C++primer 第五版 第七章 答案

7.1 答：

```
#include <iostream>
#include <string>
using namespace std;
struct Sales_data
{
	string bookNo;
	unsigned units_sold = 0;
	double revenue = 0.0;
};

int main()
{
	double price=0;
	Sales_data total;
	Sales_data trans;
	if (cin >> total.bookNo >> price >>total.units_sold )
	{
		total.revenue = total.units_sold*price;
		while (cin >> trans.bookNo	>> price >>trans.units_sold )
		{
			trans.revenue = trans.units_sold*price;
			if (total.bookNo == trans.bookNo)
			{
				total.revenue += trans.revenue;
				total.units_sold += trans.units_sold;
			}
			else
			{
				cout << total.bookNo <<' '<< total.revenue <<' '<< total.units_sold;
				total.bookNo = trans.bookNo;
				total.revenue = trans.revenue;
				total.units_sold = trans.units_sold;
			}
		}
		cout << total.bookNo << ' ' << total.revenue << ' ' << total.units_sold;
	}
}
```

7.2 答：

```
Sales_data &combine(const Sales_data&rhs)
	{
		units_sold += rhs.units_sold;
```

```
            revenue += rhs.revenue;
            return *this;
        }
        string isbn() const { return bookNo; }
```

7.3 答：

```
#include <iostream>
#include <string>
using namespace std;
struct Sales_data
{
        Sales_data &combine(const Sales_data&rhs)
        {
            units_sold += rhs.units_sold;
            revenue += rhs.revenue;
            return *this;
        }
        string isbn() const { return bookNo; }
        string bookNo;
        unsigned units_sold = 0;
        double revenue = 0.0;
};

int main()
{
        double price = 0;
        Sales_data total;
        if (cin >> total.bookNo >> price >> total.units_sold)
        {
            Sales_data trans;
            total.revenue = total.units_sold*price;
            while (cin >> trans.bookNo >> price >> trans.units_sold)
            {
                trans.revenue = trans.units_sold*price;
                if (total.isbn() == trans.isbn())
                {
                    total.combine(trans);
                }
                else
                {
                    cout << total.bookNo << ' ' << total.revenue << ' ' << total.units_sold;
                    total.bookNo = trans.bookNo;
                    total.revenue = trans.revenue;
                    total.units_sold = trans.units_sold;
                }
            }
```

```
            cout << total.bookNo << ' ' << total.revenue << ' ' << total.units_sold << endl;
        }
}
```

7.4 答：class person

```
{
public:
        person(string pName, string pAddress)
        {
            name = pName;
            address = pAddress;
        }
private:
        string name;
        string address;
};
```

7.5 答：应该是 cosnt 成员函数，在这两个函数体内不应该修改成员数据的值。

```
string get_name( ) const
{
        return (*this).name;
}
string get_addes( ) const
{
        return this->address;
}
```

7.6 答：

```
//类的非成员函数接口
Sales_data add(const Sales_data &lhs, const Sales_data &rhs)
{
        Sales_data sum = lhs;
        sum.combine(rhs);
        return sum;
}

istream &read(istream &is, Sales_data &item)
{
        double price = 0;
        is >> item.bookNo >> item.units_sold >> price;
        item.revenue = price*item.units_sold;
        return is;
}

ostream &print(ostream &os, Sales_data &item)
{
        os << item.isbn() << ' ' << item.units_sold << ' '
```

```
            << item.revenue/*<<item.avg_price()*/;
        return os;

}
```
7.7 答：
```
#include <iostream>
#include <string>
using namespace std;
struct Sales_data
{
public:
        Sales_data &combine(const Sales_data&rhs)
        {
                units_sold += rhs.units_sold;
                revenue += rhs.revenue;
                return *this;
        }

        string isbn() const { return bookNo; }

        double    avg_price() const;


        string bookNo;
        unsigned units_sold = 0;
        double revenue = 0.0;
};
//类的非成员函数接口
Sales_data add(const Sales_data &lhs, const Sales_data &rhs)
{
        Sales_data sum = lhs;
        sum.combine(rhs);
        return sum;
}

istream &read(istream &is, Sales_data &item)
{
        double price = 0;
        is >> item.bookNo >> item.units_sold >> price;
        item.revenue = price*item.units_sold;
        return is;
}

ostream &print(ostream &os, Sales_data &item)
{
```

```
    os << item.isbn() << ' ' << item.units_sold << ' '
        << item.revenue/*<<item.avg_price()*/;
    return os;

}


int main()
{
    Sales_data item1, item2;
    read(cin,item1);
    read(cin,item2);
    print(cout,add(item1,item2));
    Sales_data total;
    if ( read(cin,total) )
    {
        Sales_data trans;
        while (read(cin, trans))
        {
            if (total.isbn() == trans.isbn())
            {
                total.combine(trans);
            }
            else
            {
                print(cout, total) << endl;
                total.bookNo = trans.bookNo;
                total.revenue = trans.revenue;
                total.units_sold = trans.units_sold;
            }
        }
        print(cout, total);
    }
}
```

7.8 答：因为写操作会该修改流的内容，所以接受 Sales_data 定义函数为普通引用，而输出操作不应该修改数据所以定义成 const 引用。

7.9 答：

```
#include <iostream>
#include <string>
using namespace std;
class person
{
public:
    person(string pName, string pAddress)
    {
```

```
        name = pName;
        address = pAddress;
    }
    istream &read(istream &is, person &data)
    {
        is >> data.name >> data.address;
        return is;
    }

    ostream &print(ostream &os, const person &data)
    {
        cout << data.name << ' ' << data.address;
        return os;
    }
private:
    string name    ;
    string address ;
};
```

7.10 答：连续输入两次数据。

7.11 答：

```
#include <iostream>
#include <string>
using namespace std;
struct Sales_data
{
public:
    //4 个构造函数：
    Sales_data() = default;
    Sales_data(const string&s) :bookNo(s){}
    Sales_data(const string&s, unsigned n, double p) :bookNo(s), units_sold(n), revenue(p*n){}
    Sales_data(istream &is)
    {
        read(is, *this);
    }

    Sales_data &combine(const Sales_data&rhs)
    {
        units_sold += rhs.units_sold;
        revenue += rhs.revenue;
        return *this;
    }

    string isbn() const { return bookNo; }

    double    avg_price() const;
```

```
    string bookNo;
    unsigned units_sold = 0;
    double revenue = 0.0;
};
//类的非成员函数接口
Sales_data add(const Sales_data &lhs, const Sales_data &rhs)
{
    Sales_data sum = lhs;
    sum.combine(rhs);
    return sum;
}

istream &read(istream &is, Sales_data &item)
{
    double price = 0;
    is >> item.bookNo >> item.units_sold >> price;
    item.revenue = price*item.units_sold;
    return is;
}

ostream &print(ostream &os, Sales_data &item)
{
    os << item.isbn() << ' ' << item.units_sold << ' '
        << item.revenue/*<<item.avg_price()*/;
    return os;

}
int main()
{
    Sales_data data1;
    Sales_data data2("chenxun");
    Sales_data data3("chenxun", 2, 3);
    Sales_data data4(cin);
}
```

7.12 答：如上题代码所示

7.13 答：

```
#include <iostream>
#include <string>
using namespace std;
struct Sales_data
{
    //这里第四个构造函数要用到类外 类的接口 read 函数，如果你不知道 friend 函数先不要管。
```

```
friend istream &read(std::istream&, Sales_data&);

public:
    //4 个构造函数
    //Sales_data() = default;
    //Sales_data(const string&s) :bookNo(s){}
    //Sales_data(const string&s, unsigned n, double p) :bookNo(s), units_sold(n), revenue(p*n){}
    Sales_data(istream &is)
    {
        read(is, *this);
    }

    Sales_data &combine(const Sales_data&rhs)
    {
        units_sold += rhs.units_sold;
        revenue += rhs.revenue;
        return *this;
    }

    string isbn() const { return bookNo; }

    double    avg_price() const;


    string bookNo;
    unsigned units_sold = 0;
    double revenue = 0.0;
};
//类的非成员函数接口
Sales_data add(const Sales_data &lhs, const Sales_data &rhs)
{
    Sales_data sum = lhs;
    sum.combine(rhs);
    return sum;
}

istream &read(istream &is, Sales_data &item)
{
    double price = 0;
    is >> item.bookNo >> item.units_sold >> price;
    item.revenue = price*item.units_sold;
    return is;
}
```

```
ostream &print(ostream &os, Sales_data &item)
{
	os << item.isbn() << ' ' << item.units_sold << ' '
		<< item.revenue/*<<item.avg_price()*/;
	return os;

}
int main()
{

	Sales_data total(cin);
	Sales_data trans(cin);
	if (read(cin, total))
	{
		while (read(cin, trans))
		{
			if (total.isbn() == trans.isbn())
			{
				total.combine(trans);
			}
			else
			{
				print(cout, total) << endl;
				total.bookNo = trans.bookNo;
				total.revenue = trans.revenue;
				total.units_sold = trans.units_sold;
			}
		}
		print(cout, total);
	}
}
```

7.14 答：`Sales_data() :bookNo(0), units_sold(0), revenue(0.0)`

7.15 答：如题 7.4 再添加一个 `person() = default;` 合成默认构造函数

7.16 答：定义在 public 说明符之后的成员可以被类的成员函数访问，public 成员定义类的接口。定义在 private 说明符之后的成员可以被类的成员函数访问，但是不能被使用该类的代码使用，private 部分封装（即隐藏了）类的实现细节。一个类可以包含 0 个或多个访问说明符，而且对于某个访问说明符能出现多少次也没严格的限制。每个访问说明符指定了接下来的成员的访问级别，其有效范围直到出现下一个访问说明符或者到达类的结尾为之。

7.17 答：唯一区别就是默认的访问权限，struct 默认公有，class 默认私有。

7.18 答：封装就是隐藏实现细节，对外只提供操作数据的函数接口，而不必关心这个操作具体是怎么实现的。好处：确保用户代码不会无意间破坏封装对象的状态，被封装的类的具体实现细节可以随时修改，而无须调整用户级别的代码。

7.19 答：数据成员 name 和 address 声明为 private，保证只能被类的成员所用，外界不可访问。函数 get_name()和 get_address()声明为 public，其目的是为外界提供接口，以便能访问类的数据成员。在类外，构造函数能对数据成员进行访问，所以构造函数也应声明为 public，这样才能初始化类的对象。

7.20 答：如果类的接口不是类的成员函数时，需要访问类的私有成员数据，那么就可以把这样的函数接口声明为友元。友元破坏了类的数据封装的特性。

7.21 答：class Sales_data

```
{
    //这里第四个构造函数要用到类外 类的接口 read 函数，如果你不知道 friend 函数先不要管。
    friend Sales_data add(const Sales_data &lhs, const Sales_data &rhs);
    friend istream &read(std::istream&, Sales_data&);
    friend ostream &print(ostream &os, Sales_data &item);
public:
    //4 个构造函数
    Sales_data() = default;
    Sales_data(const string&s) :bookNo(s){ }
    //Sales_data() :bookNo(0), units_sold(0), revenue(0.0)
    //这里显示调用类内初始值显示初始化成员数据,同第二种构造函数是等价的；
    Sales_data(const string&s, unsigned n, double p) :bookNo(s), units_sold(n), revenue(p*n){ }
    Sales_data(istream &is)
    {
        read(is, *this);
    }

    Sales_data &combine(const Sales_data&rhs)
    {
        units_sold += rhs.units_sold;
        revenue += rhs.revenue;
        return *this;
    }

    string isbn() const { return bookNo; }

    double    avg_price() const;

private:
    string bookNo;
    unsigned units_sold = 0;
    double revenue = 0.0;
};
```

7.22 答：

```
class person
{
```

```
public:
    person() = default;
    person(string pName, string pAddress)
    {
        name = pName;
        address = pAddress;
    }
    istream &read(istream &is, person &data)
    {
        is >> data.name >> data.address;
        return is;
    }

    ostream &print(ostream &os, const person &data)
    {
        cout << data.name << ' ' << data.address;
        return os;
    }
private:
    string name;
    string address;
};
```

7.23答：

```
class Screen{
public:
    typedef string::size_type pos;

    Screen() = default;
    Screen() : cursor(0), height(0), width(0) { }
    Screen(pos ht, pos wd, char c) :height(ht), width(wd),
contents(ht*wd, c){ }
    //类的成员函数;
    char get() const{ return contents[cursos]; }
    inline char get(pos ht, pos wd) const;//返回光标所指的字符
```

```
    Screen &move(pos r, pos c);


private:
    pos cursos;
    pos height , width ;
    string contents;//注意本class的成员string对象初始化的方式;
};
```

7.24答：
```
Screen() = default;
    Screen(pos ht = 0, pos wd = 0) : cursor(0),height(ht), width(wd),contents(ht * wd, ' ') { }
    Screen(pos ht, pos wd, char c) :height(ht), width(wd), contents(ht*wd, c){ }
```

7.25 答：不能完全依赖，含有内置类型或符合类型成员的类应该在类的内部初始化这些数据，或者定义一个自己的默认构造函数。否则用于在创建对象时就可能得到为定义的值。

7.26 答：加上 inline 即可。

7.27 答：
```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
class Screen{
public:
    typedef string::size_type pos;

    Screen() : cursor(0), height(0), width(0){ }
    Screen(pos ht = 0, pos wd = 0) : cursor(0),height(ht), width(wd),contents(ht * wd, ' ') { }
    Screen(pos ht, pos wd, char c) : cursor(0),height(ht), width(wd),contents(ht * wd, c) { }

    friend class Window_mgr;
    //类的成员函数；
    char get() const{ return contents[cursor]; }
    inline char get(pos ht, pos wd) const;//返回光标所指的字符
    Screen &clear(char = bkground);
```

```
private:
    static const char bkground = ' ';
public:
    Screen &move(pos r, pos c);
    Screen &set(char);
    Screen &set(pos, pos, char);
    Screen &display(std::ostream &os)
    {
        do_display(os);
        return *this;
    }
    const Screen &display(std::ostream &os) const
    {
        do_display(os);
        return *this;
    }


private:
    pos cursor;
    pos height , width ;
    string contents;//注意本 class 的成员 string 对象初始化的方式；
    void do_display(std::ostream &os) const { os << contents; }
};

Screen &Screen::clear(char c)
{
    contents = std::string(height*width, c);
    return *this;
}

inline Screen &Screen::move(pos r, pos c)
{
    pos row = r * width;
    cursor = row + c;
    return *this;
}

char Screen::get(pos r, pos c) const
{
    pos row = r*width;
    return contents[row + c];
}
```

```
inline Screen &Screen::set(char c)
{
	contents[cursor] = c;
	return *this;
}

inline Screen &Screen::set(pos r, pos col, char ch)
{
	contents[r*width + col] = ch;
	return *this;
}

int main()
{
	Screen myScreen(5, 5, 'x');
	myScreen.display(cout); cout << endl;
	myScreen.move(4, 0).set('#').display(cout);
	cout << endl;
	myScreen.display(cout);
	cout << endl;
}
```

7.28 答：放回的都是*this 的副本，对其进行操作只是改变其副本，而不能真正作于对象本身。

7.29 答：利用，7.27 的代码进行测试就可以。

7.30 答：实际上 this 代表是对象的首地址，显示的调用对于代码消耗维护方面的代价。

7.31 答：class Y;

```
class X
{
	Y *p;
};
class Y
{
	X xobject;
};
```

7.32 答：

```
class Screen{
public:
	typedef string::size_type pos;
	Screen() : cursor(0), height(0), width(0){ }
	Screen(pos ht, pos wd,const char &c) : cursor(0),height(ht), width(wd),contents(ht * wd, c)
{ }
	Screen(pos ht = 0, pos wd = 0) : cursor(0),height(ht), width(wd),contents(ht * wd, ' ') { }
	friend class Window_mgr;
	//friend    void Window_mgr::clear(ScreenIndex);
```

```
    //类的成员函数；
    char get() const{ return contents[cursor]; }
    inline char get(pos ht, pos wd) const;//返回光标所指的字符
    Screen &clear(char = bkground);
private:
    static const char bkground = ' ';
public:
    Screen &move(pos r, pos c);
    Screen &set(char);
    Screen &set(pos, pos, char);
    Screen &display(std::ostream &os)
    {
        do_display(os);
        return *this;
    }
    const Screen &display(std::ostream &os) const
    {
        do_display(os);
        return *this;
    }


private:
    pos cursor;
    pos height , width ;
    string contents;//注意本 class 的成员 string 对象初始化的方式；
    void do_display(std::ostream &os) const { os << contents; }
};

Window_mgr:
class Window_mgr
{
public:
    using ScreenIndex = std::vector<Screen>::size_type;
    void clear(ScreenIndex);

private:
    vector<Screen> screens{ Screen(24,80,' ') };
};
```

7.33 答：pos 的作用域，指明 pos 是属于哪个类的；

7.34 答：编译出错，成员使用了未定的 pos。

7.35 答：非常混乱的代码。造成了 setVal 非法的重载，类定义之前所使用的 type 是 string，类内 setVal 和 initVal 使用的 double，类后面哪个 setVal 返回类型 type 使用的 string，而参数类型所使用的 type 是 double，函数定义体重 initVal 是类中定义的，返回类型是 string，所以

造成非法重载。
修改：去掉类中 type 定义和 setVal 定义，int val 的改成用 type 就可以了。
7.36 答：先初始化了 rem，再初始化 base。
7.37 答：（1）使用默认构造函数或者使用提供一个实参为空 string 的构造函数
（2）使用提供一个参数的构造函数；
7.38 答：salse_data（std：：istream &is）{ read（is，*this）；}
7.39 答：不合法，无法知道到底使用哪一个构造函数。
7.40 答：b:年、月、日
7.41 答：
7.42 答：
7.43 答：class NoDefault

```
{
public:
    NoDefault(int i);
};

class c
{
public:
    c() :i(0), Ndf(i){ }
private:
    int i;
    NoDefault Ndf;
};
```

7.44 答：不对，因为 NoDefault 没有默认构造函数；
7.45 答：合法；
7.46 答：（a）不对，不提供构造函数的时候，编译器会给类合成一个；
（b）错误，可以带有实参；
（c）错误，有的类没提供默认构造函数时候，编译器要隐式的使用默认构造函数去创建对象的时候就无法通过编译了。请看书本默认构造函数的作用。
（d） 当自己定义构造函数的时候，会阻止编译器合成默认构造函数，所以在定义了构造函数的时候最好定义一下默认构造函数。
7.47 答：应该可以声明为 explicit，这样可以防止编译器进行隐式的类型转换即把一个 string 对象转换成 sales_data 对象，避免造成不必要的语义错误。加上 explicit 后，要创建 sales_data 对象时候我们必须显示的去创建，不要依赖编译器去帮你隐式的转换。
7.48答：
（1）.调用接收一个C风格字符串形参的string构造函数，创建一个临时的string对象，然后调用string类的复制构造函数，将null_isbn初始化为该该临时对象的副本。
（2）.使用string的对象null_isbn为实参，调用Sales_item类的构造函数创建Sales_item对象null1.
（3）.使用接受一个 C 风格字符串形参的 string 类的构造函数，生成一个临时 string 对象，然后用这个临时对象作为实参，调用 Sales_item 类的构造函数来创建 Sales_item 类的对象 null.
如果 sales_data 的构造函数时 explicit，那么第三个将是未定义的行为。编译通不过。

7.49 答：（a）可以进行隐式的类型的转换（b）可以进行隐式类型转换（c）可以

7.50 答：

```
class person
{
public:
    person() = default;
    person(string pName, string pAddress)
    {
        name = pName;
        address = pAddress;
    }
    istream &read(istream &is, person &data)
    {
        is >> data.name >> data.address;
        return is;
    }

    ostream &print(ostream &os, const person &data)
    {
        cout << data.name << ' ' << data.address;
        return os;
    }
private:
    string name;
    string address;
};
```

没有可以声明为 explicit 的构造函数。

7.51 答：假如在一个函数调用一个 vector<int>对象的时候，我们只给函数传递一个实参 42 的时候，不知道编译发生何种情况，这个实参倒是被隐式的转成一个 vector 对象还是一个整形变量 42.所以我们有必须把 vector 函数一个参数的构造函数声明为 explicit。而 string 则没有必要声明一个参数的构造函数为 explicit，这样就可以把 c 风格的字符串转换成 string 对象。

7.52 答：sales_data item（“978-0590353403”，25,15.99）；或者把类的数据成员都改成 public 且必须按照成员数据的顺序才能用列表初始化的方法。

7.53 答：

```
class Debug {
public:
    constexpr Debug(bool b = true): hw(b), io(b), other(b) { }
    constexpr Debug(bool h, bool i, bool o):
                                            hw(h), io(i), other(o) { }
    constexpr bool any() { return hw || io || other; }
    constexpr bool hardware() { return hw || io; }
    constexpr bool app() { return other; }
```

```
        void set_io(bool b) { io = b; }
        void set_hw(bool b) { hw = b; }
        void set_other(bool b) { hw = b; }
private:
        bool hw;
        bool io;
        bool other;
};
```

7.54 答：不能，因为 constexper 要求函数拥有唯一可执行语句就是返回语句。

7.55 答：不是，它的成员数据不是字面值常量。

7.56 答：在类中，如果成员前有 static 关键字，那么表明这个成员是静态成员，静态成员不属于某个对象所有，也不是对象的组成部分，他属于整个类。

优点：（1），static 成员的名字是在类的作用域中，因此可以避免与其他类的成员或全局对象名字冲突；（2）可以实施封装，static 成员可以是私有成员，而全局对象不可以。（3）通过阅读程序看出 static 成员是与特定的类关联的，这种可见性可清晰地显示程序员的意图。

不同点：普通成员是与对象相关联的，是某个对象的组成部分，而 static 成员与类相关联，由该类的全体对象所共享，不是任意对象的组成部分。

7.57 答：

```
class Account {
public:
        Account(): amount(0.0) { }
        Account(const std::string &s, double amt):
                owner(s), amount(amt) { }

        void calculate() { amount += amount * interestRate; }
        double balance() { return amount; }
public:
        static double rate() { return interestRate; }
        static void rate(double);
private:
        std::string owner;
        double amount;
        static double interestRate;
        static double initRate() { return .0225; }
        static const std::string accountType;
        static const int period = 30;
        double daily_tbl[period];
};
```

7.58 答：

.h 文件中 rate 和 vec 错误

.c 文件中 rate 和 vec 错误

更正为：

```
class Example    {
public:
```

```
static double rate;
static const int vecSize = 20;
static vector<double> vec;
};
// example C
#include “example.h”
double Example::rate=6.5;
vector<double> Example::vec(vecSize);
```