

《C 程序设计》谭浩强第三版知识概括

说明：

以下的內容是对《C 程序设计（第三版 谭浩强著）》知识点的概括，主要是提取全书详细的知识点。我所呈现给大家的内容绝大部分是图表的形式，结构清晰、层次分明。一些上机操作易出错的细节也在具体内容中作了强调，其中包含了本人的一些上机实践经验。**对于要参加国家二级计算机 C 语言等级考试的学生特别有帮助。**对初学 C 语言的人可作为工具书来参考。另外我对原书中 3 到 6 章的知识点顺序稍作调整重组。值得注意的是，《C 程序设计（第三版 谭浩强著）》书中对 C 编译系统并未指定，而**以下内容完全是按照 Visual C++6.0 的编译系统来写的**，所以涉及的具体的情况均是按 Visual C++6.0 的编译系统给出的。这也正是特别适用参加国家二级计算机 C 语言等级考试学生的一个原因。

Mr. Hope

目 录

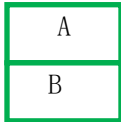
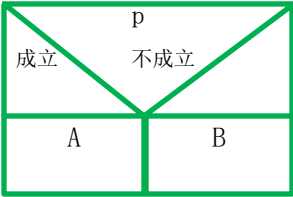
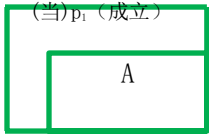
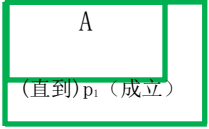
| | | |
|-------|--------------------|----|
| 第一章 | C 语言的特点 | 1 |
| 第二章 | 算 法 | 2 |
| 第三章 | 数据类型 | 4 |
| 第四章 | 四种基本运算 | 7 |
| 第五、六章 | 程序的三种基本结构 | 10 |
| 第七章 | 数 组 | 18 |
| 第八章 | 函 数 | 22 |
| 第九章 | 预处理命令 | 28 |
| 第十章 | 指 针 | 30 |
| 第十一章 | 结构体与共用体 | 37 |
| 第十二章 | 位运算 | 44 |
| 第十三章 | 文 件 | 47 |
| 附录 | C 语言的运算符和结合性 | 50 |

第一章 C 语言的特点

一种语言之所以能存在和发展，并具有较强的生命力，总是有其独特之处。C 语言的主要特点概括如下：

- (1) 语言简洁、紧凑，使用方便、灵活。
- (2) 运算符丰富。
- (3) 数据类型丰富。
- (4) 具有结构化的控制语句。用函数作为程序的模块单位，便于实现程序的模块化。
- (5) 语法限制不太严格，程序设计自由度大。
- (6) C 允许直接访问物理地址，能进行位 (bit) 操作，能实现汇编语言的大部分功能，可以直接对硬件进行操作。
- (7) 生成目标代码率高，程序执行效率高。C 语言一般只比汇编程序生成的目标代码率低 10%~20%。
- (8) 用 C 语言编写的程序移植性好（与汇编语言相比）。

第二章 算 法

| 目 录 | 内 容 |
|-----------|--|
| (一) 算法的概念 | 广义的说,为解决一个问题而采取的方法和步骤,就成为算法。不过这里只限于计算机算法,即计算机能执行的算法。计算算法可分为两大类:数值运算算法和非数值运算算法。 |
| (二) 算法的特性 | 1) 有穷性。一个算法包含的操作步骤应有限。 2) 确定性。算法中的每一个步骤应当都是确定的。 3) 有零个或多个输入。 4) 有一个或多个输出。没有输出的算法是没有意义的。 5) 有效性。算法中的每一个步骤都应有效的执行,并得到确定的结果。 |
| (三) 算法的表示 | 1) 用自然语言表示算法。 2) 用流程图表示算法。 3) 3 种基本结构和改进的流程图。 I 3 种基本机构:顺序结构、选择结构和循环结构。 II 3 种结构的共同特点: A. 只有一个入口; B. 只有一个出口; C. 结构内的每一部分都有机会被执行到; D. 结构内不存在“死循环”。 III 用 N—S 流程图表示算法,流程图符号: A. 顺序结构  B. 选择结构  C. 循环结构   4) 用伪代码表示算法: 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。 5) 用计算机语言表示算法。 |

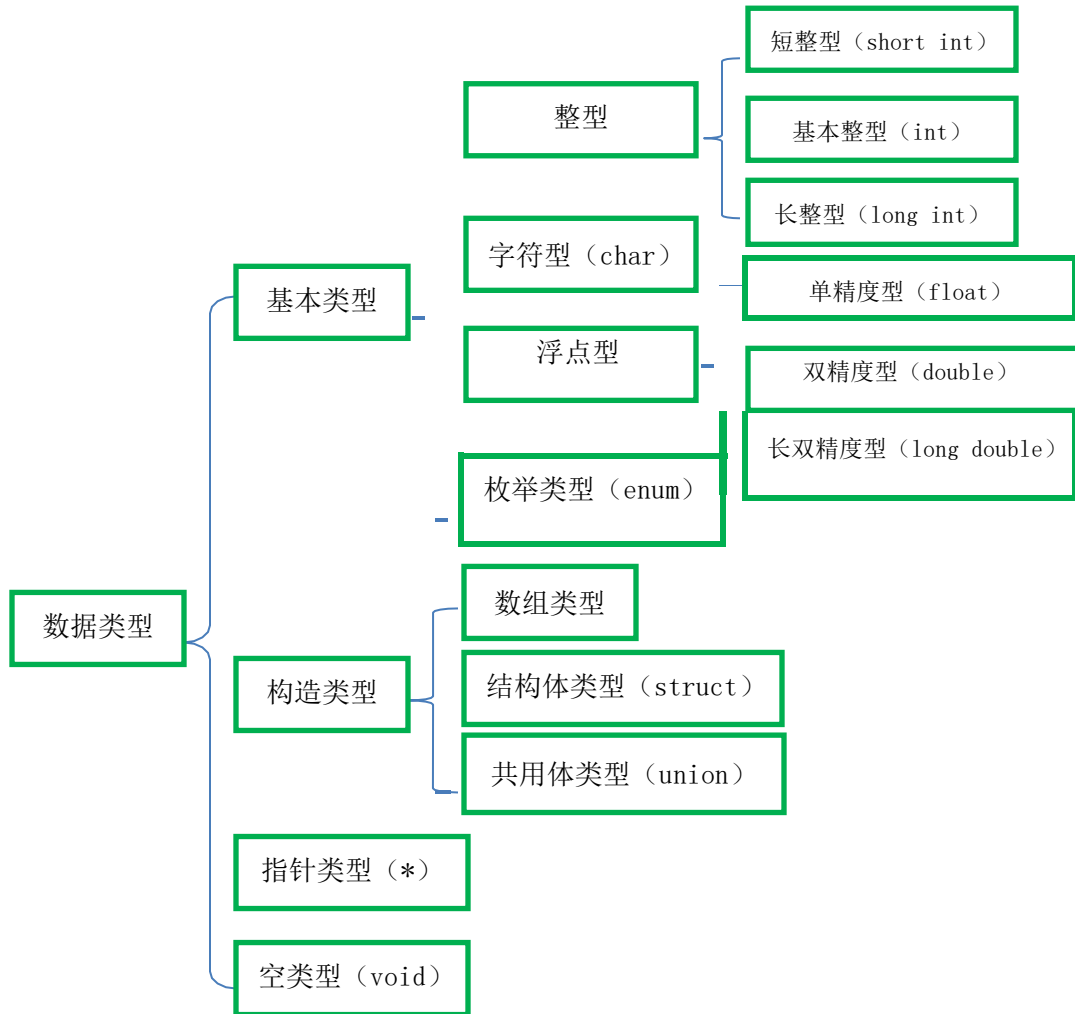
| | |
|--------------|---|
| (四) 结构程序设计方法 | 为得到结构化程序的需实施： 1) 自顶向下； 2) 逐步细化； 3) 模块化设计； 4) 结构化编码。 |
|--------------|---|

第三章 数据类型

一、概述

(一) 数据类型

C 语言提供以下的数据类型，由这些数据类型可以构造出不同的数据结构。



(二) 常量与变量

| 名称 | 举例 | 习惯 | 声明 | 使用 |
|----|-------------|-------------------|-----------------|--------|
| 常量 | 'a' | 用#define 声明时标志符大写 | #define 标识符 字符串 | |
| 变量 | a (int a;) | 标识符小写 | 类型 标识符; | 先定义后使用 |

标识符：只能由字母、数字和下划线组成，且第一个不能为数字。大小写敏感。

合法的用户标识符：不与系统关键字相同的标识符。

二、最常见的三种基本类型

| 目录 | 内 容 | | |
|---------|--|-------------|--|
| (一) 整型 | 类 型 | 位数(字节) | 范 围 |
| | [signed] int | 32 (4) | $-2^{31} \sim (2^{31}-1)$ |
| | unsigned int | 32 (4) | $0 \sim (2^{32}-1)$ |
| | [signed] short [int] | 16 (2) | $-32768 \sim 32767 \{-2^{15} \sim (2^{15}-1)\}$ |
| | unsigned short [int] | 16 (2) | $0 \sim 65535 \{0 \sim (2^{16}-1)\}$ |
| | long [int] | 32 (4) | $-2^{31} \sim (2^{31}-1)$ |
| | unsigned long [int] | 32 (4) | $0 \sim (2^{32}-1)$ |
| (二) 浮点型 | ◇ 两种形式 | ➤ 小数形式 | 123.0 |
| | | ➤ 指数形式 | <ul style="list-style-type: none"> ● 1. 23e2 和 12. 3E1 都代表 123.0 (e 前必须有数字, 后必须为整数) ● 规范化指数形式: e (E) 前小数部分中, 小数点前有且只有一位非零数字。 ● 浮点数用指数形式输出时按规范化指数形式输出。 |
| | ◇ 分类 | 类 型 | 位数(字节) |
| | | float | 32 (4) |
| | | double | 64 (8) |
| | | long double | 64 (8) |
| |  由于浮点型在有效数字外的数字据随机性, 应当避免将一个很大的数和一个很小的数直接相加或相减, 否则就会丢失“小数”。 | | |

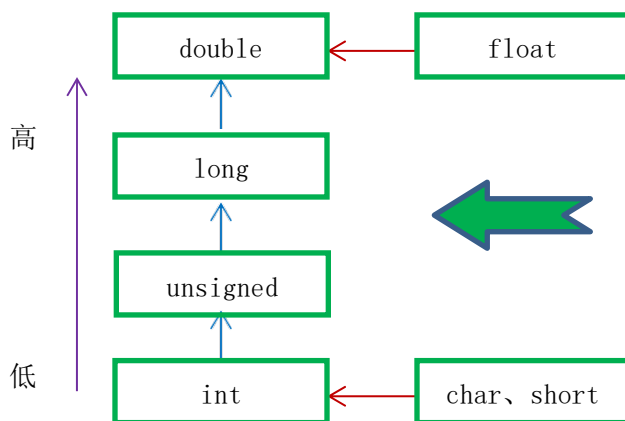
| | | | | | |
|---------|----------|---|---|--------------------------|---------|
| (三) 字符型 | 1. 字符型常量 | ◆ 表示 | 用单撇号括起来的一个字符。如： ‘a’ 、 ‘X’ 、 ‘?’ 、 ‘\$’ 等。 | | |
| | | ◆ 转义字符 | 形式 | 含 义 | ASCII 码 |
| | | | \n | 换行 | 10 |
| | | | \t | Tab 键，8 个空格 | 9 |
| | | | \b | Backspace，退格 | 8 |
| | | | \r | Enter，将当前位置移到本行开头 | 13 |
| | | | \f | 换页，移到下页开头 | 12 |
| | | | \\ | 代表字符 “\” | 92 |
| | | | \' | 代表单引号字符 | 39 |
| | | | \” | 代表双引号字符 | 34 |
| | | | \ddd | 1 到 3 位八进制所代表的字符 | |
| | | | \xhh | 1 到 2 位十六进制所代表的字符，x 不可大写 | |
| | 2. 字符变量 | 定义形式 | | char 字符变量名； | |
| | | 说明 | I. 字符变量用来存放字符，它只能放一个字符。 II. 它只占一个字节。另外它可以、与整型整数数据之间通用。在一定范围内两种数据可以相互输入输出。而且 C 语言允许字符数据与整数直接进行算术运算。 | | |
| | 3. 字符串变量 | ◆ 用一对双撇号括起来的字符序列。注意与字符常量的区别。 ◆ 不能把一个字符串常量赋给一个字符常量。 ◆ C 中没有专门的字符串常量，如果想将一个字符串放在变量中以便保存，必须用字符的数组。 | | | |

第四章 四种基本运算

一、算术运算

| 目录 | 内 容 | |
|-------------|--|---|
| (一) 算术运算符 | 运算符 | 说 明 |
| | + | 加法或正运算符 |
| | - | 减法或负运算符 |
| | * | 乘法运算符 |
| | / | 除法运算符，如 $5/3$ ，若两边均为整型则 $5/3=1$ |
| | % | 模（取余）运算符，%两侧应均为整型数据，如 $7\%4=3$ |
| (二) 优先级和结合性 | ◇ 优先级 | 多个不同的运算符出现时，进行运算的先后顺序叫做运算符的优先级。以上各运算符的优先级可查附录。 |
| | ◇ 结合性 | 若干个相同的运算符同时出现时，进行运算的顺序叫做运算符的结合性。算术运算符的结合性为“自左向右”。 |
| (三) 强制类型转换 | <p>I. 算术表达式中的数据类型不不同时，按其中较高级别①的数据类型为准对级别低的一侧数据进行转换，然后再计算，计算结果为表达式结果（只有一级）。如果表达式为多级运算，则每一级运算均按上述过程进行，最终结果为表达式结果。</p> <p>II. 算术运算要想得到指定的类型的数据结果，常常需要进行类型转换才能达到；另外有些数据只有进行类型转换后才能满足某些运算的要求。例如%两边需为整型数据，$(int) 5.4\% (int) 3.1=2$使用的一般形式为：</p> <p style="text-align: center;">（类型名）（表达式）</p> <p>作用是将表达式结果转换为指定的类型。值得注意的是强制类型转换并不会把变量的类型改变，只是在运算时形成一个中间数据。</p> <p>III. 一个整形数据后跟一个字母u或U，该数据被认为是unsigned int 类型，而后跟 l 或 L，则认为是 long int 类型。</p> | |

◆ ① 数据类型高低：



- I. 红色（横向）箭头表示必定的转换。如两个 char 型相加也要先转换为 int 型再相加。
- II. 蓝色（纵向）箭头表示运算对象不同类型时的转换方向。
- III. 不过箭头的方向并不表示转换是这样一级一级进行的。类型不同时是直接（一步）转换为目标（高级）类型的。

二、赋值运算

| 目 录 | 内 容 | | |
|-------------|----------------|---|---|
| (一) 运算符 | 名 称 | 符 号 | 说 明 |
| | ✧ 赋值运算符 | = | 运算符左侧是变量 ，右侧是表达式运算结果是将表达式的值赋给左侧变量。当右侧运算结果与左边变量类型不同时要进行类型转换。 |
| | ✧ 复合的赋值运算符 | $\text{+=, -=, *=, /=, \%}, \text{<<=, >>=, \&=, ^=, =}$ | 举例说明使用方法： $a+=b$ 相当于 $a=a+b$ ； b 可以是表达式，如 $x\%=y+3$ 与 $x\%=(y+3)$ 等价均可表示为 $x=x\%(y+3)$ 。另外后五个是关于位运算的。 |
| | ✧ 自增、自减运算符 | $++$ $--$ | $++i, --i$ （在使用 i 之前，先对 i 的值加（减）1）； $i++, i--$ （在使用 i 之后，使 i 的值加（减）1）。 自增自减只能针对变量使用。当其用在表达式中时有些问题要注意①。 |
| | ✧ 逗号运算符 | , | 该运算符又被称为“顺序求值运算符”。其一般形式为： 表达式 1, 表达式 2, …, 表达式 n 逗号表达式的结果为表达式 n 的值。逗号运算符的优先级是所有运算符中最低的②。 |
| (二) 类型转换 | 左值③类型 | 右值类型 | 转换方法 |
| | 整型 | 实型 | 舍弃小数部分 |
| | 实型 | 整型 | 数值不变，添加小数位 |
| | float | double | 截取前面 7 位有效数字，但应注意数值范围 |
| | double | float | 数值不变，有效位扩展到 16 位 |
| | 整型 | 字符型 | 将字符数据放到整型变量低 8 位，并进行符号扩展 |
| | 字符型 | 整型 | 只将整型低 8 位原样送到字符型变量存储位置 |
| | long | signed int | 都是 32 位，直接原样转送 |
| | long | unsigned int | 都是 32 位，直接原样转送，可能会出现错误 |
| | 非 unsigned 型整型 | unsigned int | 长度相同时原样转送；不同时则低位转送，此时要注意数据范围 |
| | unsigned | 非 unsigned 型整型 | 长度相同时原样转送，不同时则低位转送，但均有可能出错 |



①在表达式或含有表达式的语句（包括函数调用语句）中凡是在使用前要自增自减的均立即执行，但是使用后的自增自减要在整个表达式或语句执行完后再统一执行，执行的个数不变。另外函数调用中是自右至左执行的。

- ②各运算符的优先级和结合性均可以查附录得知。
- ③赋值运算符左侧的标识符称为左值，右侧的表达式成为右值。

三、关系运算

| 目 录 | 内 容 | | | | |
|-----------|---|-----|------|-------|--------|
| (一) 关系运算符 | 序号 | 运算符 | 名 称 | 优先级 | 结合性 |
| | ① | < | 小于 | 相同（高） | 都是自左至右 |
| | ② | <= | 小于等于 | | |
| | ③ | > | 大于 | | |
| | ④ | >= | 大于等于 | | |
| | ⑤ | == | 等于 | 相同（低） | |
| | ⑥ | != | 不等于 | | |
| (二) 关系表达式 | 用关系运算符将表达式连接起来的式子，称为关系表达式。 关系表达式的值是一个逻辑值， 非“真”即“假”。 C 逻辑运算中“1”代表“真”；“0”代表“假”。 | | | | |

四、逻辑运算

| 目 录 | 内 容 | | | | |
|-----------------------|--|-----|-----|------|------|
| (一)逻辑运算符 | 序号 | 运算符 | 名 称 | 优先级 | 结合性 |
| | ① | && | 逻辑与 | 三者居中 | 自左至右 |
| | ② | | 逻辑或 | 三者居底 | |
| | ③ | ! | 逻辑非 | 三者居首 | 自右至左 |
| (二)逻辑表达式 | 用逻辑运算符将表达式连接起来的式子，称为逻辑表达式。 逻辑值判断：非 0 为“真”，0 为“假”。 | | | | |
| (三)本章涉及的运算符 的优先级排列 | <div>！（逻辑非）（高）</div> <div>算术运算符</div> <div>关系运算符</div> <div>&&和 （逻辑与和逻辑或）</div> <div>赋值运算符（低）</div> | | | | |

第五、六章 程序的三种基本结构

一、顺序结构

| 目 录 | 内 容 | | | |
|--------------|---------------|---------|------------------------------|--|
| (一) C 语句概述 | 分 类 | | 说 明 | |
| | 1) 控制语句 | | C 控制语句只有 9 种。详见附录。 | |
| | 2) 函数调用语句 | | 由一个函数调用加一个分号构成。 | |
| | 3) 表达式语句 | | 由一个表达式加一个分号构成。 | |
| | 4) 空语句 | | 只有一个分号的语句。 | |
| | 5) 复合语句 | | 用 { } 把一些语句括起来成为复合语句（又称分程序）。 | |
| (二) 基本输入输出函数 | 分 类 | 函数名 | 使用格式 | 说 明 |
| | 一 字符输入输出函数 | putchar | putchar (字符) | 可以向终端输出一个字符（包括转义字符）或整型量。 |
| | | getchar | getchar (字符) | 从终端接收一个字符，并可将其赋给一个字符变量或整型变量。 |
| | 二 格式输入输出函数 | printf | printf (格式控制, 输出表列) | I. 格式控制包括格式说明①和普通字符②（包括转义字符）两部分。整体是一个字符串形式，要用双引号括起。 II. 输出表列是要输出的数据，可以是表达式、常量或变量，用逗号隔开。 |
| | | scanf | scanf (格式控制, 地址表列) | I. 格式控制的内容同上，但在输入数据时要严格按格式输入③。 II. 地址表列是由若干个地址组成，可以是变量地址，或字符串首地址。 |
| | | | | |

注：① 格式说明内容在输出时将用输出表列的数据按格式替换，在输入时作为读取数据的格式。格式说明的个数一般应与后面输出表列中数据个数或地址表列的个数相等。一个格式说明的构成：

| | | |
|---|--------|------|
| % | 附加格式字符 | 格式字符 |
|---|--------|------|

其中绿色部分不可省，蓝色部分可按需选择或省或留。三者相互无间隔。对格式字符和附加格式字符有以下两表说明：

表 1，格式字符

| 数据类型 | 格式字符 | 说 明 |
|------|-------|---|
| 整型 | d 或 i | 带符号十进制整数（输出正数时不带符号） |
| | u | 无符号十进制整数 |
| | x 或 X | 无符号十六进制整数（输出时不带前导符 0x）。输出时小写表 a~f 用小写输出，大写则用大写字母输出；输入时大小写无区别 |
| | o | 无符号八进制整数（输出时不带前导符 0）【地址格式符】 |
| 字符型 | c | 字符形式，只输入或输出一个字符 |
| 字符串 | s | 输入输出字符串。输入时从第一个非空白字符（空格）开始到第一个空白字符结束 |
| 浮点型 | f | 小数形式实数（包括单双精度）。输出时隐含 6 位小数；输入时也可用指数形式 |
| | e 或 E | 输入时与 f 作用相同，大小写无区别。输出时是按指数形式，大小写区别在于输出结果中 e 的大小写 |
| | g 或 G | 输入时与 f 作用相同，大小写无区别。输出时选用%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，大写时用指数形式输出 e 大写 |

表 2，附加格式字符

| 字 符 | 说 明 |
|--------|--|
| l 或 L | 用于长整型整数（在 d 或 i、x 或 X、u、o 前）或 long double 型数据（在 f、e 或 E、g 或 G 前） |
| h | 用于短整型数据 |
| m（正整数） | 数据最小宽度。输出数据域宽大于 m 时，突破 m 限制；域宽小于 m 时，左补空格。输入时按域宽读取数据 |
| n（正整数） | 输出时，对实数表小数位数；对字符串表截取的字符个数。对输入无效 |
| - | 输出时数字或字符在域内左靠。对输入无效 |
| * | 表示本输入项（该格式说明的长度）在读入后不赋给变量 |

补充：

- l. 一个格式说明中格式字符只能有一个，而附加格式字符可以多个。

II. 对附加格式字符的使用详细说明（见下页图表）：

| 分类 | 类型 | 可用格式 | 说 明 |
|----|-----|--|--|
| 输出 | 整型 | m, -m | 按附加字符说明表格操作即可。 |
| | | mh, -mh | |
| | | m1 或 mL, -m1 或 -mL | |
| | 字符串 | m, -m | 按附加字符说明表格操作即可。 |
| | | m.n, -m.n | I. $m \geq n$ 时, 占 m 列, 但只取左 n 个字符, 看有无“-” 选择右补还是左补空格。 II. $m < n$ 时, m 自动取 n 值, 保证字符正常输出。 |
| | | .n | III. 无 m 时, 取 n 个字符。 |
| | 浮点型 | m, -m | 按附加字符说明表格操作即可。 |
| | | m1 或 mL, -m1 或 -mL | |
| | | m.n, m.n1 或 m.nL, -m.n1 或 -m.nL | |
| | | .n, .n1 或 .nL | |
| 输入 | 整型 | m, *m | 按附加字符说明表格操作即可。 |
| | | mh, *mh | |
| | | m1 或 mL, *m1 或 *mL | |
| | 字符型 | m, *m | 按附加字符说明表格操作即可。 |
| | 浮点型 | m, *m, m1 或 mL, -m1 或 -mL | 按附加字符说明表格操作即可。 |

②普通字符在输出时按原样输出。但是要**输出%**，要连续用两个%表示；要**输出英文的双引号**需在前加反斜杠；要**输出反斜杠**需要连续用两个\表示。不过英文单引号不必前加反斜杠。

③使用 scanf 函数还应注意：

- I. 在%后有*表示跳过它指定的列数，例如：
scanf（“%2d %*3d %2d”，&a,&b）；
输入 12 345 67
结果 a=12, b=67. 345 被跳过。
- II. 输入数据时不能规定精度。
- III. 在“格式控制”字符串中除了格式说明以外的字符，在输入数据时对应位置要输入相同字符。另外：
空格再输入时应保证>=原空格数
- IV. 用“%c”输入字符时，空格和转义字符都作为有效字符输入，例如：
scanf（“%c%c%c”，&c1,&c2,&c3）；
若输入 a b c
结果：c1=a, c2= , c3=b。
正确输入 abc
- V. 在输入数据时，遇到以下情况时认为该数据结束：
 - a, 遇到空格，或按“回车”或“Tab”；
 - b, 按指定宽度结束，如“%3d”，只取 3 列；
 - c, 遇到非法输入。

二、选择结构

| 目录 | 内 容 | | | |
|-------|--------|---|---|---|
| if 语句 | ◆ 基本形式 | 1 | if（表达式）语句 | 说明： I if 后括号中的表达式为真时执行其后的语句，为假则执行 else。 II 表达式真假的判断标准为： （a） 非 0 为真； （b） 0 为假。 III else 不能单独使用，只能是 if 语句的一部分。 |
| | | 2 | if（表达式）语句 1 else 语句 2 | |
| | | 3 | if（表达式 1）语句 1 else if（表达式 2）语句 2 else if（表达式 3）语句 3 ” else if（表达式 n）语句 n else 语句 n+1 | |

| | | | | |
|-----------|---------|--|---|--|
| | ◆ 嵌套 | <pre>if (if () 语句 1 else 语句 2 else if () 语句 3 else 语句 4</pre> <div><div></div>内嵌 if</div> <div><div></div>内嵌 if</div> | | 注意：if 与 else 的配对关系。 else 总是与上面最近一个未配对的进行配对。 |
| | ◆ 条件运算符 | 形式 | 表达式 1? 表达式 2: 表达式 3 | |
| | | 说明 | I 执行顺序：先求解表达式 1，为真（非 0）则求解表达式 2，并将表达式 2 的值作为结果；若为假（0），则求解表达式 3，其值作为结果。 II 表达式 2、3 还可以是赋值表达式或函数表达式。 III 表达式 2 和表达式 3 类型不同时，结果的类型要以其中较高的类型为准，对结果进行转换。 | |
| switch 语句 | 一般形式 | <pre>switch (表达式) { case 常量表达式 1 : 语句 1 case 常量表达式 2 : 语句 2 ” case 常量表达式 n : 语句 n default : 语句 n+1 }</pre> | | |
| | 说明 | I switch 后括号中表达式的值可以是整型、字符型、枚举类型数据。 II 当表达式的值与某一个常量表达式的值相等时，执行该 case 后面的语句；若均未有匹配的，执行 default 后的语句。 III 执行完一个 case 后的语句之后，流程自动转向执行自此之下的所有语句。 IV 每一个 case 后的常量表达式的值须不同，否则出现矛盾。 V 各个 case 的出现顺序不影响结果（前提在语句 1~语句 n 后加了 break 语句）。 VI 多个 case 可以共用一组执行语句。即在需共用的最后一个 case 后写共用语句，之前 case 后的语句省略。 | | |

三、循环结构

| 目录 | 内 容 | |
|--------------|-----|---|
| goto 语句 | 形式 | goto 语句标号; "" 标号: 语句; "" |
| | 说明 | <p>I goto 语句为无条件转向语句。语句标号的定名规则与变量名相同。goto 语句可先前转, 也可向后转。</p> <p>II 结构化程序设计主张限制使用 goto 语句, 但也不是绝对禁止。一般来说, 有两个用途:</p> <p>(a) 与 if 构成循环结构 (一般为当型结构, 可用 while 循环替换)。</p> <p>(b) 从循环体跳到循环体外, 但在 C 语句中已有 break 和 continue 语句来跳出本层循环和结束本次循环。只有需从多层循环跳到外层循环时才用到 goto 语句, 但这种用法不符合结构化原则, 一般不宜采用。</p> |
| while 循环 | 形式 | while (表达式) 语句 |
| | 说明 | <p>I 当表达式为真 (非 0) 时, 执行语句。</p> <p>II 循环体包含一个以上的语句要用花括号括起, 组成复合语句。否则会出现错误。</p> <p>III 循环体中应有使循环体趋向于结束的语句。</p> |
| do,,while 循环 | 形式 | do 循环体语句while (表达式); |

| | | |
|-------------|----|--|
| | 说明 | 先执行一次循环体语句，再判断表达式的真假，当为真时再返回执行循环体语句。 |
| for 循环 | 形式 | for (表达式 1; 表达式 2; 表达式 3) 语句 |
| | 说明 | <p>I 执行流程图</p> <pre> graph TD A[求解表达式 1] --> B{表达式 2} B -- 真 --> C[语句] C --> D[求解表达式 3] D --> A B -- 假 --> E[下一个语句] </pre> <p>II 表达式 1 和表达式 3 可以省略，但表达式 1 后面的分号不能省。</p> <p>III for 语句最简单的应用形式也就是最易理解的形式如下：</p> <ul style="list-style-type: none"> • for(循环变量赋初值; 循环条件; 循环变量增值) 语句 <p>IV for 语句使用灵活，完全可以代替 while 语句。</p> |
| 循环的嵌套 | | 一个循环内又可以包含另一个完整的循环，这就是循环嵌套。内嵌的循环还可以在嵌套循环，这就是多层循环。以上所讲的几种循环是可以相互嵌套的。 |
| break 语句 | 形式 | break; |
| | 说明 | 它用于 while 循环、do„while 循环和 for 循环的循环体中来跳出循环，即提前结束循环。另外在 switch 语句用来跳出 switch 选择。注意用 if 和 goto 构成的循环 break 无法起作用。 |
| continue 语句 | 形式 | continue; |

| | | |
|--|----|---|
| | 说明 | <p>它用于 while 循环、do,,while 循环和 for 循环的循环体中来结束本次循环，即跳过循环体中下面的语句，接着进行下次是否执行循环的判定。同样在用 if 和 goto 构成的循环中 continue 也无法起作用。</p> |
|--|----|---|

第七章 数 组

一、一维数组和二维数组

| 目 录 | 内 容 | | |
|------|-----|--|---|
| 一维数组 | 项目 | 形 式 | 说 明 |
| | 定义 | 类型说明符 数组名[常量表达式]; | (1) 数组名的命名规则遵循标识符命名规则。 (2) 方括号中常量表达式用来表示元素个数，即数组长度。 【注意】 ，下标是从 0 开始的。 (3) 常量表达式可以包括常量和符号常量，但不能包含变量。C 不允许对数组大小进行动态定义。 |
| | 引用 | 数组名[下标] | (1) C 规定只能逐个引用数组元素，而不能一次引用整个数组。 (2) 下标可以是整型常量或整型表达式。 |
| | 初始化 | 1) 在定义数组时对元素赋初值。将数组元素的初值依次放在一对大括号内。 例如：int a[10]={0, 1, 2, 3, 4, 5, 6, 7, 8, 9}; 2) 可以只给一部分元素赋值。例如：int a[10]={0, 1, 2, 3, 4};只给前 5 个赋初值，后 5 个元素值为 0。 3) 如果想使一个数组中全部元素值为 0，可以写成： • int a[10]={0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; 或 int a[10]={0}; 4) 对数组全部元素赋初值时，由于数据个数已经确定，可以不必指定数组长度。例如： • int a[6]={0, 1, 2, 3, 4, 5};可写成 int a[]={0, 1, 2, 3, 4, 5}; | |
| 二维数组 | 项目 | 形 式 | 说 明 |
| | 定义 | 类型说明符 数组名[常量表达式] [常量表达式]; | (1) C 语言允许使用多维数组。 (2) 多维数组原元素在内存中的排列顺序：第一维的下标变化最慢，最右边的下标变化的最快。 |

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|-----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|----|---|---|---|---|---|---|---|---|---|---|----|---|---|--|
| | 引用 | <p style="text-align: center;">数组名[下标] [下标];</p> | <p>(1) 数组元素可以出现在表达式中, 也可以被赋值。</p> <p>(2) 使用数组元素时应保证下标值在已定义的数组大小范围内。</p> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | 初始化 | <p>1) 分行给二维数组赋初值。例如:</p> <ul style="list-style-type: none"> int a[3][4]={ {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} }; 可以将所有的数据写在一个花括号内, 按数组排列顺序对各个元素赋初值。例如: int a[3][4]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}; <p>2) 可以对部分元素赋初值。例如:</p> <ul style="list-style-type: none"> int a[3][4]={ {1}, {5}, {9} }; 相当于 <table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>1</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>5</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>9</td><td>0</td><td>0</td><td>0</td></tr> </table> 也可对各行中的某一元素赋初值。如: int a[3][4]={ {1}, {0, 6}, {0, 0, 11} }; 相当于 <table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>1</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>6</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>11</td><td>0</td></tr> </table> 也可以对第三行赋值, 而对第二行不赋值。如: int a[3][4]={ {1}, {}, {9} }; <p>3) 如果对全部元素赋初值, 则定义数组时对第一维的长度可以不指定, 但第二维不能省。例如:</p> <ul style="list-style-type: none"> int a[3][4]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}; 等价于: int a[][4]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}; 另外定义时也可以只对部分元素赋初值而省略第一维的长度, 但应分行赋初值。例如: int a[][4]={ {0, 0, 3}, {}, {0, 10} }; <ul style="list-style-type: none"> 其各组元素为: <table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>0</td><td>0</td><td>3</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>10</td><td>0</td><td>0</td></tr> </table> | 1 | 0 | 0 | 0 | 5 | 0 | 0 | 0 | 9 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 6 | 0 | 0 | 0 | 0 | 11 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | 0 | |
| 1 | 0 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | 0 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 9 | 0 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 0 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 6 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0 | 11 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0 | 3 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 10 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

二、字符数组

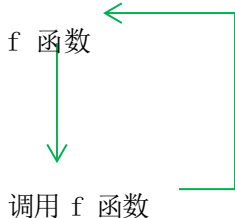
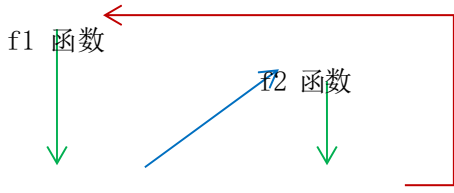
| 目 录 | 内 容 | | |
|--------|---|--|---|
| | 项 目 | 形 式 | 说 明 |
| 一、字符数组 | 定 义 | <code>char 字符数组名[数组长度];</code> | 字符型和整型可以通用，故也可以定义一个整型数组来存放字符数据。 |
| | 初 始 化 | 1) 逐个初始化。 2) 字符串初始化： <code>char 数组名[长度]={字符串};</code> 或 <code>char 数组名[]={字符串};</code> 或作 <code>char 数组名[]=字符串;</code> | 逐个初始化时与其他类型数组的形式类似。 |
| | 引 用 | 可以逐个引用元素。 | |
| | 字 符 串 结 束 标 志 | <ul style="list-style-type: none"> ◆ C 语言规定字符串结束标志为 ‘\0’。即遇到 ‘\0’ 时，表示字符串结束，由它前面的字符组成字符串。 ◆ 字符串常量的最后系统会自动加上 ‘\0’。而对字符数组来说，如果它的长度比赋值个数多，则多出的元素全为 ‘\0’，如果刚好，且没有人为加上 ‘\0’，也是可以的，但用 %s 格式输出该数组时，可能并不按数组长度或实际字符串长度停止。 | |
| | 字 符 数 组 的 输 入 输 出 | 1) 逐个输入输出。用格式符 “%c”。 2) 将字符串一次输入或输出。用格式符 “%s”。 | 注意： (1) 输出字符不包括 ‘\0’。 (2) 用格式符 “%s” 输出遇到 ‘\0’ 才结束。 (3) 若要用一个 scanf 函数输入多个字符串，则用空格作各个字符串的分隔符。但要用这种格式输入一个包含空格的字符串是不行的。 (4) 用 scanf 函数且是 “%s” 格式输入字符串时，字符数组名即表示地址，不能再加取地址符 &。 (5) 关于字符串的一次输入输出，由于字符串结束标志符的原因（第（2）条），有些具体细节上机实践更易领会。 |

| | | | | |
|-----------|----|---------|---------------------------|---|
| 二、字符串处理函数 | 序号 | 函数名 | 使用形式 | 功能介绍 |
| | 1 | puts | puts(字符数组) | 将一个字符串(以 ‘\0’ 结束的字符序列) 输出 到终端。字符串中可以包含转义字符。 |
| | | gets | gets(字符数组) | 从终端 输入 一个字符串到字符数组，并得到一个函数值。该函数值为字符数组的起始地址。 |
| | 2 | strcat | strcat(字符数组 1, 字符数组 2) | 其作用是将字符串 2 连接 到字符串 1 后面，结果放在字符数组 1 中，字符数组 1 应足够大。函数的返回值是字符数组 1 的地址。 |
| | 3 | strcpy | strcpy(字符数组 1, 字符串 2) | 它的作用是将字符串 2 复制 到字符数组 1 中去。同样字符数组 1 应足够大。 |
| | | strncpy | strncpy(字符数组 1, 字符串 2, n) | 用字符串 2 前面 n 个字符去 替换 字符数组 1 的前 n 个字符。注意替换的字符个数不应多于字符数组 1 原有的个数（不包括 ‘\0’ ）。。 |
| | 4 | strcmp | strcmp(字符串 1, 字符串 2) | 它的作用是 比较 字符串 1 和字符串 2: A, 相等, 值为 0; B, 大于, 值为一个正整数; C, 小于, 值为一个负整数。 |
| | 5 | strlen | strlen(字符串) | 作用是测算字符串的 长度 （不含 ‘\0’ ）。值为测算结果。 |
| | 6 | strlwr | strlwr(字符串) | 作用是将字符串中的大写字母换成 小写 字母。 |
| | | strupr | strupr(字符串) | 作用是将字符串中的小写字母换成 大写 字母。 |

第八章 函 数

一、函数

| 目 录 | 内 容 | | |
|---------------|---|---|------------------------------|
| 一· 函数的概述 | 1) C 程序的执行是从 main 函数开始的, 如是在 main 函数中调用其他函数, 在调用后流程返回到 main 函数, 在 main 函数中结束整个程序的运行。 2) 所有函数都是平行的, 即在定义函数时是分别进行的, 是互相独立的。函数间可以互相调用, 但不能调用 main 函数。main 函数是系统调用的。 3) 函数的简单分类 : a) 从用户使用角度: i. 标准函数; ii. 用户自己定义的函数。 b) 从函数的形式来看: i. 无参函数; ii. 有参函数。 | | |
| 二· 函数的定义形式 | ◆ 无参函数定义形式 | ◆ 有参函数定义形式 | ◆ 空函数 |
| | 类型标识符 函数名 () { 声明部分 语句部分 } | 类型标识符 函数名 (形参表列) { 声明部分 语句部分 } | 类型标识符 函数名 () { } } |
| 三· 函数的参数和函数的值 | ◆ 参 数 | ✧ 形参: 在定义函数时在函数名后括号中的变量名称为形参。 ✧ 实参: 在调用函数时, 函数名后括号中的参数称为实参。 ■ 说明: (1) 在未出现函数调用时形参不占内存单元。调用时才分配内存单元, 且调用结束后, 所占的内存单元也随之释放。 (2) 实参可以是常量、变量或表达式, 但要求它们有确定的值。 (3) 在被定义的函数中, 必须指定形参的类型。 (4) 形参与实参的类型应相同或者赋值兼容。 (5) 在 C 中, 实参和形参的数据传递时“值传递”, 单向传递。在内存中形参和实参是不同的内存单元, 在执行一个被调用函数时, 形参的值发生改变, 并不会改变主调函数实参的值。 | |
| | ◆ 返 回 值 | 1) 返回值通过 return 语句获得: return (返回值); 或 return 返回值; 。一个函数中可以有多条 return 语句, 结果以执行到的第一个为准, 且执行一个 return 语句后该函数就被调用结束; 也可以没有。不需要返回值的函数一般定义为 void 类型, 此时不得出现 return 语句。 2) 函数的类型应与返回值类型一致, 若不一致, 则以函数类型为准。 | |

| | | | | |
|--------------|---|----------------------------------|---|---|
| 四·函数的调用 | 项 目 | 内 容 | | |
| | ◆ 函数的调用 | 1) 调用形式 | 函数名（实参表列） 当然无参函数没有实参表列。 | |
| | | 2) 调用方式 | (1) 函数语句 | 函数调用作为一个语句。 |
| | | | (2) 函数表达式 | 函数作为一个表达式的一部分。 |
| | | | (3) 函数参数 | 函数调用作为另一个函数的实参。 |
| 3) 函数声明和函数原型 | ➤ 被调函数应具备的条件： (1) 被调函数必须存在（库函数或自定义函数）； (2) 若为库函数，必须用#include 命令包含所需的文件； (3) 若为自定义函数，有以下情况： i. 函数的定义位置在主调函数之前：可不必声明； ii. 函数定义位置在主调函数之后，则需要声明： a) 在主调函数中声明被调函数； b) 在文件开头声明，则从声明处到文件结尾中间所有函数可调用该函数而不用再声明； c) 若被调用函数类型为整型，C 允许调用前不用声明。 ➤ 函数原型的一般形式有两种： (1) 函数类型 函数名（参数类型 1, …, 参数类型 n）； (2) 函数类型 函数名（参数类型 1 参数名 1, …, 参数类型 n 参数名）； | | | |
| ◆ 嵌套调用 | ✧ 嵌套定义：在定义一个函数时，其函数体有包含另一个函数的完整定义。 ✧ 嵌套调用：在调用一个函数的过程中又可以调用另一个函数。 ✧ C 语言不能嵌套定义函数，但可以嵌套调用函数。 | | | |
| ◆ 递归调用 | 概念 | 在调用一函数的过程中又直接或间接地调用该函数本身，称为递归调用。 | | |
| | 直接和间接调用 | 直接 |  | 左边两种调用示意图都是无终止的调用，在程序设计是只能是有限次的、可终止的调用。 |
| 间接 |  | | | |

调用 f2 函数 调用 f2 函数

| | | |
|------------|--------------|--|
| 五·数组作函数的参数 | ◆ 数组元素作函数参数 | 作用效果与变量作实参一样，均为“值传递”方式。 |
| | ◆ 数组名作函数参数 | 1) 用数组名作函数参数，应该在主调和被调函数分别定义数组。 2) 形参数组的大小不起作用，所以可以在定义时在数组名后跟一个方括号。 3) 用数组名作函数实参时，不是把数组元素的值传递给形参，而是把实参数组的首地址传递给了形参数组，这样形参数组中元素的值发生变化时，实参的元素值也同时发生了变化。 |
| | ◆ 多维数组名作函数参数 | ✧ 可以用多维数组名作为函数的实参和形参，在被调用函数中对形参数组定义时可以指定每一维的大小，也可省略第一维的大小说明，C 编译系统不检查第一维的大小。 ✧ 但是不能把第二维以及其他高维的大小说明省略。另外指定一维而省略第二维也是不对的。不过第二位相同时，形参一维可以与实参不同。 |

二、作用域和存储类别

| 目 录 | 内 容 | | | |
|-------|-----------|---------|--|---|
| 一、作用域 | 1) 变量的作用域 | 项 目 | 概 念 | 说 明 |
| | | 局 部 变 量 | 在一个函数内定义的变量是内部变量，他只在本函数范围内有效，在此函数以外是不能使用它们的。这称为“局部变量”。 | (1) 各个函数内定义的变量只在本函数有效，主函数内的变量也不例外。形参也是局部变量。 (2) 不同函数中可以使用相同的变量名而互不干扰。 (3) 在函数内部，复合语句中定义的变量，只在本复合语句中有效。 |
| | | 全 局 变 量 | 定义在函数之外的变量成为外部变量，外部变量是全局变量。它的有效范围为从定义变量的位置开始到本源文件结束。 | (1) 为了区别全局变量和局部变量，习惯将全局变量名的第一个字母大写。 (2) 不必要时不要使用全局变量： <ul style="list-style-type: none"> i. 全局变量占内存的时间长； ii. 它使函数的通用性降低了； iii. 多使用会降低程序清晰性。 (3) 同一文件中，若外部变量与局部变量同名，则局部变量作用的范围内，外部变量被“屏蔽”。 |
| | 2) 函数的使用域 | 项 目 | 概 念 | 定义形式 |
| | | 内 部 函 数 | 若一个函数只能被本文件中其它函数调用，它称为内部函数。 | static 类型标识符 函数名（形参表）； |
| | | 外 部 函 数 | 一个函数可供其他源文件中函数调用，它称为外部函数。 | extern 类型标识符 函数名（形参表）； 注： a. 定义时 extern 表示该函数为外部函数，省略则默认为外部函数； b. 在需要用到该函数的文件中，需用 extern 对其声明，可省写 extern。 |

| | | | | |
|---------|-------------|---|------|---|
| 二. 存储类别 | 目 录 | 内 容 | | |
| | ☒ 内存中的用户区 | 内存中共用户使用的存储区分为三部分： 1) 程序区； 2) 静态存储区；全局变量存放在该单元。 3) 动态存储区；该区域主要存放函数形参、自动变量及函数调用时的现场保护和返回地址。 变量存储方式分为静态存储、动态存储这两大类，具体又分为以下 4 种。 | | |
| | ◆ auto 变量 | 自动存储类别，局部变量中未用 static 声明的变量都隐含为此类别，即 auto 可省写。声明形式： auto 类型标识符 变量名； （以下几种声明形式类似）。 | | |
| | ◆ static 变量 | ▼ 声明局部变量 | 作用 | 使局部变量的值在函数调用结束后不消失而保留原值。下次调用时变量值还在。 |
| | | | 说 明 | (1) 静态局部变量属于静态存储类别，在整个程序运行期间都不释放占用内存。 (2) 对静态局部变量时在编译时赋初值，即只赋初值一次，在程序运行时已有初值。 (3) 静态局部变量和全局变量若未赋初值，则编译时自动赋初值为 0（对数值变量）或空字符（对字符变量）；而对自动变量不赋初值其值是不确定的。 (4) 虽然静态局部变量在函数调用结束后仍存在，但其它函数还是不能够引用它。 (5) 若非必要尽量不要多用静态局部变量。 |
| | | | 适用情况 | (1) 需要保留函数上一次调用结束的值。 (2) 若初始化后，变量只被引用而不改变其值。 |
| | | ▼ 声明外部变量 | 作用 | 在程序设计中希望某些外部变量只限于本文件引用，而不能被其他文件引用，这时便可用 static 在外部变量定义时声明为静态外部变量。 这种声明利于在程序设计中由若干人完成各个模块时，各个模块中使用相同的外部变量名而不互相干扰。 |
| | | | 说明 | 外部变量本身就是静态存储变量，不要误认为只有加上 static 声明才是静态存储变量。 |

| | | | |
|--|------------------|---|--|
| | ◆ register 变量 | 作用 | 如果有一些变量使用特别频繁，则为存取变量的值要花费不少时间。为了提高效率，C 允许将局部变量的值放在 CPU 中的寄存器中，这种变量叫做寄存器变量，用 register 作声明。 |
| | | 说明 | <p>(1) 只有局部自动变量和形参作为寄存器变量。</p> <p>(2) 计算机中的寄存器数目有限，不能定义任意多个寄存器变量。不同的编译系统对 register 变量的处理方法不同（如有的作自动变量处理）。</p> <p>(3) 对一个变量只能声明一种存储类别。</p> <p>(4) 现在的优化编译系统能够识别使用频繁的变量，从而自动地将这些变量放在寄存器中，而不用程序设计者指定。</p> |
| | ◆ extern 变量 | 作用 | 外部变量的作用域是从变量的定义处到本源程序文件结束。用 extern 声明外部变量可以扩展外部变量的作用域。 |
| | | 适用情况 | <p>(1) 若外部变量的定义不是在开头进行的，要想在定以前使用该变量，则可用 extern 对其声明，使其作用域扩展为从“声明”开始。</p> <p>(2) 若要从本文件中引用另一个文件中的外部变量，那么用 extern 在本文件中对该变量进行声明。</p> <p>◆注：用 extern 声明外部变量时，类型名可以省写。例如：</p> <pre>extern int Num, Add;</pre> <p>可写为：extern Num, Add;</p> <p>但对于 auto、register、static 声明变量时，只能在定义变量的基础上加上这些关键字，而不能像 extern 这样单独使用。</p> |
| | ◇ 变量的声明和定义 | <p>广义的声明：声明包括定义，但并非所有的声明都是定义。</p> <p>狭义的声明：不需要建立存储空间的声明称为声明。</p> <p>定义：建立存储空间的声明称为定义。</p> <p>◆ 外部变量的定义只能有一次，而声明可以有多次；另外对外部变量的初始化只能在“定义”时进行，而不能在声明时进行。</p> | |

第九章 预处理命令

一、宏定义

| 目录 | 内 容 | |
|------------|--------|---|
| 一、不带参数的宏定义 | 形式 | #define 标识符 字符串 |
| | 说明 | <ol style="list-style-type: none"> 1) 宏名一般习惯用大写字母表示，以和变量名区别，但并非规定； 2) 使用宏名代替一个字符串，可以减少程序中重复书写某些字符串的工作量；而且可以使用宏定义来提高程序的通用性； 3) 宏定义是用宏名代替一个字符串，也就是简单的置换，不做正确性检查； 4) 宏定义不是 C 语句，不必在行末加分号。如果加了分号则连分号一起进行置换； 5) #define 命令出现在程序中函数的外面，宏名的有效范围为定义命令之后到本源文件结束； 6) 可以用#undef 命令终止宏定义的作用域； 7) 在进行宏定义时，可以引用已定义的宏名，可以层层置换； 8) 对程序中用双撇号括起来的字符串内的字符，即使与宏名相同也不进行置换； 9) 宏定义是专门用于预处理命令的一个专有名词，它与定义变量的含义不同，只作字符替换，不分配内存空间。 |
| 二、带参数的宏定义 | 形式 | #define 宏名（参数表） 字符串 |
| | 说明 | <ol style="list-style-type: none"> 1) 对带参数的宏的展开只是将程序语句中宏名后面括号内的实参按#define 命令行中字符串进行对应替换； 2) 在宏定义时，在宏名与参数表括号之间不应加空格；否则将空格以后的字符都作为字符串的一部分。 |
| | 与函数的区别 | <ol style="list-style-type: none"> 1) 函数调用时，先求出实参表达式的值，然后代入形参。而使用带参数的宏定义只是进行简单的字符替换； 2) 函数调用时在程序运行是处理的，为形参分配临时的内存单元。而宏展开则是在编译前进行的，在展开时并不分配内存单元，不进行值的传递处理，也没有“返回值”的概念； 3) 对函数中的实参和形参都是要定义类型的，二者的类型要求一致，如不一致需进行类型转换。而宏不存在类似的问题，宏名无类型，它的参数也无类型，只是一个符号代表，展开时带入指定的字符串即可； 4) 调用函数只可能得到一个返回值，而用宏可以设法得到几个结果； 5) 使用宏次数多时，宏展开后源程序变长，因为每展开一次都使程序增长，而函数调用不会使源程序变长； 6) 宏替换不占用运行时间，只是占编译时。而函数调用则占用运行时间。 |

二、文件包含

| 项目 | 内 容 |
|----|---|
| 形式 | <code>#include “文件名”</code> 或 <code>#include<文件名></code> |
| 说明 | <ol style="list-style-type: none"> 1) 一个#include 命令只能制定一个被包含文件，如果要包含 n 个文件，要用 n 个#include 命令； 2) 如果文件 1 包含文件 2，而在文件 2 中要用到文件 3 的内容，则可以在文件 1 中用两个 include 命令分别包含文件 2 和文件 3，而且文件 3 应出现在文件 2 之前； 3) 在一个被包含文件中有可以包含另一个被包含文件，即文件包含是可以嵌套的； 4) 在#include 命令中，文件名可以用双撇号或尖括号括起来，二者的区别： <ul style="list-style-type: none"> ➢ 用尖括号时，系统查找是到 C 库函数头文件的目录中寻找要包含的文件，这称为标准方式； ➢ 用双撇号时，系统先到用户当前目录中寻找要包含的文件，要找不到，再按标准方式查找（若用户自己编写的文件不在当前目录中，则在双撇号内应给出文件路径）； 5) 被包含文件与其所在的文件，在编译后已成为一个文件（而不是两个）。 |

三、条件编译

| 项目 | 内 容 | | |
|----|---|-------------------------------|---|
| 概念 | 一般情况下，源程序中所有行都参加编译。但是有时希望程序中一部分内容只在满足一定条件时才进行编译，也就是对这一部分内容指定编译条件，这就是条件编译。 | | |
| 形式 | 1) #ifdef 标识符 程序段 1 #else #else 可省 程序段 2 #endif | #ifdef 标识符 程序段 1 #endif | 作用：当标识符已经被#define 命令定义过，则在程序编译阶段编译程序段 1；否则编译程序段 2。（这里的程序段可以是语句组，也可以是命令行。） |
| | 2) #ifndef 标识符 程序段 1 #else 程序段 2 #endif | | 作用：若标识符未被定义过则编译程序段 1，否则编译程序段 2，作用形式刚好与第一种相反。 |
| | 3) #if 表达式 程序段 1 #else 程序段 2 #endif | | 作用：当指定的表达式值为真（非零）时就编译程序段 1，否则编译程序段 2。 |

第十章 指 针

一、 指针的基础知识

| 目 录 | 内 容 | |
|---------------|---|--|
| 1. 概念 | <ul style="list-style-type: none"> ◆ 变量的指针就是变量的地址； ◆ 指针变量就是存放变量地址的变量。 | |
| 2. 定义 | 形 式 | 基类型 *指针变量名； |
| | 说 明 | <ul style="list-style-type: none"> ➤ “*”表示该变量是指针类型，它并不是变量名的一部分； ➤ 定义指针变量时必须指定基类型，基类型决定着指针移动和运算时的移动量。 |
| 3. 指针变量的引用 | <p>a) 两个运算符：</p> <p>I) &：取地址运算符；</p> <p>II) *：指针运算符（或称“间接访问”运算符），取指针所指向的对象的内容。</p> <p>III) 以上两种运算符互为逆运算。</p> <p>b) 地址存放于指针变量：</p> <p>指针变量名=&变量名；</p> <p>（注意：“变量名”的类型应与指针的基类型一致）</p> <p>c) 指针变量的运算：</p> <p>举例说明，首先有如下语句：</p> <pre>int a,b; int *pointer; pointer=&a;</pre> <p>I) &*pointer 的含义：&和*的优先级相同，按自右至左有&(*pointer)，(*pointer)与 a 等价，那么&*pointer 与&a 等价；</p> <p>II) *&a 的含义：&a 与 pointer 等价，则*&a 与*pointer 等价，即*&a 的结果是 a；</p> <p>III) (*pointer)++ 的含义：相当于 a++。注意括号不能省。</p> | |
| 4. 指针变量作为函数参数 | <ul style="list-style-type: none"> ● 作用：将一个变量地址传送到另一个函数中。 ● 说明：C 中形参与实参之间是单向的“值传递”方式。指针变量也同样遵守这一规则。不可能通过调用函数来改变实参指针变量的值，但可以改变实参指针变量所指变量的值。 | |

二、 指针与数组

下表中使用到如下定义：

```
int a[10],b[3][4], *p;
```

| 目录 | 内 容 | | |
|--------|--------------|--|---|
| 一 维 | ◆ 数组元素的指针 | p=&a[0];与 p=a;等价 | |
| | ◆ 通过指针引用数组元素 | <div><div><div><div>p+i</div><div>a+i</div><div>&a[i]</div></div><div>}</div><div>三者等价</div><div>➡</div><div>{</div><div><div>*(p+i)</div><div>*(a+i)</div><div>a[i]</div></div><div>}</div></div><div><div>➢ 注意 p+i 不是将 p 的值简单的加 i, p+i 代表的地址是 p+i*d, d 是一个数组元素所占的字节数;</div><div>➢ 引用一个数组元素有两种方法:<div><div>1) 下标法, 如 a[i];</div><div>2) 指针法, 如*(p+i)或*(a+i);</div></div></div><div>➢ []是变址运算符。</div></div></div> | <div>使用指针变量指向数组元素时, 值得注意的几个问题:</div> <div>a) 可以改变指针变量值以指向不同的元素;</div> <div>b) 要注意指针变量的当前值;</div> <div>c) 注意指针变量的运算。</div> <div>☒ 注意: 可以有 p++但不能有 a++, ++只能用于变量, a 为数组名, 其值是地址, 是常量!</div> |
| | ◆ 用数组名作函数参数 | 当用数组名作函数参数时, 如果形参数组中个元素的值发生变化, 实参数组的值随之变化。这是由于 C 编译都是将形参数组名作为指针来处理。 | |
| 多 维 | 以二维数组为例 | <div><div><div>列地址</div><div>b[0]</div><div>b+1</div><div>b+2</div><div>行地址</div></div><div><div><div>b[0]+0</div><div>b[0]+1</div><div>b[0]+2</div><div>b[0]+3</div></div><div><div><div>2000 1 2000 9 2000 17</div><div>2000 3 2000 11 2000 19</div><div>2000 5 2000 13 2000 21</div><div>2000 7 2000 15 2000 23</div></div></div></div><div><div>◆ 行指针与列指针:<div><div>1. 行指针前加*转换为列指针: b→*b, b+1→*(b+1)</div><div>2. 列指针前加&转换为行指针: b[0]→&b[0]</div></div></div><div>◆ b+i=&b[i]=b[i]=*(b+i)=&b[i][0], 值相等, 含义不同:<div><div>1) b+i, &b[i]表示第 i 行首地址, 指向行</div><div>2) b[i], *(b+i), &b[i][0]表示第 i 行第 0 列元素地址, 指向列</div></div></div><div>◆ 二个等价: b[i]+j=*(b+i)+j 等价, 表示第 i 行第 j 列的元素地址; *(b[i]+j)=*(*(b+i)+j)=b[i][j] 等价, 表示第 i 行第 j 列的元素值。</div></div></div> | |

三、 指针与字符串

| 目 录 | 内 容 | | | | | | | | | | | | | | |
|----------------|---|---|--|----|----|----|----|-----|-----|--------|--------|-----|--------|--------|-----|
| 一、字符串的表示形式 | 项目 | 形式举例 | 说 明 | | | | | | | | | | | | |
| | 1. 字符数组 | <pre>char string[]=" I love XYQ" ; printf("%s\n,string);</pre> | 第二句是通过字符数组名对整个字符串的输出，也可以用字符指针变量输出一个字符串。但是对于一个整型数组是不能企图用数组名输出它的全部元素！ | | | | | | | | | | | | |
| | 2. 字符指针 | <pre>char *string=" I love XYQ" ; printf("%s\n,string);</pre> | 第一句等价于： <pre>char *string; string=" I love XYQ" ;</pre> 注意 只是把" I love XYQ" 的第一个字符的地址赋给了指针变量 string。 | | | | | | | | | | | | |
| 二、字符指针作函数参数 | <p>将一个字符串从一个函数传递到另一个函数，可以用地址传递的办法，即用字符数组名作参数，也可以用指向字符的指针变量作参数。</p> <p>归纳起来，作为函数参数，有以下几种情况：</p> <table border="1"> <tr> <td>实参</td><td>形参</td><td>实参</td><td>形参</td></tr> <tr> <td>数组名</td><td>数组名</td><td>字符指针变量</td><td>字符指针变量</td></tr> <tr> <td>数组名</td><td>字符指针变量</td><td>字符指针变量</td><td>数组名</td></tr> </table> | | | 实参 | 形参 | 实参 | 形参 | 数组名 | 数组名 | 字符指针变量 | 字符指针变量 | 数组名 | 字符指针变量 | 字符指针变量 | 数组名 |
| 实参 | 形参 | 实参 | 形参 | | | | | | | | | | | | |
| 数组名 | 数组名 | 字符指针变量 | 字符指针变量 | | | | | | | | | | | | |
| 数组名 | 字符指针变量 | 字符指针变量 | 数组名 | | | | | | | | | | | | |
| 三、字符指针与字符数组的区别 | <p>用字符数组和字符指针变量都能对字符串的存储和运算，但它们二者之间是有区别的，不应混为一谈，主要由以下几点：</p> <p>(1) 字符数组由若干个元素组成，每个元素中放一个字符，而字符指针变量中存放的是地址（字符串第一个字符的地址），决不是将字符串放到字符指针变量中。</p> <p>(2) 赋值方式。对字符数组只能对各个元素赋值，不能用以下办法对字符数组赋值：</p> <pre>char str[11]; str=" I love XYQ" ;</pre> <p>但是对字符指针变量可以。</p> <p>(3) 字符数组在编译时为它分配内存单元，它有确定的地址。而字符指针变量分配的内存单元只能放一个字符变量的地址，但若为对它赋值（地址），则它并不指向一个确定的字符数据。</p> <p>(4) 指针变量的值是可以改变的。</p> <p>(5) 用指针变量指向一个格式字符串，可以用它代替 printf 函数中的格式字符串。</p> | | | | | | | | | | | | | | |

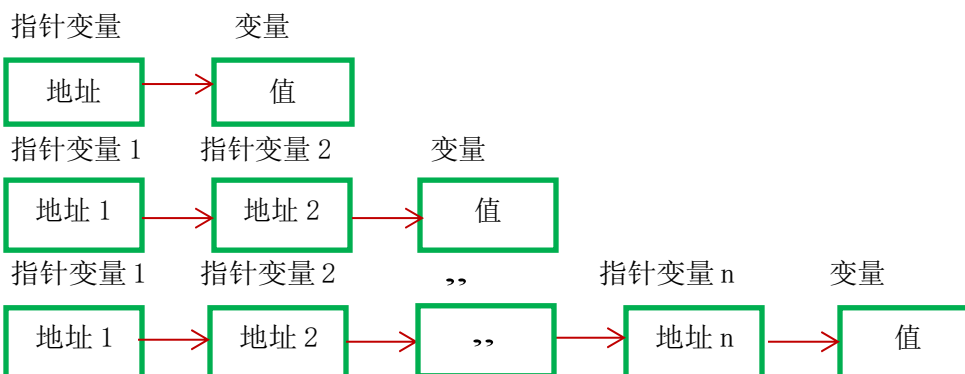
四、指针与函数

| 目录 | 内 容 | |
|------------------|---|--|
| 一. 用函数指针变量调用函数 | 项目 | 说 明 |
| | ◆ 指向函数的指针变量的定义形式： 数据类型 (*指针变量名) (函数参数列表)； | 第一个括号不能省，否则由于括号的优先级高于“*”使其变成返回指针类型值的函数的定义了。 |
| | ◆ 用指向函数的指针变量调用函数的形式： (*指针变量名) (函数实参列表)； | <p>➤ 在调用前必须将函数入口地址赋给指针变量，赋值形式： 指针变量名=函数名； (注：函数名代表该函数的入口地址)</p> <p>➤ 对指向函数的指针变量，像 p++、p--、p+n 等运算是无意义的。</p> |
| 二. 用指向函数的指针作函数参数 | <p>函数指针变量通常的用途之一是把指针作为参数传递到其他函数。以实现函数地址的传递，这样就能在被调用的函数中使用实参函数。</p> <p>这里主要用于每次调用的函数不是固定的。</p> | |
| 三. 返回指针值的函数 | <p>定义形式： 类型名 *函数名 (参数表列)； 注意与指向函数的指针变量的定义形式区别。</p> | |

五、 指针数组和指向指针的指针

| 目录 | 内 容 | |
|---------|--|---|
| 一. 指针数组 | 一维指针数组的定义形式： 类型名 *数组名[数组长度]； | 注意不要写成： 类型名 (*数组名)[数组长度]； 这是指向一维数组的指针变量定义。 |
| | <p>指针数组的用处主要体现在它比较适合于用来指向若干个字符串，是字符串的处理更加方便灵活。</p> <p>其中一个常用的地方是对多个字符串排序，只需改变指针数组中个元素的指向便可以对多个字符串排序。</p> | |

- ◆ “间接访问”变量的方式理论上可以有多级，但实际用途中很少有超过二级的。指向指针的指针用的就是“二级间址”的方法。



- ◆ 指向指针的变量定义形式：
类型名 **指针变量名；

✎ 使用时有个值得注意的地方，举例说明：

```
#include<stdio.h>
void main()
{
    char *name[]={"i love you ", "follow me", "goods", "great wall"};
    int a[4]={1, 2, 3, 4};
    char **p, **p1, **p2;
    p=name+3;
    p1=a+1;
    p2=&p1;
    printf("%o\n", *p); //①
    printf("%s\n", *p); //②
    //printf("%s\n", **p); //③
    printf("%s\n", name[3]); //④
    printf("%o\n", *p2); //⑤
    printf("%d\n", **p2); //⑥
}
```

程序运行结果：

```
20420050
great wall
great wall
6177444
2
```

- ✧ 其中采用“%o”是为了输出八进制地址。
- ✧ 程序中，标号①②中*p 都是指向 name[3]的值，在①中输出地址，在②中输出该地址指向的字符串（注意字符串的输入输出，在参数列表是代表字符数组首地址的指针变量或字符数组名）。
- ✧ 标号③是错误的语句，原因可见上一条。②④语句等效。
- ✧ ⑤语句*p2 指向 p1 的值；⑥语句**p2 是指向 p1 指向的变量值。注意①②句与⑤⑥句的异同。

指针数组的一个重要应用是作为 main 函数的形参。main 可以有以下形式的形参：

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

argc 和 **argv** 就是 main 函数的形参。main 函数是由操作系统调用的。当处于操作命令状态下，输入 main 所在的文件名（经过编译、连接后得到的可执行文件名，后缀为 .exe），操作系统就调用 main 函数。那么实参是不可能由程序给出的。实际上实参是和命令行一起给出的，也就是在一个命令行中包括命令和需要传给 main 函数的参数。命令行的一般形式：

命令名 参数 1 参数 2,,,参数 n

其中命令名是 main 所在的执行文件名（包括盘符、路径以及文件的扩展名）。注意，文件名也作为一个参数。输入命令行后，**argc** 的值为 n+1，指针数组 **argv** 的元素按顺序依次指向“**命令名 参数 1 参数 2,,,参数 n**”这 n+1 个字符串。

若 main 函数中包含以下语句：

```
void main(int argc, char *argv[ ])  
{  
while(argc>1)  
    {++argv;  
    printf( "%s\n" , *argv);  
    --argc;  
    }  
}
```

则以上命令行执行后输出一下信息：

参数 1

参数 2

,,,

参数 n

当然 main 函数中的形参不一定命名为 **argc** 和 **argv**，可以使任意的名字，只是人们习惯用 **argc** 和 **argv** 而已。

六、 小结

| 目录 | 内 容 | |
|---------------|--|-----------------------------|
| 一、有关指针的数据类型小结 | 定 义 | 含 义 |
| | int i; | 定义整型变量 i |
| | int *p; | p 为指向整型数据的指针变量 |
| | int a[n]; | 定义有 n 个元素的整型数组 a |
| | int *p[n]; | 定义由 n 个指向整型数据的指针元素组成的指针数组 p |
| | int (*p)[n]; | p 为指向含 n 个元素的一维数组的指针变量 |
| | int f(); | f 为可返回整型函数值的函数 |
| | int *p(); | p 为返回一个指针的函数，该指针指向整型数据 |
| | int (*p)(); | p 为指向函数的指针，该函数返回一个整型值 |
| | int **p; | p 为一个指针变量，它指向一个指向整型数据的指针变量 |
| 二、指针运算小结 | <p>(1) 指针变量加（减）一个整数： 如：p++；p--；p+i；p-i；p+=i；p-=i 等，加减的值与类型有关。</p> <p>(2) 指针变量赋值： p=&a （将变量 a 的地址赋给 p） p=array；（将数组 array 首地址赋给 p） p=&array[i]；（将数组 array 第 i 个元素的地址赋给 p） p=max；（max 为已定义的函数，将 max 函数的入口地址赋给 p） p1=p2；（p1 和 p2 都是指针变量，将 p2 的值赋给 p1）</p> <p>(3) 指针变量不指向任何变量，即取空值。表示为：p=NULL；</p> <p>(4) 两个指针变量可以相减： 如果两个指针变量指向同一个数组为元素，则两个指针变量值之差是两个指针之间的元素个数。但 p1+p2 并无实际意义。</p> <p>(5) 两个指针变量比较： 如果两个指针变量指向同一个数组为元素，则可以进行地址比较。</p> | |
| 三、void 指针类型 | <ul style="list-style-type: none"> ❖ 定义时不指定指向哪一类数据； ❖ 用动态存储分配函数时返回 void 指针； ❖ 它的值赋给另一指针变量时，要强制类型转换。 | |

第十一章 结构体与共用体

一、结构体

| 目 录 | 内 容 | | 补充说明 |
|------------|------|--|---|
| 1. 结构体类型声明 | 声明形式 | <pre>struct 结构体名 {成员表列}; ←（分号不能掉）</pre> | 成员表列构成： 类型名 成员名； |
| 2. 结构体变量定义 | 定义形式 | <ul style="list-style-type: none"> ● 先声明再定义： struct 结构体名 变量名； | 1) 类型与变量是不同的概念！ 只能对变量赋值、存取或运算，而不能对一个类型赋值、存取或运算。在编译时，对类型是不分配空间的，只对变量分配空间。 2) 成员也可以是一个结构体变量。 3) 成员名可以与程序中别的变量名相同，而这不代表同一对象。 |
| | | <ul style="list-style-type: none"> ● 声明的同时定义： struct 结构体名 { 成员表列 }变量名表列； | |
| | | <ul style="list-style-type: none"> ● 直接定义： struct { 成员表列 }变量名表列； | |
| 3. 结构体变量引用 | 引用形式 | <pre>结构体变量名. 成员名</pre> <p>注：“.”是成员（分量）运算符。</p> | 1) 不能将一个结构体变量作为一个整体输入输出。必须具体到成员。 2) 若成员本身是结构体类型，则要用若干个成员运算符，一级一级的找到最低一级成员。只能对最低级的成员进行赋值或存取以及运算。 3) 结构体变量成员可以像普通变量一样按自身的类型进行相应的运算。 |

| | | |
|--------------|--|--|
| 4. 结构体变量初始化 | 和其他类型变量一样，对结构体变量可以在定义时指定初始值。 | |
| 5. 指向结构体类型指针 | <p>◆ 指向结构体变量的指针： 指向结构体变量的指针，要利用指针访问其成员需要采用如下格式： (*p).成员名 注：p 为指向结构体变量的指针。</p> | <p>其实以下 3 种形式等价： I) 结构体变量名.成员名 II) (*p).成员名 III) p->成员名 其中->称为指向运算符。</p> |
| | <p>◆ 指向结构体数组元素的指针： 与指向数组元素的指针的用法类似（可参见指向数组元素指针的使用方法）。</p> | <p>程序中指定 p 为一个指向结构体类型变量的指针，则它只能指向变量，而不能指向变量的成员。若要指向成员，可以进行强制转换： p=(struct student *)stu[0].name;</p> |
| | <p>◆ 用结构体变量和指向结构体的指针作函数参数：</p> <ul style="list-style-type: none"> i. 用结构体变量的成员作参数（值传递）。 ii. 用结构体变量作实参。这时形参也必须是同类型的结构体变量（该类型也是值传递）。 iii. 用指向结构体（或数组）的指针作实参，将结构体变量（或数组）的地址传给形参。 | |

二、链表

| 目 录 | 内 容 | 补充说明 |
|--------|---|--|
| 一、链表简介 | <ul style="list-style-type: none"> ◇ 链表是一种重要的数据结构。 ◇ 链表有一个“头指针”变量（head），head 指向第一个元素；第一个元素又指向第二个元素，...直到最后一个元素，该元素不再指向其他元素，它称为“表尾”，它的地址部分放一个空指针“NULL”，链表到此结束。 ◇ 链表每个节点包括两个部分：用户要用的实际数据和下一个节点的地址。 ◇ 据链表的特点可知结构体变量作链表中的节点最合适。 | <p>链表的特点：</p> <ul style="list-style-type: none"> ➤ 可以根据需要临时开辟内存空间，避免内存浪费。 ➤ 链表中的元素不是连续存放的，这也在一定程度上可以节省空间。但是正因为如此，链表没有头指针就无法访问，在访问其中的元素时必须需要上一个元素提供地址。这样对其中一个节点进行操作时可能比其他数据类型耗时长。 |
| 二、静态链表 | <p>节点均为开始预设的，不是临时开辟的，用完后也不能立即释放所占的空间。</p> | <p>该种链表没有发挥链表自身的优势。</p> |
| 三、动态链表 | <p>◆ 动态链表的操作包括：建立，输出，删除，插入。</p> <ul style="list-style-type: none"> ◇ 对于建立简单动态链表，一般需要三个指针。 ◇ 对于输出链表需知道 head 指针外还要借助一个指针。 ◇ 对于删除要借助另外两个指针，还要分删除的节点在头位置还是中间，还要考虑链表中是否包含寻找项及链表是否为空。 ◇ 对于插入是删除的逆过程，可参照删除进行。 | <p>在用 malloc 等开辟空间后将其返回值（地址）赋给结构体指针变量时要注意，malloc 等返回值是无类型（void *类型）数据指针，要加上(struct 结构体名 *)进行数据类型强制转换。</p> |
| | <p>结构体和指针的应用领域很广，这里介绍的链表都是单向链表，还有环形链表和双向链表。另外还有队列、树、栈、图等数据结构。关于这些内容可学习“数据结构”课程。</p> | |

| ◆ 处理动态链表所需要的函数 | | | | |
|----------------|--|---|--|--|
| 函数 | ✧ malloc | ✧ calloc | ✧ free | |
| 原型 | void *malloc(unsigned int size); | void *calloc(unsigned n, unsigned size); | void *free(void *p); | |
| 功能 | 在内存的动态存储区分配一个长度为size的连续空间。 | 在内存的动态存储区分配n个长度为size的连续空间。 | 释放由p指向的动态存储区。p是最近一次调用calloc或malloc函数时返回的值。 | |
| 返回值 | 开辟的一个分配域的起始地址；开辟失败则返回空指针。 | 分配域起始地址。失败则返回空指针“NULL”。 | 无返回值。 | |

三、共用体

| 目 录 | 内 容 | | 补充说明 |
|----------------|--------|---|---|
| └ 共用体类型声明及变量定义 | 声明形式 | union 共用体名 {成员表列 }; | “共用体”和“结构体”的定义形式相似。但它们的含义是不同的。结构体变量所占的长度是个成员占的内存长度之和；而共用体变量所占内存长度等于最长的成员长度。 |
| | 变量定义形式 | ✧ 声明的同时定义： union 共用体名 {成员表列 }变量表列； | |
| | | ✧ 先声明后定义： union 共用体名 变量名； | |
| | | ✧ 直接定义： union {成员表列 }变量表列； | |

| | | |
|---|---|-------------------------------|
| <ul style="list-style-type: none"> • ∞, 共用体引用 | 引用形式: 变量名. 成员名 | 注意: 只能引用共用体变量中的成员, 不能引用共用体变量。 |
| <ul style="list-style-type: none"> • ∞, 共用体类型数据的特点 | <ul style="list-style-type: none"> ✓ 同一内存段可以用来存放几种不同类型的成员, 但每一瞬间只能存放其中一种, 而不是同时存放几种。 ✓ 共用体变量中起作用的成员是最后一次存放的成员。 ✓ 共用体变量的地址和它的成员的地址都是同一地址。 ✓ 不能对共用体变量名赋值, 也不能企图引用变量名来得到一个值, 又不能在定义共用体变量时对变量初始化。 ✓ 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, 但可以用指向共用体变量的指针 (与结构体变量这种用法相仿)。 ✓ 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的定义中, 数组也可以作为共用体的成员。 | |

四、 枚举类型

| 目 录 | | 内 容 | | 补充说明 |
|------|------|---|--|---|
| 枚举类型 | 声明定义 | 类型声明 | <pre>enum 枚举名 { 枚举元素(或称枚举常量) };</pre> | 枚举元素：是用户定义的标示符，它们所代表的含义完全有程序员决定。枚举元素相互之间用“，”隔开。 |
| | | 变量定义 | <p>✧ 先声明后定义： <pre>enum 枚举名 变量名</pre></p> <p>✧ 直接定义： <pre>enum { 枚举元素(或称枚举常量) }变量表列;</pre></p> <p>✧ 声明的同时定义： <pre>enum 枚举名 { 枚举元素(或称枚举常量) }变量表列;</pre></p> | 定义的枚举类型变量，它们的值只能是枚举元素或枚举常量之一。 |
| | 使用说明 | <p>1) 在 C 编译中，对枚举元素按常量处理。它们不是变量，不能对它们赋值。</p> <p>2) 枚举元素作为常量，它们是有值的。C 编译按定义时先后顺序使它们的值为 0, 1, 2,...。也可以改变枚举元素的值，在定义时由程序员指定。例如： <pre>enum weekday {sun=7, mon=1, tue, wed, thu, fri, sat} weekday, week_end;</pre> 定义 sun 为 7，mon 为 1，以后顺序加 1，sat 为 6。</p> <p>3) 枚举值可以用来作判断比较。</p> <p>4) 一个整数不能直接赋给一个枚举变量。按 2) 中例子： <pre>weekday=2; （不能直接赋值）</pre> 应先进行强制类型转换才能赋值： <pre>weekday= (enum weekday) 2;</pre> 它相当于 <code>weekday=tue</code>；当然所赋的整数值必须在枚举元素的范围内。</p> | | |


五、用 typedef 命名已有类型

| 目 录 | 内 容 | | |
|------|--|--------------------|-----------------------|
| 功能描述 | 用 typedef 可以声明新的类型名来代替已有的类型名。 | | |
| 使用方法 | 声明一个新的类型名步骤 | 举例 1 | 举例 2 |
| | 1) 先按定义变量的方法写出定义体; | int i; | int n[100]; |
| | 2) 将变量名换成新变量名; | int COUNT; | int NUM[100]; |
| | 3) 在前面加 typedef; | typedef int COUNT; | typedef int NUM[100]; |
| | 4) 然后可以用新的类型去定义变量。 | COUNT a, b, c; | NUM m, n; |
| 说 明 | <p>✧ 习惯上常把用 typedef 声明的类型名用大写字母表示，以便与系统提供的标准类型标示符相区别。</p> <p>✧ 用 typedef 可以声明各种类型名，但不能用来定义变量。（用 typedef 声明数组类型、字符串类型使用比较方便。举例 2 中用 NUM 可以定义含 100 个元素的一维数组。这样可以将数组类型和数组变量分离开来，使用方便。同样可以定义字符串类型、指针类型等。）</p> <p>✧ 用 typedef 只是对已经存在的类型指定一个新的类型名，而没有创造新的类型。</p> <p>✧ typedef 与#define 有相似之处，例如：</p> <pre>typedef int count;</pre> <p>和</p> <pre>#define COUNT int</pre> <p>作用都是用 COUNT 代表 int。但事实二者又有区别。# define 是在预编译时处理的，它只做简单的字符串替换，而 typedef 是在编译时处理的。且它并不是做简单的字符串替换。</p> <p>✧ 当不同源文件中用到同一类型数据（尤其是像数组、指针、结构体、共用体、等类型数据）时，常用 typedef 声明一些数据类型，把它们单独放在一个文件中，然后在需要用到文件中用#include 命令包含进来。</p> <p>✧ 用 typedef 有利于程序的通用与移植。（类似#define 在这方面的使用）</p> | | |

第十二章 位运算

一、位运算符及使用

| 运算符 | 含 义 | 运算规律 | 运算对象 个数及结 合性 | 用 途 |
|-----|------|-----------------------------|--------------------|---|
| &① | 按位与 | 两个相应二进制位都为 1 时该位结果为 1，否则为 0 | <u>双目，自左至右</u> | <ul style="list-style-type: none"> ● 清零； ● 保留指定位（要保留的位取 1，其他取 0 即可）。 |
| | 按位或 | 两个相应二进制位中只要有一个为 1，该位结果为 1 | <u>双目，自左至右</u> | <ul style="list-style-type: none"> ● 对指定位置 1（置 1 位取 1，其他位取 0 即可）。 |
| ^ | 按位异或 | 两个二进制位相同则结果为 0，相异结果为 1 | <u>双目，自左至右</u> | <ul style="list-style-type: none"> ● 使特定位翻转（翻转位取 1，其他位取 0 即可）； ● 交换两个变量的值，不用中间变量②。 |
| ~ | 取反 | 0 变 1, 1 变 0 | <u>单目，自右至左</u> | • |
| << | 左移 | 将一个数全部左移若干位，高位溢出舍弃，右补 0 | <u>双目，自左至右</u> | <ul style="list-style-type: none"> ● 左移一位相当于乘 2，将乘 2^n 的幂运算处理为左移 n 位，要比乘法运算快得多。 |
| >> | 右移 | 将一个数全部右移若干位，低位舍弃，高位补 0 ③ | <u>双目，自左至右</u> | <ul style="list-style-type: none"> ● 右移一位相当于除 2，将除 2^n 的幂运算处理为右移 n 位，要比除运算快得多。 |

 **注**：① 当有负数参与与运算时，以补码形式进行运算。

② 举例：a=3, b=4, 让 a 和 b 的值互换，可用以下语句实现：

$a=a^b; \quad b=b^a; \quad a=a^b;$

其等效于 $b=b^a=a^b; a=a^b=(a^b)^{(b^b^a)}=b^b^b^a^a=b$ 。

③ 在 vc6.0 中，对于有符号的负值编译系统会保留符号位不动。左移右移就是乘除 2^n ，运算规律如整数乘除，不过负数时进位方向向小的方向（如 $-9 \gg 2 = -3$, $9 \gg 2 = 2$ 而 $-9/4 = -2$, $9/4 = 2$ ）。

二、 位段

1. 向一个字节中的一个或几个二进制位赋值和改变其值的方法有以下两个：

- 1) 可以使用位运算进行赋值和改值；
- 2) 使用位段。

2. 位段

- 定义：在一个结构体中以位为单位来指定其成员所占内存长度，这种以位为单位的成员称为“位段”或称“位域”。

举例：

```
struct packed_data
{
    unsigned a:2;

    unsigned b:6;

    unsigned c:4;

    unsigned d:4;

    int i;

}data;
```

例子中，2、6、4、4 是指定所占的位数，在使用时要注意位段允许的最大值。

- 说明：

- 1) 位段的成员必须指定位 unsigned 或 int 类型；
- 2) 若某一位段要从另一字节开始存放，可以用以下形式定义：

```
unsigned a:2;
unsigned b:1; } (一个存储单元)

unsigned :0;

unsigned a:3; (另一个存储单元)
```

- 3) 一个位段必须存储在同一个存储单元中，不能跨两个单元。
- 4) 可以定义无名位段。

```
unsigned a:1;
```

```
unsigned :2; (这两位空间不用)
```

```
unsigned b:3;
```

```
unsigned a:4;
```

- 5) 位段长度不能大于存储单元长度，也不能定义位段数组。
- 6) 位段可以用整型格式符输出。
- 7) 位段可以在数值表达式中引用，它会被系统自动地转换成整型数。

第十三章 文 件

一、对处理文件的常用函数整理如下表格：

| 分 类 | 函数名 | 调用形式 | 功 能 | 返回值 |
|------|--------|--------------------------|------------------------------------|------------------------|
| 打开方式 | fopen | fopen(文件名, 打开方式①); | 打开文件 | 成功: 文件地址 失败: NULL |
| 关闭文件 | fclose | fclose(fp②); | 关闭文件 | 顺利关闭: 0 否则: EOF(-1) |
| 文件定位 | fseek | fseek(文件指针, 位移量③, 起始量④); | 改变文件位置指针的位置 | |
| | rewind | rewind(fp); | 使文件位置指针重新回到文件开头 | |
| | ftell | ftell(fp); | 获取当前文件位置指针的位置 | 成功: 当前位置 失败: -1L |
| 文件状态 | feof | feof(fp); | 检测文件位置指针是否处于文件末尾 | 真: 1 假: 0 |
| | ferror | ferror(fp); | 检测文件操作是否出错 | 真: 非零 假: 0 |
| | clearr | clearr(fp); | 使 ferror 和 feof 函数置零 | |
| 文件读写 | A | fgetc, getc | ch=getc(fp); | 从文件中读出一个字符 |
| | | fputc, putc | fputc(ch, fp); | 写入一个字符到指定文件 |
| | B | fgets | fgets(str, n, fp); | 读取含 (n-1) 个字符的字符串 |
| | | fputs | fputs(str, fp); | 写入字符串到指定文件 |
| | C | getw | i=getw(fp); | 读出一个字 (int 型) |
| | | putw | putw(i, fp); | 写入一个字 (int 型) |
| | D | fread | fread(buffer⑤, size⑥, count⑦, fp); | 读取指定个数的数据项 |
| | | fwrite | fwrite(buffer, size, count, fp); | 写入指定个数的数据项 |
| | E | fscanf | fscanf(文件指针, 格式符号, 输出列表); | 按格式读取数据 |
| | | fprintf | fprintf(文件指针, 格式符号, 输出列表); | 按格式写入数据 |

注：○ 文件名和打开方式均为字符串格式，且打开方式如下表：

| 文件使用方式 | 含 义 | 说 明 |
|---------|-----------|----------|
| 文本文件使用 | “r”（只读） | 为输入打开文件 |
| | “w”（只写） | 为输出打开文件 |
| | “a”（追加） | 向文件尾添加数据 |
| | “r+”（读写） | 为读写打开文件 |
| | “w+”（读写） | 为读写新建文件 |
| | “a+”（读写） | 为读写打开文件 |
| 二进制文件使用 | “rb”（只读） | 为输入打开文件 |
| | “wb”（只写） | 为输出打开文件 |
| | “ab”（追加） | 向文件尾添加数据 |
| | “rb+”（读写） | 为读写打开文件 |
| | “wb+”（读写） | 为读写新建文件 |
| | “ab+”（读写） | 为读写打开文件 |

②表中使用的变量及其定义如下：

```
FILE *fp;                int i, n;                char ch, str[ ];
```

③位移量可正可负，正表前移，负表后移，单位为：L（表示 long 型）。

④起始点规定如下：

| 起始点 | 名 字 | 用数字代表为 |
|--------|----------|--------|
| 文件开始 | SEEK_SET | 0 |
| 文件当前位置 | SEEK_CUR | 1 |
| 文件末尾 | SEEK_END | 2 |

⑤buffer:一个地址指针，对 fread 来说为读出数据存放的地址；对后面的 fwrite 来说是输出数据到文件的地址。

⑥size: 要读写的字节数。

⑦count: 要读写多少个 size 字节的数据项。

二、文件基本知识

1. C 语言的文件是一种字符序列，这种文件常被称为流式文件；且对文件的存取是以字节为单位的，另外 ANSI C 标准采用缓冲文件系统。

2. 其中按数据的组织形式可分为 ASCII 文件（text 文件）和二进制文件，其特点如下：

| 文件类型 | 优 点 | 缺 点 |
|----------|-------------|------------------|
| ASCII 文件 | 便于字符的输入输出 | 所占储存空间大，读写时有转换时间 |
| 二进制文件 | 节省空间，没有转换时间 | 但不能直接输出字符 |

3. 文件类型：每个被使用的文件在内存中都有一个结构体变量存储该文件的信息，该结构体变量有系统定义，取为 FILE。用 FILE 可定义文件类型变量，其常用于定义文件类型指针变量，如 FILE *fp; 。

4. 文件使用前需打开，在使用结束后要关闭，以节省内存和防止信息丢失。

附录

C 语言的运算符和结合性

| 优先级 | 运算符 | 含义 | 运算对象个数 | 结合方向 |
|-----|---|------------|--------|------|
| 1 | () | 圆括号 | | 自左至右 |
| | [] | 下标运算符 | | |
| | -> | 指向结构体成员运算符 | | |
| | . | 结构体成员运算符 | | |
| 2 | ! | 逻辑非运算符 | 1 (单目) | 自右至左 |
| | ~ | 按位取反运算符 | | |
| | ++ | 自增运算符 | | |
| | -- | 自减运算符 | | |
| | - | 负号运算符 | | |
| | (类型) | 类型转化运算符 | | |
| | * | 指针运算符 | | |
| | & | 取地址运算符 | | |
| | sizeof | 长度运算符 | | |
| 3 | * | 乘法运算符 | 2 (双目) | 自左至右 |
| | / | 除法运算符 | | |
| | % | 求余运算符 | | |
| 4 | + | 加法运算符 | 2 (双目) | 自左至右 |
| | - | 减法运算符 | | |
| 5 | << | 左移运算符 | 2 (双目) | 自左至右 |
| | >> | 右移运算符 | | |
| 6 | < <= > >= | 关系运算符 | 2 (双目) | 自左至右 |
| 7 | == | 等于运算符 | 2 (双目) | 自左至右 |
| | != | 不等于运算符 | | |
| 8 | & | 按位与运算符 | 2 (双目) | 自左至右 |
| 9 | ^ | 按位异或运算符 | 2 (双目) | 自左至右 |
| 10 | | 按位或运算符 | 2 (双目) | 自左至右 |
| 11 | && | 逻辑与运算符 | 2 (双目) | 自左至右 |
| 12 | | 逻辑或运算符 | 2 (双目) | 自左至右 |
| 13 | ? : | 条件运算符 | 3 (三目) | 自右至左 |
| 14 | = += -= *= /= %= >>= <<= &= ^= != | 赋值运算符 | 2 (双目) | 自右至左 |
| 15 | , | 逗号运算符 | | 自左至右 |