# 第一章 绪 论

## 1.1 综 述

计算机软件开发一直被两大难题所困扰：一是如何超越程序复杂性障碍；二是如何在计算机系统中自然地表示客观世界，即对象模型。由C++语言写的面向对象程序设计是软件工程学中的结构化程序设计、模块化、数据抽象、信息隐藏、知识表示、并行处理等各种概念的积累与发展，是解决上述两大难题的90年代最有希望、最有前途的方法。面向对象程序设计是软件开发方法的一场革命，它代表了新颖的计算机程序设计的思维方法。该方法与通常结构程序设计十分不同，它支持一种概念，即旨在使得计算机问题的求解更接近人的思维活动，人们能够利用C++语言充分挖掘硬件潜在能力，在减少开销的前提下，提供更强有力的软件开发工具。

面向对象程序设计是软件系统的设计与实现的新方法。这种新方法是通过增加软件可扩充性和可重用性，来改善并提高程序员的生产能力，并能控制维护软件的复杂性和软件维护的开销。当使用面向对象程序设计方法时，软件开发的设计阶段更加紧密地与实现阶段相联系。在软件设计与实现中当今有许许多多方法，面向对象方法是在实践中超越其他许多方法的潜在的最有前途的方法。虽然在各个应用领域中报道了面向对象程序设计的巨大成功，但就一般的软件设计和开发舞台来评价面向对象方法学的成功尚为时太早。面向对象方法学包含了分析、设计和实施的面向对象方法。这部分是当今最弱的部分，面向对象对软件系统开发起着关键作用。本书的目的是介绍软件开发崭新的方法和C++语言，C++语言非常适合于面向对象程序设计。

C++这个名字表示了从C语言进化的特征。"++"是C的增量操作符，顾名思义，C++是C的扩充，即C++语言在提供面向对象程序设计的同时保留C语言的高度简洁和高效率等优点。我国已法定用UNIX操作系统，而UNIX操作系统90%的代码用C语言编写，C语言具有高效灵活、易于理解、可移植性好等优点，C++语言不仅保留这些优点而且包含了几乎所有面向对象特征，C++将代替C是肯定的。所以，本书为了学习与研究的一致性和连贯性，第二章将简洁地介绍C语言知识，对于非常熟悉C语言的读者可跳过此章。

本书的逻辑梗概是：绪论→C语言→C++语言→面向对象特征→典型实例剖析。

## 1.2 面向对象程序设计

面向对象程序设计中心是围绕几个主要概念：抽象数据类型和类，类型层次(子类)，继承性和多态性。类和继承是符合人们一般思维方式的描述模式。

什么叫对象(object)？

对象通常作为计算机模拟思维，表示真实世界的抽象。一个对象像一个软件构造块，

它包含了数据结构和提供相关的行为(操作)。对象本身可为用户提供一系列服务，可以改变对象状态、测试、传递消息等等，用户无需知道服务的任何实现细节，操作完全是封闭的。

抽象数据类型是面向对象程序设计的中心概念之一。一个抽象数据类型是一个模型，此模型包含一个类型和与之相关的操作集。定义这些操作集的是基本类型的行为。

在大多数面向对象语言中，类的定义描述以抽象数据类型为基础的对象行为，抽象数据类型定义了能以类型为基础执行所有操作的接口。类定义也指定了实现细节或类型的数据结构。通常这些实现细节仅在该类范围内存取，我们称这种类型为私有(private)类型。当数据类型的全部或部分在该类范围外存取，我们称这样的类型为公共(public)类型。

按面向对象的说法，为一个类而定义的所有操作称之为方法(Methods)。这些方法类似于非面向对象语言中的过程和函数。如果一个类的 Public 操作能在许多应用领域中被充分应用，那么可以说类是组成可重用软件程序块的基础。

一个对象(object)是被一个特定类说明的变量。这样一个对象通过包含在类定义中的所有域的拷贝(private 和 public 两部分)来封装。通过访问一个或几个在类定义上的方法，可以对这个对象实施各种操作。引用一个方法的过程称之为向这个对象发送一个消息(Message)。一个典型的消息包含的一些参数，正像在非面向对象语言中过程或函数调用(或引用)的一些参数一样，一个典型引用方法的操作是修改存储于特定对象中的数据。

每个类变量或对象表示该类的一个实例(instance)。如果几个对象被定义具有同样的类，它们将是包含有不同值的一个集合。

面向对象方法通过子类提供类型的等级层次。一个子类(subclass)定义一个对象集合的行为，该对象继承父类(parent class)的各种特征，子类反过来又将它自己的或继承来的特征传递到它的子类中。借助允许建立子类的方法，可使软件开发的周期缩短，从而降低开发复杂性和费用。

子类能导致增加问题解的能力。用基本类的集合建立子类代替修改现存的软件，或坏的软件，或重写的软件。这些新的子类的对象组成了软件构造的从属结构。

## 1.3 面向对象问题解

一个面向对象的软件系统的结构建立在一系列类上，这些类刻划了系统中所有要处理的基本数据类型的行为。每个类的许多对象可通过引用该类的方法来操纵，即向这些对象发送消息，这些消息表示了可在这个类的对象集合上发生的操作(活动)。

面向对象程序设计集中在对数据操作而不是对过程操纵。数据构成了软件分解的基础。的确，面向对象软件设计的主要挑战是将软件系统分解为基本数据类型或者类和子类，以及对每个基本类和子类特性的定义。对象即类(或子类)变量相应于实际问题领域中的物理的或逻辑的实体。

定义一个面向对象软件系统的结构框架和构成一个高层系统设计，最终只表现为一系列类(子类)、它们的定义和对象。用方法接口描述每个类的行为特征。这个方法的实现

细节不是系统高层设计所关心的部分。

在典型的面向对象语言中，类中定义与方法的接口可以和实现细节的定义分开，允许系统的设计和系统实现分开。概念(与方法接口有关的类定义)和它的实现(实现这个类的数据结构和算法的代码)的分离在面向对象的程序设计中是非常重要的。

概念和它的实现的分离在构成重复使用性以及控制维护的花费上也是很重要的。

可重用性在面向对象程序设计中十分重要，其原因为在类中的封装概念由方法接口形式提供。用户仅需要了解类的对象作为在方法接口上的特定行为，而无需关心他们的实现。从用户观点来说，方法实现被包含在隐藏细节的"黑盒"中。

在面向对象程序设计中可维护性也很重要。这是因为改变数据结构或算法(即实现类的代码)仅局限于实现这个类(或类的一部分)的代码区域内。这对程序的维护提供了方便。

面向对象软件开发的主要目的是：

- 用可重用软件分解(基于基本类)和用子类加快问题求解，缩短开发时间和减少软件开发费用。
- 通过改变一个或多个类的实现，使其影响局部化，从而降低软件维护的花费。

面向对象系统的可靠性可以得到增强，因为初始设计阶段就注重高层的整体性，构成系统的主要部件从一开始就被正确组合在一起。每个主要部件是用其抽象特性定义，高层集成整体性的测试在许多低层详细设计之前已经设计好或已实现了，这些贡献改善了可靠性。

面向对象程序设计提供了一个快速原型可使用平台。在软件系统高层分解进入类并从这些类的对象被完成之后，许多重要方法(赋予系统行为)能快速简单地实现，最后可达到细节的完美实现。

## 1.4 类、对象和封装

一个类描述涉及到定义该类任一对象的一个实例的所有行为特征。实际上，一个类定义的是一种对象类型。

一个类定义的私有部分通常定义了以数据类型为基础的数据结构。仅仅在该类范围内部可以存取指定的方法，这些方法经常支持公共(public)方法的实现。有时一个类的私有部分可以包含其他类的对象，这个其他类仅在给定类的范围内部才能被操纵。

一个类的公共部分通常指定对方法的接口，在许多交叉应用领域中它们构成了这个类的可重用性的基础。这些方法能引用到该类的范围之外，当然是用发送消息到给定类的对象的方式。

封装是用于被定义的各个软件对象的过程中。封装定义为：

1. 所有对象内部软件范围具有清晰的边界；
2. 描述该对象与其他对象如何相互作用的一个接口；
3. 受保护的内部实现。该实现给出了由软件对象提供的功能的细节，实现细节不能在定义该对象的类的范围外进行访问。

封装概念不仅涉及到类的描述，而且如何将问题解的各个组件组装在一起的求精过程。封装的单位是对象，该对象的特性由它自己的类说明来描述。这些特性为相同类的其他对象所共享，对象的封装比讲一个类表示的封装更具体化。有了封装这个定义，一个类的每个实例(instance)在一个问题求解中是一个独立的封装，或称作组件(问题解的分量)。

下例介绍类、对象和封装。在此简要研究一下二叉查找树(a binary search tree)的抽象(详见本书第五章)。

在最高层上，两个对象由基本表示二叉查找树来标识。这些对象的第一个是由称之为类树的实例来表示，因为二叉查找树是由许多节点组成，对象的第二个用称为树节点类的实例来表示。

树节点类赋于下述特点：

- 树节点类的实例是由二叉查找树构成的对象
- 每个树节点包含了确定在结构中数据排序关系的一个对象
- 一个二叉查找树包含了指向左子树和指向右子树的许多对象

在树节点类的对象定义的方法为：

- new_node　建立一个树节点类的新实例
- key　在确定树排序的节点中，返回数据存储域
- left　返回给定节点的左子节点
- right　返回给定节点的右子节点

以下图 1.1 描绘了树节点类。

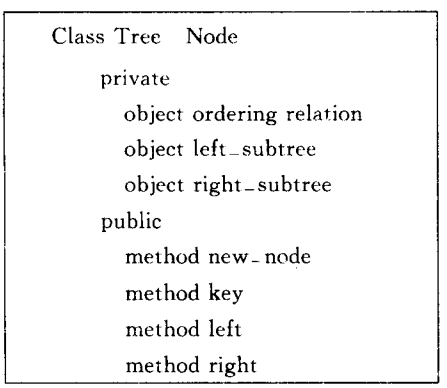


图 1.1　树节点类的描述

树类有以下特征：

- 树类的实例是二叉查找树
- 一个二叉查找树是以称为根节点的一个具体节点及由许多树节点组成，对树对象的所有存取均通过根节点进行
- 在二叉查找树的节点均相对于某些关键对象排序，在左子树中具有较小键值而在右子树中具有较大键值
- 在二叉查找树的节点的插入和删除必须产生二叉查找树的一个对象

在树类的对象定义的方法为：

- define　　建立一个新树
- insert　　在树中插入一个新节点
- remove　　从树中删除已存的节点
- is_present　　确定是否是树中一个具体节点
- display　　在树中按序输出节点的集合

图 1.2 描绘了树类

```
Class Tree
    private
      object root_node
    public
      method define
      method insert
      method remove
      method is_present
      method display
```

图 1.2　树类的描述

## 1.5　子类——继承性和多态性

我们现在介绍类层次概念。在层次体系中，某些类是从属于其他类，或称为子类，在 C++ 中称为导出类(derived classes)。子类是被认为层次体系组合下的类的具体情况，通常在类层次体系中的下层表示一个增加的详细说明，而高层通常表示更高的概括。

在大多数面向对象语言中，如果类 P 是子类 S 的一个父辈，则子类 S 的一个对象 s 能使用父类 P 的一个对象 p。这蕴含着消息(即操作)公共集合能发送到类 P 和类 S 的各个对象。当同样消息能被发送到父类的对象和它的子类的对象时，我们把此定义为多态性(polymorphism)。

多态性允许每个对象响应公共消息格式，即用合适的方式从一个对象取来送到子类对象去。例如，一个打印方法可定义输出到赋予一个对象状态的某个数据域。多态性的相同消息 print，被送到一个类和它的子类里面的所有对象，这样，每个对象知道如何响应这个消息，也允许用与其他子类对象的不同方式响应。例如，不同数据域可以从每个不同的子类的对象输出。在不同类型对象上，对一个类似的操作，使用相同消息的能力是与问题解的人的思维方式相一致的。对打印整数、浮点数、字符、字符串和数据记录等使用不同的术语是不自然的。多态性是对问题解的面向对象方法中的关键特征之一。

不同的面向对象语言提供了一个能力范围，此能力涉及到一个子类的对象的规模，即可继承、可扩充或取而代之父类的特征等。某些面向对象语言支持多重继承，即在一个子类中可多于一个父类。这些内容在后面几章将陆续介绍。

## 1.6 面向对象程序设计的挑战

面向对象程序设计的挑战的某些方面是，通过寻找共通性把软件系统划分为类，建立子类来处理具体个性，这样一步步对现存的软件系统构造出一个合适的类和子类的层次体系结构。

### 1.6.1 划分软件分类

对初学的面向对象程序员，用类设计建立一个程序，似乎是不自然的，也比较困难。例如，写一个 rollbook 数据库程序，在此程序中有存储、修改和删除学生的名字和成绩，以及打印这些学生的名字和成绩单。典型的方法是定义 rollbook 的数据类型，然后定义插入、删除、修改和打印方法作为以 rollbook 数据类型为基础的相联系的操作。

插入、删除和修改操作可以用比 rollbook 的类型更合适的与过程相联系的 edit 来代替，这样过程的 edit 能说明为一个类，这个 edit 类能在其他要求数据库信息联机交互编辑应用中重用。rollbook 类能包含 edit 类的一个对象，去完成插入、删除和修改学生的名字和成绩单。

### 1.6.2 对已存的软件系统增加功能

在面向对象程序中每个函数或过程是与一个类相联系的。对已存在的软件系统增加一个新功能，那么设计者或程序员必须决定是否增加一个特殊过程到已存在的类中，或定义一个新类或一个子类。

可重用性问题是决定新功能应当加到面向对象系统什么地方。如果增加的功能从几个已存在类中被对象重用，则有理由基于函数或过程建立一个新类。如果新功能的实现要求访问已存的类型的内部细节，则有理由增加新功能作为添加方法加到现存类中。如果新功能表示对一个给定类中一个现存函数(方法)的修改，则有理由作为一个方法去建立一个子类和增加功能到这个子类中。这个方法就取而代之父类中的类似的方法。

### 1.6.3 类型和子类型的层次结构

建立类和子类的一种方法是自顶向下数据类型分解，在分解中程序员应在类的系统中辨认并按模型建造主要数据元素。从高层的数据分量(类)一层一层地划分为更专门的子类，此过程一直继续到类和对象的层次结构构造完成为止。

作为一个例子，考虑作为系统中主要类的一个汽车，汽车类可以划分成子类发动机，变速器，闸，教练设备，排气装置，悬置等。子类发动机，像其他子类一样，能进一步划为点火装置，燃料喷射装置和起动装置。点火装置子类也能进一步划分为火花活塞和螺线管。对每个这样的类和子类，必须定义相应的操作(方法)集合，汽车类分解图的一部分由图 1.3 示出。

另外一个方法是建立类和子类的所谓自底向上分解，在此分解中许多子类生长(扩充)成一个父类，并对每个相继的子类的对象提供更大的功能。

通常采用自底向上的分解例子是一个屏幕窗口类(screen_window)作为一个父类。这样一个类概括了所有屏幕窗口的最小特性。我们能构造一系列子类,对基本的 screen_window 类加上许多功能,并具有各种各样子类的对象,诸如作为前景和后景颜色的属性、窗口边界的限定,等等。对 screen_window 的部分类分解在图 1.4 描绘。

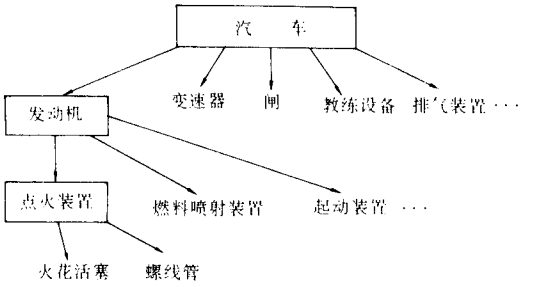


图 1.3 汽车部分类分解图

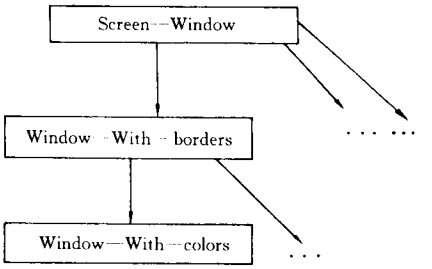


图 1.4 屏幕窗口部分类分解图

通常是两种方法共同使用,即通过从一个父类开始逐步建立子类,当软件工程师了解到几个不相同的子类的之间关系后,再抽象出它们的共有特征到一个或几个父类中,如此反复从而设计出较完美的面向对象程序。

### 1.6.4 应改变典型软件开发过程

当今典型软件开发包含三个独立活动,在它们之间有高墙隔开。

(1) 第一个独立活动是分析和设计活动,它使用各种概念模型,如 ER 模型和 SASD 模型。此活动是把现实世界用 ER 模型描述并进行分析和设计。

(2) 第二个独立活动是程序设计。它使用不同的程序设计语言(如 COBOL,FORTRAN,C 等)。每种语言有自己的概念模型,因而模型的设计和分析技术是十分不同的。程序设计是从一种概念空间(设计现实世界特定表示)到另一种概念空间(编译或解释理解)的事务处理一大部分工作。

(3) 第三个活动是用另外的语言和概念模型定义和存取数据库。对关系型数据库来说,COBOL,FORTRAN 和 C 与处理记录的 SQL 语言有着严重的不匹配。这种典型软件开发,效率不高,应用受到了极大限制。

另一种开发方法称之为无保留库的软件开发过程(No_Vault Software development)。这种开发过程使用面向对象分析和设计方法以及相应的工具,再加上面向对象程序设计语言($C^{++}$)和面向对象数据库。这样具有单一的、又统一的概念模型——对象——在所有开发和维护阶段都使用。其结果是:高生产率(两堵高墙拆除,无需中间保留库,过去那些方法是人为加入的,不符合人们的思维活动);高质量(避免了有保留库时发生的各种错误和不匹配问题);高灵活性(对象组装程序,可重用,移植容易)。

### 1.6.5 对"算法+数据结构=程序设计"的挑战

我们知道,非面向对象的过程语言,如 FORTRAN、C 和 PASCAL,其数据结构是问

题解的中心。一个软件系统的结构是围绕一个或几个关键数据结构为核心而组成的。在这种情况下，计算软件开发一直被两大难题所困扰：一是如何超越程序的复杂性障碍；二是计算机系统中如何自然地表示客观世界即对象模型。诚然，Niklans wirth 提出的“算法＋数据结构＝程序设计”，在软件开发进程中产生了深远的影响。但软件系统的规模越来越大，复杂性不断增长，以致不得不对“关键数据结构”重新评价。数据结构的主要欠缺是应用范围和可视性。在这种系统中，许多重要的过程和函数（子程序）的实现严格依赖于关键数据结构，如果这些关键数据结构的一个或几个数据有所改变，则涉及到整个软件系统，许多过程和函数必须重写，甚至因为几个关键数据结构改变，导致软件系统整个结构彻底崩溃。

近三十年来，计算机科学家为提高软件生产率所作的种种探索与努力，或多或少与面向对象的程序设计这一思想有关联。作为克服复杂性的手段，在面向对象程序设计中，把密切相关的数据与过程定义为一个整体（即对象），而且一旦作为一个整体（包）定义了之后，就可以使用它而无需了解其内部的实现细节。面向对象软件系统的构造不依赖于对象的内部结构，而仅依赖于定义能在对象内部数据上实行操作的方法。

封装和数据隐藏是面向对象问题解和面向对象程序设计的基本要素。它可使一个软件工程师避免许多维护问题。一旦数据（类型）结构修改了，仅对实现模型的代码局部区域作适当变化，软件系统其余部分原封不动，因为修改部分相关联的数据仅仅是在定义模块中的过程和函数的接口部分。

面向对象技术提供给人们的封装和数据隐藏，表示了真正的面向对象语言的事务——在此事务中，是对象而不是函数或过程代表了问题解的中心环节。只有当我们重点不是送数据到函数，而送消息到对象的时候，我们才有了真正的面向对象语言。

# 第二章　C 语言程序设计

## 2.1　C 语言历史和特点

C 语言是一种受到广泛重视并已得到普遍应用的计算机程序设计语言，也是国际上公认的最重要的少数几种通用程序设计语言之一，1990 年底已(通过)成为 ISO 标准通用语言。它适用于作为系统描述语言，即用来写系统软件，也可用来写应用软件。

C 语言原本是 D. M. Ritchie 1972 年在贝尔实验室中为编写 UNIX 系统而设计并加以实现的。因为 C 语言是在 UNIX 系统上研制成的，而且 UNIX 操作系统源代码的 90% 以上以及绝大多数 UNIX 系统实用程序都使用 C 语言编写。所以，C 语言与 UNIX 操作系统紧密地联系在一起。但是，C 语言并没有被束缚在任何特定的硬件或 OS 上，它具有的可移植性可与汇编语言媲美。当然，C 语言也受到了 UNIX 系统的强有力支持。C 语言可以直接使用 UNIX 操作系统面向程序设计的界面——数十条系统调用，可以直接调用 UNIX 系统的各种实用程序等。C 语言既是 UNIX 程序设计环境的一个最重要组成部分，它又与环境中的其它部分有机地联系在一起，这大大加强了其自身的能力。

1973 年 K. Thompson 和 D. M. Ritchie 把 UNIX 系统用 C 重写了一遍。系统代码量比以前的版本大了三分之一，加进了多道程序设计功能，特别是整个系统(包括 C 语言编译本身)建立在 C 语言基础上，而 C 语言又具有良好的可移植性，所以第五版的 UNIX 系统(UNIX $V_5$)就奠定了 UNIX 系统的基础。以后 UNIX $V_6$，$V_7$，System Ⅲ 和最新的 System Ⅴ 都是在 $V_5$ 基础上发展、扩充的。

图 2.1 示出了 C 语言的由来。

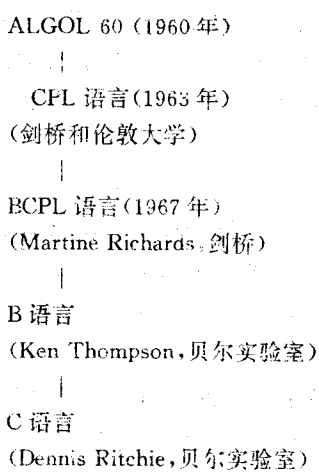


图 2.1　C 语言发展概况

C 语言的成功与它产生的环境及历史背景有关，但起决定作用的是它自身特点：

(1) C 语言的基本组成部分紧凑、简洁，使用一些简单、规整的方法就可以构造出相当复杂的结构；

(2) 表达式简练、灵活、实用，既提高了编译效率和目标代码的质量，又提高了程序的可读性；

(3) 提供了某些接近于汇编语言的功能，如地址处理、二进制数位运算以及指定用寄存器存放变量等。所以，C 语言适合于编写系统程序和各种软件工具；

(4) 具有编写结构良好的程序所需要的各种控制流结构；

(5) 具有各种现代程序设计语言普遍配置的数据结构；

(6) 提供了与地址密切相关的指针以及有关运算符。指针可以指向各种类型的简单变量、函数、数组和结构等；

(7) 为字符和字符串处理提供了良好基础；

(8) 预处理程序和预处理语句提高了程序的可读性、可移植性，给程序调试提供了方便；

(9) I/O 依靠函数调用实施，C 语言的标准程序库具有多种使用方便、功能强的 I/O 函数；

(10) 生成的目标代码质量高。即用 C 语言和汇编语言分别编写程序，那么前者生成的目标代码的运行效率仅比后者的低 10%～20%。

## 2.2 用 C 语言开始编写程序

### 2.2.1 C 语言程序开发的基本过程

基本过程如图 2.2 所示。

在系统帮助下，开发 C 语言程序的第一步是编辑，也就是使用系统提供的编辑程序(例如 UNIX 系统的文本行编辑程序 ed、屏幕编辑程序 vi)将以 C 语言编写的程序以文件形式存入文件系统，文件名最好与程序功能有一定联系。在 UNIX 系统下，C 程序文件名的最后两个字符应为.C。用 C 语言编写的程序称为源程序，它一旦以文件形式存放在文件系统中，就可以调用 C 语言编译程序对它进行编译，在 UNIX 系统中，这是通过使用 CC 命令实施的。即

CC　文件名.C

这就是要求对“文件名.C”中的 C 源程序进行编译处理，在编译过程中如果发现源程序在词法、语法、语义等方面有错，则编译程序向用户报告这些错误并适时结束编译过程。然后用户再使用编辑程序修改源程序中的错误，接着重新启动编译过程。如此不断反复，直至排除了源程序中所有语法错误。

编译程序对一个语法上正确的源程序进行处理的结果是构成一个功能相同的汇编语言形式的程序。编译过程的下一步是调用汇编处理程序将汇编程序翻译成机器指令形式的程序，这种程序称为目标程序。在 UNIX 系统中，汇编处理程序调用是由 CC 命令执行程序自动进行的，在另外一些系统中，用户可能需要使用系统提供的另一条命令。

汇编处理程序将汇编程序转换成目标程序，并将它存放到另一个文件中。在 UNIX

系统下，目标程序文件名与源程序文件名除最后一个字符外其余字符相同，其最后两个字符为.o。

汇编处理结束后，编译过程进入第三步——连接。在UNIX系统下，这同样是由CC命令执行程序自动进行的。连接的目的是构成在计算机上可以立即执行的目标程序。连接的对象除上一步刚刚获得的目标程序外，还可以包括另外一些以前编译好的目标程序或者目标码形式的系统库程序。

连接程序构成的可执行目标程序存放在另一个文件中。在UNIX系统下，该文件名可以在调用CC命令时指定，例如：

CC -o 文件名.out 文件名.c

将可执行目标程序文件名指定为文件名.out，如果在使用CC命令时没有指定，那么将它取为省缺名a.out，要想执行该程序时，就用a.out作为命令a.out即可。此命令a.out将a.out文件中可执行目标码装入内存并分配一定的数据区和工作区，然后启动执行该程序。

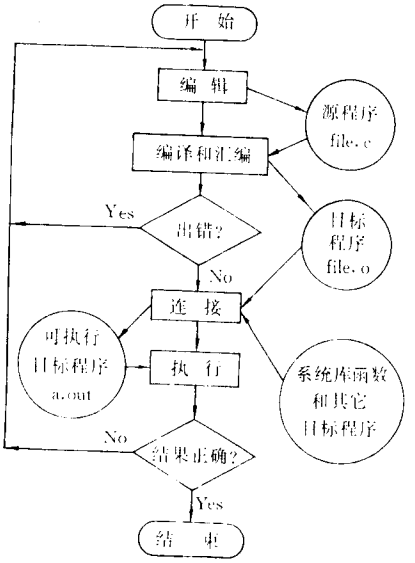


图 2.2 用C语言开发程序的基本过程

程序在执行时如果没有产生预期的结果，就应当检查程序的逻辑，包括使用的算法、数据结构以及程序的组织等。这种检查过程称为排错。在进行排错处理时，通常需要修改源程序，于是编辑、编译、汇编、连接、装入和执行过程再次反复，直至程序提供了各种预期功能，整个C语言程序的开发过程才告一段落。

在此，我们还要介绍一下C语言的检查程序lint。这个检查程序是为了弥补CC对C源程序的语法检查不够而开发的。它可检查：

(1) 语法上的明显错误，如括号不配对等等；

(2) 警告可能出错的地方，如数据类型与说明不一致等；

(3) 警告指出这段程序在这种编译环境下编译是正确的，但如果将这段程序移植到其他环境可能要作改动的地方。

lint的格式：$ lint [可选项] C源程序文件名

例 $ lint a.c b.c

注意使用lint速度比CC慢，而且是在我们需要之时才用，它使用的场合是：

* 程序compiler(编译)通过了，但运行执行文件时仍不正确且很难找出错误的原因，这时使用lint可提供大量信息，帮你分析错误原因。
* 一个程序要移植到其他系统时，审查哪些地方可能要修改。

### 2.2.2 用C语言编写程序

让我们从最简单的程序开始。

```
main ( )
{
  Printf(" Welcome. \n");
}
```

如果将这段程序输入计算机，进行编译，最后执行，则在终端上将显示：

welcome.

通过这段程序可知道：

(1) 最简单的C语言程序只包括一个函数，函数名为main。通常，一个C语言程序可以包括若干个函数，但是其中应该有一个也只有一个以main命名的函数。程序总是从main函数开始执行。函数名后的一对括号中可以包含若干个参数。

(2) 紧接着的一对花括号组相当于pascal语言中的BEGIN和END，分别表示程序的开始和结束，其中包含的语句提供程序的预定功能。在上述程序中只包含一条语句，该语句调用名为printf的标准系统库函数。

(3) printf函数的作用是进行格式化打印。而这里其作用是将参数字符串"Welcome. \n"在终端上显示或打印出来。

(4) 参数字符串中的最末两个字符\和n合并起来表示终端控制字符"新行"。在新行符后的任何字符将显示在终端的下一行上。

(5) 在printf的右括号后立即出现一个分号，它表示语句终止。

(6) 考虑到人们读、写英语的习惯，C语言主要使用小写英文字母。

(7) 在书写和显示格式上，C语言并不计较每行的起始位置，也就是说在每一行的任何位置上都可以输入语句。这一点有助于清楚地显示程序的结构特性，因而大大地提高了程序的可读性。

对上述C语言稍为扩充如下：

```
main ( )
{
  printf ("Welcome. \n");
  printf ("Welcome to Beijing. \n");
  printf ("Right! \n OK! \n OK !! \n");
}
```

输出：

```
Welcome.
Welcome to Beijing.
Right!
OK!
OK!!
```

此程序中包括了三条printf函数调用语句。第三条语句则在终端上显示出三行。

printf函数也可显示其它参数的值和计算结果。如：

**例 2.1** 加两个整数并显示结果

```
/* this program adds two integer values and display the results.
 */
main ( )
{
  /* Declare variables */
```

```
int value1, value2, sum;
/* Assign values and compute the result
*/
value 1=50;
value2=25;
sum=value1+value2;
/* Display the results */
printf ("The sum of %d and %d is %d \n", value1, value2, sum);
}
```

输出：

The sum of 50 and 25 is 75

与前面两个程序相比，要复杂一些，作如下解释：

(1) 关于注释。在C语言程序中，注释语句由字符/和*开始，以*和/结束。其作用是使程序文本化并增强其可读性，说明了程序设计者对整个程序和某些语句的想法。

(2) 变量说明。C语言程序中的所有变量在使用之前都需要进行说明。变量说明语句通知C编译程序，程序将如何使用有关变量。C编译程序利用这些信息产生正确的指令来存取变量的值。

int value1, value2, sum;

它说明变量value1，value2和sum的类型为整型(int)。变量说明语句至少包括变量类型和变量表两部分。

(3) 赋值语句。本程序中包含了三条赋值语句，它们分别将50赋给变量value1；25赋给value2；value1及value2值的和赋给sum。语句中的字符'='为赋值符；'+'为加法运算符。

(4) 带几个参数的printf函数调用。本程序中printf调用与前面两个程序不同，它除了打印格式说明字符串以外，还带有三个参数value1、value2和sum，相互之间用逗号分隔。打印格式说明字符串由两部分组成，一部分是参数打印格式说明，另一部分是直接显示的字符。参数打印格式说明由特殊字符%开始，接着的一个字符说明在这一位置上要显示的值的类型，例如字符d表示要显示的是一个整型值。printf函数在打印格式说明字符串中如果发现%d，它就自动在该位置上以整型格式显示下一个参数。

printf提供了显示程序执行结果的方便手段。

## 2.3 变量、常数、数据类型和算术表达式

### 2.3.1 变量

从上述的最简单的例子中，可以看出在程序执行过程中，变量的值是可以改变的。在C语言中，变量的值不仅可以是整型数也可以是浮点数、字符等，所以，在使用变量之前必须对变量的类型作适当的说明。

变量名的构造规则非常简单，它们必须是字母(大、小写)、下划线字符和数字(0～9)的组合，而且第一个字符应当是字母或下划线字符。下面是合法的变量名：

```
element
i
```

day_of_year
—abc

而

price $
day of year
3class
int

是不合法的。

在 C 语言中，小写和大写字母是有区别的，所以变量名 element、Element、ELEMENT 表示三个不同的变量名。组成变量名的有效字符数随系统而变，有些系统允许变量名长达 31 个字符。但在很多系统中，只取变量名的前八个字符，其它则被舍弃。在选择变量名时应尽可能使名字反映变量的预定用途。如同注释语句一样，有意义的变量名能够显著地增加程序的可读性，并且在纠错和文献化方面也会得到好处。

### 2.3.2 基本数据类型和常数

C 语言最基本的数据类型只有四种：

| | |
|---|---|
| char | 字符型； |
| int | 整数型； |
| float | 单精度浮点数型(即浮点型)； |
| double | 双精度浮点型。 |

在 C 语言中，整数、浮点数、单个字符以及字符串统称为常数。只包含整型、浮点型、字符型常数的算式称为常数表达式。

(1) 整型变量和常数

整型变量的说明方式是：

```
int counter;
int j;
```

一个典型的 int 变量能具有的数值范围为－32768 到 32767。该范围可随机器不同而改变。上述范围意味着可用 16 个二进制数位来存储一个整数。

整型常数由包含一个或多个数字字符的序列组成，数字字符间不允许嵌入空格和逗号，数字字符序列前如带有负号，则指明该常数为负数。除十进制表示法以外，整型常数还可以表示成十六进制和八进制形式。若整型常数的最高位为 0，则它就是以八进制形式表示的数，例如，十进制数 127 表示成八进制形式为 0177。在 printf 的打印格式字符串中，使用格式说明符%o(小写英文字母 o)就可以在终端上以八进制记数法显示相应整型参数值。(要注意：它并不显示引导字符 o)

如果整型常数以 0x 或 0X 开头，那就表示用十六进制记数法。它的其它数位必须由 0～9 以及 a～f(或 A～F)构成。a～f 代表 10～15。例如十进制数 127 表示成十六进制形式为 0x7f(或 0X7F)。如将十六进制数 6FA 赋给整型变量 int counter，则可使用语句：

```
counter=0X6FA;
```

在 printf 函数的打印格式字符串应使用格式说明符%x，即

```
printf("value=%x\n",counter);
```

八进制和十六进制常数常用于系统程序和高级应用程序。

(2) 单、双精度浮点型变量和常数

单精度浮点型变量说明方式是：

```
float radius, circum;
```

双精度浮点型变量说明方式是：

```
double arclen, result;
```

单、双精度这两种浮点类型非常近似，但是如果单精度浮点型提供的精度不能满足要求，则可使用双精度浮点型。double 型变量值的最大有效位数通常是 float 型的两倍。在 DEC VAX 系列机中，float 型变量的精度近似于 7 位十进制数而 double 型变量则几乎为 16 位十进制数。

为了在终端上以十进制形式显示单、双精度浮点数，在 printf 中使用浮点打印格式字符%f；为了以科学记数法显示单、双精度浮点数，在 printf 中使用打印格式符%e。

(3) 字符型变量和常数

字符型变量的说明方式是：

```
char alpha, beta,c;
```

一个字符型变量能存储一个字节信息，即单个字符。书写 C 中的字符常数，用单引号将该字符括起来即可。例如，

```
char alpha='a';
char beta='b';
char plus='+';
char dolsign='$';
```

'\n'也是一个合法的字符常数，它表示新行字符，不过这种类型的字符常数是字符转义序列，它以一个反斜杠(\)起始，以表示特定字符。例如：'\n'被 C 编译解释为单个字符——新行，'\+'被解释为制表符(跳格符)，'\b'被解释为退格符。转义序列'\×××'允许程序员通过说明某个字符的八进制值来定义字符常数，例如，char c='\071'

将八进制 071 作为初值赋给 c，其中×××可以是 1、2 或 3 位八进制数字。

转义序列还能用在预处理程序语句中。例如：

```
#define TAB '\t'
#define CARRIAGE_RETURN '\r'
#define NULL '\o'
```

弄清楚以下两种初始化方法：

```
char alpha='0';
```

和

```
char alpha='\0';
```

之间的区别是十分重要的。上面第一条语句用字符零为 alpha 赋初值(即字符零具有 ASCII 值 48)。第二条语句把零为初值赋给 alpha，在 ASCII 中零值称作 null(空)字符。

表 2.1 表示了常用的 C 字符转义序列。

表 2.1 常用的C字符转义序列

| 转义序列 | 意　义 | ASCII码符号 |
|---|---|---|
| \b | 退格 | BS |
| \f | 换页 | FF |
| \n | 换行 | LF |
| \r | 回车 | CR |
| \t | 水平制表 | HT |
| \\ | 反斜杠 | \ |
| \' | 单引号 | ' |
| \0 | 八进制值0 | NUL |
| \×××  | 八进制值××× | (任意值×××) |

(4) 长整型、短整型和不带符号整型

C语言还提供了另外三种基本数据类型——长整型、短整型和不带符号的整型，分别用long int、short int和unsigned int。例如：

```
short int x;     或  short x;
long int y;      或  long y;
unsigned int z;  或  unsigned z;
```

长整型变量的实际精度与计算机系统有关。在整型常数的末尾加上字母L或l就构成了长整型常数(在数和L之间不允许有空格)。因此，说明语句

long int y=123456L;

将变量y说明为long int型，并赋给它初值123456。如果某个整数超过了整型所能表示的范围，那么即使不加L，编译程序也将它自动处理成长整型常数。

为了在终端上显示长整型变量的值，须在整型格式字符d、o或x前增加修饰字符l。于是，格式字符串%ld表示以十进制格式显示长整型参数值，%lo，%lx表示八进制和十六进制显示长整型参数值。

短整型变量用于存放比较小的整型数(一般只占用整型变量存储空间的一半)。如果一个变量只存放非负整数，就可以将它说明为不带符号整型unsigned int。这种类型的变量和int型变量占用的存储空间通常相等，所以，它可以存放的正整数范围较一般整型显著扩大。如在PDP-11中，一般整型能够存放－32768～32767数值范围，unsigned int型变量能够保存值的范围则为0～65535。

### 2.3.3 算术运算符、赋值算符和算术表达式

(1) 加减乘除运算符

在C语言中加减乘除运算符分别表示为＋－＊/。因为这些算符是针对两个操作数进行算术运算的，我们称为双目算术运算符。

在一个表达式中如果有多个运算符，那么相应于这些算符的计算过程是有先后的，这种先后次序称为它们的优先级。与一般算术运算相同，在C语言中，＊、/算符的优先级高于＋、－算符。

如果在一个表达式中出现多个优先级相同的算符，那么对表达式可能从左到右进行计算，也可能从右到左进行处理，这取决于算符的另一种特性，这种特性称之为结合性。+、-、*、/算符的结合性都是从左到右。

使用左、右圆括号组可以改变算符的处理顺序，例如： a+b*c/d

等效于

a+((b*c)/d)

而对 (a+b)*c

则先执行加法运算(a+b)，然后再执行乘法运算。括号可以嵌套，表达式将从最内层括号向外扩展逐层进行处理。

(2) 整数运算和一元负运算符

在算术表达式中，如果包含的都是整型变量或整型数，那么计算结果也一定取为整型数。这对于+、-、*算符是显而易见的，但对于除法运算则须略加说明。例如：

25/2*2

的结果不是25而是24。其原因是*、/算符的优先级相同，结合性都是自左向右，于是先计算25/2，其结果取整数(舍去小数部分)得12，再乘2得24。

在双目算术运算中，只要两个操作数中有一个是浮点型变量或用浮点形式表示的常数，则计算结果必取为浮点数。

单目运算符'-'的优先级高于其它算术运算符。使用了单目运算符'-'，则对变量的值取负。

(3) 取模运算符

取模运算符的符号是%，作用是取两个整数相除的余数，如25%5的结果为0，25%10的结果为5。模除的优先级与*、/算符相同，它不能用于单、双精度浮点数。

(4) 增、减1算符

C语言设置了增、减1算符，分别用符号++、--表示。例如：

```
x=n++;
```

相当于

```
x=n;
n=n+1;
```

而 `x=++n;`

相当于

```
n=n+1;
x=n;
```

这对于减1算符也是相同的。也就是说，如果增减1算符用在变量前，则先对该变量值进行增减1运算，然后该值参加表达式中其它运算或赋值操作；反之亦然。

注意：增减算符只能用于变量或可以赋值的表达式。增减1算符不能用于不可赋值的表达式，例如：

(i+j)++

是非法运算。在C语言中，可以赋值表达式称为左值(lvalue)。