高等院校教材

数据结构

C++语言描述

熊岳山　陈怀义　姚丹霖　编著

国防科技大学出版社

高等院校教材

数据结构

（C++语言描述）

熊岳山　陈怀义　姚丹霖　编著

国防科技大学出版社
·长沙·

内 容 简 介

数据结构是计算机专业的重要基础课，是该专业的核心课程之一，是一门集技术性、理论性和实践性于一体的课程。

本书重点介绍抽象数据类型、基本数据结构以及如何用 C++ 的类来描述抽象数据类型，进一步使读者理解面向对象的程序设计思想，提高用计算机解决实际问题的能力。内容包括：面向对象的程序设计语言——C++ 概述、基本数据类型、抽象数据类型、顺序表、链表、树和二叉树、图、稀疏矩阵、广义表、文件等结构以及在这些数据结构上的常用基本运算。

本书结构合理，内容丰富，算法描述清晰，便于自学。

本书可作为高等院校计算机专业和其他相关专业的教材和参考书，也可供从事计算机软件开发的科技工作者参考。

图书在版编目(CIP)数据

数据结构：C++ 语言描述/熊岳山，陈怀义，姚丹霖编著 .—长沙：国防科技大学出版社，2002.2

ISBN 7-81024-792-1

Ⅰ. 数… Ⅱ. ①熊… ②陈… ③姚… Ⅲ. ①数据结构②C 语言—程序设计 Ⅳ. TP311.12

中国版本图书馆 CIP 数据核字(2001)第 092316 号

国防科技大学出版社出版发行
电话：(0731)4572640 邮政编码：410073
E-mail：gfkdcbs@public.cs.hn.cn
责任编辑：黄 煜 责任校对：唐卫威
新华书店总店北京发行所经销
国防科技大学印刷厂印装

开本：787×1092 1/16 印张：18.5 字数：427 千
2002 年 2 月第 1 版第 1 次印刷 印数：1－5000 册

*

定价：25.00 元

前　言

数据结构是计算机科学与技术一级学科相关专业的重要基础课程之一，是软件开发和维护的基础。计算机的数据处理能力是计算机解决各种实际问题的关键。现实世界中的实际问题经过抽象，得出反映实际事物本质的数据表示后，才有可能被计算机处理。从实际问题抽象出数学模型，得出它的数据表示后，如何用计算机所能接受的形式来描述这些数据（包括数据本身与数据之间的关系），如何将这些数据以及它们之间的关系存储在计算机中，如何用有效的方法去处理这些数据，是数据结构研究的主要问题。

面向对象的程序设计（OOP）是当今流行的一种程序设计方法。它是人们以一种与传统的面向过程的程序设计方法完全不同的思维方式来认识软件设计的各个方面，并用于指导软件生产的整个过程。面向对象的程序设计有以下几个优点：首先，由于对象包含属性和行为，因此它支持模块化的程序设计，而模块化程序设计又便于程序的开发和维护；其次，在面向对象程序设计过程中可以充分地利用已有类库，在此基础上所做的修改与扩充不会损害原有类库，因此应用程序的新增代码明显减少，应用程序的可靠性得以提高。C++是一种支持面向对象的程序设计的语言。C++是C语言的超集，它不但实现了面向对象的程序设计的要求，而且还继承了C语言本身的所有优点，是开发系统软件和应用软件的有力工具，这就是C++倍受青睐的主要原因。

用面向对象的程序设计语言C++描述数据结构，与用面向过程的程序设计语言（C或PASCAL）描述数据结构相比，有很大的变化。用面向对象的C++语言的类来描述抽象数据类型，可利用封装、继承和多态等特性。

本书是在深入研究国内外同类教材的基础上，结合多年“数据结构”课程教学经验编写而成的。第一章是面向对象的程序设计与C++概述，主要是为已具有C语言基础、但对C++不太熟练的读者写的，熟悉C++的读者可跳过此章。这一章的内容包括面向对象的程序设计思想、对象与C++类的描述、继承性、多态性、虚函数和纯虚函数、generic类、函数模板和模板类、友元函数、引用等。第二章为数据结构概论，主要介绍数据结构概念：逻辑结构、存储方法、算法复杂度分析、基本数据类型、抽象数据类型与类。第三章介绍向量、栈和队列及其应用，内容有：向量、栈和队列的逻辑结构，抽象数据类型向量、栈和队列的描述；Josephus问题求解、栈与后缀表达式求值、栈与递归、队列与离散事件模拟等应用实例。第四章介绍链表及其应用，内容有：动态存储、单链表、循环表、双链表、栈和队列的链接存储。第五章介绍各种排序方法，内容有：排序的基本概念，被排序文件的存储表示，折半插入排序、Shell排序、起泡排序、快速排序、归并排序和外排序等各种排序方法，各种排序方法的时空效率，算法的实现细节和算法的时空效率等。第六章介绍了线性表的查找，内容有：有关查找的概念，顺序查找、折半查找、分块查找和散列查找。第七章介绍树和二叉树，内容有：树（树林）和二叉树的概念、树（树林）和二叉树的遍历、抽象数据类型BinaryTree与类BinaryTree、二叉树的遍历算法。第八章介绍树形结构的应用，内容

有：二叉排序树、平衡的二叉排序树、B－树和 B^+ －树、键树和 2－3 树、Huffman最优树、堆排序和判定树。第九章介绍图结构，内容有：图的基本概念、图的存储表示、Graph 类的构造与实现、图的遍历、最小代价生成树、单源最短路径问题、每一对顶点间的最短路径问题、有向无回路图。第十章为多维数组和广义表，内容有：多维数组的存储方法、稀疏矩阵的存储和抽象数据类型——稀疏矩阵；广义表的概念和存储。第十一章介绍文件结构，内容有：外部存储设备简介、有关文件的基本概念、顺序文件、索引文件、散列文件、倒排文件和多重表文件。

本书是作为计算机专业本科生“数据结构”课程教材编写的，也可供从事计算机软件开发和计算机应用的工程与科技人员参考。具备了 C 语言基础的读者便可学习本书，熟悉 C++ 的读者可跳过第一章学习。全书是国防科技大学计算机学院从事“数据结构”课程教学小组共同讨论定稿的，第一章、第五章的 5.6 节以及第七章至第十一章由熊岳山教授执笔，第二、三章由陈怀义教授执笔，第四、五(5.1～5.5 节)、六章由姚丹霖博士执笔。

书中不打“＊”的章节可用 60 学时的时间讲授，全部内容讲授可在 80～100 学时内完成。此外，为配合课堂教学，便于学生理解和掌握所学知识，提高程序设计编程能力，应另外配有 20～30 小时的上机时间。

本书的出版得到国防科技大学计算机学院、计算机系、603 教研室以及国防科技大学出版社的大力支持，特别是计算机系宁洪教授、殷建平教授自始至终关心本书的出版，给予了很大的帮助，在此深表谢意。由于时间仓促，加之作者水平有限，书中错误在所难免，敬请广大读者和专家批评指正。

作　者

2002 年 1 月

目　录

第一章　面向对象的程序设计与C++

1.1　什么是面向对象的程序设计

面向对象的程序设计(OOP)是当今流行的一种程序设计方法。它是人们以一种与传统的面向过程的程序设计方法完全不同的思维方式来认识软件设计的各个方面，并用于指导软件生产的整个过程。

众所周知，20世纪60年代诞生的面向过程的程序设计方法——结构化程序设计方法，是一种“自顶向下、逐步求精”的方法，其具体过程就是将一个问题划分为若干个子问题，再根据需要对子问题作进一步的划分，直到问题能够被编程实现。然而，随着实际问题复杂程度的增大，一旦项目达到一定的规模，这种结构设计的思想就变得不可控制，其原因是项目的复杂性超出了程序员使用结构化程序设计技术所能管理的限度。

面向对象的程序设计吸收了结构化程序设计的优点，并结合了一些新的、强有力的概念，从而开创了程序设计工作的新天地。面向对象的程序设计使用户能容易地把一个大问题分解成为多个与该问题有关的子组，然后，应用面向对象的程序设计语言就能把这些子组翻译为被称为对象的自包含单元。

1.2　面向对象的程序设计与C++

通常把C++称作“带有类的C语言”，它是1983年在美国新泽西州Murray Hill的贝尔实验室由Bjarne Stroustrup开发出来的。面向对象的程序设计中最重要的特性就是对象。简而言之，对象即为包含数据和对数据操作的代码的实体。在一个对象中，代码和(或)数据可以是这个对象私有的，不能被对象外的部分直接访问，这种代码和数据的联系称为“封装”(encapsulation)。在程序设计的过程中，面向对象的程序设计把重点放在对象之间的关系上，而不是在其实现的细节上。对象之间的联系是通过它们之间相互的类属(或称层次)关系确定的。面向对象的程序设计当然也就包括对象，而对象又是由它的属性和行为组成的。属性不能从对象的外部直接存取，只能由对象的行为来操纵。

面向对象的程序设计有几个优点。首先，由于对象包含属性和行为，因此它支持模块化的程序设计，而模块化程序设计又便于程序的开发和维护。其次，面向对象程序设计过程中可以充分地利用已有的类库，在此基础上所做的修改与扩充不会损害原有的类库，因此一个应用程序的新增代码明显减少，应用程序的可靠性有所提高。

C++是一种支持面向对象的程序设计语言。由于C++是C语言的超集，所以它不

但实现了面向对象的程序设计的要求，而且还继承了 C 语言本身的所有优点，这就是 C++ 倍受青睐的主要原因，是开发系统软件和应用软件的有力工具。

1.3 C++ 类与对象设计

在 C++ 中创建一个对象时先要用关键字 class 定义，class 说明的一般形式是：

```
class Class_name
{
  //这些数据和方法不能被直接访问
  private:
    data and functions
  //这些数据和方法只能被派生类的对象访问
  protected:
    data and functions
  //这些数据和方法可以直接访问
  public:
    data and functions
}object list;
```

这里 object list 可以为空。显而易见，一个类包含 public、private、protected 三类元素，在缺省的情况下，类中定义所有的项均为私有的。创建一个对象也可用下面的方式：

```
Class_name object_name;
```

下面给出创建 Location 类的 C++ 源代码，用来表示屏幕上的一个坐标位置(X, Y)。

```
class Location
{
  protected:
    int X;      //横坐标
    int Y;      //纵坐标
  public:
    Location(int InitX, InitY);      //构造函数，对坐标进行初始化
    ~Location(void);                 //析构函数(这里可以不需要析构函数)
    int GetX(void);                  //成员函数，取当前横坐标
    int GetY(void);                  //成员函数，取当前纵坐标
};
```

构造函数(constructor)和析构函数(destructor)的名与类名相同，不同的是析构函数前有“~”，在对象创立时对象的构造函数即被调用；而析构函数是构造函数的逆函数。对于构造函数有如下要求：

- 一个类的构造函数的名称必须与类相同。

- 一个构造函数不可以确定一个返回类型。编译器有其自身的构造函数返回类型。
- 当用户编制了多个构造函数说明时，各参数表之间应彼此互不相同。

而对于析构函数有：

- 类的析构函数的名称必须与类相同，不同的是前面必须加上字符“～”。
- 一个析构函数不可以确定一个返回类型。编译器自身也有特定的适合析构函数的返回类型。
- 一个析构函数不接受参数。没有参数表，重载是不可能的。

此外，编译器可以在缺省条件下自动生成构造函数和析构函数。编译器生成的构造函数和析构函数具有 public 属性。

前面定义的 Location 类的构造函数、析构函数和成员函数的 C++ 代码为：

```
Location::Location(int InitX, int InitY)
{
  X=InitX;
  Y=InitY;
}
Location::~Location(void)
{
  cout<<"This destructor is not required.";
}
Location::GetX(void)
{
  return X;      //返回当前点的坐标 X
}
Location::GetY (void)
{
  return Y;      //返回当前点的坐标 Y
}
```

1.4 继承性

继承是一个对象获得另一个对象的过程。C++ 的类具有继承性(inheritance)。被别的类继承的类称为基类，有时也称为父类；执行继承的类则称为派生类，也称为子类。继承的一般形式为：

```
class class_name1: access class_name0
{
  ...
};     //类 class_name1 为类 class_name0 的派生类
```

这里，access 为 public、private 或 protected，也可以缺省（当基类为一结构时，其缺省值为 public；当基类为类时，其缺省值为 private）；基类的 private 成员不能传递给继承者。

• access 为 public

基类的 public 成员成为派生类的 public 成员，基类的 protected 成员成为派生类的 private 成员。

• access 为 protected

基类的 public 成员成为派生类的 protected 成员，基类的 protected 成员成为派生类的 private 成员。

• access 为 private

基类的 public 和 protected 成员成为派生类的 private 成员。

此外，一个类可以继承两个或两个以上类的属性。要实现这一点，只需在派生类的（base_class list）中用逗号把多个基类分开即可，一般形式为：

```
class derived_class_name:base_class list
{
  ...
};
```

在派生类的构造中，只要基类的构造函数没有参数则派生类就不必有构造函数。然而，当基类的构造函数使用一个或多个参数时，任何派生类也必须包含构造函数。其原因在于这样就提供了一种把参数传递给基类构造函数的途径。

以下是一个关于 C++ 实现继承的例子，派生类 Point 继承了基类 Location.

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
class Location
{
  protected:
    int X;
    int Y;
  public:
    Location(int InitX, int InitY);
    int GetX( );
    int GetY( );
};

class Point: public Location
{
   public:
    Point(int InitX, int InitY);
```

```
    //show this point on the screen
      void Show(int pixelcolor);
};

//Member functions
Location::Location(int InitX, int InitY)
{
  X=InitX;
  Y=InitY;
}
int Location::GetX(void)
{
  return X;
}
int Location::GetY(void)
{
  return Y;
}
Point::Point(int InitX, int InitY):Location(InitX, InitY)
{}
void Point::Show(int pixelcolor=15)
{
  int j;
  j=pixelcolor;
  putpixel(X, Y, j);
}
main(void)
{
  int GraphDriver=DETECT, GraphMode;
  Point b(40,20);
  Point c(100,200);
  initgraph(&GraphDriver, &GraphMode,"c:\\borlandc\\bgi");
  b.Show( );
  c.Show( );
  getch( );
  cout<<"GetX( )="<<b.GetX( )<<"\n";
  cout<<"GetY( )="<<b.GetY( )<<"\n";
  cout<<"GetX( )="<<c.GetX( )<<"\n";
```

```
    cout<<"GetY( )="<<c.GetY( )<<"\n";
    closegraph( );
    return 0;
}
```

1.5 多态性、虚函数和纯虚函数

面向对象的程序设计中最重要的一个概念就是多态性(polymorphism)。在C++中,用一个名称对应功能相近的不同函数,这就是多态性。C++支持编译时的(或静态,或操作重载)多态性和运行时的(或动态,或虚函数)多态性。因此,多态性有时又称为"同一接口,多种方法"。操作符重载和函数重载就是编译时的多态性。虽然操作符重载和函数重载的功能很强,但它们仍不能完成一个真正的面向对象语言所需完成的全部任务。因此,C++也允许通过使用派生类和虚函数等来支持运行时的多态性。下面的例子说明了这种多态性的用法。

```
#include <iostream.h>
class Figure
{
  protected:
    double x, y;
  public:
    void set_dim(double i, double j=0)
    {
      x=i;
      y=j;
    }
    virtual void show_area(void)
    {
      cout<<"基类无面积定义\n";
    }
};
class Triangle:public Figure
{
  public:
    void show_area(void)
    {
      cout<<"三角形的高为";
      cout<<x<<",底边长为"<<y;
```

```
        cout<<", 面积=";
        cout<<x*0.5*y<<"\n";
    }
};
class Square:public Figure
{
  public:
    void show_area(void)
    {
        cout<<"长方形的边长分别为";
        cout<<x<<"和"<<y;
        cout<<",面积=";
        cout<<x*y<<"\n";
    }
};
class Circle:public Figure
{
  public:
    void show_area(void)
    {
        cout<<"圆的半径为";
        cout<<x;
        cout<<", 面积=";
        cout<<3.14*x*x;
    }
};
main(void)
{
  Figure *p;          // 建立指向基类的指针
  Triangle t;         // 创建对象 t
  Square s;           // 创建对象 s
  Circle c;           //创建对象 c
  p=&t;               //p 指向 Triangle 的对象
  p->set_dim(10.0,5.0);
  p->show_area( );

  p=&s;
  p->set_dim(10.0,5.0);
```