7.1 定义抽象数据类型

7.1.1 设计sales_data类 ¶

- isbn：成员函数，返回ISBN编号
- combine：成员函数，将两个sales_data对象相加
- add：函数，sales_data加法
- read：函数，从istream计入到sales_data中
- print：函数，将sales_data对象值输出到ostream 基于以上的成员函数，一个使用他们的例子：

```
sales_data total;              // 保存当前求和结果的变量
if(read(cin,total)){           // 计入第一条交易
    sales_data trans;          // 保存下一条交易数据的变量
    while(read(cin,trans){       // 读入剩余的交易
        if(trans.isbn == total.isbn){
            total.combine(trans); // 若书的编号一致，更新total的值
        } else {
            print(cout,total)<<endl; // 输出结果
            total = trans; // 处理下一本书
        }
    }

   print(cout,total)<<endl; // 输出最后一笔交易

} else { // 没有任何输入信息，通知用户
    cerr<<"No data?!"<<endl;
}
```

7.1.2 定义改进的sales_data类

```
struct sales_data{
    // 成员函数：
    // 定义在类内部的函数是隐式的inline函数
    std::string isbn()const{return bookNo}
    sales_data& combine(const sales_data &);
    double avg_price()const;
    // 成员变量
    std::string bookNo;
    unsigned units_sold = 0;
    double revenue = 0.0;

    // 所有成员必须在类内声明，函数体的实现可以在类外
};

// sales_data的非成员接口函数
sales_data add(const sales_data&, const sales_data &);
std::ostream &print(std::ostream&,const sales_data &);
std::istream &read(std::istream&, sales_data &);

// 关于this常量指针（顶层const，不允许修改地址），sales_data *const
/*
    * 关于const成员函数，如std::string isbn()const{return bookNo}中,使用const的成员函数称为常量成员函数
    * const用来赋予指针this底层特性，不可以通过this修改成员属性
*/

// 类作用域和成员函数：成员函数体可随意使用类中的其他成员，无须在意次序（编译器先编译声明，再编译函数体）



// 在类的外部定义函数体：
double sales_data::avg_price()const{
    if(units_sold) return revenue/units_sold;
    else return 0;
}


// 定义一个返回this对象的函数：
// combine函数模型“+=”，调用该函数的对象代表左侧运算对象，右侧对象作为实参传入，“+=”把左侧运算对象当成左侧返回，combine函数返回引用类型
sales_data &sales_data::combine(const sales_data &rhs ){
    units_sold += rhs.units_sold;
    revenue += ths.revenue;
    return *this;
}
```

7.1.3 定义类相关的非成员函数：

如果非成员函数是类接口的组成部分，则函数声明应该和类在同一个头文件中

- 定义read和print函数

```
// 关于io类，istream,ostream，不能被拷贝，且读写操作会改变输入输出流的内容，所以返回的是普通引用istream&,ostream&


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, const sales_data &item ){
    os << item.isbn() << " " << item.units_sold<< " "
        << item.revenue << " " << item.avg_price();
    return os;
} // print函数不换行，输出内容的函数应该尽量减少对格式的控制，由用户代码块决定
```

- 定义add函数：接受两个sales_data对象作为参数，返回一个新的sales_data对象

```
sales_data add(const sales_data &item1,const sales_data &item2){
    sales_data item3 = item1;
    item3.combine(item2);
    return item3;
```

7.1.4 构造函数： 类名 （）{}

- 默认构造函数（程序自动生成）：如果没有类内初始值，则自动默认初始化
- 如果自己已经定义了一个构造函数，但还想让程序自动生成一个默认构造函数，需要手动声明 `sales_data()=default;`
- 注意：若类内有内置类型或复合类型，则只有当这些成员全部被赋予了类内的初始值时，这个类才适合使用默认构造函数。 这是因为快中的内置类型或复合类型（数组和指针），默认初始化的值将是未定义的
- 构造函数初始化列表：

7.3 类的其他特性

7.3.1 类成员再探

- 定义一个类型成员

```
// 注意：定义类型的成员必须先定义再使用（与普通成员有所区别）
class Screen{
public:
    typedef std::string::size_type pos; // 等价于：using pos = std::string::size_type;
prviate:
    pos cursor = 0;
    pos height = 0,width = 0;
    std::string contents;
}
```

- Screen类的成员函数：

```
class Screen{
public:
    typedef std::string::size_type pos;
    Screen() = default; // 想要默认构造，这一步是必须的
    Screen(pos ht,pos wd,char c):height(ht),width(wd),contents(ht * wd , c){} // 有参构造,cursor被类内初始值初始化为0
    char get() const{return contents[cursor];} // 隐式内联
    inline char get(pos ht,pos wd)const; // 显式内联，可以在类外定义函数体
    Screen &move(pos r,pos c); // 定义函数体时声明是内联函数
private:
    pos cursor = 0;
    pos height = 0, width = 0;
    std::string contents;
}

// 在类外定义move函数体时声明是内联函数：
inline Screen &Screen::move(pos r,pos c){ // 移动到r行c列
    pos row = r*width;
    cursor = row +c ;
    return *this;
}
char Screen::get(pos r,pos c){ const
    pos row = r*width; // 计算行的位置（一维数组表现二维数组）
    return contents[row + c];
}
```

- 重载成员函数

```
Screen myscreen;
char ch = myscreen.get(); // 调用Screen::get();
char ch = myscreen.get(0,0); // 调用Screen::get(pos r, pos c);
```

- 可变数据成员：

```
// 利用mutable关键字，const函数可以改变一个可变成员的值
mutable size_t access_ctr;
```

- 类数据成员的初始值：

```
    // 类内初始值：
```

In [ ]:

```

```