Advanced Programming Concepts with C++

CSI 2372

Tutorial # 8

Selected exercises from chapters 14

uOttawa

Ahmedou Jreivine

Exercise 14.2, :

- **Write declarations for the overloaded input, output, addition, and compound-assignment operators for `Sales_data`.**

Answer:

```
#pragma once
#include <iostream>
#include <string>
// added overloaded input, output, addition, and compound-assignment operators
class Sales_data {
    friend std::istream& operator>>(std::istream&, Sales_data&);      // input
    friend std::ostream& operator<<(std::ostream&, const Sales_data&); // output
    friend Sales_data operator+(const Sales_data&, const Sales_data&); // addition
public:
    Sales_data(const std::string& s, unsigned n, double p): bookNo(s), units_sold(n), revenue(n * p){}
    Sales_data() : Sales_data("", 0, 0.0f) {}
    Sales_data(const std::string& s) : Sales_data(s, 0, 0.0f) {}
    Sales_data(std::istream& is);
    Sales_data& operator+=(const Sales_data&); // compound-assignment
    std::string isbn() const { return bookNo; }
private:
    inline double avg_price() const;
    std::string bookNo;
    unsigned units_sold = 0;
    double revenue = 0.0;
};
```

Answer 14.2:

```
std::istream& operator>>(std::istream&, Sales_data&);

std::ostream& operator<<(std::ostream&, const Sales_data&);

Sales_data operator+(const Sales_data&, const Sales_data&);

inline double Sales_data::avg_price() const {
    return units_sold ? revenue / units_sold : 0;
}
```

```
#include "SalesData.h"
Sales_data::Sales_data(std::istream& is) : Sales_data(){
      is >> *this;
}
Sales_data& Sales_data::operator+=(const Sales_data& rhs){
      units_sold += rhs.units_sold;
      revenue += rhs.revenue;
      return *this;
}
std::istream& operator>>(std::istream& is, Sales_data& item){
      double price = 0.0;
      is >> item.bookNo >> item.units_sold >> price;
      if (is)
            item.revenue = price * item.units_sold;
      else
            item = Sales_data();
      return is;
}
std::ostream& operator<<(std::ostream& os, const Sales_data& item){
      os << item.isbn() << " " << item.units_sold << " " << item.revenue << " " << item.avg_price();
      return os;
}
Sales_data operator+(const Sales_data& lhs, const Sales_data& rhs){
            Sales_data sum = lhs;
            sum += rhs;
            return sum;
}
```

Test:

```
#include "SalesData.h"
int main()
{
      Sales_data cp5;
      std::cin >> cp5;
      std::cout << cp5 << std::endl;
      return 0;
}
```

Exercise 14.3:

- Both string and vector define an overloaded == that can be used to compare objects of those types. Assuming svec1 and svec2 are vectors that hold strings, identify which version of == is applied in each of the following expressions:
  - **(a)** "cobble" == "stone"
  - **(b)** svec1[0] == svec2[0]
  - **(c)** svec1 == svec2

Answer:

- (a) "cobble" == "stone" built-in operator==
- (b) svec1[0] == svec2[0] overloaded operator== in string
- (c) svec1 == svec2 overloaded operator== in vector
- (d) "svec1[0] == "stone" overloaded operator== in string

Exercise 14.4:

- Explain how to decide whether the following should be class members:
  - **(a)** %
  - **(b)** %=
  - **(c)** ++
  - **(d)** ->
  - **(e)** <<
  - **(f)** &&
  - **(g)** ==
  - **(h)** ()

(a) % symmetric operator. Hence, non-member
(b) %= changing state of objects. Hence, member
(c) ++ changing state of objects. Hence, member
(d) -> = () [] must be member
(e) << non-member
(f) && symmetric , non-member
(g) == symmetric , non-member
(h) = () [] must be member

Exercise 14.6 and 14.9+14.10 and 14.20 and 14.45 and 14.46:

Define an output operator for your Sales_data class.

```
#define USE_OPERATOR
/* Sales_data.h */
class Sales_data;
std::istream &read(std::istream &, Sales_data &);
class Sales_data {
    friend Sales_data add(const Sales_data &, const Sales_data &);
#ifndef USE_OPERATOR
    friend std::istream &read(std::istream &, Sales_data &);
    friend std::ostream &print(std::ostream &, const Sales_data &);
#else
    friend Sales_data operator+(const Sales_data &, const Sales_data &);
    friend std::istream &operator>>(std::istream &, Sales_data &);
    friend std::ostream &operator<<(std::ostream &, const Sales_data &);
#endif
public:
    Sales_data() : Sales_data("", 0, 0.0) {}
    explicit Sales_data(const std::string &no) : Sales_data(no, 0, 0.0) {}
    Sales_data(const std::string &no, unsigned us, double price) : bookNo(no), units_sold(us),
    revenue(price * us) {}
    explicit Sales_data(std::istream &is) : Sales_data() { read(is, *this); }
    Sales_data &combine(const Sales_data &);
```

```
    #ifndef USE_OPERATOR
    #else
        Sales_data &operator+=(const Sales_data &);
    #endif
    std::string isbn() const { return bookNo; }
private:
    double avg_price() const;
    std::string bookNo;
    unsigned units_sold = 0;
    double revenue = 0.0;
};
#ifdef USE_OPERATOR
    Sales_data operator+(const Sales_data &, const Sales_data &);
    std::istream &operator>>(std::istream &, Sales_data &);
    std::ostream &operator<<(std::ostream &, const Sales_data &);
#endif
    inline
    double Sales_data::avg_price() const {
        return units_sold ? revenue / units_sold : 0;
    }
```

```
/* Sales_data.cpp */
bool compareIsbn(const Sales_data &sd1, const Sales_data &sd2) {
    return sd1.isbn() < sd2.isbn();
}
Sales_data &Sales_data::combine(const Sales_data &rhs) {
    units_sold += rhs.units_sold;
    revenue += rhs.revenue;
    return *this;
}
Sales_data add(const Sales_data &lhs, const Sales_data &rhs) {
    Sales_data sum = lhs;  // Use default copy constructor
    sum.combine(rhs);
    return sum;
}
```

```
#ifndef USE_OPERATOR
    std::istream &read(std::istream &is, Sales_data &item) {
        double price;
        is >> item.bookNo >> item.units_sold >> price;
        item.revenue = item.units_sold * price;
        return is;
    }
    std::ostream &print(std::ostream &os, const Sales_data &item) {
    os << item.isbn() << " " << item.units_sold << " "<< item.revenue << " " << item.avg_price();
    return os;
}
#else
    std::istream &operator>>(std::istream &is, Sales_data &item) {
        double price;
        is >> item.bookNo >> item.units_sold >> price;
        if (is)
            item.revenue = item.units_sold * price;
        else
            item = Sales_data();
        return is;
    }
    std::ostream &operator<<(std::ostream &os, const Sales_data &item) {
        os << item.isbn() << " " << item.units_sold << " " << item.revenue << " " << item.avg_price();
        return os;
    }
#endif
```

```
/* Test.cpp */
int main(int argc, char **argv) {
        if (argc < 2) {
                std::cerr << "Usage: 14.6 <input filename>" << std::endl;
                return -1;
        }
        std::ifstream in(argv[1]);
        if (!in) {
                std::cerr << "Fail to open file: " << argv[1] << std::endl;
                return -2;
        }
        std::vector<Sales_data> vsd;
        #ifndef USE_OPERATOR
                for (Sales_data sd; read(in, sd); vsd.push_back(sd)) {}
        #else
               for (Sales_data sd; in >> sd; vsd.push_back(sd)) {}
        #endif
               std::cout << "Before sort:\n";
               for (const auto &sd : vsd)
                       #ifndef USE_OPERATOR
                               print(std::cout, sd) << std::endl;
                       #else
                               std::cout << sd << std::endl;
                       #endif
        std::sort(vsd.begin(), vsd.end(), compareIsbn);
        std::cout << "\nAfter sort:\n";
        for (const auto &sd : vsd)
               #ifndef USE_OPERATOR
                       print(std::cout, sd) << std::endl;
               #else
                       std::cout << sd << std::endl;
               #endif
        return 0;
}
```

Exercise 14.10

Describe the behavior of the Sales_data input operator if given the following input :

a) 0 - 201 - 99999 - 9 10 24.95
b) 10 24.95 0 - 210 - 99999 - 9

Answer:

a) correct format.
b) illegal input.But .95 will be converted to a float stored in this object.As a result, the data inside will be a wrong one.
   - Output: 10 24 22.8 0.95

Exercise 14.5, 14.8, 14.12, 14.15, 14.17, 14.9, 14.48 and 14.49:

In exercise 7.40 from § 7.5.1 (p. 291) you wrote a sketch of one of the following classes. Decide what, if any, overloaded operators your class should provide.

(a) Book
(b) Date
(c) Employee
(d) Vehicle
(e) Object
(f) Tree

Ex14_5.h (declarations)

```
#pragma once
#include <iostream>
#include <string>
class Book {
    friend std::istream& operator>>(std::istream&, Book&);
    friend std::ostream& operator<<(std::ostream&, const Book&);
    friend bool operator==(const Book&, const Book&);
    friend bool operator!=(const Book&, const Book&);
public:
    Book() = default;
    Book(unsigned no, std::string name, std::string author, std::string pubdate) :no_(no),
    name_(name), author_(author), pubdate_(pubdate) { }
    Book(std::istream &in) { in >> *this; }
private:
    unsigned no_;
    std::string name_;
    std::string author_;
    std::string pubdate_;
};
```

```
std::istream& operator>>(std::istream&, Book&);
std::ostream& operator<<(std::ostream&, const Book&);
bool operator==(const Book&, const Book&);
bool operator!=(const Book&, const Book&);
```

Ex14_5.cpp (definitions)

```
#include "exercise14_5.h"
std::istream& operator>>(std::istream &in, Book &book){
    in >> book.no_ >> book.name_ >> book.author_ >> book.pubdate_;
    return in;
}
std::ostream& operator<<(std::ostream &out, const Book &book){
    out << book.no_ << " " << book.name_ << " " << book.author_ << " " <<
    book.pubdate_;
    return out;
}
bool operator==(const Book &lhs, const Book &rhs){
    return lhs.no_ == rhs.no_;
}
bool operator!=(const Book &lhs, const Book &rhs){
    return !(lhs == rhs);
}
```

main.cpp (executions)

```
#include "Exercise14_5.h"
int main()
{
    Book book1(123, "CP5", "Lippman", "2012");
    Book book2(123, "CP5", "Lippman", "2012");

    if (book1 == book2)
    std::cout << book1 << std::endl;

    system("pause");
    return 0;
}
```

Exercise 14.11:

What, if anything, is wrong with the following Sales_data input operator? What would happen if we gave this operator the data in the previous exercise?

```
istream& operator>>(istream& in, Sales_data& s){
    double price;
    in >> s.bookNo >> s.units_sold >> price;
    s.revenue = s.units_sold * price;
    return in;
}
```

Answer:

- This input operator does not maintain the proper state of the destination object if errors occur during input.
- If we gave this operator the data in previous exercise, the first group would be read correctly, while the second wouldn't and the state of the object would be inconsistent.

Exercise 14.13:

Which other arithmetic operators (Table 4.1 (p. 139)), if any, do you think Sales_data ought to support? Define any you think the class should include.

Exercixe14_13.h

Asnwer

```
#include <string>
#include <iostream>
class Sales_data {
	friend std::istream& operator>>(std::istream&, Sales_data&); // input
	friend std::ostream& operator<<(std::ostream&, const Sales_data&); // output
	friend Sales_data operator+(const Sales_data&, const Sales_data&); // addition
	friend Sales_data operator-(const Sales_data&, const Sales_data&); // substraction
public:
	Sales_data(const std::string &s, unsigned n, double p) :bookNo(s), units_sold(n), revenue(n*p) { }
	Sales_data() : Sales_data("", 0, 0.0f) { }
	Sales_data(const std::string &s) : Sales_data(s, 0, 0.0f) { }
	Sales_data(std::istream &is);
	Sales_data& operator+=(const Sales_data&); // compound-assignment
	Sales_data& operator-=(const Sales_data&); // compound-substraction
	std::string isbn() const { return bookNo; }
private:
	inline double avg_price() const;
	std::string bookNo;
	unsigned units_sold = 0;
	double revenue = 0.0;
};
std::istream& operator>>(std::istream&, Sales_data&);
std::ostream& operator<<(std::ostream&, const Sales_data&);
Sales_data operator+(const Sales_data&, const Sales_data&);
Sales_data operator-(const Sales_data&, const Sales_data&);
inline double Sales_data::avg_price() const{
	return units_sold ? revenue / units_sold : 0;
}
```

Exercixe14_13.cpp

Answer

```
#include "Exercise14_13.h"
Sales_data::Sales_data(std::istream &is) : Sales_data(){
        is >> *this;
}
Sales_data& Sales_data::operator+=(const Sales_data &rhs){
        units_sold += rhs.units_sold;
        revenue += rhs.revenue;
        return *this;
}
Sales_data& Sales_data::operator-=(const Sales_data &rhs){
        units_sold -= rhs.units_sold;
        revenue -= rhs.revenue;
        return *this;
}
std::istream& operator>>(std::istream &is, Sales_data &item){
        double price = 0.0;
        is >> item.bookNo >> item.units_sold >> price;
        if (is)
                item.revenue = price * item.units_sold;
        else
                item = Sales_data();
        return is;
}
std::ostream& operator<<(std::ostream &os, const Sales_data &item){
os << item.isbn() << " " << item.units_sold << " " << item.revenue << " " << item.avg_price();
return os;
}
Sales_data operator+(const Sales_data &lhs, const Sales_data &rhs){
        Sales_data sum = lhs;
        sum += rhs;
        return sum;
}
        Sales_data operator-(const Sales_data &lhs, const Sales_data &rhs){
                Sales_data sum = lhs;
                sum -= rhs;
                return sum;
        }
```

Answer

Test:

```
#include "Exercise14_13.h"
int main()
{
    Sales_data s1("book1", 150, 10);
    Sales_data s2("book1", 200, 20);
    std::cout << s1 << std::endl;
    // Assignment
    s1 = s1 + s2;
    std::cout << s1 << std::endl;
    // Compound assignment
    s1 += s2;
    std::cout << s1 << std::endl;
    // Compound substraction
    s1 -= s2;
    std::cout << s1 << std::endl;
    // Substraction
    s1 = s1 - s2;
    std::cout << s1 << std::endl;
    system("pause");
    return 0;
}
```

Exercise 14.14:

Why do you think it is more efficient to define operator+ to call operator+= rather than the other way around?

Answer:

Suppose we have a class A defines as :

```
class A {
    friend A operator+(const A &, const A &);
public:
    A & operator+=(cosnt A &);
private:
    int i;
}
```

If we define operator+= to call operator+ as:

```
A & A::operator+=(const A &rhs) {
    *this = *this + rhs;
    return *this;
}
A operator+(const A &lhs, const A &rhs) {
    A sum = lhs;
    sum.i += rhs.i;
    return sum;
}
```

Answer:

Then, operator+= would create an unnecessary tempoary object when calling operator+ and copy / move this temporary object to this.

If we define operator+ to call operator+= as:

```
A & A::operator+=(const A &rhs) {
    i += rhs.i;
    return *this;
}
A operator+(const A &lhs, const A &rhs) {
    A sum = lhs;
    sum += rhs;
    return sum;
}
```

Then, there will be no temporary object when calling operator+=, since it will call operator+= on all data members, which do not create any temporary object.

Thus, it is more efficient to define operator+ to call operator+=.

Exercise 14.(27,28,30):

- Add increment and decrement operators to your StrBlobPtr class.

Answer:

StrBlob.h

```
class StrBlobPtr;
class ConstStrBlobPtr;
#include <vector>
#include <string>
#include <initializer_list>
#include <memory>
#include <iostream>
class StrBlob {
    friend class StrBlobPtr;
    friend class ConstStrBlobPtr;
    friend bool operator==(const StrBlob &, const StrBlob &);
    friend bool operator!=(const StrBlob &, const StrBlob &);
    friend bool operator<(const StrBlob &, const StrBlob &);
    friend bool operator>(const StrBlob &, const StrBlob &);
    friend bool operator<=(const StrBlob &, const StrBlob &);
    friend bool operator>=(const StrBlob &, const StrBlob &);
public:
    typedef std::vector<std::string>::size_type size_type;
    StrBlob();
    StrBlob(std::initializer_list<std::string> il);
    StrBlob(const StrBlob &);
    StrBlob &operator=(const StrBlob &);
```

Exercise 14.(27,28,30):

StrBlob.h

```
        std::string &operator[](size_type n) { return (*data)[n]; }
        const std::string &operator[](size_type n) const { return (*data)[n]; }
        size_type size() const { return data->size(); }
        bool empty() const { return data->empty(); }
        void push_back(const std::string &s);
        void push_back(std::string &&s);
        void pop_back();
        std::string &front();
        const std::string &front() const;
        std::string &back();
        const std::string &back() const;
        StrBlobPtr begin();
        StrBlobPtr end();
        ConstStrBlobPtr cbegin() const;
        ConstStrBlobPtr cend() const;
private:
        std::shared_ptr<std::vector<std::string>> data;
        void check(size_type pos, const std::string &msg) const;
};
bool operator==(const StrBlob &, const StrBlob &);
bool operator!=(const StrBlob &, const StrBlob &);
bool operator<(const StrBlob &, const StrBlob &);
bool operator>(const StrBlob &, const StrBlob &);
bool operator<=(const StrBlob &, const StrBlob &);
bool operator>=(const StrBlob &, const StrBlob &);
inline void StrBlob::push_back(const std::string &s) {
        data->push_back(s);
}
inline void StrBlob::push_back(std::string &&s) {
        std::cout << "StrBlob::push_back(std::string &&s)" << std::endl;
        data->push_back(std::move(s));
}
```

Exercise 14.(27,28,30):

StrBlob.cpp

```
#include "StrBlob.h"
#include "StrBlobPtr.h"
#include "ConstStrBlobPtr.h"
StrBlob::StrBlob() : data(std::make_shared<std::vector<std::string>>()) {}
StrBlob::StrBlob(std::initializer_list<std::string> il): data(std::make_shared<std::vector<std::string>>(il))
{}
StrBlob::StrBlob(const StrBlob &sb): data(std::make_shared<std::vector<std::string>>(*sb.data)) {}
StrBlob &StrBlob::operator=(const StrBlob &sb) {
    data = std::make_shared<std::vector<std::string>>(*sb.data);
    return *this;
}
void StrBlob::check(size_type pos, const std::string &msg) const {
    if (pos >= data->size())
    throw std::out_of_range(msg);
}
void StrBlob::pop_back() {
    check(0, "pop_back on empty StrBlob");
    data->pop_back();
}
std::string &StrBlob::front() {
    check(0, "front on empty StrBlob");
    return data->front();
}
```

StrBlob.cpp

```
const std::string &StrBlob::front() const {
    check(0, "front on empty StrBlob");
    return data->front();
}
std::string &StrBlob::back() {
    check(0, "back on empty StrBlob");
    return data->back();
}
const std::string &StrBlob::back() const {
    check(0, "back on empty StrBlob");
    return data->back();
}

StrBlobPtr StrBlob::begin() {
    return StrBlobPtr(*this);
}
StrBlobPtr StrBlob::end() {
    return StrBlobPtr(*this, data->size());
}
ConstStrBlobPtr StrBlob::cbegin() const {
    return ConstStrBlobPtr(*this);
}
```

Exercise 14.(27,28,30):

StrBlob.cpp

```
ConstStrBlobPtr StrBlob::cend() const {
return ConstStrBlobPtr(*this, data->size());
}
bool operator==(const StrBlob &lhs, const StrBlob &rhs) {
    //return lhs.data == rhs.data;  // compare identity(address)
    return *lhs.data == *rhs.data;  // compare value
}
bool operator!=(const StrBlob &lhs, const StrBlob &rhs) {
    return !(lhs == rhs);
}
bool operator<(const StrBlob &lhs, const StrBlob &rhs) {
    return *lhs.data < *rhs.data;  // compare value
}
bool operator>(const StrBlob &lhs, const StrBlob &rhs) {
    return rhs < lhs;
}
bool operator<=(const StrBlob &lhs, const StrBlob &rhs) {
    return !(lhs > rhs);
}
bool operator>=(const StrBlob &lhs, const StrBlob &rhs) {
    return !(lhs < rhs);
}
```

Exercise 14.29:

- We did not define a const version of the increment and decrement operators. Why not?

- **Answer:**
  - Because++ and --change the state of the object. Hence, it's meaningless to do so.

Refereces

Accreditation:

- This presentation is prepared/extracted from the following resources:
  - C++ Primer, Fifth Edition.
    Stanley B. Lippman Josée Lajoie Barbara E. Moo
  - https://github.com/jaege/Cpp-Primer-5th-Exercises
  - https://github.com/Mooophy/Cpp-Primer
  - https://github.com/pezy/CppPrimer