第十一章　C＋＋程序设计基础

【教学目标】

理解面向对象的概念，理解面向对象程序设计的思想，理解面向对象程序设计的基本方法。

理解抽象、封装、继承、多态等概念。

理解类与对象的概念，掌握C＋＋程序设计中类与对象的定义方法。

理解基类、派生类概念、了解派生类的定义方法。

理解多态性与虚拟函数的概念，了解虚拟函数的定义方法。

掌握C＋＋程序结构，掌握输入／输出流使用方法。

【教学内容】

面向对象的概念，面向对象程序设计的思想及基本方法。

类、对象的概念及定义；构造函数、析构函数的定义方法。

函数重载的概念、方法。

基类与派生类的概念、派生类定义方法。

多态性、虚拟函数概念、虚拟函数定义方法。

C＋＋程序结构，输入／输出流概念及使用方法，引用的概念及使用方法。

【重点和难点】

重点

理解面向过程的程序设计与面向对象的程序设计的差异与联系。

理解面向对象程序设计的概念、特点：抽象、封装、继承、多态等。

掌握类、对象的概念及定义方法。

理解函数重载、基类、派生类、虚拟函数的概念及定义方法。

难点

对象的概念、特点：抽象、封装、继承、多态。

类成员：公有成员、私有成员、成员函数的概念及定义方法。

构造函数、析构函数的概念、作用及定义方法。

【问题的提出】

【面向对象的思想】

计算机技术的发展用日新月异来描述是十分贴切的，其中软件的升级和更新更使人目不暇接。因此，一个好的软件，只有具有很好的可扩充性和可重用性，才能够在激烈的竞争中得以不断发展、完善、生存。实际上，从计算机诞生之日起，人们就一直在追求这样的目标，发展能够提高可扩充性和可重用性的程序设计方法。

面向对象的程序设计（object-oriented programming，简称 OOP）正是这一发展现在所达到的高度，它通过抽象、封装、继承等一系列手段，达到程序源代码最大限度的可重用性和可扩展性，提高程序员的生产能力，控制软件开发和维护的费用。在面向对象程序设计方法

之前，广泛使用的是面向过程的结构化程序设计方法。

【结构化程序设计】

结构化程序设计诞生于六十年代，发展到八十年代，已经成为当时程序设计的主流方法，几乎为所有的程序员接受和使用，它的产生和发展形成了现代软件工程的基础。其基本思想是采用自顶向下、逐步求精的方法和单入口单出口的控制结构。自顶向下，逐步求精的方法使所要解决的问题逐步细化，并最终实现由顺序、选择和循环这三种基本结构构成的描述。

结构化程序设计方法将解决问题的重点放在如何实现过程的细节方面，而把数据和对数据进行操作的函数截然分开，以数据结构为核心，围绕着功能实现或操作流程来设计程序。这样设计出来的程序，其基本形式是主模块与若干子模块的组合，即一个主函数（main 函数）和若干子函数。程序以函数为单位，函数之间以数据作为连接的纽带。正是由于把数据和操作分开对待，因而有着方法本身无法克服的缺点。由于数据和操作代码（函数）的分离，一旦数据的格式或结构发生变化，相应的操作函数就得改写，而且对于核心数据的访问往往也得不到有效控制。同时，如果程序进行扩充或升级改进，也需要大量修改函数。这样，程序开发的效率就难以提高，大大限制了软件产业的发展。

【面向对象的程序设计】

面向对象的程序设计是对结构化程序设计的继承和发展，它吸收了结构化程序设计的全部优点，同时又考虑到现实世界与计算机空间的关系，认为现实世界是由一系列彼此相关并且能够相互通信的实体组成，这些实体就是面向对象方法中的对象，而一些对象的共性的抽象描述，就是面向对象的核心——类。面向对象的程序设计方法就是运用面向对象的观点来描述现实问题，然后用计算机语言来描述并处理该问题。这种描述和处理是通过类与对象实现的，是对现实问题的高度概括、分类和抽象。每个对象都有自己的数据和相应的处理函数，整个程序是由一系列相互作用的对象来组成，对象之间是通过发送消息来实现。这样的程序，数据和操作很好地封装在对象中，数据的访问权限也可以得到有效的控制，数据结构和格式的改变只是局限于拥有该数据的对象和类中，因此修改也较为方便。同时，通过由现有的类进行派生，可以得到更多的具有特殊功能的新类，对更为复杂的问题进行描述，达到对现有源代码的最大限度的重用，实现软件设计的产业化。

从人类认识问题的角度来讲，结构化程序的设计方法假定我们在开始程序设计的时候，就对所要解决的问题有深入、彻底的认识，对程序有全面的规划。事实上，人类对问题的认识有一个逐步深入的过程，在程序设计之初，可能对问题还没有彻底的认识。因此，在程序的编写，甚至运行和测试过程中，如果我们有了新的认识和需要，要对其中的关键数据或算法有所改动，有可能整个程序都会修改。而我们知道，这种认识的过程是完全合符现实的，是必然的，也是结构化程序设计无法克服的困难在哲学高度的解释。

【教学要点】

1. 了解并掌握Ｃ＋＋程序程结构,学习输入、输出流使用方法

【例 11.1】由键盘输入两个整数，调用函数，输出最大值。

程序

```
#include<iostream.h>                    //输入、输流出所在头文件
void main()
{   int x,y,z;
    int max(int,int);                   //函数声明
    cout<<"请输入两个整数："; //屏幕提示
    cin>>x>>y;                          //输入变量值
    z=max(x,y);                         //函数调用
```

```
    cout<<"max="<<z<<'\n';       //输出变量 z 的值
}
int max(int x,int y)              //函数定义
{   return(x>y?x:y);              //返回函数值
}
```

2. 引用的概念及使用方法，掌握C＋＋程序结构

【例 11.2】使用“引用”作函数形参，交换两变量的值。

程序

```
#include<iostream.h>
void swap(int &x,int &y)
{   int x1;
    x1=x;x=y;y=x1;
    cout<<"     swap;x="<<x<<"     y="<<y<<endl;
}
void main()
{   int x=10,y=20;
    swap(x,y);
    cout<<"     main;x="<<x<<"     y="<<y<<endl;
}
```

3. 掌握类、对象的定义格式、方法

【例 11.3】编程，输出年、月、日，判定该年是否是润年。

程序

```
#include <iostream.h>
class TDate   //定义类
{
public:             //说明公有成员（可包含数据成员和成员函数）
    void SetDate(int y,int m, int d);          //说明公有成员函数
    int   IsLeapYear();      //说明公有成员函数
    void Print();              //说明公有成员函数
private:   //说明私有成员（可包含数据成员和成员函数）
    int   year,month,day;   //数据成员（用户不能访问）
};
//成员函数的实现（定义）
void TDate::SetDate(int y,int m,int d)   //定义成员函数（:: 作用域运算符）
{
    year=y;
    month=m;
    day=d;
}
int   TDate::IsLeapYear()                  //定义成员函数
{
     return(year%4==0&&year%100!=0)||(year%400==0);
}
```

```
void TDate::Print()                    //定义成员函数
{
    cout<<"          "<<year<<","<<month<<","<<day<<endl;cout<<endl;
}
void main()
{
    cout<<endl;cout<<endl;cout<<endl;
    TDate date1,date2;                  //定义类的对象（类似于定义变量）
    date1.SetDate(1996,5,4);            //用对象引用公有成员函数（ . 成员选择符）
    date2.SetDate(1998,4,5);            //用对象引用公有成员函数

    int leap=date1.IsLeapYear();        //用对象引用公有成员函数
    cout<<"          "<<leap<<endl;cout<<endl;
    date1.Print();                      //用对象引用公有成员函数
    date2.Print();                      //用对象引用公有成员函数
    cout<<endl;cout<<endl;cout<<endl;
}
```

4. 理解并掌握构造函数、析构函数的概念及定义方法。

【例 11.4】在头（.h）文件中定义类，在类中定义构造函数、析构函数；在主函数（.cpp 文件）中实现年、月、日的输出。

程序（11-4.h　11-4.cpp）

```
class TDate1                    //定义类
{           //边界
public:             //处部接口，公有成员
    TDate1(int y,int m, int d); //说明构造函数
    ~TDate1();          //说明析构函数
    void Print();           //说明成员函数
private:        //私有数据成员
    int year,month,day;         //
};                  //边界
TDate1::TDate1(int y,int m,int d)//定义构造函数
{
    year=y;
    month=m;
    day=d;
    cout<<"          Constructor（构造函数） called."<<"\n";cout<<endl;
}
TDate1::~TDate1()           //定义析构函数
{
    cout<<"          Destructor（析构函数） called."<<"\n";cout<<endl;
}
void TDate1::Print()    //定义成员函数
{
    cout<<"          "<<year<<"."<<month<<"."<<day<<endl;cout<<endl;
```

```
}
#include <iostream.h>
#include <11-4.h> //包含头文件
void main()
{
    cout<<endl;cout<<endl;cout<<endl;
    TDate1 today(1998,4,9),tomorrow(1998,4,10);//定义对象，并初始化(调用构造函数）
    cout<<"          today is :";
    today.Print(); //通过对象调用成员函数
    cout<<"          tomorrow is ";
    tomorrow.Print(); //通过对象调用成员函数
    cout<<endl;cout<<endl;cout<<endl;
}
```

5. 函数重载实现方法。

【例 11.5】求两个操作数之和（参数类型上不同的函数重载）

程序

```
#include <iostream.h>
int add(int,int);                    //函数说明
double add(double,double);   //函数说明
void main ()
{
    cout<<endl;cout<<endl;cout<<endl;
    cout<<"add(5,10)="<<add(5,10)<<endl;cout<<endl;//调用函数
        cout<<"and(5.1,10.1)="<<add(5.1,10.1)<<endl;cout<<endl;//调用函数
}
int add(int x,int y)//定义函数
{
    return x+y;
}
double add(double a,double b)//定义函数
{
    return a+b;
}
```

6. 类成员函数重载应用。

【例 11.6】求两个整数或实数中的最大者。

程序（10-6.cpp）

```
#include<iostream.h>
class max_class
{
public:
          int max(int a,int b);   //重载函数
         float max(float a,float b);
```

```
};
int max_class::max(int a,int b)     //类重载函数定义
{   if(a>b)   return a;
    else      return b;
}
float max_class::max(float a,float b)//类重载函数定义
{   if(a>b)   return a;
    else      return b;
}
void main()
{   max_class outmax;
    int x,y;
    float a,b;
    cout<<"Enter int x y    :";
    cin>>x>>y;
    cout<<"Enter float a    b: ";
    cin>>a>>b;
    cout<<"max(x,y)="<<outmax.max(x,y)<<endl;
    cout<<"max(a,b)="<<outmax.max(a,b)<<endl;
}
```

7. 派生类定义方法

【例 11.7】定义派生类实例。

程序

```
#include<iostream.h>
#include<string.h>
class person
{   private:
            char name[10],sex[3];int age;
    public:
            void init(char *str1,char *str2,int k)   //定义初始成员函数
            {   strcpy(name,str1);
                strcpy(sex,str2);
                age=k;
            }
            void disp()                              //定义输出成员函数
            {   cout<<"name:"<<name<<endl;
                cout<<"sex:"<<sex<<endl;
                cout<<"age:"<<age<<endl;
            }
};
class student:public person                          //派生类定义
{   private:
            char sclass[11];int num;float avg;
public:
```

```
        void init_stud(char *str,int i,float j)
        {   strcpy(sclass,str);
            num=i;
            avg=j;
        }
        void disp_stud()
        {   cout<<"class:"<<sclass<<endl;
            cout<<"num:"<<num<<endl;
            cout<<"avg:"<<avg<<endl;
        }
};

void main()
{   student stud;
    stud.init("李小刚","男",19);
    stud.disp();
    stud.init_stud("20033070101",28,29);
    stud.disp_stud();
}
```

8. 虚拟函数定义方法、调用实例。

【例 11.8】通过基类指针调用虚拟函数。

```
#include<iostream.h>
#include<string.h>
class person
{   protected:
        char name[10],sex[3];int age;
    public:
        void init1(char *str1,char *str2,int k)   //定义初始成员函数
        {   strcpy(name,str1);
            strcpy(sex,str2);
            age=k;
        }
        virtual void disp()                       //定义输出成员函数
        {   cout<<"name:"<<name<<endl;
            cout<<"sex:"<<sex<<endl;
            cout<<"age:"<<age<<endl;
        }
        ~person(){}
};
class student:public person                       //派生类定义
{   protected:
        char sclass[11];int num;float avg;
    public:
        void init2(char *str,int i,float j)
```

```
        {   strcpy(sclass,str);
            num=i;
            avg=j;
        }
        void disp()
        {   cout<<"class:"<<sclass<<endl;
            cout<<"num:"<<num<<endl;
            cout<<"avg:"<<avg<<endl;
        }
        ~student(){}
};

void main()
{   person A,*p;
    student B;
    A.init1("李小刚","男",19);
    p=&A;
    p->disp();
    B.init2("20033070101",28,29);
    p=&B;
    p->disp();
}
```

小结：

本章为一分界，承上（面向过程程序设计）启下（面向对象程序设计）；学习本章内容，重在对基本概念、基本方法的理解；读者不必强求通过本章学习，便能掌握面向对象程序设计的方法；若读者能够深入领会本章的基本概念与方法，可为今后进一步学习打下良好基础。