# C++编程(1)

Tang Xiaosheng
北京邮电大学电信工程学院

# 课程简介

- 参考资料：
  - C++程序设计语言(特别版) 裘宗燕(北京大学)译
    Bjarne Stroupstrup 机械工业出版社
    ISBN 7-111-10202-9/TP.2424
    85RMB
  - C++ Primer, Lippman
  - C++标准程序库 侯捷 孟岩译
  - STL源码剖析 侯捷著
- 课时：34学时(17周)
- 教学网站：
  http://young.byr.edu.cn/

# 考试

- 100道选择题（闭卷）

# 第一章 导论(致读者)

- 书本结构
- 学习C++
- C++的设计
- 历史注记
- C++的使用
- C和C++
- 忠告

# 1 书本结构

- 导论：1-3章，有关C++语言，所支持的关键性程序设计风格，有关C++标准库的综述
- 第一部分：4-9章，C++内部类型
- 第二部分：10-15章，使用C++做面向对象和通用型程序设计
- 第三部分：16-22章，C++标准库
- 第四部分：23-25章，设计和软件开发

# 2 学习C++

- 最重要的事情：关注概念，不要迷失在语言的技术细节中
- C++支持多种不同的程序设计风格
- C++支持一种逐步推进的学习方式
- 直接学习C++，而不是先学习C
- 大量查阅资料，每一种至少参考两个来源

# 3 C++的设计

- 重要设计原则：简单性，与C的高度兼容
- 无内部高级数据类型，没有高级的基本操作
- 尽力避免了那些即使不用也会带来运行时间或者空间开销的特征
- 能够使用传统的编译和运行时的环境
- C++的类型检查和数据隐藏特征依赖于编译时对程序的分析，以防止因为意外而破坏数据的情况

# (1) 效率和结构

- C++以C为基础开发设计，大量维持了C作为一个子集，C++可以使用与C一样的函数调用及返回序列—或者其他效率更高的方式，而C的一个初始目标则是在大部分苛刻的系统程序设计中代替汇编
- C++中特别强调程序的结构，这反映了C以来程序规模增长的情况
- 本书强调的是为提供通用功能、普遍有用的类型、库等的各种技术

# (2) 哲学注记

- 程序设计语言要服务于两个相互关联的目的
- 1 为程序员提供一种描述所需执行的动作的载体，这要求一种“尽可能接近机器的”语言
- 2 为程序员提供一组概念，使他们能利用这些概念去思考什么东西是能够做的，这要求该语言“尽可能接近需要解决的问题”
- C++的类概念已经被证明是一种极为强有力的概念工具

# 4 历史注记

- C++大大受惠于C[Kernighan,1978]
- 其前驱为BCPL[Richards,1980](//注释)
- 一些灵感来自Simula67[Dahl,1970,1972](如类的概念、派生和虚函数)
- Algol68[Woodward,1974]，如重载运算符和自由的将声明放置在可以出现语句的任何位置
- 模板机制功能部分受到Ada中generic的启发，部分受到Clu语言参数模块的影响

- 异常处理机制部分受到Ada[Ichbiah,1979]、Clu[Liskov,1979]和ML[Wikstrom,1987]语言的影响
- C++语言从1980年开始被研究组织使用，研究组织之外的最初使用起于1983年
- C++读做C plus plus
- 名称意义：C+,C++,++C,D
- C++设计的主要用途是为了个人程序员

- 87年，爆炸性使用导致了标准化工作的开始
- AT&T的贝尔实验室允许作者将C++参考手册的草稿和各种修订版本进行分发和共享
- ANSI的X3J16委员会1989年12月在HP的建议下建立起来
- 1991年该ANSI C++标准化变成ISO的C++标准化工作的一部分
- 标准化草案1995年4月给公众阅览，ISO C++标准1998年最终被批准(ISO/IEC 14882)

# 5 C++的使用

- C++被应用到几乎每个领域
- 高效率使得C++被用来写操作系统或者驱动程序
- 可靠性、可管理、易扩充、易测试使得C++被用于银行、贸易、保险、通信以及军事领域
- 由于编写用户界面，C++也被用于很多数值的、科学的以及工程计算中去

# C++被广泛应用于教学与研究

- 对于教授基本概念而言足够清晰
- 对于深刻的项目而言足够现实、高效和灵活
- 对依赖各种不同开发和执行环境的组织或者研究机构而言，使用起来足够方便
- 对作为教高级概念和技术的媒介而言，足够的容易理解
- 对作为从学习到非学习使用的工具而言，足够商业化

# 6 C和C++

- C被选做C++的基础语言的原因
  - 通用的、简洁的、相对低级的
  - 适合用于大部分系统的程序设计工作
  - 可以在每个地方的任何系统运行
  - 适应于UNIX程序设计环境

# C++继续与C兼容的原因

- 存在着成百万行的C代码可能从C++中获益，先决条件是不必将他们用C++重写
- 存在着成百万行的用C写出的库函数和功能软件代码可以从C++里使用，先决条件是C++能够与C连接兼容，语法相似
- 存在着数以十万计的程序员先了解C，这样他们能够很快速的学会C++
- C++和C将在很多年中被同一些人用于同样的系统，因此其差异必须或者很小、或者很大，以最大限度地减少错误和混乱的发生

# (1) 给C程序员的建议

- C++里几乎不需要宏(const,enum定义明显的常量, inline避免函数调用开销, template刻画一组函数或类型, 用namespace避免名字冲突)
- 使用变量时可以随时声明
- 不要用malloc(new或者vector)
- 避免使用void*，指针算术，联合和强制
- 少用数组和C风格的字符串(string, vector)

# (2) 给C++程序员的建议

- C++在不断发展中，应该注意引进新的特征，很多以前看来是全新的程序设计技术现在已经变成可行的东西了
- 不断的学习（作者本身在写这本书的过程中就学到了不少东西）

# 7 忠告

- 编程是在为某个问题的解决方案中的思想建立起一种具体表示
  - 如果能将“它”看成一个独立的概念，将其作成一个类
  - 如果能将“它”看成一个独立的实体，将它作成某个类的一个对象
  - 如果两个类有共同的界面，将此界面做成一个抽象类
  - 如果两个类的实现有某些显著的共同东西，将这些共性做成一个基类
  - 如果一个类是一种对象的容器，将它做成一个模板
  - 如果一个函数实现对某容器的一个算法，将它实现为对一族容器可用的模板函数
  - 如果一组类、模板等相互之间有逻辑关系，将它们放进一个名字空间去

- 在定义一个并不是实现某个像矩阵或复数这样的数学对象的类时，或者定义一个低层的类型如链表时：
  - 不要使用全局数据（使用成员）
  - 不要使用全局函数
  - 不要使用公用数据成员
  - 不要使用友元
  - 不要在一个类里面放“类型域”：采用虚函数
  - 不要使用内联函数，除非有效果显著的优化