Advanced Programming Concepts with C++

CSI 2372

Tutorial

Selected exercises from chapter 7

Ahmedou Jreivine

Exercise 7.4:

- Write a class named Person that represents the name and address of a person. Use a string to hold each of these elements. Subsequent exercises will incrementally add features to this class.

Answer:

```
#include <string>
struct Person {
    std::string name;
    std::string address;
};
int main() {
    return 0;
}
```

Exercise 7.5:

- Provide operations in your `Person` class to return the name and address. Should these functions be `const`? Explain your choice.
- **Solution:**

```
#include <string>
struct Person {
        std::string getName() const { return name; }
        std::string getAddress() const { return address; }
        std::string name;
        std::string address;
};
// The member functions getName and  getAddress  should be const, because
// they don't change the object.
int main() {
        return 0;
}
```

Exercise 7.6:

- Define your own versions of the `add`, `read`, and `print` functions
- **Answer:**

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

uOttawa

Anwer 7.6:

```
Sales_data add(const Sales_data &lhs, const Sales_data &rhs) {
        Sales_data sum = lhs;  // Use default copy constructor
        sum.combine(rhs);
        return sum;
}
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;
        return os;
}
int main() {
        return 0;
}
```

uOttawa

Exercise 7.8:

- Why does ***read*** define its Sales_data parameter as a plain reference and ***print*** define its parameter as a reference to const?

- **Solution:**
  - The ***read*** function will change its Sales_data parameter and pass the information back via plain reference.
  - The ***print*** function won't change its Sales_data parameter, and by using a reference to const, we can print const Sales_data object as well.

Exercise 7.9:

- Add operations to read and print `Person` objects to the code you wrote for the exercises in § 7.1.2 (p. 260).

```
#include <string>
#include <iostream>
struct Person {
        std::string getName() const { return name; }
        std::string getAddress() const { return address; }
        std::string name;
        std::string address;
};
std::istream &read(std::istream &is, Person &rhs) {
        is >> rhs.name >> rhs.address;
        return is;
}
std::ostream &print(std::ostream &os, const Person &rhs) {
        os << rhs.getName() << " " << rhs.getAddress();
        return os;
}
int main() {
        Person p1;
        read(std::cin, p1);
        print(std::cout, p1) << std::endl;
        return 0;
}
```

uOttawa

Exercise 7.10:

- What does the condition in the following if statement do?
  - if (read(read(cin, data1), data2))

- **Answer:**
  - The condition test if both data1 and data2 are read correctly.

Exercise 7.11:

- Add constructors to your `Sales_data` class and write a program to use each of the constructors.
- **Solution:**

```
#include <string>
#include <iostream>
struct Sales_data {
        Sales_data() = default;
        Sales_data(const std::string &no) : bookNo(no) {}
        Sales_data(const std::string &no, unsigned us, double price): bookNo(no), units_sold(us), revenue(price * us) {}
        Sales_data(std::istream &is);
        std::string isbn() const { return bookNo; }
        Sales_data &combine(const Sales_data &);
        std::string bookNo;
        unsigned units_sold = 0;
        double revenue = 0.0;
};
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;
}
```

```
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;
        return os;
}
Sales_data::Sales_data(std::istream &is) {
        read(is, *this);
}
int main() {
        Sales_data d1;
        Sales_data d2("0-201-78345-X");
        Sales_data d3("0-201-78345-X", 5, 2.5);
        Sales_data d4(std::cin);
        print(std::cout, d1) << std::endl;
        print(std::cout, d2) << std::endl;
        print(std::cout, d3) << std::endl;
        print(std::cout, d4) << std::endl;
        return 0;
}
```

Exercise 7.12:

Move the definition of the `Sales_data` constructor that takes an `istream` into the body of the `Sales_data` class.

```
#include <string>
#include <iostream>
struct Sales_data;
std::istream &read(std::istream &is, Sales_data &item);
struct Sales_data {
    Sales_data() = default;
    Sales_data(const std::string &no) : bookNo(no) {}
    Sales_data(const std::string &no, unsigned us, double price) : bookNo(no), units_sold(us), revenue(price * us) {}
    Sales_data::Sales_data(std::istream &is) {
        read(is, *this);
    }
    std::string isbn() const { return bookNo; }
    Sales_data &combine(const Sales_data &);
    std::string bookNo;
    unsigned units_sold = 0;
    double revenue = 0.0;
};
Sales_data &Sales_data::combine(const Sales_data &rhs) {
                units_sold += rhs.units_sold;
                revenue += rhs.revenue;
                return *this;
}
```

uOttawa

```
Sales_data add(const Sales_data &lhs, const Sales_data &rhs) {
        Sales_data sum = lhs;  // Use default copy constructor
        sum.combine(rhs);
        return sum;
}
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;
        return os;
}
int main() {
        Sales_data d1;
        Sales_data d2("0-201-78345-X");
        Sales_data d3("0-201-78345-X", 5, 2.5);
        Sales_data d4(std::cin);
        print(std::cout, d1) << std::endl;
        print(std::cout, d2) << std::endl;
        print(std::cout, d3) << std::endl;
        print(std::cout, d4) << std::endl;
        return 0;
}
```

Exercise 7.14:

- **Write a version of the default constructor that explicitly initializes the members to the values we have provided as in-class initializers.**
- **Solution:**

```
#include <string>
#include <iostream>
struct Sales_data;
std::istream &read(std::istream &is, Sales_data &item);
struct Sales_data {
    Sales_data() : bookNo(""), units_sold(0), revenue(0.0) {}
    Sales_data(const std::string &no) : bookNo(no) {}
    Sales_data(const std::string &no, unsigned us, double price): bookNo(no),
    units_sold(us), revenue(price * us) {}
    Sales_data(std::istream &is) { read(is, *this);}
    std::string isbn() const { return bookNo; }
    Sales_data &combine(const Sales_data &);
    std::string bookNo;
    unsigned units_sold = 0;
    double revenue = 0.0;
};
```

```
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;
}
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;
        return os;
}
int main() {
        Sales_data d1;
        Sales_data d2("0-201-78345-X");
        Sales_data d3("0-201-78345-X", 5, 2.5);
        Sales_data d4(std::cin);
        print(std::cout, d1) << std::endl;
        print(std::cout, d2) << std::endl;
        print(std::cout, d3) << std::endl;
        print(std::cout, d4) << std::endl;
        return 0;
}
```

Exercise 7.15: Add appropriate constructors to your Person class

```
#include <string>
#include <iostream>
struct Person {
        Person() = default;
        Person(const std::string &n) : name(n) {}
        Person(const std::string &n, const std::string &a): name(n), address(a) {}
        Person(std::istream &);
        std::string getName() const { return name; }
        std::string getAddress() const { return address; }
        std::string name;
        std::string address;
};
std::istream &read(std::istream &is, Person &rhs) { is >> rhs.name >> rhs.address;return is;}
std::ostream &print(std::ostream &os, const Person &rhs) {os << rhs.getName() << " " << rhs.getAddress(); return os;}
Person::Person(std::istream &is) { read(is, *this); }
int main() {
        Person p1;
        Person p2("Zhang San");
        Person p3("Zhang San", "Earth");
        Person p4(std::cin);
        print(std::cout, p1) << std::endl;
        print(std::cout, p2) << std::endl;
        print(std::cout, p3) << std::endl;
        print(std::cout, p4) << std::endl;
        return 0;
}
```

Exercise 7.16:

- What, if any, are the constraints on where and how often an access specifier may appear inside a class definition? What kinds of members should be defined after a public specifier? What kinds should be private?
- **Answer:**
  - There are no restrictions on how often an access specifier may appear.
  - The specified access level remains in effect until the next access specifier or the end of the class body.
  - The members which are accessible to all parts of the program should define after a ***public*** specifier.
  - The members which are accessible to the member functions of the class but are not accessible to code that uses the class should define after a ***private*** specifier.

Exercise 7.19:

- Indicate which members of your Person class you would declare as public and which you would declare as private. Explain your choice.

- **Answer:**
  - The interface should be defined as ***public***, while the data member shouldn't be exposed to outside of the class.
  - ***public*** include: constructors, getName(), getAddress().
  - ***private*** include: name, address.

Exercise 7.30:

- It is legal but redundant to refer to members through the `this` pointer. Discuss the pros and cons of explicitly using the `this` pointer to access members.
- **Answer:**
- **Pros**
  - more explicit
  - less scope for misreading
  - member function parameters can have the same names as a member names.
    - void setAddr(const std::string &addr) { this->addr = addr; }
- **Cons**
  - more to read
  - sometimes redundant
    - std::string getAddr() const { return this->addr; } // unnecessary

Exercise 7.31:

- Define a pair of classes X and Y, in which X has a pointer to Y, and Y has an object of type X.
- **Answer**

```
struct Y;
struct X {
    Y *y;
};
struct Y {
    X x;
};
int main() {
    return 0;
}
```

Exercise 7.36:

- The following initializer is in error. Identify and fix the problem.

```
struct X {
    X(int i, int j) : base(i), rem(base % j) { }
    int rem, base;
};
```

- The order of member initialization is the same with the order they appear in the class definition. Since *rem* appears first, it will be initialized first. But the value of ***base*** is undefined when ***rem*** is initialized, thus the value of ***rem*** is undefined.
- To fix this, we can either switch the order of definitions of rem and base or we can use the constructor parameters i and j direct initialize rem(i % j).

Exercise 7.37:

- Using the version of Sales_data from this section, determine which constructor is used to initialize each of the following variables and list the values of the data members in each object:

```
Sales_data first_item(cin);
int main() {
   Sales_data next;
   Sales_data last("9-999-99999-9");
}
```

Answer 7.37:

```
Sales_data first_item(cin);
                // use Sales_data(std::istream &is) , its value is up to your input.
int main() {
    Sales_data next;      // use Sales_data(std::string s = "");
                          // bookNo = "",
                          //cnt = 0, revenue = 0.0
    Sales_data last("9-999-99999-9"); // use Sales_data(std::string s = "");
                                      // bookNo = "9-999-99999-9",
                                      // cnt = 0, revenue = 0.0
}
```

Exercise 7.43:

- Assume we have a class named `NoDefault` that has a constructor that takes an `int`, but has no default constructor. Define a class `C` that has a member of type `NoDefault`. Define the default constructor for `C`.
- **Answer:**

```
class NoDefault {
public:
        NoDefault(int i) : i_(i) {}
private:
        int i_;
};

class C {
public:
        C() : nd(0) {}
private:
        NoDefault nd;
};

int main() {
        C c;
        return 0;
}
```

Exercise 7.46:

- Which, if any, of the following statements are untrue? Why?
  - **(a)** A class must provide at least one constructor.
  - **(b)** A default constructor is a constructor with an empty parameter list.
  - **(c)** If there are no meaningful default values for a class, the class should not provide a default constructor.
  - **(d)** If a class does not define a default constructor, the compiler generates one that initializes each data member to the default value of its associated type.

Exercise 7.46:

- a) A class must provide at least one constructor. (**untrue**, "The compiler-generated constructor is known as the synthesized default constructor.")
- b) A default constructor is a constructor with an empty parameter list. (**untrue**, A default constructor is a constructor that is used if no initializer is supplied.What's more, A constructor that supplies default arguments for all its parameters also defines the default constructor)
- c) If there are no meaningful default values for a class, the class should not provide a default constructor. (**untrue**, the class should provide.)
- d) If a class does not define a default constructor, the compiler generates one that initializes each data member to the default value of its associated type. (**untrue**, only if our class does not explicitly define **any constructors**, the compiler will implicitly define the default constructor for us.)

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