第一章 绪论

**①程序设计语言**

程序是指以某种程序设计语言为工具对解决问题的操作序列的描述。

程序设计语言是用户用于编写程序的语言，属于人造语言，是计算机软件系统的重要组成部分

1. **机器语言**

由一系列基本指令组成，可以由机器直接执行

机器语言编写的程序是由二进制代码组成的代码序列，但具有可读性差，可移植性差，描述处理方式与通常思维方式有较大差距的缺点。

1. **汇编语言**

用某种代表机器语言中的01代码的助记符号描述的指令系统。又称为符号语言。将汇编语言转换成机器语言程序的程序叫做汇编程序。

汇编语言指令与机器语言指令是一一对应的。与计算机硬件有较大联系，依赖具体机器特性

有可移植性差，描述处理方式与通常思维方式有较大差距的缺点。也称为计算机低级语言。

1. **高级语言**

克服了机器语言和汇编语言的所有缺点，最成功的属于C语言。

1. 一开始，高级语言为**非结构化的语言（BS算法）**。没有严格规范要求，程序中的流程可以随意跳转。使程序变得难以阅读
2. 后来，高级语言为**结构化的语言**。规定程序必须由具有良好特性的基本结构构成。这种程序结构清晰，易于编写、阅读、维护。

* **三种基本结构**

1. **顺序结构**

执行a之后执行b

1. **选择结构**

若条件p成立，执行a；否则执行b

1. **循环结构**
2. **while型。**若条件p1成立，执行a；若条件p1仍成立，执行a；若条件p1不成立，循环结束。
3. **until型。**若条件p2不成立，执行a；若条件p2仍不成立，执行a；若条件p2成立，循环结束。

以上两种语言都是**面向过程的语言**，编写程序时需要具体指定每一个过程的细节。

1. 现在，有了**面向对象的语言**。更易于处理规模较大的问题。

**②程序设计**

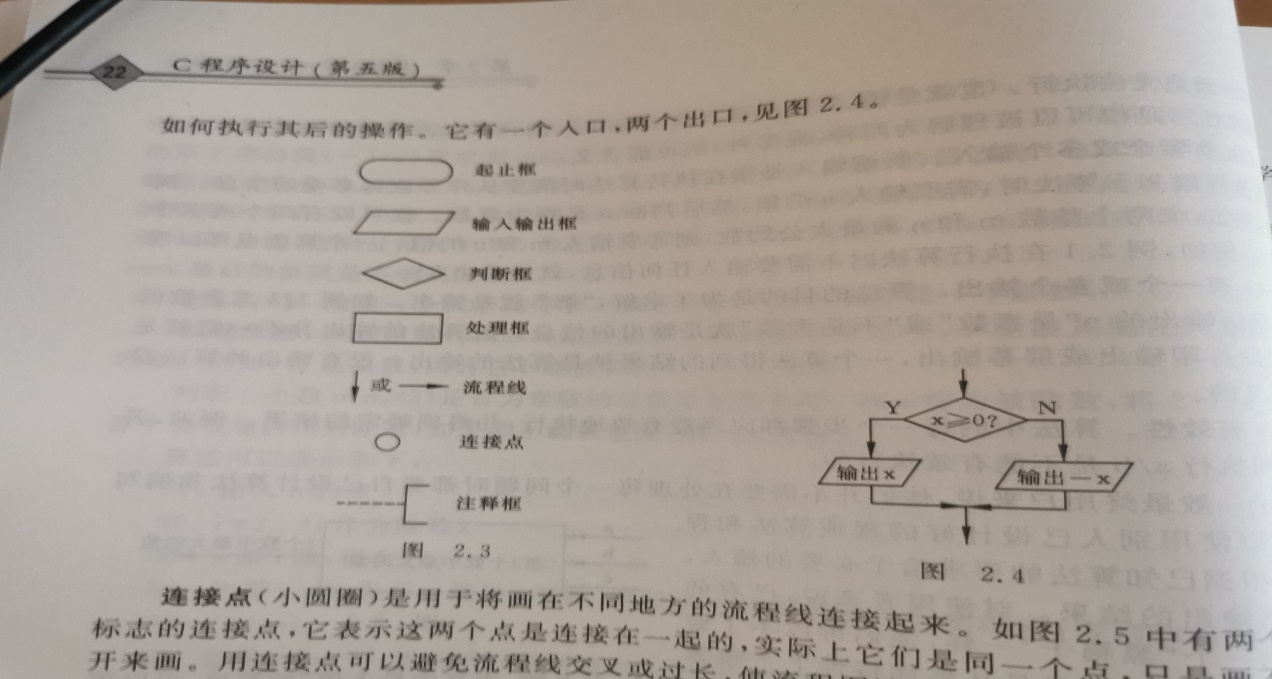
**a. 概念**

程序设计的目的就是用计算机解决问题。

**b.算法**

算法泛指解决某一个问题的方法和步骤。

1. **特性**

* **有穷性**。算法应该包含有限的操作步骤，不能是无限的。
* **确定性**。算法中的任何一个步骤不能是有歧义，概念模糊的。
* **有零个或多个输入**。输入是指在执行算法时需要从外界取得必要的信息。
* **有一个或多个输出**。没有输出的算法是没有意义的，算法的目的是为了求解，必须有输出。
* **有效性**。算法中的每一个步骤都应当能有效地执行并得到确定的结果。

1. **表示算法的方法**

* **用自然语言表示算法**

用人们日常使用的语言来表达，虽然通俗易懂但是文字冗长且易于产生歧义。而且自然语言描述**包含分支和循环的算法不太方便**。除了简单问题，很少用自然语言表示算法。

* **用流程图表示算法**

用一些图框来表示操作。直观形象且易于理解。

* **用N-S流程图（盒图）表示算法**

电脑屏幕的照片上有文字

描述已自动生成

* **用伪代码表示算法**

**伪代码**是用介于自然语言与计算机语言之间的文字和符号来描述算法。自上而下地写下基本操作。可以用中文，只要能够便于书写和阅读即可。

* **用计算机语言表示算法**

**c.程序设计的步骤**

粗略步骤为：

1. **算法设计**：分析问题构造一个解决问题的算法
2. **编码**：用一种程序设计语言将该算法表达为程序

还可以细分为：

1. **分析问题**

确定需要计算机做什么来解决这个问题

1. **系统设计**

考虑如何做

**首先**，对反映问题本质的数据抽象（问题的数据结构设计）

**然后**，考虑对数据如何进行操作以获得问题的结果（算法设计）

在*面向过程程序设计*时，将两者分开考虑

在*面向对象程序设计*时，将两者结合考虑

1. **编码**

考虑如何体现

用伪代码（或自然语言或框图）描述算法，再用某种程序设计语言编写程序（编程实现）

程序文本保存在一个文件或多个文件中，**包含程序文本的文件称为源文件，即源程序文件**

1. **测试与调试**

源程序文件要经过编译程序翻译成等价的目标程序文件，并将一系列目标文件及库文件连接在一起生成可执行文件，程序才能被计算机执行，程序还要经过测试与调试才能正式运行

**调试**：找出程序中的错误并改正，即查错。

**测试**：用预先设计的一批测试用例（输入数据和预期结果）运行程序检验程序是否出错

通过测试发现程序中的错误，通过调试进一步找出错误的位置并改正错误。

错误分类：

1. **语法错误**

源程序中存在违反C语法规则的地方。程序编译时可以发现这类错误并予以提醒

1. **逻辑错误**

使程序无法完成预期功能的错误。如除打成乘之类的

1. **运行异常错误**

程序对程序运行环境的非正常情况考虑不足而导致的程序运行异常终止。

测试与调试只能发现程序有错，不能证明程序正确。

1. **整理文档资料**

将平时的源程序和各种文字资料进行归类保存以便查询。

还要编写使用和维护该程序的说明书，供程序用户参考。

1. **运行与维护**

通过测试后的程序可以运行，但程序仍需要维护。

维护包括：

*正确性维护*，纠正程序在测试中未被发现的错误

*完善性维护*，扩充程序的功能

*适应性维护*，保证程序在修改后能继续工作。

程序的功能如进行较大更改，应发布程序的新版本。

**③C语言发展**

1. **C语言发展简述**

1972年，为编写UNIX操作系统，D.M.Ritchie设计C语言。

C语言在发展过程中有众多不同版本，其中有三个重要的标准

1. 1978年的标准C。定义了C语言但是没有C库。
2. 1983年的ANSI C，1989年被通过，在1990年ISO对其少量修改。亦称为C89或C90。添加了C库。
3. 1999年的C99。并非所有的编译器都能接受C99.
4. 2011年的C11

C语言纵使版本不断更新升级，其特征保持不变。

1. **C语言的优点**

* 有自顶向下的**规划设计特性**
* **运行效率高**
* **可移植性**好。源代码无需修改就能在不同计算机系统中成功编译。
* 强大灵活
* 应用**范围广**，包括操作系统、游戏、电影、软件开发等等

**④C语言程序的构成**

与自然语言相似的是，C语言有基本符号（字符）构成一系列单词（语法元素），由多个单词构成句子（语句），再由多个语句构成程序。但C语言语法规则较为严格，程序文本所代表的语义是单一的。

通常C语言程序构成为：

1. **注释**

是给用户看的，便于使用者理解代码意义。

C语言中的注释为**“/\*”和符号“\*/”之间的文字**，可以占连续的几行。

C99标准中还有另一种注释。某语句行从符号**“//”开始至本行结束的所有字符均为注释内容**。

1. **预处理命令**

预处理命令并非C语句，是在源程序编译之前由预处理程序处理的命令。预处理命令**以#开头，写在C语言的开始部分**。常见的编译预处理命令有**#include<stdio.h>。**用头文件的内容来**替换#include指令**

Include命令将函数信息包含进本程序中。Stdio.h文件内有库函数的信息。

1. **程序级定义**

对整个程序有效的一些定义。写在#include命令之后。如：

#define PI 3.14159

1. **函数原型声明**

在C程序中，所有计算都是有相应函数完成的。**函数**是能够完成**一定操作、具有具体名称**的一组**语句**。函数需要**被定义后使用**，即需要函数原型声明。

1. **main函数**

**main函数**（main（））是被称为**主函数**的特殊函数。该函数作为程序执行的**起点**，不论该函数在程序中的哪个位置。main函数最好被指定为int型。void并非所有标准都接受。

书写模板

Int main()

{

（编程语句）

return 0;/\*程序结束运行\*/

}

return语句在void函数中使用时**不需要返回某一个具体的值**，仅仅是**起到结束程序**的作用。

**f.** **用户自定义函数**

为了使程序结构清晰、管理方便，通常将一些关系密切的函数组织在一起放在同一个文件中。因此C语言通常由多个源文件组成，且每个源文件都可以单独编译。

* 函数组成

1. 函数首部

包括**函数名、函数类型、函数属性、函数参数名、参数类型**。若函数没有参数，可以打空括号或在括号内写void

1. 函数体

**声明部分**：定义在本函数用到的变量。

**执行部分**：指定在函数中所进行的操作。

某些情况下，也可以没有声明部分和执行部分。

**g. 其他书写注意事项**

* 程序设计中应使用英文的标点符号
* 每个语句的最后以分号结束
* printf函数书写模板

printf(“格式控制字符串”,表达式列表);

可按格式控制字符串中的格式依次输出表达式的值。**%d称为整形格式控制符**，可输出**整数**表达式的值。要输出几个整数表达式的值，就需要几个%d。

如printf(“%d,%d,%d,”5,2\*5,5\*2\*5)。输出结果为5,10,50（逗号**为非格式符，原样输出**）

printf(“length is :%d,width is :%d”,5,2\*5)。输出结果为length is :5,width is :10

如要使内容在下一行输出，则可以在格式控制字符串中**添加\n (换行符)**

如printf(“the result is:\n%d”,2\*5)。输出结果为：

the result is:

10

* 用printf函数书写文字时要记得加双引号，不然会被当作变量。
* 程序在被编译后生成.exe文件

**⑤Dev-C++5.11简介**

Dev-C++是windows下的C/C++程序的集成开发环境。集编辑、编译、连接、执行、调试功能于一体，提供语法加亮度显示、自动补全代码、函数提示功能。

1. **C程序的编辑与运行**
2. **编辑源程序**

启动Dev-C++5.11

选择“文件→新建→源代码”命令，出现编辑窗口。

源程序输入完毕后，保存。

Dev-C++默认格式为**.cpp**，是C++的源文件。如保存为C程序的源文件，保存类型应选为**.c**

1. **编译和运行的机制**

* **编译**：将源代码文件转换成**目标文件**。目标文件包含机器语言代码，不必是完整的程序代码。
* **链接**：将多个**目标文件**、**启动代码**、**库函数代码**链接形成可执行文件。

启动代码是程序与操作系统的接口。由于不同操作系统处理程序的逻辑不同，所以

启动代码也不同。

1. **Dev-C++的调试**
2. **程序调试的一般过程和手段**
3. **程序调试的一般过程**
4. **人工检查**

编写好源程序后对源程序进行人工检查，仔细阅读分析以找出其中错误。

1. **编译连接检查**

编译源程序，根据系统提示修改错误

要注意的是，有时系统提示的出错行并不是真正出错的行，而是**在提示行的上一行**。

且源程序中的一个错误可能会产生**若干条错误信息**，而第一条信息最能反映错误的位置。最好**修改一次编译一次**。

编译器也有可能输出警告信息，警告信息并非错误，但是有可能是潜在的错误，也要重视。

1. **运行分析**

运行程序，如出现实际情况与预期结果相差甚远或程序运行直接中止死机，通过调试手段找出错误

1. **调试程序的常用手段**
2. **标准数据检验**

用若干组已知结果的标准数据对程序进行检验，结果与预期不符则分析算法是否正确。

1. **程序跟踪**

让程序逐句执行，并通过观察和分析程序执行过程中数据的变化和程序执行流程的变化来查找错误。

* 直接利用集成环境的单步执行、设置断点、观察变量的值和控制程序的运行等功能程序跟踪
* 在程序中直接设置断点，输出重要变量的值等来掌握程序运行情况。

1. **边界检查**

设计检查用的数据时，要重点检查边界和特殊情况，对于分支程序，每一条路径都通过检查

1. **简化**

通过减少程序循环次数、缩小数组规模、用注释屏蔽某些次要程序简化程序

1. **Dev-C++的调试方法和工具**
2. **启用调试工具**

选择“工具→编译”，打开“**编译器选项**”，点击其中的**“代码生成/优化”选项卡中的连接器标签**，将上面的**“产生调试信息”**的值no改成yes，确定。

1. **设置断点**

在需暂停的行上右击，选择**“运行→切换断点”/单击该行的行号**。

1. **进入调试界面**

“编辑”窗口下面有一组选项卡，单击“调试”选项卡上的“调试”按钮/按**F5**

1. **查看变量与表达式的值**

通过观察变量或表达式的值的变化，可以提高查错效率。

按**Alt+a**键，打开**“新变量”**对话框。

在框内输入要跟踪的变量或表达式，它们会被列在左边的窗口内，跟随程序的执行流程显示它们的当前值。

1. **单步执行**

程序执行到断点时会暂停，按**F7**可以使程序逐步执行。

1. **结束调试**

**F6**键结束调试

**⑥提高程序可读性的好习惯**

* 选择**有意义的标识符**并且**写注释**。比如说标识符前面打一个类型的缩写再加一个下划线，就可以知道变量的类型。
* 用**空行分隔**概念上的多个部分

第二章 C语言基础

**①C语言词法**

1. **基本字符集**

基本字符集是语言所能使用的**字母、数字、特殊符号**的集合。C语言中的单词是字符集中的字符根据词法规则构成的，**不能使用基本字符集以外的字符**。

基本字符集包括：

* 英文字母
* 数字符号
* 英文标点符号

C语言中字母大小写有区别。

1. **单词**

单词是由基本字符集中的字符按照一定规则构成的最小语法单位。C语言中的单词由基本符号组成，包括关键字、标识符、运算符、分隔符、字面常量、特定符。

1. **关键字**

关键字也称保留字，是C语言中具有特定作用和含义的单词。

电脑屏幕的照片上有文字

描述已自动生成

注意：

* **不能另作其他用途**，如将其作为变量名或函数名使用。
* 关键字均**由小写字母**组成（除最后三个），用大写就不算了。

1. **特定字**

以固定的形式用于专门的位置，地位等同于关键字。

define undef include ifdef ifndef endif

1. **标识符**

以**字母或下划线开头**，由**字母、数字、下划线**组成的字符序列，用于表示**变量名、常量名、函数名、类型名**。标识符**不包括特殊符号，不能以数字开头**。

注意：

* C语言标识符长度是有限的，**不超过63个字符**。
* **变量名、函数名用小写字母**。**常量名用大写字母**
* **避免用下划线做开头。**因为操作系统和c库经常使用这样的标识符。

1. 白板上的文字

   描述已自动生成**运算符**

也称操作符，表明对数据如何操作。用单字符、双字符、关键字表示。

算术运算符中要注意**%是求余符号**。

* **自增（++）自减（--）符号**

i++→i=i+1 ++i→i=i+1

自增/减在前，那么变量变化后再使用；否则先使用再变化。**自增/自减不能用于常量或表达式。**同时函数参数不要用自增自减

**算术运算符从左至右，赋值运算符从右至左。**如a=b=c就是c先赋值给b，b再给a(三重赋值)

取字节运算符sizeof若是对赋值表达式求长度，则赋值操作不会进行。

* 运算符的优先级：

1. **单目运算符(从右向左)**

取负-，按位取反~，自增减，解引用\*，取地址&，非！，取字节sizeof。

1. **算术运算符**
2. 乘除余
3. 加减
4. **关系运算符**
5. 大于、小于、大于等于、小于等于
6. 等于、不等于
7. **逻辑运算符或与**
8. 与&&
9. 或||
10. **条件运算符**
11. **赋值运算符**
12. **逗号运算符**
13. **分隔符**

是用来**界定或分割**语句中的语法成分

包括**逗号，分号，单引号和双引号，大括号，注释，空格**

分号代表语句的结束，双引号表示字符串的开始和结束，空格在相邻的关键字和标识符起分割作用。（连续多个空格与一个空格的作用相同）

**运算符也具有分隔符的作用**。

编程时，由单词组成语句，一系列语句组成函数体，由函数体组成C程序。

**②图示, 示意图

描述已自动生成数据类型**

程序通常由数据输入、数据处理、数据输出三部分组成。为了对数据做到正确处理，要对数据的特性进行描述。数据的特性包括**数据的结构和对数据的运算**。

1. **数据类型概述**

C语言中的数据被分为不同的类型，分别以不同的方式进行存储。

每一种数据类型可以看成由两个集合构成：**值集和运算集**。

* **值集**：描述该数据类型包含哪些值
* **运算集**：描述对值集中的值可以进行哪些运算。

如整型的值集是由一定范围的整数构成的集合，运算集包括加减乘除余等。

* 同一类型的数据占用相同大小的存储空间。
* **存储的基本概念**

位、字节、字是描述计算机数据单元或存储单元的术语。

**最小的存储单元是位(bit)**

**8位组成一个字节，能够表示一个0-255的整数或一组字符。**

1. **数据类型分类**
2. **基本数据类型**

C语言系统已预先定义的数据类型，可被用户直接使用。

1. **构造数据类型**

又称为用户自定义型。在基本数据类型的基础上，由用户利用C语言提供的数据类型构造机制构造出来的数据类型。

不同类型的数据在数据表示形式、合法取值范围、占用内存空间大小、参与的运算种类都有所不同。

1. **基本数据类型**

包括整型、实型、字符型、空值型。

1. **整型**

用关键字**int**定义。值集理论上是所有整数，但是受到计算机存储单元的限制是有限的。且不同C语言可表示整数的范围不同。

在程序中，使用超出表示范围的整数称为溢出。

1. **基本整型（int）**

用**四个字节**表示。值集范围为-2147483648~2147483647。在16位机上int只占2个字节。用int型的参数传递速度更快。

1. **短整型（short）**

用**2个字节**表示。值集范围为-32768~32767.

1. **长整型（long）**

用**4个字节**表示。值集范围**同int**。除此之外还有**long long int（双长整型）**。占用**8字节**。部分C标准函数**要求使用long类型的值**。

1. **无符号基本整型（unsigned int/unsigned）**

值集范围为0~4294967295

1. **无符号短整型（unsigned short int/unsigned short）**

值集范围为0~65535

1. **无符号长整型（unsigned long int/unsigned long）**

值集范围为0~4294967295

无符号整型与有符号整型所占内存**大小相同**，但由于有符号整型16位中的第一位用来表示符号（0为正，1为负），无符号整型不用表示符号，所以无符号整型表示正数的范围比有符号整型**大一倍**。

整型数据允许进行**算术运算、关系运算。可参与逻辑运算、位运算**。

1. **实型**

又称为**浮点型**。同样的，实型也只能表示实数的一个子集。有关浮点数的运算输出时默认保留6位小数。

浮点数用**科学计数法**表示。即a\*2^b.其中a为尾数，b为指数。浮点数在实际存储时是储存其尾数和指数，两者均用二进制表示。

值得注意的是，一些十进制小数无法精确地表示成二进制小数，只能近似的表示。如7.0可能会存成6.99999.而0可能要写成1e-16这样很小的数。浮点数**不应该用相等关系**判断

**单精度浮点型（float）**

占用**4个字节**。数值范围约为±（3.4\*10^-38~3.4\*10^38），有效位数为6~7位。

**双精度浮点型（double）**

占用**8个字节**。数值范围约为±（1.7\*10^-308~1.7\*10^308），有效位数为15~16位。

双精度浮点型还可以加上修饰符成为**long double**用以表示精度更高的浮点数。不过多数C编译器没有设置long double。

有效位数表示浮点数的精度。如VC中float型有7位有效数字，那么765432.179就会变成765432.1。

1. **复数类型和虚数类型**

**C99和C11支持**。

如**float\_Complex,double**\_**Complex,long double\_Complex.**

这样的变量应该包含**两个值**以来分别表示复数的**实部和虚部**。其中Complex换为**Imiginary**即为虚数类型。

若包含**complex.h 头文件**，即可用**complex**代替**\_Complex**,**imaginary**代替**\_Imaginary**,用**I**代替**-1**的平方根。

1. **字符型（char）**

用于描述**单个字符数据**。在内存中以ascii码形式存储。一个字符型数据既可以按字符形式输出，也可以按整数形式输出。以整数形式输出时，实际上是输出该字符的ASCII码。字符型数据**可以当作整型数据进行算术运算**。

字符型数据也可以用unsigned的修饰。有符号时取值范围为0~127.无时0~255.

1. **空值型（void）**

又称为**无值型**，其**值集为空集**。可出现在函数定义的头部。

* 当函数返回值类型说明为void时，则该函数没有返回值。
* 当void出现在函数定义的形参位置时，表示该函数没有参数。

此外，它也可用来表示**通用指针类型**。

当数据超过其类型范围时，会产生**数据溢出**现象。C语言并**不会对数据溢出报错**。

如32767为short时，32767+10=-32759

还有如2.0e20f加1.加1后存储该数据需要21位，超出了float的长度，便会产生溢出现象。

**③常量与变量**

1. **常量**

指在C程序运行过程中其值不可改变的量。

1. **字面常量**

也称**直接常量**。以字面值的形式**直接**出现在程序中，可分为以下四种。

1. **整型常量**

C程序中的整型常量可以用**十进制**、**八进制**、**十六进制**三种形式书写，编译系统可自动将其转换为二进制形式存储。

* **十进制整数形式**：与数学中整数表示方法一致。但程序中不允许用逗号分隔数据，如将30000写成30，000.
* **八进制整数形式**：以0开头，再接着八进制数字串。
* **十六进制整数形式**：以0x或0X开头，再接着十六进制数字串。

八进制和十六进制通常用于简洁地表示长串的二进制数，如存储单元地址。

用**0/0x**开头的数字会被默认为**八进制/十六进制**.0010是正确的八进制数，**百位可以为0**.

* 在字面整型常量后加上**字符后缀l/L**，表示该常量为long int型。（5983672L）
* 在字面整型常量后加上**字符后缀u/U**，表示该常量为unsigned int型。（5983276U）
* 十进制负数转换进制时，**不管符号先化成二进制**，再**按位取反**，再**加1**，再换成**对应的进制**。
* **同时加上U和L**，表示该常量为unsigned long int型。

根据字面整型常量的类型，内存空间也会不同。如12L可占4个字节，而12只占两个。

1. **实型常量**

实型常量简称**浮点数或实数**。浮点数仅采用十进制形式书写。可用小数点表示法和指数表示法表示

* **小数点表示法**：与数学中实数表示一致，可以出现正负号。**当小数点前或后的数为0时，0可以省略。**如23.即23 .0 .23即0.23 都将会被记为浮点数。但如果打成23就会被记成整型。
* **指数表示法**：类似科学计数法。但是**将×10改为e/E**。如31.4记为0.314E＋2.其中**0.314被称为尾数，b为阶码（必须为整数）。**

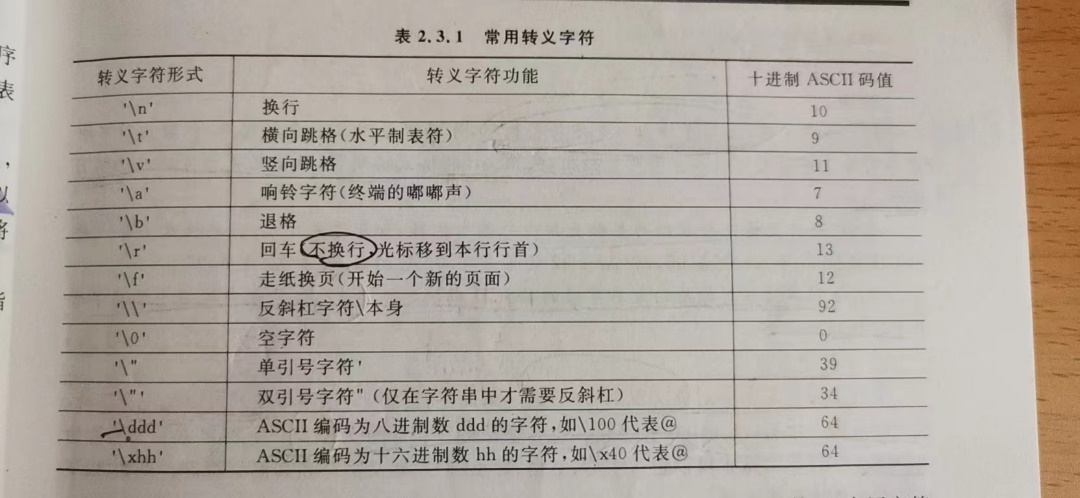
**e前面不能没有数字，且e后面的数字不能加小括号。**

浮点数默认为**双精度型**。可在常量后加上**后缀f表示该常量为单精度型**。

若要输出单精度型浮点数，可使用**格式控制字符%f**，默认情况下输出6位小数。

如，printf（“%f”，0.314e+2f）可输出31.400000

double型浮点数输出可使用**格式控制符%lf**，输出结果相同。

1. **字符型常量**

C语言中的字符型常量是**用英文单引号‘括起来的单个字符**，通常占据**一个字节**的存储空间。实际上，存储的是该字符的ASCII编码。**当字符型常量参与数值运算时，换算成其ASCII码值**

**单引号是区分字符与非字符的标志。**

**ASCII码值小于32的特殊字符，无法直接用字符常量表示，在C语言中用以字符\开头的“转义字符”表示**

* **不常见的转义字符**
* **响铃\a**

响铃**不允许转变**活跃字符(光标)。

* **退格\b**

通常退格**不会擦除**退回所经过的字符，但在键入新字符时会**取代**原字符。

* **回车\r**

光标移到**本行行首**

* **其他**
* 每行各个水平制表符的位置是对齐的，通常相隔**8个字符**的宽度。
* **%c用于输出一个字符型数据**，如用%d则输出其ASCII码。
* 小写字母与其相应的大写字母的ASCII码值之差为**32**.
* 用转义字符输入八进制数时，必须写三位**/ddd**。转义字符对**八进制或十六进制**的表达是**有范围**的，超过范围的值转义字符的值将是**未定义的**。
* 赋给一个字符常量大于256的数，整型输出时会输出256的余数。

1. **字符串常量**

简称**字符串**。是由一对英文双引号括起来的字符序列。该字符序列可以为单个字符或多个字符，也可以为**空串**。程序中使用字符串常量会自动生成一个指向字符的常量指针。

当字符串常量出现在一个表达式中时，表达式所使用的值是其**常量指针**而并非字符

因此可以将**字符串常量**赋值给一个**指针**，但不能将字符串常量赋值给一个**字符数组**。

当双引号本身作为字符串中的字符时，应写成**：\”**

字符串在存储时，系统会自动在字符串末尾加一个字符**‘ \0’**。因此字符串常量实际占用的内存字节数比字符串长度**多一个字节**。

用scanf()读入字符串时，**不用&**。

1. **常变量**

关键字**const**定义

格式为：

**const类型名 常量名=值**

const常量其实是值不允许改变的变量，占用内存。const用起来比#define会更加灵活。因为const是值的替换，而#define仅仅是输入的替换。

有个不常用的命名约定，会在名称前带c或k前缀来表示常量。

* **const的扩展**
* **const数组**

在数组声明时类型前加上const，则该数组的数据被改动时，编译器会发出警告。

* **const指针**

1. 指针声明时**类型前加上const**，这样的指针**不能通过解引用来改变值**，但是可以改变**指向的对象**。
2. 指针声明时**在\*的后面加上const**，这样的指针**可以改变值**，但是**不改变指向对象**。
3. 在两处**都加上const**，这样的指针**不可以改变值也不可以改变指向对象**。
4. **符号常量**

用编译预处理命令**#define**定义

格式为：

**#define 常量名 值常量**

此为**宏常量**，没有数据类型，不占内存。不能对其赋新值。

使用符号常量的好处：

1. 含义清楚，增强程序可读性。
2. 便于修改
3. 便于保持常量的一致性
4. **变量**

变量是指**在程序运行过程中值可以被改变的量，代表一个有名字的、具有特定属性的一个存储单元**。程序用变量来存储数据。

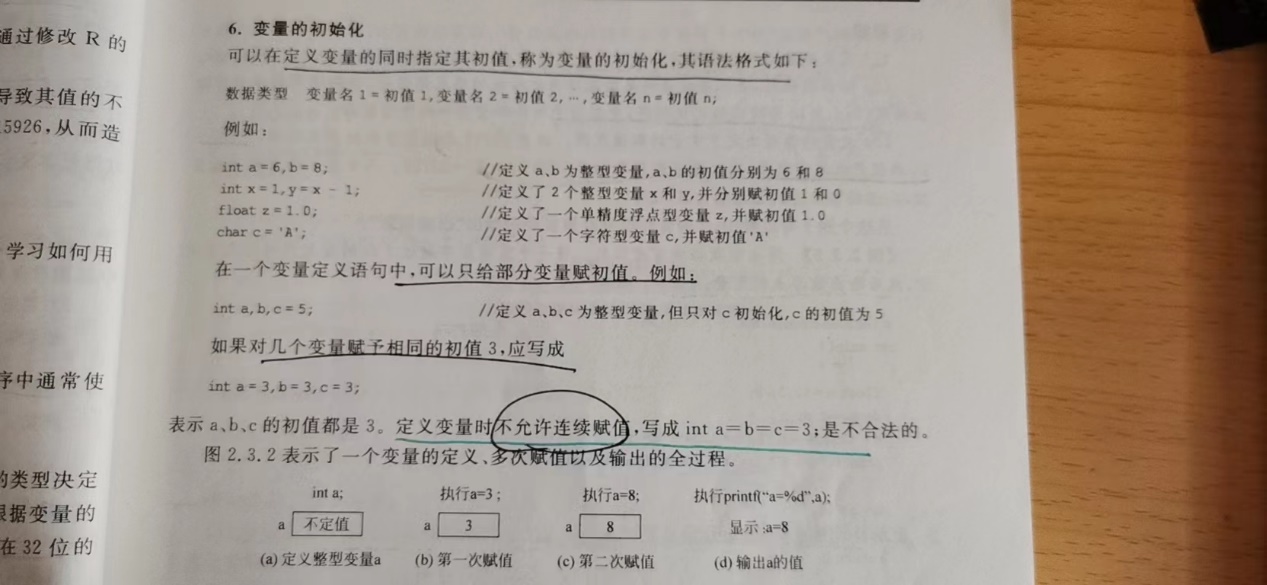
1. **变量的属性**
2. **变量名**

变量的名字。是标识符。

1. **变量类型**
2. **变量的值**

指变量**所表示的数据**，是与该变量相关的存储单元的内容。通过赋值可以改变变量的值，**在下一次赋值之前，变量的值将不会改变。在程序运行的某一时刻，每个变量只能存放一个值**。

1. **变量的地址**

指在**编译或运行时程序中的变量分配到的一定大小的内存空间的首个字节地址**。访问变量内存时，可以通过变量名或变量地址访问。

1. **变量的定义**

格式如下：

数据类型 变量名1，变量名2，… ，变量名n

1. **变量的初始化**

可以在定义变量的同时指定其初值，成为变量的初始化，其语法格式如图：

**④表达式**

可分为**算术表达式**、**赋值表达式**、**关系表达式**、**逻辑表达式**、**条件表达式**、**逗号表达式**

1. **算术表达式**

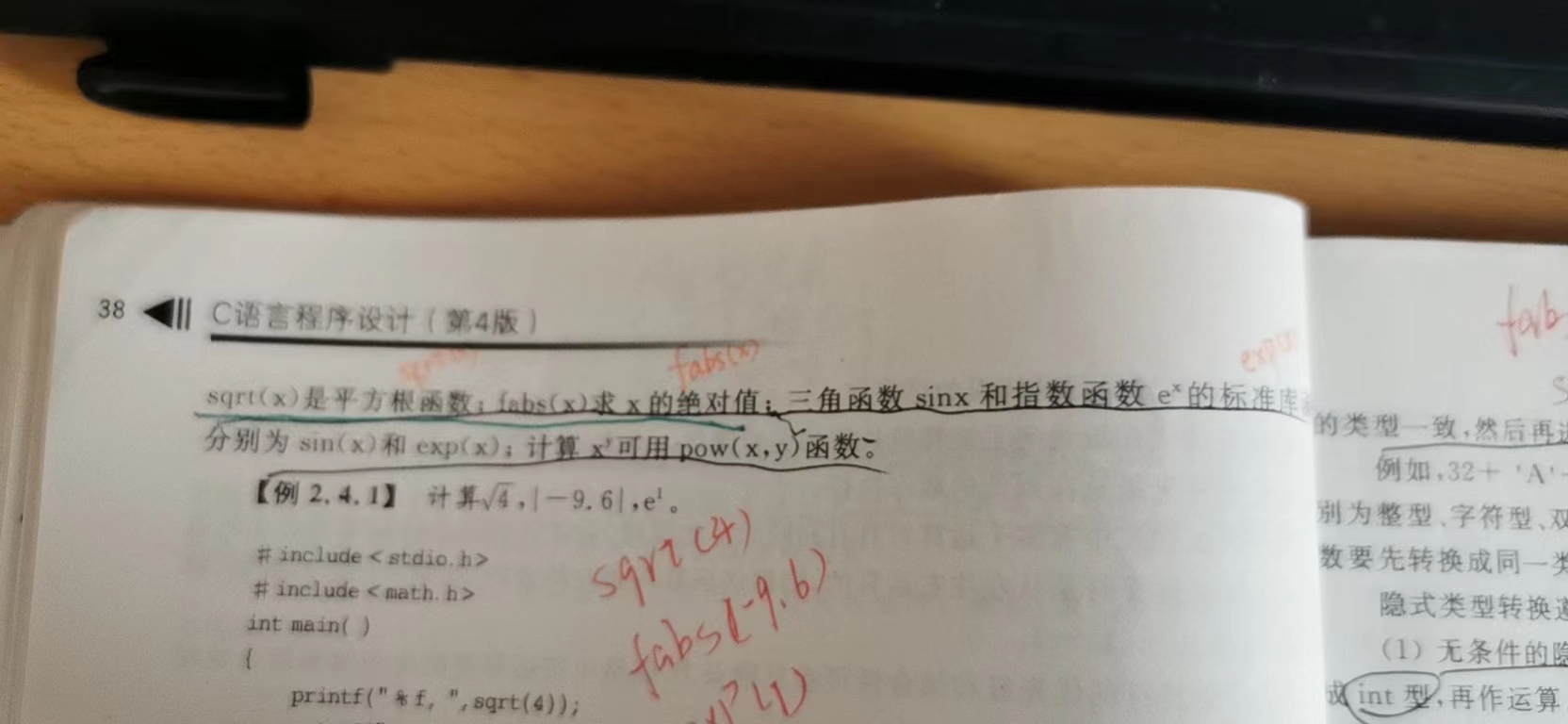
实现通常意义上的数值运算。其运算对象为数值类型，即**整型**、**字符型**、**浮点型**。

1. **基本算术运算**

* **单目**算术运算符：**正负**
* **双目**算术运算符：**加减乘除**
* 任意两个整型相除时，其结果**也为整型**。如5除以2为2.
* 两个数据**至少有一个为实型**，则运算结果为**双精度实型**。如5.0除以2为2.5
* **求余运算符%**两侧的数据必须均为**整型数据**，其运算结果为**两个数据相除的余数**。
* 当**除法运算符和求余运算符**的数据**有负数存在时**，使用时要谨慎，结果不确定。
* 两个**符号相异**的数求余，结果的符号取决于**被除数的符号**。

C99规定负数应当**趋零截断**，-3.8→-3

1. **数学函数与数学公式**

C语言不提供开方和乘方运算符，需要调用标准库函数中的相应数学函数完成。

调用标准库函数，需要加一条**预处理命令为#include<math.h>** 。否则用不了。

1. **其他**
2. 小括号可以多层嵌套。中括号和大括号另有用途。
3. Π（派）不是基本字符，不能出现。
4. **类型转换**
5. **隐式转换（自动转换）**

当不同数据类型的数据混合运算时，编译系统自动将其调整为同一数据类型。

1. **无条件的隐式类型转换**：char型和short型先转化为int
2. **同一类型的隐式类型转换**：将精度较低的数据类型转为精度较高、范围较大的类型

即int→unsigned→long→float→double

3. **赋值时进行的隐式类型转换**：对变量赋值时，表达式计算的**最终结果**会被转换成**被赋值变量的类型**，这样的过程可能导致**类型升级或降级**。降级将会带来**精度的损失**。

**例如当double赋值给float时，会四舍五入；要求保留几位小数输出时也会四舍五入。**

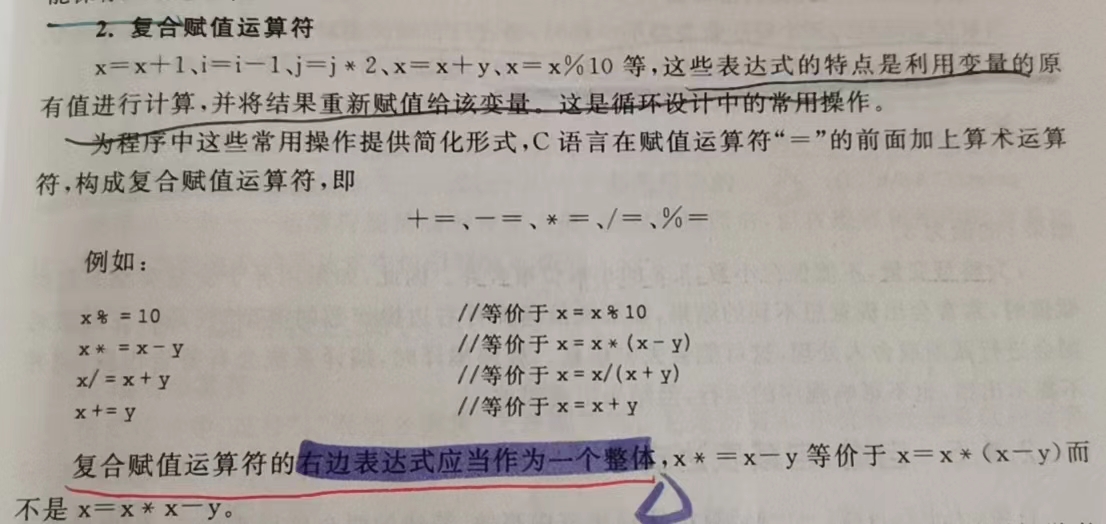
1. **显式类型转换**

**格式为：**

**（类型名）数据**

1. 从表示范围大的类型强制转换到表示范围小的类型，可能会**丢失精度**。
2. 类型转换不会改变变量**本身的数据类型和值**，只**对其进行的运算有影响**。
3. **赋值表达式**
4. **简单赋值运算符（=）**

方向为**从右至左**。赋值运算符左边**不可以为常量或表达式，只能为变量**。

1. **复合赋值运算符**

**复合赋值运算符为右结合，先算右边。**

* 赋值运算时的隐式类型转换

将赋值运算符**右边表达式类型转换为左边变量的类型**。

如int i=3.8 i会变成3

1. **逗号表达式**
2. **逗号运算符**

逗号既是分隔符，也是运算符。从**左向右看**。

1. **逗号表达式**

用逗号运算符将表达式连接起来的表达式称为逗号表达式。

从左至右依次计算每个表达式的值，**最后一个表达式的值为整个逗号表达式的值**。

**e.** **副作用与序列点**

1. **副作用**

C语言认为，函数的主要目的是对表达式求值，而副作用是**对数据对象或文件的修改**。

例如printf函数的返回值是待显示字符的个数，**副作用是打印数据**。

1. **序列点**

是**程序执行的点**，所有函数的副作用必须在**下一个序列点**(下一步程序)之前完成。

如赋值运算符、自增自减运算符必须在程序执行下一条语句之前完成。

* **序列点种类**

1. **完整表达式**

完整表达式**并非子表达式**，而是**整个表达式。**

1. **逻辑运算符||和&&**

逻辑运算符的**两侧各产生一个序列点**。

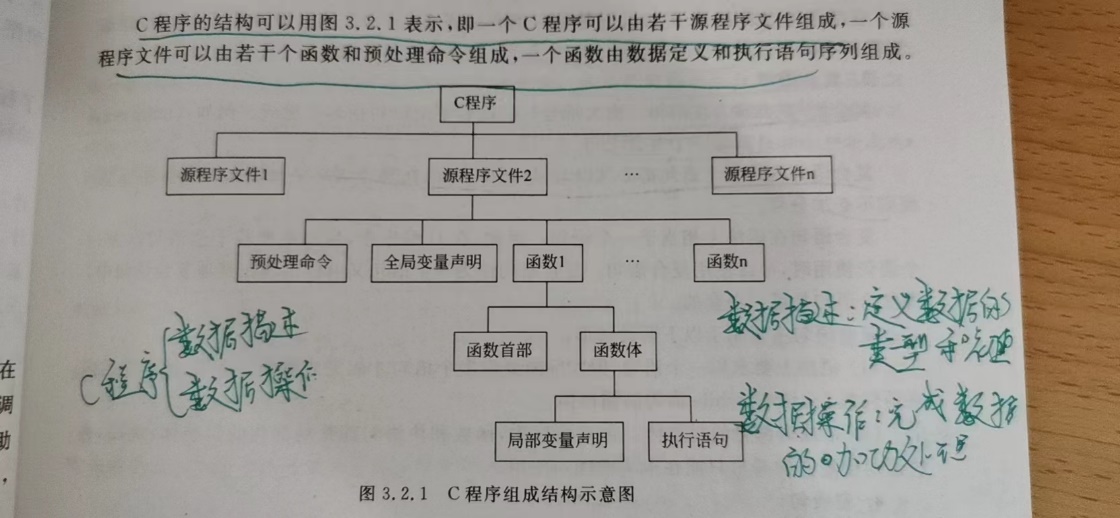
1. **条件运算符？**

条件运算符中的**条件是一个序列点**。

1. **逗号运算符**

一个逗号运算符就代表**一个序列点**。

第三章 顺序程序设计

**①结构化程序设计方法**

1. **自顶向下分析设计问题**
2. **模块化程序设计**

**②语句**

语句是C程序的**基本功能单元**，一个C程序包含若干语句。**语句的意义称为该语句的语义。**

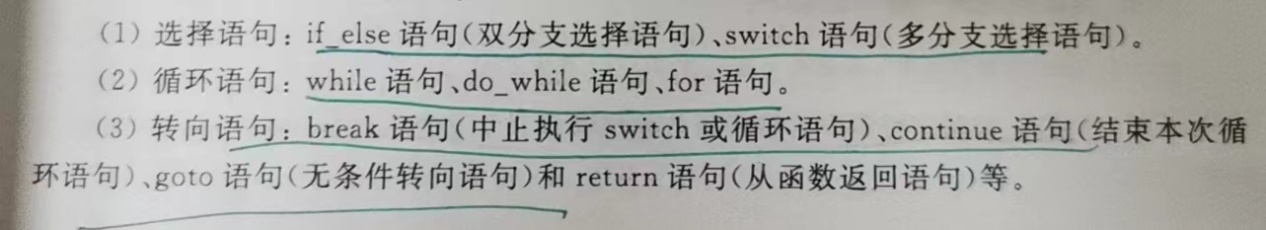
C程序一般都包括**数据描述**和**数据操作**。数据描述**定义数据类型和初值**，数据操作完成**对数据的加工处理**。

1. **表达式语句**

在表达式的末尾加上分号。表达式语句实现对数据的处理。

最典型的表达式语句是赋值语句和函数调用语句。

1. **控制语句**

能够改变程序的执行顺序。有以下9种语句。结构化程序设计基本不用goto语句。

1. **复合语句（块语句）**

由大括号把若干个语句括起来组成。**块语句中的每个语句都要分号，但是大括号后不用。**

**复合语句可以嵌套。**

复合语句主要用于以下情形：

* 语法上要求用一个语句，但实际需要多个语句才能完成操作。如**if的内嵌语句或for、while的循环体**
* 形成局部化的封装体。在块语句中定义的变量只能在本块范围内使用。

1. **空语句**

只有一个分号的语句。可用来作为流程的标记点，从程序其他地方转到此处。或用来作为循环语句中的循环体，表示循环体什么也不做。

**③输入输出函数(I/O函数)**

程序一般由三部分组成：输入初始数据、计算处理、输出结果。

1. **有关概念**

* 输入和输出是以计算机主机为主体而言的。输入设备向计算机称为输入，计算机向输出设备称为输出
* C语言**本身不提供输入输出语句**。printf和scanf都是库中的函数，**并非关键字**。各种C语言编译系统提供的系统函数库是各软件公司编制的，其中的函数已进行过编译并称为目标文件。它们在程序连接阶段与由源程序经过编译而得到的目标文件连接形成.exe。
* 不同的编译系统所提供的函数库中，函数数量、名字、功能是**不完全相同**的。
* 要在程序文件的开头用预处理指令#include把有关头文件（.h）放在本程序中。调用不同的库函数，要**把不同的头文件**包含起来。
* #include指令用双引号和尖括号的区别
* 用**尖括号为标准方式**。编译系统会在**C编译系统的子目录**中找头文件。适用于**系统提供的头文件**
* 用**双引号**，编译系统会在**当前目录**找头文件。双撇号中最好把文件路径写出来。适用于**用户自己编写的头文件**

1. **缓冲区**

内存中预留的**存储输入/输出数据**的部分。

若**无缓冲输入**，字符输入后程序**立即使用**。比如打入12会直接使用1再是2.

若**有缓冲输入**，输入的字符会被**存储在缓冲区**中，按**换行**之后程序才可使用输入的字符。

* **完全缓冲I/O**

缓冲区**被填满时**才刷新缓冲区，通常出现在**文件输入**中

* **行缓冲I/O**

在出现**换行符时**就刷新缓冲区，通常出现在**键盘输入**。

1. **格式输出函数**

**printf函数称为格式输出函数**。以**字符串的地址**作为参数。调用形式为：

printf（“格式控制字符串”，表达式1，。。。，表达式n）

* 格式控制字符串是用双引号括起来的字符串，用于**指定输出格式**。由普通字符和格式控制符组成。格式控制符由**%和类型描述符**等组成，将输出的数据按指定的格式输出。

格式控制字符串中以%开头的格式控制符个数**必须**与输出列表中表达式的个数及类型精确地匹配。如果**要输出%，则必须用%%来打印**。

1. 白板上的文字

   描述已自动生成**整形格式控制符**

用于控制整型数据的输出格式，包括十进制、八进制、十六进制。

1. **十进制格式**

* 基本整型：%d/%md
* 长整型：%ld/%mld
* 无符号基本整型：%u/%mu
* 无符号长整型：%lu/%mlu

1. **八进制形式**

* 基本整型：%o/%mo
* 长整型：%lo/%mlo

1. **十六进制形式**

* 基本整型：%x/%mx
* 长整型：%lx/%mlx

**4. 输入确定字段宽度**

m为整数值，**表示字段宽度**

若数据位数小于m，则左端补以空格；大于m，则按实际位数输出。

不打m，就输出数据的所有数位。

足够大的固定字段宽度可以让输出变得整齐美观。

**5. 显示各进制数的前缀**

**%#0/%#0x**

**6. 限定数据字节数**

只取内存中二进制格式下数据的前几个字节，可能会改变数据。

修饰符为**h。如%hd。**

**7. 指定输出short型/long型**

指定**输出short型，后缀为h；**指定**输出long型，后缀为l**

**8. 补零输入**

**使不足字段宽度的空位全部补上0；当指定左对齐(-)或有指定精度时，忽略补0**

**9. 带正号输出**

**当输出数据为正数时，在%后打+可以使数前面输出一个正号；不是正数没有正号**

1. **浮点型格式控制符**
2. **以小数形式输出浮点型数据**

%f/%m.nf/%-m.nf

%f用于输出单精度和双精度浮点数，没给出输出数据的位数，则按系统默认6位数输出。

m表示输出数据所占的总宽度（**包括小数点所占的1列）**，n表示**小数部分所占的位数**。

补空格规则同整形格式控制符。

%-m.nf跟不加负号的基本相同，但它**会使数据向左靠，右边补足空格**。

double型双精度数据输出时，还可以使用%lf，但与%f无区别。

1. **以指数形式输出浮点型数据**

%e/%m.ne 如3.5→3.50000e+000 默认保留小数位中算上e。

适用于非常大或非常小的数，输出结果比%f简洁。**只能输出浮点型数据**，输出整型就会错误。

1. **以普通形式输出浮点型数据**

%g

由系统根据数值大小**自动选用%f或%e**。若实现不能确定输出浮点数的宽度，可以用%g格式输出。不输出无意义的0.

1. **字符型格式控制符**

%c/%mc

m是输出宽度，同理。

1. **字符串格式控制符**

%s/%ms/%m.ns/%-m.ns

m表示输出宽度，n表示只输出**字符串的左边n个字符**。

1. **格式输入函数**

**scanf函数称为格式输入函数**，按指定格式从键盘读取数据并赋给指定变量。调用形式为：

scanf（“格式控制字符串”，输入项地址列表）

1. 输入项地址列表由若干个地址组成，以逗号分隔。每个地址对应输入数据所要存储的内存地址，可以是变量的地址或数组名。
2. **输入long int必须用%ld，输入double必须用%lf/%le。**
3. scanf中也可以使用%md、%\*md的格式控制符。**%md是从左向右截取m列输入。%\*md是跳过m列输入。%\*d是直接跳过这次输入。**
4. 以**%d、%f**格式控制符输入多个数据时，数据之间以一个或多个空格间隔，也可以用回车键、tab间隔。**但用%c输入时，空格、回车键、跳格键被当作字符输入**。
5. 如果格式控制字符串中还有其他字符，则在输入数据时，**对应**的位置上应输入**同样**的字符。
6. 输入数值数据时，输入**空格、回车、tab或非法字符**，数据输入视为结束
7. 当输入错误时，程序会死机。
8. scanf检测到输入**非法字符**时，scanf返回值为**0**；检测到**文件结尾**时，会返回**EOF**。
9. **字符输入函数**

**getchar（）**

接受从键盘输入的一个字符。输入abc则相当于打进去一个a。接受的字符包括回车、空格等

**特殊用法**：当希望程序在显示结果后按任意键再退出时，可在return语句前加一个getchar。

1. **字符输出函数**

**putchar（）**

在屏幕的当前光标位置处显示c所表示的一个字符。**如果输入0~127内的整数，会输出对应字符。**

事实上，getchar和putchar都并非真正的函数，而是**类函数宏**。

**f. 其他字符判断函数**

这类函数被放在**ctype.h文件**中，可以用来判断字符**是否是某一类特殊的字符**。

若属于，则返回值为非零，否则为0

****

第四章 选择结构程序设计

**①关系表达式和逻辑表达式**

关系表达式和逻辑表达式是选择结构的条件。

1. **关系表达式**

关系表达式是由关系运算符将两个表达式连接起来的式子，一般形式为：

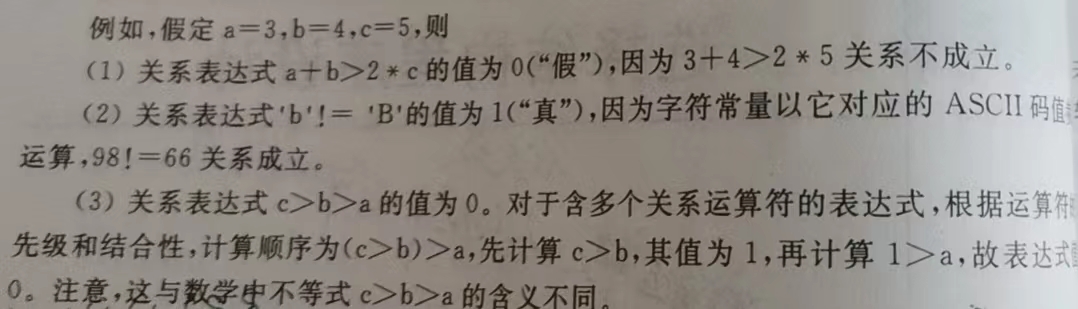
**表达式1 关系运算符 表达式2**

关系运算符有

>（大于）<（小于）>=（大于等于）<=（小于等于）==（等于）!=（不等于）

关系运算符均为双目运算符，优先级低于算术运算符，高于赋值运算符。

**前四者优先级高于后二者**。

* 当判断一个值是否为真时，若**该值非0，则其为真**；否则为假。
* 当判断一个赋值表达式是否为真时，若该赋值表达式**赋值非0，则为真**；否则为假。
* 关系表达式的值是一个逻辑值，即真假。真为1，假为0.

1. **逻辑表达式**

逻辑表达式可以表达比关系表达式更为复杂的条件。形式与关系表达式相同。

逻辑运算符有

！（非）&&（与）||（或）

其中与和或均为双目运算符，左结合；而**非为单目运算符，右结合**。

* 逻辑运算符的运算规则

1. **非**：参与运算量为真时，结果为假；否则为真。
2. **与**：参与运算的两个量都为真时，结果为真；否则为假。
3. **或**：参与运算的两个量都为假时，结果为假；否则为真。

4. **优先级为非>与>或**。

与算术运算符、关系运算符、赋值运算符相比，优先级为：

**非>算术运算符>关系运算符>与>或>赋值运算符**

在计算含有或、与的逻辑表达式时，当从左至右计算时**已能确定总表达式的值**，后面的子表达式**便不计算**。

若包含**iso646.h头文件**，可以用**and代替&&**、**or代替||**，**not代替！**

**c.** **条件表达式**

条件运算符有**？和：**。为三目运算符。？和：**必须配对使用**

条件表达式的一般形式为：

**表达式1？表达式2：表达式3**

先计算表达式1，若其值为真，则计算表达式2，并将其值作为条件表达式的值；否则用表达式的值作为条件表达式的值。只计算其中一个表达式，零一个表达式不计算。

**②if语句**

1. **if语句（单分支）**

格式为

**if（选择条件） 语句**

表达式后的语句为**内嵌语句（子句）**，可以是简单句也可以是复合句（要用大括号）。

若选择条件为真，则执行子句；若为假，则执行if后面的语句。

if的选择条件后面加上；不算错。因为**；算个空语句**。

1. **if else语句（双分支）**

格式为

**if（选择条件）**

**子句 1**

**else**

**子句2**

else必须和if**配对使用**。当if语句嵌套时，else总是与前面**最近的未配对的if**配对。

else后面绝对不能加；否则else控制的语句会变成**；代表的空语句**。

1. **if else if语句（多分支）**

格式为

**if（选择条件1）**

**子句1**

**else if（选择条件2）**

**子句2**

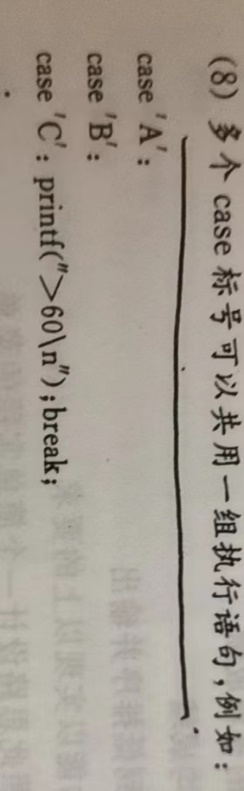
**…………**

当分支中有多个表达式同时满足时，只执行第一个满足条件的。

1. **switch语句（多分支）**

可代替嵌套if语句实现一系列选择。格式为

**swtich（表达式）**

**{**

**case 常量表达式1： 子句**

**case 常量表达式n： 子句**

**default：子句**

**}**

* case后面必须为**常量表达式**，常量表达式起到**标号**的作用。
* 每个case的常量表达式的值**不能相同**，否则会错误。但不同的case可以共用一个语句序列。
* switch语句执行时，从swtich表达式的值确定入口标号，从该入口依次**执行下面所有语句**。
* case子句可用break语句结束switch语句。**有break时，switch结束**；否则**下面所有语句都会执行**
* case子句**可以不用大括号**
* 当表达式的值与所有case的常量表达式不相同时，系统会执行**default后面的子句**。否则不执行任何操作，执行swtich后面的语句。
* 当swtich的分支条件能够表示成**有限个离散值**时，swtich才比较适用。

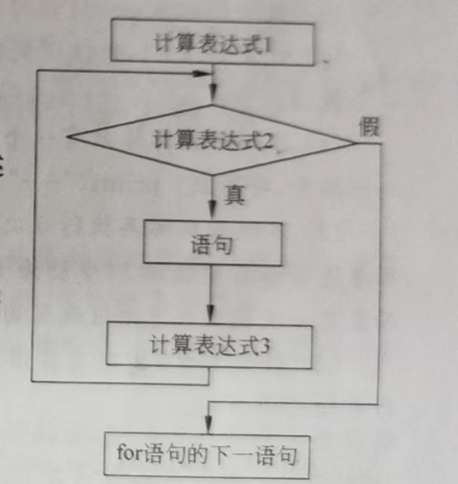
第五章 循环结构程序设计

**①循环简介**

循环结构一般由**循环初始化**、**循环条件**、**循环体**三个部分组成。

1. **循环初始化**：提供初始数据，为循环条件提供初值。
2. **循环条件**：描述重复操作需要满足的条件。
3. **循环体**：重复执行的语句。

**②for语句**

称为**计数型循环**，适合描述**循环次数已知**的循环。格式为：

**for（表达式1；表达式2；表达式3）**

**循环体**

相关概念：

1. 通常根据某个变量的值判断是否继续进行循环，该变量称为**循环变量**。
2. **表达式1为循环初始化**，即给循环变量**赋初值**。
3. **表达式2为循环条件**，为**关系表达式或逻辑表达式**。也可以是算术表达式。只要表达式2的值非0就判定为真。
4. **表达式3为循环变量更新**。
5. 表达式1，2，3**均可以省略**，但是for中的**两个分号不可以省略**。
6. 当表达式2被省略时，则**默认表达式2为真值**，要在循环体中设计其他停止循环的语句。
7. 循环体是循环结构的主体，循环体多于一个语句时则需要用大括号将其括起来。循环体是for语句的内嵌语句。
8. for语句的（）后若加分号，则会使循环体成为空语句，后面的操作**只进行一次**。产生逻辑错误。**不能加分号。**

**③while语句**

在**不知道循环次数**的情况下，根据某个条件是否成立决定是否循环。格式为：

**while（表达式）**

**语句**

相关概念：

1. 表达式称为循环条件。表达式可以是任何表达式，不限性质。
2. 语句称为循环体。
3. while循环结构**先进行条件判断，后执行循环体**。若循环条件一开始就不成立（表达式值为0），循环体一次都不会进行。
4. 循环体中要**有改变循环条件的语句**，否则形成死循环。
5. while中的表达式若含有**自增或自减**，则先对变量进行赋值操作。

**④do while语句**

**先执行循环体后判断条件。一直执行直到满足条件**。格式为：

**do**

**循环体**

**while(表达式);**

当表达式值**非0**（真）时，则**进入循环**；一直到表达式为**0**时**退出循环**。

while表达式后面的**分号不能少**。

**⑤三种循环语句的比较**

一般情况下三种语句是可以互相代替的。

* for语句的功能更为强大，可以完成while、do while的工作。适用于**已知循环次数**的情况下。
* while语句适用于**不能确定循环次数**的情况下；do while语句适用于**先判断后执行循环条件判断**的情况下。
* while语句和do while语句都需要在循环体内加入**循环条件的更新**，否则会形成死循环。

**⑥退出循环的方法**

1. **break语句**

提前结束**整个循环体**。通常作为**选择语句的内嵌语句**，不能单独使用。

break语句除了在循环语句后搭配选择语句使用或者在switch语句中使用外，不能出现在程序的其他位置。**break结束for循环后，变量不会更新**

1. **continue语句**

提前结束**此次循环**，进行下一次的**循环条件判定**。同样是在循环语句中作为选择语句的内嵌语句。

1. **goto语句**

转向语句。语法格式为：

**goto 语句标号：**

**语句标号：语句**

语句标号是一个**标识符**，类似于switch语句中的case子句的常量。语句标号不能重合。

goto的跳跃范围仅限于函数体内，**不能跳到函数体之外**。可以从循环体内到循环体外，但是**不能从循环体外到循环体内**。

一般用于**一次性退出多层循环**。结构化程序设计方法不提倡使用goto语句。

**⑦常用算法**

1. **累加法**

用累加法计算**求和**问题。

1. **穷举法**

先预估问题的求解范围，再对所有可能的解进行逐一验证，最终找出符合条件的解。能用穷举法解决的问题，其解结构一般为**离散结构**。

1. **递推法**

常用递推法求解有关**序列**的问题。

第六章 函数

函数是程序的基本单位，结构化程序设计中的模块。每个函数本质上是一个自带声明和语句的小程序。可以利用函数把程序划分成小块，也可以一次定义之后多次使用。

**①函数定义与调用**

1. **函数定义**

编写完成函数功能的程序块。由**函数首部**和**函数体**组成。格式为：

**函数类型 函数名（【类型名 形参1，类型名 形参2】）----函数首部**

**{**

**声明与定义部分**

**语句部分**

**return 形参n ----函数体**

**}**

函数参数可有可无。

1. **函数类型**：指调用函数后函数值的类型。如函数没有返回值，函数类型要指定为**void**
2. **函数名**：为标识符。函数之间**不能同名**。
3. **形参**：定义函数时，参数没有确定的值，称为**形式参数**。形参的类型需要指定。
4. **return语句**：函数体通过return语句返回函数值，函数体中可以包含多个return语句，return语句也是函数的结束语句。对于void函数，可以不需要return语句。

调用一个**没有return的函数**，函数将**返回任意值**

1. **标准库函数**可以被**重新定义，**但将**失去其原有定义。**
2. **函数调用**

定义好的函数可以被单独编译但不能单独运行。必须被调用才能运行。**主函数main()**由系统调用，其他函数由主函数或别的函数调用。

调用其他函数的函数称为**主调函数**，被调用者称为**被调函数**。格式为：

函数名([参数列表])

* 函数被调用后，参数列表中的参数个数、类型、顺序必须与形式参数**一一对应**。函数调用时的参数称为**实际参数**，实际参数是一个值，而形式参数是用于存储值的变量。
* 若调用无参数函数，就没有参数列表，但是**括号不能省略**。实参之间用逗号隔开。
* 函数被调用后将函数值返回给主调函数，并将控制返回到主调函数的调用点。
* 函数需要**先定义后调用**。因此需要使用的函数要写在主调函数的前面。

1. **函数调用的形式**
2. **作为表达式的一部分**。要求被调用函数能返回一个值。不可以用void函数。
3. **作为另一个函数的参数。**
4. **作为独立语句。**有返回值的函数作为独立语句时丢弃函数值。

* 函数调用**普通变量**是**计算或处理值**
* 函数调用**指针变量**是在**被调函数中改变主调函数的变量**

1. **函数调用机制**
2. 计算实参列表中**各参数表达式的值**
3. 控制流程从主调函数转移到被调函数，为被调函数的所有形参分配存储空间并将实际参数值依次赋予位置对应的形参。若实参类型与形参类型不同，则自动进行类型转换，以形参为准。
4. 执行被调函数的函数体各语句，直到遇到return语句或函数体结束为止
5. 计算return语句中的表达式的值（除了void），将函数值返回给主调函数。
6. 释放被调函数定义的存储空间。
7. 控制返回到主调函数的调用点，并用返回值替代函数调用，接着执行主调函数中的后续语句。

**②函数原型声明**

为了使函数能够先被调用，再被定义。在函数调用之前，给出函数原型声明即可，完整定义可以放在主调函数之后。格式如下：

**函数类型 函数名（形参类型1，形参类型2）；**

* 函数原型就是**函数头加上分号形成一个独立语句。**
* 函数原型声明中，参数列表可以只写函数类型，不写形参的名字。
* 函数原型声明一般**放在源文件的开始部分**，即写在编译预处理命令之后，也可以放在主调函数函数体的开头部分。
* 若函数调用和函数定义分属两个不同的源文件，在函数调用的源文件，也需要进行函数原型声明。
* 当多个函数需要函数原型声明时，可将这些函数原型存放在**一个头文件**当中，用预处理命令包括头文件。而事实上，函数原型声明最安全的做法也是**将函数原型存放在头文件中**。

**③函数间的数据传递**

1. **函数参数**

**实参**和**形参**是**主调函数与被调函数**传递数据的通道。定义函数时形参不会被分配内存单元，在函数被调用之后，形参会被分配内存单元来**储存对应实参传递的数据**。

实参只能将**数据单向传递给形参**。因此实参可以是常量或变量或表达式，形参就是变量。在被调函数中改变形参的值，不会影响到主调函数对应的实参。当实参为变量时，实参和形参可以**同名**。

* **函数参数表达式的计算顺序：**

函数各项表达式参数先**从右向左**计算所有会改变变量值的式子。

对于**后置自增**，则将进行到此处的变量作为该后置自增表达式的值；

对于**复合赋值运算、前置自增、赋值运算**，则将此处视为变量本身。

然后再从**左往右**计算各项参数。

1. **函数返回值**

函数调用结束后返回到主调函数的调用点，可以从被调函数中返回一个值（函数值）。通过return语句来实现。格式如下：

**return 表达式**

表达式数据类型最好和函数类型一致，否则**以函数类型为准**。

return立即结束当前函数运行，后面的return都不会被执行。

* **传值调用和传址调用**

**传值调用**：使用函数时，函数获得实参的一份拷贝，对拷贝进行操作并得到一个返回值。因此函数内的操作就不会影响到主调函数

**传址调用**：本质上同样是传值调用。但当函数获得指针实参的一份拷贝，对拷贝进行解引用操作时，由于访问了某个确切地址上的值，这样的函数可以改变主调函数的值。

函数能够直接对数组进行操作也是由于传址调用。

**④函数的嵌套和递归**

1. **函数嵌套调用**

C语言中的函数体可以调用别的函数。主调函数调用的被调函数中又调用别的函数，称为**函数嵌套调用。**

但是函数体**不可以嵌套定义**。不允许在函数体内定义另一个函数。

1. **函数递归调用**

递归调用是一种**特殊的嵌套调用**。在定义一个函数的过程中**直接调用**或**间接调用函数本身**称为函数的递归调用，这样的函数称为**递归函数**。

**c. 递归的特点**

* 每级函数调用都有**各自的变量**，这些变量同名但实际上并不是一个变量。
* 递归调用通常用**堆栈**来实现。每次递归后的函数都会放在**前一次递归后的函数上面**，达到递归终止条件后，最上面的函数开始执行操作然后**丢出堆栈**，一直到最下面的函数。
* 递归函数中**递归调用前的语句**，按**被调函数的顺序**执行
* 递归函数中**递归调用后的语句**，按**被调函数相反的顺序**执行
* **每级**递归函数的调用都会**从头开始**
* 递归函数必须**包含能让递归调用停止的语句**。
* **直接调用**：在函数的定义中调用函数本身。
* **间接调用**：在函数的定义中调用另一个函数，另一个函数又调用该函数。

能用递归解决的问题有如下特点：

1. 原始问题**可转化为解决方法相同的新问题**，新问题又可转化直至**终结条件**为止。
2. **新问题规模比原始问题小**。

3. 递归在处理**倒序问题非常方便**。

但是递归算法每次调用都会再**产生新的变量**，会**快速消耗内存**，**效率较循环要低**。能用循环代替还是用循环。

**⑤变量属性**

1. **变量的生存期和可见性**
2. **生存期**

指变量占用存储空间的时间期限。

1. **可见性**

指在程序中的哪些位置可以引用该变量。

变量只要可见就一定存在，可以被访问；但是**存在却不一定可见**。例如主函数定义的变量，在主函数调用子函数期间，这些变量虽然存在，但不能被子函数通过变量名引用。

1. **变量的作用域**

指变量在程序中内能被访问的那部分程序段，即**变量可用的那一段程序**。有文件作用域、函数作用域、代码块作用域、原型作用域。

1. **局部变量**

在**函数**或**函数体中的复合语句内部**定义的变量，又称**内部变量**。只在本函数或本复合语句内有效。其他部分不可以使用，局部变量的作用域是**函数级**或是**块级**的。

不同函数变量可以同名，他们代表不同的存储空间，互不干扰。

1. **全局变量**

在**所有函数之外**定义的变量称为**全局变量**，也称为**外部变量**。不仅可以定义在**源文件的开头**，也可以定义**在两个函数的中间或源文件的尾部**。**全局变量和局部变量可以同名**，优先使用**局部变量**

作用域是**从定义的位置开始到该源文件结束**。可以通过引用声明，使其作用域扩展至整个源文件。全局变量在程序的整个执行期间都占有固定的内存单元，并保留其值。**未被初始化的全局变量的值默认为0**

不过除非十分必要，是不建议用全局变量的：

1. 增加程序大小
2. 影响函数之间的独立性
3. 全局变量可能会被各函数调用的时候改变其值，出现错误难以排查。

* 选择变量时原则

1. 变量只在某函数中使用时，不应该定义成全局变量
2. 多个函数都引用同一个变量时，在这些函数之前定义全局变量，定义部分应当尽量靠近这些函数。
3. 若全局变量和局部变量同名，在**局部变量的作用范围内**，同名全局变量不**可见**。
4. **全局变量引用声明**

当希望全局变量的**作用于整个程序**时，可以在引用之前进行全局变量引用声明。格式为：

**extern 类型 变量名**；

全局变量要在其他源文件使用时，必须在别的源文件定义的全局变量进行声明。

**⑥变量的存储类别与链接属性**

存储类别指的是**数据在内存中的存储方式**，变量在程序中可以存在静态内存、运行时堆栈、寄存器。指定存储类别的变量定义格式如下：

**存储类型名 数据类型名 变量名列表**

1. **变量存储区**
2. **运行时堆栈**

在函数调用过程中**进行动态分配**的存储单元。可以存放**auto局部变量**、**形参**、**函数调用时的现场保护和返回地址**等数据

1. **静态存储区**

程序编译时就分配的**固定存储单元**。在块之外声明的变量总是**存储于静态内存**。

若对在块内的函数使用static声明，则只是**改变了其存储位置**，实际上**并不改变其作用域**。

1. **变量存储类别**
2. **auto局部变量**

简称为**自动变量**。属于**自动存储类**。auto可以省略，函数内未加存储类别说明符定义的局部变量均为**自动变量**。自动变量仅在定义该变量的**函数体**或**程序块内**有效。函数的形参也是自动变量。

当该变量被使用时，系统自动为其分配动态存储区的存储单元；语句结束时其存储单元释放。

对自动变量初始化不是在编译时进行，而是在**函数调用时进行**。每调用一次函数重新对自动变量初始化。若自动变量在定义时未初始化，则它的值无法确定。

1. **register局部变量**

**寄存器变量**。属于**自动存储类**。这类变量会存储在CPU的寄存器中，而不像普通变量存储在内存中。但由于CPU寄存器数量少，register变量有限，超过限制的register变量转成自动变量。同时，**寄存器中的变量**没有内存中的**地址**。无法获取他的地址

对register变量操作速度**远快于**存储在内存中的普通变量。

register只能是**局部变量和函数形参**。

现代编译器会自动将**使用频繁的变量**放在寄存器中，无需特别指定。而声明寄存器变量时，若没有足够的寄存器，变量和自动变量无异

1. **static局部变量**

**静态局部变量**。属于**静态存储类**。

在程序运行期间，其占据的存储单元**不释放**。在第二次被调用时，保留上一次被调用后的值。

static不论位置在哪，初值是在**编译时**赋的，在程序运行已经初始化。只是只能在**被定义的块中可见**。在定义时未赋初值，编译时会自动被初始化为0.

函数的形参**不允许使用static**

1. **全局变量**

**外部变量**。属于**静态存储类**。全局变量的初值是在编译时赋的，如未赋初值，编译时自动初始化为0.

如在全局变量前**加static**修饰，则该全局变量的作用域仅限于本源文件。其他源文件使用extern外部变量声明**也不可**使用该变量。

**⑦类型限定符**

1. **const**

const所限定的值**不能够被修改**。

const用于**指针**时情况较为复杂，简而言之，就是**const在\*左边时**，**修饰\*p**，**指针指向的数据不能改变**；**右边时，修饰p，指向指针本身不能改变。**

const最常见的用法就是声明**为函数形参的指针**，这样可以保证一个不改变主调函数的被调函数不会意外改变主调函数的值。

1. **restrict**

只用于**指针**。说明使用该指针是**访问该对象唯一的方式**，给编译器优化的机会。

**⑧随机数函数(在stdlib中)**

1. **rand函数**

rand函数通过线性同余法生成**伪随机数**，在**种子相同**的情况下其得到的**数值相同**。它可以产生一个**0-32767以上的数字**。

1. **srand函数**

可以生成一个**随机种子**。一般可以用**srand(time(0))/srand(getpid(0))**的方式得到一个随机数种子。

1. **生成[m,n]间随机数的步骤**
2. **设定种子**
3. **用rand函数取余.如：**

srand(time(0));

int a=rand()%21+10;// 可以得到一个10到30中间的数字。

**⑨可变参数列表(需要stdarg.h头文件)**

使用可变参数列表时，参数列表的参数可以有**任意个**。

**stdarg.h头文件**定义了**一个类型va\_list和三个宏va\_start,va\_arg,va\_end.**

一个含有可变参数列表的函数包括以下三个部分：

1. **准备访问可变参数**

average(int n\_start,…)

{

**va\_list 可变参数列名**;

int count;

float sum=0;

1. **访问可变参数**

**va\_start(可变参数列名，省略号前最后一个有名字的参数名（即可变参数列起始点）)**

for(count=0;count<n\_values;count++)

{

sum+=**va\_arg(可变参数列名，下一个参数的类型)**

}

1. **完成处理**

**va\_end;(可变参数列名)**

return sum/count;

}

可变参数列的访问可以在某处中止，但是不能从中间开始。

第七章 预处理命令

以#开头的命令。必须从新的一行开始，以#开头，**以回车结束**。

**①宏定义（#define）**

一种**文本替换**机制，这种替换称为**宏扩展**或**宏替换**。

1. **不带参数的宏**

格式为：

**#define 标识符 [字符序列]**

1. 标识符与字符序列应相隔一个以上的空格或一个制表符。
2. 标识符即为宏名。**字符序列为替换文本(字符串)**。程序中一旦出现宏名就会**替换为替换文本**。
3. 宏定义以**回车**结束，不是分号。
4. 宏定义超过一行，则可以在该行行末加一个**反斜杠**打到第二行去。
5. **字符常量**、**字符串常量**、**注释**中出现宏名，不会发生替换。如定义HI hello。print“HI”

还是HI

1. 宏允许**被嵌套使用**。一个宏名可以出现在**另一个宏的替换文本中**。
2. 宏名不是变量，更类似于一个**符号常量**。宏名习惯一般用**大写**。
3. 宏定义中可以没有替换文本。这样的宏定义通常作为条件编译检测的一个标志。
4. **带参数的宏**

格式为：

**#define 标识符(参数表) [字符序列]**

参数表是一系列由**逗号分隔**的**标识符**，类似于函数中的**形参**。标识符与括号之间**不可有空格**。字符序列是一个**表达式**。

这样的宏可起到类似于自定义函数的作用。带参数的宏也被称为**类函数宏**。但它本质上还是文本替换。

* 区别

1. 函数调用时先对实参表达式**求值**，再把实参的**值赋给形参**。而类函数宏只是**替换文本**，可能会导致算式优先级的改变。
2. 函数调用时要求**实参**和**形参数据类型一致**。但类函数宏只是一种符号，文本替换，没有类型。
3. 函数有**返回值**，但是类函数宏**没有返回值**。
4. 函数需要为变量分配内存，但类函数宏不需要
5. **取消宏定义**

若希望宏命令作用于**局部代码**，可以在**不需要该宏定义的代码前**加上**取消宏定义**命令。

格式为：

**#undef 宏名**

有时可能需要**重新定义宏**，在重新定义宏前也**要先取消宏定义**。

**②文件包含**

指一个源文件可以将**另一个源文件的内容全部包含**进来。

格式为：

**#include<文件名>**

或：

**#include“文件名”**

1. 用双引号可以把文件路径**写出来**。但完整的文件名及扩展名**不可省略**
2. 被包含的文件应当是源文件而**不能编译过的目标文件**。
3. 文件包含可以嵌套，但出于对文件可读性考虑，应尽量避免使用文件包含的嵌套。

**③条件编译**

使源程序中的部分代码在**满足给定条件**时进行编译。

利用条件编译可以方便快捷的提供和维护同一软件的多重版本。

1. **#if和#endif命令**

格式为：

**#if 常量表达式1**

**程序1**

**[#elif 常量表达式n**

**程序n]**

**[#else**

**程序n+1]**

**#endif**

其中**elif等同于else if**。中括号中的内容可以省去。

1. **#ifdef和#ifndef命令**

通过检测指定的宏名**是否定义**来参加编译的程序段。两者都可以与**#else**配对使用，但是**不能和#elif配对使用**。

当**有定义**时，**#ifdef**会执行程序，而**#ifndef**不会执行。

要区别**#if和#ifdef，当#if后面判断的宏定义为0时，表达式不执行**

1. **defined预处理运算符**

格式为：

**defined 宏名**

宏名若**已被定义**，此表达式值为**真**。

可与**if**或**elif**连在一起起到**检测定义**的作用。如**#if defined宏名**即为**#ifdef**

**④行号重置和文件重命名(#line)**

#line指令可以**重置当前所在的代码的行号**。如#line 1000就可以让当前行号变为1000

或#line “cool.c”可以让**文件名**改为cool.

**⑤报错预处理(#error)**

#error Not C11可以让屏幕上出现error:#error Not C11.

第八章 数组

**①数组概念**

由一组同类型、相关联的数据组成，组成数组的数据称为**数组元素**。

为了表示数组中的各个数据，通过**下标**来表示某数组元素在**数组中的位置**。通过**数组名**和**位置**就可以引用任一元素。

根据数组的组成规则，数组可分为**一维数组**、**二维数组**、**多维数组**。

字符串被定义为**字符数组**。即数组元素为字符型的数组。

**②一维数组**

只包含一个下标的数组。格式为：

**类型说明符 数组名[常量表达式]；**

* 类型说明符即为**数组元素类型**，也可以是已经定义的**构造类型**。
* 数组名为**标识符**，同一作用域内，数组之间不可以同名，也不可以与简单变量同名。
* 常量表达式中的值代表数组有几个元素，即**数组长度**。
* C99中允许**使用变量**作为数组的长度，这样的数组称为**变长数组**。

变长数组必须是**自动存储类别**，不能在声明时**初始化**，也不能加**static或extern**

* 有几个**中括号**，代表该数组为**几维**数组。
* 数组若未初始化，则其中元素为**任意值**。即使是部分初始化数组也可以避免这样的情况

1. **一维数组的引用**

数组要**先定义后引用**。定义之后，数组各个元素在系统分配的**连续内存**中依次存放。

数组元素的序号从0开始，最后一个元素序号为**数组长度减1**。引用数组格式为：

**数组名[下标]**

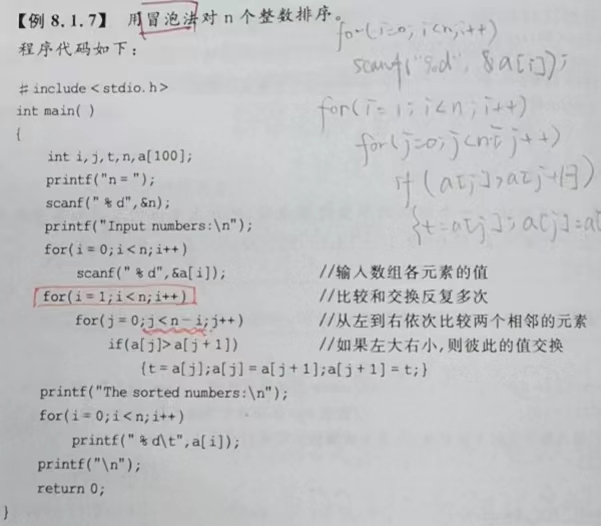
* 下标是一个整型表达式，代表**序号**。如下标值非整型，则自动转换为整型。
* C语言中**不允许引用整个数组**，只能引用单个数组元素。因此数组引用**常与循环结构结合在一起，以便依次输出数据。**
* C语言不会对数组下标进行越界检查。若序号**超出下标上界**，可能会破坏别的变量。必须注意引用最后一个元素时，它的数值**比数值长度少一**。
* 下标引用实际上并**不局限于数组**。下标引用本质上就是指针的间接访问，即：

array[subscipt]=\*(array+subscript)

此处的下标可以理解为偏移量，自然也可以是负值。

1. **输入数组元素**

通过引用数组，可以给数组元素如变量一般赋值。格式即**数组引用加赋值**。

1. **输出数组元素**

同样通过引用数组，实现输出数组元素。

1. **一维数组的初始化**

初始化数值时，数组元素的初值用**花括号**括起来，常量之间用逗号分割，形成常量列表，一一对应为数组元素赋值。

当初始值的个数比数组元素少时，无初值的元素自动以0为初值。初始值的个数**不能多于**数组元素个数。

当确定**对全部数组元素赋初始值**时，数组长度可以不指定，但是**中括号不能省**。

如果**不对数组初始化**，在**编译阶段**数组会得到**0/0.0/’\0’这三种初值**。执行阶段数组元素**乱赋**。

初始化和赋值。**赋值只能一个一个赋**。

* **指定初始化器(C99)**

可以**初始化指定的数组元素**。格式为：

**类型 数组名[元素个数]={[指定元素下标]=值}**

当后面的值**不止一个**，用逗号分隔时；剩下的值会被用于**初始化指定元素后面的元素**

1. **冒泡排序**

排序指将**一组随意存储的数据**重新安排存储位置，使这些数据按**从小到大的顺序(升序)**排列或**从大到小(降序)**排列。

冒泡排序是反复比较相邻的两个元素，不满足排序要求则**交换其位置**。相邻元素的比较需要反复多次。

**③多维数组**

包含**多个下标的数组**。

对于多维数组，可以将其想象为数组的数组。如果一维数组是**一行数据**，那么二维就是**一张表**，三维就是**一叠表**。从指针的角度考虑，那么多维数组可以看成是**指针的数组**

此处以二维数组为例。

1. **二维数组的定义和引用**

定义形式为：

**类型说明符 数组名[常量表达式1][常量表达式2]**

第一个中括号为**第一维**，第二个为**第二维**。数组元素个数为两个表达式的乘积。

一般称第一维为行，第二维为列。

在C语言中存储数组**按行优先**。先存储第0行，再存储第1行，直至最后一行。可以认为，二维数组是由**一维数组构造而成**。二维数组中的一维数组**不能单独引用**，如a[0][1]

中不能调出a[0]。

1. **二维数据的初始化**

二维数组的初始化**按行的顺序**排列，每组加**花括号**。如：

int a[2][4]={{1,2,3,4},{5,6,7,8}}；

二维数组的初始化也**可以只对部分元素赋初值**。

若为全部元素提供初值，内层的花括号可以省去，将所有初始数据**按行为序**写在一个花括号内，且定义数组时**可以省略第一维长度**。

**c. 二维数组和指针的联系**

**对二维数组名解引用**得到的是二维数组中**首个数组的地址**，两次解引用后是**首个数组首元素的值**。可以说，**二维数组名是地址的地址**。

* **指向二维数组的指针**

格式为：

**类型名(\*指针名)[二维数组所内含的数组的元素个数]**

假如括号没有把\*括进去的话，就变成**指针数组**了

**④字符串**

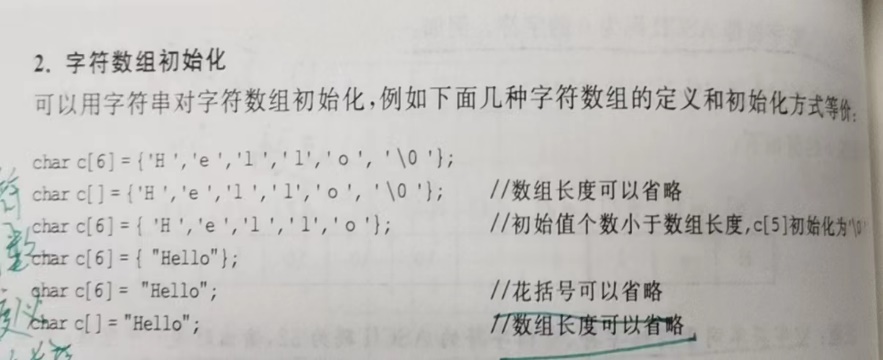
1. **字符型数组**

简称字符数组，其数组元素类型为**字符型**。

初始化字符型数组时，初始值列表应为**字符常量表**。

定义字符数组也可以用int来定义，不过会造成空间浪费。

空字符与空格字符不同，空字符没有任何输出，但是空格字符可以输出空格。

1. **字符串**

字符串在c语言中被当做**字符数组**来处理。属于**静态存储**类别，字符串只会被**存储一次**并**一直存在**。

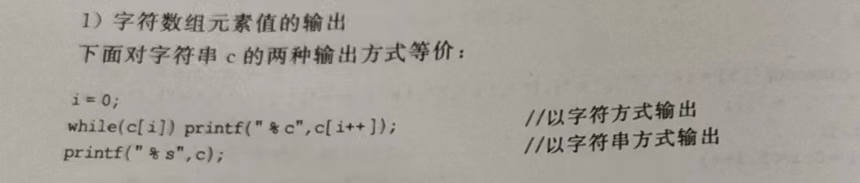
可用字符串对字符数组进行初始化。

存储字符串时系统会自动加一个**结束符**。故**存储字符串的数组长度比字符串本身长度多1**。若为字符串的字符数组，最后一个数组元素必须为**\0**

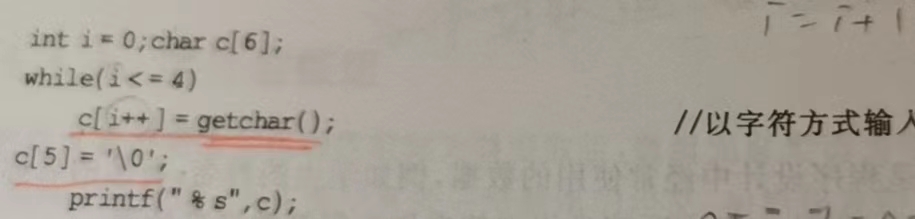
若字符串字面量之间**没有间隔**或用**空白**

**字符间隔**，这些字符串将会被视为**串连**

**的字符串字面量。**

1. **字符数组的输入和输出**

除了用%c逐个输入或输出外，还可以用**%s整个输入或输出**

* 用字符数组输出
* 用字符串输出
* 用字符方式输入
* **用字符串方式输入**

从键盘读取字符串，直到**遇到空格或回车键**(因此输入时字符串不能包含空格字符)。系统会自动在接受到的字符后面加一个结束符。

当数组c接受的**字符串长度时超过数组长度时**，不会产生错误。但是字符串会**占用其他存储空间**，可能引起运行错误。

使用scanf函数一次性输入**多个字符串**时，各个输入串之间**以空格或回车符**分隔。此处的输入项前可以**不加&**，因为**数组名本身就是起始地址**。

1. **字符串处理函数**

除了**printf**和**scanf**外，其他的字符串处理函数都在**<string.h>**内。

* **字符串输出函数puts**

格式为：

**puts(字符串)**

将字符串输出到屏幕并换行，指定的字符串可以是**字符串常量或者字符数组**。

相当于比printf多了个换行。

puts会一直读到**空字符**，**没有空字符**时puts会一直读下去。

* **字符串输入函数gets**

格式为：

**gets(字符串)**

将字符串输入赋给指定的字符数组。与scanf不同的是，这里输入的字符串**只会以回车符**结束，因此**允许包含空格字符**。

但gets输入函数对于**特别长的输入行**，存在**溢出风险；**

* **字符串连接函数strcat**

格式为：

**strcat(字符数组，字符串)**

将字符串连接到字符数组后。函数调用后，**字符串的值不变**，**字符数组**的长度为**原来两串长度之和**，函数返回值为两串之和。

* **字符串复制函数strcpy**

格式为：

**strcpy(字符数组，字符串)**

将字符串**复制到字符数组**。

第一个字符数组**不能为字符串常量**。

函数调用之后，用**字符串的值代替字符数组的值**，字符串的值不变。起到了**字符串赋值运算**的功能。假如字符串没字符数组长，就只覆盖掉对应的元素(\0也会被盖掉)。

注意：不能用a=b来表示字符串的复制运算。

strncpy函数还多了**第三个参数n**，只拷贝**前n个字符**(可能**没能拷贝到空字符**从而使字符串没有终止符)

当**结果字符串超过数组大小**时，就会侵占其他地方的内存空间。

* **字符串比较函数strcmp**

格式为：

**strcmp(字符串1，字符串2)**

比较两个字符串的大小：

* 字符串**相等**时返回函数值**0**
* 字符串**1较小**返回**负值**
* 字符串**1较大**返回**正值**
* 字符串的大小比较
* 优先从左至右**依次比对字符**。
* 遇到字符均相同，则判定为相等。
* 字符串长度大者为大。
* 当字符有不相等时，字符**ascii码大者**所在**字符串大**。
* 两个字符串比较**不能用关系运算符**，只能用strcmp函数
* strncmp函数还多了**第三个参数n**，只比较两个字符串**前n个字符的大小**
* **测字符串长度函数strlen**

格式为：

**strlen(字符串)**

**输出字符串长度**(不是字符数组长度)。如果字符串中间出现/0，函数运算到此为止。不计入\0

strlen函数的结果为一个**无符号数**，无符号数的运算结果也为无符号数，**不会为负值**

* **小写字符串函数strlwr**

格式为：

**strlwr(字符数组)**

将字符数组(不能是字符串)中所有**大写字母转化为小写**。其他不变。

* **大写字符串函数strupr**

格式为：

**strupr(字符数组)**

将字符数组(不能是字符串)中所有**小写字母转化为大写**。其他不变。

* **sprintf函数(被包含在stdio.h中)**

格式为：

**sprintf(字符串名，”格式控制符”，待写入项列表)**

格式上类似于**printf函数**，但是是将这一列字符串输入进某个**字符数组**中存储。

* **查找首次出现字符strchr函数**

格式为：

**strchr(字符串，待查找字符)**

若字符串中**包含查找字符**，函数将返回该字符**第一次出现时元素的指针**。否则返回**空指针**

* **strpbrk函数**

格式为：

**strpbrk(字符串1，字符串2)**

若**字符串1**中包含**字符串2中的任意一个字符**，函数返回指向**字符串1首位置的指针**。否则返回**空字符**。

* **查找最终出现字符strrchr函数**

格式为：

**strrchr(字符串，待查找字符)**

返回字符串中**待查找字符最后一次出现的位置(**允许查询**空字符**)，未找到返回**空指针**。

* **查找字符串strstr函数**

格式为：

**strstr(字符串1，字符串2)**

返回字符串1中**字符串2出现的首位置**，否则返回空指针。

* **将字符串转换数字的函数**

以**一个字符串**为参数，输出对应整数值。

* 转换为**整数**atoi
* 转换为**double类**atof、strtod
* 转换为**long型**atol、strtol、strtoul(无符号)

前者只需要一个字符串，而后者还需要**指定结束转换位置的地址，指定进制**。

**(char\* nptr,char \*\* endptr,int base)**

**⑤复合字面量**

相当于**临时可用的数组**。

字面量是**除了符号常量之外的常量**。复合字面量的声明格式为：

**(类型[元素个数]){值}**

复合字面量是**匿名**的，必须在**创建的同时用指针存储他**。例如：

int\*p1;p1=(int[2]){10,20}

使用复合字面量就不需要先创建数组。不过复合字面量**只能在它被定义的块中使用。**

**⑥内存操作**

* **内存复制函数memcpy**

格式为：

**memcpy(待复制位置(地址)，起始复制位置，复制长度(用sizeof求得大小))**

如果起始复制位置和待复制位置有重叠，行为是未定义的。

* **内存移动函数memmove**

与memcpy类似，但是允许源和目标参数重叠

* **初始化函数memset**

格式为：

**memset(初始化地址，初始值，作用长度)**

第九章 自定义类型

c语言中允许用户自定义的数据类型包括**结构体类型**、**共用体类型**、**枚举类型**。其中结构体和共用体类型属于构造类型，枚举类型属于简单类型。

**①结构体类型**

由一组**相关联**的数据元素构造而成，各元素的数据类型**可以相同或不相同**。数组就属于一种结构体类型

定义

格式为：

**struct 结构体名**

**{**

**数据类型 成员项1;**

**数据类型 成员项n;**

**};**

这样的变量其数据类型名为**struct 结构体名。各个成员项各占一行方便代码阅读**

结构体的类型定义只是定义了一种数据类型，规定这类变量的存储分配方式，并未分配实际的内存空间。其中的成员项可以是构造体类型，此时就形成了**嵌套构造体**。

此声明**仅在声明的块内**可以使用，若放在函数外部，则后面的函数都可以使用他。

1. **结构体变量定义和初始化**
2. **结构体变量的定义**

结构体类型反映的是**处理对象的抽象特征**。对具体对象的描述需要定义结构体变量。

格式为：

**struct 结构体类型名 结构体变量名**

结构体变量的定义也可以与结构体类型的定义同时进行。格式为：

**struct 结构体名**

**{**

**数据类型 成员项1;**

**数据类型 成员项n;**

**};结构体变量1,结构体变量n;**

在结构体类型和结构体变量同时定义时，可以**省略结构体名**。

但若省略结构体名，程序其他位置**不能再使用**这种结构体类型定义其他结构体变量。

* **结构体的不完整声明**

当需要两个互相包含彼此的结构体类型时，可以**先声明其中一个**。然后定义，如：

**struct B;**

**struct A{**

**struct B\* p**

**};**

**struct B{**

**struct A\*p**

**};**

1. **结构体变量初始化**

格式为：

**struct 结构体类型 结构体变量={初始化数据表};**

初始化数据表中的各个数据之间用**逗号**隔开，各个数据与定义中的成员项必须**一一对应**。

初始化列表中，内层的花括号可以省略，但**最外层的花括号是必需的**。

对于一个**静态存储期**的结构体，初始化必须用**常量**(预处理时就已加载完毕)；一个**自动存储的结构体**初始化可以不用常量。

* **结构体中存储字符串**

除了以字符数组作为成员项外，还可以通过**字符指针**的方式存储。

为了保证指针指向已知的空间，需要先用**malloc函数分配内存空间**，使用完毕后用**free**

1. **结构体变量引用**
2. **对结构体变量成员项的引用**

格式为：

**结构体变量名.成员名**

.为分量运算符，具有最高优先级。可以将**结构体变量名.成员名**看作一个整体。

**结构体的成员项**也称为**成员变量**，可以像简单变量一样进行各种操作运算。

嵌套结构体成员的引用需要用两次**点运算符**。

1. **对结构体变量的整体引用**

**同类型**的结构体变量可以**相互赋值**。其他情况下结构体变量不允许整体引用。

1. **结构体数组**
2. **结构体数组的定义**

格式为：

**struct 结构体名 结构体数组名[整常量表达式];**

结构体数组名表示该结构体数字的**存储区域首地址**。在内存中，依次存**储各个结构体数组元素**，结构体数组元素中又**依次存储成员项**。

1. **结构体数组的初始化**

格式为：

**struct 结构体类型 结构体数组名[整常量表达式]={初始化数据表};**

各个元素的初始化数据要用**括号括起来**，分别对应其**成员项**。

**d. 返回结构体的函数**

在函数体内定义一个结构体，函数**返回结构体**。然后用这个函数给**另一个结构体赋值**，就可以起到返回结构体的作用。

向函数传递一个结构是低效的，**传递其指针**更为高效。

**②共用体**

将不同用途的数据存放在同一个存储区域，称为**共用体/联合体类型**。各成员项数据类型**可以相同，也可以不同。**

共用体与结构体的**定义与引用**均相似，主要区别在于**数据的存储方式**上。

* 区别

**结构体类型**：系统为结构变量的各个成员项分配自己的存储空间，一个结构体类型的长度为其**各个成员项之和**。

**共用体类型**：系统将**长度最大的成员项的存储空间**作为共用体变量的**存储空间**。三个成员项都放在这个存储空间。但由于共用，同一时间共用体只能**存储一个成员项**的数据。

1. **共用体类型的定义**

格式为：

**union 共用体名**

**{**

**数据类型 成员项1**

**数据类型 成员项n**

**}**

1. **共用体变量的定义**

**union 共用体类型名 共用体变量列表**

同结构体，共用体变量的定义也可以和共用体类型**一起定义**。

1. **共用体变量的初始化**

**共用体类型名 共用体变量 ={初始值}**

只能有**一个初始值**。

**d. 共用体变量的引用**

类似结构体变量。用.即可。用指针访问共用体跟结构体类似。共用体本身不能为函数参数。

事实上，可以将共用体放在结构体内。按不同的需要，共用体可以显示不同的内容。

**③枚举类型**

枚举类型变量只能取确定的几个量之一，属于**简单数据类型**。

1. **枚举类型的定义**

定义枚举类型，就是定义该类型的值集合。格式为：

**enum 枚举类型名{枚举元素表}；**

* 枚举元素也称为枚举常量，是**标识符**。
* 每个枚举元素都有与之对应的整数，**默认情况下第一个为0，第二个为1**……这样的整数称为**枚举元素的序号**。
* 枚举元素的序号可以被**指定**，在枚举元素之后**加一个等号和一个整数**可以实现。
* 枚举元素的序号没有被指定时，默认为**前元素序号加1**.

1. **枚举变量的定义**

格式为：

**enum 枚举类型名 枚举变量名；**

枚举类型和枚举变量的定义也**可以同时进行**，同时进行时**枚举类型可以省略**。

1. **枚举变量初始化**

可以用枚举元素或整数对枚举变量进行初始化。但如果使用整数时必须进行**类型转换**。

如：Day enDau=（Day）0；

1. **枚举变量的使用**
2. 枚举值可以被**赋给整型变量。**如int i;i=Mon；但**整型值**不能直接赋给**枚举变量**，要**转换，不同枚举类型**之间的变量不能互相赋值。
3. 枚举变量可以进行**算术运算**，但是得到的值是**整型值**，不能赋给枚举变量
4. 枚举值可以进行**关系运算**，系统以**枚举元素序号大小**作为比较依据
5. 枚举变量**不能直接输入和输出，**直接输出枚举变量只能得到序号

**④typedef语句**

为已定义的数据类型定义别名。格式为：

**typedef 现有的类型名 新的类型名**

在之后的程序中，系统会自动识别新的类型名，等效于原类型名。**类似于#define**。

第十章 指针

**①地址与指针变量**

1. **内存单元地址**

存储器由一批存储单元组成。每个存储单元具有唯一的地址，存储单元的地址是一个**无符号整数**，主存储器的所有存储单元是连续的。

数据在内存中占用存储单元（字节），**第一个字节的地址作为变量的地址**。访问变量的数据需要通过该变量的**内存地址**。

变量地址由系统分配，是一个常量。但**并非整数类型**，**不能进行整数类的运算**。如果要打印地址的话有专门的**%p**格式控制符。指针变量的大小是**8字节。**

数组名也是地址，代表该**数组首元素的地址**。如a==&a[0]。

通过**变量名**直接访问存储单元中的数据称为**直接访问**。

通过**指针变量的值得到某变量的地址**，再通过该地址取得变量的值称为**间接访问**。

事实上，程序都是**通过地址访问变量**的。通过变量名访问变量实际是通过变量名间接访问变量。

1. **指针**

存储某个变量地址的变量称为**指针变量**，属于简单变量。指针变量指向的是另一个变量。

1. **指针变量的定义和初始化**

格式为：

**基类型\*指针变量名=&被指向的变量名**

基类型是已经被定义的一种变量类型。

\*表示定义的是**一个指向基类型**的指针变量。连续声明**多个基类型相同的指针**时，指针变量名前面**都要加\***。

指针变量只能存储基类型变量的地址。

没有**被初始化**的指针变量被用来操作存储值时**会访问非法空间**，使用极为危险。因此指针必须被初始化后**指向某一个合法的空间**再使用，**并非是通过解引用**对他指向的值初始化。

不知道指针将会指向什么空间，就先把它指向内存0，即**空指针**。

1. **指针的运算**
2. **指针专用运算符**

* **取地址运算符&**

&的运算对象只能是**变量名**、**数组元素**、**结构体成员**，**不能是表达式**。运算结果即为对象的地址。

* **解引用运算符\***

也称为**间接访问运算符**。\*的运算对象只能是**指针变量**，其运算结果**得到运算对象所指变量**。如p1=&a，\*p1=a。

指针在声明时使用空格，在解引用变量时省略空格

普通变量把**值**作为**基本量**，把**地址**作为通过**&运算符**获得的**派生量**

指针变量把**地址**作为**基本量**，把**值**作为通过**\*运算符获得的派生量**

使用**地址和间接运算符**可以**间接完成语句**的功能，因此叫做**间接运算符**

1. **赋值运算**

**基类型相同的指针**才能进行赋值运算。进行赋值之间的两个指针，必须**指向同一个类型**，或者是**同一个类型的数组**。**二维数组和一维数组的指针之间也不能相互赋值。**

1. **算术运算**

* **指针和整数进行加减运算**

只有当指针**指向数组**时，这种运算才有意义。指针加1在数组中指的是**指向下一个元素**。

当为**加法**时，**整数和指针可以互换**；当为**减法**时，指针必须为**被减数**。

* **两个基类型相同的指针相减**

两个指针相减的结果表示两个指针所指**数组元素下标值的差**。

**4. 关系运算**

指针的关系运算要求两个指针**都指向相同类型的对象**。

判断**是否相等**的关系运算用于确定它们**是否指向同一对象**。

判断**大小**的关系运算用于确定它们所指数据对象存储位置的**前后关系**。这样的比较只有对**指向数组元素的指针**才有意义，指向前面元素的指针变量**小于**指向后面元素的指针变量。

**算术运算和关系运算合法**的前提均为指针都指向**同一个数组**。

**②指针与函数**

1. **指针变量作为函数参数**

当使用指针作为函数参数时，主调函数将 **变量的地址**传递给被调函数。

在没有返回值得函数中，当参数为**其他类型的变量**时，对形参的修改由于没有返回值，**不影响实参**的值；但参数为**指针变量**时，**形参和实参共用存储空间**，修改形参即可修改实参。

如要通过函数调用得到n个要改变的值，可以：

* 在主调函数中设**n个变量**，用n个指针变量指向它们。
* 设计一个函数，有**n个指针形参**。在这个函数中改变**n个形参的值**
* 在主调函数中调用这个函数，在调用时将这**n个指针变量**作实参，将它们的值传给该函数的形参。
* 在执行该函数的过程中，通过**形参指针变量**，改变它们所指向的**n个变量**的值。

**b. 函数的返回值为指针**

函数调用后**返回一个地址**

格式为：

**类型名\*函数名（参数表）**

**c.** **指向函数的指针**

事实上，**函数名**指定的就是**函数的入口地址**。也可以用**一个指针变量指向函数**。

格式为：

**数据类型（\*指针变量名）（函数的参数表【不需要形参名】）**

**如int (\*p)(int,int);p=max;**

通过函数指针变量，可以**将函数作为另一个函数的参数**。使得**函数参数能够传递代码**。

**③指针与数组**

1. **一维数组与指针**
2. **指向数组元素的指针**

各个数组元素都有各自的地址。数组**第一个元素的地址称为数组的首地址**。数组名就代表该**数组的首地址**。如a=&a[0]。

数组的首地址作为**一个常量**，**无法被赋值**。

a代表首地址，a+1代表a[1]的地址，以此类推，\*(a+i)就表示a[i]。

\*p++或\*++p都等价于\*(p++)/\*(++p)，必须要用++(\*p)/++ \*p。

1. **指针表示法和数组表示法**

可以通过**指针表示数组元素**，同样的，也可以用数组的方式表示指针。

相对而言，指针表示法编译后可以**形成效率更高的代码**

1. **数组名作为函数参数**

用**数组名作为参数**实际上就是用**指针**作为函数参数。对形参数组元素的操作就是对实参数组元素的操作。

**b. 字符串与指针**

可以用**字符指针**表示**字符串**，定义一个字符指针后，用**字符指针指向字符串中的首字符**。此时指针变量只会保存**字符串的首地址**。

1. **赋值运算**

对字符数组不能进行赋值运算，因为**字符数组名**是一个**指针常量**，只能对字符指针进行赋值运算。如可以：

char \* a；a=“This is a test.”

1. **输入**

字符串可以**直接输入字符数组**中，但字符串不能直接输入到字符指针中。要输入到**字符指针所指向的存储空间中**。

1. **指针数组**

元素类型为**指针型的数组**。格式为：

**类型名\*数组名[数组长度];**

如:char \*name[ ]={“zhang san”,”li si”,”wang wu”};

那么name[0]就会指向字符串 “zhang san”.

通过**定义指针数组**，**字符串数组**可以被更方便地处理。

1. **指针数组作为main函数的形参**

主函数的形参必须为2个，第一个必须是整型，第二个必须是字符指针数组型。一般上第一个参数写成argc，第二个为argv。在命令提示符中输入“文件名 主函数的其他实参（空格隔开）“即可启动。主函数实际参数个数不限，argc等于实际参数个数。

**④指针与结构体**

1. **指向结构体的指针**
2. **结构体指针的定义**

指向结构体的指针存储的是**结构体变量所占内存区域的首地址**。

1. **用指针引用结构体成员**

例如：

**p1=&tv**.对于tv.price，可以写成**(\*p1).price**

或者也可以写成**p1->price**.表示p1指针所指向的结构体变量的**price成员**。

此处的**圆括号不能省略**。

1. **动态存储分配**

存储空间动态分配方式在**程序运行时为变量分配存储空间**。采用动态空间分配方式的变量**没有变量名**，但用户可以**通过存储空间的地址访问该变量**

头文件**<stdlib.h>** 提供了以下用于分配和释放的存储空间的函数。

1. **malloc函数**

格式为：

**(指针类型\*)malloc(存储空间字节数)**

可以使系统分配一块存储空间。函数返回**存储空间的首地址**。

当不知道具体分配多少存储空间时，可以用**sizeof**，如：p=(int\*)malloc(sizeof(int))

sizeof(int)获得int型变量的字节数。需要注意的是，**sizeof是运算符而并非函数**。

sizeof运算符的返回值可用**%zd**打印，其**返回值类型**为**ize\_t。**称为**底层类型**

对数组用sizeof，是**整个数组的大小**，即元素大小乘以个数

1. **calloc函数**

格式为：

**(指针类型\*)calloc(数组长度，数组元素字节数)**

动态分配方式定义数组，**数组长度可以是变量**。

* 与变长数组的区别

变长数组为自动存储类型，使用结束空间释放；动态分配方式需要free；

变长数组只能在所定义的函数中使用；动态分配方式得到的内存可以以返回指针的方式使用

变长数组对于定义多维数组更为简洁

1. **free函数**

格式为：

**free(指针变量名)**

用动态分配方式定义的变量属于**全程变量**，但**指向动态变量的指针为局部变量**。当函数调用结束后，**局部变量空间被释放，但全程变量空间不释放**。**动态分配方式分配**的内存与静态变量和自动变量等不同，**必须用free函数来释放内存。**

例如，在某个定义的函数中使用动态分配内存但却不回收，每次使用这个函数都会浪费掉一部分内存。

为了避免空间的浪费，要先用**free函数释放其指针变量的空间**。

free函数**释放的是空间**，因此其所用的**指针变量可以不同**，但必须**指向同一个空间**

另外，同一个空间**不能释放两次**

**⑤单链表**

用动态分配方式定义的变量没有变量名，要通过**变量地址引用此变量**。

而这个变量地址又存储在**另一个先定义的指针变量中**。例如：

int \*p

p=(int \* )malloc(sizeof(int));

链表存储的是**一批同类型的相关联的数据**。与数组类似，但不同的是：

* 数组各元素的存储空间**连续、固定**
* 数组长度**固定**
* 链表中元素(结点)个数可以改变
* 链表中结点的**存储空间动态分配**墙上的钟表

  中度可信度描述已自动生成

1. **结构**
2. **头指针**：一般名为**head**。头指针指向链表的**第一个结点(头结点)**，包含**头结点的地址**。头结点不存数据，那么对链表中头结点的数据处理更方便。
3. **结点**：中间的每个结点都包含**数据域**和**指针域**。数据域包含存储**结点本身的值**，指针域存储**下一个结点的地址**。

每个结点的地址都在上一个结点当中，因此对**每个结点的访问**都要经过**头指针**。

1. **最后结点**：也称为**表尾结点/终端结点**。其**指针域为0**，表示不指向任何节点。

各个结点逻辑上相关联，但**地址不一定连续**。

各个结点存储空间**在需要时向系统动态申请**。

1. **单链表的建立**

**反复执行**以下三个步骤

1. 调用**malloc函数**向系统申请**一个结点的存储空间**。
2. **输入该结点的值**，并将该结点的**指针域置0**
3. 将该结点加入到链表中。若链表为空，该结点为链表的**头结点**，否则就**加到表尾**。

链表中结点一定为**构造体**，在定义这样的构造体时其指针域中的**指针变量允许自己定义自己**

1. **单链表的输出**

执行以下三个步骤

1. 执行**h=head**。反复执行2和3直到**h=0**
2. **输出结点的值**
3. 找到**下一个结点**
4. **单链表的查找**

类似于**单链表的输出**。对数据一个个地扫描并进行处理。

1. **单链表的结点删除**
2. 从表头结点开始，确定要**删除结点的地址p**，以及p的**前一个结点地址q**。
3. 如果p为**表头结点**，删除则**修改表头指针head**，否则**修改q结点的指针域**
4. **回收p结点的空间**
5. **单链表的结点插入**
6. 调用**malloc函数**分配一个**结点空间**，**输入新结点的值**
7. 查找**合适的插入位置**
8. **修改相关结点的指针域**。

**⑥有参数的main函数**

1. **格式**

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

argc,argv是main函数的形参，是程序的**命令行参数**。argc是程序中的**参数个数**，argv为一个**char指针数组，包含命令行中的各个字符串**。

带参数的main函数启动时**自带参数**，使用时将程序**拖进命令提示行**。

传入参数格式：

**文件名 字符串1 字符串2….**

1. **命令行的一般形式**

**命令名(可执行文件名) 参数1 参数2…**

其中，**命令名**也算一个参数。如flie1 china beijing就有三个参数

命令行的各个参数均应为**字符串。**

**⑦void指针**

1. **概念**

是**没有确定类型的指针**。任何类型的指针**都可以向它赋值**，但它在给其他变量赋值前**要类型转换**。

1. **注意事项**
2. void函数**不能直接进行指针的算术操作**，因为不知道其类型的大小。
3. 当函数参数可以**传入任意类型的指针**时，应该将**形参设定为void**\*

第十一章 文件

**①文件概述**

在C语言中，文件被视为一连串的能够被单独读取的字节。

**a. 文本文件**

也称为**ascii文件**。将内存中的数据转化为字符后以字符的**ascii码**存储到文件中。

**b. 二进制文件**

将内存中的数据**原样存入文件**中。运行效率高于文本文件。但是人不能读。

**c. 结构文件**

由记录组成。各记录具有相同且确定的数据类型。

C语言**并不支持结构文件**。

**②文件的打开和关闭**

1. **文件的打开**

系统自动设置了一个用于保存文件相关信息的**结构体变量file**。

通过定义**一个指向file类型的指针(文件指针)**，就可以调用文件指针指向的文件。

打开文件过程：

1. 在外部设备中**找到或创建指定的文件**
2. 在内存中**建立缓冲区**
3. 建立**文件变量**
4. 确定文件的**使用方式**
5. 用**fopen函数打开文件**

格式为：

**fopen(“文件名”，“文件使用方式”)**

**文件名必须包含文件扩展名**，可以包含文件路径(fopen中的文件名由于在字符串中，斜杠要打双斜杠)。此函数返回**文件变量的指针**，文件打开失败则**返回0**

**多个文件**在编译中可能以**一个文件**的形式出现一起编译。

**b. 文件使用方式**

* 以**r或rb**方式打开文件，文件为**只读模式**。若**文件不存在，则会出错**。文本文件用r打开，二进制文件用rb打开。
* 以**w或wb**方式打开文件，只能向**文件写入数据**，**清除原文件内容**，保存新写入的数据。若文件不存在，则**创建一个文件**。
* 以**a**方式打开文件，可以**向文件添加内容**。原文件**不存在则会报错**。
* 以**r+或w+**打开文件，功能相同，都可以**读取或写入数据**。
* 以**a+**打开文件，可以**读取或追加数据**。

由于w方式打开文件，文件位置指示器(光标)指向第一个字符;a方式指向文件末尾。因此w会全部清空，a可以追加。

**c. 文件的关闭**

若文件**以写或读写方式**打开。

1. 缓冲区未存入文件的数据存储到文件中，文件末尾加入**文件结束符-1**。
2. 解除文件指针变量与文件的关联。
3. 释放**文件缓冲区**。

c语言中**stdio.h头文件**中宏定义**EOF为-1**。

关闭文件用**fclose函数**，格式为：

**fclose(文件指针变量名)**

**如成功执行，返回值为0；否则为-1；**

**③文件的读写**

1. **文本文件的读写**

标准**输入文件(stdin)**和标准**输入文件(stuout)**都是文本文件。

一般文本文件有其对应的读写函数。

1. **字符输出函数putc或fputc**

将一个字符写到文件中。输出成功函数**返回值为写入数据的值**，否则为**-1**

格式为：

**fputc(待写入字符，文件指针变量名)**

1. **字符输入函数getc或fgetc**

从文件中读取一个字符。输入成功函数**返回值为输入数据的值**，否则为**-1**

格式为：

**fgetc(文件指针变量名)**

1. **字符串输出函数fputs**

格式为：

**fputs(写入字符串的变量名，文件指针变量名)**

**文件指针变量名**若写为**stdout，**则会**打印在显示器上。**fputs**不会添加换行符。**

**成功输入返回0，否则-1**

1. **字符串输入函数fgets**

格式为：

**fgets(字符串，读取字符个数，文件指针变量名)**

**实际读取字符少一个，会读一个结束符。**

**当文件指针变量名写为stdin时，将会读入从键盘输入的数据**

应注意**puts与gets配对使用；fputs和fgets配对使用**

**返回值即字符串地址，否则为NULL**

1. **格式化输出函数fprintf**

类似于printf。可对一般文件输出。

格式为：

**fprintf(文件指针变量名，格式控制符(要加\t便于文件使用)，变量)**

1. **格式化输入函数fscanf**

类似

1. **二进制文件的读写**
2. **写数据块函数fwrite**

格式为：

**fwrite(数据块的首地址，写入单位字节大小，写入单位数量，文件指针变量名)**

1. **读数据块函数fread**

**fread(待读入数据块首地址，读入单位的大小，读入单位数量，文件指针变量名)**

**c. 文件的定位**

1. **复位函数rewind**

使文件位置指示器**重新指向文件的第一个字符**

格式为：

**rewind(指针变量名)**

1. **定位函数fseek**

格式为：

**fseek(文件指针变量名，偏移量，起点)**

其中起点有0.1.2三个值。

* 取0时，以**开头**为基准偏移。
* 取1时，以**当前位置**为基准偏移。
* 取2时，以**末尾**为基准偏移。

stdio.h宏定义SEEK\_SET为0

SEEK\_CUR为1

SEEK\_END为2

偏移量为**正时向后偏**，**为负时向前**。

新位置**落在开头之前，自动跳到开头**;落到**末尾之后**，可能导致难以预料的错误。

一般**fseek用于二进制文件**。文本文件数据长度难以确定，易混乱。

1. **显示位置指针函数ftell**

返回位置指针**相对于文件开头的偏移量**。

格式为：

**ftell(文件指针变量名)**

1. **文件结束检测函数feof**

当文件处于文件结束位置时，返回1，否则0

格式为：

**feof(fp)**

1. **判断出错函数ferror**

调用输入输出函数出现错误时，函数返回**非零值**，无误**返回0**.

在执行fopen时，ferror初值为0.

格式为：

**ferror(文件指针变量名)**

**④重定向和文件**

默认情况下，C使用**标准I/O**包查找标准输入作为**输入源**，这称为stdin流。可以通过**重定向**，使文件作为程序的输入源或输出源。

1. **重定向输入**

用**<**运算符。格式为：

**./程序名<文件名**

1. **重定向输出**

用**>**运算符。格式为：

**./程序名>文件名**

没有指定文件时会**创建**一个文件。

1. **组合重定向**

* **/程序名<文件名1>文件名2**

文件1输入程序1和文件2中

* **/程序名>文件名1<文件名2**

程序输出到文件1，文件2输入文件1中。

* 在一条命令中，输入文件名和输出文件名**不能相同**。
* 重定向运算符必须连接**一个程序和一个数据文件**，不能是两个程序或者是两个文件。
* 重定向运算符不能读取**多个文件的输入**，也不能把输出定向到多个文件。
* windows环境还有**>>**，可以**将数据添加到文件的末尾。**
* 而|运算符可以将**一个文件的输出连接到另一个文件的输入**。

第十二章 位操作

**①二进制数、位、字节**

1. **有符号整数的表示**
2. **二进制补码**
3. **概念**

用1字节的后7位表示0-127，第8位设置为0时表示正数，1为负数。为0时，补码就是真码。

1. **相反数**

求补码数所代表的相反数，要对每一位进行反转，末尾加1，就可以得到相反数的原码，从而求得其真值。

1. **二进制反码**

正数的反码即其补码。

对于每个数，求它的相反数只需把每一位都反转。反码可以表示-127~+127之间的数。

1. **二进制浮点数的表示**
2. **二进制小数**

二进制小数点后的每一位都是用2的次幂作分母，如.101就是0.5+0+0.125=0.625.因此，二进制小数不能精确的表示某些十进制小数。

1. **浮点数表示法**

一个浮点数的表示是存储二进制小数和指数同时进行的。

**②按位运算符**

1. **按位取反符(~)**

对每一位都取反码。单目运算符。

1. **按位与(&)**

二元运算符。对两个数每一位逐次比较生成一个新值，如果这两个数的同一位都为1，新值相应的位上记1，否则记0.如(10010011)&(00111101)=(00010001)

* **掩码**

通过按位与，可以利用一个掩码将另一个数的**相应位置盖住**。例如：

(00000010)(掩码)&(11001111)=(00000010)

被遮盖的码，对应的位会变成0，因此可以说掩码中的0是不透明，1是透明

通过掩码，可以检查某一位的值。

* **清空位**

与掩码类似。将需要清空的位，清空码改成0.

1. **按位或(|)**

类似按位与

* **打开位**

通过按位或，可以利用一个控制码将另一个数**相应的位置变成1.**

1. **按位异或(^)**

对两个数逐次比较，若两个数的**该位不一致，则记为1.**

**③移位运算符**

左移(<<)/右移(>>)

将每一位向左/右移动指定的位数，空出来的部分**用0补齐**。位数最好不要为负。

如(10001010)<<2=(00101000)

* **算术移位与逻辑移位**

两者的区别仅在**右移**时有所体现。

**算术移位**：原值的**符号位如为1**，则右移时移入的位**均为1**；否则**为0**

**逻辑移位**：移入的位**均为0**

有符号值的右移操作究竟使用**逻辑移位**还是**算术移位取决于编译器**。因此，使用有符号数的右移操作的程序是**不可移植**的。

移位运算符相当于十进制通过**移动小数点的办法进行移位**。