第2讲（1）3-6章复习与重点串讲

《C++语言程序设计》在线直播

清华大学　郑　莉

目录

- 第3章 函数的基础语法
- 第4章 类与对象
- 第5章 数据的共享与保护
- 第6章 数组与指针

第3章函数的基础语法

函数基础语法内容

- 函数的声明和调用
- 函数间的参数传递
- 内联函数
- 带默认形参值的函数
- 函数重载
- C++系统函数

函数

- 函数是定义和实现一个功能模块的机制
- 在面向过程的程序设计中，函数是程序模块的最小单位。
- 在面向对象的程序设计中，函数是类的行为的体现。

函数的定义、声明和调用

- 函数定义是函数功能的具体实现代码
- 使用函数以前要声明函数原型
- 函数调用和返回的过程要讲透
- 函数递归调用是难点，从以下几方面复习
  - 一些实际问题和解决问题的方法本身的描述是递归的，典型例子是阶乘，自己在找一些例子。
  - 通过开发环境的调试工具观察递归调用的过程。

函数的参数传递

- 形参在形实结合时才分配内存空间并被初始化
- 形实结合时参数单向传递（传值）。复习数据交换不成功的典型例子。
- 运行观察引用作形参的例题。

内联函数

- 从兼顾执行效率和程序结构的角度来理解内联函数。理解透了函数调用的执行过程，这里才能理解函数调用的效率问题。
- 编译器在执行优化时可能会自主决定函数是否内联

带默认参数值的函数

- 重点要理解默认参数值的位置、实参的传递次序。
- 在后续例题中，类的构造函数会较多用到默认参数值。

函数重载

- 重载的思想，与人类自然的思维方式是一致的。
- 重载使得具有相似但不同功能的函数可以拥有相同的函数名，方面了函数的使用者。
- 重载类的成员函数可以提高类通用性
- 区分重载的因素：参数表、const（是否常函数）
- 后续例题中大量用到重载函数

C++系统函数

- C++的系统函数是从c语言继承来的，对应的头文件是<cxxx>
- 系统函数的说明，可以查阅在线的参考文档。

第4章 类与对象

面向对象的思想背景

- 现实中的类与对象
- 计算机程序对现实问题的模拟
- 程序中基本数据类型的局限性及自定义类型的必要性
- “类型”的含义：数据+处理函数
- 自定义类型的途径——类

超越语法理解类与对象

- 从现实问题出发理解对象的实际意义
- 从对现实事物的分类管理角度认识抽象与分类观点理解引出C++中的封装机制——类
- 以现实中封装物体的内部构造与外部接口，以及程序对现实的模拟角度，看成员的访问属性。

构造与析构函数

- 回顾基本类型变量初始化的过程
- 对象初始化的必要性及其困难
- 自定义的初始化——构造函数
- 对象的“克隆”——复制构造
- 对象消亡前的“善后”——析构函数

类重用的方式之一——组合

- 部件组装的思想背景
- 组合类与成员对象间的“has-a”关系
- 组合类的部件成员
- 组合类对象的构造与析构

第5章 数据的共享与保护

变量的寿命、作用域、可见性

- 理解变量的寿命、作用域、可见性是数据共享的基础
- 寿命
  - 静态生存期：全局寿命
  - 动态生存期：局部寿命
- 复习例题演示的寿命、作用域、可见性的效果，可以借助调试工具观察

数据共享的需求与途径

- 函数间需要数据共享
- 同一个类的对象间需要数据共享
- 不同类之间需要数据共享

数据共享的相关语法（一）

- 函数间的数据共享
  - 全局变量、函数参数、返回值
  - 将共享的数据与共享数据的函数封装成类

数据共享的相关语法（二）

- 同一个类的对象间数据共享
  - 静态成员，也称类成员
- 静态数据成员
  - 该类的所有对象共享该成员的同一个拷贝
- 静态函数成员
  - 类外代码可以使用类名和作用域操作符来调用静态成员函数。
  - 静态成员函数只能直接引用属于该类的静态数据成员或静态成员函数。

数据共享的相关语法（三）

- 类与全局函数间、不同类之间的数据共享
  - 友元函数
  - 友元类

共享数据的安全保护

- 常类型
  - 常引用：保护被引用的对象
  - 常对象：对象本身不可改变，用常成员函数访问
  - 常指针：保护被指向的对象

编译预处理命令与多文件结构

- 主要通过实验理解和掌握
  - 学生用书“实验五”——不提交不计分

第6章 数组与指针

如何处理大量同类型对象

- 对同类型对象群体中每一个体往往使用同样的处理方法。可否使用循环语句？关键是在循环体中如何以统一的方式标识每一个体变量。
- 群体中的对象往往具有逻辑上的密切联系，或存在次序关系。如何以数据结构来描述这种关系？
- 数组恰恰是解决上述问题的数据结构。

数组的要点

- 数组的声明与引用
  - 以统一的名称命名一组变量
  - 以下标表示元素的序号
- 数组的实质
  - 数组名即数组的首地址
  - 数组元素在内存中是按次序连续存放的
- 对多维数组的理解
  - 二维数组可以看成是一维数组的数组
  - 二维数组按行存放
- 对象数组的构造需要调用元素类的构造函数

数组的适用场合举例

- 用于存储同类型的数据序列
- 用于在函数间传递大批数据
  - 以数组名为参数，传递数组的首地址而不是元素值
  - 要注意：对形参数组的修改会直接反映到实参数组中

数组的主要优点与缺点

- 优点
  - 便于按统一方式处理大量同类型数据
  - 支持对同类型数据序列的随机访问
  - 便于在函数之间共享大量数据
- 缺点
  - 插入/删除/重新排列元素时需要大量移动元素。
  - 存在下标越界等安全隐患
  - 长度不可动态调整

指针的要点

- 访问程序的内存单元既可以通过变量名，也可以通过地址
- 指针是存放地址的变量
- 指针只有被赋予合法地址以后，才能使用
- 指针的运算规则与指向的类型相关
- 指针可以接受动态分配的内存地址
- 指向函数的指针可以增加程序的通用性
- this指针使得非静态成员函数可以识别处理的对象
- 指向类非静态成员的指针必须与对象名/对象指针联合使用

一张图看懂指针

- 概念
  - **指针**：内存地址，用于间接访问内存单元
  - **指针变量**：用于存放地址的变量
- 声明和定义

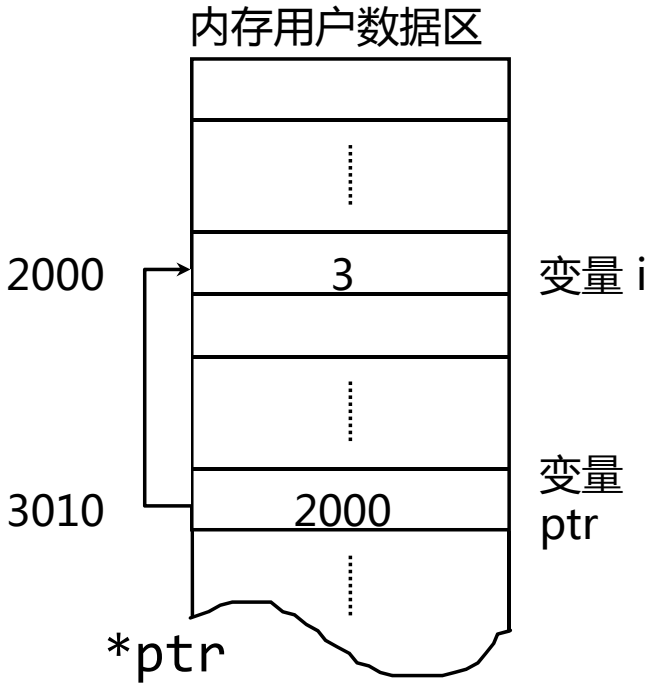
  例：static int i;

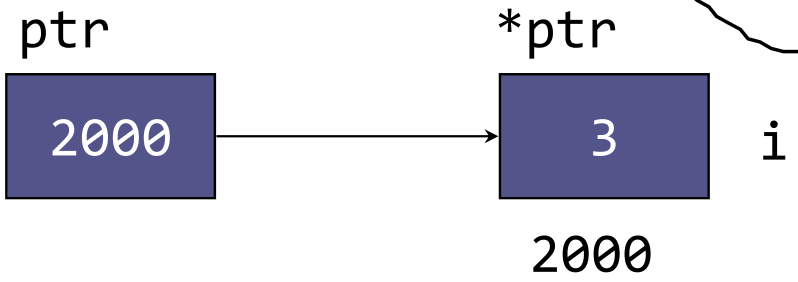
  static int* ptr = &i;

  指向int变量的指针
- 引用

  例1：i = 3;

  例2：*ptr = 3;





指针与数组

- 指向数组元素的指针
  - 数组是一组连续存储的同类型数据，可以通过指针的算术运算，使指针依次指向数组的各个元素，进而可以遍历数组。
  - 静态定义的数组不必使用指针访问，动态数组只能使用指针访问。
  - 理解数组的存储结构，是理解用指针访问数组的基础。
- 指向整个数组的指针
  - 接受动态分配的多维数组首地址需要指向整个数组的指针。
- 指针数组
  - 数组的每一个元素是指针。
  - 用于：
    - 数组元素是大对象/数组，且需要频繁交换元素的情况。
    - 用基类的指针素组存放指向各不同派生类对象，以实现多态性。

指针与函数

- 以指针作为函数参数
  - 需要数据双向传递时（引用也可以达到此效果）
    - 用指针作为函数的参数，可以使被调函数通过形参指针存取主调函数中实参指针指向的数据，实现数据的双向传递
  - 需要传递一组数据，只传首地址运行效率比较高
    - 实参是数组名时形参可以是指针
- 指针类型的函数
  - 函数返回值是指针
  - 不要将非静态局部地址用作函数的返回值

指针与函数

- 指向函数的指针——实现函数回调
  - 通过函数指针调用的函数
    - 例如将函数的指针作为参数传递给一个函数，使得在处理相似事件的时候可以灵活的使用不同的方法。
  - 调用者不关心谁是被调用者
    - 需知道存在一个具有特定原型和限制条件的被调用函数。

this指针

- 隐含于类的每一个非静态成员函数中。
- 指出成员函数所操作的对象。
  - 当通过一个对象调用成员函数时，系统先将该对象的地址赋给this指针，然后调用成员函数，成员函数对对象的数据成员进行操作时，就隐含使用了this指针。
- 例如：Point类的getX函数中的语句：

```
return x;
```

相当于：

```
return this->x;
```

使用内存地址的优点及缺

- 优点
  - 灵活高效
  - 支持动态内存分配
  - 有利于数据共享
- 缺点
  - 安全性差，可能会出现对内存的非法访问
  - 使程序的可读性下降
- 注意事项
  - 使用指针时要确保其含有有效地址，例如不要使用未被初始化的指针，函数不要返回动态局部变量的地址等

例题：动态数组类

- 主要知识点
  - 动态内存分配
  - 在析构函数中释放动态分配的内存空间
  - 深层复制与前层复制问题

字符串的存储与处理

- 以字符数组存储字符串——C语言的方法
  - 具有数组固有的安全隐患
  - 只能使用库函数进行处理
- 使用String类
  - 安全性好
  - 封装了处理函数，重载了运算符