

C++面向对象程序设计-教学要求

- 课程总体目标

- 学习、了解当前软件系统开发与维护的主流软件技术，**掌握面向对象程序语言的封装、继承和多态的程序实现方法。**
- **培养面向对象的程序思维方法、分析方法和编程能力，能够应用C++的类、继承、多态、STL等面向对象的技术和方法对客观世界的问题域进行系统建模和软件设计。**
- **理解面向对象程序设计的本质，**为以后深入学习Web程序设计等课程打下良好的基础，同时也为面向对象编程技术在专业中的应用打下良好的基础。

C++面向对象程序设计-教学要求

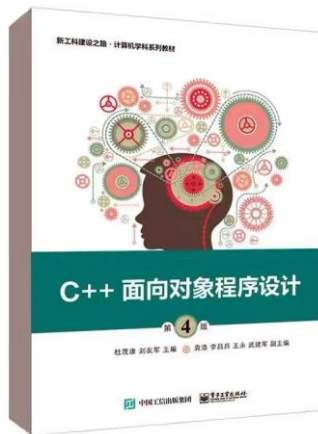
• 课程基本要求

1. 掌握用C++进行面向对象编程的方法，能够运用C++编写求解简单应用问题的面向对象程序；
2. 掌握C++对C的语言扩展。如常量、引用、输入输出流、函数重载等。
3. 掌握类与封装、对象、继承、多态等面向对象的语言特征和概念，及其在C++中的实现技术和方法；
4. 理解类与对象、构造函数、析构函数，类型转换、重载函数、虚函数、抽象类等面向对象程序设计中的重要概念及其实现方法；
5. 掌握面向对象的异常处理机制及C++中的编程方法；
6. 掌握函数模版、类模版及简单元编程的实现方法，能够熟练运用C++STL库设计程序；

主要参考书

- 教材

1. 《C++面向对象程序设计》(第4版) --电子工业出版社, 杜茂康
2. 《现代C++面向对象程序设计实验指导》(第1版) --科学出版社, 张俊



- 教参 (选1即可)

1. 《C++Primer 中文版》(第5版) --电子工业出版社, 王刚、杨巨峰译
2. 《程序设计教程—用C++语言编程》(第4版) --机械工业出版社, 陈家俊

- 课后交流

教学立方, 课程邀请码: QMRHZSVY

课程 QQ 群号: 685439280

课程考核方式

- 考核主要依据
 - 平时成绩 10% （考勤+提问+课后作业）
 - 上机作业 20% （考核）
 - 期末考试 70% （闭卷）
- 注意：一定要按时交作业。
 - 每次作业布置后一周内交
 - 教学立方提交（直接作答，或提交PDF格式）
 - 开发平台不限（VC、QT）版本不限（2022、2019、2010）

第1章 C++与面向对象程序设计概述

- **教学目标：** 了解C++和C语言程序的关系，理解面向对象和面向过程程序的主要区别，掌握面向对象程序设计语言的基本特征，能够搭建C++程序开发环境，掌握C++输入输出流并能运用它编写简易的C++程序。
- **教学重点：** 面向对象程序语言的基本特征，C++输入输出流的概念及应用。
- **教学难点：**
 - 输入输出流的程序实现和数据格式控制问题。
 - 字符串输入程序设计的常见问题及解决方法。

第1章 C++与面向对象程序设计概述

本章主要讲述：

1. 面向对象与面向过程程序的结构差异
2. 面向对象程序语言的基本特征
3. //C++标准的演化
4. C++程序的数据输入/输出
5. //VC++2022程序实现方法

1.1 面向对象程序设计概述

1.1.1 面向过程程序设计

- 主要采用结构化程序设计语言编程，因此也常称为结构化程序设计

(1) 结构化程序设计的基本内容

- 结构的类型

- 顺序、分支、循环

- 结构化程序设计思想

- 利用过程或函数来抽象和模拟客观现实。

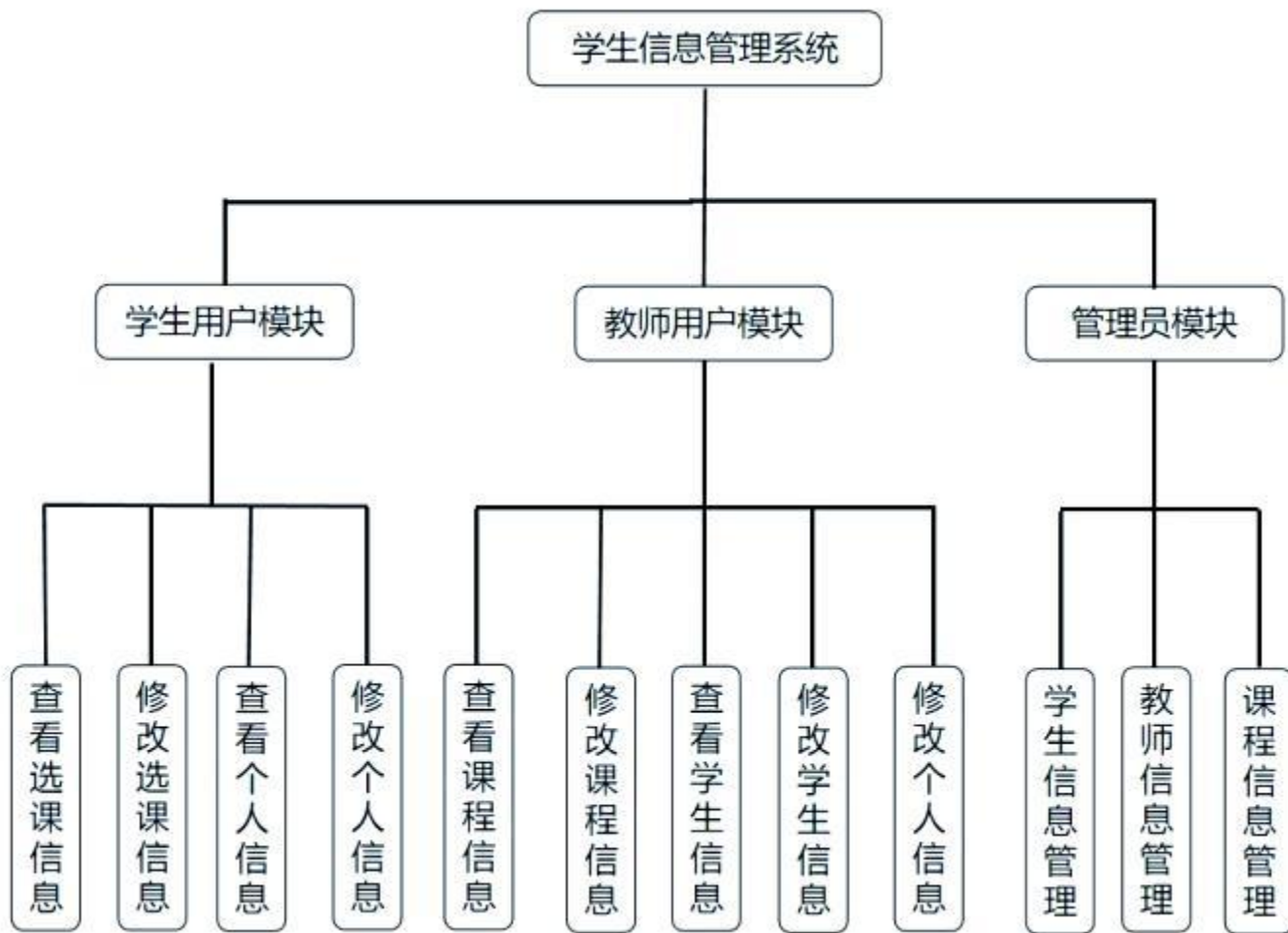
1.1 面向对象程序设计概述

(2) 结构化程序设计的方法

- 重点放在如何实现细节过程方面，将数据与函数分开。
- 形式：主模块+若干个子模块
(如 C: `main()`+子函数)。
- 特点：
自顶向下，逐步求精——功能分解。
- 缺点：效率低，是手工作坊式的编程。

1.1 面向对象程序设计概述

- 自顶向下，逐步求精——软件基本结构模型示例



1.1 面向对象程序设计概述

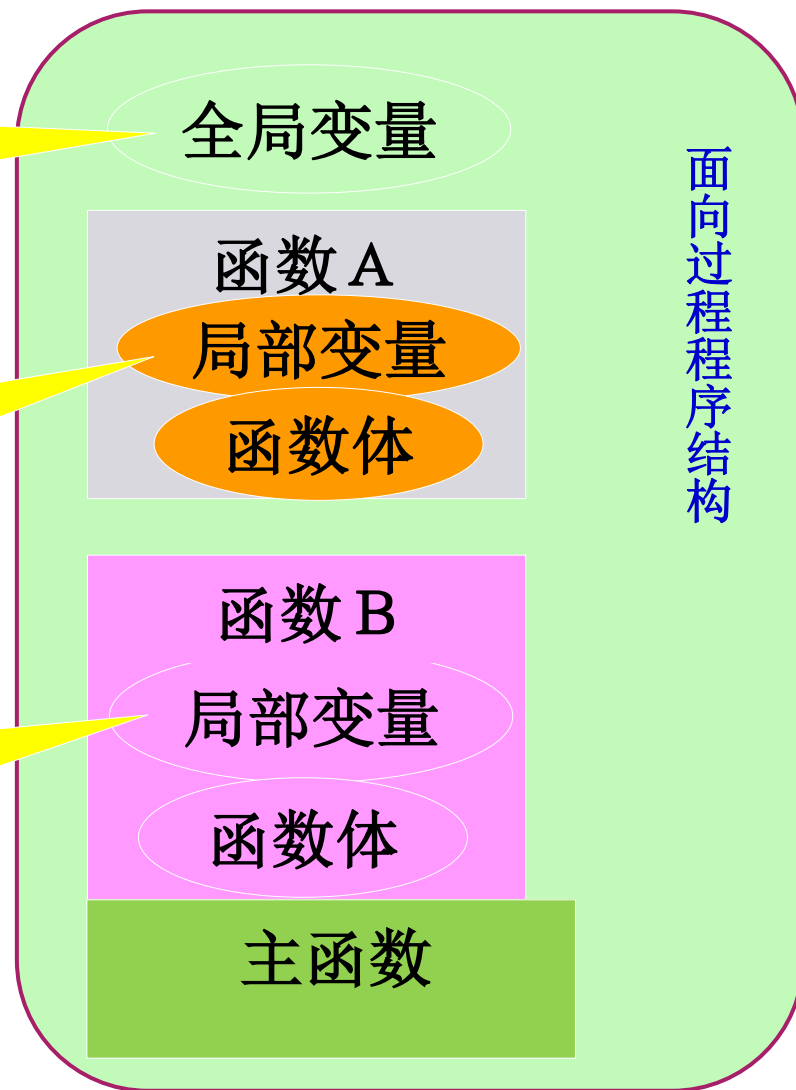
(3) 结构化程序设计的问题

——数据同算法分离。即程序数据和操作数据的函数是分离的。

全局变量
可被任何
函数访问

只有函数
A 才能访问

只有函数
B 才能访问



1.1 面向对象程序设计概述

(4) 结构化程序设计的案例——个人通信录程序

```
struct Person{  
    char name[10];  
    char addr[20];  
    char phone[11];  
}
```

1.定义数
据结构

```
Person p[100];  
int n=0;
```

2.定义全
局数据

```
void InputData(){ .....}  
void SearchAddr(char *name){.....}  
void SearchPhone(char *name){.....}  
void PrintData(){.....}
```

3.定义操作
数据的函数

```
Void main(){
```

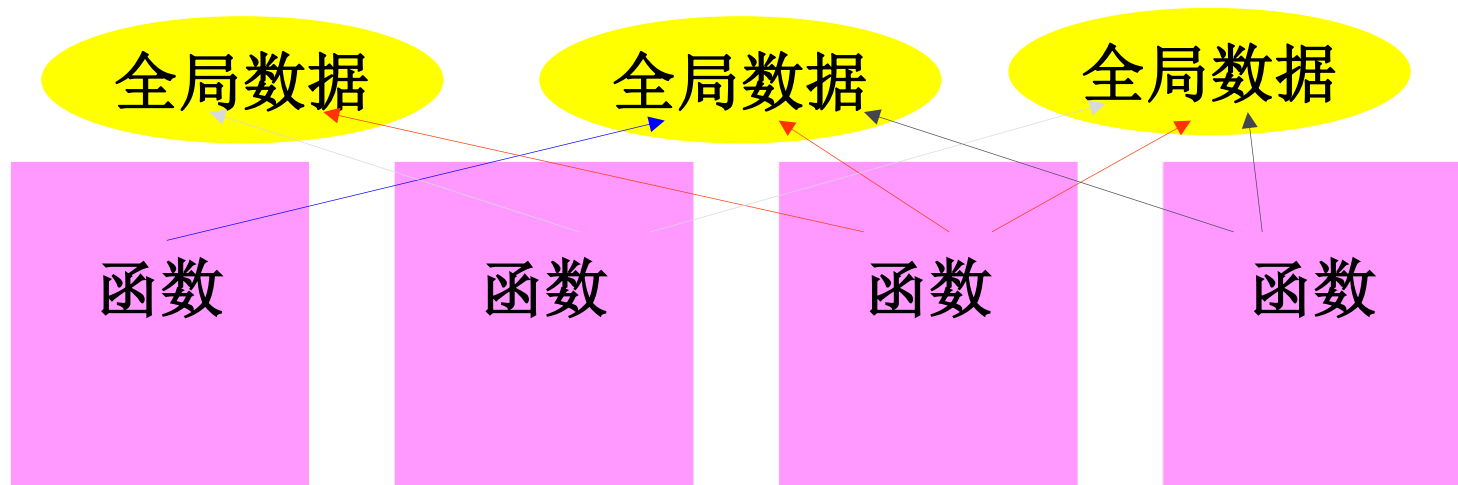
4.主函数
控制程序
流程

```
.....  
//调用前面编写的函数，完成通信录数据处理  
.....
```

```
}
```

1.1 面向对象程序设计概述

(5) 结构化程序设计范型



- 大型程序中，有很多全局数据和全局函数，这导致了函数和数据之间数目巨大的**潜在连接**！

如：**Windows Vista**的代码行数达到了**5000**万行

- 若全局数据有所改动，可能会导致所有访问这个数据的全部函数的重写，程序维护困难！

1.1 面向对象程序设计概述

1.1.2 面向对象程序设计

(1) 面向对象程序设计的观点

- ①现实世界是由各种实体（对象）所组成,每种对象都有自己的内部状态和运动规律；②不同对象之间的相互联系；③对象之间相互作用就构成了各种不同的系统,进而构成整个客观世界。
- 用软件对象来描述模仿现实中的对象及其关系,进而处理现实问题。对应解决上面三个问题：
 - ①用软件对象表示客观对象；
 - ②用软件对象之间的关系表示客观对象之间的关系；
 - ③用软件表示客观对象之间的交流与驱动。



Class Cat
& class Rat



Class Dog

1.1 面向对象程序设计概述

1.1.2. 面向对象程序设计

(2) 面向对象观点的实现方法

- ①将数据和操作数据的过程（函数）绑在一起，形成一个相互依存、不可分离的整体（即对象），仿真现实对象；
- ②通过继承，对象成员，对象依赖等语言机制来描述客观事物之间诸如父子关系，汽车与其组成部件的包含关系，某人与他的宠物狗之间的依赖关系.....
- ③通过对象之间的消息传递机制表示客观事物之间的交流与驱动关系



**Class Cat &
class Rat**



Class Dog

1.1.2. 面向对象程序设计

(3)面向对象程序设计基本概念

对象由属性和行为构成

– 对象

- 客观存在的实体称为对象

– 属性

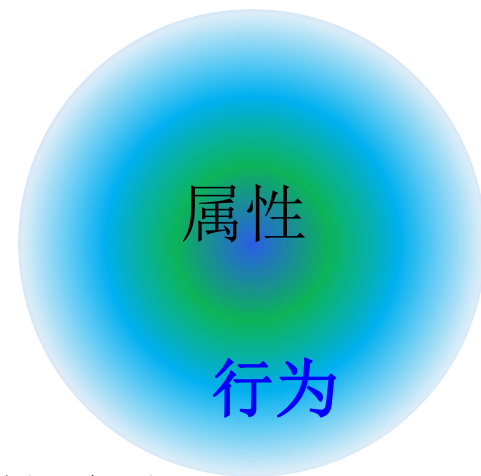
- 描述对象的特征的数据

– 行为

- 对象自身的行为，对现实世界某些信息的响应。

– 消息

- 对象之间通过传递消息相互影响，消息可简单理解为传递给被调用函数的实参。



(4) 程序范型对比：猫狗的结构化程序

```
struct Dog{
```

```
    char name[10], owner[10];  
    int high;
```

```
};
```

Cat tom;

```
void setDogName(Dog *dog, char name[10]) {strcpy(dog->name, name);}  
void setCatName(Cat *cat, char name[10]) {strcpy(cat->name, name);}  
void Growth(Dog *dog, int h1, Cat *cat, int h2) {dog->high = h1; cat->high = h2;}
```

```
void SetOwner(Cat *cat, char w1[], Dog *dog, char w2[]){  
    strcpy(cat->owner, w1); strcpy(dog->owner, w2);  
}
```

```
char * getDogOwner(Dog dog) { return dog.owner; };
```

```
int getCatHeight() { return tom.high; }
```

```
int main(int argc, _TCHAR* argv[])  
{
```

Dog dog1, dog2, dog3;

```
strcpy(dog1.name, "joo");  
setDogName(&dog1, "joo");  
SetOwner(&tom, "zhang", dog1, "wang");  
Growth(&dog1, 10, &tom, 5);  
printf("%s %s", getDogOwner(dog1), dog1.name );  
printf("%d", getCatHeight());  
return 0;
```

```
}
```

```
struct Cat{
```

```
    char name[10], owner[10];  
    int high;
```

```
};
```



**Class Cat &
class Rat**



Class Dog

如果修改high
或name的类型
，可能引起全
程中大量的代
码修改！

(5) 程序范型对比：猫狗的 O O P 程序

```
struct Dog{
private:
    char name[10], owner[10];
    int high;
public:
    void setName(char Name[10]) { strcpy_s(name, name); }
    void Growth(int h)          { high += h; }
    void setOwner(char Owner[10]) { strcpy_s(owner, Owner); }
    char* getName()             { return name; }
    .....
```

如果修改high
或name的类
型，可能只需
修改Dog内部
的函数！



```
};
struct Cat{
    char name[10], owner[10];
    float high;
public:
    void setCatName(...) { ..... }
    void Growth(int h) { .....; }
    void setOwner(.....)
    char* getName() { ..... }
    char* getOwner() { ..... }
    float getHigh() { return high; }
};
```



```
Cat tom;
int main(int argc, _TCHAR* argv[])
{
    Dog dog1, dog2, dog3;
    dog1.setName("joe");
    dog2.setOwner("john");
    dog1.Growth(2);
    //dog1.high=9; error
    .....
    printf("%d", tom.getHigh());
    return 0;
}
```

1.1.2. 面向对象程序设计

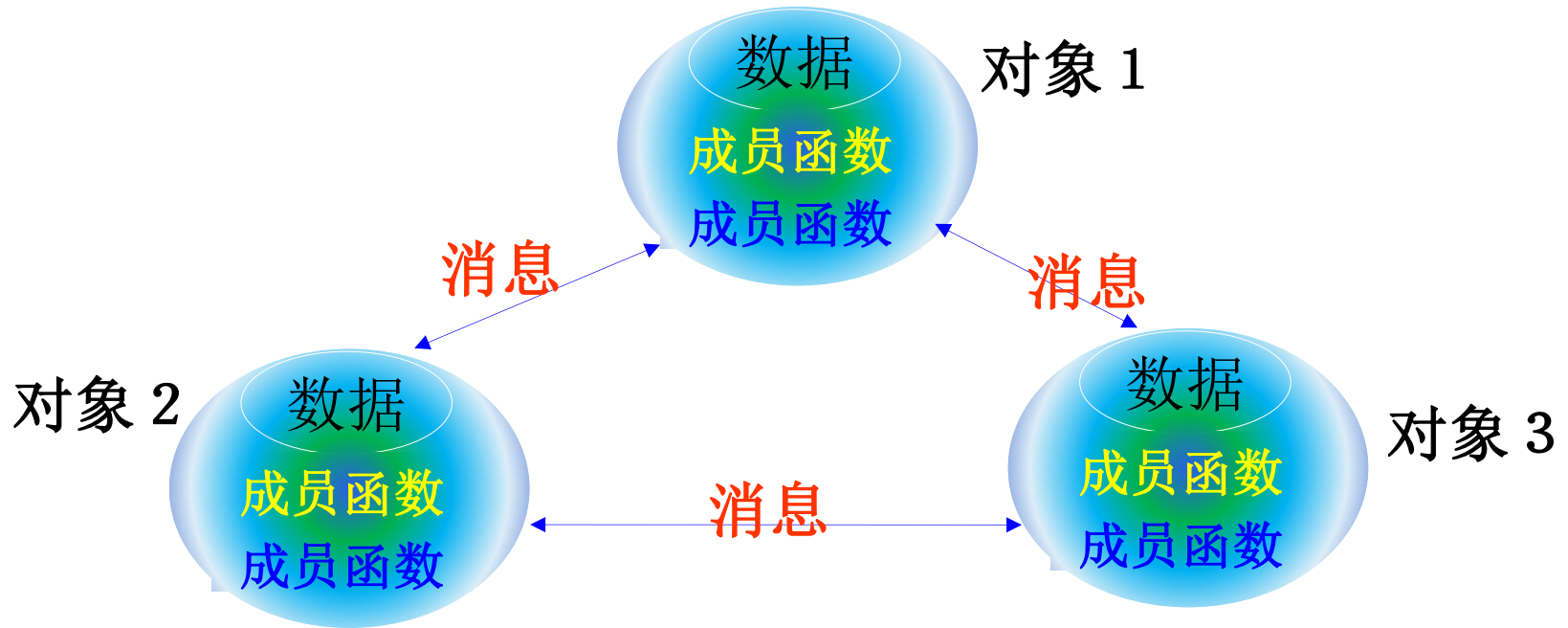
(6) 面向对象的通信录程序的类形式

```
struct Person{  
    private:  
        char name[10];  
        char addr[20];  
        char phone[11];  
    public:  
        void InitData(){.....}  
        void SearchAddr(char *name){.....};  
        void SearchPhone(char *name){.....};  
};
```

在C++中，struct的功能被扩展了，在struct中不仅可以定义数据，还可以定义函数。数据与函数构成了一个整体。其中的private和public是访问权限。

1.1.2. 面向对象程序设计

(7) 面向对象的程序范型



- 将客观事物的属性和行为抽象成数据和操作数据的函数，并把它们组合成一个不可分割的整体（即对象）的方法能够实现对客观世界的真实模拟，反映出世界的本来面目。从客观世界中抽象出一个个对象，对象之间能够传递消息。

1.1.3 面向对象程序设计语言的特征

本节主要介绍类与对象的基本概念，以及面向对象程序设计的主要特征。

1.1.3 面向对象程序设计语言的特征

1、抽象 (abstract)

- 抽象是指**有意忽略问题的某些细节和与当前目标无关的方面**，以便把问题的本质表达得更清楚。
- 抽象描述了一个对象的**基本特征**，可以将该对象与所有其他类型的对象区分开来，因此需要为对象定义出清晰的边界概念。
- 抽象关注一个对象的**外部视图 (边界外部)**，用来分离对象的基本行为和实现 (**边界内部**)。
- 可以理解为抽象关注接口 (**对象边界外部**)，即可观察到的行为；而**封装**则关注这些行为的实现 (**边界内部**)。
- 抽象的结果是形成对应客观事物的抽象数据类型，简称**ADT** (Abstract Data Type)



抽象结果

并未实现内部函数

类型名称

Light

接口

On()

Off()

Brighten()

Dim()

电灯抽象：

忽略灯泡的形状、大小、品牌、内部实现细节，所有的灯都具有开、关、调节明、暗的按钮，这就是灯的接口和功能，用类**Light**将它们组织在一起，就形成了灯的**ADT**类型。

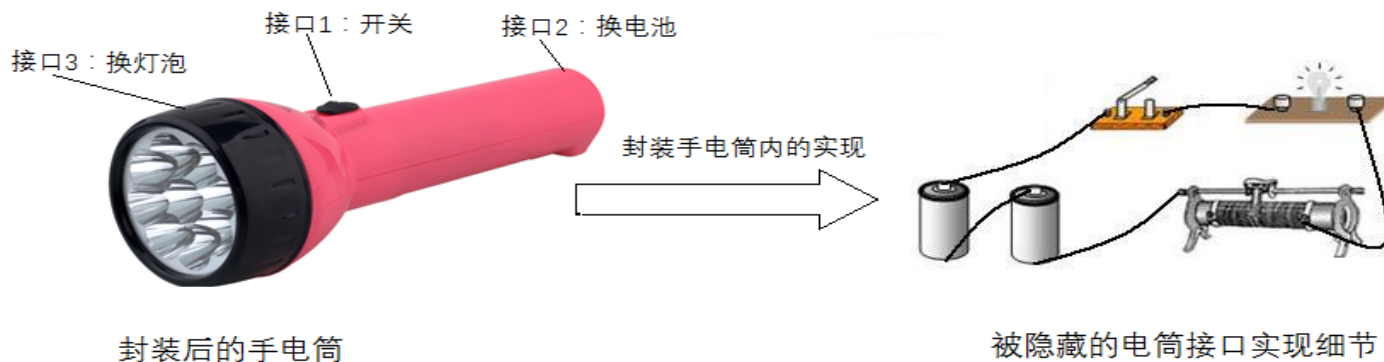
1.1.3 面向对象程序设计语言的特征

2、封装（encapsulation）

（1）封装是对ADT的具体实现。它将抽象出的特征（用数据表示）和行为（用函数表示）捆绑成一个整体，并且编码实现抽象所设计的接口功能。

（2）封装形成接口与实现的分离。从外面看只能看到对象的外部特性，即能够受理哪些信息，具有哪些处理能力，可供其它对象调用，称为**接口**；对象的内部，即处理能力的执行细节和内部状态，称为**实现**，对外是不可见的。

（3）信息隐藏。从对象外面不能直接使用对象的处理能力，也不能直接修改其内部状态，对象的内部状态只能由其自身改变。



1.1.3 面向对象程序设计语言的特征

(4) 封装的实现方式

class classname

{

public:

//public members

//friend function

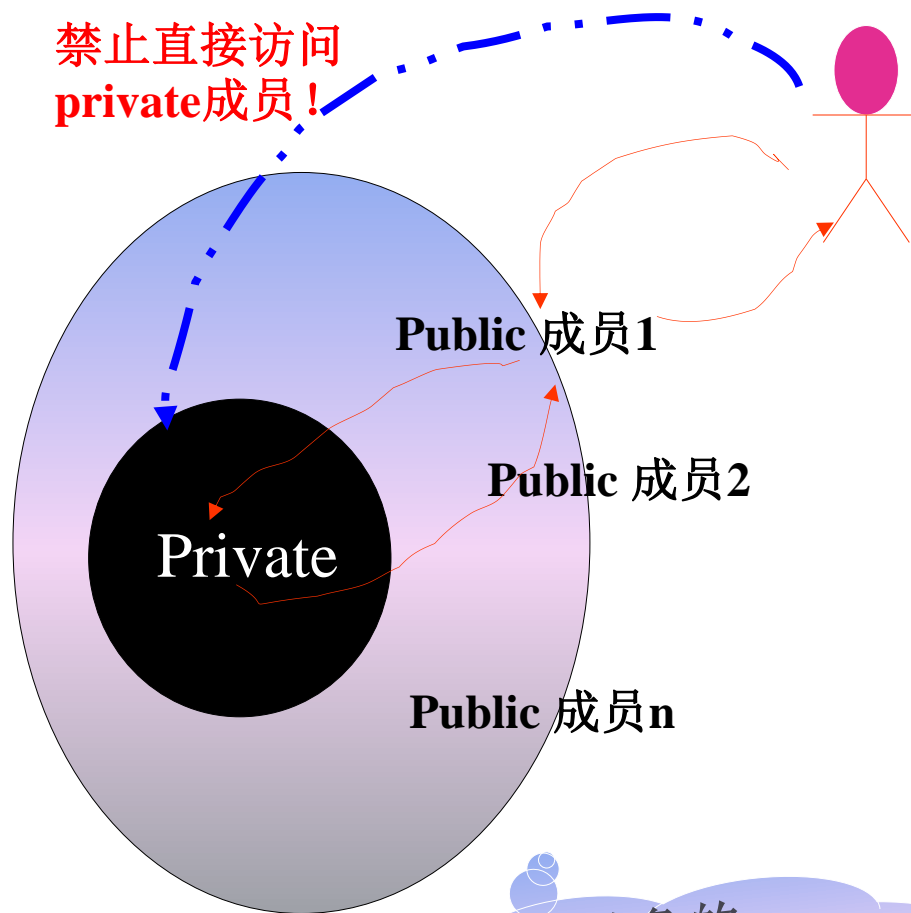
private:

//private members

};

边界

禁止直接访问
private成员!



一个class定义了一种抽象的数据类型,用户只能访问public成员,不能直接访问private成员

1.1.3 面向对象程序设计语言的特征

(5) 封装的优点

- 1.降低部件间的耦合度，提高部件的独立性
- 2.具有隐藏性和安全性（如银行的帐户）
- 3.易于维护(由于数据独立,易于发现问题)
- 4.封装将对象的使用者与设计者分开,使用者只需要通过接口访问对象,不必须了解对象的内部细节，能够有效地实现软件复用。

(6) 封装的缺点

- 可能需要更多的输入输出函数。

1.1.3 面向对象程序设计语言的特征

3、继承 (inheritance)

(1) 基本概念

某类（派生类）对象可以继承另外一类对象（基类）的特征和功能。

(2) 实现方式：

派生类复制了基类的数据和函数

(3) 继承的特性

- 类间具有共享特征（包括数据和程序代码的共享）：遗传
- 类间具有细微差别或新增部分（包括非共享的程序代码和数据）：变异
- 类间有层次结构（如同人类通过继承构成了家族关系一样）



狗爷爷



狗儿子



狗孙子



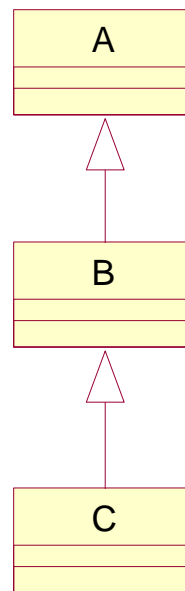
1.1.3 面向对象程序设计语言的特征

(4) 继承的类型

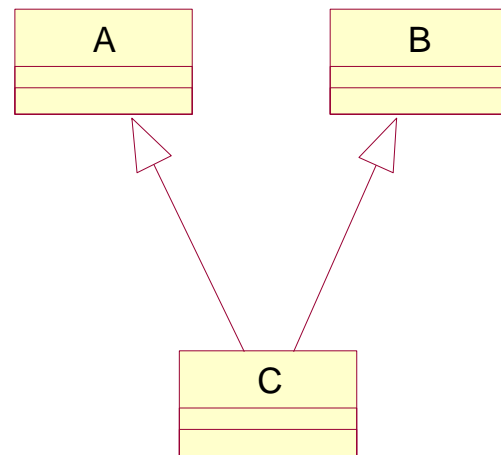
- 从继承源上划分：单继承（一个派生类只有一个基类）、多继承（一个派生类有多个基类）
- 从继承内容上划分：取代继承、包含继承、受限继承、特化继承。

(5) 继承的作用：

- 实现软件的可重用性
- 实现软件的独立性
- 增加软件的可维护性



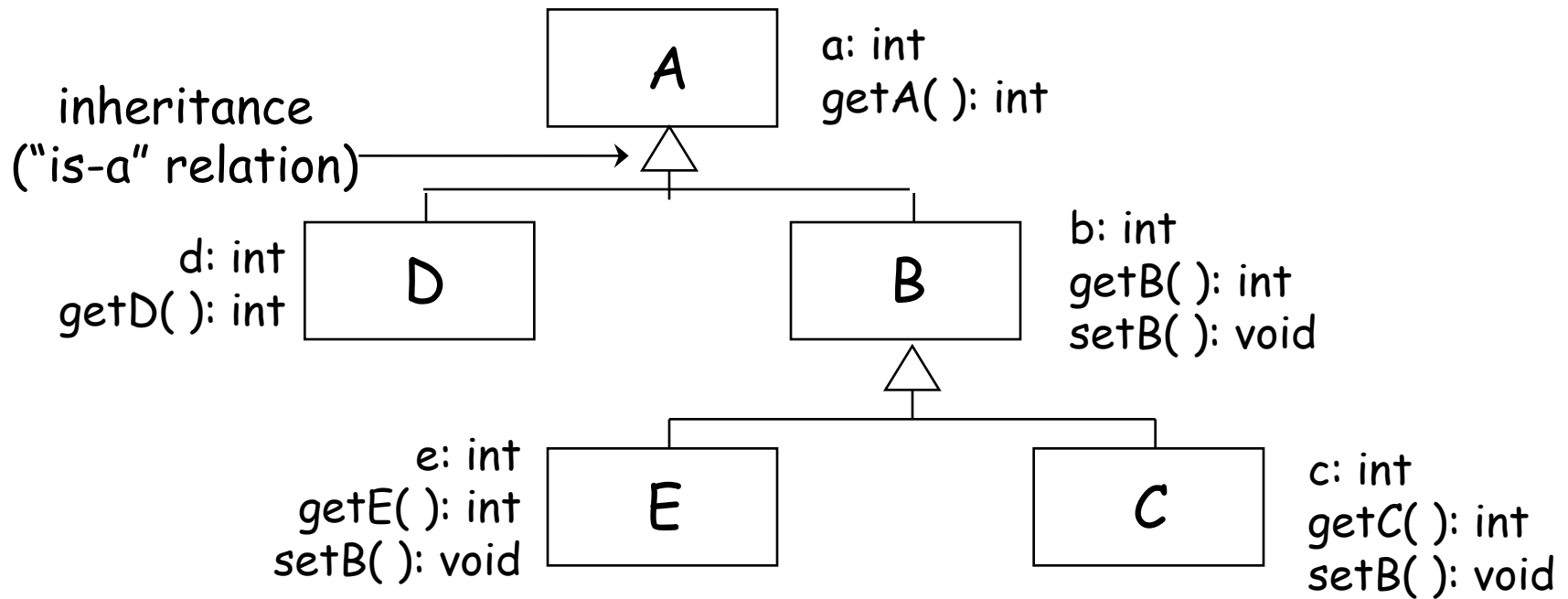
单继承



多继承

1.1.3 面向对象程序设计语言的特征

(6) 继承的层次结构



—讨论：E具有哪些成员？

1.1.3 面向对象程序设计语言的特征

4、多态 (polymorphism)

– 概念

对象根据所接受的消息而做出动作，同样的消息为不同的对象接受时可导致完全不同的行动，该现象称为**多态性**。简单的说：**单接口，多实现**。

当猎人的枪声响起时，不同动物开始run()，
它们不同的奔跑方式就是多态

– 作用

- 方便软件功能的扩展

– 举例

```
f(动物 *p) { p->run(); }
```

f即为多态函数，当
传递狗对象给p时，
执行**狗.run()**
传递猫对象给p时，
执行**猫.run()**.....



动物

1.2 C++语言概述

• 1.2.1 C++简史

1.面向对象语言概况

- 最早最成熟的领域

- 萌芽于60年代（Simula），成熟于80年代
- 70年代,纯面向对象语言,smalltalk

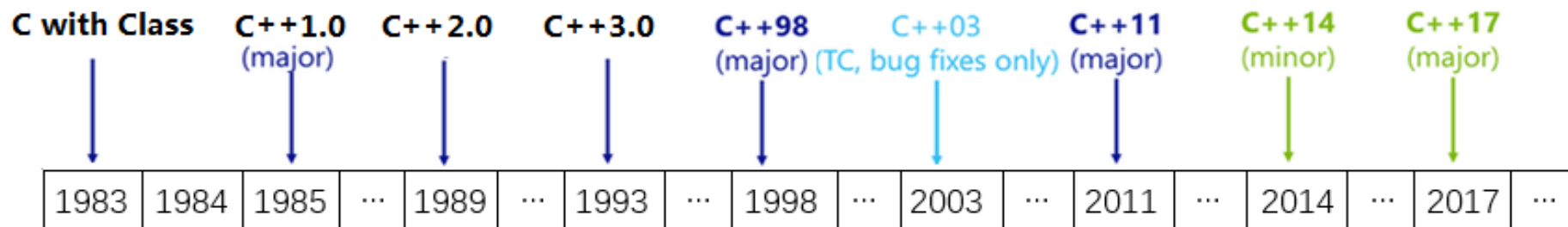
- 常见面向对象的语言

- Java: 纯面向对象的语言
- Python, R
- C++, Object Pascal

1.2.1 C++语言简史

2、C++简史

- C++起源
 - 20世纪80年代, AT&T Bell 实验室 Bjarne Stroustrup
- C++发展
 - B → C → 带类的C → C++ → 标准C++ → 托管C++
- C++标准的修订



传统C++ | 标准C++

1.2.1 C++语言简史

- **3. 传统C++与标准C++**

- **6次主要修订：1985年，1990年，1998年，2011年（C++ 11）、2014年、2017年、2020年**
- **1998年确定的版本为标准C++，之前的称传统C++。**
- **标准C++包括传统C++的全部功能，且更庞大、全面**
- **.h和无扩展名的头文件：**
 - 传统C++为.h头文件，如*iostream.h*、*fstream.h*、*string.h*；
 - 标准C++对应的头文件为：*iostream*、*fstream*、*string*。

1.2.2 C++的特点

C++在对**C**语言进行扩展的同时，保留了**C**语言的原有特征和优点，具有以下特征：

- 兼容 C 语言，支持面向过程程序设计
- 扩展 C 语言，支持面向对象程序设计
- 丰富的运算符和数据类型
- 高效性
- 灵活性
- 可移植性强

1.2.3 C++程序的结构

1、C++程序的构成

- 声明部分
- 主函数部分
- 函数定义

2、C++程序文件

- 头文件: **.h .hpp**
- 源文件: **.cpp**

3、程序结构的一个例程如下

1.2.3 C++程序的结构

```
0  // Eg1-1.cpp
1  #include <iostream>
2  #define      N      10
3  void sort(int a[], int n);
4  void print(int a[], int);
5  using namespace std;
6  void main() {
7      int a[N];
8      cout<<"input 10 numbers:\n";
9      for(int i = 0; i < N; i++)
10         cin>>a[i];
11      sort(a, N);
12      print(a, N);
13  }
14
15  void sort(int a[], int n) {
16      for(int i = 0; i < n-1; i++) {
17          for(int j = i+1; j < n; j++) {
18              if(a[i] < a[j]) {
19                  int t = a[i];
20                  a[i] = a[j];
21                  a[j] = t;
22              }
23          }
24      }
25  }
26  void print(int a[], int n) {
27      for(int i = 0; i<n; i++) {
28          cout<<a[i]<<" ";
29          cout<<endl;
30      }
31  }
```

声明部分

主函数部分

函数定义部分

1.2.3 C++程序的结构

- 对例程的补充：
 - C++的注释
 - `//`
 - `/* */`
 - C++的函数需要先声明，然后才能调用

1.2.4 标准C++程序设计

1. 标准C++程序设计的概念

- ANSI/ISO标准化委员会1998年完成的C++98标准，人们称之为标准C++，按此版本及之后的规范进行编程均称之为**标准C++程序设计**。而以前C++规范进行编程，则被称为**传统C++程序设计**。
- C++98 VS C++11、
 - C++98称得上最经典，目前应用仍然广泛。
 - C++11注入了当前OOP语言的许多新特征，如**类型自动推断、范围for，lambda函数、移动函数，构造函数继承，类内初始值列表**，为程序设计带来了许多方便，与JAVA等语言的某些程序特征更为接近和溶恰，是当前C++编程的首选！
 - C++14、C++17及C++20对C++11在C++11基础上进行了逐步修改，但本质上的变化不大。

1.2.4 标准C++程序设计

2. 标准C++与传统C++的编程差异

(1) 系统库函数头文件区别

- 传统C++为.h头文件。如
 - iostream.h、fstream.h、string.h、stdio.h、math.h
- 标准C++为同名无.h文件：
 - iostream、 fstream、 string、 cstdio、 cmath

(2) 命名空间限定

- 传统C++的库函数：直接调用函数就行了。
- 标准C++中的任何内容（C库函数除外）则用“std::”前缀限定，函数全名是“std::x”
 - x可以是函数、常量、数据结构、系统变量等内容。

1.2.4 标准C++程序设计

• 3. 标准C++程序和传统C++程序对比

//Eg1-2.cpp :传统C++

#include <iostream.h>

#include <stdio.h>

#include <math.h>

void main() {

int x;

cout<<"输入数字: ";

scanf("%d", &x);

bool prime=true;

for(int i=2;(i<=x-1)′i++)

if(x%i==0) prime=false;

if(prime)

cout<< x<<"是素数!"<<endl;

else

cout<<x<<"不是素数!"<<endl;

}

库函数
头文件
差异

//Eg1-2.cpp :标准C++

#include <iostream>

#include <cstdio>

#include <cmath>

void main() {

int x;

std::cout<<"输入数字: ";

scanf("%d", &x);

bool prime=true;

for(int i=2;(i<=x-1)′i++)

if(x%i==0) prime=false;

if(prime)

std::cout<< x<<"是素数!"<<std::endl;

else

std::cout<<x<<"不是素数!"<<std::endl;

}

库函数引
用差异

1.2.4 标准C++程序设计

- 4. 标准库的引用

- 用 **std::x** 限定标准库中的标识符（如 **std::cin**）
- 用 **using std::x** 将标准库中的 **x** 引入程序后，然后在程序中直接用 **x**
- 用 **using namespace std**；一次性引入包含标准库头文件中的全部标识，然后在程序中直接应用。如 **Eg1-3** 所示。

1.2.4 标准C++程序设计

//Eg1-3.cpp

```
#include <iostream>
```

```
#include <cstdio>
```

```
#include <cmath>
```

```
using namespace std;
```

```
void main() {
```

```
    int x;
```

```
    cout<<"输入数字: ";
```

```
    scanf("%d", &x);
```

```
    bool prime=true;
```

```
    for(int i=2;(i<=sqrt(x))&&prime;i++)
```

```
        if(x%i==0) prime=false;
```

```
    if(prime)
```

```
        cout<< x<<"是素数!"<<endl;
```

```
    else
```

```
        cout<<x<<"不是素数!"<<endl;
```

```
}
```

若输入a?



1.3 数据输入与输出

- 本节主要介绍**C++**程序中数据输入输出的方法，是**C++**程序设计的基础。应该掌握
 - 流的概念
 - **iostream**
 - **cin**
 - **cout**
 - 数据输入常见问题及解决方法

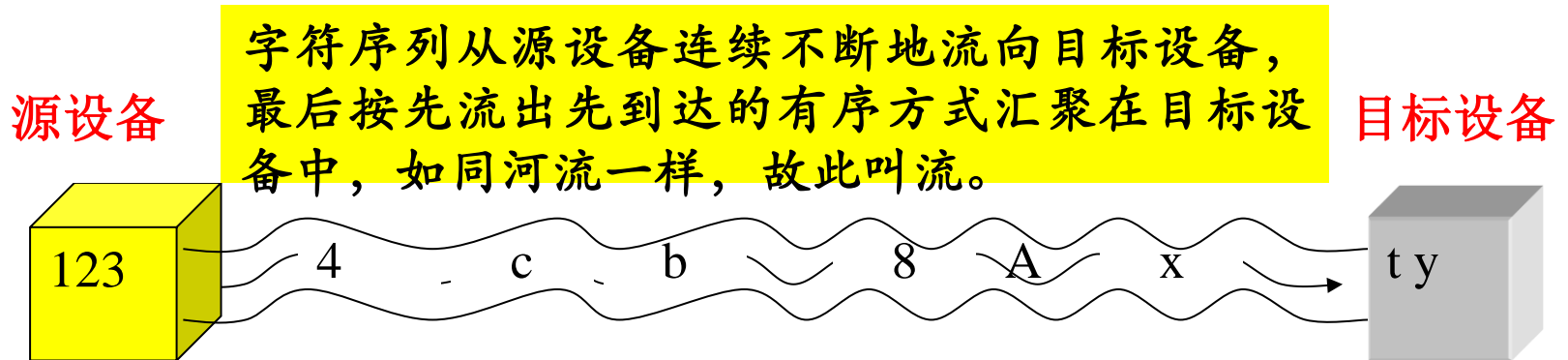
1.3.1 数据类型

- **C++基本类型**
 - int,char,long,wchar_t,char16_t,char32_t
- **实型**
 - float,double,long double
- **逻辑类型和空类型**
 - bool, void
- **自定义类型**
 - struct,class,union,enum,指针,数组
- **STL中的类型**
 - vector,string,list,stack,map,set.....
- **C++11: long long**

1.3.2 流的概念

1、C及C++中的流概念

- I/O (input/output, 输入/输出) 数据是一些从源设备到目标设备的字节序列, 称为字节流。除了图像、声音数据外, 字节流通常代表的都是字符, 因此在多数情况下的流 (stream) 是从源设备到目标设备的字符序列,



1.3.2 流的概念

– 输入流:istream

- 输入流（**input stream**）是指从输入设备流向内存的字节序列。

– 输出流:ostream

- 输出流（**output stream**）是指从内存流向输出设备的字节序列。

– C++中的输入输出流

- **iostream** : **istream**, **ostream**
- **cin** 输入流对象，C++已将其与键盘关联
- **cout** 输出流对象，C++已将其与显示器关联

内存



1.3.3 cin和提取运算符>>

1、cin的用途

- **cin**是**istream**中用**istream**定义的一个输入流对象，它被默认关联到了标准输入设备（键盘），定义类似于：

istream cin;

- 在**C++**程序，主要用**cin**从键盘输入数据到内存变量中，当然也可以使用**scanf**函数，但**cin**更简单。

2、cin的用法

- 输入单个变量的值
cin>>x;
- 输入多个变量的值
cin>>x1>>x2>>x3>>x4.....>>xn

其中**x,x1,.....,x2**可以是内置数据类型如**int, char, float, double**等。

1.3.3 cin和提取运算符>>

3、用cin时的注意事项

- 在一条cin语句中同时为多个变量输入数据。
各输入数据之间用一个或多个空白（包括空格、回车、Tab）作为间隔符，全部数据输入完成后，按Enter键结束。
- 在>>后面只能出现变量名，下面的语句是错误的。

cin>>"x=">>x; //错误，>>后面含有字符串"x="

cin>>12>>x; //错误，>>后面含有常数12

cin>>'x'>>x; //错误，>>后面含有字符'x'

1.3.3 cin和提取运算符>>

– cin具有自动识别数据类型的能力

- 提取运算符>>将根据它后面的变量的类型从输入流中为它们提取对应的数据。比如：

cin>>x;

- 假设输入数据**2**，提取运算符>>将根据其后的**x**的类型决定输入的**2**到底是数字还是字符。
 - 若**x**是**char**类型，则**2**就是字符；
 - 若**x**是**int**，**float**之类的类型，则**2**就是一个数字。
- 再如，若输入**34**，
 - 若**x**是**char**类型，则只有字符**3**被存储到**x**中，**4**将继续保存在流中；
 - 若**x**是**int**或**float**，则**34**就会存储到**x**中。

1.3.3 cin和提取运算符>>

– 数值型数据的输入

- 在读取数值型数据时，提取运算符>>首先略掉数据前面的所有空白符号，如果遇到正、负号或数字，就开始读入，包括浮点型数据的小数点，并在遇到空白符或其他非数字字符时停止。例如：

```
int x1;
```

```
double x2;
```

```
char x3;
```

```
cin>>x1>>x2>>x3;
```

- 假如输入“35.4A”并按Enter键，x1是35；x2是.4；x3是'A'

1.3.3 cin和提取运算符>>

4、输入数据案例分析 p16

【例1-4】 假设有变量定义语句如下：

```
int a,b;  
double z;  
char ch;
```

下面的语句说明数据输入的含义。

语句

输入

内存变量的值

1 cin>>ch;	A	ch='A'
2 cin>>ch;	AB	ch='A', 而'B'被保留在输入流中等待被读取
3 cin>>a;	32	a=32
4 cin>>a;	32.23	a=32, .23留在输入流中等待被读取
5 cin>>z;	76.21	z=76.21
6 cin>>z;	65	z=65.0
7 cin>>a>>ch>>z	23 B 3.2	a=23, ch='B',Z=3.2
8 cin>>a>>ch>>z	23B3.2	a=23, ch='B',Z=3.2
9 cin>>a>>b>>z	23 32	a=23, b=32, 等待输入下一个数据存入z
10 cin>>a>>z	2 3.2 24	a=2, z=3.2, 24被保留在输入流中等待被读取
11 cin>>a>>ch	132	a=132, 计算机等待输入 ch的值
12 cin>>ch>>a	132	ch='1', a=32

1.3.4 **cout**和插入运算符<<

1、**cout**的用途

- **cout**是*iostream*库中用*ostream*定义的一个输出流对象，类似于：

ostream cout;

- **cout**已被C++默认关联到显示器，用于在屏幕上输入数据。在C++程序中，也可使用C语言的*sprintf*输出数据，但**cout**更简单。

2、**cout**的用法

cout<<x;

- 其中**x**可是以内置数据类型如**int**，**char**，**float**等。

1.3.4 **cout**和插入运算符<<

3. 输出字符类型的数据

对于字符变量和字符串变量，**cout**将把变量的值输出到显示屏幕上。对于字符常量和字符串常量，**cout**将把它们原样输出在屏幕上

【例1-5】 **cout**输出字符数据。

//Eg1-5.cpp

```
#include<iostream.h>
void main()
{
    char ch1='c';
    char ch2[]="Hellow C++!";
    cout<<ch1;
    cout<<ch2;
    cout<<"C";
    cout<<"Hello everyone!";
}
```

1.3.4 **cout**和插入运算符<<

4. 连续输出

- **cout**能够同时输出多个数据，用法如下：

cout<<x1<<x2<<x3<<...;

例：

cout<<ch1<<ch2<<"C"<<"Hello everyone!";

- 与**C**一样，在**C++**中也可以将一条命令写在多行上。比如，上面的语句也可写成下面的形式：

cout<<ch1

<<ch2

<<"C"

<<"Hello everyone!";

1.3.4 **cout**和插入运算符<<

5. 输出换行

在**cout**语句中换行可用：“\n”或**endl**

【例1-6】 在例1-5的输出语句中增加换行符。

// **Eg1-6.cpp**

```
#include<iostream.h>
```

```
void main(){
```

```
    char ch1='c';
```

```
    char ch2[]="Hello C++!";
```

```
    std::cout<<ch1<<std::endl;
```

```
    std::cout<<ch2<<"\n";
```

```
    std::cout<<"C"<<endl;
```

```
    std::cout<<"Hello everyone!\n";
```

```
}
```

1.3.4 **cout**和插入运算符<<

6. 输出数据间隔符

在连续输出多个数据时，应注意在数据之间插入间隔符。如

```
int x1=23;
```

```
float x2=34.1;
```

```
double x3=67.12;
```

```
cout<<x1<<x2<<x3<<900;
```

- 其中的**cout**语句将在屏幕上输出，

2334.167.12900

谁知道这是个什么数据呢？因此需要插入间隔符

```
cout<<x1 << ' ' <<x2 << ' ' <<x3 << ' ' <<900;
```

输出： 23 34.1 67.12 900

1.3.5 输出格式控制符

1 . 设置浮点数的精度

`setprecision(n)` //最后一有效位将对其右边数字 4 舍 5 入

```
cout<<setprecision(3)<<3.14126<<" "<<2.4576<<endl;
```

输出: 3.14 2.46

2 . 设置输出域宽和对齐方式

`setw(n)`

3 . 设置对齐方式 (默认右对齐)

- `setiosflags(long f);`
- `resetiosflags(long f);`

f 取值为`ios::left` 或 `ios::right`

1.3.5 输出格式控制符

【例 1 - 7】用 **setiosflags** 和 **resetiosflags** 设置和取消输出数据的对齐方式。P20

//Eg1-7.cpp

```
#include<iostream>
```

```
#include<iomanip>
```

```
void main(){
```

```
std::cout<<"123456781234567812345678"
```

```
<< std:: endl;
```

```
std:: cout<<setiosflags(ios::left)<<setw(8)
```

```
<<456<<setw(8)<<123<< std::endl;
```

```
std:: cout<<resetiosflags(ios::left)<<setw(8)
```

```
<<123<< std:: endl;
```

```
}
```


1.3.5 输出格式控制符

4. 输出填充字符(用指定字符填充空白) —— 补充内容

```
std::cout.fill(ch);  
std::cout<< std::setfill(ch);
```

【例】用fill和setfill设置输出填充字符。

//ch1-fill.cpp

```
#include<iostream>
```

```
#include<iomanip>
```

```
void main()
```

```
{
```

```
    std::cout<<"123456781234567812345678"<<std::endl;
```

```
    std::cout<<std::setw(8)<<123<<std::setw(8)<<456<<std::setw(8)<<789<<std::endl;
```

```
    std::cout.fill('@');
```

```
    std::cout<<std::setw(8)<<123<<std::setw(8)<<456<<std::setw(8)<<789<<std::endl;
```

```
    std::cout<<std::setfill('^');
```

```
    std::cout<<std::setw(8)<<123<<std::setw(8)<<456<<std::setw(8)<<789<<std::endl;
```

```
}
```

1.3.6 数制基数

- 数制基数操纵符（在*iostream*头文件中定义）
 - **hex**: 16进制 **oct**: 8进制, **dec**: 10进制
- 不同进制的常量表示
 - **0x/0X** 16制常量
 - **0** 表示八进制常量
 - **0b/0B** 表示二进制 **C++14**
 - 单引号 对数字进行分位表示 **C++14**
- 输入不同进制的数据
 - 在**cin**输入流中先插入数制操纵符, 再输入数据
- 输出不同进制的数据
 - 在**cout**输出流中先插入数制操纵符, 再输出数据
- 注意
 - 数制基数设置后将一直有效, 直到下一次设置新数制基数才取消。

1.3.6 数制基数

【例1-8】 数字常量的进制控制和千分位间隔表示。

// Eg1-8

P21

```
#include <iostream>
using namespace std;
void main() {
    int x1 = 23;
    int x2 = 023;
    int x3 = 0x23;
    int x4 = 0b11011101;
    int x5 = 6'123'456'789;
    int x6 = 0b101'111'001'111;
    cout<< "x1 = " << x1 << "\tx2 = "
         << x2 << "\tx3 = " << x3 << "\n"
         << "x4 = " << x4 << "\tx5 = " << x5
         << "\tx6 = " << x6 << endl;
    getchar();
}
```

// C++ 14
// C++ 14
// C++ 14

1.3.6 数制基数

//Eg1-9.cpp 不同进制数据的输入输出及转换

P22

```
#include<iostream>
using namespace std;
void main(){
    int x=34;
    cout<<hex<<17 <<" "<<x<<" "<<18<<endl;
    cout<<17 <<" "<<oct <<x<<" "<<18<<endl;
    cout<<dec<<17 <<" "<<x<<" "<<18<<endl;
    int x1, x2, x3, x4;
    cout<<"输入 x1(oct), x2(oct), x3(hex), x4(dec):"<<endl;
    cin>>oct>>x1;           //八进制数
    cin>>x2;                 //八进制数
    cin>>hex>>x3;            //输入十六进制数
    cin>>dec>>x4;            //输入十进制数
    cout<<"x1="<<x1<<"\tx2="<<x2<<"\tx3="<<x3<<"\tx4="<<x4<<endl;
}
```


1.3.7 string与字符串输入/输出

3 . string类型的赋值

- string类型的赋值操作与int等基本类型的赋值操作相同，不必用strcpy函数。例如，

```
string s1, s2, s3[3];           //定义string对象及数组
```

```
//string对象数组定义与初始化
```

```
string name[3] = { "tom","jerry","duck" };
```

```
s1 = "this is a string!";      //string赋值
```

```
s2 = s1;
```

```
s3[0] = s1;                    //string数组元素访问
```

```
s3[1]="string arr";
```

1.3.7 **string**与字符串输入/输出

4 . **string**类型的连接

“+、+=”可以连接两个**string**类型对象，例如：

```
string s1("I am a boy"), s3;
```

```
string s2 = "i come from china!";
```

```
s3 = s1 + "," + s2;    //s3: I am a boy, i come from china!
```

```
s1 += "," + s2;        //s1: I am a boy, i come from china!
```

1.3.7 string与字符串输入/输出

5. string类型的输入输出和大小比较

- string类型可以用cin和cout直接输入或输出；用“>、>=、==、<、<=、!=”进行大小比较，比较的是两个string对象对应位置字符的Ascii码。

//Eg1-10.cpp

```
#include<iostream>
```

```
#include<string>
```

```
using namespace std;
```

```
int main(){
```

```
    string s1,s2,big;
```

```
    cout << "输入两个字符串:" << endl;
```

```
    cin >> s1 >> s2;
```

```
    cout << "参加比较的两个字符串是: " << s1 << ", " << s2 << endl;
```

```
    if (s1 > s2) big = s1;
```

```
    else if (s1 == s2) big = "same";
```

```
    else big = s2;
```

```
    cout << "大字符串是: " << big << endl;
```

```
    return 0;
```

```
}
```


1.3.8 数据输入的典型问题

1. 输入数据类型不匹配引发的问题

- 即使程序完全正确，但输入数据有问题，程序也可能出现运行错误，甚至无法正常运行。

```
//Eg1-11.cpp
#include<iostream>
using namespace std;
void main(){
    int a, b;
    double z;
    char ch;
    cin>>ch;
    cin >> a>>b;
    cin >> z;
    cout << "ch=" << ch
        << "\ta=" << a
        << "\tb=" << b
        << "\tz=" << z<< endl;
}
```

错误原因：输入类型不对的数据，**cin**会设置输入失效位，并关闭输入流！影响后续输入语句的执行！

cin依次从输入流读数据，**A**被提取给**ch**后，**B**将被提取给**a**，因类型不对，导致**C++**关闭输入流！因而**a\b\z**都是未初始化的值！

AB 32 49 8.7 //VC6.0键盘输入, 将产生下面的错误输出
ch=A a=-858993460 b=-858993460 z=-9.25596e+61

1.3.8 数据输入的典型问题

2. 为变量输入空白字符的问题

- **cin** 输入数据时,空白作为数据之间的间隔,无法输入
- 可用**cin**的**get**或**getline**成员函数输入

```
char c1,c2;
```

```
int n;
```

```
std::cin>>c1>>c2>>n;
```

若输入: X 5

则X将存入c1, 5被存入c2, n没有输入值

1.3.8 数据输入的典型问题

(1) **get**输入空白字符

- **get**输入流函数完成单个空白字符（包括空格、回车换行等、**Tab**等）的输入，
- **get**函数的用法如下：

std::cin.get(char varChar);

例如：

char c1,c2;int n;

std::cin.get(c1);

std::cin.get(c2);

std::cin>>n;

若输入 1 3,则C1为1, C2为空白字符 , n为3

1.3.8 数据输入的典型问题

(2) getline输入包括空白字符的长字符串

– **getline**函数一次读取一行字符，其用法如下

std::cin.getline(char *c ,int n, char ='\n');

功能：为字符数组**c**输入一行可包括空白的字符串

结束方式1：为**c**读取了**n-1**个字符；(第**n**个位置存放'\0')

结束方式2：读取到了指定的结束符，默认为回车符

【例1-12】 getline读取一行输入存入字符串中 P25

```
#include<iostream>
```

```
void main(){
```

```
    char s1[100];
```

```
    std::cout<<"use getline input char: ";
```

```
    std::cin.getline(s1, 10);
```

```
    std::cout<<s1<<std::endl;
```

```
}
```

1.3.8 数据输入的典型问题

3. 前一条getline输入过多符号

【例1-13】从键盘为两个字符串输入数据，字符串中可能包括空白字符。

//Eg1-13.cpp

```
#include<iostream>
```

```
using namespace std;
```

```
void main(){
```

```
    char s1[100];
```

```
    char s2[10];
```

```
    cout << "use getline input s1: ";
```

```
    cin.getline(s1, 10);
```

```
    cout << "input s2: " << endl;
```

```
//    cin.clear();
```

```
//    cin.ignore(1024, '\n');
```

```
    cin.getline(s2, 6);
```

```
    cout << "s1=" << s1 << endl;
```

```
    cout << "s2=" << s2 << endl;
```

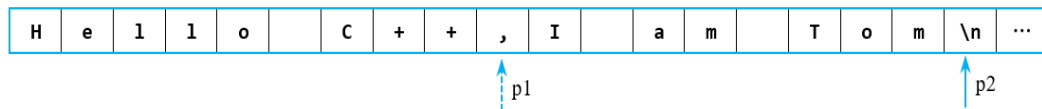
```
}
```

输入字符超过**getline**设置的字符长度，**getline**会设置输入失效位，并关闭输入流！
影响后续输入语句的执行

！

解决办法：**clear()**清理设置输入流有效，**ignore()**忽略多余符号。

使用此处注释掉的语句！



1.3.8 数据输入的典型问题

4.getline没有读数据

【例1-14】 设计一个程序，从键盘输入学生的学号和姓名，外国学生的姓名由first name和second name组成，两者之间用空白作间隔。

```
#include<iostream>
#include<string>
using namespace std;
void main() {
```

```
    int Sno;
    char name[10];
    cout << "input Sno: ";
    cin >> Sno;
    cout << "input name: ";
```

//getchar(); //或cin.get() 或 cin.ignore(1)

```
    cin.getline(name, 10);
    cout << "Sno:" << Sno << endl;
    cout << "name:" << name << endl;
```

紧接在读取数字等类型的
cin语句后，**getline**会读
取其前一条语句留在输入
法中的“\n”而结束

解决方法：“吃掉\n”，
比如用**getchar()**;

//L1



图 1-10 例 1-13 的输入流

1.4 编程实作——VC++ 2022编程简介

1、.NET平台主要功能

- **.NET**是基于因特网和**Web**的，它独立于任何语言或平台，对于程序开发的语言不作限制，开发者可以使用多种**.NET**兼容语言的任意组合创建**.NET**程序，这就允许在同一软件项目中，多个程序员分别使用自己精通的**.NET**语言编写程序代码。

1.4 编程实作——VC++ 2022编程简介

1、Visual C++

Visual C++是微软公司对**C++**的一个特定实现，支持**C**和**C++**程序设计，并在其中添加了微软公司的语言扩展。

Visual C++是一个集成开发环境，具有编辑、编译、调试、链接、装配和执行**C++**程序的功能。能够编制基于控制台和**Windows**平台的**C++**程序。

2、Visual C++版本

- Visual C++6.0
- Visual C++.NET

1.4 编程实作——VC++ 2022编程简介

3、Visual studio.net

- 托管程序设计
- 窗体程序设计
- FCL和CLR
- C#、J#、VC++、Visual Basic
- 多语言混合程序设计
- 垃圾回收

1.4 编程实作——VC++ 2022编程简介

1、Visual C++2022简介

- **visual studio 2022**中的编译器之一
- 支持标准**C++**和托管**C++**程序设计
- 托管**C++**采用新语法
- 支持**MFC**和窗体程序设计

1.4 编程实作——VC++ 2022编程简介

- **【例1-15】** 某次考试成绩如下，编写程序计算每位同学的平均分。要求成绩从键盘输入，程序输出结果的形式与下面相同，但要输出每位同学的平均分。
- 语文 数学 政治 化学 英语 平均分
- 学生1 67 76 87 89 76
- 学生2 78 87 78 90 87
- **程序设计思路：**
 - 设计一个二维数组s保存学生的成绩和平均分；设计一个读入学生成绩表的函数ReadData，该函数将学生成绩读入数组s的前5列中；
 - 设计一个计算平均成绩的函数AveScore，该函数计算每位同学的平均成绩，并将计算结果放入s数组的第6列；
 - 设计一个输出数据的函数OutData，该函数将s数组的数据按指定格式输出。

1.4 编程实作——VC++ 2022编程简介

- <1> 选择“**开始 | 所有程序 | Visual Studio 2022**”菜单命令，启动VC++ 2022。
- <2> 选择“**文件 | 新建 | 项目**”菜单命令，弹出“新建”对话框，如图1-11所示。
- <3> 在“新建”对话框的“位置”标签后面，单击“浏览...”，选择要保存源程序的目录。
- <4> 在“**名称**”对话框中的输入项目名称“**Eg1-22**”。然后单击“确定”按钮，然后单击弹出对话框中的“完成”按钮，进入Visual C++的编程序环境，

1.4 编程实作——VC++ 2022编程简介

/Eg1-15.cpp

#include <iostream>

#include <iomanip>

//setw在此头文件中定义

using namespace std;

#define StuNum 5

//StuNum代表学生人数

void ReadData(double s[][6],int n);

//这3行是函数声明

void AveScore(double s[][6],int n);

void OutData(double s[][6],int n);

void main(){

double s[StuNum][6];

//定义保存学生成绩的数组

ReadData(s,2);

//读入学生成绩

AveScore(s,2);

//计算各学生的平均分

OutData(s,2);

//输出学生成绩表

}

1.4 编程实作——VC++ 2022编程简介

```
void ReadData(double s[][6],int n){
    for(int i=0;i<n;i++){
        cout<<"输入学生 "<<i+1<<" 的5科成绩: ";           //提示输入学生成绩
        for(int j=0;j<5;j++)                                //输入学生的5科成绩
            cin>>s[i][j];
    }
}

void AveScore(double s[ ][6],int n) {
    for(int i=0;i<n;i++){
        double sum=0;
        for(int j=0;j<5;j++)
            sum=sum+s[i][j];
        s[i][5]=sum/5.0;
    }
}
```

1.4 编程实作——VC++ 2022编程简介

```
void OutData(double s[ ][6],int n) {           //在屏幕上输出科目名称
    cout<<setw(17)<<"语文"<<setw(8)<<"数学"<<setw(8)<<"政治"
        <<setw(8)<<"化学"<<setw(8)<<"英语"<<setw(8)<<"平均分"<<endl;
    for(int i=0;i<n;i++){
        cout<<setw(8)<<"学生 "<<i+1;
        for(int j=0;j<6;j++)
            cout<<setw(8)<<s[i][j];
        cout<<endl;
    }
}
```

作业

- 课后作业（教学立方提交）
 - 2道单选
 - 2道简答（课本 P32 1.2 1.6）
- 实验作业（实验课上检查）
 - 重现课本 P29 【例1-15】
 - 重现实验指导书 P6 【例1.1.3】
 - 上机完成题目（课本 P33 1.7 1.8）