2024年春

# 程序设计实习：C++程序设计

# 第三讲 类和对象（1）

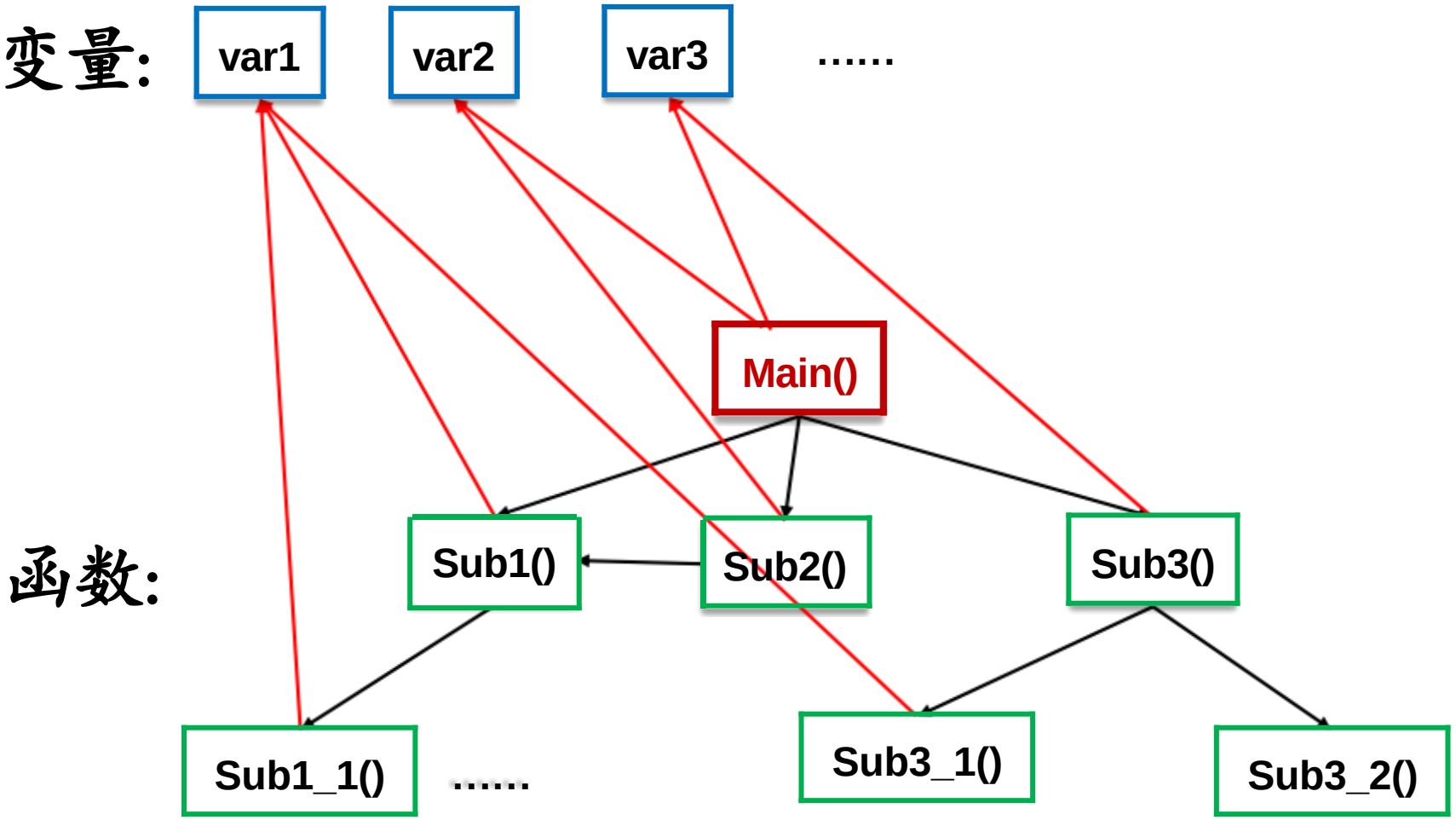
贾川民
北京大学

# 主要内容



- 面向对象的**基本概念**
  - 例子 — 矩形类
  - 三种方式使用
  - 引用 & 常引用
  - 类成员的访问权限

# 结构化程序设计



# 面向对象的程序设计

- **面向对象的程序设计方法：**

  将某类客观事物**共同特点**(**属性**) 归纳出来

  → 形成一个**数据结构**(用多个变量描述事物的属性)

  将这类事物所能进行的**行为**也归纳出来

  → 形成一个个**函数**

  → 这些函数可以用来操作数据结构

**C++关键字：抽象**

# 面向对象的程序设计

- 通过某种语法形式，

将**数据结构**和操作该数据结构的**函数**“捆绑”在一起 → 形成一个“类”

- 使得数据结构和操作该数据结构的算法呈现出显而易见的紧密关系

**C++关键字：封装**

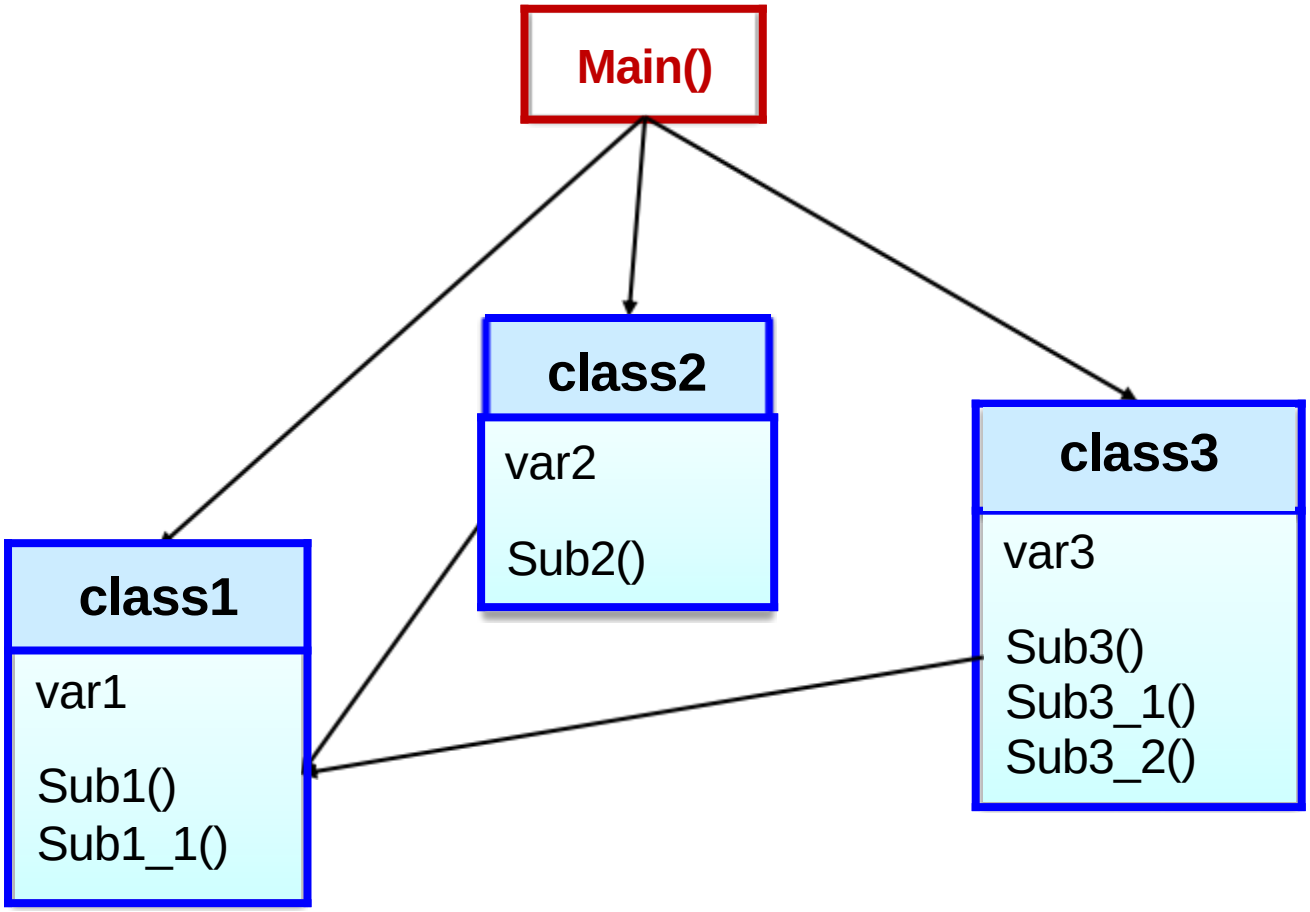
# 面向对象的程序设计

- 面向对象的程序设计方法，能够较好解决上述问题

  面向对象的程序 = 类 + 类 + … + 类

- 设计程序的过程，就是设计类的过程

# 面向对象的程序模式


Main()
class2
var2
Sub2()
class3
var3
Sub3()
Sub3_1()
Sub3_2()
class1
var1
Sub1()
Sub1_1()

# 从客观事物抽象出类

- **写一个程序，输入矩形的长和宽，输出面积和周长**
  - 如对于 "矩形" 这种**对象**，要用一个**类**来表示，该如何做 "抽象" 呢？
  - 矩形的**属性**就是长和宽
    - 因此需要两个变量，分别代表长和宽
  - 可以对矩形进行哪些**操作**
    - 矩形可以有设置长和宽，计算面积和计算周长这三种行为
    - 这三种行为，可以各用一个**函数**来实现，都需要用到长和宽这两个变量

# 从客观事物抽象出类

- 将长，宽变量和设置长，宽，求面积，以及求周长的三个函数封装在一起，就能形成一个"矩形类"
  - 长和宽变量成为该"矩形类"的"成员变量"
  - 三个函数成为该类的"成员函数"
  - 成员变量和成员函数统称为类的成员
- 类 → 带函数的结构

# 从客观事物抽象出类



```
class CRectangle
{
      public:
            int w, h;
            int Area()  {
                  return w  * h;
            }
            int Perimeter(){
                  return 2  *  (w + h);
            }
            void Init(int w_, int h_)  {
                  w = w_;   h = h_;
            }
}; //必须有分号
```

# 从客观事物抽象出类

```
int main( )  {
   int w, h;
   CRectangle   r; //r是CRectangle类的一个对象
   cin >> w >> h;
   r.Init(w, h);
   cout << r.Area() << endl << r.Perimeter();
   return 0;
}
```

# 对象

- 通过类，可以定义变量
- 类定义出来的变量，也称为**类的实例**，即**“对象”**
- C++中，类的名字就是用户自定义的类型的名字

可以像使用基本类型那样来使用它

  - CRectangle 就是一种**用户自定义**的类型

# 对象的内存分配

- 和结构变量一样，**对象**所占用的内存空间的大小，

  = 所有成员变量的大小之和

- 对于上面的**CRectangle**类，**sizeof(CRectangle) = 8**
- 每个对象各有自己的存储空间
- 一个对象的某个成员变量被改变了，不会影响到另一个对象

# 对象间的运算

- 和结构变量一样，对象之间可以用 "=" 进行赋值
- 不能用 "==" "!=" ">" "<" ">=" "<="进行比较，除非这些运算符经过了 "重载"

# 使用类的成员变量和成员函数

用法1: 对象名.成员名

```
CRectangle r1, r2;
r1.w = 5;

r2.Init(3,4);
```

- Init函数作用在 r2上, 即Init函数执行期间访问的w和h是属于r2这个对象的, 执行r2.Init不会影响到r1

# 使用类的成员变量和成员函数

用法2: 指针->成员名

```
CRectangle  r1, r2;
CRectangle * p1 = & r1;
CRectangle * p2 = & r2;

p1->w = 5;
p2->Init(3,4); //Init作用在p2指向的对象上
```

# 使用类的成员变量和成员函数

用法3: 引用名.成员名

```
CRectangle r2;
CRectangle & rr = r2;
rr.w = 5;
rr.Init(3, 4); //rr的值变了， r2的值也变
```

# 引用

**类型名 & 引用名 = 某变量名;**

- 定义了一个引用，并将其初始化为引用某个变量
- 某个变量的引用，和这个变量是一回事，相当于该变量的一个别名

```
int n = 4;
int & r = n;
r = 3;
cout << r; //输出 3
cout << n; //输出 3
n = 5;
cout << r; //输出5
```

- 定义引用时一定要将其初始化成引用某个变量，不初始化编译不过
- 引用只能引用变量，不能引用常量和表达式

# 引用的引用

- C语言中，交换两个整型变量值的函数，只能通过**指针**

```
void swap( int * a, int * b )
{
     int tmp;
     tmp = * a; * a = * b; * b = tmp;
}
int n1, n2;
swap(&n1, &n2) ;     // n1, n2的值被交换
```

# 引用的引用



- 使用引用:

```
void swap( int & a, int & b )
{
      int tmp;
      tmp = a; a = b; b = tmp;
}
int n1, n2;
swap(n1, n2)  ; // n1, n2的值被交换
```

# 引用作为函数的返回值

- 函数的返回值可以是引用,如:

```
#include <iostream>
using namespace std;
int n = 4;
int & SetValue()   {     return n;   }
//返回对n的引用
int main()
{
      SetValue() = 40; //对返回值进行赋值   对n赋值
      cout << n;
      return 0;
} //程序输出结果是40
```

# 使用类的成员变量和成员函数

用法3: 引用名.成员名

```
CRectangle r2;
CRectangle & rr = r2;
rr.w = 5;
rr.Init(3, 4); //rr的值变了,r2的值也变

void PrintRectangle(CRectangle & r)
{
   cout << r.Area() << "," << r.Perimeter();
}
CRectangle r3;
r3.Init(3, 4);
PrintRectangle(r3);
```

# 常引用

- 定义引用时，
  在前面加 const关键字 — 该引用为“常引用”

```
int n;
const int & r = n;
```

不能通过常引用去修改其引用的内容

```
int n = 100;
const int & r = n;
r = 200; //编译出错，
         //不能通过常引用修改其引用的内容
n = 300; //没问题，n的值变为300
```

# 常引用

- 请注意, **const T &** 和 **T &** 是不同的类型
- **T &**类型的引用或 **T**类型的变量可以用来初始化 **const T &**类型的引用
- **const T** 类型的常变量和**const T &**类型的引用则不能用来初始化**T &**类型的引用, 除非进行强制类型转换

# 类的成员函数的另一种写法

- 成员函数体和类的定义分开写

```
class CRectangle
{
    public:
        int w, h;
        int Area(); //成员函数仅在此处声明
        int Perimeter();
        void Init( int w_, int h_ );
};
```

# 类的成员函数的另一种写法

```
int CRectangle::Area()   {
      return w * h;
}
int CRectangle::Perimeter() {
      return 2 *  ( w + h);
}
void CRectangle::Init( int w_, int h_ ){
     w = w_;  h = h_;
}
```

- CRectangle::说明后面的函数是CRectangle类的成员函数，而非普通函数
- 一定要通过对象或对象的指针或对象的引用才能调用

# 类成员的访问权限

- 结构化程序设计 *Vs.* 面向对象程序设计 → **封装**
- 类成员的**可访问范围**
  - 用**访问范围关键字**来说明类成员可被访问的范围：
    - **private: 私有成员，只能在成员函数内访问**
    - **public: 公有成员，可以在任何地方访问**
    - **protected: 保护成员**

```
class className {
  private:
    //私有属性和函数
  public:
    //公有属性和函数
  protected:
    //保护属性和函数
};
```

# 类成员的访问权限

- 类成员的**可访问范围**
  - 如过某个成员前面没有上述关键字，则**缺省地被认为是私有成员**

```
1  class Man {
2    int nAge; //私有成员
3    char szName[20]; // 私有成员
4  public:
5    void SetName(char * szName){
6      strcpy( Man::szName,szName);
7    }
8  };
```

以上三种关键字**出现的次数**和**先后次序**都没有限制

# 类成员的访问权限



- **在类的成员函数内部，能够访问：**
  - 当前对象的全部属性，函数
  - 同类其它对象的全部属性，函数
- **在类的成员函数以外的地方，只能够访问该类对象的公有成员**

```
class CEmployee   {
    private:
        char szName[30]; //名字
    public:
         int salary; //工资
         void setName(char * name);
         void getName(char * name);
         void averageSalary(CEmployee e1,
CEmployee e2);
};
void CEmployee::setName(char * name)   {
      strcpy( szName, name); //ok
}
void CEmployee::getName(char * name)   {
     strcpy( name, szName); //ok
}
```

```
void CEmployee::averageSalary(CEmployee e1,
CEmployee e2)
{
    salary =  (e1.salary + e2.salary)/2;
}
int main()
{
    CEmployee e;
    strcpy(e.szName,"Tom1234567889");//编译错,
                                    //不能访问私有成员
    e.setName("Tom");    // ok
    e.salary = 5000;     // ok
    return 0;
}
```

```
int main(){
    CEmployee e;
    strcpy(e.szName,"Tom1234567889");
    //编译错, 不能访问私有成员
    e.setName( "Tom"); // ok
    e.salary = 5000;    // ok
    return 0;
}
```

- 设置私有成员的**目的**
  - ① 强制对成员变量的访问一定要通过成员函数进行
  - ② 方便修改成员变量的类型等属性→只需更改成员函数

  **Vs.** 否则, 所有直接访问成员变量的语句都需要修改
- 设置私有成员的机制叫**隐藏**

- 例如,如将上述程序 → 内存空间紧张的手持设备上

将 **szName** 改为 **char szName[5],**

若**szName**不是私有,那么就要找出所有类似

```
strcpy(man1.szName, "Tom1234567889");
```

这样的语句进行修改,以防止数组越界 → 太麻烦!

- 如果将szName变为私有,

那么程序中就不会出现 (除非在类的内部)

```
strcpy(man1.szName, "Tom1234567889");
```

所有对**szName**的访问都是通过成员函数来进行,

例如:

```
man1.setName( “Tom12345678909887” );
```

- 如需将**szName**改短,也不需将上面的语句找出来修改,只要改**setName成员函数**,在里面确保不越界即可

# 用struct定义类

## 用struct定义类

```
1  struct CEmployee {
2    char szName[30]; //公有!!
3  public :
4    int salary; //工资
5    void setName(char * name);
6    void getName(char * name);
7    void averageSalary(CEmployee e1, CEmployee e2);
8  };
```

和用 class 的唯一区别，就是**未说明**是公有还是私有的成员，就是**公有**

# Thanks!