最新出版

最新系列软件入门教程

Visual C++
for Windows
程序设计基础

● 马希荣 孙华志 魏新俊 等 编著 ● 方裕 徐国平 审校

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

最新系列软件入门教程

Visual C++ for Windows 程序设计基础

马希荣　孙华志　魏新俊　等编著

電子工業出版社

内 容 提 要

Visual C++ +1.5 是 Microsoft 公司新近推出的面向对象程序设计语言，使用 Visual C++ + 进行程序设计大大提高了编程效率。本书全面、系统地介绍了使用 Visual C++ + 进行程序设计的基本方法，以及如何使用 MFC 库来构造复杂的应用程序。全书分五部分，共二十七章。第一部分介绍面向对象技术的基础知识；第二部分对 Visual C++ + 语言进行全面介绍，重点强调了 C++ + 中不同于普通 C 语言的方面；第三部分讨论 Windows 编程主要涉及到的概念；第四部分完整地描述使用 Visual C++ +1.5 提供的集成开发环境及 MFC 类库进行程序设计的方法；在第五部分中介绍快速创建 Windows 核心源代码的三个实用程序。

本书内容丰富、结构合理，对使用 Visual C++ + 从事设计、开发 Windows 应用程序的初学者及计算机应用技术人员有重要的参考价值，同时还可作为一本系统、实用的教材供大专院校及各类培训班使用。

与本书配套有 5 英寸演示软盘一张，供读者学习使用。

欢迎提出宝贵意见，来信请寄：中国 UNIX 协会培训中心（北京学院路 31 号，100083）Tel：(010)2042233—322，2054733 Fax：(010)2024674。

最新系列软件入门教程

Visual C++ + for Windows 程序设计基础

马希荣　孙华志　魏新俊　等编著

责任编辑：王明君　毛兆余

*

电子工业出版社出版

北京市海淀区万寿路 173 信箱（100036）

电子工业出版社发行　各地新华书店经销

电子工业出版社计算机排版室　排版

北京市牛山世兴印刷厂印刷

*

开本：787 × 1092 毫米 1/16　印张：36.75　字数：940 千字

1996 年 3 月第一版　1996 年 3 月第一次印刷

印数：5000 册　　定价 52.00 元

ISBN7-5053-3177-9/TP · 1154

前　言

九十年代面向对象程序设计技术的兴起，在全球软件业掀起了一场不小的风波，各公司先后推出了自己的支持面向对象的程序设计语言。可以说，面向对象的程序设计技术是九十年代软件开发的最新潮流。在众多的面向对象程序设计语言中，Visual C++以其将程序设计方法同可视的软件开发环境完美的结合得以脱颖而出，它一出现便受到广大软件设计人员的青睐，纷纷将其作为设计、开发 Windows 应用程序的首选语言。具体说来，Visual C++1.5 具有以下特性：

·通过可视的集成开发环境，将编辑器、编译器、连接器、调试器以及各种实用工具完美地结合在一起，大大降低了程序开发的复杂程度。

·提供完备的 MFC 类库，在类库中封装了几乎所有的 Windows API 函数，将面向对象与事件驱动两种编程规范结合在一起，充分显示了两种编程方式共同工作的优势。

·提供了三个快速、自动创建 Windows 应用程序源代码的实用工具，使得编程工作更加快捷、简单。

全书共分五部分。

第一部分包括第一章至第三章。介绍了面向对象技术的基础知识，比较了传统结构化程序设计方法与新的面向对象程序设计方法的差别。

第二部分包括第四章至第十章。对 Visual C++语言进行了全面、系统地介绍。重点强调了 C++不同于 C 的方面，即对 C 的面向对象及非面向对象方面的扩充。

第三部分包括第十一章至第十五章。介绍了 Windows 编程涉及到的概念，简述了 Windows 应用程序的所有基本组成成分：WinMain 函数、消息循环、窗口过程、资源等，并以实例说明了 DLL 的定义及使用。

第四部分包括第十六章至第二十二章。介绍了 Visual C++集成开发环境及 MFC 类库，以及如何利用系统提供的工具、类库进行 Windows 程序设计。强调了用 MFC 编写 Windows 应用程序的一般原则，并用实例说明了 MFC 中实现 Windows 控制、窗口、菜单、对话框、MDI、OLE 等功能的类及其使用方法。

第五部分包括第二十三章至第二十七章。介绍了快速创建基于 MFC 类库的 Windows 应用程序源代码的三个实用工具，并在最后一章给出了一 OLE 综合实例。

参加本书编写工作的有：孙华志（第一章至第三章、第十五章至第十八章）、张琳（第四章、第六章）、梁琪（第五章、第七章、第八章）、张立新（第九章、第十章）、马希荣（第十一章至第十四章、第十九章、第二十章、第二十二章）、魏新俊（第二十一章、第二十六章、第二十七章）、王慧芳（第二十三章至第二十五章）。全书由马希荣统编，由方裕、徐国平审校。

在本书编著的过程中，得到了北京大学、南开大学机器智能研究所、天津师范大学计算机系及中国 UNIX 协会培训中心众多老师的热情支持和帮助，在此一并致谢。

由于时间仓促，作者水平有限，书中难免有不妥之处，恳请读者批评指正。

编　者

1995 年 4 月

目　录

第一部分　面向对象技术

第二部分　Visual C++程序设计语言

第三部分 Windows 编程基础

第四部分
Visual C++ +1.5 集成开发环境与 MFC 类库在 Windows 编程中的应用

第五部分 快速创建 Windows 应用程序的工具

附　录

第一部分
面向对象技术

第一章 概述

本章给出面向对象程序设计的总体描述，在此并不深入讨论具体的程序设计语言，而是着眼于对面向对象技术的理解。本章主要讨论：

·传统的结构化程序设计方法存在的问题

·什么是面向对象程序设计

·面向对象程序设计(OOP)方法与传统的结构化程序设计(SP)方法之间的主要区别

1.1 传统的结构化程序设计方法回顾

在过去近三十年的时间里，程序开发几乎全部采用结构化程序设计(Structured Programming - SP)模式。使用 SP 方法构造程序时，首先把精力集中在为解决实际应用而实施的算法的结构上，在确定了算法之后，为了抽象算法在计算机上的具体实现，应为该算法选择或构造适当的数据结构，通过对数据的操纵过程体现算法的思想。也就是说，程序是在数据的某种特定表示方式和结构的基础上，对抽象算法的具体描述。

采用 SP 方法编制的程序，其结构均可由图 1.1 概括：

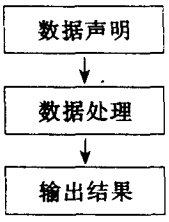

图 1.1 SP 方式程序结构

对于较复杂的数据处理过程，通常的做法是分解处理的复杂性，即采用的是自顶向下，分而治之的手段，将其按功能划分为几个可独立编程的子过程模块，每一子模块完成指定子任务，并且提供一个清晰、严格的调用界面，主过程通过调用各子过程完成全部处理工作。这种程序设计方法力求算法描述准确；对每一子模块容易进行程序正确性证明；对数据进行的编译检查在一定程度上增强了程序的可靠性，从历史上看，这一切确实给软件设计带来了巨大发展。但是，由于这种程序设计方法从本质上是面向“过程”的，而“过程”和“操作”又是不稳定的、多变的，不能直接反映人类求解问题的思路，因此这种模式存在固有缺陷，不可能从根本上解决软件危机。以过程为中心构造系统，设计应用程序时，随着软件工作量的增加，人们势必感到这种程序设计模式存在着局限性。

首先，程序可重用性差。在程序复杂性增加时，代码重用是提高生产率的关键。而采用传统的面向过程模式，每次进行软件开发，系统中除了一些接口十分简单的标准数学函数库外，几乎一无所有。程序员总是从零做起，并且要针对具体问题做大量的重复又繁琐的工作，即使重用代码，通常也是通过拷贝或编辑重新生成一份。也就是说这种模式不能对已编制的部分应用程序直接继承引用，或通过仅编写与已存在代码的相异之处，生成新的应用程

序。

其次，维护程序一致性困难。面向过程模式将数据与过程分离很可能产生问题空间与方法空间在结构上的不一致，这对程序开发者来说是一个很大的障碍。用 SP 方法构造程序，对程序运行起重要作用的数据一般要做为全局数据处理，若为了适应新的需求，对某一数据结构做了修改，那么，所有处理数据的过程都需要重新考察，以保证与数据的一致性。维护数据与过程的协调一致必然花费大量的资源，同时也使产生错误的机会大大增加。

1.2 新的面向对象的程序设计方法

为克服 SP 方法在编制大型软件工程项目和系统设计时所存在的弊端，面向对象程序设计(Object - Oriented Programming——OOP)模式应运而生。它的兴起，打破了传统过程模式的束缚，推动了软件开发的进程。

OOP 方法总的设计原则是：按人们通常的思维方式建立问题域的模型，设计出尽可能自然地表现求解方法的软件。根据这一原则，将设计目标从模拟现实世界的行为转向了模拟现实世界中存在的对象及其各自的行为。在 OOP 中，将“对象”做为系统中最基本的运行实体，“对象”中封装了描述该对象的特殊属性(数据)和行为方式(方法)。整个应用程序即由各种不同类型的对象组成，各对象既是一独立的实体，又可通过消息(即让对象以某种方式进行操作的请求)相互作用，对象中的方法决定要向哪个对象发消息、发什么消息、以及收到消息时如何进行处理。对象的内部结构如图 1.2 所示。

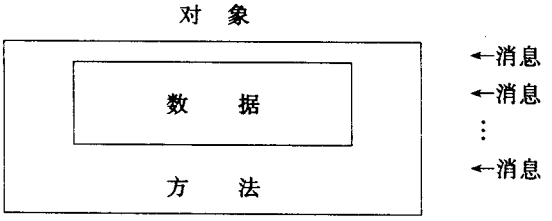

图 1.2 对象的内部结构

也可以这样说，OOP 是以“对象”或“数据”为中心的，这时的数据与传统的被动数据不同，它具有“行动”的功能，而这种动作是在对象接收了消息时发生的。由于对象自然地反映了应用领域的模块性，因此具有相对稳定性，可以被用作一个组件，去构成更复杂的应用；又由于对象一般封装的是某一实际需求的各种成分，因此某一对象的改变，对整个系统几乎没有影响。

为了描述功能上十分相近的对象，引入了类(class)。类与类以层次结构组织，属于某个类的对象除具有该类所描述的特性外，还具有层次结构中该类上层所有类描述的全部性质，OOP 中称这种机制为继承。

OOP 方法的模块性与继承性，保证了新的应用程序设计可在原有对象的数据类型和功能的基础上，通过重用、扩展和细化来进行，而不必从头做起或复制原有代码，这样，大大减少了重新编写新代码的工作量，同时降低了程序设计过程中出错的可能性，达到了事半功倍的效果。

1.3 SP方法与OOP方法程序结构比较

尽管面向对象程序设计方法中不以过程抽象为主要特征形式,但传统的SP方法确实在面向对象分析及描述对象方面发挥了作用。新的面向对象技术并没有取代或抛弃结构化设计思想,应用程序的设计者仍需用SP方法来设计软件对象的具体实现。OOP方法主要用于软件设计大的方面:如怎样组织程序代码、程序代码怎样被重用与扩展。因此,OOP方法除了具有SP的一切优点和机制外,同时,由于它引入了若干强有力的、更能反映事物本质的新概念、新机制,从而给软件设计领域带来了深刻的变革。

下面分别使用C及C++语言给出两个实例,说明SP方法与OOP方法在设计原则和程序结构方面的区别,两个程序都完成对雇员记录的管理。

例1.1 使用传统的SP方法修改雇员记录的程序代码

```
# include <stdio.h>
Struct Employee{
  char Name[10];
  int Age;
  }Employees[20];          //用结构数组定义雇员记录
Void AddEmp(int n);        //增加雇员记录函数
Void DelEmp(int n);        //删除雇员记录函数
Void UpdEmp(int n);        //修改雇员记录函数
      ⋮
int main()
{char f_name;              //操作名,可能取值为'A','D','U','E',
                           //分别代表加、删、改、退出操作
 int i;                    //雇员记录
 do{                       //提示用户输入雇员序号
    printf("Please input employee number(-1~20),-1 for end loop:");
    Scanf("%d",&i);
    if(i>=0)
      {                    //提示用户输入对该记录进行什么操作
      printf("please input operation name (A,D,U,E):");
      scanf("%c",&f_name);
      switch(f_name)       //根据用户输入决定使用什么函数处理该记录
         {case 'A':AddEmp(i);break;
          case 'D': DelEmp(i);break;
          case 'U': UpdEmp(i);break;
          case 'E': return(0);
          default:printf("error\ \n");
         }//switch end
```

```
    }//if end
  }while(i = = -1)
}
```

下面的图 1.3 给出了该程序的粗略结构：

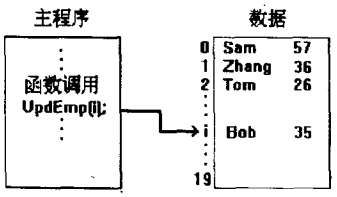

图 1.3　修改雇员记录传统的 SP 方法

显而易见，对于这样的程序结构，数据与处理数据的过程是相互分离的。通过在主程序中的函数调用语句，完成对某一数据记录的操作。若数据有变化，比如，数据结构由原来的数组改为链表，那么所有与此数据有关的函数都要重写，程序改动量很大；尤其当不仅需要处理雇员记录还要处理部门经理记录时(假设增加了本部门人数字段)，尽管只需要增加一个描述字段，可是原数据结构和函数都要重新定义，并且新加代码与原已存在代码有许多相似之处。

例 1.2　使用新的 OOP 方法修改雇员记录的程序代码：

```
# include <stdio.h>
class Emp{
  protected:
     char Name[10];
     int Age;
  public:
     Virtual AddEmp(){…};
     Virtual DelEmp(){…};
     Virtual UpdEmp(){…};
  };
int main()
   {char f_name;
    int i;
    Emp Employees[20];  //数据 Employees 的每个元素为一有自动处理能力的类
    do{//提示用户输入雇员序号
       printf("please input employee number(-1~19):-1 for end loop");
       scanf ("%d",&i);
       if (i>=0)
         {//提示用户输入对该记录进行什么操作
          printf("please input operation name(A,D,U,or E):");
```

```
        scanf ("%c",&f_name);
        switch (f_name)
         {case 'A': Employees[i].AddEmp();break;
          case 'D': Employees[i].DelEmp();break;
          case 'U': Employees[i].UpdEmp();break;
          case 'E': return(0);
          default: printf("error \ \n");
         }//Switch end
        }//if end
    }while (i==-1)
  }
```

下面的图 1.4 给出了该程序的粗略结构：

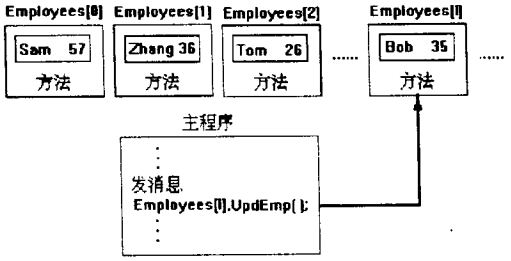

图 1.4 修改雇员记录的 OOP 方法

OOP 程序结构中的对象将数据与处理数据的方法封装起来，不仅提供了信息隐蔽，而且也为今后代码重用奠定了基础。主程序将消息（例中 UpdEmp()）传递给对象（例中是 Employees[i]），对象在收到消息后使用自带的方法进行数据处理。若例中组织雇员数据记录的数据结构有变化，比如，对象由数组结构改变为链表结构，则对象内的所有方法不必重新定义，只要在主程序中做一些相应的改动即可。OOP 方法还有一个重要的特点就是继承。比如，若使程序增加处理部门经理记录的能力，则原代码不必修改，只需声明一个类 Employee 的子类 Manager，在 Manager 中不必复制 Employee 的代码，便可继承了它所有的数据结构与方法，当然 Manager 与 Employee 不可能完全相同，那么仅在 Manager 类中定义与 Employee 的相异之处就可自动完成所期望的 Manager 的定义。

例 1.3 C++中类 Manager：

```
class Manager: Employee
  {
   int Num_of_emp;              //新数据
   public:
      virtual void AddEmp();
      virtual void UpdEmp();   //两个需重新定义的方法
  };
```