程序开发基础

2.1 C++语法基础

在这一节中，将介绍C++的特点和基本功能。主要包括：

C++中的语法、类、对象、类的继承和多态性以及C++中的输入 / 输出流。

2.1.1 C++程序的构成

一个标准的C++程序由三个部分构成：

- 预处理命令

  预处理命令位于行首，以符号＃开始，C++提供的预处理命令有宏定义命令、文件包含命令和条件编译命令三种。

- 函数

  函数是根据进去的信息（输入）和产生的东西（输出结果）所定义的一个黑盒。C++程序由若干个函数构成，有且只有一个主函数main（）。函数分为库函数和自定义函数两大类。

C++程序的构成（续）

- 程序语句

程序语句是程序的基本组成部分，一个语句是给计算机的一条完整的指令。在 C++里，一个语句是在结尾处用分号结束的。C++提供了说明语句、赋值语句、程序控制语句、复合语句及空语句等。

2.1.2 C++的语言基础

- C++的词法规则：
  - 关键字
  - 标识符
  - 语言符号
  - 注释符
- C++的数据类型：
  - 基本类型：包括整型（int）、浮点型（float）、字符型（char）和逻辑型（bool）

C++的数据类型（续）

- 空类型 ： void类型
- 构造类型 ： 常见的构造类型有数组、结构体、联合体和枚举。
- 指针类型：指针类型变量用于存储另一变量的地址，而不能用来存放基本类型的数据。它在内存中占据一个存储单元。
- 类类型 ： 类是体现面向对象程序设计的最基本特征，也是体现C++与C最大的不同之处。类也是一个数据类型，它定义的是一种对象类型，由数据和方法组成，描述了属于该类型的所有对象的性质。

2.1.3 C++中的类与对象

- 对象是构成世界的一个独立单位，它具有自己的静态特征和动态特征。
  - 静态特征是可以用某种数据来描述的特征。
  - 动态特征即对象所表现的行为或对象所具有的功能。
- 类是面向对象语言必须提供的用户定义的数据类型，它将具有相同状态、操作和访问机制，多个对象抽象成为一个对象类。

C++中的类与对象 （续１）

- 类的一般定义格式如下：

```
class  〈类名〉
{
private:
      〈私有数据成员和成员函数〉;
protected:
      〈保护数据成员和成员函数〉;
public:
      〈公有数据成员和成员函数〉;
}
       <各个成员函数的实现>;
```

C++中的类与对象 （续2）

- 定义类的函数成员的格式如下：

返回类型　类名::成员函数名（参数列表）

{

　　函数体

}

- 对象的定义格式如下：

　　<类名><对象名表>;

C++中的类与对象（续3）

- 对象的成员（一个对象的成员是该对象的类所定义的成员）表示如下：

〈对象名〉.〈成员名〉

或者：

〈对象名〉—〉〈成员名〉

2.1.4 类的继承和多态性

- 继承是面向对象设计的基本特征之一，是从已有的类基础上建立新类。通过C++语言中的继承机制，一个新类既可以共享另一个类的操作和数据，也可以在新类中定义已有类中没有的成员。
- C++的另一个重要的特征是支持多态。所谓多态性是指当不同的对象收到相同的消息时，产生不同的动作。

继承

如在定义类B时，如果继承类A，就会自动得到类A的操作和数据属性，使得程序员需定义类A中所没有的新成分即可完成在类B的定义，这样称类B继承了类A，类A派生了类B。这种机制称为**继承**。称类A为基类或父类，类B为派生类或子类。

继承的定义格式如下：

class〈派生类名〉：〈继承方式〉〈基类名〉

继承（续）

<继承方式>有三种关键字给予表示：

- public：公有继承，其特点是基类的公有成员和保护成员作为派生类的成员时，它们都保持原有的状态，而基类的私有成员仍然是私有的。
- protected：保护继承，其特点是基类的所有公有成员和保护成员都成为派生类的保护成员，并且只能被它的派生类成员函数或友元访问，基类的私有成员仍然是私有的。
- private：私有继承，其特点是基类的公有成员和保护成员作为派生类的私有成员，并且不能被这个派生类的子类访问。

继承实例

```
//基类
class CBase
{
public:
      void FuncA（void）;
      void FuncB（void）;
};
//派生类
class CDerive : public CBase
{
public:
      void FuncC（void）;
      void FuncD（void）;
};
//实例主程序
main（）
{
    CDerive  b;  // CDerive的一个对象
    b. FuncA（）; // CDerive从CBase继承了函数FuncA
    b. FuncB（）; // CDerive从CBase继承了函数FuncB
    b. FuncC（）;
    b. FuncD（）;
}
```

多态

C++的多态性具体体现在运行和编译两个方面：

- 在程序运行时的多态性通过继承和虚函数来体现。
- 在程序编译时多态性体现在函数和运算符的重载上。

多态实例

```
class CBase
{
public:
      virtual void FuncA（void）{cout<<”This is CBase : : FuncA\n”}
//用关键字virtual声明一个虚函数
};
void Test（CBase *a）
{
a->FuncA（）;
}
class CDeriveA : public CBase
{
public:
      virtual void FuncA（void）{cout<<”This is CDeriveA: : FuncA\n”}
};
class CDeriveB : public CBase
{
public:
      virtual void FuncA（void）{cout<<”This is CDeriveB: : FuncA\n”}
};
```

多态实例（续）

```
//主程序
main（）
{
    CBase  b; // CBase的一个对象
    CDeriveA  objectA; // CDeriveA的一个对象
    CDeriveB  objectB; // CDeriveB的一个对象
    Test（&b）;
    Test（& objectA）;
    Test（& objectB）;
}
//输出结果
This is CBase : : FuncA
This is CDeriveA: : FuncA
This is CDeriveB: : FuncA
```

2.1.5 C++中的输入/输出流

例：

```
cout<<"Enter your name: ";
cin>>name;
cout<<"Your name is: "<<name<<'\n';
```

第一条语句用到了标准输出流cout和运算符<<，<<称为流插入运算符。第二条语句用到了标准输入cin和运算符>>，>>称为流提取运算符。与C中的printf和scanf不同的是，流插入运算符和流提取运算符不需要指示输出/输入数据类型的格式，控制串、转换说明符和运算符能自动识别要用的类型。

C++中的输入／输出流（续）

用C++风格的面向流的输入／输出可以使得程序具有更好的可读性，并且能减少出错的可能。

- 注意：C++程序必须包含头文件iostream.h后才能使用输入／输出流，这一文件包含了所有输入／输出流操作所需的基本信息。