# C++ Programming

## Chapter 8 Objects and Classes

Zheng Guibin
（郑贵滨）

哈爾濱工業大學
Harbin Institute of Technology

# 目录

CONTENT

## *Objects and Classes*

## 对象和类

- 什么是对象、类？
- 类的定义
- 类的使用
- 构造函数
- 类的接口、类的实现
- 析构函数（补充）
- const成员函数（补充）

# 复习...

- 类的定义格式

```
class 类名{
  public:
            公有数据成员和成员函数;
  protected:
            保护数据成员和成员函数;
  private: //默认访问类型
            私有数据成员和成员函数;
};
```

  - 数据成员通常是私有的、公有的通常是成员函数;
  - 在类定义中，public部分通常放的private部分的前面。

# 复习...

- 类（Class）是将数据(属性)和操作数据的函数(行为)**封装**成的一个整体，类可以隐藏数据和操作细节
- 对象：类的一个实例,用自己的方法完成对数据的操作
- 一个对象具有如下特征：
  - 标识，即对象的名字
  - 状态，即对象的数据成员
  - 行为，即对象的成员函数。

# 复习...

- 构造函数（Constructor）
  - 与类同名的成员函数，对象创建的时候自动被调用/执行
  - 经常用来进行数据成员的初始化
  - 构造函数不会被显示调用，无法返回一个数值，因此构造函数无返回类型，也不用 void声明。
  - 缺省构造函数无参数：系统自动生成的构造函数和无参构造函数属于缺省/默认构造函数； 一个类中只能有一个默认构造函数。
  - 当自定义一个构造函数后，系统不再自动生成无参构造函数
  - 定义每个对象时只执行多个构造函数中的一个。
  - 自身与成员对象的构造函数顺序：先成员构造，再自己构造
  - 初始化列表、默认参数值

# ????.....

- 银行账号如何生成?

人工设定?

太原始、太笨拙、效率低;

考验人的记忆力、责任心，没人记得住下一个账号值该是多少;

人总会出错....

# 8.13 static class data members(静态类数据成员)

- 类的静态数据成员：static修饰的数据成员。
- 静态数据成员独立于该类创建的所有对象。
- 类的静态数据成员只有一个实例(copy)，无论该类有多少对象；
- 类的所有对象共享这个静态数据成员，因此静态数据成员的数值对所有对象都是相同的；
- 每个对象修改了静态数据成员的值，则所有对象的该成员数值均改变；
- 静态数据成员的数值必须在类的外部单独初始化。

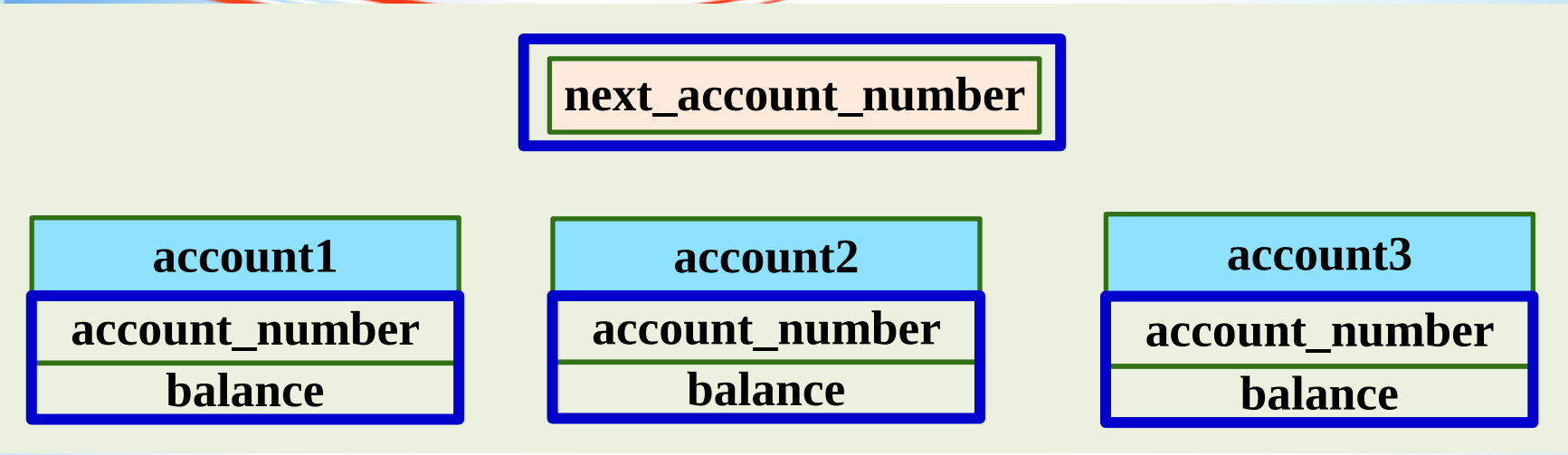
```
// Program exa
class bank_acc
{
public:
  bank_accoun
  bank_accoun
  bank_accoun
  void deposit(
  void withdra
  void display_
private:
  static int next_account_number ;
  int account_number ;
  double balance ;
} ;
```

```
//...
bank_account::bank_account()
{
  account_number = next_account_number++ ;
  balance = 0.0 ;
}
bank_account::bank_account( int acc_no )
{
  account_number = acc_no ;
  balance = 0.0 ;
}
```

```
int bank_account::next_account_number = 1 ;

main()
{
  bank_account account1, account2, account3 ;
  account1.deposit( 25.50 ) ;
  account2.deposit( 30.50 ) ;
  account3.deposit( 10.00 ) ;
  account1.withdraw( 20.04 ) ;
  account1.display_balance() ;
  account2.display_balance() ;
  account3.display_balance() ;
}
```

# 8.13 static class data members(静态数据成员)




- ***类变量（class variable）***
  - 静态数据成员和类相关联，而非和对象关联，因此成为类变量；
- ***实例变量（instance variables）：***
  - 和类的实例（对象）关联的数据成员称为实例变量

# 8.14 Using return in a member function

- 如果从类的外部获取/设置数据成员的数值?

  可通过成员函数实现:

  - 读值或检查函数(inspector or accessor function)

    类的成员函数可以有返回值

  - 赋值/设值函数 (mutator function)

    存取函数

- **Program Example P8G**

# 8.15 Inline class member functions（内联成员函数）

- **内联函数(Inline function)**

  有函数优点（代码书写、修改），但没有函数调用的开销

- **定义内联成员函数**
  - 使用inline前缀定义成员函数

```
inline  double get_balance();
```

  - 将成员函数的定义放在类的定义体内部。(不推荐)

# 8.15 Inline class member functions（内联成员函数）

- **内联函数工作原理:**

  函数代码扩展嵌入到调用处，并非调用，为编译指令，代码容量增大（一般是小函数），但减少调用的系统开销；程序运行速度快。

- **使用限制：与编译器有关**

  （1）不能包括循环分支转移语句。

  （2）不能递归。

  （3）不能包含static语句。

- **存取函数通常被定义为内联函数。**

# 8.16 Class interface and class implementation

使用类只需知道类的接口，不必知道私有部分、实现细节。

## 8.16.1 分离类的接口与实现

- C++无法完全隐藏类的实现
- 类的声明写入头文件，如：class_name.h
- 成员函数的定义放在源文件，如：class_name.cpp文件
- 接口注释/说明文档
- file ***bank_ac.h***

include语句用双引号：先搜索程序路径，然后搜索系统的默认路径

**软件工程的一个最基本的原则就是将接口与实现分离，信息隐藏是软件工程中非常重要的概念。**

# 8.16 Class interface and class implementation

- 使用.h 和.cpp

```
// Example P8H
#include <iostream>
#include <iomanip>
#include "bank_ac.h"
#include "bank_ac.cpp" //for Code::Blocks
using namespace std ;
main()
{
  bank_account my_account ;
  my_account.deposit( 889912.34 ) ;
  my_account.display_balance() ;
}
```

# 8.16 Class interface and class implementation

## 8.16.2 头文件中使用名字空间(namespace)

命名冲突:

- 包含了多个头文件
- 每个头文件有自己的变量、类，可能存在名字重复/冲突，导致编译错误。

**命名空间(namespace): 程序中命了名的语句块。**

- 一个命名空间内，标识符必须唯一;
- 不同的命名空间，可以有同名的标识符。

# 8.16 Class interface and class implementation

- classlib1.h

```
#if !define CLASS_LIB1_H
#define CLASS_LIB1_H
namespace classlib1
{
  class alpha
  {
  ...
  };
  class beta
  {
  ...
  };
  ...
}
#endif
```

classlib2.h

```
#if !define CLASS_LIB2_H
#define CLASS_LIB2_H
namespace classlib2
{
  class alpha
  {
  ...
  };
  class beta
  {
  ...
  };
  ...
}
#endif
```

# 8.16 Class interface and class implementation

- 区分同名标识符的方法
  - 使用作用域运算符::(名字解析运算符, scope resolution operator)

```
#include "classlib1.h"
#include "classlib2.h"
#include <iostream>
using namespace std;
int main( )
{
   classlib1::alpha  A1 ;  //  object of classlib1 alpha
   classlib2::alpha  A2 ;  //  object of classlib2 alpha
    ...
   return 0;
}
```

# 8.16 Class interface and class implementation

- 区分同名标识符的方法
  - 使用*using 语句*——指定命名空间中的特定元素

```
#include "classlib1.h"
#include "classlib2.h"
#include <iostream>
using namespace std;
int main( )
{
    using classlib1::alpha;
    alpha  A1 ;  //  object of classlib1 alpha
    alpha  A2 ;  //  object of classlib1 alpha
   ...
  return 0;
}
```

# 8.16 Class interface and class implementation

- 获取命名空间中的所有元素
- 使用*using 语句*——使用命名空间中的所有元素

std 标准C++ 库使用的名字空间，包含：类、cin 、cout 等

```
#include "classlib1.h"
#include "classlib2.h"
#include <iostream>
using namespace std;
int main( )
{
    using namespace  classlib1;
    alpha  A1 ;  //  object of classlib1 alpha
    alpha  A2 ;  //  object of classlib1 alpha
    belta   A3;   //  object of classlib1 beta
   ...
```

# 补充：析构函数（destructor）

- **析构函数(destructor)——构造函数的反函数**
  - 析构函数是用于取消对象的成员函数，当一个对象生命期结束时，系统自动调用析构函数清除对象，释放内存等。
  - 没有定义析构函数时，系统提供缺省版本的析构函数。
  - 析构函数名为：**~类名**
  - 析构函数没有参数，也没有返回类型
  - 一个类中只可能定义一个析构函数，析构函数不能重载。
  - 如果一个类没有定义析构函数，编译器会自动生成一个默认析构函数，默认析构函数是一个空函数。

# 补充：析构函数（destructor）

- 析构函数的调用

构造函数 析构函数

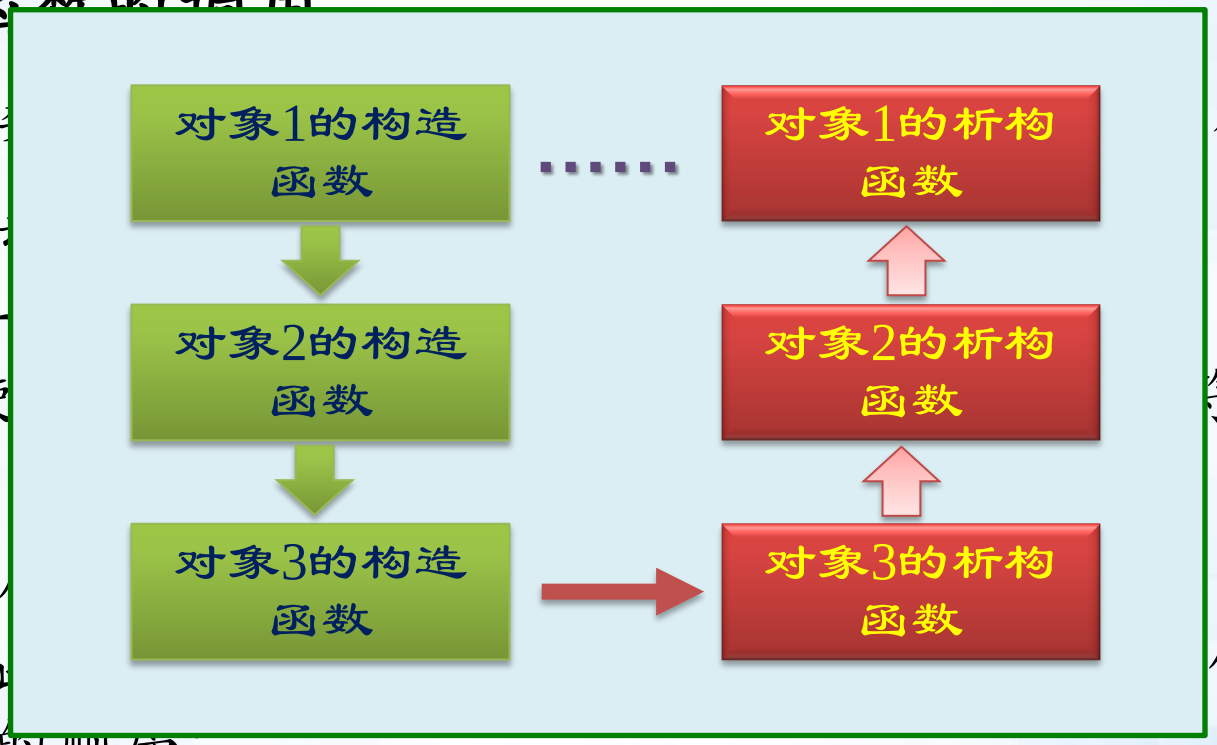
将被自动

(1) 一

(2) 使 等释放该对象时。

调用顺序

- 析构函数 例化对象范围的顺序；
- 一般析构函数的调用顺序与构造函数相反。
- 先自己析构，再成员析构(验证！)



# 补充：const成员函数

- **const成员函数**
  - 编译器**不允许**const成员函数修改对象；
  - 编译器**不允许**非const成员函数调用const对象。

**const 成员函数的声明、定义:**

```
class bank_account
{   …
    double get_balance() const;
    …
};
double bank_account::get_balance() const
{
  return balance;
}
```

**const对象：** `const bank_account my_account;`

**代码例子：const_member_func.cpp**

# 函数中应用对象

- 对象可以作为函数的参数

例子：class AAA{…};

```
void myfunction1(AAA    x);
void myfunction2(AAA * ptr);
void myfucntion3(AAA & x);
```

- 函数可以返回对象

例子：class AAA{…};

```
AAA     myfunction1();
AAA *   myfunction2();
AAA & myfucntion3();
```

```
class Box            //Box . h
{ public:
    Box(int =1 ,int=1, int=1);
    int volume( );
 private:
    int height;
    int width;
    int length;
};
Box::Box(int h,int w,int len): height(h),  width(w), length(len)
//Box . cpp
{ }
int Box::volume( )
{
   return (height*width*length);
}
```

# Programming Pitfalls

- 1. A class **constructor is called automatically** when an object of the class is created, it cannot be called explicitly.
- 2. Don't forget to place a semi-colon after the last } in a class declaration.
- 3. A constructor has no return type, not even void.
- 4. Do not include parentheses when a default constructor is used to create an object of a class.

```
bank_account my_account(); // Incorrect.
bank_account my_account;    // Correct.
```

如果用new呢？ bank_account *p= new bank_account();

# Programming Pitfalls

- 5. Non-inline functions must use the scope resolution operator (::).
- 6. static class data members must be initialised before the start of main().
- 7. The default access of members in a class is private. Don't forget to place public: before member functions that are intended to be public.

# Naming Objects and Classes (为对象和类命名)

- **When you declare a custom class, capitalize the first letter of each word in a class name; (声明一个自定义的类时，类名中的单词要首字母大写)**

  for example, the class names **Circle**, **Rectangle**, and **Desk**.

- **The class names in the C++ library are named *in lowercase*. (C++标准库中的类名是小写的)**
- **The objects are named like variables. (对象的命名方式与变量类似)**

# 代码练习：

- 定义一个盒子类，
包括数据成员长、宽和高等属性，
求体积的成员函数、构造函数和析构函数。

# Q & A

Thank You!