§8. 类和对象的特性

8.1. 面向对象程序设计方法概述

暂时省略

§8.类和对象的特性

8.2.类的声明和对象的定义

8.2.1.类的引入

在结构体只包含数据成员的基础上，引入成员函数的概念，使结构体同时拥有数据成员和成员函数

8.2.2.声明类类型

```
class student {
    int num;
    char name[20];
    char sex;
    void display()
    {
        cout << "num:"  << num  << endl;
        cout << "name:" << name << endl;
        cout << "sex:"  << sex  << endl;
    }
};
```

```
struct student {
    int num;
    char name[20];
    char sex;
};
```

§8. 类和对象的特性

8.2. 类的声明和对象的定义

8.2.2. 声明类类型

★ 类类型的使用与结构体的使用方法基本相同

§7. 用户自定义数据类型

7.2.2. 结构体类型的定义

所有“结构体”替换为“类”，均有效

★ 结构体名,成员名命名规则同变量

★ 同一结构体的成员名不能同名,但可与其它名称(其它结构体的成员名,其它变量名等)相同

★ 每个成员的类型可以相同，也可以不同

★ 每个成员的类型既可以是基本数据类型，也可以是已存在的自定义数据类型

★ 每个成员的类型不允许是自身的结构体类型

★ 结构体类型的定义既可以放在函数外部,也可以放在函数内部

★ 结构体类型的大小为所有数据成员的大小的总和，可用sizeof(struct 结构体名) 计算，但不占用具体的内存空间(结构体变量占用一段连续的内存空间)

★ 在不同的编译系统中,有时为了加快程序运行速度，采用按数据总线宽度对齐的方法来计算结构体类型的大小，可能出现填充字节(需了解,本书不讨论)

★ C的结构体只能包含数据成员，C++还可以包含函数

§8. 类和对象的特性

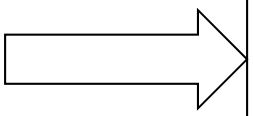
8.2. 类的声明和对象的定义

8.2.2. 声明类类型

★ 用sizeof(类名)计算类的大小时，成员函数不占用空间

★ 缺省情况下，类的数据成员和成员函数都是私有的，不能被外界所访问，因此是无意义的

★ 通过类的成员访问限定符(private/public)，
可以指定成员的属性是私有(private)或
公有(public)，私有不能被外界访问,公有
可被外界所访问，由实际应用决定
(通常数据成员private,成员函数public)



```
class student {
  private:
    int num;
    char name[20];
    char sex;
  public:
    void display()
    {
        cout << "num:"  << num  << endl;
        cout << "name:" << name << endl;
        cout << "sex:"  << sex  << endl;
    }
};
```

★ 在类的定义中，private/public出现的顺序，次数无限制

```
class student {
  public:
    void display()
    {
        cout << "num:"  << num  << endl;
        cout << "name:" << name << endl;
        cout << "sex:"  << sex  << endl;
    }
  private:
    int num;
    char name[20];
    char sex;
};
```

```
class student {
  private:
    int num;
  public:
    void display()
    {
        cout << "num:"  << num  << endl;
        cout << "name:" << name << endl;
        cout << "sex:"  << sex  << endl;
    }
  private:
    char name[20];
    char sex;
};
```

§8. 类和对象的特性

8.2. 类的声明和对象的定义

8.2.3. 定义类的实例对象

8.2.3.1. 先定义类，再定义对象

```
class student {
    ...
    };
student s1;
student s2[10];
student *s3;
```

```
struct student {
   ...
   };
struct student s1;
struct student s2[10];
struct student *s3;
```

★ 结构体变量/对象占用实际的内存空间，根据不同类型在不同区域进行分配

8.2.3.2. 在定义类的同时定义对象

```
class student {
    ...
    } s1, s2[10], *s3;
student s4;
```

```
struct student {
   ...
   } s1, s2[10], *s3;
struct student s4;
```

★ 可以再次用8.2.3.1的方法定义新的变量/对象

8.2.3.3. 直接定义对象(类无名)

```
class {
   ...
   } s1, s2[10], *s3;
```

```
struct {
   ...
   } s1, s2[10], *s3;
```

★ 因为结构体/类无名，因此无法再用8.2.3.1的方法进行新的变量/对象定义

§8. 类和对象的特性

8.2. 类的声明和对象的定义

8.2.4. 类与结构体的比较

★ 在C++中，结构体也可以加成员函数，能够实现和类完全一样的功能

```
class student {
  private:
    int num;
    char name[20];
    char sex;
  public:
    void display()
    {
        cout << "num:"  << num  << endl;
        cout << "name:" << name << endl;
        cout << "sex:"  << sex  << endl;
    }
};
```

替换为struct,功能完全相同

§8. 类和对象的特性

8.2. 类的声明和对象的定义

8.2.4. 类与结构体的比较

★ 在C++中，结构体也可以加成员函数，能够实现和类完全一样的功能

★ 若不指定成员访问限定符，则struct缺省为public，class缺省为private

```
class student {
    int num;
    char name[20];
    char sex;
    void display()
    {
        cout << num  << endl;
        cout << name << endl;
        cout << sex  << endl;
    }
};
```

全部是private

```
struct student {
    int num;
    char name[20];
    char sex;
    void display()
    {
        cout << num  << endl;
        cout << name << endl;
        cout << sex  << endl;
    }
};
```

全部是public

```
class student {
    int num;
    char name[20];
    char sex;
  public:
    void display()
    {
        ...
    }
};
```

私有

公有

```
struct student {
    int num;
    char name[20];
    char sex;
  public:
    void display()
    {
        ...
    }
};
```

公有

公有

§8. 类和对象的特性

8.3. 对象成员的访问(P.231 8.4提前)

书：8.4 对象成员的引用 => 访问更直观

8.3.1. 通过对象名访问对象中的成员

8.3.2. 通过指向对象的指针访问对象中成员

8.3.3. 通过对象的引用来访问对象中的成员

```
class student {
  private:
    int name[20];
    char sex;
  public:
    int num;
    void display()
    {
        ...
    }
};
```

类定义

```
int main()
{
    student s1, *s3=&s1;
    s1.num    = 10001; ✓
    (*s3).num = 10001; ✓
    s3->num   = 10001; ✓
    s1.display();      ✓
    (*s3).display();   ✓
    s3->display();     ✓
}
```

②

```
int main()
{
    student s1, s2[10];
    s1.sex = 'm';      ×
    s1.num=10001;      ✓
    s1.display();      ✓
    s2[0].sex = 'f';   ×
    s2[0].num=10002;   ✓
    s2[3].display();   ✓
}
```

①

```
int main()
{
   student s1, &s3=s1;
   s1.num = 10001; ✓
   s3.num = 10001; ✓
   s1.display();   ✓
   s3.display();   ✓
}
```

③

§8. 类和对象的特性

8.3. 对象成员的访问(P.231 8.4提前)

8.3.4. 访问规则

★ 只能访问公有的数据成员和成员函数

★ 数据成员可出现在其基本类型允许出现的任何地方

```
class student {
  private:
    int name[20];
    char sex;
  public:
    int num;
    void display()
    {
        ...
    }
};
```

```
int i;
i=10;
k=i+10;
cout << i;
cin >> i;
```

```
student s1;
s1.num=10;
k=s1.num+10;
cout << s1.num;
cin >> s1.num;
```

★ 成员函数的参数传递规则仍为实参单向传值到形参

§8. 类和对象的特性

8.4. 类的成员函数(P.226 8.3)

8.4.1. 成员函数的实现

体内实现：class中给出成员函数的定义及实现过程

★ 体内实现缺省是inline，具体是否inline由系统决定

第4章的说明

★ 可执行程序的代码长度增加，但执行速度加快，适用于函数体短小而调用频繁的情况（1-5行）

★ 不能包含分支、循环等复杂的控制语句

★ 系统编译时自动判断是否需要采用内置方式

```
class student {
  ...
  public:
    void display()
    {
        cout<<"num:" <<num <<endl;
        cout<<"name:"<<name<<endl;
        cout<<"sex:" <<sex <<endl;
    }
};
```

体外实现：class中给出成员函数的定义，class外部(class后)给出成员函数的实现

★ 函数实现时需要加类的作用域限定符

★ 缺省不是inline，可用inline显式指定，具体是否inline仍由系统决定

```
class student {
  public:
    void display();
};

void student::display()
{
    cout << "num:"  <<num  <<endl;
    cout << "name:" <<name <<endl;
    cout << "sex:"  <<sex  <<endl;
}
```

§8. 类和对象的特性

8.4. 类的成员函数(P. 226 8.3)

8.4.2. 成员函数的性质

★ 对应类的成员函数(类函数)，一般的普通函数称为全局函数

★ 成员函数的定义、实现及调用时参数传递的语法规则与全局函数完全相同

★ 成员函数也受类的成员访问限定符的约束，只有公有的成员函数可以被外部调用

★ 私有和公有的成员函数均可以访问/调用本类的所有数据成员/成员函数，不受private/public的限制(private/public是用来限制外部对成员的访问)

```
class test {
  private:
    int a;
    int f1();
  public:
    int b;
    int f2();
    int f3();
};
```

```
int test::f1()
{
    a=10;   ✓
    b=15;   ✓
    f2();   ✓
}
int test::f2()
{
    ...
}
int test::f3()
{
    a=20;   ✓
    b=25;   ✓
    f1();   ✓
}
```

```
int main()
{
    test t1;

    t1.a=10;   ×
    t1.f1();   ×
    t1.b=15;   ✓
    t1.f2();   ✓
    t1.f3();   ✓
}
```

§8. 类和对象的特性

8.4. 类的成员函数(P. 226 8.3)

8.4.2. 成员函数的性质

★ 全局函数与成员函数可以同名，按照低层屏蔽高层的原则进行，也可以通过域运算符(::级别最高)强制访问高层

```
class test {
  ...
  public:
    int fun();
    int f1();
};

int fun() 全局函数
{
    ...
}
```

```
int test::fun() 类函数
{
    ...
}

int test::f1()
{
    fun();    类函数
    ::fun();  全局函数
}
```

成员函数内部

```
int main()
{
    test t1;

    t1.fun(); 类函数
    fun();    全局函数
}
```

全局函数中表示不会冲突

§8. 类和对象的特性

8.4. 类的成员函数(P. 226 8.3)

8.4.2. 成员函数的性质

★ 全局函数与成员函数可以同名，按照低层屏蔽高层的原则进行，也可以通过域运算符(::级别最高)强制访问高层

=> 类的数据成员与全局变量也遵循此强制访问规则

```
int a;          全局变量
void fun()
{
   int a;       局部变量
   a=10;        局部变量
   ::a=15       全局变量(第4章中遗留问题,当时说不行)
}               这种方式仅适合C++，纯C编译器不支持
```

```
int a; 全局变量
class test {
  ...
  public:
    int a; 类成员
    int f1();
  };
```

成员函数内部

```
int test::f1()
{
    a=10;     类数据成员
    ::a=10;   全局变量
}
```

```
int test::f1()
{
    int a; 成员函数的自动变量

    a=5;          自动变量
    test::a=10;   类数据成员
    ::a=10;       全局变量
}
```

```
int main()
{
    test t1;

    t1.f1();
    cout << t1.a << endl; //类数据成员
    cout << a << endl;    //全局变量
}
//外部无法访问f1自动变量
```

§8. 类和对象的特性

8.4. 类的成员函数(P.226 8.3)

8.4.3. 成员函数的存储方式

★ 每个类的实例对象仅包含数据成员(sizeof(类)=所有数据成员之和)，根据不同的定义位置占用不同的数据空间(静态数据区或动态数据区)

★ 类的成员函数占用函数(代码)区，每个类的每个成员函数(包括体内实现和体外实现)只占用一段空间，所有该类的对象共用成员函数的代码空间

★ 当通过对象调用成员函数时，系统会缺省设置一个隐含的this指针，指向被调用的对象，并以此来区分成员函数对数据成员的访问

```
class student {
  private:
    int num;
  public:
    void set(student *this, int n)
    {
        this->num = n;
    }
    void display(student *this)
    {
        cout << this->num << endl;
    }
};
```

s1.set(10) ⇔ s1.set(&s1, 10);
s2.display()⇔s2.display(&s2);

```
class student {
  private:
    int num;
  public:
    void set(int n)
    {
        num = n;
    }
    void display()
    {
        cout << num << endl;
    }
};
```

s1,s2占用不同的4字节
为什么 s1.set / s2.display 时会指向不同的4字节

```
int main()
{
    student s1, s2;
    s1.set(10);
    s2.set(15);
    s1.display(); 10
    s2.display(); 15

    return 0;
}
```

§8. 类和对象的特性

8.5. 类的封装性和信息隐蔽

8.5.1. 公有接口和私有实现的分离

★ 公有函数可被外界调用，称为类的公共/对外接口通过对象.函数(实参表)的方法进行调用，将函数称为方法，将调用过程称为消息传递

★ 如果允许外界直接改变某个数据成员的值，可直接设置属性为public(不提倡)

★ 其它不愿公开的数据成员和成员函数可设置为私有,对外部隐蔽，但仍可通过公有函数进行访问及修改

```
class student {
  private:
    int num;
  public:
    void set(int n)
    {
        num = n;
    }
    void display()
    {
        cout << num << endl;
    }
};
```

```
int main()
{
  student s1, s2;
   s1.set(10);
   s2.set(15);
   s1.display(); 10
   s2.display(); 15

   return 0;
}
```

set/display函数均间接访问了私有成员num

§8. 类和对象的特性

8.5. 类的封装性和信息隐蔽

8.5.1. 公有接口和私有实现的分离

★ 公有函数的形参称为提供给外部的访问接口，在形参的数量、类型、顺序不变的情况下，私有成员的变化及公有函数实现部分的修改不影响外部的调用

①
```
class student {
  private:
    int num;
  public:
    void set(int n)
    {
        num = n;
    }
    void display()
    {
        cout << num << endl;
    }
};
```

②
```
class student {
  private:
    int xh;
  public:
    void set(int n)
    {
        xh = n;
    }
    void display()
    {
        printf("%d\n", xh);
    }
};
```

③
```
class student {
  private:
    int xh;
  public:
    void set(int n)
    {
        xh = (n>=0 ? n:0);
    }
    void display()
    {
        printf("%d\n", xh);
    }
};
```

```
int main()
{
   student s1, s2;
   s1.set(10);
   s2.set(15);
   s1.display(); 10
   s2.display(); 15

   return 0;
}
```

假设class student由甲编写
main函数由乙编写
则：甲用三种方法
乙的程序均不需要变化

§8.类和对象的特性

8.5.类的封装性和信息隐蔽

8.5.1.公有接口和私有实现的分离

应用实例1：

谷歌公司的Android 4.x内核

```
class picture {
    类的私有数据成员
    及成员函数
    外界不可见
  void show(char *图片名)
    函数实现，不可见
};
```

***公司的游戏软件

```
int main()
{
    ....
    picture p1;
    p1.show(文件名);
    ....
}
```

谷歌公司的Android 7.x内核

```
class picture {
    类的私有数据成员
    及成员函数
    外界不可见
    可能已进行过很大调整
  void show(char *图片名)
    函数实现，不可见
    实现过程可能与2.3完全不同
};
```

谷歌称7.x的显示速度
经优化后比4.x快**%
用户程序不需要变化

§8.类和对象的特性

8.5.类的封装性和信息隐蔽

8.5.1.公有接口和私有实现的分离

应用实例2：

A公司的甲团队：V1.0版本

```
class translation {
    类的私有数据成员
    及成员函数
    外界不可见
  void trans(char *英文)
    函数功能为输出中文
    具体实现过程不可见
};
```

A公司的乙团队

```
int main()
{
    ....
    translation t1;
    t1.trans("****");
    ....
}
```

A公司的甲团队：V1.1版本

```
class translation {
    类的私有数据成员
    及成员函数
    外界不可见
    可能已进行过很大调整
  void trans(char *英文)
    函数实现，不可见
    实现过程可能与1.0完全不同
};
```

V1.1比V1.0的翻译结果
更准确，更贴切
用户程序不需要变化
两个团队能同时工作

§8. 类和对象的特性

8.5. 类的封装性和信息隐蔽

8.5.1. 公有接口和私有实现的分离

8.5.2. 类声明和成员函数定义的分离

★ 将类的声明(*.h)与类成员函数的实现(*.cpp)分开

假设程序由ex1.cpp、ex2.cpp和ex.h共同构成

```
/* ex.h */
class student {
  private:
    数据成员1;
    ...;
    数据成员n;
  public:
    成员函数1;
    ...;
    成员函数2;
}
```

```
/* ex2.cpp */
#include <iostream>
#include "ex.h"
using namespace std;
main及其它函数的实现
```

```
/* ex1.cpp */
#include <iostream>
#include "ex.h"
using name space std;

返回值 student::成员函数1()
{
    成员函数1的实现;
}

...

返回值 student::成员函数n()
{
    成员函数n的实现;
}
```

§8. 类和对象的特性

8.5. 类的封装性和信息隐蔽

8.5.1. 公有接口和私有实现的分离

8.5.2. 类声明和成员函数定义的分离

★ 将类的声明(*.h)与类成员函数的实现(*.cpp)分开

★ 在需要外部调用的地方，只要提供声明部分即可，类的实现可通过库文件(*.lib)或动态链接库(*.dll)的方式提供，而不必提供实现的源码

★一个程序包含多源程序文件的方法见第4章补充文档

★ 建立库文件/动态链接库的方法请自学(本课程不做要求)

§8.类和对象的特性

8.6.类和对象的简单应用

P.236 例8.1：注意3，因为是自动对象

P.247 例8.2 程序(b)中：

set_time/show_time都是全局函数

P.239 例8.2 程序(c)中：

set_time是带缺省参数的全局函数

P.240 例8.3：

set_time/show_time是成员函数，其中的hour、minute、sec都是数据成员,不带对象名,直接用

P.241 例8.4：

主函数是简单的顺序结构，分支和判断隐含在成员函数中