C++ Exercises
Set4

Author(s): Tino Alferink, Robert van Ark, Owen Givlin

–

22:17

October 5, 2024

25

This assignment we will describe encapsulation and data hiding using a small example.

Data hiding describes that private data inside a specific section, or class, cannot be modified or be seen by other sections. This can be done indirectly using public member functions of the class implemented by the programmer. This way, due to the encapsulation, the class is responsible for it's own data integrity.

Listing 1: `bottle.h`

```
#ifndef INCLUDED_BOTTLE_
#define INCLUDED_BOTTLE_

class Bottle
{
    double d_capacity;
    double d_amount;

    public:
        Bottle(double capacity);

        double capacity() const;
        double amount() const;

        void addWater(double capacity);
        void removeWater(double amount);
};

#endif
```

This example shows a simple class describing a bottle with a capacity, where water can be added or removed. The functions of Bottle are public, and the data members are private.
The implementation of the class is irrelevant to show encapsulation and data hiding, as the interface provides controlled access and the implementation remains hidden from the user.

26

In this assignment the class Person, as described in C++ Annotations, is implemented and extended with insert and extract members.

Listing 2: `person/person.h`

```
#ifndef INCLUDED_PERSON_
#define INCLUDED_PERSON_

#include <string>
```

```
#include <iostream>

class Person
{
    public:
        Person(); // 1
        Person(std::string const &name,
               std::string const &address = "unknown",
               std::string const &phone = "unknown",
               std::string const &mass = "0"); // 2
        void setName(std::string const &name);
        void setAddress(std::string const &address);
        void setPhone(std::string const &phone);
        void setMass(std::string const &mass);

        std::string const &name() const;
        std::string const &address() const;
        std::string const &phone() const;
        size_t mass() const;

                                    // Insert info on this person into os.
        void insert(std::ostream &os) const;
                                    // Set data members using data from is.
        void extract(std::istream &is);
    private:
        std::string d_name;         // name of person
        std::string d_address;      // address field
        std::string d_phone;        // telephone number
        size_t      d_mass;         // the mass in kg
};

#endif
```

Listing 3: person/person.ih

```
#include "person.h"

using namespace std;
```

Listing 4: person/person1.cc

```
#include "person.ih"

Person::Person()
{
    d_mass = 0;
}
```

Listing 5: person/person2.cc

```
#include "person.ih"

Person::Person(string const &name, string const &address, string const &phone,
               string const &mass)
{
    setName(name);
    setAddress(address);
    setPhone(phone);
    setMass(mass);
}
```

Listing 6: person/setName.cc

```
#include "person.ih"

void Person::setName(string const &name)
{
    if(name.empty())
        cout << "A person must have a name.\n";
```

```
    else
        d_name = name;
}
```

Listing 7: person/setAddress.cc

```
#include "person.ih"

void Person::setAddress(string const &address)
{
    if(address.empty())
        d_address = "unknown";
    else
        d_address = address;
}
```

Listing 8: person/setPhone.cc

```
#include "person.ih"

void Person::setPhone(string const &phone)
{
    if(phone.empty())
        d_phone = "unknown";
    else if(phone.find_first_not_of("0123456789") == string::npos)
        d_phone = phone;
    else
    {
        cout << "Please provide a phone number only containing digits.\n";
        d_phone = "wrong input";
    }
}
```

Listing 9: person/setMass.cc

```
#include "person.ih"

void Person::setMass(string const &mass)
{
    if(mass.find_first_not_of("0123456789") != string::npos or mass.empty())
        cout << "Provide a correct mass, or 0.\n";
    else
    {
        size_t intMass = stoi(mass);
        if(intMass < 0)
            cout << "Provide a positive mass, or 0.\n";
        else
            d_mass = intMass;
    }
}
```

Listing 10: person/name.cc

```
#include "person.ih"

string const &Person::name() const
{
    return d_name;
}
```

Listing 11: person/address.cc

```
#include "person.ih"

string const &Person::address() const
{
    return d_address;
}
```

Listing 12: person/phone.cc

```
#include "person.ih"

string const &Person::phone() const
{
    return d_phone;
}
```

Listing 13: person/mass.cc

```
#include "person.ih"

size_t Person::mass() const
{
    return d_mass;
}
```

Listing 14: person/insert.cc

```
#include "person.ih"

void Person::insert(ostream &os) const
{
    os << "Name:    " << d_name << '\n'
       << "Address: " << d_address << '\n'
       << "Phone:   " << d_phone << '\n'
       << "Mass:    " << d_mass << '\n';
}
```

Listing 15: person/extract.cc

```
#include "person.ih"
#include <iostream>
#include <sstream>
#include <vector>

using namespace std;

void Person::extract(istream &is)
{
    string input;
    getline(is, input);
    istringstream iss(input);

    string part;
    getline(iss, part, ',');
    setName(part);
    getline(iss, part, ',');
    setAddress(part);
    getline(iss, part, ',');
    setPhone(part);
    getline(iss, part);
    setMass(part);
}
```

27

In this assignment we use our implementation of class Person in a simple program that reads and writes the data of multiple Person objects form and onto cin and cout.

Listing 16: program.h

```
#ifndef PROGRAM_H
#define PROGRAM_H

#include <vector>
```

```
#include "person/person.h"

void readPersons(std::vector<Person> &persons, size_t amt);
void writePersons(std::vector<Person> const &persons);

#endif
```

Listing 17: program.ih

```
#include "program.h"

#include <iostream>

using namespace std;

enum
{
    STD_NUM_PEOPLE = 5
};
```

Listing 18: readPersons.cc

```
#include "program.ih"

void readPersons(std::vector<Person> &persons, size_t amt)
{
    for (size_t num = 0; num < amt; ++num)
    {
        cout << "? ";
        Person person = Person();
        person.extract(cin);        // Read data from cin to person.
        persons.push_back(person);
    }
}
```

Listing 19: writePersons.cc

```
#include "program.ih"

void writePersons(const vector<Person> &persons)
{
    for (const auto &person : persons)
        person.insert(cout);
}
```

Listing 20: main.cc

```
#include "program.ih"

int main(int argc, char *argv[])
{
    size_t numPeople;
    if(argc > 1)                    // Get numPeople from arguments if provided.
        numPeople = stoi(argv[1]);
    else                            // Use predefined numPeople.
        numPeople = STD_NUM_PEOPLE;

    vector<Person> people;
    readPersons(people, numPeople);
    writePersons(people);
}
```

28

Design a small class Line returning substrings from lines.

Listing 21: line/line.h

```
#ifndef LINE_INCLUDED_
#define LINE_INCLUDED

#include <string>

class Line
{
    size_t d_pos;
    std::string d_currentLine;

    public:
        bool getLine();
        std::string next();
};

#endif
```

Listing 22: line/line.ih

```
#include "line.h"
#include <iostream>
#include <string>

using namespace std;
```

Listing 23: line/getline.cc

```
#include "line.ih"

bool Line::getLine()
{
    d_pos = 0;              // reset position index upon reading a new line
    getline(cin, d_currentLine);

    return d_currentLine.find_first_of(" \t") != string::npos;

}
```

Listing 24: line/next.cc

```
#include "line.ih"

string Line::next()
{
    if (d_pos == string::npos)
        return "";

    size_t startPos = d_currentLine.find_first_not_of(" \t", d_pos);
    if (startPos == string::npos)
    {
        d_pos = string::npos;
        return "";
    }

    size_t endPos = d_currentLine.find_first_of(" \t", startPos);
    if (endPos != string::npos)
        d_pos = endPos;
    else
        d_pos = string::npos;

    return d_currentLine.substr(startPos, endPos - startPos);

}
```

29

Design the class Parser. It's a class using the facilities of the class Line (cf. the previous exercise), and having a Line d_line data member.

Listing 25: parser/parser.h

```
#ifndef PARSER_INCLUDED_
#define PARSER_INCLUDED_

#include "../../28/line/line.h"
#include <string>

class Parser
{
    Line d_line;
    bool d_integral;

    public:
        enum Return                 // declared public so it can be used by
        {                           // class calculator
            NO_NUMBER,
            NUMBER,
            EOLN
        };

        bool reset();
        Return number(double *dest);
        bool isIntegral();
        std::string next();

    private:
        Return convert(double *dest, std::string const &str);
        bool pureDouble(double *dest, std::string const &str);

};

#endif
```

Listing 26: parser/parser.ih

```
#include "parser.h"

#include <string>

using namespace std;
```

Listing 27: parser/convert.cc

```
#include "parser.ih"

Parser::Return Parser::convert(double *dest, string const &str)
{
    try
    {
        return pureDouble(dest, str) ? NUMBER : NO_NUMBER;
    }
   catch (...)                      // conversion failed
    {
        return NO_NUMBER;
    }
}
```

Listing 28: parser/isintegral.cc

```
#include "parser.ih"

bool Parser::isIntegral()
{
    return d_integral;
}
```

Listing 29: parser/next.cc

```
#include "parser.ih"

string Parser::next()
{
    return d_line.next();
}
```

Listing 30: parser/number.cc

```
#include "parser.ih"

Parser::Return Parser::number(double *dest)
{
    string subString = d_line.next();
    if (subString.empty() == true)
        return Parser::EOLN;

    return convert(dest, subString);// Return (NO_)NUMBER based on successful
                                    // or failing conversion.
}
```

Listing 31: parser/puredouble.cc

```
#include "parser.ih"

bool Parser::pureDouble(double *dest, std::string const &str)
{
    size_t pos = 1;                 // ensure pos is nonzero
    *dest = stod(str, &pos);

    if (pos != str.length())        // no complete conversion
        return false;
    else                            // complete conversion
    {
        d_integral = str.find_first_of(".eE") == string::npos;
        return true;
    }
}
```

Listing 32: parser/reset.cc

```
#include "parser.ih"

bool Parser::reset()
{
    return d_line.getLine();
}
```

30

In this exercise, we create a Calculator class responsible for making simple calculations from the input evaluated from cin.

This class also has no constructor. No constructor is needed since we do not need any instance of Calculator. We have a single static public function, run(), which can be called directly on the class itself with no need to create a Calculator object. Next to that, Calculator has no non−static data members that work independently of class states.

Listing 33: calculator/calculator.h

```
#ifndef INCLUDED_CALCULATOR_
#define INCLUDED_CALCULATOR_

#include <string>
#include <sstream>

enum Operator
```

```
{
    NONE,                               // No operator
    PLUS,                               // +
    MINUS,                              // -
    TIMES,                              // *
    DIVIDE,                             // /
    MODULUS                             // %
};

class Calculator
{
    static std::istringstream s_iss;
    static double s_num1;
    static bool s_int1;
    static enum Operator s_op;
    static double s_num2;
    static bool s_int2;

    private:
        static bool const number(double *dest, bool *isInt);
        static bool const getOperator();
        static bool const expression();
        static void evaluate();

    public:
        static void run();
};

#endif
```

Listing 34: calculator/calculator.ih

```
#include "calculator.h"

#include <iostream>

using namespace std;
```

Listing 35: calculator/data.cc

```
#include "calculator.ih"

std::istringstream Calculator::s_iss;
double Calculator::s_num1;
bool Calculator::s_int1;
enum Operator Calculator::s_op;
double Calculator::s_num2;
bool Calculator::s_int2;
```

Listing 36: calculator/number.cc

```
#include "calculator.ih"

#include <cmath>
#include <regex>

namespace {
    bool isValidDouble(string const &input)
    {                                   // Create regex that matches double format.
        std::regex pattern(R"(^[+-]?(\d+(\.\d*)?|\.\d+)([eE][+-]?\d+)?$)");
                                        // Return true if input matches the format.
        return regex_match(input, pattern);
    }

    double checkDoubleSmall(double num)
    {
        if (abs(num) < 1e-8)
            return 0;
        return num;
    }
```

```
    bool checkInt(double num)
    {
        if (num == floor(num))
            return true;
        return false;
    }
}

bool const Calculator::number(double *dest, bool *isInt)
{
    string input;
    s_iss >> input;
    if (!isValidDouble(input))      // Check if string represents a double.
    {
        return false;
    }                               // If value is too small set it to 0.
    double num = checkDoubleSmall(stod(input));
    *dest = num;
    *isInt = checkInt(num);         // Set isInt depending on num value.

    return true;
}
```

Listing 37: calculator/getOperator.cc

```
#include "calculator.ih"

bool const Calculator::getOperator()
{
    string sign;
    s_iss >> sign;                  // Extract the operator from d_iss.
    if (sign.length() != 1)         // Ensure string sign is only 1 char.
        return false;
    switch (sign.front())           // Checking first char == checking sign.
    {
        case ('+'):
            s_op = PLUS;
            return true;
        case ('-'):
            s_op = MINUS;
            return true;
        case ('*'):
            s_op = TIMES;
            return true;
        case ('/'):
            s_op = DIVIDE;
            return true;
        case ('%'):
            s_op = MODULUS;
            return true;
        default:
            return false;
    }
}
```

Listing 38: calculator/expression.cc

```
#include "calculator.ih"

namespace {                         // test for non-neg ints in mod case.
    bool expressionModulus(double num1, bool int1, double num2, bool int2,
                           Operator op)
    {
        if (op != MODULUS) // If operator is not modulus then don't check this.
            return true;
        if ((num1 >= 0 and int1) and (num2 >= 0 and int2))
            return true;
        cout << "When using modulus, ensure both operands are non-negative "
             << "integrals.\n";
        return false;
```

```
    }
}

bool const Calculator::expression()
{
    bool correct = true;
    if (not number(&s_num1, &s_int1))
    {
        cout << "First parameter has to be a number.\n";
        correct = false;
    }
    if (not getOperator())
    {
        cout << "Second parameter has to be an operator.\n";
        correct = false;
    }
    if (not number(&s_num2, &s_int2))
    {
        cout << "Third parameter has to be a number.\n";
        correct = false;
    }
    if (!s_iss.eof())               // Expression should only have 3 parameters:
    {                               // Number Operator Number.
        cout << "No more than 3 parameters (formatted: num op num) should be "
             << "given.\n";
        return false;               // Immediately return false.
    }
    return expressionModulus(s_num1, s_int1, s_num2, s_int2, s_op) and correct;
}
```

Listing 39: calculator/evaluate.cc

```
#include "calculator.ih"

void Calculator::evaluate()
{
    cout << ": ";
    switch (s_op)                   // Make calculations.
    {
        case (PLUS):
            cout << s_num1 + s_num2;
            break;
        case (MINUS):
            cout << s_num1 - s_num2;
            break;
        case (TIMES):
            cout << s_num1 * s_num2;
            break;
        case (DIVIDE):
            cout << s_num1 / s_num2;
            break;
        case (MODULUS):
            cout << static_cast<int>(s_num1) % static_cast<int>(s_num2);
            break;
        default:
            cout << "Error.\n";     // Should not be reached.
    }
    cout << '\n';
}
```

Listing 40: calculator/run.cc

```
#include "calculator.ih"

#include <string>

void Calculator::run()
{
    cout << "? ";
    string input;
    while (getline(cin, input))
```

```
    {
        if (input.empty())          // If no input, end calculations.
            break;
        s_op = NONE;                // At start, set d_op to hold no operator.
        s_iss.clear();
        s_iss.str(input);
        if (expression())           // Evaluate whether input is correct.
            evaluate();             // Calculate and display the expression.
        cout << "? ";
    }
}
```

31

In this exercise, we use the Calculator class created in the previous assignment in a program and use it to perform some calculations. We also provide an example of how the program is used, so calculations and incorrect input handling are shown.

Listing 41: main.cc

```
#include "calculator/calculator.h"

int main()
{
    Calculator::run();
}
```

The following is a small illustration of how correct calculations are done and how incorrect input is dealt with:

```
? 13 + 15
: 28
? 15 - .5
: 14.5
? -4 * 4
: -16
? 16.5 / 1.5
: 11
? 4e3 % 45
: 40
? 13+15
First parameter has to be a number.
Second parameter has to be an operator.
Third parameter has to be a number.
? abc - def
First parameter has to be a number.
Third parameter has to be a number.
? 4 . 4
Second parameter has to be an operator.
? 12 % -5
When using modulus, ensure both operands are non-negative integrals.
? 2.5 % 1
When using modulus, ensure both operands are non-negative integrals.
?
```

32

The goal of this exercise is to compare different methods of precompiling headers when dealing with large projets with many source files.

Compiling and constructing the program on its own without precomiling the headers took 2:41.

While using precompiled headers it took 0:46 to compile and construct the program. However, this increase in speed came at the cost of the space taken up by the precompiled headers :

Name	Size (in MB)
uio	25.4
exception	31.3
string	34.8
mbuf	36.6
mstream	38.9
main	41.1
keys	43.9
student	44.6
arg	48.3
datetime	50.9
students	51.6

For a total combined of 436 MB taken up on the disk.

Using a single precompiled header (SPCH) the compilation and construction took 2:42. This however, includes the compilation of the header so constructing the program again without having to recompile the beader would be a big time saver. The SPCH takes up about 54.8 MB on the disk, which is an order of magnitude less than having all the headers precompiled individually.

When doing the compilation using all the CPUs the user time shows 8:26 but the real time shows 0:38 to compile and construct the program once again including the compilation time of the headers.

After deleting all the object, library and precompiled headers recompiling the program takes 0:02

Depending on the on the situation different methods would be more appropriate. Considering the size of precompiled headers, it makes most sense to use them in very large programs when compilation times become very long.

In a situation where the libraries in use are undergoing changes it makes sense to precompile them individually even though it takes up more disk space. Doing it this way means only the libraries that have been changed need to be recompiled before constructing the program thus saving a consierable amount of time.

While in a situation where the libraries are mostly set, or are public and very stable it makes sense to build a SPCH which cuts down significantly on the amount of storage used by the file and saves time when building the program. However if there were a change in one of the libraries the whole header would have to be recompiled.