

C++ Programming

Chapter 8 Objects and Classes

Part 1/4

Zheng Guibin
(郑贵滨)



哈爾濱工業大學
Harbin Institute of Technology

目录

Objects and Classes

对象和类

CONTENT

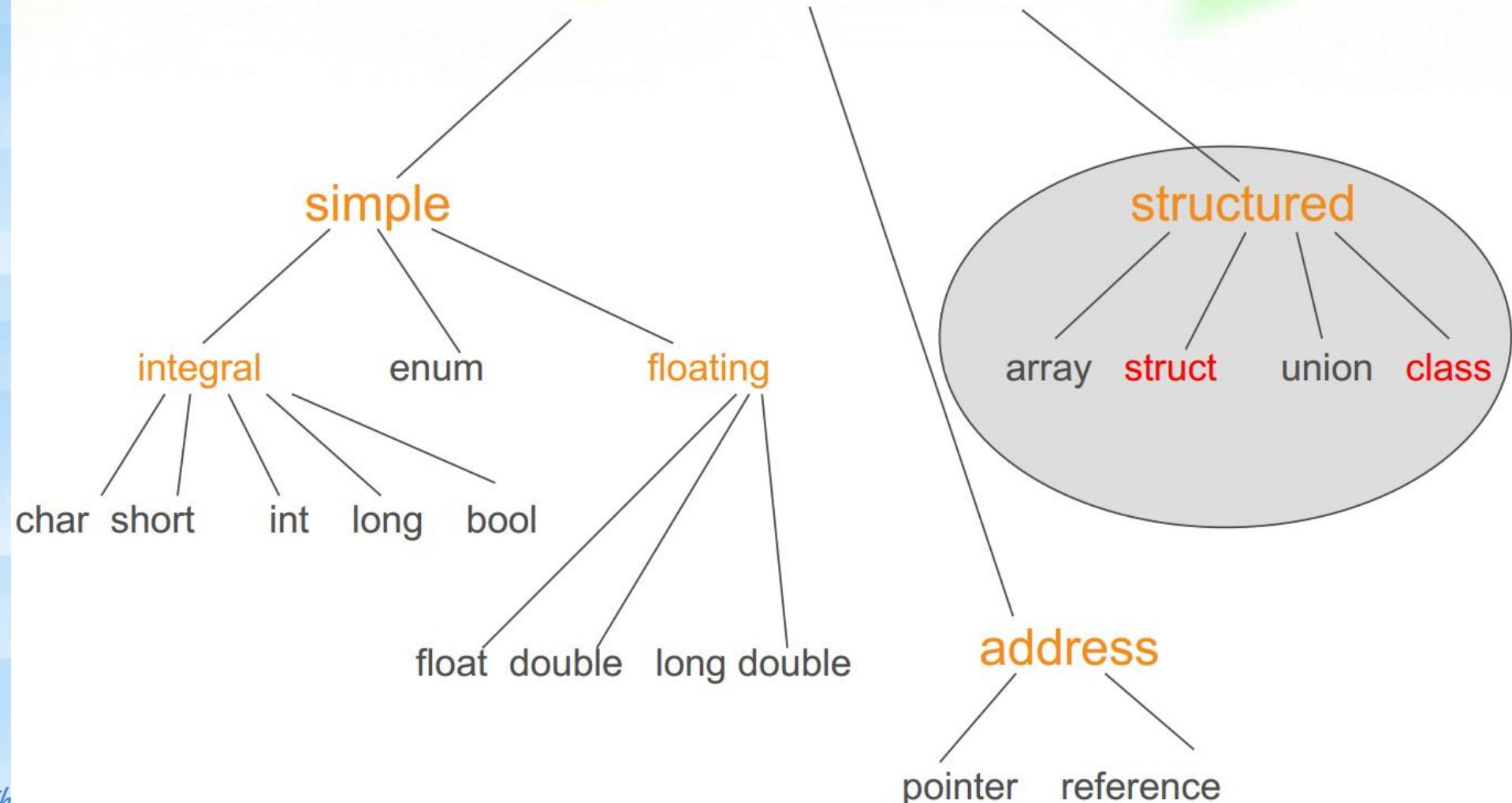
- 从“结构体”到“类”
- 什么是对象、类？
- 类的定义
- 类的使用
- 构造函数
- 类的接口、类的实现
- 析构函数（补充）
- const成员函数（补充）



哈爾濱工業大學
Harbin Institute of Technology

C++的数据类型

C++ Data Types

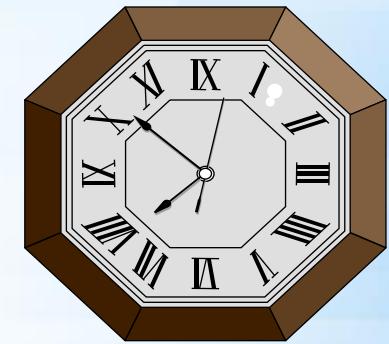


1、从“结构体”到“类”

◆ 1.1 结构体实例

```
struct Time  
{  
    int hour;           // 0-23  
    int minute;         // 0-59  
    int second;         // 0-59  
};
```

时间
如何调整



1.1 结构体实例

```
#include <iostream>
using namespace
std;
struct Time
{
    int hour;
    int minute;
    int sec;
};
```

void set_time(Time& t);
void show_time(Time& t);

```
int main( )
```

```
{
```

```
Time t1;
```

```
set_time(t1);
```

```
show_time( t1 );
```

```
Time t2;
```

```
set_time(t2);
```

```
show_time( t2 );
```

```
return 0;
```

```
}
```

```
#include <iostream>
#include <conio.h>
using namespace std;
struct Time{
    int hour;
    int minute;
    int sec;
};
void set_time( Time& t,  int h, int m, int s ) {
    t.hour = h;  t.minute = m;  t.sec = s;
}
void show_time(Time& t){
    cout<<"CurrentTime:";
    cout<<t.hour<<"<<t.minute<<"<<t.sec<<endl;
}
```

```
int main( )
{
    Time t1, t2;
    int h, m, s;

    cin>> h >>m >>s;
    set_time(t1,h,m,s);
    show_time(t1);
    cin>> h >>m >>s;
    set_time(t2);
    show_time(t2);
    cout<<"\nPress any key to exit...";
    getch(); // cin.get();
    return 0;
}
```

1.2 类的实例

```
class Time
{
```

private:

关于时间的数据；

public:

读取时间值；
调整时间值；

};

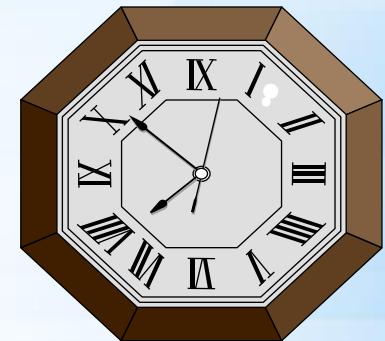
特定的访问权限

外部接口

时间



如何调整



类的构成：

数据成员

成员函数（操作数据的函数）



1.2 类的实例

```
class Time
```

```
{
```

```
private:
```

```
    int hour;
```

```
    int minute;
```

```
    int second;
```

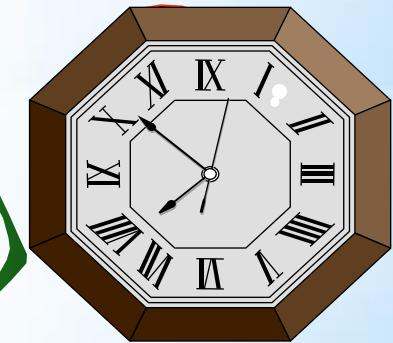
```
public:
```

```
    void set_time(int,int,int);
```

```
    void show_time( );
```

```
};
```

构造类



有自己的方法

对数据操作

```
#include <iostream>
using namespace std;
#include <conio.h>

class Time{
public:
    void
    set_time(int,int,int);
    void show_time( );
private:
    int hour;
    int minute;
    int sec;
};

};
```

```
void Time::set_time(int h,int
m,int s )
{
    hour = h;   minute = m;   sec =
s; }

void Time::show_time( )
{
    cout<<"CurrentTime:";
    cout<<hour<<":"<<minute<<
 ":"<<sec<<endl;
}
```



```
int main( ){
    Time t1, t2;
    int h, m, s;
    cin>> h >>m >>s;
    t1.set_time( h,m,s);
    t1.show_time( );
    cin>> h >>m >>s;
    t2.set_time(h,m,s);
    t2.show_time( );
    cout<<"\nPress any key to exit...";
    getch();
    return 0;
}
```

```
class Time
{
public:
    void set_time( int h,int m, int s){
        hour = h;
        minute = m;
        sec = s;
    }
    void show_time(){
        cout<<hour<<"::"  
             <<minute<<"::"  
             <<sec<<endl;
    }
private:
    int hour;
    int minute;
    int sec;
};
```

练习 : setTime() 加上时间有效性检验



2.2 类的实例

◆ 注意事项

1. 在主函数中调用两个成员函数时，应指明对象名(t1,t2)。表示调用的是哪一个对象的成员函数。
2. 在类外定义函数时，应指明函数的作用域,如：

```
void Time::set_time()
```

在成员函数引用本对象的数据成员时，直接写数据成员名，这时C++系统会把它默认为本对象的数据成员。也可以显式地写出类名并使用域运算符。

3. 注意区分什么场合用域运算符“::”，什么场合用成员运算符“.”，不要搞混。



8.1 What is an object (对象) ?

◆ 对象(object)

程序的组件，知道如何执行特定的动作/操作，如何与程序的其他部分交互。

◆ 对象的构成

- 属性：一个或多个数值，描述对象状态或者属性（静态特征）
- 行为：应用于对象的函数，表示针对该对象可以做什么、如何做（动态特征）

◆ 例子：计算机冒险游戏

- 游戏中的对象：玩家、敌人（龙）、武器；奖励：Fair maiden
- 玩家：

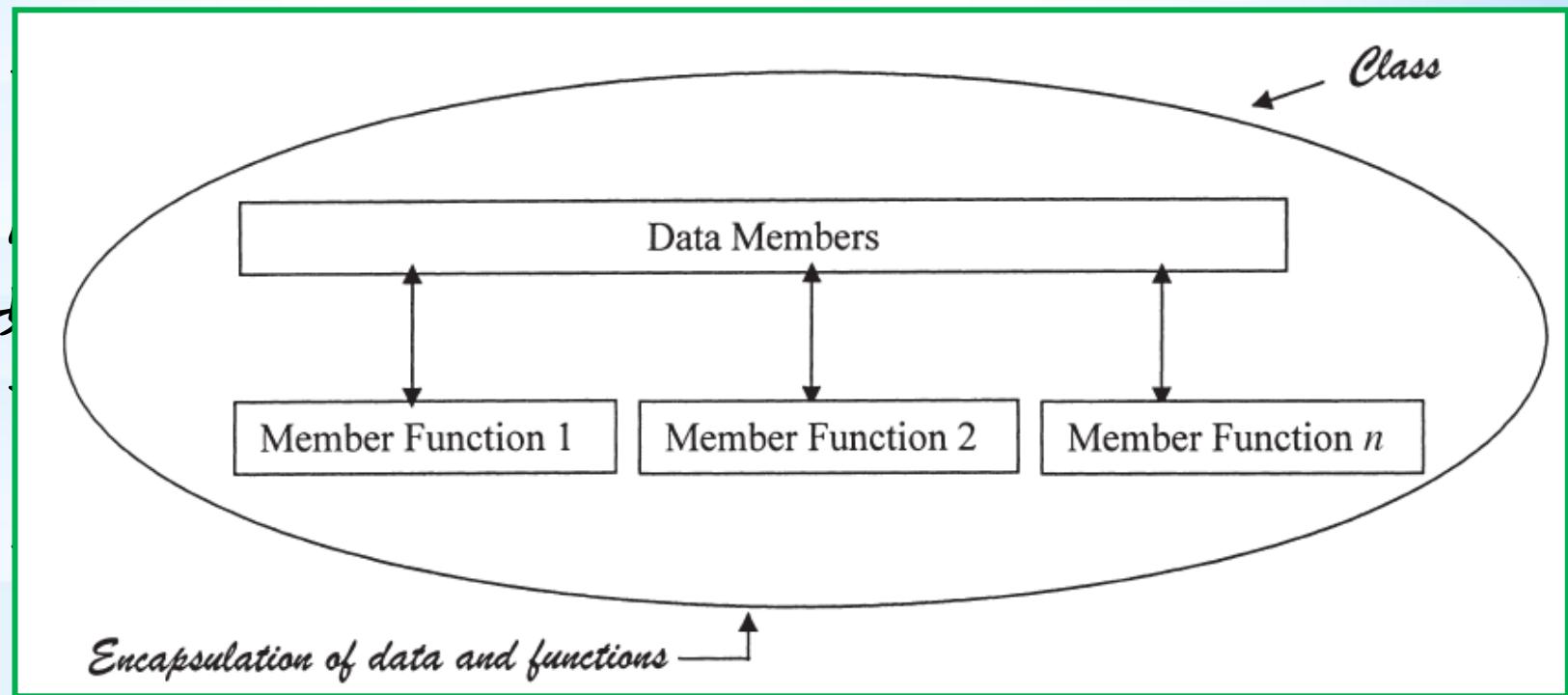
- 数值表示的属性：表示生命状态、拥有的武器等；
- 可以执行的函数：走、跑、攻击一个敌人、营救Fair maiden并获胜



8.2 What is a class? (什么是类?)

◆ 类 (class)

- 从上到下
-
-
-



◆ 封装(encapsulation)

- 类同时定义了数据类型、可用于这些数据的操作，并将它们打包成一个单元 (one unit)。封装 (把对象的属性和服务结合成一个独立的系

8.2 What is a class?

```
class player
{ //functions:
    walk()
    run()
    jump()
    attack()
    rescue()

    ...
//data:
    state_of_health
    type_of_weapon
    ...

};
```

code)

```
class dragon
{ //functions:
    walk()
    spit_fire()
    use_claws()
    use_tail()
    die()

    ...
//data:
    size
    number_of_claws
    state_of_health
    ...

};
```



8.2 What is a class?

◆ 类和类的实例 (instance)

- 属于某类的对象称为该类的一个实例
- 类是蓝图(blueprint)或 模板 (template) , 可用于生成很多该类的实例。
- 实例是从类创建的 实际的对象，并可用类的成员函数操作它们。
- “类”与“类的实例”的关系
 - 犹如 模具与铸件之间的关系
 - 名词(a noun) 和 a 专有名词 (proper noun).
 - person is a noun, John Smith is a proper noun
 - Person is the class and John Smith is the object.



8.2 What is a class?

◆ Example: 冒险游戏

➤ 生成dragon的实例(对象):

```
class dragon george(10,4);
```

➤ 类成员函数的调用:

```
george.spit_fire();
```

➤ 可以生成任意数量的 dragon的对象.

```
class dragon ivan(6,2), baby( 1, 0);
```



8.2 What is a class?

- 每条龙可以执行不同的函数（与其他龙无关）。

```
ivan.use_claws(); //ivan attacks with claws  
ivan.die(); //ivan dragon dies. sorry!  
george.spit_fire(); //george attacks with fire.
```

◆ 概括：

- 对象是类的实例
- 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。



8.3 Further examples of classes and objects

◆ 8.3.1 学生类

➤ 属性 (properties) —— 类的数据成员

- name
- address
- student number
- date of birth
- course
- course mark.

➤ 函数——(类的成员函数/方法)

更新课程分数;
显示进度报告;

```
class student
{
    //functions:
    update_mark()
    display_report()
    //data:
    name
    address
    student_number
    date_of_birth
    course_code
    course_marks
};
```



8.3 Further examples of classes and objects

8.3.2 银行账号类

- 银行账号属性：账号(**account number**)和余额(**balance**.)
- 银行账号的操作：

- open the account with an amount of money
- deposit money
- withdraw money

class member functions:

- open(amount)
- withdraw(amount)
- deposit(amount).

```
class bank_account
{ //functions:
    open( amount )
    deposit( amount )
    withdraw( amount )
```

```
//data:
account_number
balance
```

}

```
class bank_account my_bank_account;
...
my_bank_account.deposit(2000000);
...
my_bank_account.withdraw(20.1);
...
}
```

8.4 Abstraction (抽象)

具有共同特征的事物划分为一组，称为“类”(Class)，是个抽象的概念。

- 分类所依据的原则——抽象，人类通常的思维方法
- 软件中的对象仅仅在某些方面模拟现实世界中的对象；
 冒险游戏的玩家...
- 根据问题的需要，选择性地模拟或者简化。
- 同一现实世界中的对象，问题不同，抽象的结果不同，对应的软件对象也不同。



8.4 Abstraction (抽象)

◆ 抽象(Abstraction)

➤ 抽象的例子：

- 课程表包含科目、教师名字、房间号、时间。
- 课程表中，授课地点房间，简单地用一个号码表示；
- 对于学生：不需要房间的细节，也不会被告知
- 对于学校管理者：房间的尺寸是相关信息，需要在包括在学校管理目标中。



8.5 Constructing a class in C++

◆ 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；
- public部分通常放的private部分的前面。



// Program example P8A

```
#include <iostream> // required later for input-output
#include <iomanip> // required later for manipulators
using namespace std ;
class bank_account
{
public:
    void open( int acc_no, double initial_balance ) ;
    void deposit( double amount ) ;
    void withdraw( double amount ) ;
    void display_balance() ;
private:
    int account_number ;
    double balance ;
};
```

•*private*(私有)、*public* (公有) 设定访问权限

8.5 Constructing a class in C++

- ◆ 类的成员函数必须定义

定义类成员函数的一般格式:

```
return_type class_name::function_name( parameter list )
{
    //function statements.
}
```



```
void bank_account::open( int acc_no, double initial_balance )
{
    account_number = acc_no ;
    balance = initial_balance ;
}

void bank_account::deposit( double amount )
{
    balance += amount ;
}

void bank_account::withdraw( double amount )
{
    balance -= amount ;
}

void bank_account::display_balance()
{
    cout << "Balance in Account " << account_number << " is "
        << fixed << setprecision( 2 )
        << balance << endl ;
}
```

8.5 Constructing a class in C++

◆ 类的信息隐藏 (*information hiding*)

➤ ***private*** (私有) : 声明为私有的成员，只能被类本身的成员函数及友元类的成员函数或友元函数访问，默认类的成员是私有； (*ch11* 介绍友元)

◆ 类的公共接口 (*public interface*)

➤ ***public*** (公有) : 声明为公有的成员，可以被程序中的任何代码访问；

➤ 类的公共接口：声明为公有的成员

◆ 保护 ***protected*** : (*ch11* 介绍)

类本身的成员函数、友元函数或友元类的成员函数、该类的派生类的成员可以访问。



8.5 Constructing a class in C++

可以从类的
外部访问

访问公有成员，全部权限

无法从类的
外部访问

无权访问私有成员

类

public (公有)

函数

数据 (慎重!)

全部
权限

private (私有)

函数

数据

// Program example P8A (C)

```
int main()
```

```
{
```

```
class bank_account my_account ;
```

```
my_account.open( 123, 10.54 )
```

```
my_account.display_balance() ;
```

```
my_account.deposit( 10.50 ) ; // D
```

```
my_account.display_balance() ;
```

```
my_account.withdraw( 20.04 ) ; // Withdraw 20.04
```

```
my_account.display_balance() ;
```

```
return 0;
```

```
}
```

与用结构体实现相比较，优缺点？

定义一个类后，就可以用类名
声明类类型的变量，即将类实
例化为不同的对象

| my_account | |
|------------|---------|
| account_no | balance |
| ? | ? |

| my_account | |
|------------|---------|
| account_no | balance |
| 123 | 10.54 |

```
Balance in Account 123 is 10.54  
Balance in Account 123 is 21.04  
Balance in Account 123 is 1.00
```

8.6 Using a class: defining and using objects

- ◆ 类的实例化即定义类的对象，类似于定义任何类型的变量（类名称为一个新的类型说明符）

定义对象有三种形式

- (1) 先声明类类型，后定义对象

```
class Tdate{...};//类Tdate的定义
```

....

```
class Tdate d1,d2; // 把class和Tdate合起来作为一个类名
```

```
Tdate d1,d2; // 直接用类名定义对象。
```



8.6 Using a class: defining and using objects

◆ (2) 在声明类类型的同时定义对象

```
class Tdate
{ public:
    void Set(int m, int d, int y ){...}
    int IsLeapYear(){...}
    void Print( ){...}
private:
    int month;
    int day;
    int year;
} d1, d2;
```



8.6 Using a class: defining and using objects

- ◆ (3) 不出现类名，直接定义对象

```
class //无类名
{
    private: //声明以下部分为私有的
    ...
    public: //声明以下部分为公用的
    ...
}d1, d2; //定义了两个无类名的类对象
```

编译系统会为这个对象分配存储空间，以存放对象中的成员。

8.6 Using a class: defining and using objects

- ◆ 允许已定义类名出现在类的说明中

```
class X
{
    .....
};

class Y
{
    X dataMember; // 声明一个类类型数据成员
    .....
};
```



8.6 Using a class: defining and using objects

- ◆ 允许已定义类名出现在类的说明中

例:

```
class X
```

```
{
```

```
X dataMember ; // 错误
```

```
.....
```

```
X *m_pNext;
```

```
};
```

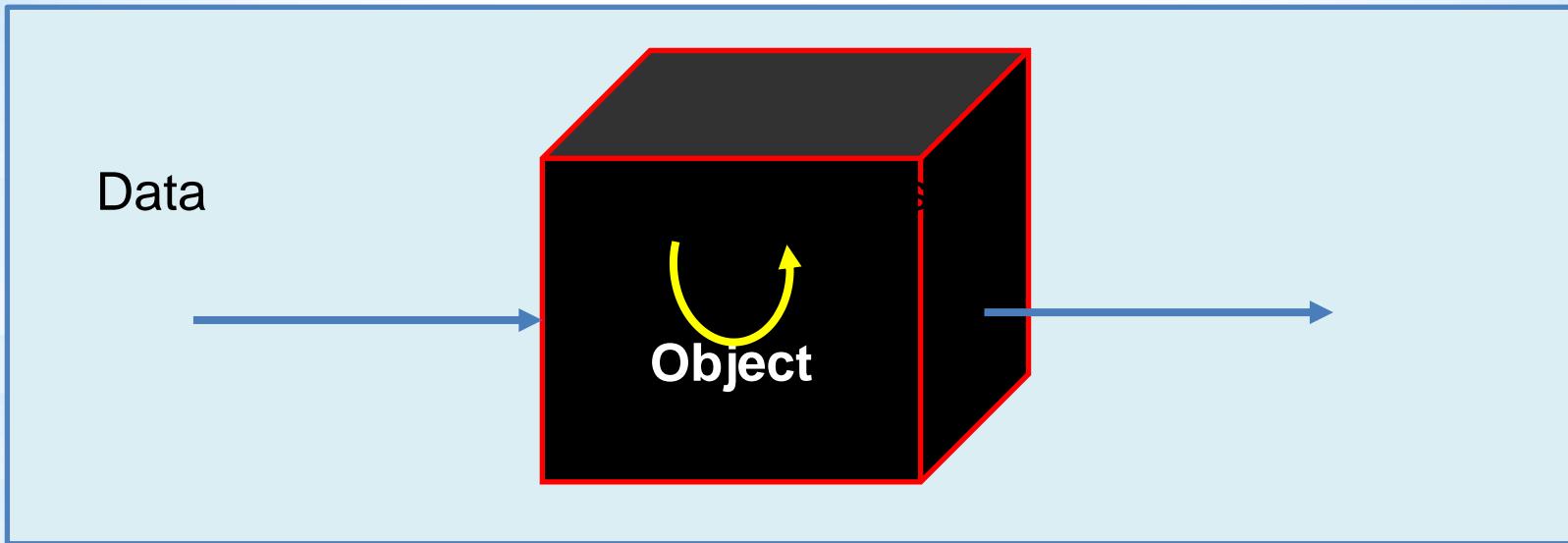
错误:
无穷递归结构

8.7 Abstract data types (抽象数据类型)

- ◆ C++ 内置数据类型(built-in data types)
 - 每种数据类型都有自己的取值范围、一组允许的操作、可以使用的函数；
 - 数据类型的实现细节对程序员是隐藏的，称为：数据抽象 *data abstraction*.
- ◆ 抽象数据类型(abstract data type, ADT)
 - 类（银行账号）是用户自定义类型（非C++内置类型），称为抽象数据类型：
 - 使用类时，只需要知道公共接口即可，如open(), deposit()and withdraw();
 - 使用类时，不需要知道内部实现细节（成员函数、数据成员...）。



8.7 Abstract data types (抽象数据类型)



8.8 Constructors (构造函数)

◆ 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**
- Example:
 - *bank_account* 类无构造函数，通过函数 *open()* 对私有的 *account_number*、*balance* 赋初值
 - 可为类 *bank_account* 增加构造函数，替代成员函数 *open()*



```
class bank_account
{
public:
    bank_account( int acc_no, double initial_balance ) ;
    void deposit( double amount ) ;
    void withdraw( double amount ) ;
    void display_balance() ;

private:
    int account_number ;
    double balance ;
} ;
```

构造函数不会被显式调用，
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。

```
bank_account::bank_account( int acc_no, double initial_balance )
{
    account_number = acc_no ;
    balance = initial_balance ;
}
```

8.8 Constructors

```
main()
{
    bank_account my_account( 123, 10.54 ) ;

    my_account.display_balance() ;

    my_account.deposit( 10.50 ) ;

    my_account.display_balance() ;

    my_account.withdraw( 20.04 ) ;

    my_account.display_balance()
}
```

8.9 Default class constructor

◆ 缺省构造函数 (default class constructor)

```
class bank_account
{
public:
    bank_account() ;
    bank_account( int acc_no, double initial_balance ) ;
    void deposit( double amount ) ;
    void withdraw( double amount ) ;
    void display_balance() ;
private:
    int account_number ;
    double balance ;
};
```

8.9 Default class constructor

```
bank_account::bank_account()
{
    account_number = 0 ;
    balance = 0.0 ;
}
```

```
bank_account::bank_account( int acc_no, double initial_balance )
{
    account_number = acc_no ;
    balance = initial_balance ;
}
```

```
void bank_account::deposit( double amount )
{
    balance += amount ;
}
```

8.9 Default class constructor

main()//Line 49

{

bank_account my_account ; **自动调用构造函数bank_account ()**

my_account.display_balance() ;

my_account.deposit(10.50) ;

my_account.display_balance() ;

my_account.withdraw(20.04) ;

my_account.display_balance() ;

} //Line 62

| my_account | |
|------------|---------|
| account_no | balance |
| 0 | 0.0 |

Balance in Account 0 is 0.00
 Balance in Account 0 is 10.50
 Balance in Account 0 is -9.54

8.10 Overloading class constructors

——重载类构造函数

◆ 构造函数可以重载(overloaded)

类可以有多个构造函数，每个构造函数有不同数量的参数，Program Example P8D

```
class bank_account{  
public:  
    bank_account();  
    bank_account( int acc_no );  
    bank_account( int acc_no, double initial_balance );  
    void deposit( double amount );  
    void withdraw( double amount );  
    void display_balance();  
....  
};
```

8.11 Constructor initialisation lists

- ◆ 构造函数中，使用数据成员初始化列表替代赋值语句：
 - 简化程序书写；
 - 必须用初始化列表的情况 (*const*, 引用, 成员对象...)

```
bank_account::bank_account( int acc_no, double initial_balance )
{ account_number = acc_no ;
  balance = initial_balance ;
}
```



```
bank_account::bank_account (int acc_no, double initial_balance )  
  account_number( acc_no ), balance( initial_balance )
```

{}

• 无赋值语句,大括号(chain brackets) {}不可少

8.12 Default argument values in a constructor

◆ 构造函数中使用默认实参值

```
class bank_account //Program Example P8E
{
public:
    bank_account() ;
    bank_account( int acc_no, double initial_balance = 0.0 ) ;
    void deposit( double amount ) ;
    void withdraw( double amount ) ;
    void display_balance() ;
private:
    int account_number ;
    double balance ;
} ;
```

构造函数使用说明

- ◆ 默认构造函数/缺省构造函数 (default constructor)
系统自动生成的构造函数和无参构造函数属于默认构造函数；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
- ◆ 如果在建立对象时用无参构造函数，或者全部采用有参构造函数的默认参数值时，注意定义对象的语句，不加括号

bank_account accout3; //OK

bank_account accout1(); // error

- ◆ 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

构造函数使用说明....

- ◆ 当构造函数中有默认参数时，注意函数重载的二义性。

...

bank_account() ;

~~**bank_account(int acc_no) ; //会和下面的函数混淆**~~

bank_account(int acc_no, double initial_balance = 0.0) ;

....

bank_accout accout1;

bank_accout account2(12,15);



补充：Anonymous Object (匿名对象)

- ◆ Occasionally, you may create an object and *use it only once*. In this case, you don't have to name the object. Such objects are called anonymous objects.

有时需要创建一个只用一次的对象。此时，无需给对象命名。这种对象叫做匿名对象。

`bank_account(); //using the no-arg`

`bank_account(12,15); //using the constructor with arguments`



课堂练习...

定义一个表示日期的类Date，包含year, month, day 三个属性，给出对应的构造函数，要求用这个Date类定义对象的时候，用户可以提供零个、一个、两个或三个参数。

并给出以下成员函数的定义（设计合理的函数返回值类型、参数数量和类型）：

setDate() //设置日期数值

showDate() //显示日期数值

getYear() //获得日期的“年”数值

getMonth() //获得日期的“月”数值

getDay() //获得日期的“日”数值

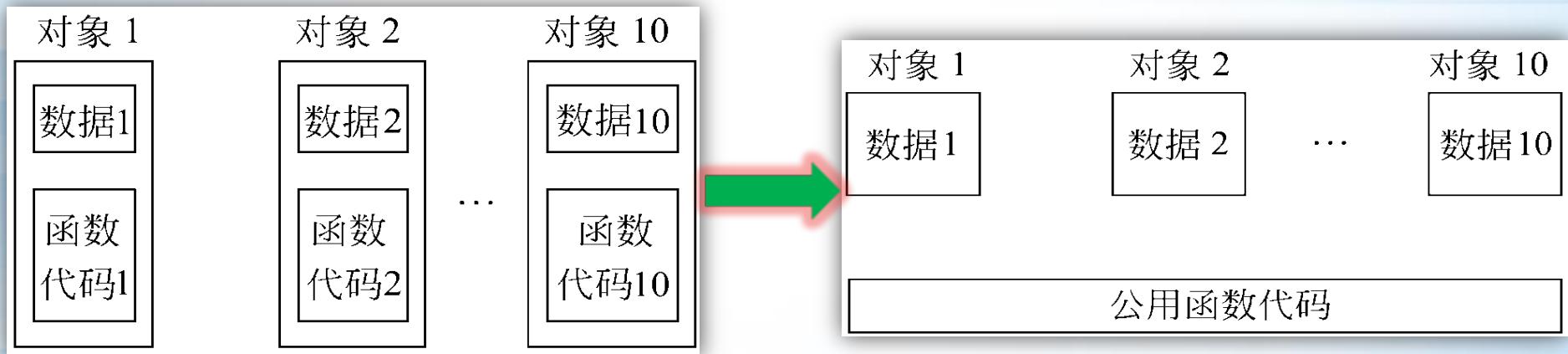
• 二、Timedate 带日期的时间类



类对象的存储

◆ 成员的存储方式

用类去定义对象时，系统会为每一个对象分配存储空间。



```
cout<<sizeof(bank_account)<<endl;
```

类和OOP

- ◆ 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
- ◆ 面向对象的方法：
 - 面向对象的分析 (OOA)
问题域中客观存在的事物 => 对象
 - 面向对象的设计 (OOD)
增加与“实现”有关的部分
 - 面向对象的编程 (Object-Oriented Programming,OOP)
编程实现，方法：用软件模拟“类”、“对象”
 - 面向对象的测试 (OOT)
 - 面向对象的软件维护 (OOSM)

