程序设计基础及语言

Part II

https://dwz.cn/p2cSSmVx

http://cse.seu.edu.cn/2019/0103/c23024a257235/page.htm

杨　明

13851793121, 计算机楼360室

yangming2002@seu.edu.cn

课程总结

❖授课内容

- **Ch09:** 类的进一步深入, 重点
- **Ch10:** 掌握运算符重载原理
- **Ch11: Inheritance**继承, 重点
- **Ch12: Polymorphism**多态, 重点, **12.7-8**不做要求
- **Ch14: File**文件, 重点, 要求掌握**14.1-4**
- **Ch15:** 迭代器与容器, 典型容器的使用
- **Ch17: Exception, 17.9 unique_ptr**不做要求
- **Ch18: Template**模板
- **Ch19:** 模板化数据结构, 链表、堆栈和队列

复习提纲

1. 类的进一步深入
2. 运算符重载
3. 类之间的关系: 组合
4. 类之间的关系: 继承与多态
5. 类模板
6. 顺序文件处理
7. 异常
8. 容器和模块化数据结构

1. 类的进一步深入探讨

- 1.1 构造和析构函数
- 1.2 常对象、常成员函数和常数据成员
- 1.3 友元函数和友元类
- 1.4 this指针
- 1.5 动态内存管理
- 1.6 静态类成员

1.1 构造和析构函数

❖ 构造函数

- 拷贝构造函数: 单参数, 同类型引用, 通过已有对象构造新对象(同类, ***is-a***)
  1. 传值方式传递对象参数
  2. 函数返回对象
  3. 使用同类对象来初始化对象

- 转换构造函数: 单参数, 非同类型
  - 已实现 **HugeInt(int)** 和 **HugeInt+HugeInt**
  - **HugeInt + 100**

1.1 构造和析构函数

❖ 析构函数

- 对象结束时自动调用, 清理

❖ 构造和析构的顺序

- 总体上, 先构造的后析构
- 结合组合和继承情况进行分析 (基类/被包含类有多个构造函数时, 在派生类/宿主类的构造函数初始化列表指定调用哪个版本的构造函数)

1.2 常对象、常成员函数和常数据成员

- 常对象: **const Time noon(12, 0, 0);**
  - 构造后不能更改数据成员
  - 仅能调用**noon**对象常成员函数

- 常成员函数
  - 显式地声明为**const**
  - 不修改本对象、不调用**non-const**成员函数

- 常数据成员
  - 构造函数初始化列表

1.3 友元函数和友元类

外部函数和类, 访问类的非公有成员:

- 典型应用: **operator overloading (运算符重载)**

```
class ClassOne
{
  friend class ClassTwo; // friend declaration
  int x, y;
};

class ClassTwo{
public:
  void setX(ClassOne &one, int x){ one.x = x; }
  void setY(ClassOne &one, int y){ one.y = y; }
  void printClassOne(ClassOne &one){
    cout  << "ClassOne.x = " << one.x
          << ", ClassOne.y = " << one.y << endl;
  }
};
```

ClassOne授权ClassTwo为其友元

1.4 this指针

成员函数隐含参数, 指向当前对象

- **Cascaded Function Calls(级联函数调用)**

```
30. Time& Time::setHour( int h ) // note Time & return
31. {
32.     hour = ( h >= 0 && h < 24 ) ? h : 0; // validate hour
33.     return *this; // enables cascading
34. } // end function setHour
```

```
1. t.setHour(18).setMinute(30).setSecond(22);
2. t.setMinute(30).setSecond(22);
3. t.setSecond(22);
```

1.5 动态内存管理

(1) new和delete

```
1. int size = 10;
2. int *gradesArray = new int[ size ];
3. delete [ ] gradesArray;
```

在类的数据成员中如果有指针存在, 通常意味着需要进行动态内存管理(分配new+释放delete)

1.5 动态内存管理

- **new** 对象:
  - **allocates storage** of the proper size for an object
  - **calls the constructor** to initialize the object
  - **returns a pointer** of the type specified to the right of the new operator

- **delete** 对象:
  - **calls the destructor** for the object to which pointer points
  - **deallocates the memory** associated with the object

1.6 静态类成员

- **static数据成员: 类变量, 所有对象共享**

  在类定义中声明, 在类定义外定义和初始化

- **static成员函数: 无this指针**

- 两种访问方式
  - 类名:: **[static成员特有的访问形式]**
  - 对象名.

复习提纲

1. 类的进一步深入
2. 运算符重载
3. 类之间的关系: 组合
4. 类之间的关系: 继承与多态
5. 类模板
6. 顺序文件处理
7. 异常
8. 容器和模块化数据结构

2. 运算符重载

- 选择一: 非静态的类成员函数

```
class String{
public:
    bool operator!() const;
};
```

- 选择二: 全局函数, friend函数

```
class PhoneNumber{
    friend ostream& operator<< ( ostream&,
                                 const  PhoneNumber & );
};
```

- 表达式形式和函数调用形式的互换
- String类的实现

复习提纲

1. 类的进一步深入
2. 运算符重载
3. 类之间的关系: 组合
4. 类之间的关系: 继承与多态
5. 类模板
6. 顺序文件处理
7. 异常
8. 容器和模块化数据结构

3. 类之间的关系: 组合

- **Composition (组合, *has-a*): A class has objects of other class as members. (vs *is-a*)**
- **Host Object (宿主对象) vs Contained Object(被包含对象)**

Employees
- LastName: String - FirstName: String - birthDate: Date - hireDate: Date

3. 类之间的关系: 组合

- 成员对象的构造和析构顺序
  - 成员对象的构造先于宿主对象
  - 成员对象之间按照类定义中的声明顺序构造 (**not in the order they are listed in the constructor's member initializer list**)
  - 成员对象的析构后于宿主对象

复习提纲

1. 类的进一步深入
2. 运算符重载
3. 类之间的关系: 组合
4. 类之间的关系: 继承与多态
5. 类模板
6. 顺序文件处理
7. 异常
8. 容器和模块化数据结构

4. 类之间的关系: 继承与多态

- 4.1 Protected Members
- 4.2 构造与析构顺序
- 4.3 Polymorphism(多态)与虚函数
- 4.4 纯虚函数与抽象类
- 4.5 一个例子

4.1 Protected Members

- 类有两种用户:
  - 对象(句柄): 对类进行实例化
  - 派生类: 在该类的基础上派生并设计新类

	类对象	派生类
public	/	/
protected	X	/
private	X	X

4.1 Protected Members

- **派生类如何访问基类的成员?**
  - 派生类可以直接通过成员名来引用基类的**public**成员和**protected**成员
  - 派生类可以重定义**(redefine)**基类的成员，并且依然可以通过以下方式访问基类的**public / protected**成员:

    基类名 :: 成员名
  - 重定义**(redefine)**: 在派生类中给出基类的同名成员

4.2 构造与析构顺序

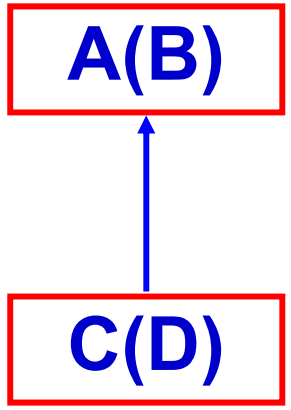
- **若类中含有其他类的对象(组合)**

  构造: 先基类后派生类, 先被包含类后宿主类

  B → A → D → C

  析构: 与构造顺序相反

  C → D → A → B



4.3 Polymorphism(多态)与虚函数

- **Polymorphism(多态):**

  通过指向派生类的基类指针，调用的是派生类函数

  综合覆盖或未覆盖不同情况判断实际调用的函数！

- **Virtual Function(虚函数)**

  调用哪个(基类/派生类)虚函数，由指向的实际对象类型而不是句柄类型决定

- 虚析构函数

4.3 Polymorphism(多态)与虚函数

- 基 类

```
1.  class Shape{
2.  public:
3.      virtual void draw() const;
4.  };
```

- 派生类

```
1.  class Rectangle : public Shape{
2.      virtual void draw() const;
3.  };
```

可省略. 只要基类声明函数为虚函数, 则所有派生类的该函数均为虚函数

4.3 Polymorphism(多态)与虚函数

- 基 类

```
1.  class Shape{
2.  public:
3.      virtual void draw() const;
4.  };
```

Override(覆盖)

- 派生类

```
1.  class Rectangle : public Shape{
2.      virtual void draw() const;
3.  };
```

4.3 Polymorphism(多态)与虚函数

- 只有类成员函数才能声明为虚函数
- 静态成员函数不能是虚函数
- 构造函数不能是虚函数(编译错误)
- 析构函数通常声明为虚函数(所有带有虚函数的类都建议设计为虚析构函数)
  - **Small *p = new Big;**
  - **delete p; // 调用Big的析构函数**

4.4 纯虚函数与抽象类

- **Pure Virtual Function(纯虚函数)**

  A pure virtual function is specified by placing "= 0" in its declaration, as in

  ```
  virtual void draw() const = 0;
  ```

- 对于纯虚函数, 不需要在类源码中给出其实现.

4.4 纯虚函数与抽象类

- **Abstract Class(抽象类)**: 包含一个或多个纯虚函数的类. 无法实例化, 但可以声明指针和引用, 只能用于继承.

```
1. Shape obj;                          // Error, 不能实例化
2. Rectangle objRectangle;
3. Shape *ptr = &objRectangle;  // OK, 可指针
4. Shape &ref = objRectangle;    // OK, 可引用
```

- **Concrete Class(具体类)**: 不包含纯虚函数, 可以实例化

4.5 一个例子

目的: 输出各类员工的基本信息和薪金信息

- **Salaried employees (普通薪金制员工)**

  Name, SSN, Weekly Salary

- **Hourly employees (计时工)**

  Name, SSN, Wage per hour, Hours

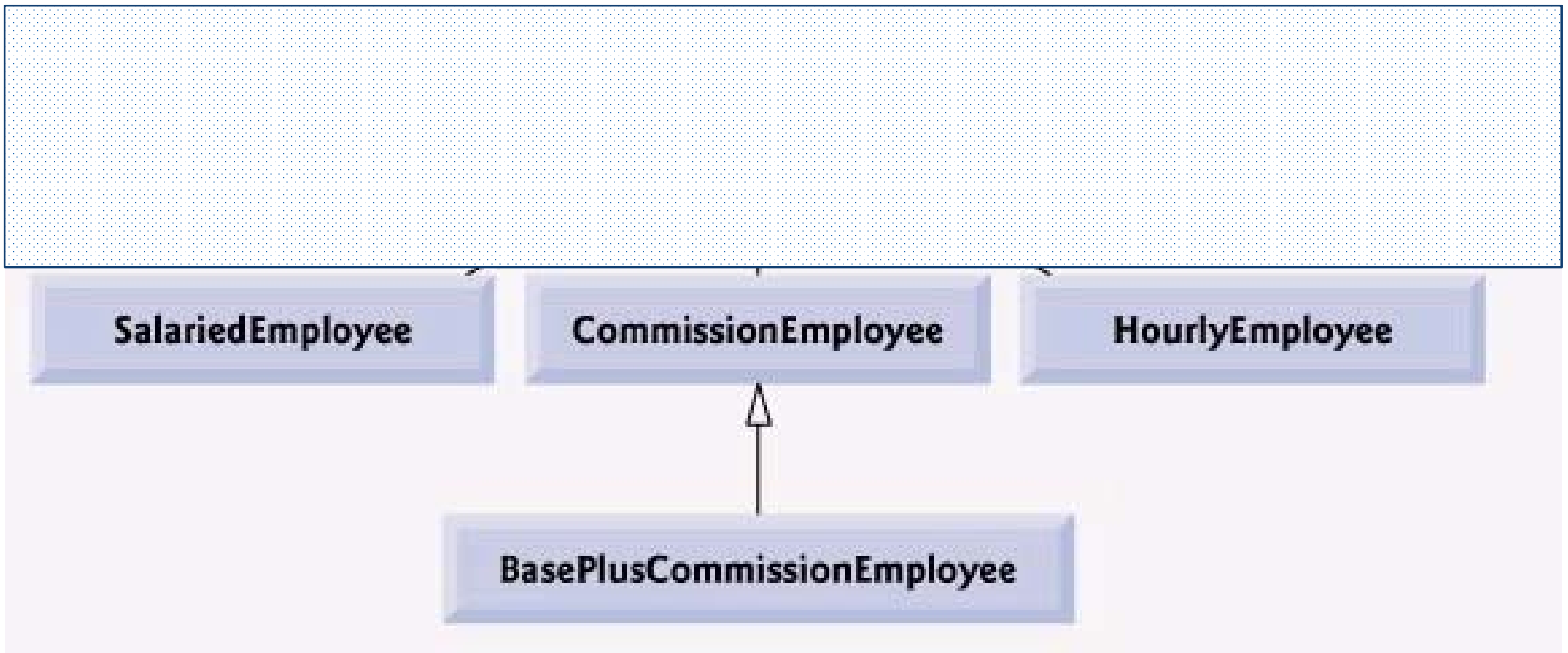
- **Commission employees (佣金制员工)**

  Name, SSN, Gross sales amount, Commission rate

- **Base-salary-plus-commission employees (带底薪的佣金制员工)**

  Name, SSN, Gross sales amount, Commission rate, Base Salary

4.5 一个例子



为何设计基类***Employee***?

- 尽可能多地抽取共有属性和操作 – 代码重用
- 多态和泛型编程, 需要设计共同的基类

复习提纲

1. 类的进一步深入
2. 运算符重载
3. 类之间的关系: 组合
4. 类之间的关系: 继承与多态
5. 类模板
6. 顺序文件处理
7. 异常
8. 容器和模块化数据结构

Class Templates

- **vector< int > integers1(7);**

  类模板, 属于 **C++ STL**

- 类模板可称为参数化类型**(parameterized types)**, 需要一个或多个类型参数来定制“通用类”以构成类模板特化.

- **vector**等的应用

Class Templates

```
1. template< typename T >
2. class Stack
3. {
4.     // class definition body
5. };
```

- 其中 T 可以用于成员函数(形参、局部变量和返回值)和数据成员

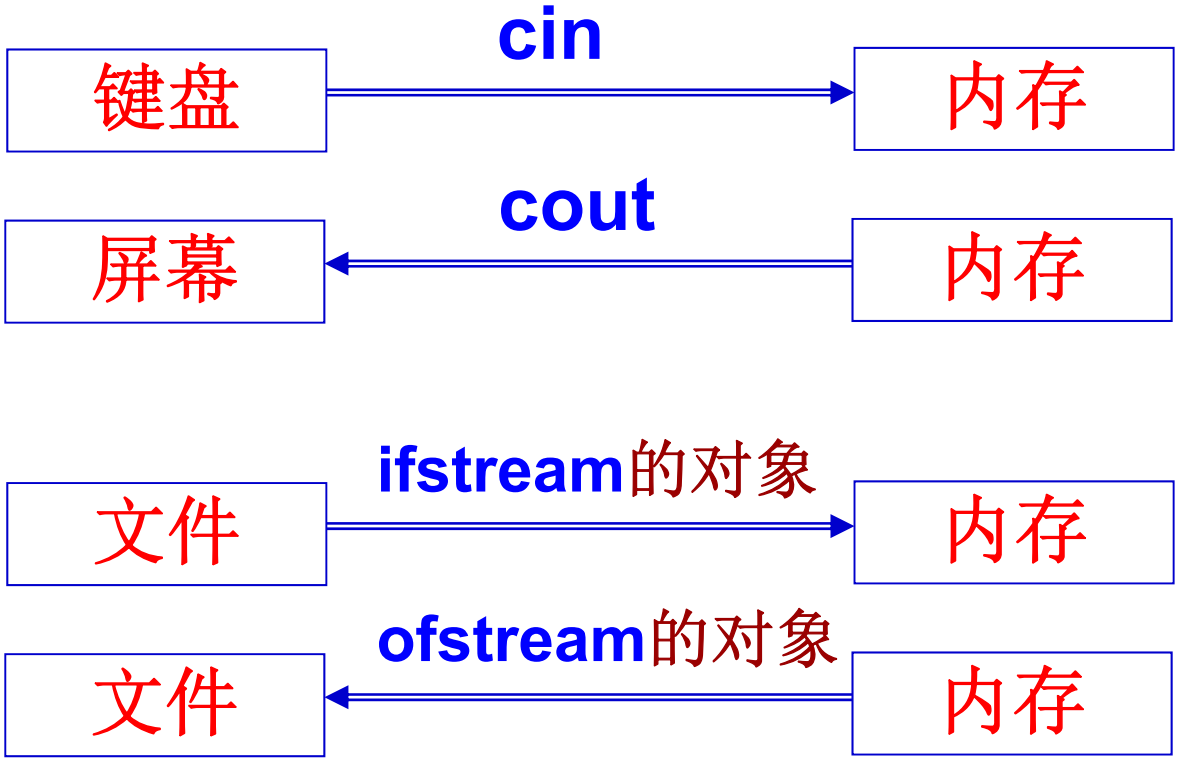
复习提纲

1. 类的进一步深入
2. 运算符重载
3. 类之间的关系: 组合
4. 类之间的关系: 继承与多态
5. 类模板
6. 顺序文件处理
7. 异常
8. 容器和模块化数据结构

6. 顺序文件处理

- 6.1 与cin/cout的对比
- 6.2 ofstream对象(写文件)
- 6.3 ifstream对象(读文件)
- 6.4 文件读写

6.1 与cin/cout的对比



- 主要差异: 文件操作时需定义 **ifstream / ofstream**对象, 以指定所具体操作的文件和操作相关的参数

6.2 ofstream对象

❖ 创建ofstream对象 – 写文件

(1) 创建流类对象的同时打开文件

ofstream(const char* filename, int mode)

- filename: 路径 + 文件名(含后缀)
  - "c:\\clients.dat"
  - "clients.dat"　　// 当前路径
- mode:
  - using std::ios;
  - ios::out　ofstream的缺省模式
    - ① 若文件存在, 则打开并丢弃现有数据
    - ② 若文件不存在, 则创建

6.2 ofstream对象

- 创建ofstream对象

(2) 先创建对象, 后打开文件

- 缺省构造函数 + **open**成员函数
- **open**与前述构造函数的参数相同

```
ofstream outClientFile;
outClientFile.open("clients.dat", ios::out);
```

6.2 ofstream对象

- 文件的关闭
  - ofstream析构时会自动关闭文件
  - 建议当文件不再需要使用时，显式调用close成员函数关闭

  - ```
    ofstream outClientFile;
    outClientFile.open("a.dat", ios::out);
    ……
    outClientFile.close();
    outClientFile.open("b.dat", ios::out);
    ……
    outClientFile.close();
    ```
    ```

6.3 ifstream对象

- 创建 **ifstream** 对象 – 读文件

(1) 创建流类对象的同时打开文件

```
ifstream inClientFile( "clients.dat", ios::in );
```

- **ios::in** – 缺省模式, 仅能从文件读取数据 (最小权限原则)

(2) 创建对象, 后打开文件

```
ifstream inClientFile
inClientFile.open("clients.dat", ios::in );
```

6.4 文件读写

- 文件读写
  - 顺序文件操作: 从文件的开始处依次顺序读写文件内容, 不能任意读写文件内容.
  - 读: **ifstream**, 文件流类的**get**、**getline**、**read**成员函数以及流抽取符“**>>**” (**cin>>**)
  - 写: **ofstream**, **put**、**write**函数以及流插入符“**<<**” (**cout<<**)

复习提纲

1. 类的进一步深入
2. 运算符重载
3. 类之间的关系: 组合
4. 类之间的关系: 继承与多态
5. 类模板
6. 顺序文件处理
7. 异常
8. 容器和模块化数据结构

Termination Model of Exception Handling

1. 抛出异常时, 当前的(try)block结束执行;
2. 寻找匹配的catch handler (*is-a*);
3. 执行catch handler代码;
4. 程序控制跳至最后一个catch handler后的首条语句. (注意: 不再执行try block中抛出异常点的后续语句)

栈展开机制

```
1.  class Obj{
2.      int id;
3.  public:
4.      Obj(int n){
5.          id = n;
6.          cout << "ctor " << id << endl;
7.      }
8.      ~Obj(){
9.          cout << "dtor " << id << endl;
10.     }
11. };
12. void f2( )
13. {
14.     Obj o(2);
15.     double a = 0;
16.     try{
17.         throw a;
18.     }
19.     catch(double){
20.         cout << "OK2!" <<endl;
21.         throw;
22.     }
23.     cout << "end2" <<endl;
24. }
```

X 栈展开不涉及o(2)

```
25. void f1( )
26. {
27.     Obj o(1);
28.     try{
29.         f2( );
30.     }
31.     catch(char){
32.         cout << "OK1!" << endl;
33.     }
34.     cout<< "end1" <<endl;
35. }
36. int main( )
37. {
38.     try{
39.         Obj o(0);
40.         f1( );
41.     }
42.     catch(double){
43.         cout << "OK0!" <<endl;
44.     }
45.     cout<< "end0" <<endl;
46.
47.     return 0;
48. }
```

栈展开涉及0, 1, 2

```
ctor 0
ctor 1
ctor 2
OK2!
dtor 2
dtor 1
dtor 0
OK0!
end0
```

栈展开机制

- 栈展开的本质：在进入异常处理代码之前，把需要析构的对象全部析构掉
- 对象构造时发生异常，不再析构

复习提纲

1. 类的进一步深入
2. 运算符重载
3. 类之间的关系: 组合
4. 类之间的关系: 继承与多态
5. 类模板
6. 顺序文件处理
7. 异常
8. 容器和模块化数据结构

容器和模块化数据结构

- 学会使用**vector/stack**等数据结构，了解元素操作特点
- 掌握链表等数据结构的实现

考试安排

程序设计基础与语言II

- 平时成绩: 实验+综合设计+到课率
- 期末考试: 笔试(40分)+机试(60分)
- 总成绩: 30%*平时成绩 +
  10%*综合设计 +
  60%*期末考试成绩

考试安排

- 时间:

  笔试 2021年6月28日 09:00-10:00 (60min)

  机试 2021年6月28日 10:10-12:10 (120min)

- 地点: 金智楼实验中心四楼5号

  补修+辅修+重修学院二楼239自带笔记本

- 形式: 半开卷, 可带中英文教材各1本

- 题型:

  代码阅读: 5题 = 20分

  代码填充: 2题 = 20分

  机试题: 3题 = 60分 (20+20+20)

机试要求

编程:

- 在本地D:盘中, 建立自己的文件夹, 用来完成程序的编写和调试.
- 建议: 第一题**Project**(项目名)为**Pro1**, 第二题**Project**(项目名)为**Pro2**, 以此类推.

提交: 考试结束前完成以下操作

- 在虚拟**Z:**盘上建立一个以自己的学号**+**姓名命名的文件夹, 用于保存上交的考试程序的源文件.
- 将每题的所有源程序文件***.cpp**、自定义头文件***.h**和可执行程序***.exe**存入**Z:**盘自己的目录中.

预祝各位同学取得好成绩!