 \times 3$ 的整型数组b，它有2行、3列共6个元素。这6个元素为：

<code>b[0][0]</code>	<code>b[0][1]</code>	<code>b[0][2]</code>
<code>b[1][0]</code>	<code>b[1][1]</code>	<code>b[1][2]</code>

一个二维数组可以看作一种特殊的一维数组， 它的元素  
又是一个一维数组。 例如：

```
int  a[3] [4] ;
```

其中：

$a[0] \rightarrow a[0][0] , a[0][1] , a[0][2] , a[0][3]$

$a[1] \rightarrow a[1][0] , a[1][1] , a[1][2] , a[1][3]$

$a[2] \rightarrow a[2][0] , a[2][1] , a[2][2] , a[2][3]$

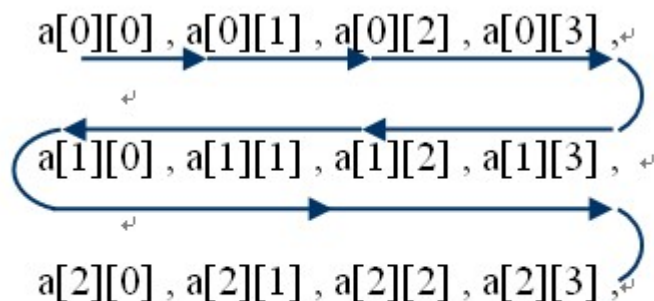


图 10-6 二维数组排列方式

## 10.3.2 二维数组的初始化

在定义二维数组时给元素赋初值，有几种方法。下面我们来一一介绍：

(1) 可以像一维数组一样，将所有的元素的初值写在一对花括号内。这样，编译系统将按行的顺序依次为各元素赋初值。一般语法为：

类型符 数组名[行数][列数]={常量表};

如：int a[2][3]={1, 2, 3, 4, 5, 6};

也可以部分赋值：int a[2][3]={1, 2, 3};

(2) 可以对数组按行的顺序排列赋值，每行都用一对花括号括起来，各行之间用逗号隔开。这种形式界限清楚，可读性强。

如：int a[2][3]={ {1, 2, 3}, {4, 5, 6} };

也可以部分赋值：int a[2][3]={ {1, 2}, {4} };

(3) 若给所有元素赋初值，“行数”可以省略，但“列数”不能省。

如： `int a[ ][3]={1, 2, 3, 4, 5, 6};`

(4) 若数组在定义时未进行初始化，则各元素的值是随机的。

## 10.3.3 二维数组的引用

二维数组的数组元素的表示形式如下：

数组名[行下标][列下标]；

可以对数组元素进行各种基本数据类型变量所能进行的各种操作。

```
int a[2][3];
```

```
a[1][0]=5;
```

```
a[1][1]=a[1][0]*2;
```

```
a[1][2]=a[1][0]+a[1][1];
```



二维数组的首地址以及数组元素的地址只能用下面的方式获得：

(1) 二维数组元素首地址：

数组名 或 数组名[0] 或 &数组名[0][0]

(2) 二维数组第i行元素组成的一维数组首地址：

数组名[i] 或 数组名+i (i表示二维数组第i行)

(3) 二维数组数组元素的地址：

&数组名[i][j] 或 数组名[i]+j (i表示二维数组第i行, j表示i表示二维数组第j列)

例如：

a[m][n]           /\*含有m行n列的二维数组\*/

a, a[0], &a[0][0]       /\*表示二维数组a的首地址\*/

a[i], a+i           /\*表示二维数组第i行数组元素的首地址\*/

&a[i][j], a[i]+j       /\*表示二维数组元素a[i][j]的地址\*/

例如：将二行三列的矩阵存入二维数组a[2][3]中。按行：

```
for(i=0;i<2;i++)          /*外层循环控制行数*/
{
    for(j=0;j<3;j++)      /*内层循环控制列数*/
    {
        scanf("%d",&a[i][j]);
    }
}
```

按列：

```
for(i=0;i<3;i++)          /*外层循环控制列数*/
{
    for(j=0;j<2;j++)      /*内层循环控制行数*/
    {
        scanf("%d",&a[j][i]);
    }
}
```

### 10.3.4 二维数组的程序举例

**【例10-12】**输入20个数，并以每行5个数据的形式输出a数组。

### 【例10-13】从键盘输入年月日，计算该日是该年的第几天。

**【例10-14】** 给一个 $10 \times 10$ 的矩阵，偶数行的方阵中所有边上的元素置1，两对角线上的元素置1，其它元素置0。要求对每个元素只置一次值。最后按矩阵形式输出。

[illegible]

# 第11章 字符数组

用来存放字符数据的数组是字符数组，常用来处理字符串。字符数组其实就是类型为**char**的数组。同其他的数组类型一样，字符数组既可以是一维的，也可以是二维，甚至多维的。本章内容如下：

- 字符数组的定义；

- 字符数组的初始化；

- 字符数组的引用；

- 字符数组与字符串的关系；

- 字符数组的输入与输出；

- 字符串处理函数；

- 应用举例。

## 11.1 字符数组的定义

一维字符数组的定义方式如下：

```
char 数组名[常量表达式];
```

例如：char c[5];

二维数组的定义方式如下：

```
char 数组名[常量表达式][常量表达式];
```

例如：char c[5][5];

在字符数组中，我们只能每个元素存放一个字符数据。例如：

```
char c[5];
```

```
c[2]='a';
```

字符型和整型是相互通用的，所以，可以定义一个整型数组来存放字符型数据。例如：

```
int c[5];
```

```
c[2]='a';
```

## 11.2 字符数组的初始化

字符数组的初始化，即对字符数组中的每个元素赋值。  
例如：

```
char c[10]={'I',' ','a','m',' ','h','a','p','p','y'};
```

把这10个字符数组元素分别赋值为：**c[0]='I'**，**c[1]=' '**，**c[2]='a'**，**c[3]='m'**，**c[4]=' '**，**c[5]='h'**，**c[6]='a'**，**c[7]='p'**，**c[8]='p'**，**c[9]='y'**。

## 11.3 字符数组的引用

通过引用字符数组中的元素，进行程序设计跟前面所讲的数组引用相似。

**【例11-1】**输入一个由5个字符组成的单词，将其内容颠倒过来，并输出。

## 11.4 字符数组与字符串的关系

在C语言中，字符串使用双引号括起来的字符序列。C语言中，是将字符串用字符数组来处理的。

**【例11-2】**编写一程序，用于合并两个已知的字符数组。



**C**语言允许用一个简单的字符串常量初始化一个字符数组，而不必使用一串单个字符。例如：

```
char str[ ]={"good"};
```

可以省略{ }，直接用双引号。

**g o o d \0**

字符数组在编译时，自动在字符串的末尾加上了一个特殊字符‘\0’。所以，字符数组的个数是**5**。

```
char str1[ ]={"good"};
```

```
char str2[ ]={'g','o','o','d'};
```

## 11.5 字符数组的输入与输出

由于字符串放在字符数组中，所以，对字符串的输出也就是对字符数组的输出。字符数组的输出有两种形式：

(1) 当字符数组中存储的字符不是以‘\0’结束时，只能像普通的数组那样，用格式符“%c”一个元素一元素的处理。

(2) 当用字符数组处理字符串时，可以与“%s”格式字符配合，完成字符串的输入输出。

注意：

(1) 在使用scanf函数输入字符串时，“地址”部分应该直接写字符数组的名字，而不是取地址运算符&。因为在C语言中，数组的名字代表该数组的起始地址。

(2) 在输出字符串时，输出项也为数组名，不能使数组元素。

利用格式符“%s”输入的字符串，以“空格”，“TAB”间隔多个字符串，“回车”结束输入。

(3) 当字符数组长度大于字符串的实际长度时，也只输出到‘\0’时结束。例如：

```
char str[20]="world";  
printf("%s",str);
```

(4) 如果字符数组中包含多个‘\0’，遇到第一个‘\0’时，输出结束。例如：

```
char str[20]="hello\0world";  
printf("%s",str);
```

(5) 用scanf函数“%s”格式输入一个字符串时，函数中输入项用数组名，并且该数组已定义，而且输入字符串的长度应小于数组长度。

例如：

```
char str[10];  
scanf("%s",str);
```

## 11.6 字符串处理函数

C语言中有很多字符串处理的库函数，这些库函数为字符串处理提供了方便。在使用时，要在程序开头将这些字符串库文件包含到程序中。即：**`#include <string.h>`**。

## 11.6.1 输入字符串函数gets

**gets**函数用于输入一个字符串，其调用形式如下：

**gets(字符数组);**

将一个字符串存放到字符数组中，并且得到一个函数值。即，**gets**函数的返回值是存放输入字符串的字符数组的起始地址。例如：

**char str[ 20];**

**gets(str);**

其中，**str**是字符数组，输入的字符串存放在字符数组**str**中。

## 11.6.2 输出字符串函数puts

puts函数用于输出一个以' \0 ' 结尾的字符串，其调用形式如下：

```
puts(字符数组);
```

例如：

```
char str[]="world";
```

```
puts (str);
```

puts函数输出的字符串可以包含转义字符。

```
char str[]="hello\nworld";
```

```
puts (str);
```

输出结果为：

```
hello
```

```
world
```

## 11.6.3 字符串测长度函数strlen

**strlen**函数用于测试字符串的长度。函数值为实际的字符串的长度，不包括'\0'所占的位置。其调用形式如下：

**strlen**(字符数组);

例如：

**char str[10]={"hello"};**

**printf("%d",strlen(str));**

输出结果为：**5**。**strlen**测试的是实际的字符串的长度，所以结果既不是**6**，更不是**10**。

**strlen**函数还可以直接测试字符串常量的长度，例如：

**strlen("hello");**

## 11.6.4 字符串比较函数strcmp

在C语言中，不允许用下列形式比较字符串。

```
if(str1==str2)
{
    printf("相等");
}
```

字符串比较不能用关系运算符。只能用strcmp函数，比较结果由strcmp函数返回。其调用形式如下：

```
strcmp(字符串1, 字符串2)
```



## 11.6.4 字符串比较函数strcmp

完成“字符串1”与“字符串2”的关系比较，即对两个字符串自左至右逐个字符按其ASCII码值相比，直到出现不同的字符或遇到'\0'为止。比较的结果由函数值获得：

```
char str1[20],str2[20];
```

```
int n;
```

```
n=strcmp(str1,str2);
```

如果"字符串1"="字符串2"，函数值n为0；

如果"字符串1">"字符串2"，函数值n为一正数；

如果"字符串1"<"字符串2"，函数值n为一负数。

## 11.6.5 字符串复制函数strcpy和strncpy

C语言中，不允许用赋值语句直接将一个字符串赋给另一个字符数组。即：

```
str1="hello";
```

```
str2=str1;
```

字符串复制必须使用**strcpy**函数。字符串复制函数，是将字符串一个字符一个字符的复制，直到遇到'\0'字符为止。其中，对'\0'字符也一起复制。其调用形式如下：

```
strcpy(字符数组1,字符数组2)
```

在将字符数组2复制到字符数组1中时，字符数组1的空间必须足够大。

## 11.6.5 字符串复制函数strcpy和strncpy

注意：

字符数组1必须是字符数组名的形式。

字符串2可以是字符数组名或字符串常量。

**strncpy**函数是将字符串2中前n个字符复制到字符数组1中。

如果需要复制字符串2中前面的若干个字符，则可指出需要复制的字符数。其调用形式如下：

**strncpy**(字符数组1,字符串2,字符数)

**strncpy**(str1, str2, 2) ;

把str2中头上二个字符复制到str1中去，str1再加一个结束‘\0’。

## 11.6.6 字符串连接函数strcat

**strcat**函数用于连接两个以'\0'结尾的字符串，其调用形式如下：

**strcat(字符数组1,字符数组2)**

例如：

**char str1[20]="how ";**

**char str2[20]="are you?";**

**strcat(str1,str2);**

运行后，输出：**how are you?**

## 11.7 应用举例

**【例11-4】** 将一个字符串复制到另一个字符串中，即完成strcpy函数的功能。

**【例11-5】** 编写程序实现输出一个字符串后，将字符串的内容颠倒过来。

## 第12章 函数

我们已经知道一个C语言程序由若干个函数构成，各个函数之间相互独立，那么这些函数是怎样定义，又在整个程序中扮演什么角色呢？下面我们将进入函数这一章节。

函数的初步认识；

函数定义；

函数参数及返回值；

函数的参数传递；

应用举例

## 12.1 函数的初步认识

在一个较大的C程序中，一般有若干个程序模块，每一个模块用来实现一个特定的、比较简单的功能。在所有的高级语言中都有子程序这个概念，用子程序实现模块的功能。在C语言中，用函数来实现子程序的作用。一个C程序由一个主函数（**main**函数）和若干个其他函数组成。在主程序中调用其他函数，其他函数之间也可以相互调用。同一个函数可以被一个或多个函数调用任意多次。当然，在一个函数中也可以调用一个或多个函数。

(1) 函数是依照规定格式编写的，能够完成一定功能的一段程序。

(2) 函数之间是相互独立的，没有从属关系，不能嵌套定义，但是可以相互调用。主函数可以调用任意函数，而其他函数不能调用主函数。主函数是由系统调用的。

(3) **C**程序的执行是从主函数开始的，在主函数中调用其他函数，在主函数中结束程序的运行。



从设计人员的角度来看，函数可以分为两种。

**标准函数：**标准函数即库函数，它是由系统提供的，用户不必自己定义而能够直接使用的。如printf()、scanf函数。需要注意的是，不同的C语言编译系统提供的库函数的数量和功能会有一些不同，但大多数基本函数都是相同的。

**用户自定义函数：**它是用户自己定义的，以实现某种特殊功能的函数，如例12-1中的funct函数。

从函数的形式来分，可以分为两类。

**无参函数：**在调用无参函数时，主调函数不向被调函数传递数据。无参函数一般用来执行一组指定的操作。如例12-1中，printwords函数是无参函数，当此函数被调用时，用来输出“I love China!”这句话。无参函数可以带回或不带回函数值，一般以不带回函数值占多数。

**有参函数：**在调用有参函数时，通过参数向被调用函数传递数据，一般情况下，被调用函数会返回一个函数值，供调用函数使用。如例12-1中，当主函数调用funct函数时，主函数向其传递x值为3，funct()返回函数值9供主函数使用。

## 12.2 函数定义

函数由函数名、参数和函数体组成。函数名是用户为函数定义的名字，用来唯一标识一个函数；函数的参数用来接收调用函数传递给它的数据，在无参函数中没有参数；函数体是函数实现自身功能的一组语句

## 12.2.1 无参函数定义

无参函数定义的一般形式为：

类型表示符 函数名( )

{

[声明与定义部分]

语句部分

}

## 12.2.2 有参函数定义

有参函数定义的一般形式为：

类型表示符 函数名(形式参数声明)

{

[声明与定义部分]

语句部分

}

例如：

```
int sum (int x, int y)
```

```
{
```

```
    int z;
```

/\* 函数体中的声明部分 \*/

```
    z=x+y;
```

```
    return(z);
```

```
}
```

## 12.2.3 空函数定义

空函数的形式为：

类型说明符 函数名( )

{ }

例如：

**void count( )**

**{ }**

## 12.3 函数参数及返回值

函数参数主要用于调用函数与被调用函数之间的数据传递。函数参数包括，实际参数和形式参数。在C语言中，参数类型不同，传递方式也不同。函数调用的目的就是得到一个返回值，同样返回值的类型也各异。

## 12.3.1 函数的参数

在一个C程序中调用函数时，通常情况下，主调用函数和被调用函数之间要有数据的传递。这就是在前面所提到过的有参函数。前面已经提到过：在定义有参函数时，函数名后面括号中的变量名称为“形式参数”（简称“形参”）。在主调用函数中调用一个有参函数时，被调用函数名后面括号中的参数称为“实际参数”（简称“实参”）。

**【例12-3】** 调用函数过程中的数据传递。

```
#include <stdio.h>
void main( )
{
    int funct(int x, int y, int z);    /*声明funct函数*/
    int a, b, c, r;
    scanf("%d, %d, %d", &a, &b, &c);
    r=funct(a, b, c);                /*调用funct函数*/
    printf("The result is %d. \n", r);
}

int funct(int x, int y, int z)        /*定义有参函数funct()*/
{
    int r;
    r=3*x+2*y+z;
    return(r);
}
```



关于形式参数和实际参数的说明：

1) 在定义函数中指定的形式参数，在没有出现函数调用时，它们并不占用内存中的存储单元。只有在发生函数调用时，函数中形式参数才被分配内存单元。在调用结束后，形式参数所占用的内存单元也被释放。

2) 实际参数可以是常量、变量或表达式，

3) 在被定义的函数中，必须指定形式参数的类型。

4) 实际参数和形式参数的类型应相同或赋值兼容。如例12-3中实际参数和形式参数都是整型，这是正确的、合法的。如果实际参数为实型，而形式参数为整型，或者相反，则系统会按照第三章介绍的不同类型数值的赋值规则进行转换。如果实际参数与形式参数类型不同，且不能进行转换，则会出现错误。

5) 在C语言中，实际参数向形式参数的数据传递是“值传递”，单向传递，只由实际参数传递给形式参数，而不能由形式参数传递给实际参数。在内存中，实际参数单元与形式参数单元是不同的单元。 【例12-5】

## 12.3.2 函数的返回值

函数的返回值是通过函数中的return语句获得的。return语句将被调用函数中的一个确定值带回主调用函数中去。如果需要从被调用函数返回一个函数值，被调用函数中必须包含return语句。如果不需要从被调用函数返回函数值，则可以不要return语句。

一个函数中可以有一个以上的return语句，执行到哪一个return语句，哪一个语句起作用。

return语句后面的括号可以不要，如“return(z)”可以写成“return z”。

return后面的值可以是一个表达式。

## 12.4 函数的参数传递

在C语言中进行函数调用时，有两种不同的参数传递方式，即值传递方式和地址传递方式。

## 12.4.1 函数参数的数值传递

在函数调用时，实际参数把它的值传递给形式参数，这种调用方式称作“值传递”。

在C语言中，实际参数向形式参数的数据传递是“值传递”，单向传递，只由实际参数传递给形式参数，而不能由形式参数传递回来给实际参数。

## 12.4.2 函数参数的地址传递

地址传递指的是实际参数将变量的地址传递给形式参数。这样实际参数和形式参数指向同一个内存单元，在调用函数过程中，如果形式参数所指向的内容发生变化，实际参数也会发生变化。

在地址传递方式中，形式参数和实际参数可以是指针变量或数组名，其中形式参数还可以是变量的地址。

## 12.5 应用举例

一个C程序由一个主函数（**main**函数）和若干个其他函数组成。函数将一个C语言程序分成几个模块，分别实现不同的功能，程序清晰，可读性较强。

**【例12-9】**编写一个函数求两个数的最大公约数和最小公部数

**【例12-10】**编一个名为**link**函数，求两个字符串连接后的字符个数。

## 12.5 应用举例

**【例12-12】**用矩形公式求 $f(x)$ 在 $[a,b]$ 的定积分: 先 $M$ 等份积分区间求得积分近似值, 再 $2M$ 等份求得积分近似值, 再 $4M$ 等份求得积分近似值, ....., 当两次积分近似值之差的绝对值小于 $\text{eps}$ 时返回计算结果。

**【例12-13】**验证哥德巴赫猜想: 任何一个大于6的偶数均可表示为两个素数之和。要求将6—100之间的偶数都表示为两个素数之和。

## 第13章 函数的调用

函数调用的一般形式；

函数调用的形式；

被调用函数的声明与函数原型；

函数嵌套与递归调用；

变量作用域；

编译预处理。



## 13.1 函数调用的一般形式

函数调用的一般形式为：

函数名（实际参数表列）；

如果是调用无参函数，则“实际参数表列”可以没有，但函数名后的括号是不能省略的。如果有多个实际参数，则各个参数之间用逗号隔开。实际参数的个数应与形式参数的个数相等，类型也应匹配。实际参数与形式参数顺序对应，一一传递数据。需要说明的是，在C语言中，实参表列的求值顺序是不确定的。有的系统按照自左向右的顺序计算，而有的系统相反。

## 13.2 函数调用的形式

根据函数在主调用函数中出现的位置，可以有以下3中调用方式。

1. 被调用函数作为函数语句单独出现

```
printf("I love China!");
```

```
scanf("%d",&a);
```

2. 被调用函数作为表达式出现

```
m=squar(a);
```

```
n=3+squar(a);
```

3. 被调用函数作为函数的参数出现

```
c=sum(a,sum(b,c));
```

## 13.3 被调用函数的声明与函数原型

在一个函数被另一个函数调用之前，需要具备如下条件：

被调用函数必须是已经存在的函数。被调用函数可以是库函数或者用户自己定义的函数。

如果被调用函数是库函数，应该在程序开头用**#include**命令将调用有关库函数时所需要用到的信息“包含”到本程序中。在前边的例子中，用到了这样的命令：

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

函数声明的一般形式是：

<返回值类型><函数名>(<参数类型声明表>);

其中<参数类型声明表>的形式是：

<参数类型>[参数名][,<参数类型>[参数名]...

如果函数是无参函数，括号中的内容可以不写。方括号中的内容也可以不写，也就是说可以只写参数类型。比如这样声明就是正确的：

```
int sum(int,int);
```

## 13.4 函数的嵌套调用和递归调用

**C**语言中的函数定义是互相平行、独立的，函数之间没有从属关系。在定义一个函数时，该函数不能包含另一个函数，即在一个函数定义中，其函数体中不能包含另一个函数的完整定义。即在**C**语言中不能嵌套定义。

## 13.4.1 函数的嵌套调用

尽管在C语言中不能嵌套定义，但可以嵌套调用函数，也就是说可以在调用一个函数的过程中调用另一个函数。

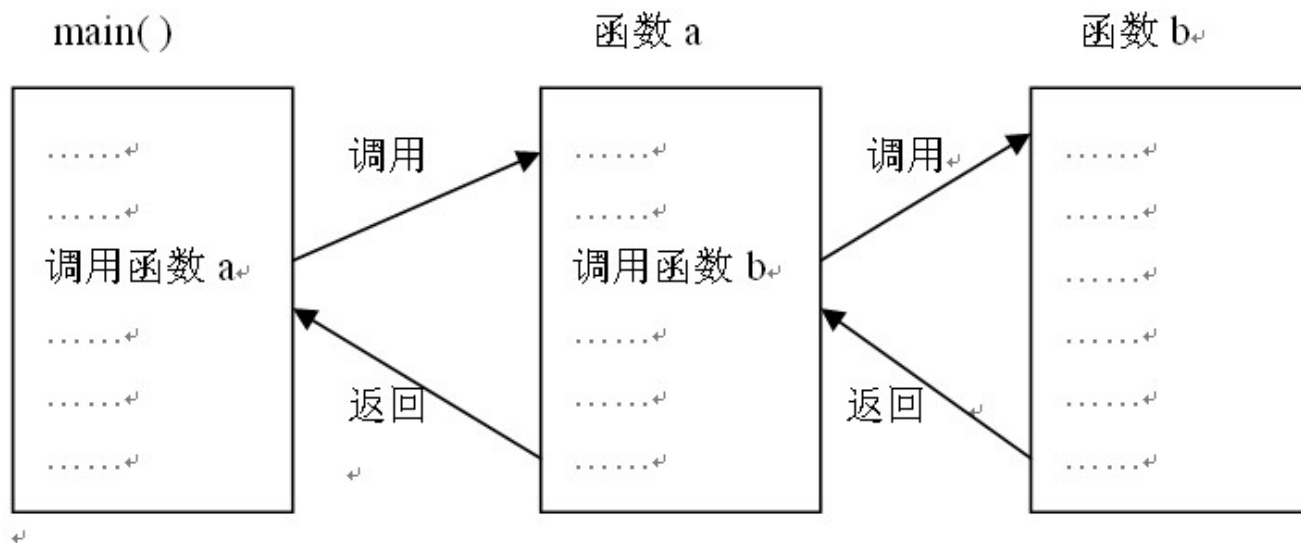


图 13-1 函数嵌套调用示意图

## 13.4.2 函数的递归调用

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，称之为函数的递归调用。例如：

```
float funct(int x)
{
    int y,z;
    ...
    z=funct(y);
    ...
}
```

```
int f1(int a)
{
    int b,c;
    ...
    c=f2(b);
    ...
}
int f2(int a)
{
    int b,c;
    ...
    c=f1(b);
    ...
}
```

## 13.5 变量作用域

在定义一个变量后，这个变量就有了一系列确定的性质，如数据长度、存储形式、数据的取值范围等等。除此之外，变量还有其他一些重要的属性，如变量在程序运行中何时有效，何时有效；变量在内存中何时存在、何时释放等等。变量的这些性质都与变量的作用域与生存期有关。



## 13.5.1 变量作用域和生存期

变量的作用域是指一个变量能够起作用的程序范围。如果一个变量在某个文件或函数范围内有效，则称该文件或函数为变量的作用域，在此作用域内可以引用此变量。

变量的生存期是指一个变量存在时间的长短。即从给变量分配内存，到所分配的内存被系统释放的时间。如果一个变量在某一时刻是存在的，则认为这一时刻属于该变量的“生存期”。

## 13.5.2 局部变量和全局变量

### 1. 局部变量

```
void main()↵
{↵
    int a,b;↵
    ...↵
    ...↵
}↵
int f1(int a,int b)↵
{↵
    int c;↵
    ...↵
    ...↵
}↵
```

```
int f1(int a,int b)↵
{↵
    int c;↵
    {↵
        int m,n;↵
        m=a+c;↵
        n=b+c;↵
        .....↵
        .....↵
    }↵
    .....↵
    .....↵
}↵
```

变量 m、n 的作用域

变量 a、b、c 的作用域。

## 2、全局变量

```
int a=3, b=5;↵
int f1()↵
{↵
    int a; ↵
    .....↵
    .....↵
}↵
float c;↵
void main()↵
{↵
    .....↵
    .....↵
}↵
int f2()↵
{↵
    .....↵
    .....↵
}↵
```

The diagram illustrates the scope of variables in the provided C code. It uses curly braces to group lines of code and label their scope:

- A large brace on the right side groups the first three lines of code (`int a=3, b=5;`, `int f1()`, and `{` with `int a;`) and is labeled "全局变量 a、b 的作用范围" (Scope of global variables a, b).
- A brace on the left side groups the `void main()` block and is labeled "全局变量 c 的作用范围" (Scope of global variable c).
- A brace on the left side groups the `int f2()` block.

全局变量的优点：

设置全局变量可以增加函数间的联系。由于同一C程序中所有函数都能使用全局变量，如果在一个函数中改变了全局变量的值，其他函数会受到影响，相当于各个函数之间有直接的传递通道。由于函数只能带回一个返回值，因此有时可以利用全局变量在函数间传递数据，通过函数调用能得到一个以上的值。

全局变量的缺点：

全局变量使函数的执行依赖于外部变量，降低了函数的通用性。

降低了函数的清晰性。各个函数执行时都可能会改变全局变量的值，很难判断出每个瞬时各个全局变量的值。

全局变量在程序运行过程中都会占用内存单元。

## 13.5.3 变量存储类别

变量的存储类别指的是数据在内存中存储的方式。变量的存储方式可分为两类：

静态存储类和动态存储类。具体包含4中：自动型（**auto**）、静态型（**static**）、寄存器型（**register**）和外部型（**extern**）。

# 局部变量的存储方式

## 1、自动变量

```
auto int m,n;          /*定义m、n为自动变量*/
```

## 2、静态变量

```
static int m,n;        /*定义m、n为静态变量*/
```

### 【例13-9】

对于静态变量做以下几点说明：

局部静态变量是在静态存储区分配存储单元的，在整个程序运行期间都不释放。因此在函数调用结束后，它的值并不消失。

局部静态变量时在程序编译过程中被赋值的，且只赋值一次，在程序运行时其初值已经确定，以后每次调用函数时不再赋值，而是保留上一次函数调用结束时的值。

局部静态变量的初值为0（对整型变量）或空字符（对字符型变量）。

虽然静态局部变量在函数调用结束后仍然存在，但是其他函数是不能引用它的。

由于静态局部变量占内存多（长期占用不释放，不能像动态存储那样一个存储单元可以供多个变量使用），而且由于其值可以改变，不能弄清楚局部静态变量的当前值是多少，降低了程序的可读性，因此不建议过多使用局部静态变量。

### 3、寄存器变量

C语言中允许将局部变量的值放在CPU中的寄存器中，需要时直接从寄存器中读取数据，不必再到内存中读取数据。这种变量称作寄存器变量，用关键字register声明。

```
register long i, f=1;          /*定义寄存器变量*/
```

对寄存器变量做以下几点说明：

只有局部自动变量和形式参数可以作为寄存器变量，其他类型的变量是不可以的。局部静态变量不能定义为寄存器变量。例如不能这样声明变量：

```
register static int a;
```

不能把变量既放在静态存储区又放在寄存器中，二者只能居其一。

一个计算机系统中的寄存器数目是有限的，不能任意定义多个寄存器变量。不同系统允许使用的寄存器变量也是不同的，而且对寄存器变量处理方式也是不同的。有的系统将寄存器变量当做自动变量处理。

# 全局变量的存储方式

## 1、外部全局变量

全局变量是在函数的外部定义的，它的作用域是从变量的定义处开始到本程序文件的结束。如果在定义点之前的函数想要引用该变量，则应该在引用之前用关键字`extern`对该变量进行声明，声明该变量为外部全局变量。声明过后，就可以从声明处起，合法的使用该变量。

### 【例13-11】

一个C程序可能由一个或多个源程序文件构成。如果程序由多个源程序文件组成，在一个文件中引用另一个文件中定义的全局变量，需要用`extern`关键字对全局变量作外部全局变量声明。



对部全局变量做以下几点说明：

(1) `extern`不能用来初始化变量，即`extern int a=3` 是不正确的。

(2) 使用`extern`的作用是扩展全局变量的作用域。

(3) 在系统编译遇到`extern`时，先在本文件中寻找全局变量的定义，如果找不到，在连接时从其他文件中寻找全局变量的定义。

(4) 在不同文件中引用全局变量时，因为全局变量的值可能会被改变，因此在使用时要特别注意。

## 2、静态全局变量

在设计一个程序时，有时不希望某些全局变量被其他文件引用，这时可以用关键字static对全局变量进行声明。

f1.c 中内容如下：

```
static int A=5;
int sum()
{
    .....
    .....
}
```

f2.c 中内容如下：

```
extern A;
int funct()
{
    .....
    .....
}
```

对于静态全局变量做以下几点说明：

(1) 使用static声明全局变量，可以避免文件中的一些全局变量被其他文件引用。

(2) 不管是否对全局变量进行static声明，全局变量均是静态存储方式。

(3) 使用static声明的全局变量，在本文件中定义在全局变量之前的函数也是不能引用的。

## 13.6 编译预处理

预处理功能主要有以下3种：

宏定义；

文件包含；

条件编译。

为了区别预处理命令与其他的C语句，所有的预处理命令均以“#”开头，占用独立的一行，且语句结尾不用分号“；”结束。

## 13.6.1 宏定义

宏定义是指用一个指定的标识符来定义一个字符序列。根据宏定义是否有参数将宏定义分为两种：不带参数的宏定义和带参数宏定义。

### 1. 不带参数的宏定义

不带参数的宏定义是用一个指定的标识符来代表一个字符串，一般形式为：

`#define 标识符 字符串`

例如：`#define PI 3.1415926`

## 对宏定义说明以下几点：

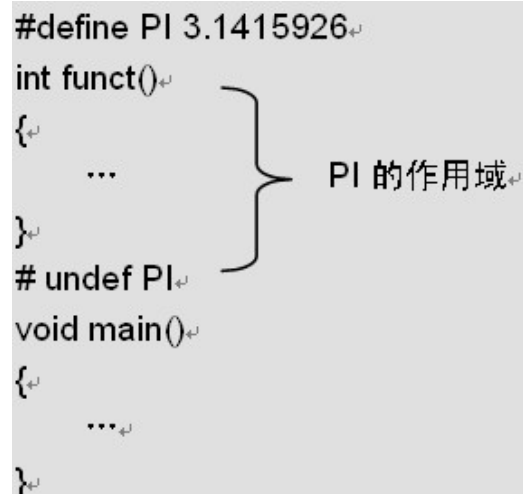
(1) 宏名的命名规则同一般标识符。为了使宏名和变量名有所区别，通常宏名用大写字母表示。当然这不是规定，也可以用小写字母表示。

(2) 使用宏名代替字符串，可以减少程序中重复书写这些字符串的工作量。

(3) 宏定义是用宏名代替一个字符串，不作正确性检查。

如： `#define size 100`

(4) 宏名的有效范围为定义命令之后到本源程序文件结束。可以用 `# undef` 命令来终止宏定义的作用域。



```
#define PI 3.1415926
int funct()
{
    ...
}
# undef PI
void main()
{
    ...
}
```

PI 的作用域

(5) 进行宏定义时，可以引用自己定义的宏名。

(6) 如果在程序中出现用双撇号括起来的字符串内包含有与宏名相同的名字，预编译时并不进行宏替换。

```
#define China Chinese
void main()
{
    printf("%s\n","China");
}
```

程序运行结果如下：

```
China
```

(7) 宏定义是专门用于预处理命令的，它与变量不同，并不占用内存空间。宏定义的末尾不必加分号。

## 2. 带参数的宏定义

C语言允许宏带有参数，在宏定义中的参数称为形式参数。带参数的宏定义不是进行简单的字符串代换，而是进行参数替换。其定义的一般形式为

```
#define 标识符(形式参数表列) 字符串
```

```
#define sum(a, b) (a+b)
```

在使用形式上带参数的宏与函数很相似，但是它与函数有本质的区别。

(1) 函数在定义和调用中所使用的形式参数和实际参数都受到数据类型的限制，而带参数宏的形式参数和实际参数可以是任意数据类型。

(2) 函数调用时是先计算实际参数表达式的值，然后代入形式参数。而宏定义展开时，只是替换。

(3) 函数调用时，编译系统会为其分配内存单元，而宏展开时系统不会为其分配内存单元，宏展开也没有“返回值”的概念。

(4) 多次使用宏，宏展开后源程序增长，而函数调用不会使源程序增长。

(5) 函数只能有一个返回值，而用宏可以设法得到多个值。

## 13.6.2 文件包含处理

文件包含处理是指一个源文件将另一个源文件的全部内容包含进来。其一般形式为：

`#include <文件名>` 或 `#include “文件名”`

文件包含说明以下几点：

编译时，并不是分别对两个文件进行编译，然后再将它们的目标程序连接，而是在经过编译预处理后将头文件包含到主文件中，得到一个新的源程序，然后再对新的源程序进行编译。因此，如果被保护文件中有全局静态变量，它也在新源程序中有效，不必用extern声明。

文件头部被包含的文件常以“.h”作为后缀名，当然用其他的后缀名甚至没有后缀名也是可以的。

文件除了可以包含函数原型和宏定义外，还可以包括结构体类型定义和全局变量等。



## 13.6.3 条件编译

条件编译命令有三种形式：

(1)

```
#if 表达式↵  
    程序段 1; ↵  
#else 表达式↵  
    程序段 2; ↵  
#endif↵  
或↵  
#if 表达式↵  
    程序段 1; ↵  
#endif↵
```

(2)

```
#ifdef 宏名↵  
    程序段 1↵  
#else↵  
    程序段 2↵  
#endif↵  
或↵  
#ifdef 宏名↵  
    程序段 1↵  
#endif↵
```

(3)

```
#ifndef 宏名↵  
    程序段 1↵  
#else↵  
    程序段 2↵  
#endif↵  
或↵  
#ifndef 宏名↵  
    程序段 1↵  
#endif↵
```

## 第14章 指针操作

指针是C语言的重要概念，也是C语言的一个重要特点。在学习本章内容时，要十分小心，多思考，多比较，在实践中掌握它。

指针与地址；

指针和指针变量；

指针和数组；

指针和函数。

## 14.1 指针与地址

在计算机中，内存是以字节为单位的连续存储空间，每一个字节都有一个编号，这个编号称为地址。

- (1) 任何变量在其生存期内都占据一定数量的字节。
- (2) 内存单元的地址与内存单元的内容是不相同的。
- (3) 一个变量的内存地址称作该变量的指针。

## 14.2 指针和指针变量

变量的内存地址就是变量的指针。指针变量顾名思义，就是用来存放内存地址的变量，也就是用来存放指针的变量。

## 14.2.1 指针变量的定义

假设p变量用来存放字符型变量所占用的存储单元的首地址，同时，假定已用某种方式将字符型变量c所占用的内存单元的首地址赋给了p变量，那么，想通过变量p取得字符变量c中的内容，可以按照以下步骤进行：

（1）根据变量p所占用的内存单元的首地址，读取其中所存放的数据，该数据就是字符变量c所占用的内存单元的首地址。

（2）根据第一步取出来的地址以及字符变量所占用的存储单元的长度，读取字符变量c的值。

指针变量定义的一般形式为：

类型说明符 \*标识符；

一般变量的定义形式为：

类型说明符 标识符；

```
int *a;    //该指针变量指向数据为整型数据
```

```
float *b;   //该指针变量指向数据为浮点型数据
```

用变量的地址给指针变量赋值，要求变量的类型必须与指针变量的类型同。我们这样为指针变量赋值：

```
int b;
```

```
int *a=&b;    /*b表示一个整型变量，下同*/
```

或

```
int *a;
```

```
a=&b;
```

/\*将变量b的地址存放到指针变量a中，因此a就“指向”了b\*/

```
int *p;
```

```
p=null;    /*赋空值null*/
```

## 14.2.2 指针变量的引用

在讲指针变量引用之前，先介绍两个有关的运算符。

**&：**取地址运算符。作用是获取变量的地址。形式为：**&变量**。例如：**&a**是获取变量所占内存空间的首地址。

**\***：取内容运算符。作用是取指针变量指向的内容。形式为：**\*指针变量名**。例如：**\*p**表示指针变量所指向内存单元中的数据。

(1) 将指针变量指向被访问的变量（先指向）

**P=&a;**

(2) 访问所指变量（再取值或赋值）

取值：**b=\*p;**

赋值：**\*p=100;**



特别的，如果在一个表达式中同时出现“&”和“\*”，一定要小心的分析该表达式。“&”和“\*”两个运算符的优先级别相同，但按自右而左的方式进行结合。例如：`&*a`；很明显的，`a`是一个指针变量，按照自右而左的顺序进行结合，首先取`a`指向的变量，然后再对该变量取地址，结果仍然是`a`。

乘法运算符“\*”与取内容运算符“\*”书写方法相同，但是这两个运算符是完全不相同的，两者之间没有任何联系。同样，位运算符“&”与取地址运算符“&”之间也没有任何联系。

## 14.2.3 指针的运算

指针变量同普通的变量一样，可以进行多种运算。对指针变量可以进行赋值运算、取地址运算、取内容运算、加减算术运算、关系运算。

### 1. &和\*运算

&和\*运算符的优先级同++ -- ! 等运算优先级别相同。并且是自右向左的结合方式。

- \*(++p)    p的地址先加1然后取值
- \*++p    取地址p加1后的值
- \*(p++)    先取p的值，然后p指针加1
- \*p++    先取p的值，然后p指针加1
- ++(\*p)    先取p的值，然后值加1

## 2. 赋值运算

赋值运算的一般形式为：

**指针变量=指针表达式**

(1) 可以将一个指针变量的值赋给相同类型变量的一个指针变量。例如：

```
int a, *b, *c;
```

```
b=&a;
```

```
c=b;
```

(2) 可以将数组的首地址赋值给指针变量。例如：

```
int a[10], *b;
```

```
b=a;
```

### 3. 算术运算

#### (1) 指针变量与整数相加、减

将指针加上或减去某个整数值，表示将指针向前或向后移动n个数据单元。如：

```
int a[5], *b, *c;
```

```
b=&a[1];
```

```
c=b+2;
```

#### (2) 指针变量的自增、自减运算

变量自增、自减的一般形式为：

**p++， p--， ++p， --p**

其中p为指针变量。

#### 4. 关系运算

指针变量之间的关系运算与一般变量之间的关系运算方式是相同的。

**【例14-8】**

## 14.3 指针和数组

一个变量有地址，一个数组同样有地址。不同的是，一个数组包含有若干个元素，每个元素都要占据内存空间，且数组占据的内存空间也是连续的。指针变量既然可以指向变量，也可以指向数组。因此，在引用数组元素的时候，可以用下标法（如`a[6]`），也可以使用指针法，即通过指向数组元素的指针找到所需的元素。使用指针法能够使目标程序质量高：占内存少，运行速度高。

### 14.3.1 数组的指针和指向数组的指针变量

一个变量有地址，一个数组元素包含若干个数组元素，每个数组元素都在内存中占用存储单元，它们都有相应的地址，这个地址就可以用指针来实现存储。

例如：

```
int a[5];  
int *p, *q;  
p=&a[0];  
q=&a[2];  
*p=5;  
*q=8;
```

若: `p=a; /* 或写成p=&a[0]; */`

则:

`p+1 ⇔ &a[1]`

`*(p+1) ⇔ a[1]`

`p+i ⇔ &a[i]`

`*(p+i) ⇔ a[i]`

数组名代表数组首元素的地址，它是一个常量，它在程序运行期间是不变的。因此对下面两个语句：

```
for (p=a; a<p+10; a++)      /*a为数组a的数组名 */  
    printf("%d", *a);
```



注意：

C语言中规定 $p+1$ （或 $p++$ ）指向数组的下一个元素（并不是地址值简单加1）。

当 $p$ 的初值为  $a$  的首地址时， $p+i$ 或 $a+i$ 。就是 $a[i]$ 的地址，因为  $a$  代表了数组的首地址。

$*(p+i)$ 或 $*(a+i)$ 所指的数组元素就是 $a[i]$ 的内容。

指向数组的指针变量也可以带下标，如： $*(p+i)$ 等价于 $p[i]$ ，即 $a[i]$ 。

## 14.3.2 指针数组和指向指针的指针

在整型数组中的元素都是整型，在字符型数组中的元素都是字符型，那么在指针数组中的元素都是指针类型数据。指针数组中的每一个元素都相当于一个指针变量。指针数组的每个元素只能存放地址型数据。

指针数组的定义形式为：

类型说明符 \*数组名[数组长度1]...[数组长度2]...;

指针数组元素的引用和普通数组元素的引用方法完全相同：

下标法      指针数组名[下标]

指针法      \*(指针数组名+i)

例如：

```
int x[5], a, b, *p[5];  
p[0]=x;            /*指针数组元素p[0]指向整形数组x的首地址*/  
p[1]=x+2;          /*指针数组元素p[1]指向整形数组x的元素x[2]*/  
*(p+2)=&a;        /*指针数组元素p[2]指向整形变量a*/  
*p[0]=1;           /*给p[0]指向的整形数组元素x[0]赋值为1*/  
*(*(p+1))=2;       /*给p[1]指向的整形数组元素x[2]赋值为21*/  
*p[2]=3;           /*给p[2]指向的整形变量a赋值为3*/  
b=p[0]<p[1];        /*比较p[0]和p[1]中的地址值，结果变量b为1*/  
p[0]++, --p[1];     /*结果p[0]和p[1]均指向整形数组元素x[1]*/
```

### 14.3.3 指向字符串的指针

在C语言没有专门的字符串变量，如果想把一个字符串存放在变量中予以保存，必须使用数组，即使用字符型数组来存放字符串，数组中的每一个元素存放一个字符。可以有两种方法获得一个字符串的指针。

用字符串数组存放一个字符串，这样可以用数组的地址代表字符串的地址。例如：

```
char a[ ]="I love China!";
```

```
p=a;
```

使用指针变量直接指向字符串。例如：

```
char *p="I love China!";
```

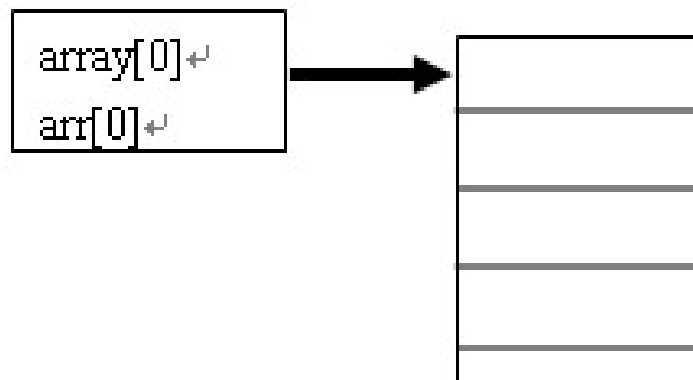
**【例14-15】**

## 14.3.4 数组名作为函数参数

数组名作函数参数时，传递的是数组的首地址。因此，形参数组中的元素的值发生变化后，返回调用函数时，实参数组的相应元素的值也发生变化。因为实参数组和形参数组共享同一段内存。

```
fun (int arr[],int n)
{
    ...
}

void main()
{
    int array[10];
    ...
    fun(array, 10);
    ...
}
```



## 14.4 指针和函数

函数是C程序的基本模块，在编译C程序时，系统会为每一个函数分内存单元，函数既然占有内存单元，那么函数也是有地址的，这个地址称作函数的入口地址。

## 14.4.1 函数的指针和指向函数的指针变量

让指针变量指向函数，代码形式如下：

指针变量=函数名

指向函数的指针一般定义形式为：

类型说明符（\*指针变量名）（函数参数表列）

用指针变量调用所指向的函数时，需要注意：

- （1）指针变量必须已经指向了某个函数。
- （2）数据类型可以是整型、实型、字符型甚至是结构型的数据类型。定义指针变量时的数据类型和定义函数时的数据类型必须一致。
- （3）指针变量名是用户所选用的标识符，表示指向函数的指针变量。
- （4）“\*指针变量”必须加括号，因为运算符\*的优先级低于运算符（）。若写成int \*p（）；因为运算符“\*”的优先级低于“（）”，所以，会是p先和（）结合，代表这是一个函数。该函数的返回值是指向整型的指针。

在定义函数指针之后，可以通过它间接调用所指向的函数。同其他类型指针相似，首先将一函数名赋给函数指针，然后才可以通过函数指针间接调用这个函数。一个函数指针既可以指向用户自定义的函数，也可以指向由C语言系统所提供的库函数。

例如：

```
int funct(int,int);    /*定义一个函数*/  
int (*p)(int,int);    /*定义一个指向函数的指针变量*/  
p=funct;              /*把函数入口地址赋值给指针变量*/
```

在定义函数的指针变量时，可以不赋初值。如果赋初值，则初值是一个已经定义好的某个函数的函数名，这时，称该指针变量指向对应的函数。例如：

```
int (*p)( )=find_max;  /*赋初值*/  
int (*p)( );          /*不赋初值，在后面赋值*/  
p=find_max;
```



## 14.4.2 用指向函数的指针作为函数参数

整型变量、字符变量、数组、指针变量、数组指针变量等都可以作为函数的参数，函数指针变量也是可以作为函数参数的。如果是指针的话，相当于将一个变量的地址传给了形参。函数指针变量作为函数参数的主要作用是把函数地址作为参数传递到其他函数，这样就能够和被调用的函数中使用实参函数。

函数间传递数据有四种形式：值传递方式、地址传递方式、返回值方式以及外部变量传递。前两种方式需要利用函数参数传递数据，后两种方式不必使用函数参数来传递数据。

## 14.4.3 指针数组作为函数参数

指针数组的一个重要作用是作为函数的参数，特别是 `main()` 函数的参数。在以前的程序中，`main()` 函数的第一行一般写成如下形式：

```
void main()
```

括号中是空的，表示 `main()` 函数为无参函数，实际上，`main()` 函数是可以有参数的。

例如：

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

主函数带参数时，我们在执行编译后的目标程序时，输入的命令格式是：

```
命令名 参数1 参数2 ...参数n
```

## 14.4.4 返回指针值的函数

一个函数不仅可以返回整型、字符型和结构类型的数据，还可以返回指针类型的数据。对于返回指针类型数据的函数，在函数定义时，也应该进行相应的返回值类型说明。定义形式为：

类型名 \*函数名(参数列表);

例如：

```
int *fun( )  
{  
    int *p;  
    ...  
    return (p);  
}
```

## 14.4.5 字符串指针作为函数参数

字符串的表示形式是：

```
char str[80]="China";
```

```
char *p=str;
```

```
printf("%s",str);
```

```
printf("%s",p);
```

```
printf("%s","China");
```

上述三个字符串输出函数的输出结构都一样，都是China。

我们还可以直接使得p指向字符串的开始地址，即，写成

```
char *p="China";
```

## 第15章 结构体

**C**语言具有丰富的数据类型。结构型、共用型和数组属于构造型数据，都是由若干个数据组合而成的，但结构型、共用型与数组不同之处在于它们可以用来处理不同的数据。**C**语言除了系统定义的数据类型之处，允许用户自己定义数据类型，结构体，共用体。

结构体类型定义；

结构体变量的定义与引用；

结构数组；

结构体指针；

链表。

## 15.1 结构体类型定义

结构体是由一批数据组合而成的一种新的数据类型。组成结构型数据的每个数据称为结构型数据的“成员”。这些成员可以具有不同的数据类型。由于结构体数据的成员类型不同，因此结构体要由用户在程序中自己定义。

## 15.1.1 结构体类型的说明

**C**语言规定，结构体是由用户在程序中自己定义的一种数据类型。用户可以按照自己的需要，先定义结构体，然后再定义这种结构型的变量、数组以及指针变量。

## 15.1.2 结构体类型的定义

结构体类型定义的一般形式为：

```
struct 结构体名
{
数据类型1 成员名1;
数据类型2 成员名2;
数据类型3 成员名3;
...
数据类型n 成员名n;
};
```

例如：

```
struct student
{
    int number;
    /*学生的学号*/
    char name[20];
    /*学生的姓名*/
    char sex;
    /*学生的性别*/
    float score[7];
    /*七门课程绩*/
};
```



对于结构体类型的定义需要做以下几点说明：

(1) 结构体名是用户用于标识结构体或选取的标识符，必须符号标识符的命名规则。

(2) 数据类型1...数据类型n，可以是任何一种数据类型，可以是基本类型说明符，也可以构造体类型说明符，也可以是已经定义过的结构体名。

(3) 成员名1...成员名n，是用户自己定义的标识符，用来标识该结构体所包含的成员名称。相当于变量定义中的变量名。

(3) 结构体定义在括号后的分号是不可少的。

(4) 如果成员名的前面有“\*”，表示该成员是指针型；如果成员名后面有“[长度]”，表示该成员是数组型。

```
struct student
{
    int number;
    char *name;          /*定义成员名为name，数据类型为指针*/
    char sex;
    float score[7];      /*定义成员名为score，数据类型为数组*/
};
```

(5) 当某个结构体成员的数据类型是另一个结构体时，称为“嵌套结构体”。此时，作为成员数据类型的结构体的定义必须出现在本结构体定义之前。

(6) 当结构体成员的数据类型选取了自身的结构体时，成员只能是本结构体的指针变量或者是结构体指针型数组，但不能是结构体变量或结构数组。例如：

```
struct strul
{
    int s1;
    struct strul *s2;
    /*正确定义，成员是指针型变量，数据类型可以是本结构体*/
    struct strul *s3[10];
    /*正确定义，成员是指针型数组，数据类型可以是本结构体*/
    struct strul s4;
    /*错误定义，成员是变量，数据类型不能是本结构型*/
    struct strul s5[3];
    /*错误定义，成员是数组，数据类型不能是本结构型*/
}
```

(7) 结构体是一种数据类型，在定义结构体时，其成员并不分配内存。只有用“struct 结构体名”来定义该结构型的变量、数组以及指针变量时，才会分配内存。

## 15.2 结构体变量的定义与引用

结构体类型本身是一个数据类型，具体的数据，也不占用系统内存空间。当程序中已经定义了结构体时，就可以使用“**struct** 结构体名”作为数据类型符来定义处理这种结构体数据的变量、数组及指针等。

## 15.2.1 结构体变量的定义与初始化

1. 先定义结构体，再定义结构体变量

这种方式的定义语句格式如下：

```
struct 结构体名
```

```
{
```

```
数据类型1  成员名1;
```

```
数据类型2  成员名2;
```

```
...
```

```
数据类型n  成员名n;
```

```
};
```

```
...
```

```
struct 结构体名 结构体变量名=初值;
```

2. 在定义结构体的同时定义变量  
这种方式的定义语句格式如下：

```
struct 结构体名
{
    数据类型1  成员名1;
    数据类型2  成员名2;
    ...
    数据类型n  成员名n;
} 结构体变量名=初值;
```

### 3. 直接定义结构体类型变量

直接定义结构体类型变量是指在定义结构体时定义变量，并将结构体名省略。这种方式的定义语句格式如下：

```
struct  
{  
    int num;  
    char name[20];  
    char sex;  
    int age;  
    float score[7];  
    char addr[30];  
    }stu1={00001,"xiaoli",'m',23,{80,90,70,91,85,88,68},  
"北京"},    /*定义结构体变量并赋值*/  
    stu2={00002,"xiaowang",'f',22,{96,68,78,88,79,76,8  
4},"南京"};
```

## 15.2.2 结构体变量的引用

在程序中使用结构体变量时，不能将其作为一个整体来使用。结构体中的成员可以当成一般的变量来使用，结构体变量的赋值、输入、输出、运算等都是通过结构体中的成员来实现的。结构体变量成员的引用方法如下：

结构体变量名. 成员名

```
printf(“%d,%s”,t1);
```

```
printf(“%d,%s”,t1.num,t1.name);
```

```
struct DATE           /*定义学生出生日期的结构体*/
{
    int year;          /*定义年*/
    int month;         /*定义月*/
    int day;           /*定义日*/
};

struct student
{
    int num;           /*定义学生的学号*/
    char name[10];      /*定义学生的姓名*/
    struct DATE birthday; /*该成员的数据类型是生日
结构体*/
    }t1={1000,"wangwu",{1987,12,5}},t2;
```



如果是嵌套的结构型，其成员的引用方法如下：

外层结构体变量名. 外层成员名. 内层成员名

上例中，如果要引用学生的出生日期，表示如下：

t1. birthday. year

t1. birthday. month

t1. birthday. day

对于结构体变量的引用做以下几点说明：

(1) 结构体变量的成员可以是任何类型，对该成员的操作与相同类型的普通变量并无区别，但需要在成员名前缀以结构体变量名。

(2) 结构成员变量的使用与普通变量没有区别，因此可根据类型进行相应的运算。

(3) 如果成员本身又是一个结构体时，必须逐级找到最低一级的成员才能使用。

(4) 可以使用成员的数据，也可以使用成员的地址。结构体变量的地址是可以使用的，结构体变量的使用主要要为结构体变量成员的引用、结构型变量成员地址的引用及结构型变量地址的引用。

(5) 允许将一个结构变量直接赋值给别一个具有相同结构的结构体变量。

## 15.3 结构数组

数组元素也可以是结构体类型的数据，当数组元素为结构体类型时，就称为结构数组。

## 15.3.1 结构数组的定义

```
struct student    /*定义名为student结构体*/  
{  
    int num;  
    char name[20];  
    char sex;  
    int age;  
    float score[7];  
    char addr[30];  
} stu1[3];  
.....  
struct student stu2[3];
```

对结构数组赋予初值的格式如下：

{ {数组元素1的各个成员的初值表}, {数组元素2的各个成员的初值表}, ..... }

## 15.3.2 结构数组的引用

```
struct student  
{  
    int num;  
    char name[20];  
    float score;  
} stu[10];
```

(1) 结构数组首地址的引用方法:

结构数组名

&结构数组名[0]

例如: 用stu或&stu[0]表示结构数组stu的首地址。

(2) 结构数组元素的引用方法与普通数组的引用方法相似:

**下标法: 结构数组名[下标]**

**指针法: \*(结构型数组名+下标)**

例如: 如果要获得结构数组的第7个学生的信息, 表示方法: `stu[6]`

`*(stu+6)`

(3) 结构数组元素的引用方法:

**&结构数组名[下标]**

**结构数组名+下标**

`&stu[6]`和`(stu+6)`表示结构数组`stu`第7个元素的地址。

(4) 结构数组元素的成员的引用方法:

**结构数组名[下标]. 成员名**

**(\*(结构型数组名+下标)). 成员名**

例如: 如果要获得结构数组第2个学生的姓名, 表示为:

`stu[1].name`

`*(stu+1)).name`

(5) 结构数组元素的成员地址的引用方法:

**&结构数组名[下标]. 成员名**

**&(\*(结构型数组名+下标)). 成员名**

如果要获得结构数组第2个学生的姓名存储地址, 表示为:

`&stu[1].name`

`&*(stu+1)).name`

## 15.4 结构体指针

指向结构全的指针称为结构体指针，结构体指针变量也是指针变量，和普通指针变量的惟一区别就是这个指针指向的同一种结构体变量或结构数组。

## 15.4.1 结构体指针变量的定义

结构体指针变量的定义与结构体变量、结构数组的定义相似。结构体指针变量的定义也分为三种形式，一是在先定义结构体，后定义结构型指针；二是在定义结构体的同时定义结构体指针；三是直接定义结构体指针。一般形式为：

```
struct 结构体名 *结构指针变量名;
```

例如：

```
struct student          /*定义名为student的结构体*/
{
    int num;
    char name[20];
    char sex;
    int age;
    float score[7];
    char addr[30];
} *p1;                   /*定义结构体同时定义结构体指针变量*/
...
struct student *p2;     /*先定义结构体再定义结构体指针变量*/
```



## 15.4.2 结构体指针变量的引用

结构体指针变量的引用有以下几种：

(1) 使用结构体指针变量指向结构体变量或结构型数组

结构体指针变量=&结构体变量

结构体指针变量=&结构数组名[下标]

结构体指针变量=结构数组名+下标

结构体指针变量=结构数组名

(2) 指向结构体变量或数组元素的指针变量的引用

p1=&stu1;

\*p

(3) 指向结构体数组首地址的指针变量的引用

p2=stu;

\*(p2+4)

(4) 使用指向结构体变量或结构数组的指针变量引用其成员

(\*结构体指针变量).成员名

结构体指针变量—>成员名

例如:

(\*p1).num

p1—>num

(5) 使用指向结构数组首地址的指针变量引用其成员

\*(结构体指针变量+下标).成员名

(结构体指针变量+下标)—>成员名

例如:

p2=stu2;

\*(p2+1).num

(p2+1)—>num

## 15.5 链表

链表是一种特殊的结构体，它的外部结构是一串存放数据的对象，它的内部结构包含两部分，即存放数据和指针。

## 15.5.1 链表概述

### 1. 静态与动态内存分配

分配固定大小的内存分配方法称之为静态内存分配

所谓动态内存分配就是指在程序执行的过程中动态地分配或者回收存储空间的分配内存的方法。

### 2. 链表的定义

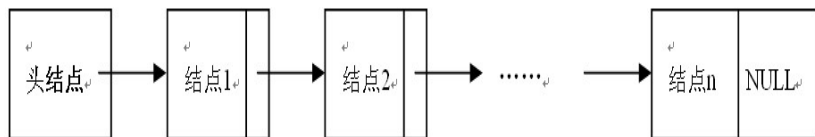


图 15-6 链表结构图

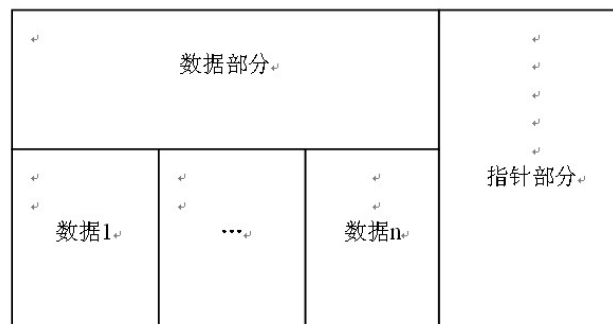


表 9-7 链表结点结构图

一个结点的结构体定义如下：

**struct** 结点结构体名

{

数据类型1 成员名1;        /\*存放数据成员1\*/

...

数据类型n 成员名n;        /\*存放数据成员n\*/

**struct** 结点结构体名 \*指针变量名;

/\*用于指向下一个结点的指针变量\*/

}

## 15.5.2 单链表建立

链表是一种动态的数据结构，它所需要的内存空间无法预先确定，取决于实际的应用情况。**C**语言提供了一些内存管理函数。运用这些内存管理函数可以按需要动态地分配内存空间，用于存放结点的数据，将结点的指针把各个结点链接起来构成一个链表，当链表或者某个结点不用时可以空间回收待用，使用内存资料得到合理的利用。

## 15.5.2 单链表建立

### 1. 内存管理系统函数

#### (1) **malloc( )**函数

(类型说明符\*) **malloc(size)**

#### (2) **calloc( )**函数

(类型说明符\*) **calloc(items,size)**

#### (3) **free (p)** 函数

### 2. 单链表的建立

建立单链表总体上分为三步：

(1) 调用**malloc**函数动态分配某个结点大小的存储空间。

(2) 向结点中的数据域存放数据。

(3) 将该结点的指针域指向一下结点的首地址。

## 15.5.3 单链表简单操作

建立了一个单链表之后，要运用链表，还需要掌握一些链表基本的操作。单链表的操作包括数据的输出、查找、插入和删除。

1. 单链表的输出
2. 从单链表中查找结点
3. 向单链表中插入新结点
4. 在单链表中删除结点



## 第16章 共用体

共用体又称为联合体，它和结构体一样也是一种由用户自己定义的数据类型，它也由若干个成员数据组成。其成员的数据类型可以是相同的，也可以是不同的。

共用体类型定义

共用体类型变量、数组和指针变量的定义

共用体类型变量、数组和指针变量的引用

共用体应用举例

用**typedef**定义数据类型

## 16.1 共用体类型定义

由于不同的共用体可以不有同的成员，因此共用体也需要用户在程序中根据自己的需要自己定义。定义共用体之后，就可以使用这种数据类型。

## 16.1.1 定义共用体

所谓共用体类型是指将不同的数据项组织成一个整体，它们在内存中占用同一段存储单元。由于不同的共用体类型的数据可以有不同的成员，因此共用体也是需要用户在程序中自己定义的一种数据类型。共用体的定义格式如下：

```
union 共用体名  
{  
    数据类型1 成员1名;  
    数据类型2 成员2名;  
    ...  
    数据类型n 成员n名;  
};
```

## 16.1.2 共用体的存储

从共用体的定义中可以看出，共用体数据类型与结构体在形式上非常相似，但两者有本质上的不同。在结构体中各成员有各自的内存空间，一个结构体变量的总长度是各成员长度之和。而共用体中，各成员共享一段内存空间，一个共用体变量的长度等于各成员中最长的长度。

```
union data      /*共用体*/  
{  
    int a;  
    float b;  
    double c;  
    char d;  
};
```

对于共用体，作以下几点说明：

(1) 共用体只有定义了该共用型的变量、数组或指针变量后，才会给该变量、数组和指针变量分配内存。

(2) 同一个内存可以用来存放几种不同类型的成员，但在每一瞬时只能存放其中一种，而不是同时存放几种。也就是说，每一瞬时只有一个成员起作用，其他的成员不起作用，即不是同时存在和起作用。

(3) 共用体变量中起作用的成员是最后一次存放的成员，在存入一个新的成员后原有的成员就失去作用。

(4) 共用体变量的地址和它的各成员的地址都是同一地址。

(5) 不能对共用体变量名赋值，不能企图引用变量名来得到一个值，也不能在定义共用体变量时对它初始化。

(6) 不能把共用体变量作为函数参数，也不能使函数带回共用体变量，但可以使用指向共用体变量的指针。

(7) 共用体类型可以出现在结构体类型定义中，也可以定义共用体数组。反之，结构体也可以出现在共用体类型定义中，数组成可以作为共用体的成员。

## 16.2 共用体类型变量、数组和指针变量的定义

定义了共用体之后，就可以用这种数据类型来定义相应的变量、数组以及指针变量等。共用体变量、数组和指针变量的定义和一般的变量、数组和指针变量的定义方法相同，惟一需要注意的是“数据类型符”必须是用户自己定义的公用体，即“**union** 共用体名”。

共用体变量、数组和指针变量的定义与结构体变量、数组及指针变量的定义方法相同。分为三种：第一种是先定义共用体，再定义共用体变量、数组及指针变量；第二种是定义共用体的同时定义共用体变量、数组及指针变量；第三种是定义共用体的同时定义共用体变量、数组及指针变量，但省略共用体名。

## 16.2.1 先定义共用体，再定义共用体变量、数组及指针变量

其定义格式如下：

**union** 共用体名

{

数据类型1 成员1名;

数据类型2 成员2名;

...

数据类型n 成员n名;

};

...

**union** 共用体名 变量名,共用体数组名[数组长度],\*共用体指针变量名;

## 16.2.2 定义共用体的同时定义共用体变量、数组及指针变量

这种方式的定义如下：

**union** 共用体名

{

数据类型1 成员1名;

数据类型2 成员2名;

.....

数据类型n 成员n名;

}变量名,共用体数组名[数组长度],\*共用体指针变量名;



## 16.2.3 定义共用体变量、数组及指针变量时省略共用体名

这种格式其实跟第二种格式相似，只是把共用体名省略掉了。格式如下：

**union**

{

数据类型1 成员1名;

数据类型2 成员2名;

...

数据类型n 成员n名;

}变量名,共用体数组名[数组长度],\*共用体指针变量名;

## 16.3 共用体类型变量、数组和指针变量的引用

```
union  
{  
    int classno;  
    char address[20];  
} a, b[5], *p;
```

(1) 用共用体变量引用其成员，引用格式如下：

共用体变量名. 成员名

例如：a.classno

(2) 用共用体数组元素来引用其成员，引用格式如下：

共用体数组名[下标]. 成员名

例如：b[0].classno

(3) 共用体指针变量引用该共用体的变量或数组，格式如下：

共用体指针变量=&共用体变量名

共用体指针变量=&共用体数组名[下标]

共用体指针变量=共用体数组名

(4) 使用共用体指针变量引用共用体的成员，引用格式如下：

(\*共用型指针变量).成员名

共用型指针变量->成员名

## 16.4 共用体应用举例

**【例16-6】**设有若干个成员的数据，其中有教师和学生。学生的数据包括号码、姓名、性别、职业、班级。教师数据包括号码、姓名、职业、职务。教师和学生的数据是不同的，现要求把它们放在同一张表格中，如表**16-1**所示。

要求输入人员的数据，然后再输出。

表 16.1 人员信息表

号码 (num)	姓名 (name)	性别 (sex)	职业 (job)	班级 (classno)/职务 (position)
1001	wufen	w	s	39
1005	lutao	m	t	prof

图 16.4-1 共用体应用举例

## 16.5 枚举类型

在实际问题中，有些变量的取值被限定在一个有限的范围内。例如，一个星期内只有七天，一年只有十二个月，一个班每周有六门课程等等。如果把这些量说明为整型，字符型或其它类型显然是不妥当的。为此，C语言提供了一种称为“枚举”的类型。

## 16.6 枚举类型的定义

枚举的定义枚举类型定义的一般形式为：

```
enum 枚举名  
{ 枚举值表 };
```

例如：

```
enum weekday  
{ sun,mou,tue,wed,thu,fri,sat };
```

对于枚举类型需要说明以下几点：

(1) 定义的枚举类型用“**enum**标识符”标识。枚举数据(枚举常量)是一些特定的标识符，标识符代表什么含义，完全由程序员决定。数据枚举的顺序规定了枚举数据的序号，从**0**开始，依次递增。

(2) 在定义枚举类型时，程序员可在枚举数据时通过“**=**”号自己规定序号，并影响后面的枚举数据的序号，后继序号以此递增。

(3) 枚举变量的定义与结构体和联合体一样，枚举变量也可用不同的定义方式，即先定义枚举类型再定义变量、定义枚举类型的同时定义变量或直接定义变量。

## 16.7 枚举类型变量的赋值和引用

枚举类型变量在定义以后，要使用这些枚举类型变量，以使其具有一定的值。枚举类型变量的赋值和引用需要注意以下几点：

(1) 枚举值是常量，不是变量。不能在程序中用赋值语句再对它赋值。

(2) 枚举元素本身由系统定义了一个表示序号的数值，从0开始顺序定义为0，1，2....。

(3) 只能把枚举值赋予枚举变量，不能把元素的数值直接赋予枚举变量。



## 16.8 用typedef定义数据类型

自定义数据类型符的语法格式为：

**typedef** 类型符1 类型符2;

### 1. 用**typedef**定义基本数据类型

**typedef** 基本数据类型符 用户自定义数据类型符;

### 2. 用**typedef**定义数组类型

**typedef** 数据类型符 用户自定义数组类型符[数组长度];

### 3. 用**typedef**定义指针类型

**typedef** 数据类型符 \*用户自定义指针类型符;

用**typedef**定义结构体的格式如下：

**typedef struct**

**{**

数据类型**1** 成员名**1**;

数据类型**2** 成员名**2**;

...

数据类型**n** 成员名**n**;

**}用户自定义结构类型符;**

对于**typedef**自定义数据类型需要做以下几点说明：

（1）用**typedef**自定义数据类型，只是对已有的数据类型加一个类型名，并没有产生新的数据类型。如：

```
typedef int INTEGER;
```

```
INTEGER a;
```

（2）用**typedef**可以定义各种数据类型名，但不能定义变量。用**typedef**定义的是数据类型的别名，可以用这个别名去定义相应的变量。

（3）有时可以使用宏定义来代替**typedef**的功能，但事实上，二者是不同的。宏定义只是简单的字符串替换，是在预编译的时候处理完成的；而**typedef**是在编译的时候完成的，其更为灵活。

## 第17章 文件

文件也是一种数据类型，是存储在外部存储设备上的数据集合，可用于保存大量的数据。对文件的处理，主要分为打开与关闭文件、从文件中读取数据和向文件中写入数据、文件的定位、文件的检测。对文件的这些处理都是利用系统函数和指向文件类型的指针变量进行的。本章的主要内容校

**C**文件概述；

文件类型指针；

文件的打开与关闭；

文件的读写操作；

文件的定位；

文件的检测。

## 17.1 文件概述

文件是按照某个规则集合在一起保存在外部存储器上的一批数据。组成文件的数据类型可以是各种类型的数据，如整型、字符型、字符串等，也可以是程序清单等。

文件可以用于永久地保存大量的数据，存储在磁盘上的数据在计算机关闭以后仍然存在，下一次使用的时候可以从磁盘上读取文件中的数据继续处理。

## 17.1.1 文件名

文件名是文件的标识符，每一个文件都以一个唯一的文件名进行存储。文件名是由一组字符构成的，目录分隔符和空操作符不能出现在文件名中。

在对文件进行处理的时候必须给出文件名。文件名的一般组成如下：

盘符：路径\文件名.扩展名

例如：

D:\web\teacher\PopCalendar2005\Cprograme.txt.

## 17.1.2 文件的类型

按文件中数据组织形式，可以把文件分为文本文件和二进制文件。

在文本文件中，存放的数据都是将其转换成对应的**ASCII**代码字符来存放的，该文件由一个个字符组成，每一个字节存放一个**ASCII**码值，代表一个字符。例如，一个整型数据**-15621**在文本文件中按字符存放，分别存放字符‘-’、‘1’、‘5’、‘6’、‘2’、‘1’，共占**6**个字节；一个单精度类型数据**3.14159**，分别存放的是字符‘3’、‘.’、‘1’、‘4’、‘1’、‘5’、‘9’，共占**7**个字节。

二进制文件中的数据都是按其二进制方式存放的，每个数据占用的字节数取决于该数据的数据类型。例如，一个整型数据**-15621**在二进制文件中占**4**个字节，单精度类型数据**3.14159**在二进制文件中占**4**个字节。

## 17.2 文件类型指针

文件型是一种特殊的结构体，该结构体用来存放文件的有关信息（如文件的名字、文件的状态及文件当前的位置等）。该结构体类型是由系统定义的，取名为“**FILE**”。对**FILE**这个结构体类型的定义是在**stdio.h**头文件中。

文件指针的定义如下：

**FILE \*文件指针变量名;**

如：**FILE \*fp;**

**#include <stdio.h>或#include “stdio.h”**



## 17.3 文件的打开与关闭

在**C**语言之中，对文件读写之前必须先打开文件，在使用以后要关闭该文件。文件的打开与关闭都是利用系统函数来实现的，通过调用文件打开函数**fopen()**和文件关闭函数**fclose()**，完成文件的打开与关闭。

## 17.3.1 文件打开函数fopen

文件打开函数**fopen**的调用格式为：

**FILE \*fp;**

**fp=fopen(filename,mode);**

文件打开函数**fopen**如果打开成功，就返回一个文件类型的指针，并将赋值给文件指针变量**fp**，如果失败，则返回**NULL**。

文件打开函数**fopen**有两个参数，**filename**指定打开的文件名，**mode**指定文件打开的方式。

(1) 文本文件的打开方式有以下几种：

r：打开一个已经存在的文本文件，只能从文本文件中读取数据。如果指定文件不存在，程序就会出错。

w：打开一个文件文件，只能将数据写入文件。如果已经存在该文件名的文件，文件被重写；如果不存在，则以该文件名建立新的文件。

a：以附加方式打开文件，将数据写入文件的尾部。如果文件不存在，创建新的文件用于写入。

r+：打开一个已经存在的文本文件，可以从中读取数据，也可以写入数据。

w+：打开一个已经存在的文本文件，可以读取数据，也可以写入数据，若文件不存在，则自动建立一个新文件，接受写入的数据；若文件存在，则删去旧文件，建立一个同名新文件，接受写入的数据。

a+：打开一个已经存在的文本文件，可以读取数据，也可以从当前文件的尾部追加写入数据。当文件不存在时，创建新的文件用于文件尾写入。

(2) 二进制文件的打开方式有以下几点：

rb：打开一个已经存在的二进制文件，只能从二进制文件中读取数据。如果指定文件不存在，程序就会出错。

wb：打开一个二进制文件，只能将数据写入文件。如果已经存在该文件名的文件，文件被重写；如果不存在，则以该文件名建立新的文件。

ab：打开一个已经存在的二进制文件，只能从当前文件的尾部追加写入数据。如果文件不存在，创建新的文件用于写入。

rb+：打开一个已经存在的二进制文件，可以从中读取数据，也可以写入数据。

wb+：打开一个已经存在的二进制文件，可以读取数据，也可以写入数据，若文件不存在，则自动建立一个新文件，接受写入的数据；若文件存在，则删去旧文件，建立一个同名新文件，接受写入的数据。

“ab+”，打开一个已经存在的文本文件，可以读取数据，也可以从当前文件的尾部追加写入数据。当文件不存在时，创建新的文件用于文件尾写入。

## 17.3.2 文件关闭函数fclose

使用完一个文件后应该去关闭它，以免它再被误用，造成数据丢失。所谓“关闭”就是使文件指针变量不指向该文件，以后不能再通过该指针对其相连的文件进行读写操作。如果需要进行读写操作，要再次打开该文件。

文件关闭函数**fclose**的调用格式为：

**fclose**(文件指针);

## 17.4 文件的读写操作

当文件以合适的方式打开以后，就可以对其进行读写操作。**C**语言提供了丰富的数据读写函数，可以按字符读写，可以按行读写，也可以按指定长度的数据块进行读写，还可以进行格式化读写。这些函数都包含在头文件**stdio.h**中。

## 17.4.1 字符读写函数

字符读写函数在处理文件中的数据时，是以字符为单位进行读写的，即每次只读写一个字符。它常用来处理文本文件，但也可以处理二进制文件。

### 1. 读取字符函数 **fgetc**

读取字符函数 **fgetc** 的调用格式为：

**fgetc**(文件指针);

### 2. 写入字符函数 **fputc**

向文件中写入字符函数 **fputc** 的调用格式为：

**fputc**(字符，文件指针变量);

## 17.4.2 字符串读写函数

字符串读写函数是将文件中的数据以字符串为单位进行处理的，即每次一个字符串。字符串读写函数所处理的文件是文本文件，但也可以是二进制文件。

### 1. 读取字符串函数fgets

从文件中读取字符串函数fgets的功能是从指定的文件中读一个字符串到字符数组中，函数调用的形式为：

fgets(字符数组名, n, 文件指针);

例如：fgets(str, n, fp)

### 2. 写入字符串函数fputs

文件写入字符串函数fputs的功能是把一个字符串写入到指定的文件中。其调用形式为：

fputs(字符串, 文件指针)

例如：fputs("abcd", fp);



使用文件字符串读写函数时，需要以下几点：

**(1)** 这两个函数主要用于处理文本文件，也可以用来处理二进制文件，每次读写的是一个字符串。

**(2)** 从文件中读取字符串时，不是用字符串结束标记符'\0'来控制字符串结束，而是用组字符串的字符数目或者回车换行符'\n'来控制字符串结束。

**(3)** 当正确地读或写一个字符串后，文件的文件指针会自动后移一个字符串的位置。

## 17.4.3 数据块读写函数

### 1. 读取数据块的函数**fread**

**fread()**函数用来从指定文件中读一个数据块，该函数的调用格式为：

**fread(buffer,size,count,fp);**

### 2. 写入数据块的函数**fwrite**

**fwrite**函数用来将一个数据块写入文件，该函数的调用格式为：

**fwrite(buffer,size,count,fp);**

## 17.4.4 格式数据读写函数

### 1. 格式数据读取函数**fscanf**

格式数据读取函数**fscanf**类似与**scanf**函数，两者都是格式化输入函数，不同的是**scanf**函数的作用对象是终端键盘，而**fscanf**函数的作用对象是文件。**fscanf**函数调用的一般格式为：

**fscanf**(文件指针，格式控制，输入列表)

### 2. 格式数据写入函数**fprintf**

格式数据写入函数**fprintf**类似于格式输出函数**printf**，两者都是格式化输出函数，只不过两者的作用对象不同，**fprintf**函数输出到文件，**printf**函数输出到终端。**fprintf**函数调用的一般格式为：

**fprintf**(文件指针，格式控制，输入列表)

## 17.5 文件的定位

**C**语言中，打开文件时会产生一个文件指针指向文件的头，在读取文件时，需要从文件头开始，每次读写完一个数据后，该位置指针会自动指向下一个数据的位置。为了能够从文件中直接读取某个数据，系统提供了能将文件内部指向直接定位到某个字节上的函数。

## 17.5.1 文件头定位函数rewind

文件头定位函数**rewind**的作用是将文件位置指针返回到文件指针变量指向的文件的开头。

函数**rewind**调用的一般格式为：

**rewind**(文件指针)

## 17.5.2 文件随机定位函数fseek

文件头定位函数rewind是将文件位置指向文件的开头，要读取某个数据需要从头开始，不方便，C语言提供了文件随机定位函数fseek能将文件位置指针按需要移动到任意位置，可以实现对文件的随机读取。

文件随机定位函数fseek的一般调用格式为：

fseek(文件指针，位移量，起始位置)

表 17-2 起始位置取值含义

整数	符号常量	对应的起始位置
0	SEEK_SET	文件开头
1	SEEK_CUR	文件指针的当前位置
2	SEEK_END	文件末尾

fseek(fp, 10L, 0);

fseek(fp, 20L, 1);

fseek(fp, -30L, 2);

## 17.5.3 测试当前位置函数ftell

在对文件进行读写时，特别是多次调用随机定义函数**fseek**以后，文件位置指针的值经常发生变化，很难确定其当前的位置。**C**语言定义的了测试当前位置的函数**ftell**。

测试当前位置函数**ftell**调用的一般格式为：

**ftell**(文件指针)

## 17.6 文件的检测

**C**语言中，对文件的检测主要是对文件末尾、读写出错等方面讲行的检查和测试，**C**语言常用的文件检测函数有文件末尾检测函数**feof()**、文件读写出错检测函数**ferror()**等。



## 17.6.1 文件末尾检测函数feof

在文本文件中，**C**语言规定**EOF**为文件结束标志，**EOF**的值为**-1**，因为在**ASCII**码表中没有**-1**所对应的字符。但在二进制文件中，**-1**可能为有效数据，就不能用**EOF**来作为文件结束标志。

**C**语言专门定义了**feof**函数作为二进制文件的结束标志，也可以作为文本文件的结束标志。

文件末尾检测函数**feof**的一般调用格式为：

**feof**(文件指针)

## 17.6.2 文件读写出错检测函数ferror

在程序执行过程中，特别是文件读写过程中，会出现一会不可预见的错误。**C**语言定义了文件读写出错检测函数**ferror**。

读写文件出错检测函数**ferror**函数的一般调用格式：  
**ferror**(文件指针);

## 17.6.3 清除文件末尾和出错标志函数 `clearerr`

在文件打开时，出错标志置为**0**，一旦文件读写过程出现错误，错误标志被置为非**0**值，直到同一文件调用**`clearerr`**函数或**`rewind`**函数才重新置**0**。

出错标志函数**`clearerr`**调用的一般格式为：

**`clearerr`**(文件指针);

**`clearerr`**函数的功能是用于将文件指针变量指向的文件的出错标志和文件结束标志设置为**0**值。

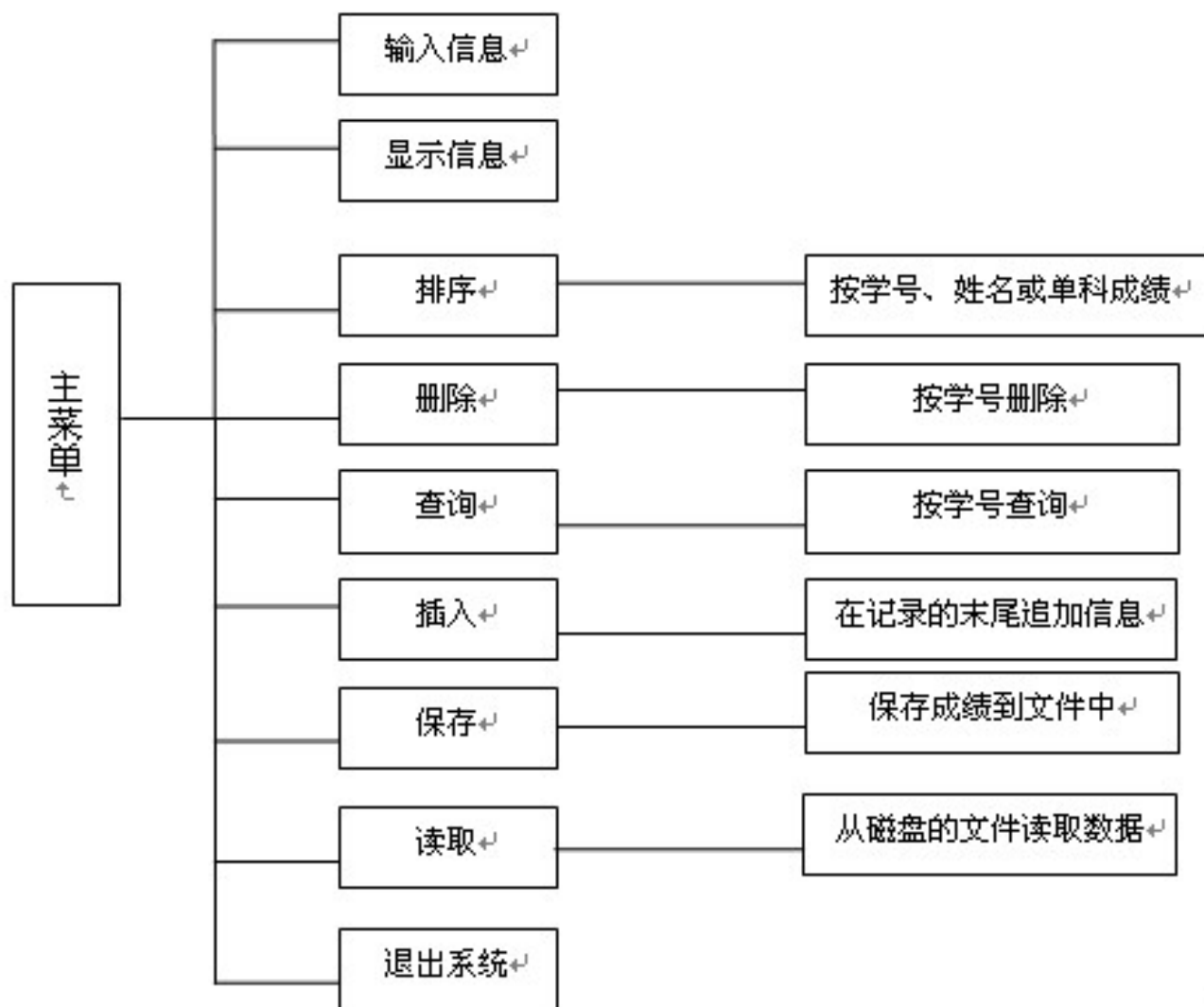
## 第18章 学生成绩管理系统设计

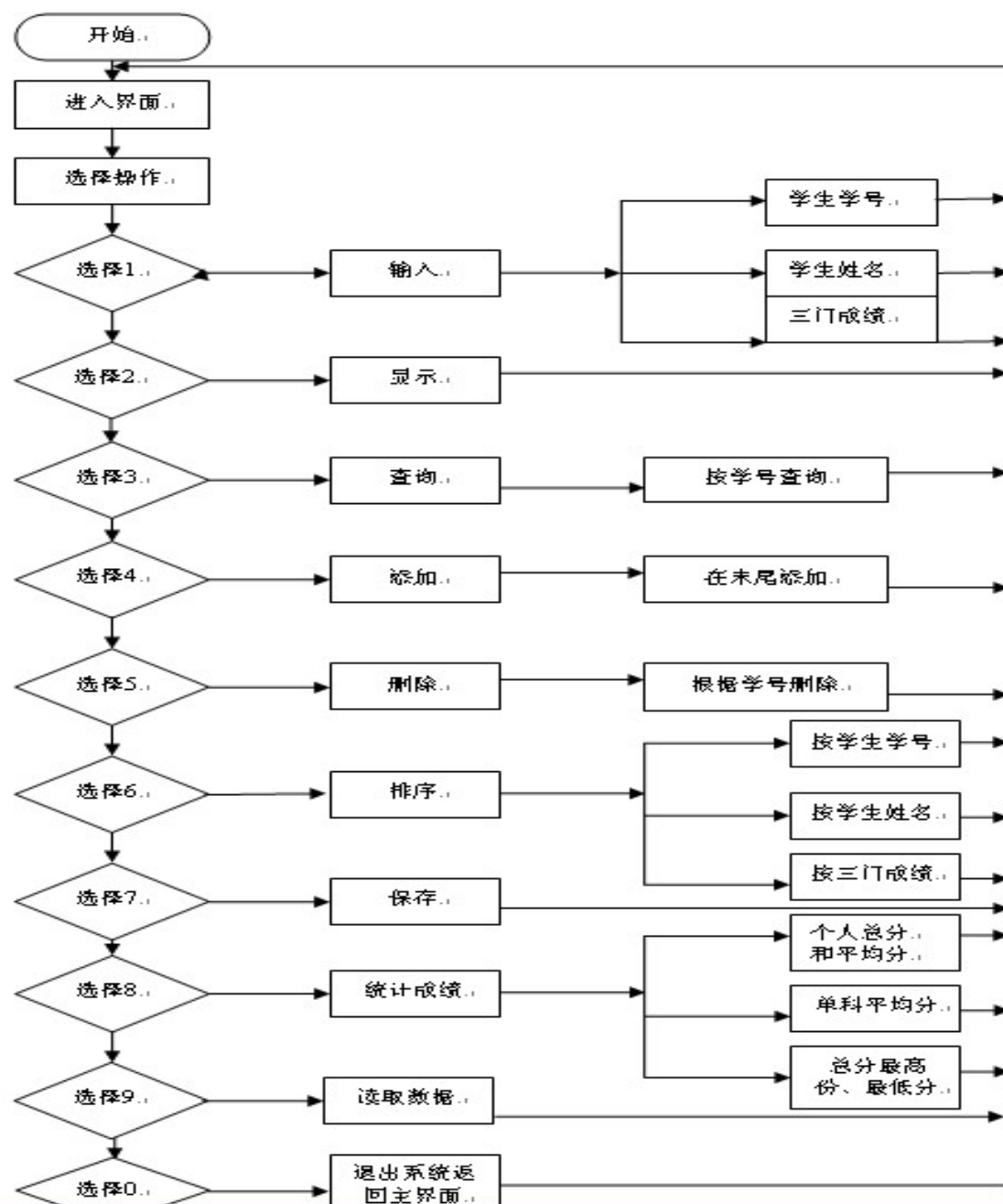
本篇运用**C**语言来设计一个学生的成绩管理系统，整个系统综合运用我们前几章所学习的**C**语言的知识，如结构化程序设计、数组、函数、结构体等等，在复习巩固**C**语言的基础知识的基础上，进一步加深对**C**语言编程的理解和掌握。利用所学知识，理论和实际结合，采用模块化的结构，锻炼学生综合分析解决实际问题的编程能力；使读者对**C**语言有更加深刻的了解与认识。

## 18.1 学生成绩管理系统功能

本系统实现的功能：

- (1) 录入学生的成绩，
- (2) 输出学生的成绩
- (3) 添加学生的成绩信息
- (4) 删除指定学生的成绩信息
- (5) 按照要求对学生成绩信息进行排序
- (6) 根据学号查询指定学生的成绩
- (7) 将学生的成绩信息保存到文件





## 18.2 功能模块的描述

### 18.2.1 数据结构

结构体说的定义如下：

```
struct scorenode  
{  
int number;           /*学生学号*/  
char name[10];        /*学生姓名*/  
float chinese;       /*语文成绩*/  
float mathematic;    /*数学成绩*/  
float english;      /*英语成绩 */  
struct scorenode *next;  
};  
typedef struct scorenode score;
```



## 18.2.2 main() 主函数

程序采用模块化设计，主函数是程序的入口，各模块独立，可分块调整，均由主函数控制。采用**while**死循环和**switch**分支语句编写菜单选择控制各个模块的功能，每个模块的功能由简单的基本函数构成。

### 18.2.3 score \*creatlink() 创建动态链表

由于记录并不是一次性全部输入，而是随时添加和删除的，而预先开辟的空间数往往大于实际的记录数，浪费内存空间，因此使用动态空间开辟函数**malloc()**为输入的数据动态分配内存空间。

## 18.2.4 void print(score \*head) 显示学生信息

使用参数**head**传递链表的首地址，首先判断链表是否为空，如果为空，则输出提示信息；如果不为空，设一个指针变量**p**，先指向第一个结点，输出**p**所指的结点，然后使**p**后移一个结点，再输出，直到链表的尾结点。

## 18.2.5 score \*add(score \*head, score \*stu) 向链表中添加学生数据

该函数有两个数，**head**头结点指向链表的首地址，**stu**指向新建立的结点，向其中输入数据，然后添加到链表上，最后根据学生的学号进行排序。

## 18.2.6 score \*search(score \*head) 查询学 生成绩

由于在向链表中输入数据、添加数据的时候，已经对链表按学号从大到小排好序了，因此在进行查找时，只需要从链表的表头开始进行查询。如果链表为空，则直接输出提示信息；如果链表不为空，则按输出的学号进行查询，查询成功就输出该学号学生的成绩，否则输出提示信息。

## 18.2.7 score \*dele(score \*head) 删除数据

该函数根据输入学生的学号，在链表中进行查找如果有匹配的，就将该学号的学生信息删除掉。最后返回删除后的链表的头结点。

## 18.2.8 score \*sortdata(score \*head)对数据进行排序

该函数提供了几种排序方法，可以按照学生的学号进行排序，按照学生的姓名，或者按照学生的单科成绩进行排序。使用**switch-case**语句根据用户的选择，判断是按照学号、姓名或是单科成绩使用交换法进行排序。

## 18.2.9 save(score \*p1) 保存数据

在程序中的数据输入和输出是以终端为对象的，当程序关闭后，数据也就丢失了，所以为了能随时查阅数据，必须将数据输出到磁盘文件上保存起来，使用时从磁盘中读入到内存中，这就用到了磁盘文件的读写操作。



## 18.2.10 `score *load(score *head)` 从文件中读取数据

为了程序关闭后丢失，我们将数据保存到磁盘文件中，下一次对已经有的数据进行的时候可以直接从文件中读取数据进行操作。

## 18.2.11 score \*statistics(score \*head) 成绩统计

该函数主要实现了对学生成绩统计的几种方式：统计个人总分和平均分、统计单科平均分、统计总分最高分和最低分。

## 18.2.12 `int menu(int k)` 菜单

该函数主要提供一个系统显示的界面，系统模块进行介绍，便于用户进行操作。

## 18. 2. 12 用到的头文件和全局变量

```
#include <malloc.h>
```

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

```
#include <stdlib.h>
```

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

```
#define LEN sizeof(struct scorenode)
```

```
#define DEBUG
```

```
int n,k;
```

**/\*n,k为全局变量，本程序中的函数均可以使用它,分别用于记数和标记\*/**

## 18.3 程序代码

根据上一节对学生成绩管理系统的数据结构的模块功能的分析，并列出了实现各个模块功能的函数以及它们的程序执行**N-S**图，下面是实现各个函数的程序代码以及在程序运行进的运行结果。

## 18.3.1 主函数main代码

主函数main功能是通过调用  
**creat,search,dele,add,print,ststatistics,save,sortdata**等函数，实现学生成绩查询系统功能。

## 18.3.2 创建链表函数creatlink

函数**creatlink**的功能是创建链表，此函数带回一个指向链表头的指针。函数体中使用了**goto**语句，方便程序执行过程中的跳转。

### 18.3.3 显示学生信息函数print

函数**print**的功能是显示学生成绩，即将所建立的学生成绩链表打印出来。如果链表不为空时，逐个打印出学生的学号，姓名和各科成绩。

}



### 18.3.4 向链表中添加学生数据函数add

函数**add**的功能是向已经建立的链表中追加学生资料，并且将所有学生资料按学号排序。

### 18.3.5 查询学生成绩函数search

函数**search**的功能是从链表中查询输入学号的学生信息。

## 18.3.6 删除数据函数delete

函数**delete**的功能是删除输入学号的学生信息。

## 18.3.7 对数据进行排序函数sortdata

函数**sortdata**的功能是对链表中的数据按照一定的要求进行排序。本函数提供了几种排序方法，使用**switch-case**语句根据用户的选择，判断是按照学号、姓名或是单科成绩使用交换法进行排序。此函数带回一个指向链表头的指针

## 18.3.8 保存数据函数save

函数**save**的功能是保存学生的资料到磁盘中，在程序关系以后，下次使用时不会丢失。

## 18.3.9 从文件中读取数据函数load

函数**loadfile**的功能是从文件读入学生记录。当把学生记录保存到磁盘上后，下次使用时还需要从保存的文件中读取。

## 18.3.10 成绩统计函数statistics

函数**statistics**的功能是统计学生成绩，该函数主要实现了对学生成绩进行统计个人总分和平均分、统计单科平均分、统计总分最高分和最低分的操作。