C++ 程序设计

王洪波

网络技术研究院

网络与交换技术国家重点实验室

宽带网研究中心

课程目标

- 本课程是一门以实践为目的的程序设计语言教学课程。

- 通过本课程的学习实践，学生应能够掌握利用C++语言进行面向对象程序设计的一般方法，并具有使用C++语言进行程序设计和解决实际问题的能力

教学方式

- 课堂教学(9月7、9、11日上午, 9月8日下午)
  - C++语言的语义、语法
  - 使用C++语言进行程序设计的基本技巧
  - 解决编程实践中存在的问题
- 编程实践（9月10下午， 9月13、14、16日上午）
  - 提高同学的实践能力
  - 练习与编程作业

主要教学内容

- 基本概念
- C++语言基础
- 面向对象的程序设计初步与C++的类
- C++的高级特性
- C++程序设计综合训练

基础

- 先修课程
  - 高级语言程序设计：C/Pascal
  - 其它（非必须）：汇编语言、编译原理、...
- 参考书目
  - 入门级别：
    - C++语言程序设计（第3版），郑莉等编著，清华大学出版社
    - Essential C++中文版，Stanley B.Lippman著，华中科技大学出版社
  - 大全类：
    - C++程序设计语言，Bjarne Stroustrup著，机械工业出版社
    - C++ Primer中文版，Stanley B.Lippman等著，人民邮电出版社
  - 高级进阶类：
    - Effective C++, More Effective C++, Thinking in C++, …

考核方式

- 主要考核同学应用编程实践的完成情况和能力
  - 必须完成的内容
    - 课后上机作业
    - 综合编程练习
  - 每个同学必须**独立**完成
- 课件资料、实验安排及通知等：
  - 邮箱： buptcw@163.com， 收件箱
  - 口令：20150907c++
- 答疑、提交作业方式：
  - 邮箱： bupthomework@163.com

程序设计的基础

程序设计语言的发展

- 硬件语言
  - 机器语言、汇编语言
- 非结构化语言
  - FORTRAN
  - BASIC
- 结构化的语言
  - PASCAL
  - C
  - Ada
- 面向对象的语言
  - Smalltalk
  - C++
  - Java
- 组件编程
  - .net
  - C#/C++/Java

结构化程序设计

结构化程序设计方法

- 自顶向下，逐步求精，模块化
  - 先考虑总体，后考虑细节；先考虑全局目标，后考虑局部目标
  - 对复杂问题，分解成若干简单问题，确立子目标，逐步细节化
- 限制直接跳转（goto）的使用

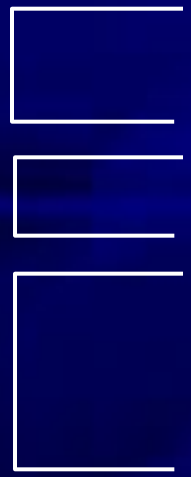
结构化程序设计语言

- 非结构化程序设计语言
  - ASM
  - Fortran
  - BASIC

程序结构

- 结构化程序设计语言
  - Algol
  - Pascal
  - C
  - Ada

Pascal 结构

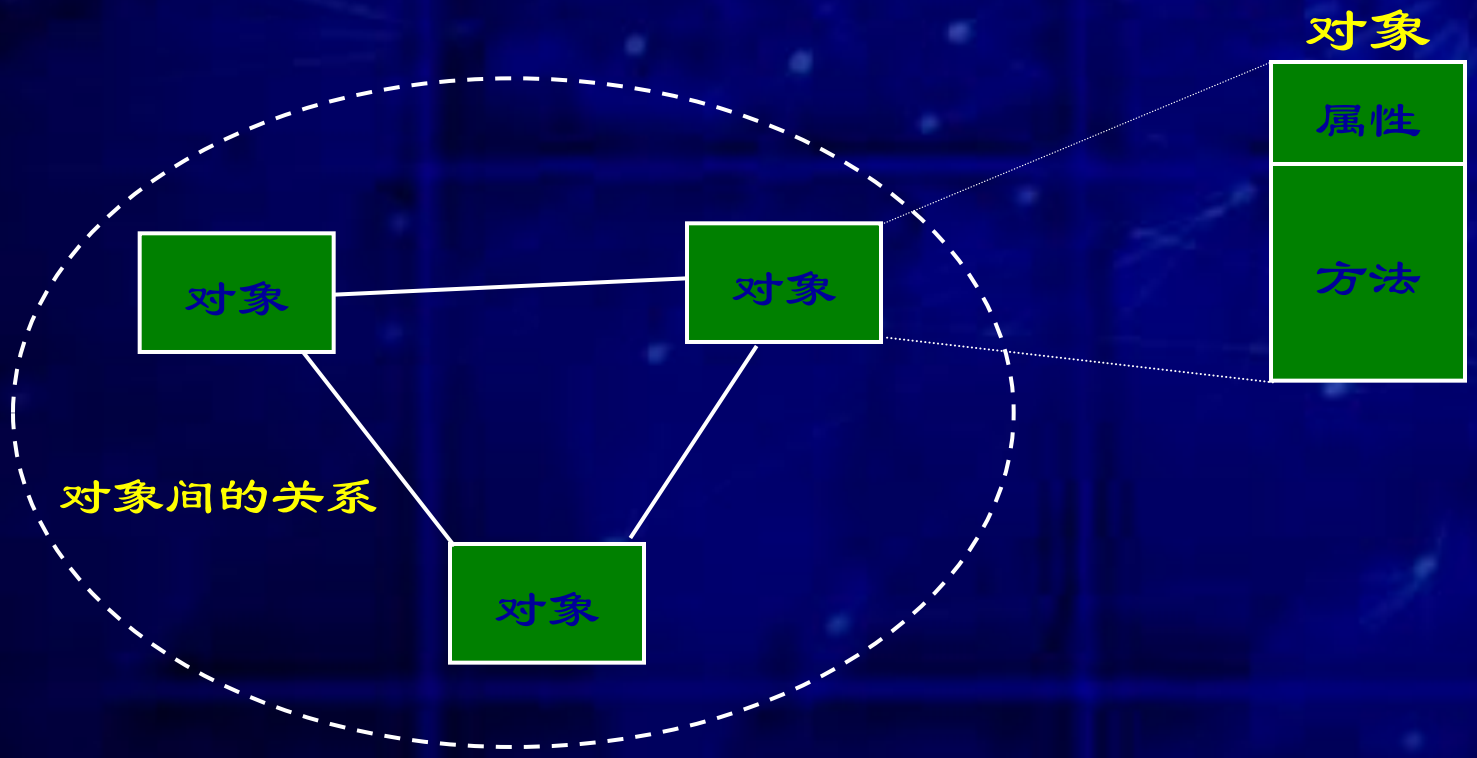


C 结构

面向对象的程序设计

- 面向对象的程序设计（Object Oriented Programming）
  - 使用对象模型来描述和解决自然世界中的问题
  - 能够大幅度的提高软件项目的成功率，减少日后的维护费用，提高软件的可移植性、重用（re-use）性和可靠性。
  - 特征
    - 对象模型，对象概念在从建模到构建程序的各个方面广泛使用
    - 抽象化，对象的属性进行抽象
    - 封装性，对对象的操作被封装在特定的作用范围
    - 多态性，派生对象的操作（方法）可以存在不同实现
    - 继承性，方法和属性可以在类间被继承和传递

面向对象的程序设计



程序 = 对象 + 对象间的相互作用

属性：描述对象性质的数据集合

方法：处理对象之间相互作用的操作方式

面向对象的程序设计语言

- Simula-67
- SmallTalk
- C++
- Java
- C#

C++ 与 C 的关系

- C++ 源于 C 语言
  - C 语言是在 B 语言的基础上发展起来的
    - 1960：ALGOL 60
    - 1963：剑桥大学推出了CPL（Combined Programming Language）语言，后来经简化为BCPL语言
    - 1970：贝尔实验室的 K.Thompson 以BCPL语言为基础，设计了一种新语言，取其第一字母 B，称为 B 语言
    - 1972：贝尔实验室的Dennis M.Ritchie为克服B语言的诸多不足，在B语言的基础上重新设计了一种语言，取其第二字母C，称为 C 语言
  - C语言的特点是极高的代码效率，但：
    - 不支持面向对象，不支持类与封装机制
    - 难以支持代码重用
  - 1980年，贝尔实验室的 Bjarne Stroustrup 对 C 语言进行了扩充，提出了“带类的C”，多次修改后起名为 C++，以后又经过不断的改进
    - C++改进了C的不足之处，支持面向对象的程序设计，在改进的同时保持了C的简洁性和高效性。

C++ 与 C 的关系

- C++语言是在C语言的基础上进行了扩充和改进而得到的
  - 它继承了 C 语言的全部内容
  - 并在 C 语言的基础之上增加了面向对象编程的内容
  - C++既支持面向过程的程序设计，又支持新型的面向对象程序设计
- C++保持了与C语言的兼容
  - 大部分的 C 代码的程序不经修改，或很少修改就可被 C++ 使用
  - 用 C 语言编写的许多库函数和应用软件也都可以用于 C++
- C++不是一个纯粹的面向对象程序设计语言
  - C 语言是面向过程的语言，C++与C兼容，支持面向过程的程序设计
  - 由于面向过程程序设计和面向对象程序设计是两种不同风格的程序设计技术，对于习惯于面向过程程序设计的程序员在学习使用 C++ 时可能存在一定的障碍

C++ 对 C 的扩展

增强的安全性

- 改进类型系统，强制的类型检查，有利于减少程序错误

```
#include <stdio.h>
main()
{
  int a=1;
  int b=2;
  printf("a+b=%d\n", func(a,b));
}
int func(int a, int b)
{
  return  a+b;
}

a+b=3
```

```
#include <stdio.h>
main()
{
  float a=1;
  int b=2;
  printf("a+b=%d\n", func(a,b));
}
int func(int a, int b)
{
  return  a+b;
}

a+b=1072693248
```

C++ 对 C 的扩展

➢ 增强的安全性

- ❑ 改进类型系统，强制的类型检查，有利于减少程序错误

```
#include <stdio.h>
main()
{
  int a=1;
  int b=2;
  printf("a+b=%d
}
int func(int a,
{
  return  a+b;
}

a+b=3
```

```
#include <stdio.h>
main()
{
  float a=1;
  int b=2;
  printf("a+b=%d\n", func(a,b));
}
int func(int a, int b)
{
  return  a+b;
}

a+b=3
```

```
#include <stdio.h>
int func(int a, int b);     // 函数原型的说明
main()
{
  float a=1;
  int b=2;
  printf("a+b=%d\n", func(a,b));
}
int func(int a, int b)
{
  return  a+b;
}

a+b=3
```

C++ 对 C 的扩展

- 增加了一些在新的运算符，使得C++应用起来更加方便
  - new，delete 用于内存管理，用户不需直接使用库函数
  - 增加了引用 &，使得引用函数参数带来了很大方便。
- 函数重载，设置缺省参数，提高编程灵活性，减少了代码冗余
- 引进了内联函数的概念，提高了程序的效率
- 变量在需要时进行说明，方便了程序编写
  - C语言先对变量的说明语句，再是执行语句

```
void func(int a, int b) { ... }
void func(float a, float b){ ... }
void foo(int k, int m=1);

int i,j;
float g,h;

main()
{
  func(i,j);
  func(g,h);
  foo(100);
}
```

```
void func(int a, int b)
{
    int i;
    float g;

    ...
    i = a+b;
    g = a-b;
    ...
}
```

C++ 对 C 的扩展

- 增加了一些在新的运算符，使得C++应用起来更加方便
  - new，delete 用于内存管理，用户不需直接使用库函数
  - 增加了引用 &，使得引用函数参数带来了很大方便。
- 函数重载，设置缺省参数，提高编程灵活性，减少了代码冗余
- 引进了内联函数的概念，提高了程序的效率
- 变量在需要时进行说明，方便了程序编写
  - C语言先对变量的说明语句，再是执行语句

```
void func(int a, int b) { ... }
void func(float a, float b){ ... }
void foo(int k, int m=1);

int i,j;
float g,h;

main()
{
  func(i,j);
  func(g,h);
  foo(100);
}
```

```
void func(int a, int b)
{
    ...              // 执行语句

    int i;
    i = a+b;
    ...              // 其他执行语句
    float g;
    g = a-b;
    ...
    for (int j=0;j<100;j++)
    { ...
    }
}
```

C++ 对 C 的扩展

- 与面向对象相关的功能：
  - 类与对象：抽象、封装、继承与派生、多态性
- 异常处理
- 模板编程，提供标准模板库（STL）
  - 容器、流式 IO、字符串、...

```
#include <stdio.h>
#include <string.h>

main()
{
    char buf_1[1024];
    char buf_2[1024];

    printf("Please input word 1:");
    sscanf(buf_1,"%s");
    printf("Please input word 2:");
    sscanf(buf_2,"%s");
    strcat(buf_1,buf_2);
    printf("%s\n",buf_1);

}
```

```
#include <iostream>
#include <string>
using namespace std;

main()
{
    string word_1, word_2;
    cout << " Please input word 1:" ;
    cin >>word_1;
    cout << " Please input word 2:" ;
    cin >>word_2;
    cout << word_1+word_2 << endl;

}
```

C++ 对 C 的扩展

- 与面向对象相关的功能：
  - 类与对象：抽象、封装、继承与派生、多态性
- 异常处理
- 模板编程，提供标准模板库（STL）
  - 容器、流式 IO、字符串、...

```
#include <stdio.h>
#include <string.h>

main()
{
    char buf_1[1024];
    char buf_2[1024];

    printf("Please inp
    sscanf(buf_1,"%s")
    printf("Please inp
    sscanf(buf_2,"%s")
    strcat(buf_1,buf_2
    printf("%s\n",buf_

}
```

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

main()
{
    ofstream outfile ( "filename_1" );
    outfile << " This is a test ! " << endl;

    ifstream infile ( "filename_2" );
    string word;
    while ( infile >> word )
    {
        cout << word <<endl;
    }
}
```

```
word 1:" ;

word 2:" ;

<< endl;
```

C++ 对 C 的扩展

- 编程方法
  - C++ 对 C 的兼容是建立在发展和完善的基础上的
  - C++ 支持面向对象的程序设计
  - C 是面向过程的程序设计语言
- 程序结构
  - C++ 程序（面向对象）的结构采用 “对象+消息” 模式
  - C 程序结构采用 “数据+算法” 模式
- 适用性
  - 底层系统程序、嵌入式程序设计、大规模高层应用设计、通用程序设计、数值科学计算 ...

程序设计

代码的版式与风格

- 关系到代码的可读性
  - 继而影响到代码的调试、管理、移植等多方面的性能
  - 因此，在编写代码时，应养成良好的习惯
  - 良好的代码风格最主要是为自己和他人提供可读性
    - 这对于大型软件开发是非常重要的
  - 没有固定的模式，关键是使代码流程清晰，易于识别

代码的版式与风格

- 一般规则
  - 文件开始写出完整的文件版本说明
  - 必需的注释
    - 例如，函数的用途、输入/输出的说明
  - 适当的空行和空格
  - 与模块化一直的对齐和缩进
  - 长行的拆分
  - 良好的命名规则

代码的版式与风格

- 一般规则：文件开始写出完整的文件版本说明

```
/* ------------------------------------------------------------------------------------
        HY-1 Communication Subsystem : event log subroutines (common file)
        ------------------------------------------------------------------
        Programer : Gong Xiangyang
        version   : 1.00
        Date        : 2006-02-25
    ********************************************************************************
        Error and event handling and logging
------------------------------------------------------------------------------------*/

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <stdio.h>
#include <sys/time.h>
```

代码的版式与风格

- 一般规则：必需的注释
  - 例如，数据结构的功能和含义；函数的用途、输入/输出的说明

```
/* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                    data structures for logging
------------------------------------------------------------------------------------------------------*/
#define    EVH_FILELOG          0x01                    // log to file
#define    EVH_CONSOLE          0x02                    // log to console

typedef struct       __EventInfoRecord__                // Event information record
{
           int       iEvLogFlag;                        // event handling flags
           int       iEvLogLevel;                       // event level
           int       iParNr;                            // parameter number
           int       iEventID;                          // event identifier
}          EvInfoRecord;

/*******************************************************************************************************
                    Global data structure
*******************************************************************************************************/
char       sMsgBuf[4096];                /* message buffer                                          */
static     FILE *     pLogFile = NULL;    // log file pointer
```

代码的版式与风格

- 必需的注释
- 适当的空格、空行，适当的对齐和缩进

```
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
          function: event handling, normal operation is to log this event
          input:    iEventID :          Event ID
                    sEvDesc:            Event decription
          output:   none
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++   */
void      LogEvent(int iEventID,char * sEvDesc)
{
    EvInfoRecord *  pEvInfo;

    if (!iEventID)
    {
        OutputEventLog(0,sEvDesc,EVH_FILELOG|EVH_CONSOLE|EVH_PROMPT);
        pEvInfo = GetEvInfoTabEntry(iEventID);

        if (iEventID != pEvInfo->iEventID)
            return;
    }
    else
    {
        if (pEvInfo->iEvLogFlag & EVH_SENDMSG)
            SendCtrlMessage(iEventID,sEvDesc);
    }

    OutputEventLog(iEventID, sEvDesc,pEvInfo->iEvLogFlag);
}
```

代码的版式与风格

- 一般规则：长行的拆分

```
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
          function:   return current date and time in a string
          format is:
                              2001-05-20 22:30:58
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++   */
char *    GetCurrentTimed(void)
{
          static char          sBuf[20];
          struct timeval       stCurTime;
          struct tm*           pTM;

          gettimeofday(&stCurTime,NULL);
          pTM = localtime(&(stCurTime.tv_sec));

          sprintf(sBuf,"%.4d-%.2d-%.2d %.2d:%.2d:%.2d",
                  pTM->tm_year+1900,pTM->tm_mon+1, pTM->tm_mday,
                  pTM->tm_hour,pTM->tm_min,pTM->tm_sec);

          return    sBuf;
}
```

代码的版式与风格

- 标识符的命名规则
  - 标识符可以“望文生义”，易于理解
  - 应当符合“min-length max-information”原则
  - 命名规则尽量与所采用的操作系统或开发工具的风格保持一致
- 常用法则：Microsoft 的“匈牙利”命名法
  - 以一个有意义的词组来描述标识符，各单词首字母大写
  - 标识符上增加适当前缀，以增进对程序的理解

```
#define         MAX_VALUE   65536

DWORD    g_dwSum;

class CConfig
{
private:
      CConfig * m_pConfig;
public:
      int       ReadKey(const char * pSecName, const char * pKeyName);
      int       WriteKey(const char * pSecName, const char * pKeyName, int iValue);
};
```