C++面向对象程序设计

第二讲：C++概述

计算机科学学院@中南民族大学

C++的起源和特点

- C的优势和局限性
  - 优势
    - 简洁灵活、运算符和数据结构丰富、具有结构化控制语句
    - 执行效率高
    - 同时具有高级语言与汇编语言的优点
  - 局限性
    - 过程范型的局限性
    - 类型检查机制较弱
    - 缺乏支持代码重用的语言结构

C++的起源和特点

- C++的特点
  - 兼容C
  - 扩充了C的功能
  - 提高了软件的可重用性、可扩充性、可维护性和可靠性
  - 几乎支持所有的面向对象程序设计特征
    - 抽象数据类型
    - 封装与信息隐藏
    - 以继承的方式实现程序的重用
    - 以函数重载、运算符重载和虚函数来实现多态
    - 以模板来实现类型的参数化

C++源程序的构成

- 注释
  - 行注释符号： //
    - //.....
    - int x; //定义变量
  - 块注释符号： /* ..... */
- 语句
  - 编译预处理指令
  - 全局变量声明
  - 主函数
    - int main(int argc, char* args[]) {
    - ……
    - }

C++程序编程规范

- 使源程序结构清晰、具有可读性
- Google
  - Google C++ Style Guide
  - 中文版
- 中兴
  - 软件编程规范C++
- C/C++编程规范
  - C_C++ 编程规范

源程序编码

- 接口/声明(头文件，.h)
  - 调用规约
- 定义/实现（源文件，.cpp/.cc）
  - 使用者无须了解的内容
- 接口与实现分离的好处
  - 隐藏实现细节
  - 加强类型安全的检查
- 符合编程规范

编译、连接、运行

- GNU
- Visual C++
- Borland C++
- ……

调试

- Microsoft
- GNU
- Borland

- 依据程序的走向判断处理流程是否合理
- 依据变量的值和内存中的数据判断程序的处理结果是否正确
- 单步、断点
- 查看变量值
- 查看内存中的数据

测试

- 局部
- 全局
- 测试方法

程序的构成要素

- 数据
  - 常数、常量、变量
  - 以变量来表示数据
    - 类型、变量名
- 数据处理过程
  - 以函数来表示数据处理过程
    - 原型： 返回值 函数名(形参列表);
      - 返回值：类型
      - 形参列表：类型列表
    - 定义：数据处理过程的实现

- 对象
- 使用对象的服务来完成任务
  - 原型（类的定义）
    - 属性（成员变量）
    - 方法（成员函数）
  - 实现（类的实现）
    - 成员函数

使用对象编写程序

- 从键盘输入两个字符串S1和S2，将S2拼接到S1的后面，并输出到屏幕上。

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string s1, s2;
    cin >> s1 >> s2;
    s1 = s1 + s2;
    cout << s1;
    return 0;
}
```

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s1[10], s2[10], s3[20];
    scanf("%s%s", s1, s2);
    strcpy(s3, s1);
    strcat(s3, s2);
    printf("%s\n", s3);
    return 0;
}
```

体会对象范型与过程范型的差异！

对象编程

- 指向对象的指针（对象指针）
  - string *s1;
  - s1 = new string("this is s string.”); //使用构造函数创建对象
  - cout << *s1;
  - delete s1;
- 指向对象的指针，对象
  - string *ps1; //指针一个string对象的指针变量
  - ps1 = new string("this is s string.”); //按需构造对象
  - delete ps1;
  - string s1; //定义一个string对象

向函数传递对象

```
void swap1(string s2, string s2)
{
    string tmp = s1;
    s1 = s2;
    s2 = tmp;
    cout << "F s2:" << s2 << endl;
    cout << "F s3:" << s3 << endl;
}

string s1, s2;
swap1(s1, s2);
```

向函数传递指向对象的指针

```
void swap(string* s1, string* s2)
{
    string tmp = *s1;
    *s1 = *s2;
    *s2 = tmp;
}

string s1, s2;
swap(&s1, &s2);
```

函数返回值：对象、指向对象的指针

```
string strcat(string s1, string s2)
{
    return s1 + s2;
}

string* strcat1(string s1, string s2)
{
    string* s = new string(s1 + s2);
    return s;
}
```

```
string s1, s2, *s3;
s1 = strcat(s1, s2);
s3 = strcat1(s1, s2);
    // delete s3;
```

引用

- 变量或对象的别名，通过别名直接访问某个变量或对象
  - int x;
  - int &rx = x;
  - rx = 50; //相当于对x赋值
  - int y = rx; //相当于取x的值赋给y
- 引用作为函数参数
  - void swap(int &, int &); //传引用，交换两个参数的值，实参可能改变
  - void swap(int, int);
  - void swap(int *, int *);

*****比较指针和引用的差异

引用

- void swap(int &x, int &y)
- {
  - int t = x;
  - x = y;
  - y = t;
- }

- int a = 5, b = 10;
- swap(a, b); //此时变量a和b的值会是多少呢?

引用：使用引用返回函数值

- 必须返回作用域内的变量的引用，即引用的变量不能是函数中的局部变量
  - int &max(int &x, int &y);
  - {
    - return (x > y) ? x : y;
  - }

  - int a, b;
  - man(a, b) = 0; //?? 将两变量中的较大者赋值为0

引用：使用引用返回函数值（错误用法）

- int &max(int x, int y)
- {
  - return (x > y) ? x : y; //这里返回的是局部变量x或y
- }

- int a, b;
- max(a, b) = 0; //错误，函数返回后，局部变量已不可用。

引用：常量引用

- 被引用的变量或对象不允许发生变更
  - void func(const string &s1, const &s2)
  - {
    - s1 = “123”; //错误
    - s2 = “234”; //正确
    - ……
  - }

引用：C++数据类型

- 引用是指对一个[广义]对象的引用
  - C++对象：
    - 类、结构、联合等复杂数据类型变量（狭义对象）
    - int、char等简单数据类型变量（广义对象）
- 引用的基本特性
  - 引用者与被引用者共用同一块存储空间；
  - 没有变量的赋值过程；
  - 使用对象和使用对象的引用在语法格式上是一样的；
  - 引用必须初始化，声明、实参传递、函数返回；
  - 从一而终（一旦与某个变量绑定，将不能再变更）
- 引用在参数传递中的作用
  - 实质是通过别名直接访问实参
- 函数的返回值也可以是引用
  - 对函数赋值的实质是对引用的变量赋值

头文件的用法

- C函数库或程序的头文件
  - #include <string.h> <stdio.h> <stdlib.h> <ctype.h> <math.h>
  - C++中的标准用法
    - #include <cstring> <cstdio> <cstdlib> <cctype> <cmath>
- C++类库或程序的头文件
  - #include <iostream>
- 注意，使用C++头文件（无“.h”后缀）的标准用法时，需要添加命名空间，使用C头文件（有“.h”后缀）时，则不需要
  - using namespace std; //C++风格时一定要用！

命名空间

- 标识符的作用域（可有效使用的范围）：文件、函数、复合语句、类
- 命名空间是ANSIC++引入的可以由用户命名的标识符作用域，用来处理程序中常见的标识符同名冲突。
- 标准命名空间 std
- 命名空间是可嵌套的
- 可以使用命名空间别名，来代替较长的命名空间名称
- 同一命名空间中的标识符名称必须唯一

定义和使用命名空间

- namespace zhangsan {
- int x;
- void func(int x, int y);
- }

- void zhangsan::func(int x, int y)
- {
  - return x * y;
- }

命名空间

- 直接访问命名空间中的标识符
  - using namespace zhangsan;
  - x、func(2, 3)
- 例外
  - namespace lishi {
  - int x;
  - }
  - namesapce wangwu {
  - int x;
  - }

- using namespace lishi ;
- using namesapce wangwu ;

- x = 5; //是访问哪个?

命名空间

- 不使用using namespace xxxx; 时，在变量名的前面加命名空间
  - zhangsan::x
  - zhangsan::func(2, 3);
- 缩短命名空间的名称
  - namesapce zs = zhangsan;
  - using namespace zs;
    - 或
  - zs::x、zs::func(2, 3)

局部变量声明

- 用前声明即可
- 源程序的代码行较多时，就近声明
- 源程序的代码行较少时，在块首声明

- 结构、联合、枚举名可直接作为数据类型的名称使用
- `struct xx { … };`
- `xx a_xx;`

强制类型转换

- 注意目标类型值域包容源类型的值域
- int x = 10;
- double xx = (double)x; //c风格
- double yy = double(x); //C++风格

四种类型转换

- const_cast 常类型转换
- reinterpret_cast
- static_cast
  - 主要用于基本的数据类型和指针
- dynamic_cast

#define VS const

- #define PI 3.1415926
- const double PI = 3.1415926;
- 宏定义
  - 仅仅是符号替换，
  - 不是变量，也没有类型，且不占存储空间
  - 容易出错
- 常量修饰符const
  - 有类型，占用存储空间
  - 和指针一块使用的情况（允许修改值的方式不同，是值还是地址）
    - const char * name=“chen”; //指向常量的指针
    - char * const name=“chen”; //常指针
    - const char * const name=“chen”; //指向常量的常指针

指针与动态数组

- new delete
- int *pa;
- pa = new int;　　//malloc
- pa = new int(23);
- int *b; // int *b;
- b = new int[23];　　//只能构建一维数组
- int *bb = new int[2][3][4];　　//与编译器有关，并非所有的编译器都支持
- delete pa;　　//free
- delete []b;

函数

- 函数原型
  - 调用前必须有完整的
  - 返回值类型 函数名（形参列表）;
- 函数参数与返回值
  - 强调的是类型，而不是可使用的符号
  - 默认的形参值（从右向左）
- 引用
  - & 符号的新含义
  - 可用于形参和返回值
- 函数重载
- 内联函数

形参缺省值

- 带有默认值的形式参数
- 从后向前给默认值
- 没有对应的实参时，该形参的实参为默认值

宏定义 VS 内联函数

- 函数调用过程
- 宏定义
  - 仅仅是符号替换
- 内联函数 inline
  - inline int max(int, int);
  - C++编译器直接将内联函数的函数体插入到调用该函数的语句处，并用实参替换形参，不会出现函数调用的过程（但也不完全这样）
- 内联函数的几点说明（P28-P29）
  - 在第1次调用之前必须进行完整的定义；
  - ……

函数重载

- 功能相同但处理数据不同的函数使用相同的函数名
- 返回值类型 函数名(形参列表);
  - 形参的个数、形参的类型
- 形参有缺省值时会出现什么情况
  - int fun(int x, int y);
  - int fun(int x)

  - int funa(int x, int y=4);
  - int funa(int x);

函数模板（泛型编程）

- C++中的一个重要特性
- 函数模板是一个独立于类型的函数，可以产生函数的特定类型版本
- 所谓函数模板，实际上是建立一个通用函数，其函数类型和形参类型不具体指定，用一个虚拟的类型来代表。这个通用函数就称为函数模板

函数模板示例

```
double add(double x, double y) {
    return x + y ;
}
double add(int x, int y) {
    return x + y;
}
double add(char x, char y) {
    return x + y;
}
```

```
template <typename T>
T add(T x, T y) {
    return x + y ;
}
```

问题：

1. 啥时确定T的真实类型?

2. 如何使用函数模板?

函数模板示例（续）

- int xd,yd;
- double xf, yf;
- char xc,yc;
- add<double>(xd,yd);
- add<char>(xf, yf);
- add<char>(xc, yc);

使用限制：

在使用函数模板前，函数模板的函数体必须有定义！

输入与输出

- 输入、输出的格式控制
  - 输入是增加对用户的提示信息
  - 输出是让用户看得明白，看得习惯
- C++流
  - 头文件 iostream fstream
  - 类：iostream, fstream, ……
  - 预定义的流对象和操作符
    - << (输出流操作符)
    - >> (输入流操作符)
  - 标准流对象
    - cin, cout, cerr, clog

输出数据的格式化定义

- 宽度、精度、对齐、……
- 使用控制符控制输出格式
- 使用流对象的成员
- 设置格式状态的格式标志

流控制符及其作用

控制符	作用
dec	设置整数的基数为10
hex	设置整数的基数为16
oct	设置 整数的基数为8
setbase(n)	设置整数的基数为n(n只能是16，10，8之一)
setfill(c)	设置填充字符c，c可以是字符常量或字符变量
setprecision(n)	设置实数的精度为n位。在以一般十进制小数形式输出时，n代表有效数字。在以fixed(固定小数位 数)形式和scientific(指数)形式输出时，n为小数位数。
setw(n)	设置字段宽度为n位。
setiosflags(ios::fixed)	设置浮点数以固定的小数位数显示。
setiosflags(ios::scientific)	设置浮点数以科学计数法(即指数形式)显示。
setiosflags(ios::left)	输出数据左对齐。
setiosflags(ios::right)	输出数据右对齐。
setiosflags(ios::shipws)	忽略前导的空格。
setiosflags(ios::uppercase)	在以科学计数法输出E和十六进制输出字母X时，以大写表示。
setiosflags(ios::showpos)	输出正数时，给出“+”号。
resetiosflags	终止已设置的输出格式状态，在括号中应指定内容。

流对象的控制输出格式 #include<iomanip>

流成员函数	与之作用相同的控制符	作用
precision(n)	setprecision(n)	设置实 数的精度为n位。
width(n)	setw(n)	设置字段宽度为n位。
fill(c)	setfill(c)	设置填充字符c。
setf( )	setiosflags( )	设置输出格式状态，括号中应给出格式状态，内容与控制符setiosflags括号中内容相同。
ubsetf( )	resetiosflags( )	终止已设置的输出格式状态。

格式标志及其作用

ios::left	输出数据在本域宽范 围内左对齐
ios::right	输出数据在本域宽范围内右对齐
ios::internal	数值的符号位在域宽内左对齐，数值右对齐，中间由填充字符填充
ios::dec	设置整数的基数为10
ios::oct	设置整数的基数为8
ios::hex	设置整数的基数为16
ios::showbase	强制输出整数的基数(八进制以0打头，十六进制以0x打头)
ios::showpoint	强制输出浮点数的小点和尾数0
ios::uppercase	在以科学计数法输出E和十六进制输出字母X时，以大写表示
ios::showpos	输出正数时，给出“+”号。
ios::scientific	设置浮点数以科学计数法(即指数形式)显示
ios::fixed	设置浮点数以固定的小数位数显示
ios::unitbuf	每次输出后刷新所有流
ios::stdio	每次输出后清除 stdout，stderr

文件的输入与输出

- 打开文件
  - ifstream　　ofstream　fstream
  - open
- 读写文件
  - 流操作符
  - 读写函数
- 关闭文件
  - close