目　录

第1章　公共基础知识

1.1　数据结构与算法

1.1.1　算法

1. 算法的基本概念

(1)概念:算法是指一系列解决问题的清晰指令。

(2)4个基本特征:可行性、确定性、有穷性、拥有足够的情报。

(3)两种基本要素:对数据对象的运算和操作、算法的控制结构(运算和操作时间的顺序)。

(4)设计的基本方法:列举法、归纳法、递推法、递归法、减半递推技术和回溯法。

2. 算法的复杂度

(1)算法的时间复杂度:执行算法所需要的计算工作量。

(2)算法的空间复杂度:执行算法所需的内存空间。

1.1.2　数据结构的基本概念

数据结构指相互有关联的数据元素的集合,即数据的组织形式。其中逻辑结构反映数据元素之间逻辑关系;存储结构为数据的逻辑结构在计算机存储空间中的存放形式,有顺序存储、链式存储、索引存储和散列存储4种方式。

数据结构按各元素之间前后件关系的复杂度可划分为:

(1)线性结构:有且只有一个根节点,且每个节点最多有一个直接前驱和一个直接后继的非空数据结构。

(2)非线性结构:不满足线性结构的数据结构。

1.1.3　线性表及其顺序存储结构

1. 线性表的基本概念

线性结构又称线性表,线性表是最简单也是最常用的一种数据结构。

2. 线性表的顺序存储结构

- 元素所占的存储空间必须连续。
- 元素在存储空间的位置是按逻辑顺序存放的。

3. 线性表的插入运算

在第i个元素之前插入一个新元素的步骤如下:

步骤一:把原来第n个节点至第i个节点依次往后移一个元素位置。

步骤二:把新节点放在第i个位置上。

步骤三:修正线性表的节点个数。

在最坏情况下,即插入元素在第一个位置,线性表中所有元素均需要移动。

4. 线性表的删除运算

删除第i个位置的元素的步骤如下:

步骤一:把第i个元素之后不包括第i个元素的n-i个元素依次前移一个位置;

步骤二:修正线性表的结点个数。

1.1.4　栈和队列

1. 栈及其基本运算

(1)基本概念:栈是一种特殊的线性表,其插入运算与

删除运算都只在线性表的一端进行,也被称为“先进后出”表或“后进先出”表。

- 栈顶:允许插入与删除的一端。
- 栈底:栈顶的另一端。
- 空栈:栈中没有元素的栈。

(2)特点。

- 栈顶元素是最后被插入和最早被删除的元素。
- 栈底元素是最早被插入和最后被删除的元素。
- 栈有记忆作用。
- 在顺序存储结构下,栈的插入和删除运算不需移动表中其他数据元素。
- 栈顶指针 top 动态反映了栈中元素的变化情况

(3)顺序存储和运算:入栈运算、退栈运算和读栈顶运算。

2. 队列及其基本运算

(1)基本概念:队列是指允许在一端进行插入,在另一端进行删除的线性表,又称“先进先出”的线性表。

- 队尾:允许插入的一端,用尾指针指向队尾元素。
- 排头:允许删除的一端,用头指针指向头元素的前一位置。

(2)循环队列及其运算。

所谓循环队列,就是将队列存储空间的最后一个位置绕到第一个位置,形成逻辑上的环状空间。

入队运算是指在循环队列的队尾加入一个新元素。当循环队列非空(s=1)且队尾指针等于队头指针时,说明循环队列已满,不能进行入队运算,这种情况称为“上溢”。

退队运算是指在循环队列的队头位置退出一个元素

并赋给指定的变量。首先将队头指针进一,然后将排头指针指向的元素赋给指定的变量。当循环队列为空(s=0)时,不能进行退队运算,这种情况称为“下溢”。

1.1.5　线性链表

在定义的链表中,若只含有一个指针域来存放下一个元素地址,称这样的链表为单链表或线性链表。

在链式存储方式中,要求每个结点由两部分组成:一部分用于存放数据元素值,称为数据域;另一部分用于存放指针,称为指针域。其中指针用于指向该结点的前一个或后一个结点(即前件或后件)。

1.1.6　树和二叉树

1. 树的基本概念

树是简单的非线性结构,树中有且仅有一个没有前驱的节点称为“根”,其余节点分成m个互不相交的有限集合T1,T2,…,T}rm m,每个集合又是一棵树,称T1,T2,…,T}rm m为根结点的子树。

- 父节点:每一个节点只有一个前件,无前件的节点只有一个,称为树的根结点(简称树的根)。
- 子节点:每一个节点可以后多个后件,无后件的节点称为叶子节点。
- 树的度:所有节点最大的度。
- 树的深度:树的最大层次。

2. 二叉树的定义及其基本性质

(1)二叉树的定义:二叉树是一种非线性结构,是有限的节点集合,该集合为空(空二叉树)或由一个根节点及两棵互不相交的左右二叉子树组成。可分为满二叉树和完

全二叉树,其中满二叉树一定是完全二叉树,但完全二叉树不一定是满二叉树。二叉树具有如下两个特点:

- 二叉树可为空,空的二叉树无节点,非空二叉树有且只有一个根结点;
- 每个节点最多可有两棵子树,称为左子树和右子树。

(2)二叉树的基本性质。

性质 1:在二叉树的第 k 层上至多有 2^{k-1} 个结点($k \geqslant 1$)。

性质 2:深度为 m 的二叉树至多有 2^{m-1} 个结点。

性质 3:对任何一棵二叉树,度为 0 的结点(即叶子结点)总是比度为 2 的结点多一个。

性质 4:具有 n 个结点的完全二叉树的深度至少为 $[\log_2 n]+1$,其中 $[\log_2 n]$ 表示 $\log_2 n$ 的整数部分。

3. 满二叉树与完全二叉树

(1)满二叉树:满二叉树是指这样的一种二叉树:除最后一层外,每一层上的所有结点都有两个子结点。满二叉树在其第 i 层上有 2^{i-1} 个结点。

从上面满二叉树定义可知,二叉树的每一层上的结点数必须都达到最大,否则就不是满二叉树。深度为 m 的满二叉树有 2^m-1 个结点。

(2)完全二叉树:完全二叉树是指这样的二叉树:除最后一层外,每一层上的结点数均达到最大值;在最后一层上只缺少右边的若干结点。

如果一棵具有 n 个结点的深度为 k 的二叉树,它的每一个结点都与深度为 k 的满二叉树中编号为 1 ~ n 的结点一一对应。

3. 二叉树的存储结构

二叉树通常采用链式存储结构,存储节点由数据域和指针域(左指针域和右指针域)组成。二叉树的链式存储结构也称二叉链表,对满二叉树和完全二叉树可按层次进行顺序存储。

4. 二叉树的遍历

二叉树的遍历是指不重复地访问二叉树中所有节点,主要指非空二叉树,对于空二叉树则结束返回。二叉树的遍历包括前序遍历、中序遍历和后序遍历。

(1)前序遍历。

前序遍历是指在访问根结点、遍历左子树与遍历右子树这三者中,首先访问根结点,然后遍历左子树,最后遍历右子树;并且,在遍历左右子树时,仍然先访问根结点,然后遍历左子树,最后遍历右子树。前序遍历描述为:若二叉树为空,则执行空操作;否则①访问根结点;②前序遍历左子树;③前序遍历右子树。

(2)中序遍历。

中序遍历是指在访问根结点、遍历左子树与遍历右子树这三者中,首先遍历左子树,然后访问根结点,最后遍历右子树;并且,在遍历左、右子树时,仍然先遍历左子树,然后访问根结点,最后遍历右子树。中序遍历描述为:若二叉树为空,则执行空操作;否则①中序遍历左子树;②访问根结点;③中序遍历右子树。

(3)后序遍历。

后序遍历是指在访问根结点、遍历左子树与遍历右子树这三者中,首先遍历左子树,然后遍历右子树,最后访问根结点,并且,在遍历左、右子树时,仍然先遍历左子树,然

后遍历右子树,最后访问根结点。后序遍历描述为:若二叉树为空,则执行空操作;否则①后序遍历左子树;②后序遍历右子树;③访问根结点。

1.1.7　查找技术

(1)顺序查找:在线性表中查找指定的元素。

最坏情况下,最后一个元素才是要找的元素,则需要与线性表中所有元素比较,比较次数为 n。

(2)二分查找:二分查找也称折半查找,它是一种高效率的查找方法。但二分查找有条件限制,它要求表必须用顺序存储结构,且表中元素必须按关键字有序(升序或降序均可)排列。对长度为 n 的有序线性表,在最坏情况下,二分查找法只需比较 $\log_2 n$ 次。

1.1.8　排序技术

(1)交换类排序法。

- 冒泡排序:通过对待排序序列从后向前或从前向后,依次比较相邻元素的排序码,若发现逆序则交换,使较大的元素逐渐从前部移向后部或较小的元素逐渐从后部移向前部,直到所有元素有序为止。

在最坏情况下,对长度为 n 的线性表排序,冒泡排序需要比较的次数为 $n(n-1)/2$。

- 快速排序:是迄今为止所有内排序算法中速度最快的一种。它的基本思想是:任取待排序序列中的某个元素作为基准(一般取第一个元素),通过一趟排序,将待排元素分为左右两个子序列,左子序列元素的排序码均小于或等于基准元素的排序码,右子序列的排序码则大于基准元素的排序码,然后分别对两个子序列继续进行排序,直至

整个序列有序。最坏情况下,即每次划分,只得到一个序列,时间效率为O(n_2)。

(2)插入类排序法。

● 简单插入排序法:把n个待排序的元素看成为一个有序表和一个无序表,开始时有序表中只包含一个元素,无序表中包含有n-1个元素,排序过程中每次从无序表中取出第一个元素,把它的排序码依次与有序表元素的排序码进行比较,将它插入到有序表中的适当位置,使之成为新的有序表。在最坏情况下,即初始排序序列是逆序的情况下,比较次数为n(n-1)/2,移动次数为n(n-1)/2。

● 希尔排序法:先将整个待排元素序列分割成若干个子序列(由相隔某个“增量”的元素组成的)分别进行直接插入排序,待整个序列中的元素基本有序(增量足够小)时,再对全体元素进行一次直接插入排序。

(3)选择类排序法。

● 简单选择排序法:扫描整个线性表,从中选出最小的元素,将它交换到表的最前面;然后对剩下的子表采用同样的方法,直到子表空为止。最坏情况下需要比较n(n-1)/2次。

● 堆排序的方法:首先将一个无序序列建成堆;然后将堆顶元素(序列中的最大项)与堆中最后一个元素交换(最大项应该在序列的最后)。不考虑已经换到最后的那个元素,只考虑前n-1个元素构成的子序列,将该子序列调整为堆。反复做步骤②,直到剩下的子序列空为止。在最坏情况下,堆排序法需要比较的次数为O($n\log_2 n$)。

1.2　程序设计基础

1.2.1　程序设计方法与风格

(1)设计方法:指设计、编制、调试程序的方法和过程,主要有结构化程序设计方法、软件工程方法和面向对象方法。

(2)设计风格:良好的设计风格要注重源程序文档化、数据说明方法、语句的结构和输入输出。

1.2.2　结构化程序设计

1. 结构化程序设计的原则

结构化程序设计强调程序设计风格和程序结构的规范化,提倡清晰的结构。

(1)自顶向下:即先考虑总体,后考虑细节;先考虑全局目标,后考虑局部目标。

(2)逐步求精:对复杂问题,应设计一些子目标做过渡,逐步细化。

(3)模块化:把程序要解决的总目标分解为分目标,再进一步分解为具体的小目标,把每个小目标称为一个模块;

(4)限制使用 GOTO 语句。

2. 结构化程序的基本结构与特点

(1)顺序结构:自始至终严格按照程序中语句的先后顺序逐条执行,是最基本、最普遍的结构形式。

(2)选择结构:又称为分支结构,包括简单选择和多分支选择结构。

(3)重复结构:又称为循环结构,根据给定的条件,判

断是否需要重复执行某一相同的或类似的程序段。

结构化程序设计中,应注意事项:

(1)使用程序设计语言中的顺序、选择、循环等有限的控制结构表示程序的控制逻辑。

(2)选用的控制结构只准许有一个入口和一个出口。

(3)程序语言组成容易识别的块,每块只有一个入口和一个出口。

(4)复杂结构应该用嵌套的基本控制结构进行组合嵌套来实现。

(5)语言中所没有的控制结构,应该采用前后一致的方法来模拟。

(6)尽量避免 GOTO 语句的使用。

1.2.3 面向对象的程序设计

面向对象方法的本质是主张从客观世界固有的事物出发来构造系统,强调建立的系统能映射问题域。

- 对象:用来表示客观世界中任何实体,可以是任何有明确边界和意义的东西。
- 类:具有共同属性、共同方法的对象的集合。
- 实例:一个具体对象就是其对应分类的一个实例。
- 消息:实例间传递的信息,它统一了数据流和控制流。
- 继承:使用已有的类定义作为基础建立新类的定义技术。
- 多态性:指对象根据所接受的信息而作出动作,同样的信息被不同的对象接收时有不同行动的现象。

面向对象程序设计的优点:与人类习惯的思维方法一致、稳定性好、可重用性好、易于开发大型软件产品、可维

护性好。

1.3　软件工程基础

1.3.1　软件工程基本概念

1. 软件的定义与特点

(1)定义:软件是指与计算机系统的操作有关的计算机程序、规程、规则,以及可能有的文件、文档和数据。

(2)特点。

- 是逻辑实体,有抽象性。
- 生产没有明显的制作过程。
- 运行使用期间不存在磨损、老化问题。
- 开发、运行对计算机系统有依赖性,受计算机系统的限制,导致了软件移植问题。
- 复杂性较高,成本昂贵。
- 开发涉及诸多社会因素。

2. 软件的分类

软件可分应用软件、系统软件和支撑软件 3 类。

(1)应用软件是特定应用领域内专用的软件。

(2)系统软件居于计算机系统中最靠近硬件的一层,是计算机管理自身资源,提高计算机使用效率并为计算机用户提供各种服务的软件。

(3)支撑软件介于系统软件和应用软件之间,是支援其它软件的开发与维护的软件。

3. 软件危机与软件工程

软件危机指在计算机软件的开发和维护中遇到的一系列严重问题。软件工程是应用于计算机软件的定义、开发和维护的一整套方法、工具、文档、实践标准和工序,包

括软件开发技术和软件工程管理。

4. 软件生命周期

软件产品从提出、实现、使用维护到停止使用的过程称为软件生命周期。

在国家标准中,软件生命周期划分为 8 个阶段①软件定义期:包括问题定义、可行性研究和需求分析 3 个阶段。②软件开发期:包括概要设计、详细设计、实现和测试 4 个阶段。③运行维护期:即运行维护阶段。

5. 软件工程的原则

软件工程的原则包括:抽象、信息隐蔽、模块化、局部化、确定性、一致性、完备性和可验证性。

1.3.2 结构化分析方法

需求分析的任务是发现需求、求精、建模和定义需求的过程,可概括为:需求获取、需求分析、编写需求规格说明书和需求评审。

1. 常用的分析方法

- 结构化分析方法:其实质着眼于数据流,自顶向下,逐层分解,建立系统的处理流程。
- 面向对象分析方法。

2. 结构化分析常用工具

结构化分析常用工具包括数据流图、数字字典(核心方法)、判断树和判断表。

(1)数据流图:即 DFD 图,以图形的方式描绘数据在系统中流动和处理的过程,它只反映系统必须完成的逻辑功能,是一种功能模型。

符号名称作用:

- 箭头代表数据流,沿箭头方向传送数据的通道

- 圆或椭圆代表加工,输入数据经加工变换产生输出
- 双杠代表存储文件,表示处理过程中存放各种数据文件
- 方框代表源和潭,表示系统和环境的接口

(2)数据字典:结构化分析方法的核心。数据字典是对所有与系统相关的数据元素的一个有组织的列表,以及精确的、严格的定义,使得用户和系统分析员对于输入、输出、存储成分和中间计算结果有共同的理解。

(3)判定树:使用判定树进行描述时,应先从问题定义的文字描述中分清判定的条件和判定的结论,根据描述材料中的连接词找出判定条件之间的从属关系、并列关系、选择关系,根据它们构造判定树。

(4)判定表:与判定树相似,当数据流图中的加工要依赖于多个逻辑条件的取值,即完成该加工的一组动作是由于某一组条件取值的组合引发的,使用判定表比较适宜。

3. 软件需求规格说明书

软件需求规格说明书是需求分析阶段的最后成果,是软件开发的重要文档之一。

(1)软件需求规格说明书的作用:①便于用户、开发人员进行理解和交流;②反映出用户问题的结构,可以作为软件开发工作的基础和依据;③作为确认测试和验收的依据。

(2)软件需求规格说明书的内容:①概述;②数据描述;③功能描述;④性能描述;⑤参考文献;⑥附录。

(3)软件需求规格说明书的特点:①正确性;②无歧义性;③完整性;④可验证性;⑤一致性;⑥可理解性;⑦可修改性;⑧可追踪性。

1.3.3 结构化设计方法

1. 软件设计的基本概念和方法

软件设计是一个把软件需求转换为软件表示的过程。

(1)基本原理:抽象、模块化、信息隐藏、模块独立性(度量标准:耦合性和内聚性,高耦合、低内聚)。

(2)基本思想:将软件设计成由相对独立、单一功能的模块组成的结构。

2. 概要设计

(1)4 个任务:设计软件系统结构、数据结构及数据库设计、编写概要设计文档、概要设计文档评审。

(2)面向数据流的设计方法:数据流图的信息分为交换流和事物流,结构形式有交换型和事务型。

3. 详细设计的工具

详细设计的工具包括:

- 图形工具:程序流程图、N - S、PAD、HIPO。
- 表格工具:判定表。
- 语言工具:PDL(伪码)。

1.3.4 软件测试

1. 目的

为了发现错误而执行程序的过程。

2. 准则

- 所有测试应追溯到用户需求。
- 严格执行测试计划,排除测试的随意性。
- 充分注意测试中的群集现象。
- 程序员应避免检查自己的程序。
- 穷举测试不可能。

● 妥善保存设计计划、测试用例、出错统计和最终分析报告。

3. 软件测试技术和方法

软件测试的方法按是否需要执行被测软件的角度，可分为静态测试和动态测试，按功能分为白盒测试和黑盒测试。

(1)白盒测试：根据程序的内部逻辑设计测试用例，主要方法有逻辑覆盖测试、基本路径测试等。

(2)黑盒测试：根据规格说明书的功能来设计测试用例，主要诊断方法有等价划分法、边界值分析法、错误推测法、因果图法等，主要用于软件确认测试。

4. 软件测试的实施

软件测试是保证软件质量的重要手段，软件测试是一个过程，其测试流程是该过程规定的程序，目的是使软件测试工作系统化。

软件测试过程分 4 个步骤，即单元测试、集成测试、验收测试和系统测试。

单元测试是对软件设计的最小单位——模块(程序单元)进行正确性检验测试。

单元测试的目的是发现各模块内部可能存在的各种错误。

单元测试的依据是详细的设计说明书和源程序。

单元测试的技术可以采用静态分析和动态测试。

1.3.5　程序的调试

(1)任务：诊断和改正程序中的错误。

(2)调试方法：强行排错法、回溯法和原因排除法。

1.4 数据库设计基础

1.4.1 数据库系统的基本概念

(1)数据(Data):描述事物的符号记录。

(2)数据库(DataBase):长期存储在计算机内的、有组织的、可共享的数据集合。

(3)数据库管理系统的概念

数据库管理系统(DataBase Management System,DBMS)是数据库的机构,它是一种系统软件,负责数据库中的数据组织、数据操作、数据维护、数据控制及保护和数据服务等。为完成以上6个功能,DBMS提供了相应的数据语言;数据定义语言(负责数据的模式定义与数据的物理存取构建);数据操纵语言(负责数据的操纵);数据控制语言(负责数据完整性、安全性的定义)。数据库管理系统是数据库系统的核心,它位于用户和操作系统之间,从软件分类的角度来说 ,属于系统软件。

(4)数据库技术发展经历了3个阶段。

人工管理阶段→文件系统阶段→数据库系统阶段

(5)数据库系统的特点:集成性、高共享性、低冗余性、数据独立性、数据统一管理与控制等。

(6)数据库系统的内部机构体系:三级模式(概念模式、内模式、外模式)和二级映射(外模式/概念模式的映射、概念模式/内模式的映射)构成了数据库系统内部的抽象结构体系。

1.4.2 数据模型

数据模型是数据特征的抽象,从抽象层次上描述了系

统的静态特征、动态行为和约束条件,描述的内容有数据结构、数据操作和数据约束。有 3 个层次:概念数据模型、逻辑数据模型和物理数据模型。

(1)E – R 模型:提供了表示实体、属性和联系的方法。实体间联系有“一对一”、“一对多”和“多对多”。

E – R 模型用 E – R 图来表示。

(2)层次模型:利用树形结构表示实体及其之间联系,其中节点是实体,树枝是联系,从上到下是一对多关系。

(3)网状模型:用网状结构表示实体及其之间联系,是层次模型的扩展。网络模型以记录型为节点,反映现实中较为复杂的事物联系。

(4)关系模型:采用二维表(由表框架和表的元组组成)来表示,可进行数据查询、增加、删除及修改操作。关系模型允许定义“实体完整性”、“参照完整性”和“用户定义的完整性”三种约束。

- 键(码):二维表中唯一能标识元组的最小属性集。
- 候选键(候选码):二维表中可能有的多个键。
- 主键:被选取的一个使用的键。

1.4.3　关系代数

(1)关系代数的基本运算:投影、选择、笛卡尔积。

(2)关系代数的扩充运算:交、连接与自然连接、除。

1.4.4　数据库设计与管理

1. 数据库设计概述

- 基本思想:过程迭代和逐步求精。
- 方法:面向数据的方法和面向过程的方法。
- 设计过程:需求分析→概念设计→逻辑设计→物理

设计→编码→测试→运行→进一步修改。

2. 数据库设计的需求分析

需求收集和分析是数据库设计的第一阶段,常用结构化分析方法(自顶向下、逐层分解)和面向对象的方法,主要工作有绘制数据流程图、数据分析、功能分析、确定功能处理模块和数据间关系。

数据字典:包括数据项、数据结构、数据流、数据存储和处理过程,是对系统中数据的详尽描述。

3. 数据库的设计

(1)数据库的概念设计:分析数据间内在的语义关联,以建立数据的抽象模型。

(2)数据库的逻辑设计:从 E - R 图向关系模型转换,逻辑模式规范化,关系视图设计可以根据用户需求随时创建。实体转换为元组,属性转换为关系的属性,联系转换为关系。

(3)数据库的物理设计:是数据在物理设备上的存储结构与存取方法,目的是对数据库内部物理结构作出调整并选择合理的存取路径,以提高速度和存储空间。

4. 数据库管理

数据库管理包括数据库的建立、数据库的调整、数据库的重组、数据库的安全性与完整性控制、数据库故障恢复和数据库的监控。

第2章　C语言概述

2.1　C语言基础知识

2.1.1　C语言的构成

(1)源程序由函数构成,每个函数完成相对独立的功能。

(2)每个源程序中必须有且只能有一个主函数,可以放在任何位置,但程序总是从主函数开始执行。

(3)函数体:在函数后面用一对花括号括起来的部分。

(4)每个语句以分号结束,但预处理命令、函数头之后不能加分号。

(5)注释:括在“/*”与“*/”之间,没有空格,允许出现在程序的任何位置。

(6)预处理命令:以“#”开头的语句。

2.1.2　C程序的生成过程

(1)C程序是先由源文件经编译生成目标文件,然后经过连接生成可执行文件。

(2)源程序的扩展名为.c,目标程序的扩展名为.obj,可执行程序的扩展名为.exe 。

2.2　常量、变量和数据类型

2.2.1　标识符

1. 标识符的命名规则

- 只能由字母、数字或下划线组成。
- 第一个字符必须是字母或下划线,不能是数字。
- 区分字母的大小写。

2. 标识符的分类

C 语言的标识符可以分为 3 类。

(1)关键字:C 语言规定的专用的标识符,它们有着固定的含义,不能更改。

(2)预定义标识符:和“关键字”一样也有特定的含义。

(3)用户标识符:由用户根据需要定义的标识符。

- 命名应注意做到“见名知义”。
- 不能与关键字相同。

2.2.2　常量

定义:在程序运行中,其值不能被改变的量。

常量的类型:整型常量、实型常量、字符常量、字符串常量和符号常量。

1. 整型常量

(1)表示形式:十进制整型常量、八进制整型常量和十六进制整型常量。

(2)书写形式。

- 十进制整型常量:基本数字范围为 0 ~ 9。
- 八进制整型常量:以 0 开头,基本数字范围为 0 ~ 7。
- 十六进制整型常量:以 0x 开头,基本数字范围为 0 ~ 15,其中 10 ~ 15 写为 A ~ F 或 a ~ f。

2. 实型常量

(1)表示形式:小数形式和指数形式。

(2)书写形式。

- 十进制小数形式:小数点两边必须有数字。
- 指数形式:e 前必须有数字,e 后必须为整数。

3. 字符常量

一个字符常量代表 ASCII 码字符集里的一个字符,在程序中用单撇号括起来,区分大小写。

特殊的字符常量:即转义字符。其中“\”是转义的意

思,后面跟不同的字符表示不同的意思。

- \n:换行。
- \\:反斜杠字符“\”。
- \ddd:1 ~3 位八进制数所代表的一个 ASCII 字符。
- \xhh:1 ~2 位十六进制数所代表的一个 ASCII 字符。

4. 字符串常量

字符串常量是用双撇号括起来的一个或一串字符。

5. 符号常量(无)

符号常量是由预处理命令“#define”定义的常量,在 C 程序中可用标识符代表一个常量。

2.2.3　变量

定义:值可以改变的量。

- 变量要有变量名,在使用前必须先定义。
- 在内存中占据一定的存储单元,不同类型的变量其存储单元的大小不同。
- 存储单元里存放的是该变量的值。

变量的类型:整型变量、实型变量、字符变量。

1. 整型变量

(1)分类:基本型(int)、短整型(short int 或 short)、长整型(long int 或 long)和无符号型(unsigned int ,unsigned short,unsigned long)。

(2)数值范围。

整型[signed]int,占 16 位;短整型[signed]short[int],占 16 位;长整型[signed]long int,占 32 位;无符号整型 unsigned[int],占 16 位;无符号短整型 unsigned short[int],占 16 位;无符号长整型 unsigned long[int],占 32 位。

2. 实型变量

(1)分类:单精度类型(float)和双精度类型(double)。

(2)定义方法:float a;,double m;。

(3)所占字节:float 型在内存中占 4 个字节(32 位),double 型占 8 个字节(64 位)。单精度实数提供 7 位有效数字,双精度实数提供 15~16 位有效数字。

(4)实型常量:不分 float 型和 double 型,一个实型常量可以赋给一个 float 型或 double 型变量,但变量根据其自身类型截取实型常量中相应的有效数字。

3. 字符变量

(1)作用:用来存放字符常量。

(2)定义:用关键字 char 定义,每个字符变量中只能存放一个字符。

(3)定义形式:char cr1 , cr2;。

(4)赋值:cr1 = 'm',cr2 = 'n';。

(5)存储方法:存储字符对应的 ASCII 码到内存单元中。

- 字符型数据与整型数据之间可以通用,一个字符能用字符的形式输出,也能用整数的形式输出。
- 字符数据进行算术运算,相当于对它们的 ASCII 码进行运算。

2.2.4　类型的自动转换和强制转换

1. 类型的自动转换

(1)当同一表达式中各数据的类型不同时,编译程序会自动把它们转变成同一类型后再进行计算。

(2)转换优先级为:char < int < float < double ,即左边级别"低"的类型向右边转换。

(3)在做赋值运算时,若赋值号左右两边的类型不同,则赋值号右边的类型向左边的类型转换;当右边的类型高于左边的类型时,则在转换时对右边的数据进行截取。

2. 类型的强制转换

(1)表示形式:(类型)(表达式);

第3章　运算符与表达式

3.1　C语言运算符简介

3.1.1　C运算符简介

- 算术运算符：+，-，*，/，%
- 关系运算符：>，>=，==，!=，<，<=
- 位运算符：>>，<<，~，&，|，∧
- 逻辑运算符：!，||，&&
- 条件运算符：?：
- 指针运算符：&，*
- 赋值运算符：=
- 逗号运算符：，
- 字节运算符：sizeof
- 强制运算符：(类型名)(表达式)
- 其他：下标，分量，函数

3.1.2　运算符的结合性和优先级

1. 结合性

所有的单目运算符、条件运算符、赋值运算符及其扩展运算符，结合方向都是从右向左，其余运算符的结合方向是从左向右。

2. 优先级比较

初等运算符 > 单目运算符 > 算术运算符(先乘除后加减) > 关系运算符 > 逻辑运算符(不包括“!”) > 条件运算

符 > 赋值运算符 > 逗号运算符。

初等运算符包括:圆括号( ),下标运算符[ ]和结构体成员运算符 - >。

3.1.3 强制类型转换运算符

1. 可以利用强制类型转换符将一个表达式转换成所需类型。

2. 一般形式:(类型名)(表达式)

3.1.4 逗号运算符和逗号表达式

1. 逗号表达式:用逗号运算符将几个表达式连接起来。

2. 一般形式:表达式 1,表达式 2,…,表达式 n。

3. 求解过程:先求解表达式 1,然后依次求解表达式 2,直到表达式 n 的值。表达式 n 的值就是整个逗号表达式的值。

3.2 算术运算符和算术表达式

3.2.1 基本的算术运算符

1. 分类:+(加法运算符或正值运算符)、-(减法运算符或负值运算符)、*(乘)、/(除)和%(求余)。

其中,“%”运算的两端必须都是整型,其余的运算对象都可以是整型或者实型。

2. 双目运算符两边的数值类型必须一致才能进行运算,如果不一致,系统先进行一致性转换。

转换规则:char - > short - > int - > unsigned - > long - > double - > float。

3. 所有实数的运算都是以双精度方式进行的,若是单

精度数值,则需要在尾数后面补 0 转换为双精度数。

3.2.2　算术表达式和运算符的优先级与结合性

1. 定义:用算术运算符和括号将运算量连接起来的、符合 C 语言语法规则的表达式。

2. 运算对象:函数、常量和变量等。

3. 运算规则。

- 可使用多层圆括号,但括号必须配对。运算时由内向外依次计算各表达式的值。
- 对于不同优先级的运算符,按运算符的优先级由高到低进行运算,若优先级相同,则按结合方向进行运算。
- 若运算符两侧的操作数类型不同,则先利用自动转换或强制类型转换,然后进行运算。

3.2.3　自加、自减运算符

1. 作用:自加运算符“ + + ”使运算变量的值增 1,自减运算符“ - - ”使运算变量的值减 1。

2. 均是单目运算符。运算对象可以是整型或实型变量,但不可以是常量和表达式。

3. 均可作为前缀运算符,也可作为后缀运算符构成一个表达式。

- + + i, - - i :在使用 i 之前,先使 i 的值加 1 或者减 1,再使用此时的表达式的值参加运算。
- i + + ,i - - :在使用 i 之后,使 i 的值加 1 或者减 1,再使用此时的表达式的值参加运算。

4. 结合方向:自右向左。

3.3　赋值运算符和赋值表达式

3.3.1　赋值运算符和赋值表达式

1.“ = ”称做赋值运算符,作用是将一个数值赋给一个变量或将一个变量的值赋给另一个变量,由赋值运算符组成的表达式称为赋值表达式。

2. 一般形式:变量名 = 表达式。

- 赋值运算符的优先级别高于逗号运算符。
- 赋值运算符“ = ”和等于运算符“ = = ”有很大差别。
- 赋值运算符的左侧只能是变量,而不能是常量或者表达式。右侧可以是表达式,包括赋值表达式。
- 规定最左边变量所得到的新值就是整个赋值表达式的值。

3.3.2　复合的赋值运算符

在赋值运算符之前加上其他运算符可以构成复合赋值运算符。例如,+ = 、- = 、* = 、/ = 、% = 等。

- 两个符号之间不可以有空格。
- 复合赋值运算符的优先级与赋值运算符的相同。

3.3.3　赋值运算中的类型转换

如果赋值运算符两侧的类型不一致,在赋值前系统将自动先把右侧表达式求得的数值按赋值号左边变量的类型进行转换(也可以用强制类型转换的方式)。

3.4　位运算

1. C 语言提供 6 种位运算符

- 按位与“&”:若两个相应的二进制位都为 1,则该位的结果为 1,否则为 0。
- 按位或“|”:两个相应的二进制位中只要有一个为 1,则该位的结果为 1,否则为 0。
- 按位异或“^”:若两个二进制位相同,则结果为 0,不同则为 1。
- 按位求反“~”:按位取反,即 0 变 1,1 变 0。
- 左移“<<”:将一个数的二进制位全部左移若干位。
- 右移“>>”:将一个数的二进制位全部右移若干位。

2. 说明

- 位运算中除“~”以外,均为双目运算符,要求两侧各有一个运算量。
- 运算量只能是整型或字符型数据,不能为实型数据。

第4章　基本语句

4.1　单个字符的输入输出

4.1.1　字符输出函数 putchar( )

putchar( ) 函数的作用是向终端输出一个字符。

4.1.2　字符输入函数 getchar( )

- getchar( ) 函数的作用是从终端输入一个字符。
- getchar( ) 函数没有参数,函数值就是从输入设备得到的字符。

4.2　数据格式的输入与输出

4.2.1　格式化输出函数 printf( )

printf( ) 函数是 C 语言提供的标准输出函数,它的作用是向终端(或系统隐含指定的输出设备)按指定格式输出若干个数据。

1. printf( ) 函数的一般形式

printf(格式控制,输出表列);

(1)“格式控制”:用双引号括起来的字符串是“格式控制”字符串,它包括两种信息。

- 格式转换说明,由“%”和格式字符组成。
- 需要原样输出的字符也写在格式控制内。

(2)“输出表列”:需要输出的一些数据,可以是常量、变量或表达式。输出表列中的各输出项用逗号隔开。

2. 格式字符

可在“%”与格式字符之间插入“宽度说明”、左对齐符号“ -”、前导零符号“0”等。

- d 格式符,用来对十进制数进行输入输出。
- o 格式符,以八进制数形式输出整数。
- x 格式符,以十六进制数形式输出整数。
- u 格式符,用来输出 unsigned 型数据,即输出无符号的十进制数。
- c 格式符,用来输出一个字符。
- s 格式符,用来输出一个字符串。
- f 格式符,用来输出实数(包括单、双精度),以小数形式输出,使整数部分全部输出。
- e 格式符,以指数形式输出实数。
- g 格式符,用来输出实数。

3. 使用 printf( ) 函数时的注意事项

- 在格式控制串中,格式说明与输出项从左到右在类型上必须一一对应匹配。
- 在格式控制串中,格式说明与输出项个数要相等。
- 在格式控制串中,可以包含任意的合法字符(包括转义字符),这些字符在输出时将被“原样输出”。
- 如果要输出“%”,则应该在格式控制串中用两个连续的百分号“%%”来表示。

4.2.2　格式化输入函数 scanf( )

1. scanf( ) 函数的一般形式

scanf(格式控制,地址表列);

其中 scanf 是函数名,“格式控制”的含义同 printf( ) 函数,“地址表列”由若干个变量地址组成,既可以是变量的

地址,也可以是字符串的首地址。

2. 格式说明

scanf( ) 函数中的格式说明也是以% 开始,以一个格式字符结束,中间可以加入附加的字符。

- 对 unsigned 型变量的数据,可以用 % d、% o、% x 格式输入。
- 在 scanf( ) 函数中格式字符前可以用一个整数指定输入数据所占宽度,但对于输入实型数则不能指定其小数位的宽度。
- 在格式控制串中,格式说明的个数应该与输入项的个数相等,且要类型匹配。

3. 使用 scanf( ) 函数时的注意事项

- scanf( ) 函数中的输入项只能是地址表达式,而不能是变量名或其他内容。
- 如果在“格式控制”字串中除了格式说明以外还有其他字符,则在输入数据时应输入与这些字符相同的字符。
- 在用 “% c”格式输入字符时,空格字符和转义字符都可作为有效字符输入。
- 在输入数据时,若实际输入数据少于输入项个数,scanf( ) 函数会等待输入,直到满足条件或遇到非法字符才结束;若实际输入数据多于输入项个数,多余的数据将留在缓冲区备用,作为下一次输入操作的数据。
- 在输入数据时,遇到以下情况时认为输入结束:空格、“回车”或“跳格”(“Tab”)键,上述字符统一可称为“间隔符”。

第5章　选择结构

5.1　关系运算符和关系表达式

5.1.1　关系运算符及其优先次序

C语言提供了6种关系运算符:小于(<)、小于等于(<=)、大于等于(>=)、大于(>)、等于(==)、不等于(!=)。

1. 结合性:自左向右。

2. 优先级:

- 前4种关系运算符(<,<=,>=,>)的优先级别相同,后两种(==,!=)优先级相同。
- 前4种优先级高于后两种。
- 关系运算符的优先级低于算术运算符,高于赋值运算符。

5.1.2　关系表达式

1. 定义:由关系运算符连成的表达式。关系运算符的两边可以是C语言中任意合法的表达式。

2. 关系运算符的结果是一个整数值——“0或者1”,用非零值来表示“真”,用零值来表示“假”。

3. 当关系运算符两边值的类型不一致时,系统将自动转化。

5.2　逻辑运算符和逻辑表达式

5.2.1　逻辑运算符及其优先级

C语言提供了3种逻辑运算符:逻辑与(&&)、逻辑或

(‖)、逻辑非(!)。其中“&&”和“‖”是双目运算符,而“!”是单目运算符,要求必须出现在运算对象的左边。

1. 结合性:自左至右。

2. 优先级:“!”>“&&”>“‖”。

“!”>算术运算符>关系运算符>“&&”>“‖”>赋值运算符。

5.2.2　逻辑表达式

1. 逻辑表达式由逻辑运算符和运算对象组成。

2. 参与逻辑运算的对象可以是一个具体的值,还可以是 C 语言中任意合法的表达式。

3. 逻辑表达式的运算结果为 1(真)或者为 0(假)。

- A&&B 运算中,只有 A、B 同为真时才为真。
- A‖B 运算中,只有 A、B 同为假时才为假。
- 关系运算符不能连用,即不能使用 0<x<10,可改写成 0<x&&x<10。

5.3　if 语句和用 if 语句构成的选择结构

5.3.1　if 语句的几种形式

1. if(表达式)语句

- if 是 C 语言的关键字。
- 表达式两侧的括号不可少,并且只能是圆括号。
- 紧跟着的语句,称为 if 子句,如果在 if 子句中需要多个语句,则应该使用大括号({})把一组语句括起来构成复合语句。

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m)语句m
else 语句n

- “语句1”是if子句,“语句2…语句m”是else子句。这些子句在语法上要求是一条语句,当需要执行多条语句时,应该使用花括号({})把这些语句括起来组成复合语句。
- else必须与if配对,共同组成if…else语句。

5.3.2　if语句的嵌套

在if语句中又包含一个或多个if语句结构,称为if语句的嵌套。

5.3.3　条件运算符构成的选择结构

1. 条件运算符:?:。

2. 条件表达式的一般形式:表达式1?表达式2:表达式3。

3. 求解过程:先求表达式1的值,当表达式1的值是非0时,以表达式2的值作为整个条件表达式的值;当表达式1的值是0时,以表达式3的值作为整个条件表达式的值。

4. 优先级:条件运算符高于赋值运算符,但低于逻辑运算符、关系运算符和算术运算符。

5.4　switch 语句

switch 语句是 C 语言提供的多分支选择语句,用来实现多分支选择结构。

一般形式:

```
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
}
```

- switch 后面用花括号括起来的部分是 switch 语句体。
- switch 后面括号内的“表达式”,可以是 C 语言中任意合法表达式,但表达式两侧的括号不能省略。
- case 与其后面的常量表达式合称 case 语句标号,常量表达式的类型必须与 switch 后面的表达式的类型相匹配,且各 case 语句标号的值各不相同,不能重复。
- default 也是关键字,起标号的作用,代表除了以上所有 case 标号之外的那些标号,default 标号可以出现在语句体中任何标号位置上,当然,也可以没有。
- case 语句标号后的语句 1、语句 2 等,可以是一条语句,也可以是若干条,在必要时,case 语句标号后的语句可以省略不写。

第6章　循环结构

6.1　while 循环语句

1. 一般形式:while(表达式) 循环体

- while 是 C 语言的关键字。
- 紧跟其后的表达式可以是 C 语言中任意合法的表达式,该表达式是循环条件,由它来控制循环体是否执行。
- 循环体只能是一条可执行语句,当多项操作需要多次重复做时,可以使用复合语句。

2. 执行过程:

- 第一步:计算紧跟 while 后括号中表达式的值,当表达式的值为非 0 时,则接着执行 while 语句中的内嵌语句;当表达式值为 0 时,则跳过该 while 语句,执行该 while 结构后的其他语句。
- 第二步:执行循环体内嵌语句。
- 第三步:返回去执行步骤(1),直到条件不满足,即表达式的值为 0 时,退出循环,while 结构结束。

3. 特点:先对表达式进行条件判断,后执行语句。

6.2　do…while 语句

1. 一般形式:

```
do{
循环体语句
}while(表达式);
```

- do 是 C 语言的关键字,必须和 while 联合使用,不能独立出现。

- do…while 循环由 do 开始,用 while 结束。
- while 后面的圆括号中的表达式,可以是 C 语言中任意合法的表达式,由它控制循环是否执行,且圆括号不可丢。
- 在语法上,在 do 和 while 之间只能是一条语句,如需要执行多条语句时,可使用复合语句。
- while(表达式)后的分号不可丢。

2. 执行过程:

- 先执行一次指定的循环体语句。
- 执行完后,判别 while 后面的表达式的值,当表达式的值为非零(真)时,程序流程返回,去重新执行循环体语句。
- 如此反复,直到表达式的值等于零为止,此时循环结束。

3. 特点:

先执行循环体一次,然后判断循环条件是否成立。

6.3　for 语句

1. 一般形式为:for(表达式 1;表达式 2;表达式 3)

- 圆括号中通常是 3 个表达式,用于 for 循环的控制。for 语句中的表达式可以部分或者全部省略,但两个“;”是不可省略的。
- 各个表达式之间用“;”隔开,且圆括号不可省略。
- 按照语法规则,循环体只能是一条语句,如需要完成多项操作,须使用复合语句。

2. 执行过程:

①先求表达式 1 的值;②求表达式 2 的值,若其值为真(非 0),则执行 for 语句中指定的内嵌语句,然后执行下面步骤③,若其值为假(0),则退出循环,执行 for 语句以下的其他语句;③求表达式 3 的值;④重复执行步骤②。

6.4　循环的嵌套

1. 定义:在某一个循环体内部又包含了另一个完整的循环结构,称为循环的嵌套。

2. 前面介绍的 3 种类型的循环都可以互相嵌套,循环的嵌套可以多层,但要保证每一层循环在逻辑上必须是完整的。

6.5　几种循环的比较

- while 和 do … while 循环,只在 while 后面指定循环条件,循环体内应包含使循环趋于结束的语句,for 中使循环趋于结束的操作可以包含在“表达式 3”中。
- 由 while 完成的循环,用 for 循环都能完成。

6.6　break 语句和 continue 语句

6.6.1　break 语句

1. 在 break 后面加上分号就可以构成 break 语句,break 语句还可以用于从循环体内跳出,即提前结束循环。

2. 说明:

- break 语句只能出现在循环体内及 switch 语句内,不能用于其他语句。
- 当 break 出现在循环体中的 switch 语句体内时,其作用只是跳出该 switch 语句体。当 break 出现在循环体中,但并不在 switch 语句体内时,则在执行 break 后,跳出本层循环,当然也不再去进行条件判断。

6.6.2　continue 语句

1. 一般形式为:continue;

2. 作用:结束本次循环,即跳过循环体中下面尚未执行的语句,而转去重新判定循环条件是否成立,从而确定下一次循环是否继续执行。

第7章　数　组

7.1　一维数组的定义和引用

7.1.1　一维数组的定义

一维数组是指数组中的每个元素只带有一个下标的数组。定义方式为:类型说明符 数组名[常量表达式];。

7.1.2　一维数组元素的引用

数组元素的引用形式为:数组名[下标表达式]。

- 一个数组元素实质上是一个变量名,代表内存中的一个存储单元,一个数组占据的是一连串连续的存储单元。
- 引用数组元素时,数组的下标可以是整型常量,也可以是整型表达式。
- 数组必须先定义后使用。
- 只能逐个引用数组元素而不能一次引用整个数组。

7.1.3　一维数组的初始化

当数组定义后,系统会为该数组在内存中开辟一串连续的存储单元,但这些存储单元中并没有确定的值。可以在定义数组时为所包含的数组元素赋初值。

如:int a[6] = { 0,1,2,3,4,5 };

- 所赋初值放在一对花括号中,数值类型必须与所说明类型一致。
- 所赋初值之间用逗号隔开,系统将按这些数值的排列顺序,从 a[0]元素开始依次给数组 a 中的元素赋初值。

- 不能跳过前面的元素给后面的元素赋初值,但是允许为前面元素赋值为0。
- 当所赋初值个数少于所定义数组的元素个数时,将自动给后面的其他元素补初值0。
- 可以通过赋初值来定义一维数组的大小,定义数组时的一对方括号中可以不指定数组的大小。

7.2　二维数组的定义和引用

7.2.1　二维数组的定义

1. 在C语言中,二维数组中元素排列的顺序是:按行存放,即在内存中先顺序存放第一行的元素,再存放第二行的元素。二维数组元素的存储总是占用一块连续的内存单元。

2. 一般形式为:

类型说明符 数组名[常量表达式][常量表达式];

7.2.2　二维数组元素的引用

1. 表示形式为:数组名[下标表达式1][下标表达式2]

- 数组的下标可以是整型表达式。
- 数组元素可以出现在表达式中,也可以被赋值。

7.2.3　二维数组的初始化

- 可以在定义二维数组的同时给二维数组的各元素赋初值。
- 全部初值放在一对花括号中,每一行的初值又分别括在一对花括号中,之间用逗号隔开。
- 当某行一对花括号内的初值个数少于该行中元素的个数时,系统将自动地给后面的元素赋初值0。
- 不能跳过每行前面的元素而给后面的元素赋初值。

• 对于二维数组,只可以省略第一个方括号中的常量表达式,但不能省略第二个方括号中的常量表达式。

7.2.4　通过赋初值定义二维数组的大小

对于一维数组,可以在数组定义语句中省略方括号中的常量表达式,通过所赋初值的个数来确定数组的大小;对于二维数组,只可以省略第一个方括号中的常量表达式,而不能省略第二个方括号中的常量表达式。

7.3　字符数组

7.3.1　字符数组的定义

字符数组就是数组中的每个元素都是字符。定义方法同普通数组的定义相同,即逐个对数组元素赋值。

7.3.2　字符数组的初始化及引用

1. 初始化:

对字符数组初始化,可逐个元素地赋值,即把字符逐个赋给数组元素。

• 如果花括号中提供的初值个数(即字符个数)大于数组长度,则编译时会按语法错误处理。

• 如果初值个数小于数组长度,则将这些字符赋给数组中前面那些元素,其余的元素定为空字符('\0')。

2. 引用形式:采用下标引用,即:数组名[下标]。

7.3.3　字符串和字符串结束标志

C 语言中,将字符串作为字符数组来处理。为了测定字符串的实际长度,C 语言规定了一个字符串结束标志,以字符'\0'代表。就是说,在遇到字符'\0'时,表示字符串结束,由它前面的字符组成字符串。

7.3.4　字符数组的输入输出

字符数组的输入输出可以有以下两种方法：

- 用“%c”格式符将字符逐个输入或输出。
- 用“%s”格式符，将整个字符串一次输入或输出。

7.3.5　字符串处理函数

C语言没有提供对字符串进行整体操作的运算符，但在C语言的函数库中提供了一些用来处理字符串的函数。在调用这些函数前，须在程序前面的命令行包含标准头文件“string.h”。

- puts()：调用形式为puts(字符数组)，将一个字符串(以‘\0’结束)输出到终端设备。
- gets()：调用形式为gets(字符数组)，从终端输入一个字符串到字符数组中，并且得到一个函数值。
- strcpy()：调用形式为strcpy(字符数组1，字符数组2)，把字符数组2所指字符串的内容复制到字符数组1所指存储空间中。函数返回字符数组1的值，即目的串的首地址。
- strcat()：调用形式为strcat(字符数组1，字符数组2)，该函数将字符数组2所指字符串的内容连接到字符数组1所指的字符串后面，并自动覆盖字符数组1串末尾的‘\0’。该函数返回字符数组1的地址值。
- strlen()：调用形式为strlen(字符数组)，此函数计算出以字符数组为起始地址的字符串的长度，并作为函数值返回。
- strcmp()：调用形式为strcmp(字符数组1，字符数组2)，该函数用来比较字符数组1和字符数组2所指字符串的大小。若字符数组1>字符数组2，函数值大于0(正数)；若字符数组1=字符数组2，函数值等于0；若字符数组1<字符数组2，函数值小于0(负数)。

第8章　函　数

8.1　函数概述

在C语言中,子程序的作用是由函数完成的。一个C程序可由一个主函数和若干个其他函数构成,并且只能有一个主函数。由主函数来调用其他函数,其他子函数之间也可以互相调用。

C程序的执行总是从main( )函数开始。调用其他函数完毕后,程序流程回到main( )函数,继续执行主函数中的其他语句,直到main( )函数结束,则整个程序的运行结束。

从用户的使用的角度看,函数可分为:

- 标准函数,即库函数。这些函数由系统提供,可以直接使用。
- 自定义的函数。用以解决用户需要时设计定义的函数。

从函数的形式看,函数可分为:

- 无参函数。
- 有参函数。

8.2　函数参数和函数返回值

8.2.1　形式参数和实际参数

- 在定义函数时,函数名后面括号中的变量称为“形式参数”(简称“形参”)。

• 在主调函数中,函数名后面括号中的参数(可以是一个表达式)称为“实际参数”(简称“实参”)。

8.2.2　函数的返回值

1. 定义:函数的返回值就是通过函数调用使主调函数能得到一个确定的值。

2. 表达形式:return 表达式; 或 return(表达式); 或 return;

• return 语句中的表达式值的类型必须与函数首部所说明的类型一致。若类型不一致,则以函数值的类型为准,由系统自动进行强制转换。

• 当函数没有指明返回值,或没有返回语句时,函数返回一个不确定的值。为了使函数不返回任何值,可以使用 void 定义无类型函数。

8.3　函数的调用

1. 函数调用的一般形式

一般形式:函数名(实参表列);

函数的调用可分为调用无参函数和调用有参函数两种:

• 调用无参函数,不用“实参表列”,但括号不能省略。

• 调用有参函数时,若实参列表中有多个实参,各参数间用逗号隔开。实参与形参要求类型一致。

2. 函数的说明

C 语言中,除了主函数外,对于用户定义的函数要遵循先定义后使用的规则。把函数的定义放在调用之后,应该在调用之前对函数进行说明(或函数原型说明)。

函数说明的一般形式如下：

类型名　函数名(参数类型 1 ,参数类型 2 ,…,参数类型 n)；

或

类型名　函数名(参数类型 1 参数名 1,参数类型 2 参数名 2 ,…,参数类型 n 参数名 n)；

8.4　函数的嵌套调用与递归调用

8.4.1　函数的嵌套调用

C 语言的函数定义都是独立的、互相平行的,C 语句不允许嵌套定义函数,即一个函数内不能定义另一个函数。但可以嵌套调用函数,即在调用一个函数的过程中,又调用另一个函数。

8.4.2　函数的递归调用

在调用一个函数的过程中又出现直接或间接地调用该函数本身的,称为函数的递归调用。

使用递归法解决问题,须符合的条件：

- 可以把要解决的问题转化为一个新的问题。而这个新的问题的解决方法仍与原来的解决方法相同,只是所处理的对象有规律地递增或递减。
- 可以应用这个转化过程使问题得到解决。
- 必须要有一个明确的结束递归的条件。

8.5　数组作为函数参数

8.5.1　数组元素作为函数实参

数组元素可以作为函数的实参,与用变量作为实参一

样,按照单向值传递的方式进行传递。

8.5.2　数组名可以作为函数参数

可以用数组名作为函数参数,此时实参与形参都应用数组名,此时的数组名是整个数组的首地址。

8.6　全局变量和局部变量

在函数内部定义的变量称为局部变量,只能在本函数内部使用。

在函数之外定义的变量称为外部变量,外部变量是全局变量。全局变量可以为本文件中其他函数所共用,它的有效范围从定义变量开始到本文件结束。

如果在同一个源文件中,外部变量与局部变量同名,则在局部变量的作用范围内,外部变量被“屏蔽”,即它不起作用。

8.7　变量的存储类别

8.7.1　auto 变量

当在函数内部或复合语句内定义变量时,如果没有指定存储类别,或使用了 auto 说明符,系统就认为所定义的变量具有自动类别。

8.7.2　register 变量

寄存器变量也是自动类变量。它与 auto 变量的区别仅在于:用 register 说明变量是建议编译程序将变量的值保留在 CPU 的寄存器中,而不是像一般变量那样占用内存单元。

8.7.3　静态存储类别的局部变量

当函数体(或复合语句)内部用 static 来说明一个变量时,可以称该变量为静态局部变量。它与 auto 变量、register 变量的本质区别是:

- 在整个程序运行期间,静态局部变量在内存中的静态存储区中占据着永久性的存储单元。即使退出函数后,下次再进入该函数时,静态局部变量仍使用原来的存储单元。由于不释放这些存储单元,这些存储单元中的值得以保留,因而可以继续使用存储单元中原来的值。
- 静态局部变量的初值是在编译时赋予的,在程序执行期间不再赋以初值。对未赋值的局部变量,C 语言编译程序自动给它赋初值为 0。

第9章　指　针

9.1　关于地址和指针

在C语言中,将地址形象地称为“指针”。一个变量的地址称为变量的“指针”。一个专门用来存放另一个变量的地址的变量(即指针),则称它为“指针变量”。

9.2　变量的指针和指向变量的指针变量

9.2.1　指针变量的定义

定义指针变量的一般形式:类型名 *指针变量名1,*指针变量名2,… ;如:int *p,*t;

9.2.2　指针变量的引用

指针变量中只能存放地址(指针),与指针相关的两个运算符是“&”(取地址运算)和“*”(指针运算符)。

9.2.3　指针变量作为函数参数

- 指针类型数据可以作为函数参数来进行传递。
- 作用:将一个变量的地址传送到另一个函数中,参与该函数的运算。
- 形参指针变量的值的改变不能使实参指针变量的值发生改变。

9.3　数组与指针

9.3.1　指向数组元素的指针

- C 语言规定数组名代表数组的首地址,也就是数组中第 0 号元素的地址。
- 定义指向数组元素的指针变量的方法,与定义指向变量的指针变量相同。

9.3.2　通过指针引用数组元素

如果指针变量 p 已指向数组中的一个元素,则 p + 1 指向同一数组中的下一个元素。

9.3.3　用数组名作为函数参数

- 当数组名作为参数被传递时,若形参数组中各元素发生了变化,则原实参数组各元素的值也随之变化。
- 如果用数组元素作为实参,情况就与用变量作为实参时一样,是“值传递”方式。

9.3.4　指向多维数组的指针和指针变量

多维数组可以看作是一维数组的延伸,多维数组的内存单元也是连续的内存单元。C 语言实际上是把多维数组当成一维数组来处理的。

9.4　字符串与指针

9.4.1　字符串的表示形式

- 用字符数组存放一个字符串,然后输出该字符串。
- 用字符指针指向一个字符串。用字符指针指向字符串中的字符。

9.4.2　字符串指针作函数参数

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,即用字符数组名作为参数或用指向字符串的指针变量作为参数,进行传递。

9.4.3　字符指针变量和字符数组的区别

- 字符数组是由若干个元素组成的,每个元素中存放一个字符,而字符指针变量中存放的是地址,绝不是将字符串的内容存放到字符指针变量中。
- 赋值方式不同。
- 字符数组可以在定义时对其整体赋初值,但在赋值语句中不能完成整体赋值。而字符指针变量既可以在定义时赋初值,也可以在赋值语句中完成。
- 编译时不同。
- 在程序中指针变量的值可以改变。而数组名虽然代表了地址,但它的值是一个固定的值,不能改变。

9.5　指向函数的指针

指针变量可以指向一个函数,编译时,一个函数将被分配给一个入口地址,这个入口地址就称为该函数的指针。因此,可以通过使用一个指向函数的指针变量调用此函数。

说明:

- 指向函数的指针变量的一般定义形式为:数据类型(*指针变量名)();。
- 在给函数指针变量赋值时,只需给出函数名而不必给出参数。

● 用函数指针变量调用函数时,只需将(* s)代替函数名即可(s 为已经定义过的指向函数的指针变量名),在(* s)之后的括号中根据需要写上实参。

● 对指向函数的指针变量,有些运算,如 + + s、- - s、s +3 等都是没有意义的。

9.6　指针数组和指向指针的指针

9.6.1　指针数组的概念

若在一个数组中,其元素均为指针类型数据,这样的数组称为指针数组。

一维指针数组的定义形式为:类型名 * 数组名[数组长度];。

9.6.2　指向指针的指针

指向指针数据的指针变量,简称为指向指针的指针,通常称为二级指针。

定义一个指向指针数据的指针变量的形式:

类型名 * * a。

第 10 章　编译预处理和动态存储分配

10.1　宏定义

10.1.1　不带参数的宏定义

1. 定义形式:#define 宏名　替换文本　或 #define 宏名。

2. 说明:

- 在 define 宏名和宏替换文本之间要用空格隔开。
- 可以用 #undef 命令终止宏定义的作用域。
- 在进行宏定义时,可以引用已定义的宏名。
- 同一个宏名不能重复定义。

10.1.2　带参数的宏定义

1. 定义形式:#define 宏名(参数表) 字符串

宏定义不只进行简单的字符串替换,还可进行参数替换。

2. 执行过程:

- 如果程序中有带实参的宏,则按#define 命令行中指定的字符串从左到右进行置换。
- 如果字符串中包含宏中的形参(如 x,y),则将程序语句中相应的实参(可以是常量、变量或表达式)代替形参。
- 如果宏定义中的字符串中的字符不是参数字符(如(x * y)中的“ * ”号),则保留。这样就形成了置换的字符串。

10.2　关于动态存储的函数

10.2.1　malloc( ) 函数

- 函数原型为:void * malloc(unsigned int size);
- 函数的作用:系统自动在内存的动态存储区中,分配长度为 size 的一段连续空间。若此函数执行成功,则函数返回值为指向被分配域的起始地址的指针(该函数的返回值的基本类型为 void)。若该函数执行失败(如内存空间不足的情况),则函数返回值为空指针(NULL)。

第 11 章　结构体和共用体

11.1　用 typedef 说明一种新类型名

1. 一般形式:typedef　类型名　标识符;

其中,“类型名”一定是在此语句之前已有定义的类型标识符。“标识符”是一个用户定义标识符,用来标识新的类型名。

2. typedef 语句的作用:

用“标识符”来代表已存在的“类型名”,并没有产生新的数据类型,因此,原有的类型名依然有效。

3. 声明一个新的类型名的具体步骤:

- 先按定义变量的方法写出定义的主体(如 float a;)。
- 将变量名换成新类型名(如将 a 换成 FLO)。
- 在最左面加上关键字 typedef(如 typedef float FLO;)。
- 然后可以用新类型名去定义其他的变量(如 FLO b;)。

11.2　结构体数据类型

1. 声明一个结构体类型的一般形式为:

struct 结构体名
{ 成员表列 };

2. 结构体类型可以用以下形式说明:

```
struct 结构体标识名
{
    类型名 1    结构体成员名表 1;
    类型名 2    结构体成员名表 2;
    ……
    类型名 n    结构体成员名表 n;
};
```

说明:

(1)“结构体标识名”和“结构体成员名表”都必须是合法的用户定义的标识符。

(2)每个“结构体成员名表”中都可以含有多个同类型的成员名,它们之间以逗号分隔。

(3)结构体类型说明中的“类型名 1”~“类型名 n”,不仅可以是简单数据类型,也可以是某种结构体类型。当结构体说明中又包含结构体时,称为结构体的嵌套。

(4)ANSI C 标准规定结构体至多允许嵌套 15 层,并且允许内嵌结构体成员的名字与外层成员的名字相同。

11.3　结构体类型变量的定义

11.3.1　先声明结构体类型再定义变量名

如已经定义了一个结构体类型 struct time,可以如下定义:

```
struct time          time1,time2;
结构体类型名         结构体变量名;
```

time1 和 time2 为 struct time 类型变量,即它们都具有 struct time 类型的结构。

11.3.2　在声明类型的同时定义变量

其一般形式为：

struct 结构体名 { 成员表列 } 变量名表列；

11.3.3　直接定义结构体类型变量

其一般形式为：

struct { 成员表列 } 变量名表列；

即不出现结构体名。

11.4　结构体变量的引用

引用结构体变量时应注意：

- 结构体变量不能作为一个整体而对其进行任何操作，只能对结构体变量中的各个成员分别进行输入和输出等操作。结构体变量中的成员用以下方式引用：结构体变量名．成员名
- 如果结构体的某个成员本身又是一个结构体类型，则可以使用若干个成员运算符“．”，一级一级地找到最低的一级成员，只能对最低一级的成员进行赋值或存取及运算。
- 结构体变量的初始化，是指逐个对结构体变量的各个成员进行初始化的过程。

11.5　结构体数组

1. 一般形式为：struct 结构体变量名 { 成员表列} 数组名[常量表达式]；

2. 结构体数组的初始值应顺序地放在一对花括号中。

11.6　指向结构体类型数据的指针

11.6.1　指向结构体变量的指针

- “ - >”称为指向运算符
- “结构体变量 . 成员名”、“(* 结构体指针变量名). 成员名”和“结构体指针变量名 - > 成员名”这 3 种形式是等价的。

11.6.2　指向结构体数组的指针

结构体数组及其元素也可以用指针变量来指向。在使用指针变量指向结构体数组时,只要把该结构体数组中的每个元素当做普通的结构体变量使用即可。

11.6.3　用结构体变量和指向结构体的指针作为函数参数

将一个结构体变量的值传递给另一个函数,有如下方法:

- 结构体变量的成员作为实参传递给主调函数。
- 可以用结构体变量作为一个整体实参。
- C 语言中,允许将结构体变量的地址作为实参传递,这时,对应的形参应该是一个基类型相同的结构体类型的指针。

11.7　链表

11.7.1　链表的概念

1. 定义:

链表是一种常见的重要的数据结构,它是动态地进行存储单元分配的一种结构。

2. 说明：

- 链表中的各元素在内存中不一定是连续存放的。
- 一个节点中应包含一个指针变量，用它存放下一节点的地址。
- 链表最后一个结点的指针域置成'\0'(NULL)值，标志着链表的结束。
- 每一个链表都用一个“头指针”变量来指向链表的开始，称为 head 指针。在 head 指针中存放了链表第一个结点的地址。

11.7.2　顺序访问链表中各节点的数据域

所谓“访问”，可以理解为取各节点的数据域中的值进行各种运算、修改各节点的数据域中的值等一系列的操作。

输出单向链表各节点数据域中内容的算法比较简单，只需利用一个工作指针(p)，从头到尾依次指向链表中的每个节点，当指针指向某个节点时，就输出该节点数据域中的内容，直到遇到链表结束标志为止。如果是空链表，就只输出提示信息并返回调用函数。

11.7.3　删除链表中的节点

为了删除单向链表中的某个节点，首先要找到待删除的节点的前趋节点(即当前要删除节点的前面一个节点)，然后将此前趋节点的指针域去指向待删除节点的后续节点(即当前要删除节点的下一个节点)，最后释放被删除节点所占的存储空间即可。

第12章　文　件

12.1　C语言文件的概念

C程序把文件分为ASCII文件和二进制文件。ASCII文件又称文本文件。

在C语言中,文件是一个字节流或二进制流,也就是说,对于输入输出的数据都按“数据流”的形式进行处理。

文件输入输出方式也称“存取方式”。C语言中,文件有两种存取方式:顺序存取和直接存取。

12.2　文件类型指针

可以用该结构体类型来定义文件类型的指针变量,一般形式为:

```
FILE *fp;
```

fp是一个指向FILE结构体类型的指针变量。

12.3　文件的打开与关闭

1. fopen( )函数

(1)调用形式:fopen(文件名,文件使用方式);

- 函数返回一个指向FILE类型的指针。
- 无论哪种使用方式,当打开文件时出现了错误,fopen函数都将返回NULL。

(2)最常用的文件使用方式及其含义:

- “r”:为读而打开文本文件。
- “rb”:为读而打开一个二进制文件。
- “w”:为写而打开文本文件。

- “wb”:为写而打开一个二进制文件。
- “a”:为在文件后面添加数据而打开文本文件。
- “ab”:为在文件后面添加数据而打开一个二进制文件。其余功能与“a”相同。
- “r +”:为读和写而打开文本文件。
- “rb +”:为读和写而打开一个二进制文件。
- “w +”:首先建立一个新文件,进行写操作,随后可以从头开始读。如果指定的文件已存在,则原有的内容将全部消失。
- “wb +”:功能与“w +”相同,只是在随后的读和写时,可以由位置函数设置读和写的起始位置。
- “a +”:功能与“a”相同,只是在文件尾部添加新的数据之后,可以从头开始读。
- “ab +”:功能与“a +”相同,只是在文件尾部添加新的数据之后,可以由位置函数设置开始读的起始位置。

2. fclose( ) 函数

调用形式:fclose(文件指针)

- 若对文件的操作方式为“读”方式,则经以上函数调用之后,要使文件指针与文件脱离联系。可以重新分配文件指针去指向其他文件。
- 若对文件的操作方式为“写”方式,则系统首先把该文件缓冲区中的剩余数据全部输出到文件中,然后使文件指针与文件脱离联系。
- 在完成了对文件的操作之后,应当关闭文件,否则文件缓冲区中的剩余数据就会丢失。
- 当执行了关闭操作后,成功则函数返回0,否则返回非0。

12.4　文件的读写

1. fread( ) 函数和 fwrite( ) 函数

当要求一次性读写一组数据时,例如,一个实数或一

个结构体变量的值,就可以使用 fread( ) 函数和 fwrite( ) 函数,它们的一般调用形式为:

fread(buffer,size,count,fp) ;

fwrite(buffer,size,count,fp) ;

其中,buffer 代表的是一个指针变量;size 代表的是要读写的字节数;count 用来指定每读写一次,输入或输出数据块的个数(每个数据块具有 size 个字节);fp 是文件类型指针。

2. fscanf( ) 函数和 fprintf( ) 函数

fscanf( ) 函数和 fprintf( ) 函数都是格式化的读写函数,与 scanf( ) 和 printf( ) 函数作用相似,但 fscanf( ) 函数和 fprintf( ) 函数读写对象是磁盘文件上的数据。

它们的一般形式如下:

fscanf(文件指针,格式字符串,输入列表) ;

fprintf(文件指针,格式字符串,输出列表) ;

3. fputs( ) 函数

fputs( ) 函数是用来把字符串输出到文件中,调用形式如下:

fputs(str,fp) ;

其中 str 是要输出的字符;fp 是文件指针,字符串末尾的 '\0' 不输出。

12.5　文件的定位

1. rewind( ) 函数

rewind( ) 函数的调用形式如下:

rewind(fp) ;

该函数的功能是使文件的位置指针重新返回到文件的开头,其中 fp 为文件指针,且该函数没有返回值。