C++概述

Andrew Huang<bluedrum@163.com>

目标

- 传统编程的缺点
- **对象**
- **类**
- **抽象**
- **继承**
- **封装**
- **多态性**

面向对象的方法 3-1

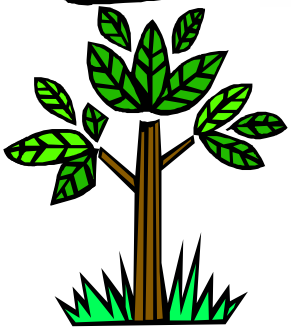
- 二十世纪七十年代发展起来的
- 结构化编程的解决方案
- 模拟人类的思维过程
- 将数据当作单个“对象”进行操作

面向对象的方法 3-2

- 现实世界中所有的事物都是对象
- 对象都具有某些特征，并展现某些行为

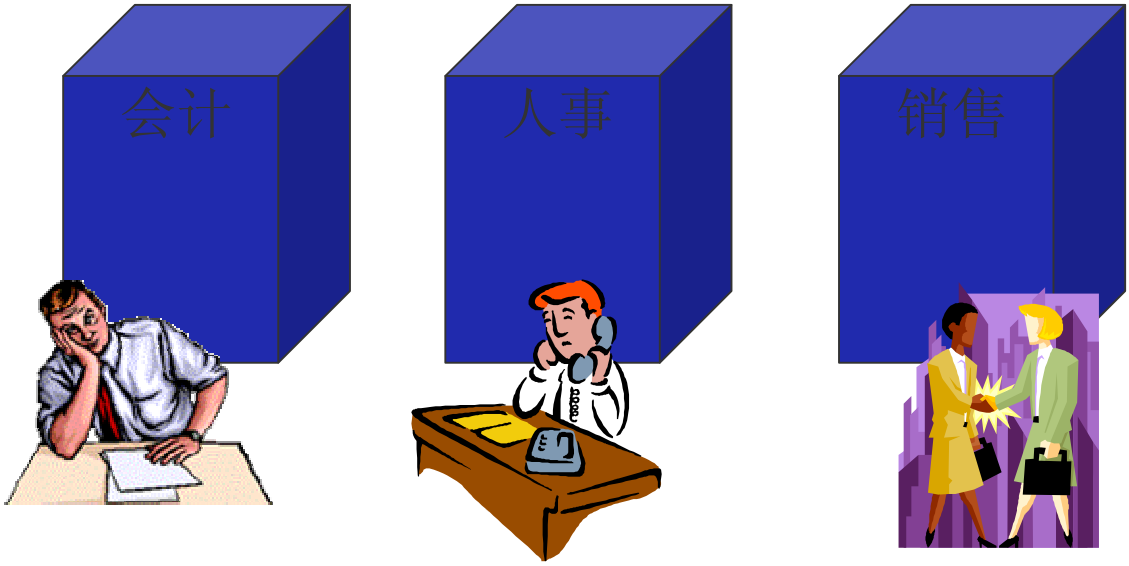






面向对象的方法 3-3

- 公司中的不同部门都是对象

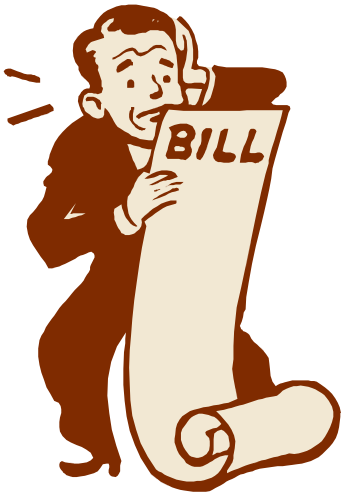


面向对象的语言

- 主要的面向对象语言
  - C++
  - Smalltalk
  - Eiffel
  - C#
  - Java

传统面向过程编程的缺点

- 程序难以管理
- 数据修改存在问题
- 难以实现
- 以函数为单元编程

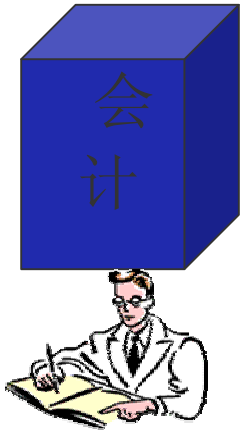


面向对象的编程

- 按照实体在现实世界中的表现来实现
- 将活动和属性与每一实体相关联

数据

员工详细资料
工资结算表
票据
凭证
收据



函数

计算工资
支付工资
支付帐单
记帐
银行交易

面向对象的基本概念

- 对象
  - 帮助理解现实世界
  - 为计算机应用提供实践基础
- 类
  - 描述一组相关对象
- 属性
  - 对象的特征，也称为特性]
  - 类中属性也称为成员
- 函数
  - 类中的函数称为方法(method)
  - 对象执行的活动

对象 4-1

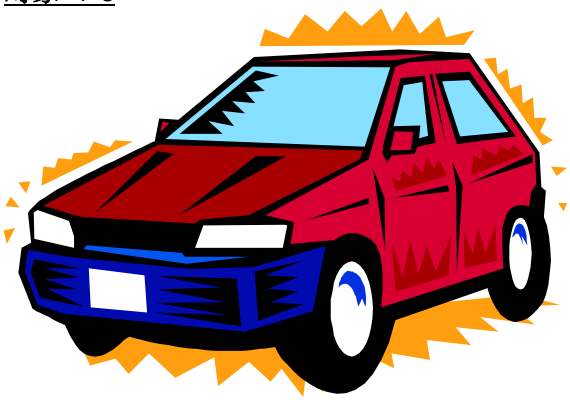
- 具有确定边界，且与所要处理的问题相关的概念或事物

- 对象的用途
  - 帮助理解现实世界
  - 为计算机应用提供实践基础

对象 4-2

- 计算机用户环境中的元素
  - 窗口
  - 菜单
- 数据集合
  - 机器零件清单
  - 员工档案
- 用户自定义的数据类型
  - 时间
  - 角度
  - 复数

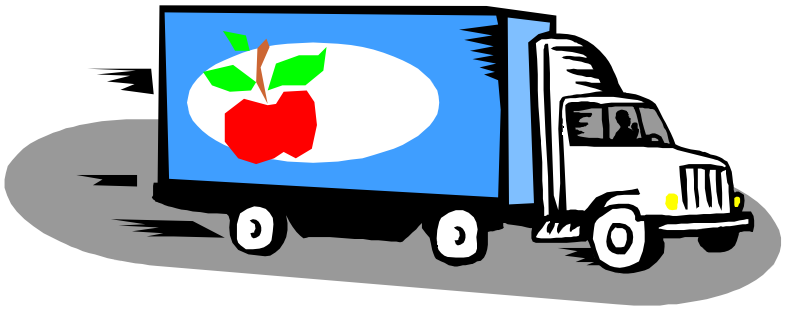
对象 4-3



车型：法拉利
颜色：红色
年份：1995
活动
发动
停车
加速

对象 4-4

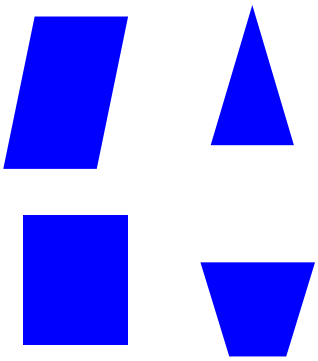
- 对象有其自身的属性，而且可以进行某些活动



活动	属性
停车	颜色
发动	重量
加速	年份
倒车	发动机功率

类

多边形对象



抽象为

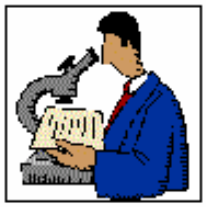
多边形类

属性
顶点
边的颜色
填充颜色

方法
绘制
擦除
移动

抽象性 3-1

- 考察特定应用程序相关问题的某些方面的过程



属性 1

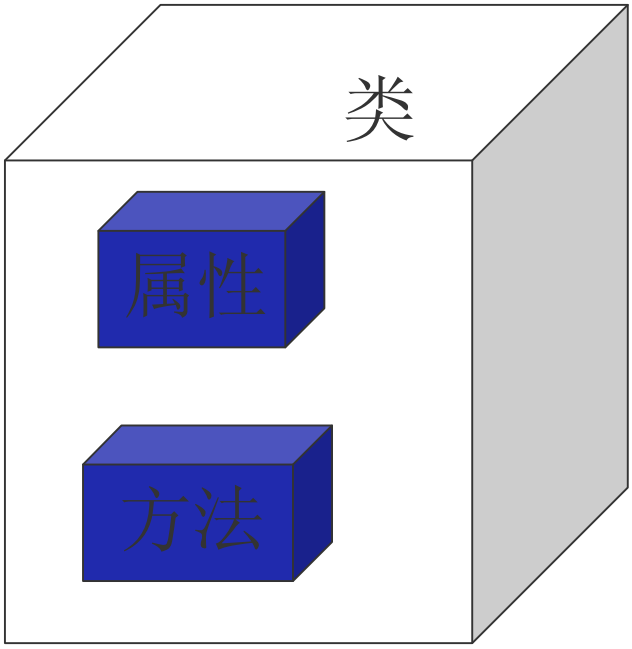
属性 3

方法 1

方法 2

抽象 3-2

- 数据抽象是识别特定实体的属性和方法的过程

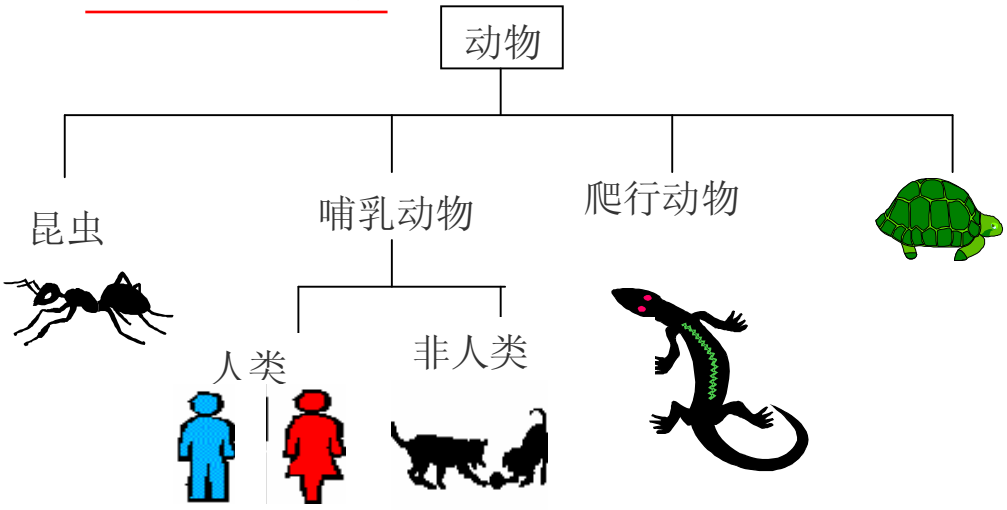


抽象 3-3

- 抽象分为两类
  - 数据抽象
    - 识别与特定的应用程序相关的属性
  - 过程抽象
    - 将注意力集中在过程的参数和返回值，而不是实现

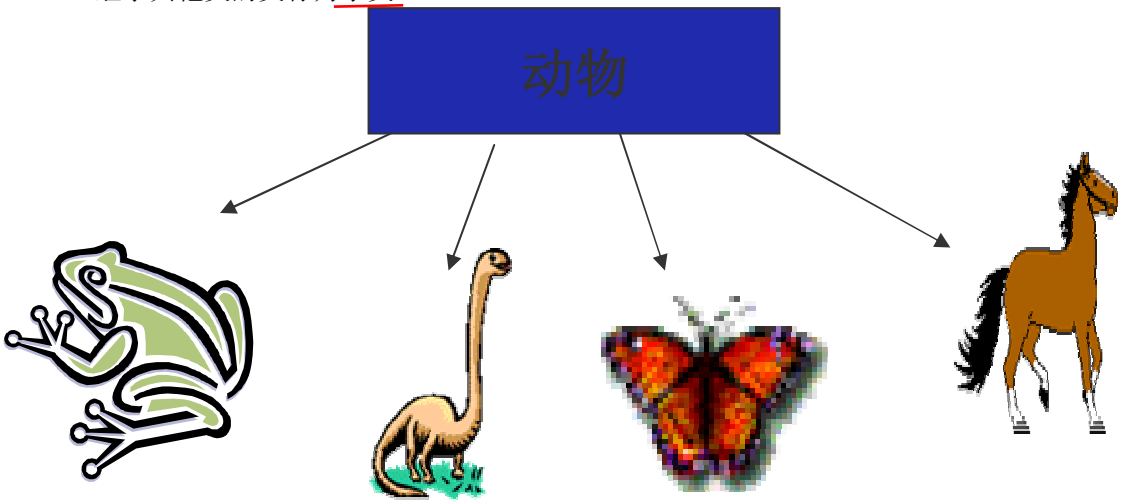
继承性2-1

- 继承重用现有类生成新类
- 可重用性通过继承来实现



继承 2-2

- 超类是有其他类继承其行为的类
- 继承其他类的类称为子类

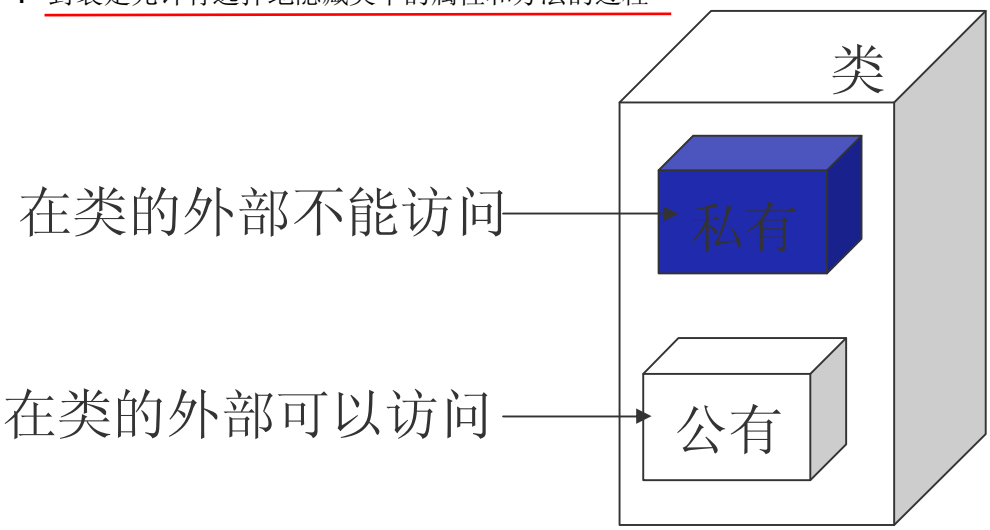


封装性 2-1

- 信息隐藏的过程
- 有选择的数据隐藏
- 防止意外的数据破坏
- 更易于隔离和修复错误

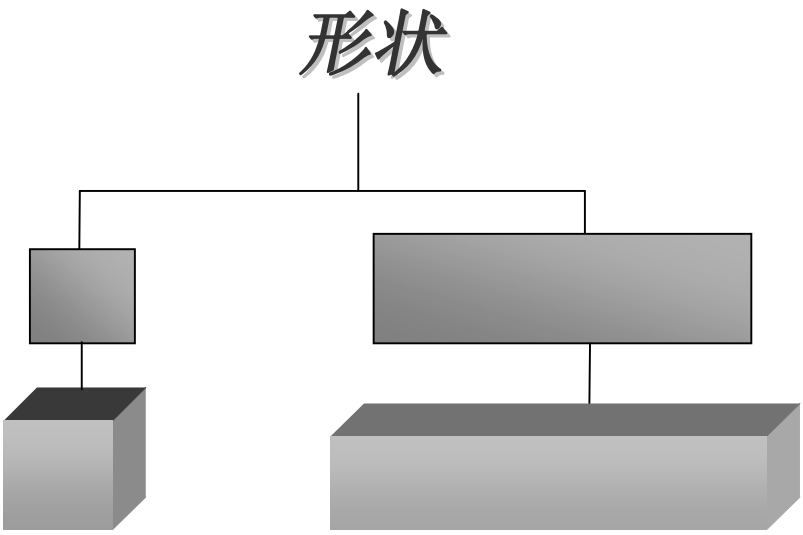
封装 2-2

- 封装是允许有选择地隐藏类中的属性和方法的过程

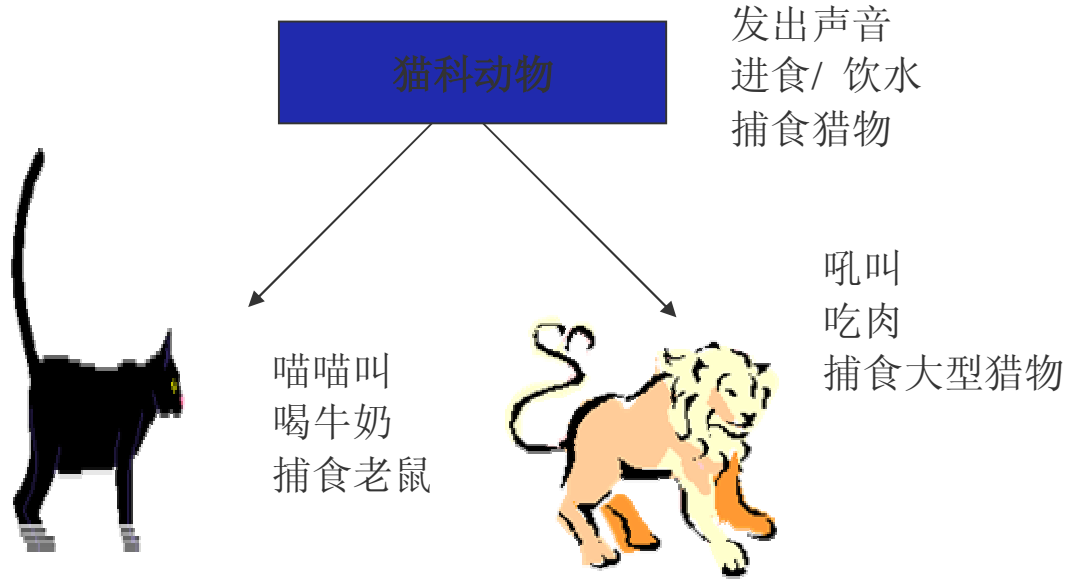


可重用性 2-1

- 程序可以分解为可重用的对象
- 现有类可以和附加功能一起使用

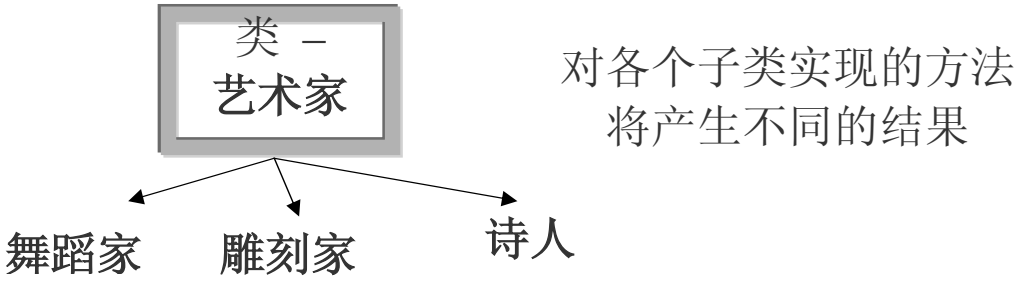


可重用性 2-2

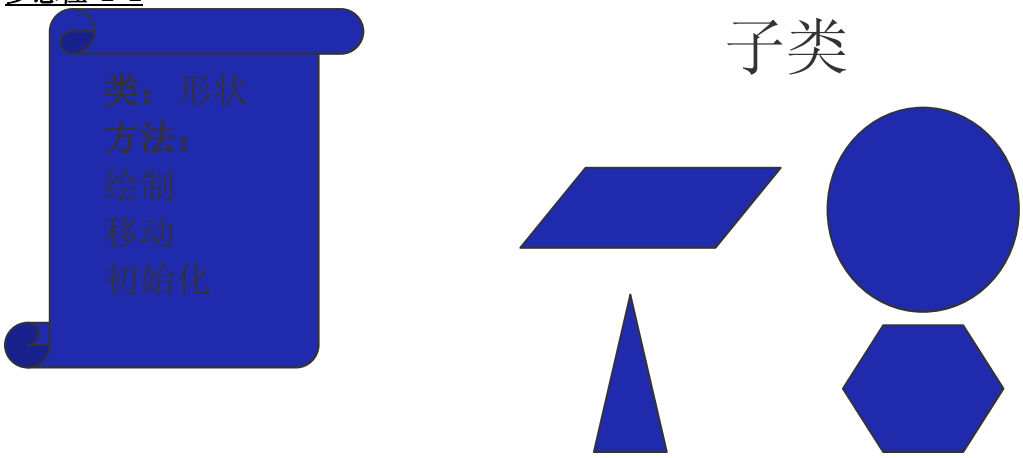


多态性 2-1

- 同样的函数在不同的类上有不同的行为



多态性 2-2



对象模式

- 对象就是思维中的概念，对象模式就是思维中的规律，定律，关系。
  - 对象组织关系
- 常见的对象模式有23种，参见《对象模式设计》
- SingleTon模式---只能建立一个对象实例变量
- 模板模式---为解决问题提供了一个模板，如函数指针做函数参数，在C++则是虚函数
- 装修模式---可在一个对象上构建另一对象。

关于类的更多内容

- 类包含
  - 数据成员
  - 函数
- 数据成员是通过函数访问的
- 对象是类的实例
- 类可以拥有其它类无法访问的部分

类定义

```
 #include <iostream.h>
class Animal
{
   public: int noOfLegs;
   public: char name[10];
   private: char gender;

   public: void showData()
   {
    cout << "名称：" << name;
    cout << "腿的数目：" << noOfLegs ;
    cout << "性别：" << gender;
   }
};
```

C++简介

- 由AT&T贝尔实验室的Bjarne Stroustrup开发
- 从C语言派生的
- 与C语言是兼容的
- 使用编译时绑定

C++简介(2)

- C++语言的ISO标准已在1997年11月被一致通过，1998年8月被正式批准。
- C++多了二个++表示对C的扩展．后来被M$学会,开发出C#,英文发音是C Sharp,中国人习惯叫C井.意思是 C后跟了++++
- 按这个命名规则,下一门语言可能是C+#+ 或者是 C爽(表示C++++++).

C++中的保留字

- class
- friend
- virtual
- inline
- private
- public
- protected
- const
- this
- new
- delete
- operator

C++对C语言的改进

- C++在增加面向对象的特征之外，还对C语言进行了扩充和增强。主要的增强点有如下几个：
  - 注释
    - C 采用/* */ 来注释,C++ 可以采用// 来注释，后被引入C 语言.
    - 以行限定符// 开始，直到本行的末尾，这种新的注释是C++ 特有的
  - 更加灵活的变量说明
    - C++ 变量可以允许在代码语句之后的定义变量，这样更加灵活，机动
    - for (int i=0;i<12;i++)
  - 一个枚举名是类型名，在使用枚举类型之前可以不使用enum
  - 结构体和类名之前不必使用限定词struct 或class,C 不允许这种写法

- 更加严格的函数原型说明 ，不准使用C 传统函数声明
- 增加了函数重载机制
- 采用新的作用域限定运算符::
  - 新的运算符:: 用于解决名字冲突,:: 运算符用于存取一个在隐藏在当前作用域内的一个项.

C++对C语言的改进(2)

- 采用const 说明符
  - 采用const在实体作用域范围可冻结一个实体值.即申明一个常量.它能冻结一个指针变量指向的数据.指针地址的值.或者指针地址和指向的数据两者的值.
  - 一个函数参数也能用const说明,即可冻结函数内参数该参数值.
  - const 后被引入C,但不是强制冻结.但在C++是严格为常量.
- const 主要有如下应用情况
  - 声明后面是一个常量
  - 如果跟指针组合,将形成复杂的组合情况
  - 作函数参数数的修饰符.表示不能修改这一参数的值.主要是指针
  - 作函数返回值，表示调用函数不能修改这返回值的值.主要用于限定返回对象
  - 用于限定函数体,放在函数声明后明,表示函数里不能用任何修改语句

C++对C语言的改进(3)

- const 让人感到复杂是的,他的位置可以与被修饰的数据类型互换!其它的修饰符没有这样用法.这样换一般情况下是等效的,如 const int c1= 5; 等效于 int const c1=5;
- 如果数据类型是一个指针,互换一样位置表示完全不同含义

C++对C语言的改进(4)

- 参看如下定义
  - int b = 500;
  - const int* a = &b; [1]
  - int const *a = &b; [2]
  - int* const a = &b; [3]
  - const int* const a = &b; [4]
  - 这种定义要看 const 的位置来确定他的用法.
- const位于星号的左侧，则const就是用来修饰指针所指向的变量，即指针指向为常量；如果const位于星号的 右侧，const就是修饰指针本身，即指针本身是常量。因此，[1]和[2]的情况相同，都是指针所指向的内容为常量（const放在变量声明符的位置无 关），这种情况下不允许对指针指向内容进行更改操作，如不能*a = 3 ；[3]为指针本身是常量，而指针所指向的内容不是常量，这种情况下不能对指针本身进行更改操作，如a++是错误的；[4]为指针本身和指向的内容均为常 量。
- Const的初始化
  - Const 在运行时不能修改值,所以只能在定义时进行初始化.

C++对C语言的改进(5)

- **作为参数和返回值的const修饰符**

- 其实，不论是参数还是返回值，道理都是一样的，参数传入时候和函数返回的时候，初始化const变量
  - 1 修饰参数的const，如 void fun0(const A* a);表示a的指向内容不准修改的.
  - 2 修饰返回值的const，如const A * fun2( );表示a指南内容不准备修改
- Const 的好处是
  - • 关键字const 的作用是为给读你代码的人传达非常有用的信息，实际上，声明一个参数为常量是为了告诉了用户这个参数的应用目的。如果你曾花很多时间清理其它人留下的垃圾，你就会很快学会感谢这点多余的信息。（当然，懂得用const 的程序员很少会留下的垃圾让别人来清理的。）
  - • 通过给优化器一些附加的信息，使用关键字const 也许能产生更紧凑的代码。
  - • 合理地使用关键字const 可以使编译器很自然地保护那些不希望被改变的参数，防止其被无意的代码修改。简而言之，这样可以减少bug 的出现。
- Const 一大副作用是，在函数调用时，你必须要把参数调成const 所要求的形式才能被编译调用. 有时这样花很大功夫才能做到

C++对C语言的改进(6)

- 函数参数的缺省值
  - 在C++ 函数中一串参数能指定为缺省值. 基本这种情况，函数能用少于全部参数个数的参数来引用. 任何一串舍弃的尾部参数均可假设为缺省值.
    - 不能跳着省略缺省值
    - 缺省值不能同时出现在声明和实现当中.
- 更加方便的动态存储分配
  - new 和delete 机制,new 它能更可靠控制存储区的分配,C++ 用delete 释放new 运算符申请的存储区.
    - new 分配空间时，如果是对象，自动调用构造函数
  - new [] 表示分配一个动态数组，要跟delete [] 配合使用

- 函数中引用参数
  - 这是C++才有机制,允许函数形参用&操作符作为引用参数说明.
- inline 说明符
  - 没有压栈的操作，相对多语句宏，有类型检查功能

一个简单的C++程序

```
#include <iostream.h>
class person
{
public:
    char name[16];
    int age;
};        //类 person 结束
void main()
{
    person myself;
```

```
        cout << "\n 请输入您的姓名:  ";
        cin >>myself.name;
        cout << "\n 您的姓名是:  " << myself.name;
        cout << "\n 请输入您的年龄: ";
        cin >> myself.age;
        cout<< "\n 您的年龄是 " << myself.age << " 岁";
    }
```

VC++ 开发C++项目注意

- 纯C++程序采用MFC支持是编译通不过
- 所有MFC项目的源码里必须包含stdafx.h

总结

- 传统编程的缺点
- 对象
- 类
- 抽象
- 继承
- 封装
- 多态性

课堂练习

高级函数特性

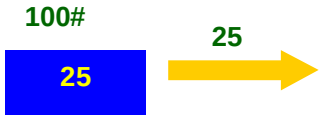
Andrew Huang<bluedrum@163.com>

目标

- 引用
- **默认参数**
- 内联函数
- **函数重载**

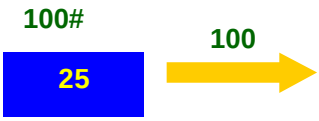
按值传递

- 函数调用中复制参数的值
- 函数只能访问自己创建的副本
- 对副本进行的更改不会影响原始变量



按引用传递

- 函数调用中传递参数的引用
- 主要优点
  - 函数可以访问主调程序中的实际变量
  - 提供一种将多个值从被调函数返回到主调程序的机制



向函数传递引用 2-1

- 引用提供对象的别名或可选名
- “&”告诉编译器将变量当作引用

```
void swap(int& i, int& j)
{
    int tmp = i;
    i = j;
    j = tmp;
}
void main()
{
    int x, y;
    swap(x,y);
}
```

向函数传递引用 2-2

- 引用就是对象本身
- 不要认为
  - 引用是指向对象的指针
  - 引用是该对象的副本

- 大的数据结构按引用传递，效率非常高

返回引用

- 返回引用不是返回变量的副本
- 函数头中包含一个“&”

```
int &fn(int &num)
{
   return(num);
}
void main()
{
   int n1, n2;
   n1 = fn(n2);
}
```

常量引用

- 用于不希望修改对象，以及要把大对象当作输入参数的情况
- 高效性和安全性

```
double distance(const point& p1, const point& p2);
```

- 将引用声明为常量，不能再绑定别的对象

```
int const &ri = num1;
```

函数

- 函数声明
  - 函数名
  - 函数返回值的类型
  - 函数的参数个数和类型
- 函数声明可以不包含参数名
- 调用函数时可以不指定全部参数

函数的默认参数

- 为可以不指定的参数提供默认值

```
void func(int = 1, int = 3, char = '*');
```

或

```
void func(int num1,int num2 = 3,char ch = '*');
```

参数的默认值 2-1

- 一旦给一个参数赋了默认值，后续所有参数也都必须有默认值

```
void errfunc(int num1=2,int num2,char ch='+');//错误
```

- 默认值的类型必须正确
- 默认值可以在原型或者函数定义中给出，但不能在两个位置同时给出
- 建议在原型声明中指定默认值

参数的默认值 2-2

- 调用上面声明的函数 func()

```
func(2,13,'+');
func(1);        //第二个和第三个参数采用默认值
func(2,25);     //第三个参数采用默认值
func();         //所有这三个参数都采用默认值
func(2,, '+' );   //错误!
```

- 如果遗漏了中间的参数，编译器将报错

默认参数的优点

- 如果要使用的参数在函数中几乎总是采用相同的值，则默认参数非常方便
- 通过添加参数来增加函数的功能时，默认参数也非常有用

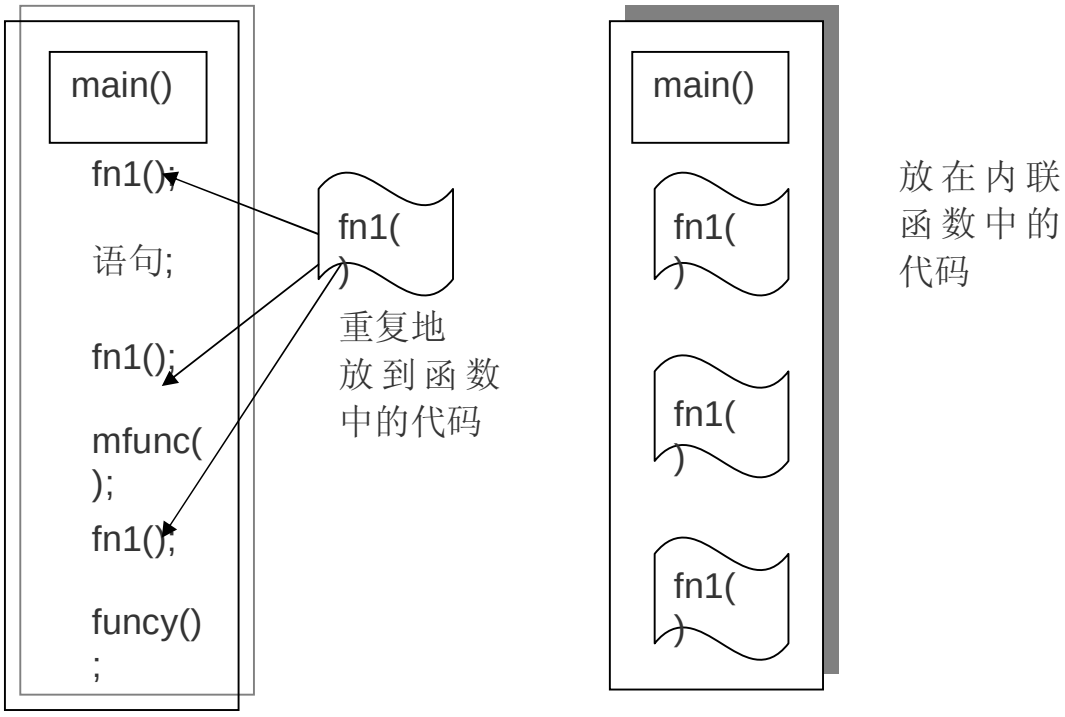
内联函数 2-1

- 通常的函数调用会节省内存空间，但是会花费一些额外的时间
- 内联函数节省短函数的执行时间

```
inline float converter(float dollars);
```

内联函数 2-2

- 非常短的函数适合于内联
- 函数体会插入到发生函数调用的地方



注意事项

- 编译器必须先看到函数定义，而不是声明
- 编译器有可能会忽略inline关键字
- 不允许为不同的源文件中的内联函数指定不同的实现

函数重载 2-1

- 具有相同的名称，执行基本相同的操作，但是使用不同的参数列表
- 函数多态性

```
void display();
void display(const char*);
void display(int one, int two);
void display(float number);
```

函数重载 2-2

- 编译器通过调用时参数的个数和类型确定调用重载函数的哪个定义
- 只有对不同的数据集完成基本相同任务的函数才应重载

函数重载的优点

- 不必使用不同的函数名
- 有助于理解和调试代码
- 易于维护代码

数据类型不同的重载

- 参数的类型不同，编译器就能够区分

```
int square(int);
float square(float);
double square(double);
```

- 同一函数名输出任何数据就是重载了输出函数

参数个数不同的重载

```
int square(int);       //函数声明
int square(int,int,int);
int asq = square(a)  //函数调用
int bsq = square(x,y,z)
```

- 编译器会调用参数匹配的函数
- 与函数的声明顺序无关
- 不会考虑返回类型

函数重载的作用域规则

- 重载机制只有在函数声明的作用域内才有效

```
class first{
  public:
    void display();
  };
  class second{
  public:
      void display();
  };
```

```
void main()
{
  first object1;
  second object2;
  //没有发生函数重载
  object1.display()
  object2.display();
}
```

总结

- 引用
- 默认参数

- 内联函数
- 函数重载

课堂练习