Introduction to Data Abstraction, Algorithms and Data Structures

With C++ and the STL

Solutions to Exercises

Spring 2025

Alexis Maciel
Department of Computer Science
Clarkson University

Contents

Preface

This document contains solutions to the exercises of the course notes *Introduction to Data Abstraction, Algorithms and Data Structures: With C++ and the STL*. These notes were written for the course CS142 *Introduction to Computer Science II* taught at Clarkson University. The solutions are organized according to the same chapters and sections as the notes.

Here's some advice. Whether you are studying these notes as a student in a course or in self-directed study, your goal should be to understand the material well enough that you can do the exercises on your own. Simply studying the solutions is not the best way to achieve this. It is much better to spend a reasonable amount of time and effort trying to do the exercises yourself before looking at the solutions.

When an exercise asks you to write C++ code, you should compile it and test it. In the real world, programmers don't have solutions against which to check their code. Testing is a critical tool for ensuring that code is correct, so you need practice it.

Now, if you can't do an exercise on your own, you should study the notes some more. If that doesn't work, seek help from another student or from your instructor. Look at the solutions only to check your answer once you think you know how to do an exercise.

If you needed help doing an exercise, try redoing the same exercise later on your own. And do additional exercises.

If your solution to an exercise is different from the solution in this document, take the time to figure out why. Did you make a mistake? Did you forget something? Did you discover another correct solution? If you're not sure, ask another student or your instructor. If your solution turns out to be incorrect, fix it, after maybe getting some help, then try redoing the same exercise later on your own and do additional exercises.

Feedback on the notes and solutions is welcome. Please send comments to `alexis@clarkson.edu`.

Chapter 1

Data Abstraction and Object-Oriented Programming

1.1 A Data Type for Times

There are no exercises in this section.

1.2 Data Abstraction

There are no exercises in this section.

1.3 Modularity and Abstraction

There are no exercises in this section.

1.4 Names

There are no exercises in this section.

1.5 Object-Oriented Programming

1.5.4.

```
class Date
{
public:
    void read(istream& in = cin)
    {
        in >> month_;
        in.ignore(); // '/'
        in >> day_;
        in.ignore(); // '/'
        in >> year_;
    }

    void print(ostream& out = cout)
    {
        out << month_ << '/' << day_ << '/'
            << year_;
    }
```

```
        void print_in_words(ostream& out = cout)
        {
            switch (month_) {
                case 1: out << "January"; break;
                case 2: out << "February"; break;
                case 3: out << "March"; break;
                case 4: out << "April"; break;
                case 5: out << "May"; break;
                case 6: out << "June"; break;
                case 7: out << "July"; break;
                case 8: out << "August"; break;
                case 9: out << "September"; break;
                case 10: out << "October"; break;
                case 11: out << "November"; break;
                case 12: out << "December"; break;
                default: out << "Invalid";
            }
            out << ' ' << day_ << ", " << year_;
        }

    private:
        int year_;
        int month_
        int day_;
    };
```

1.5.5.

```
class ThreeDVector
{
public:
    void read(istream& in = cin)
    {
        char dummy;
        in >> dummy; // '('.
            // >> used instead of get() so
            // whitespace is skipped.
        in >> x_;
        in >> dummy; // ','
        in >> y_;
        in >> dummy; // ','
        in >> z_;
        in >> dummy; // ')'
    }

    void print(ostream& out = cout)
    {
        out << '(' << x_ << ", "<< y_ << ", "
            << z_ << ')';
    }
```

```
    ThreeDVector sum(const ThreeDVector& v)
    {
        ThreeDVector result;
        result.x_ = x_ + v.x_;
        result.y_ = y_ + v.y_;
        result.z_ = z_ + v.z_;
        return result;
    }

private:
    double x_;
    double y_;
    double z_;
};
```

1.5.6.

```
class Fraction
{
public:
    void read(istream& in = cin)
    {
        in >> a_;
        in.ignore(); // '/'
        in >> b_;
    }
```

```
    void print(ostream& out = cout)
    {
        out << a_ << '/' << b_;
    }

    Fraction sum(const Fraction& r)
    {
        Fraction result;
        result.a_ = a_ * r.b_ + r.a_ * b_;
        result.b_ = b_ * r.b_;
        return result;
    }

    Fraction product(const Fraction& r)
    {
        Fraction result;
        result.a_ = a_ * r.a_;
        result.b_ = b_ * r.b_;
        return result;
    }

private:
    int a_;
    int b_;
};
```

1.6 Constant Methods

1.6.3. In `Date`, the two print methods should be constant. In `ThreeDVector`, the methods `print` and `sum` should be constant. In `Fraction`, the methods `print`, `sum` and `product` should be constant.

1.7 Constructors

1.7.14.

```
class Date
{
public:
    Date() : Date(1, 1, 2000) {}
    Date(int month0, int day0, int year0) :
        month_(month0), day_(day0), year_(year0)

    ...
};
```

1.7.15.

```
class ThreeDVector {
public:
    ThreeDVector(): ThreeDVector(0, 0, 0) {}
    ThreeDVector(double x0, double y0,
                 double z0) :
        x_(x0), y_(y0), z_(z0) {}
```

```
    ThreeDVector sum(const ThreeDVector& v)
    {
        return ThreeDVector(x_ + v.x_,
                            y_ + v.y_,
                            z_ + v.z_);
    }

    ...
};
```

1.7.16.

```
class Fraction
{
public:
    Fraction() : Fraction(0) {}
    Fraction(int a0) : Fraction(a0, 1) {}
    Fraction(int a0, int b0) : a_(a0), b_(b0) {}

    Fraction sum(const Fraction& r) const
    {
        return Fraction(a_ * r.b_ + r.a_ * b_,
                        b_ * r.b_);
    }
```

```
    Fraction product(const Fraction& r) const
    {
        return Fraction(a_ * r.a_, b_ * r.b_);
    }

    ...
};
```

Chapter 2

More About Classes

2.1 Inline Methods

2.1.3.

```
class Date
{
public:
    Date() : Date(1, 1, 2000) {}
    Date(int month0, int day0, int year0) :
        month_(month0), day_(day0), year_(year0)

    void read(istream& in = cin);
    void print(ostream& out = cout) const;
    void print_in_words(ostream& out = cout)
        const;
```

```
private:
    int year_;
    int month_
    int day_;
};

void Date::read(istream& in)
{
    in >> month_;
    in.ignore(); // '/'
    in >> day_;
    in.ignore(); // '/'
    in >> year_;
}

inline void Date::print(ostream& out) const
{
    out << month_ << '/' << day_ << '/'
        << year_;
}
```

```
void Date::print_in_words(ostream& out) const
{
    switch (month_) {
        case 1: out << "January"; break;
        case 2: out << "February"; break;
        case 3: out << "March"; break;
        case 4: out << "April"; break;
        case 5: out << "May"; break;
        case 6: out << "June"; break;
        case 7: out << "July"; break;
        case 8: out << "August"; break;
        case 9: out << "September"; break;
        case 10: out << "October"; break;
        case 11: out << "November"; break;
        case 12: out << "December"; break;
        default: out << "Invalid";
    }
    out << ' ' << day_ << ", " << year_;
}


class ThreeDVector
{
public:
    ThreeDVector(): ThreeDVector(0, 0, 0) {}
    ThreeDVector(double x0, double y0,
                 double z0) :
        x_(x0), y_(y0), z_(z0) {}

    void read(istream& in);
```

```
    void print(ostream& out) const;

    ThreeDVector sum(const ThreeDVector& v)
            const;

private:
    double x_;
    double y_;
    double z_;
};

void ThreeDVector::read(istream& in)
{
    char dummy;
    in >> dummy; // '('.
        // >> used instead of ignore() so
        // whitespace is skipped.
    in >> x_;
    in >> dummy; // ','
    in >> y_;
    in >> dummy; // ','
    in >> z_;
    in >> dummy; // ')'
}
```

```
inline void ThreeDVector::print(ostream& out)
        const
{
    out << '(' << x_ << ", "<< y_ << ", "
        << z_ << ')';
}

ThreeDVector ThreeDVector::sum(
        const ThreeDVector& v) const
{
    return ThreeDVector(x_ + v.x_,
                        y_ + v.y_,
                        z_ + v.z_);
}


class Fraction
{
public:
    Fraction() : Fraction(0) {}
    Fraction(int a0) : Fraction(a0, 1) {}
    Fraction(int a0, int b0) : a_(a0), b_(b0) {}

    void read(istream& in = cin);
    void print(ostream& out = cout) const
    {
        out << a_ << '/' << b_;
    }
```

```
    Fraction sum(const Fraction& r) const
    {
        return Fraction(a_ * r.b_ + r.a_ * b_,
                        b_ * r.b_);
    }
    Fraction product(const Fraction& r) const
    {
        return Fraction(a_ * r.a_, b_ * r.b_);
    }

private:
    int a_;
    int b_;
};

inline void Fraction::read(istream& in)
{
    in >> a_;
    in.ignore(); // '/'
    in >> b_;
}
```

2.2 Get and Set Methods

2.2.4.

```
void print_format_12h(const Time& t,
                      ostream& out)
{
    int hours = t.hours();
    int minutes = t.minutes();

    if (hours == 0) {
        out << "12";
    }
    else if (hours <= 12) {
        out << hours;
    }
    else {
        out << hours - 12;
    }

    out << ':';
    if (minutes < 10) {
        out << '0';
    }
    out << minutes;
```

```
        if (hours < 12) {
            out << " a.m.";
        }
        else {
            out << " p.m.";
        }
    }
```

2.2.5.

```
class Time
{
public:
    Time() : total_minutes_(6039) {}
        // 6039 = 99*60 + 99
    Time(int h) : Time(h, 0) {}
    Time(int h, int m)
    {
        total_minutes_ = h*60 + m;
    }

    void read(istream& in = cin);
    void print(ostream& out = cout) const;
    bool less_than(const Time& t) const;

    int hours() const
    {
        return total_minutes_ / 60;
    }
    int minutes() const
    {
        return total_minutes_ % 60;
    }

    void set_hours(int new_hours);
    void set_minutes(int new_minutes);
```

```
    void set(int new_hours,
             int new_minutes = 0);

private:
    int total_minutes_;
};

void Time::read(istream& in)
{
    int hours;
    in >> hours;
    in.ignore(); // colon
    int minutes;
    in >> minutes;
    total_minutes_ = hours*60 + minutes;
}

void Time::print(ostream& out) const
{
    out << hours() << ':';
    int m = minutes();
    if (m < 10) {
        out << '0';
    }
    out << m;
}
```

```
inline bool Time::less_than(const Time& t)
        const
{
    return (total_minutes_ < t.total_minutes_);
}

inline void Time::set_hours(int new_hours)
{
    total_minutes_ = new_hours * 60 + minutes();
}

inline void Time::set_minutes(int new_minutes)
{
    total_minutes_ = total_minutes_ - minutes()
                          + new_minutes;
}

inline void Time::set(int new_hours,
                      int new_minutes)
{
    total_minutes_ = new_hours * 60
                          + new_minutes;
}
```

2.2.6.

```
inline void Date::set(int month0, int day0)
{
    month_ = month0;
    day_ = day0;
}

inline void Date::set(int month0, int day0,
                      int year0)
{
    set(month0, day0);
    year = year0;
}
```

2.2.7.

```
class ThreeDVector
{
public:
    ...

    void set(double x0, double y0, double z0);

    double x() const { return x_; }
    double y() const { return y_; }
    double z() const { return z_; }
};
```

```
inline void ThreeDVector::set(double x0,
                              double y0,
                              double z0)
{
    x_ = x0;
    y_ = y0;
    z_ = z0;
}
```

2.2.8.

```
class Fraction
{
public:
    void set(int a0, int b0 = 1);

    int numerator() const { return a_; }
    int denominator() const { return b_; }

    ...
};

inline void Fraction::set(int a0, int b0)
{
    a_ = a0;
    b_ = b0;
}
```

2.3 Operator Overloading

2.3.5.

```
inline bool operator==(const Time& t1,
                       const Time& t2)
{
    return t1.hours() == t2.hours() &&
        t1.minutes() == t2.minutes();
}
```

2.3.6.

```
class Date
{
public:
    friend istream& operator>>(istream& in,
                               Date& d);
    friend ostream& operator<<(
              ostream& out, const Date& d);

    void operator+=(int num_days);

    ...
};
```

```
istream& operator>>(istream& in, Date& d)
{
    in >> d.month_;
    in.ignore(); // '/'
    in >> d.day_;
    in.ignore(); // '/'
    in >> d.year_;
    return in;
}

inline ostream& operator<<(ostream& out,
                           const Date& d)
{
    out << d.month_ << '/' << d.day_ << '/'
        << d.year_;
    return out;
}

void Date::operator+=(int num_day)
{
    ... // same as Date::add
}
```

2.3.7.

```
class ThreeDVector
{
public:
    friend istream& operator>>(
            istream& in, ThreeDVector& v);

    ThreeDVector operator+(
            const ThreeDVector& v2) const
    {
        return ThreeDVector(x_ + v2.x_,
                            y_ + v2.y_,
                            z_ + v2.z_);
    }

    ...
};
```

```
istream& operator>>(istream& in,
                    ThreeDVector& v)
{
    char dummy;
    in >> dummy; // '('.
        // >> used instead of ignore() so
        // whitespace is skipped.
    in >> x_;
    in >> dummy; // ','
    in >> y_;
    in >> dummy; // ','
    in >> z_;
    in >> dummy; // ')'
    return in;
}

inline ostream& operator<<(
        ostream& out, const ThreeDVector& v)
{
    out << '(' << v.x() << ", "<< v.y() << ", "
        << v.z() << ')';
    return out;
}
```

2.3.8.

```
inline Fraction operator+(const Fraction& r1,
                          const Fraction& r2)
{
    return Fraction(r1.numerator()
                        * r2.denominator()
                        + r2.numerator()
                        * r1.denominator(),
                    r1.denominator()
                        * r2.denominator());
}
```

2.3.9.

```
inline bool operator<(const Fraction& r1,
                      const Fraction& r2)
{
    return r1.numerator() * r2.denominator()
             < r2.numerator()
                   * r1.denominator();
}

inline bool operator==(const Fraction& r1,
                       const Fraction& r2)
{
    return r1.numerator() * r2.denominator()
             == r2.numerator()
                    * r1.denominator();
}
```

2.4 Compiling Large Programs

2.4.3.

```
// Date.h

#ifndef _Date_h_
#define _Date_h_

#include <iostream>

class Date
{
public:
    friend std::istream& operator>>(
                std::istream& in, Date& d);
    friend std::ostream& operator<<(
                std::ostream& out,
                const Date& d);

    Date() : Date(1, 1, 2000) {}
    Date(int month0, int day0, int year0);

    void print_in_words(
        std::ostream& out = cout) const;

    void set(int month0, int day0);
    void set(int month0, int day0, int year0);

    void operator+=(int num_days);
```

```
private:
    int year;
    int month;
    int day;
};

inline Date::Date(int month0, int day0,
                  int year0) :
    month_(month0), day_(day0), year_(year0) {}

inline std::ostream& operator<<(
        std::ostream& out, const Date& d)
{
    ...
}

inline void Date::set(int month0, int day0)
{
    ...
}

inline void Date::set(int month0, int day0,
                      int year0)
{
    ...
}

#endif
```

```
// Date.cpp

#include "Date.h"

using namespace std;

istream& operator>>(istream& in, Date& d)
{
    ...
}

void Date::print_in_words(ostream& out) const
{
    ...
}

void Date::operator+=(int num_days)
{
    ...
}
```

2.5 The make Utility

There are no exercises in this section.

Chapter 3

Strings and Streams

3.1 C Strings

3.1.4.

```
void println(const char cs[])
{
    int i = 0;
    while (cs[i] != '\0') {
        cout << cs[i];
        ++i;
    }
    cout << endl;
}
```

3.1.5.

```
void my_strlwr(char cs[])
{
    int i = 0;
    while (cs[i] != '\0') {
        cs[i] = tolower(cs[i]);
        ++i;
    }
}
```

3.1.6.

```
void my_strcpy(char dest[],
               const char source[])
{
    int i = 0;
    while (source[i] != '\0') {
        dest[i] = source[i];
        ++i;
    }
    dest[i] = '\0';
}
```

```
void my_strncpy(char dest[],
                const char source[],
                int n)
{
    int i = 0;
    // i is the next location to be copied and
    // also the number of chars copied so far
    while (i < n && source[i] != '\0') {
        dest[i] = source[i];
        ++i;
    }
    if (i < n) {
        dest[i] = '\0';
    }
}

void my_strcat(char dest[],
               const char source[])
{
    int i = strlen(dest); // index in dest
    int j = 0; // index in source
    while (source[j] != '\0') {
        dest[i] = source[j];
        ++i;
        ++j;
    }
    dest[i] = '\0';
}
```

```
void my_strncat(char dest[],
                const char source[],
                int n)
{
    int i = strlen(dest); // index in dest
    int j = 0; // index in source
    while (j < n && source[j] != '\0') {
        dest[i] = source[j];
        ++i;
        ++j;
    }
    dest[i] = '\0';
}
```

```
int my_strcmp(const char cs1[],
              const char cs2[])
{
    int i = 0;
    while (cs1[i] != '\0'
             && cs2[i] != '\0'
             && cs1[i] == cs2[i])
        ++i;
    // i is first position where strings differ
    // or where one has ended
    if (cs1[i] == '\0' && cs2[i] == '\0')
            // both strings ended
        return 0;
    else if (cs1[i] == '\0')
            // cs1 ended but not cs2
        return -1;
    else if (cs2[i] == '\0')
            // cs2 ended but not cs1
        return 1;
    else if (cs1[i] < cs2[i])
            // strings differ at index i
        return -1;
    else
        return 1;
}
```

3.2 C++ Strings

3.2.3.

```
void println(const string& s)
{
    for (int i = 0; i < s.length(); ++i) {
        cout << s[i];
    }
    cout << endl;
}
```

3.2.4. All three parts begin with

```
string s = "Jane Doe";
```

a)

```
string s2;
s2.resize(s.length() + 1);
    // + 1 for the comma

// Find space in s
int i = 0;
while (s[i] != ' ') {
    ++i;
}
int ix_space = i;
```

```
// Copy last name
i = ix_space + 1; // index in s1
int j = 0; // index in s2
while (i < s.length()) {
    s2[j] = s[i];
    ++i;
    ++j;
}

// Add comma and space
s2[j] = ',';
++j;
s2[j] = ' ';
++j;

// Copy first name
i = 0;
while (i < ix_space) {
    s2[j] = s[i];
    ++i;
    ++j;
}
```

b)
```
string s2;
int ix_space = s.find(' ');

// Copy last name
for (int i = ix_space + 1;
        i < s.length();
        ++i) {
    s2 += s[i];
}

s2 += ", ";

// Copy first name
for (int i = 0; i < ix_space; ++i) {
    s2 += s[i];
}
```

c)
```
string s2;
int ix_space = s.find(' ');
s2.append(s, ix_space + 1, s.npos);
    // last name
s2 += ", ";
s2.append(s, 0, ix_space); // first name
```

Chapter 4

Vectors

4.1 A Simple File Viewer

There are no exercises in this section.

4.2 Vectors in the STL

4.2.6.

```
void println(const vector<int>& v)
{
    for (int x : v) {
        cout << x << ' ';
    }
    cout << endl;
}
```

```
void println(const vector<string>& v)
{
    for (const string& x : v) {
        cout << '\"' << x << "\" ";
    }
    cout << endl;
}

int main()
{
    vector<int> v0;
    cout << "An empty vector of integers: ";
    println(v0);
    vector<int> v1(3);
    cout << "A vector with three "
         << "value-initialized integers: ";
    println(v1);

    vector<string> v2(3);
    cout << "A vector with three empty "
         << "strings: ";
    println(v2);

    vector<int> v3(3, 17);
    cout << "A vector with three 17's: ";
    println(v3);

    vector<int> v4(v3);
    cout << "A copy of the previous vector: ";
    println(v4);
```

```
v4.front() = 1;
v4.back() = 23;
cout << "The last vector with its first "
     << "and last elements changed to 1 "
     << "and 23: ";
println(v4);
v4.resize(2);
cout << "The last vector shrunk to size 2: ";
println(v4);

v4.resize(4);
cout << "The last vector grown to size 4: ";
println(v4);

v4.resize(6, 42);
cout << "The last vector grown to size 6 "
     << "and padded with 42's: ";
println(v4);

v4.push_back(60);
cout << "The last vector with a 60 added to "
     << "its back: ";
println(v4);

v4.pop_back();
cout << "The last vector with its back "
     << "element removed: ";
println(v4);
```

```
vector<int> v5;
vector<int> v6;
v5 = v6 = v4;
cout << "Two new copies of the last "
     << "vector:\n";
println(v5);
println(v6);

v6.clear();
cout << "The last vector with all its "
     << "elements removed: ";
println(v6);

cout << "The last vector is empty: ";
if (v6.empty())
    cout << "true";
else
    cout << "false";
cout << endl;

cout << "The other one before that is "
     << "empty: ";
if (v5.empty())
    cout << "true";
else
    cout << "false";
cout << endl;
```

```
    cout << "The maximum size of the last "
         << "vector: "
         << scientific << setprecision(2)
         << double(v6.max_size()) << endl;

    v6.assign(5, 12);
    cout << "The last vector with five 12's: ";
    println(v6);

    cout << "The last two vectors:\n";
    println(v5);
    println(v6);
    v5.swap(v6);
    cout << "The same vectors with their "
         << "contents swapped:\n";
    println(v5);
    println(v6);

    return 0;
}
```

4.2.7. The easiest (and safest) solution is to use a range-for loop:

```
void print(const vector<int>& v, ostream& out)
{
    for (int x : v) {
        out << x << '\n';
    }
}
```

But it's also possible to use the indexing operator:

```
void print(const vector<int>& v, ostream& out)
{
    for (int i = 0; i < v.size(); ++i) {
        out << v[i] << endl;
    }
}
```

4.2.8.

```
void replace(vector<int>& v, int x, int y)
{
    for (int& e : v) {
        if (e == x) {
            e = y;
        }
    }
}
```

4.2.9.

```
int count(const vector<int>& v, int x)
{
    int total = 0;
    for (int e : v) {
        if (e == x) {
            ++total;
        }
    }
    return total;
}
```

4.2.10.

```
void concatenate(const vector<int>& v1,
                 const vector<int>& v2,
                 vector<int>& v3)
{
    v3.clear();
    v3.reserve(v1.size() + v2.size());
    for (int x : v1) {
        v3.push_back(x);
    }
    for (int x : v2) {
        v3.push_back(x);
    }
}
```

4.3 The Software Life Cycle

There are no exercises in this section.

4.4 The Software Development Process

There are no exercises in this section.

4.5 Specification of the File Viewer

There are no exercises in this section.

4.6 Design of the File Viewer

There are no exercises in this section.

4.7 Implementation of the File Viewer

No solutions are available for this section.

4.8 Arrays

No solutions are available for this section.

Chapter 5

Generic Algorithms

5.1 Introduction

5.1.3.

```
cout << "4 = " << max(3, 4) << endl;
cout << "bob = "
     << max<string>("alice", "bob") << endl;

vector<int> v1 = {37, 12, 23, 37, 60, 37};
cout << "3 = "
     << count(v1.begin(), v1.end(), 37) << endl;
cout << "0 = "
     << count(v1.begin(), v1.end(), 5) << endl;
```

```
sort(v1.begin(), v1.end());
for (int x : v1)
    cout << x << ' ';
cout << endl;

vector<string> v2 = {"bob", "alice", "charlie",
                     "elaine", "diane"};
cout << "1 = "
     << count(v2.begin(), v2.end(), "alice")
     << endl;
cout << "0 = "
     << count(v2.begin(), v2.end(), "john")
     << endl;

sort(v2.begin(), v2.end());
for (const string& s : v2)
    cout << s << ' ';
cout << endl;
```

5.2 Iterators

5.2.7.

a)
```
int count = std::count(
                    v.begin(),
                    v.begin() + v.size()/2,
                    0);
```

b)
```
auto itr = find(v.begin(), v.end(), 0);
if (itr != v.end())
    *itr = 1;
```

c)
```
auto itr = find(v.begin(), v.end(), 0);
for (auto itr2 = v.begin();
        itr2 != itr;
        ++itr2)
    ++(*itr2);
```

5.2.8.

```
int count(vector<int>::iterator start,
          vector<int>::iterator stop,
          int e)
{
    int count = 0;
    for (auto itr = start; itr != stop; ++itr)
        if (*itr == e)
            ++count;
    return count;
}

void fill(vector<int>::iterator start,
          vector<int>::iterator stop,
          int e)
{
    for (auto itr = start; itr != stop; ++itr)
        *itr = e;
}
```

```
vector<int>::iterator find(
        vector<int>::iterator start,
        vector<int>::iterator stop,
        int e)
{
    for (auto itr = start; itr != stop; ++itr)
        if (*itr == e)
            return itr;
    return stop;
}

void replace(vector<int>::iterator start,
             vector<int>::iterator stop,
             int x, int y)
{
    for (auto itr = start; itr != stop; ++itr)
        if (*itr == x)
            *itr = y;
}
```

5.3 Iterator Types and Categories

No solutions are available for this section.

5.4 Vectors and Iterators

No solutions are available for this section.

5.5 Algorithms in the STL

5.5.4. Because elements in the subrange `[start2,stop1)` would be overwritten before they are copied. For example, if a sequence contains `1 2 3 4 5 6` and we tried to copy the elements `1 2 3 4` to the position where `3` is, then the result would be `1 2 1 2 1 2`, not `1 2 1 2 3 4`.

5.6 Implementing Generic Algorithms

5.6.4.

```
template <typename T>
void swap(T& x, T& y)
{
    T temp = x;
    x = y;
    y = temp;
}
```

```
template <typename Iterator>
Iterator max_element(Iterator start,
                     Iterator stop)
{
    if (start == stop)
        return stop;

    Iterator itr_max_so_far = start;
    for (Iterator itr = next(start);
            itr != stop;
            ++itr) {
        if (*itr_max_so_far < *itr)
            itr_max_so_far = itr;
    }
    return itr_max_so_far;
}
```

```
template <typename Itr1, typename Itr2>
Itr2 copy(Itr1 start1,
          Itr1 stop1,
          Itr2 start2)
{
    while (start1 != stop1) {
        *start2 = *start1;
        ++start1;
        ++start2;
    }
    return start2;
}

template <typename Itr1, typename Itr2>
Itr2 copy_backward(Itr1 start1,
                   Itr1 stop1,
                   Itr2 stop2)
{
    while (stop1 != start1) {
        --stop1;
        --stop2;
        *stop2 = *stop1;
    }
    return stop2;
}
```

```
template <typename Iterator>
void reverse(Iterator start, Iterator stop)
{
    if (start == stop)
        return;

    auto first = start;
    auto last = prev(stop);
    while (first != last) {
        swap(*first, *last);
        ++first;
        if (first != last)
            --last;
    }
}
```

5.7 Initializer Lists

5.7.2. The easiest solution is to use a range-for loop:

```
template <typename T>
void print(initializer_list<T> init_list,
           ostream& out)
{
    for (const T& e : init_list)
        out << e << endl;
}
```

It's also possible to use iterators directly:

```
template <typename T>
void print(initializer_list<T> init_list,
           ostream& out)
{
    for (auto itr = init_list.begin();
            itr != init_list.end();
            ++itr) {
        out << *itr << endl;
    }
}
```

5.8 Function as Arguments

5.8.2.

```
inline bool even(int x)
{
    return x % 2 == 0;
}

count_if(v.begin(), v.end(), even)
```

5.8.3.

```
bool shorter_than(const string& s1,
                  const string& s2)
{
    return (s1.length() < s2.length()) ||
               (s1.length() == s2.length() &&
                   s1 < s2);
}

sort(v.begin(), v.end(), shorter_than);
```

5.8.4.

```
template <typename Iterator,
          typename UnaryPredicate>
Iterator find_if(Iterator start,
                 Iterator stop,
                 UnaryPredicate f)
{
    for (Iterator itr = start;
            itr != stop;
            ++itr) {
        if (f(*itr))
            return itr;
    }
    return stop;
}
```

```
template <typename Iterator,
          typename UnaryPredicate,
          typename T>
void replace_if(Iterator start,
                Iterator stop,
                UnaryPredicate f,
                T y)
{
    for (Iterator itr = start;
            itr != stop;
            ++itr) {
        if (f(*itr))
            *itr = y;
    }
}
```

```
template <typename Iterator,
          typename BinaryPredicate>
Iterator max_element(Iterator start,
                     Iterator stop,
                     BinaryPredicate less_than)
{
    if (start == stop)
        return stop;

    Iterator itr_max_so_far = start;
    for (Iterator itr = next(start);
            itr != stop;
            ++itr) {
        if (less_than(*itr_max_so_far, *itr))
            itr_max_so_far = itr;
    }
    return itr_max_so_far;
}
```

```
template <typename SourceItr,
          typename DestItr,
          typename UnaryPredicate>
DestItr copy_if(SourceItr start,
                SourceItr stop,
                DestItr dest_begin,
                UnaryPredicate f)
{
    DestItr dest_itr = dest_begin;
    for (SourceItr itr = start;
            itr != stop;
            ++itr) {
        if (f(*itr)) {
            *dest_itr = *itr;
            ++dest_itr;
        }
    }
    return dest_itr;
}
```

```
template <typename Iterator,
          typename UnaryFunction>
UnaryFunction for_each(Iterator start,
                       Iterator stop,
                       UnaryFunction f)
{
    for (Iterator itr = start;
            itr != stop;
            ++itr) {
        f(*itr);
    }
    return f;
}
```

5.9 Function Objects

5.9.2.

```
class Multiple
{
public:
    Multiple(int m0) : m(m0) {}
    bool operator()(int x)
    {
        return x % m == 0;
    }
private:
    int m;
};

count_if(v.begin(), v.end(), Multiple(3))
```

Chapter 6

Linked Lists

6.1 A Simple Text Editor

There are no exercises in this section.

6.2 Vector Version of the Text Editor

No solutions are available for this section.

6.3 Vectors and Linked Lists

There are no exercises in this section.

6.4 Linked Lists in the STL

6.4.3.

```
template <typename T>
void print(const list<T>& ls)
{
    for (const T& e : ls) {
        cout << e << '\n';
    }
}
```

6.4.4.

```
template <typename T>
void concatenate(const list<T>& ls1,
                 const list<T>& ls2,
                 list<T>& result)
{
    result = ls1;
    for (const T& e : ls2) {
        result.push_back(e);
    }
}
```

6.4.5.

```
void make_double_spaced(list<string>& ls)
{
    string empty_string;
    for (auto itr = ls.begin();
            itr != ls.end();
            ++itr) {
        ls.insert(itr, empty_string);
    }
}
```

6.4.6.

```
template <typename T, typename Iterator>
typename list<T>::iterator insert(
        list<T>& ls,
        typename list<T>::iterator itr,
        Iterator start,
        Iterator stop)
{
    if (start == stop) {
        return itr;
    }
    else {
        auto source = start;
        auto result = ls.insert(itr, *source);
        ++source;
        while (source != stop) {
            ls.insert(itr, *source);
            ++source;
        }
        return result;
    }
}
```

6.4.7.

```
template <typename T>
typename list<T>::iterator erase(
        list<T>& ls,
        typename list<T>::iterator start,
        typename list<T>::iterator stop)
{
    auto target = start;
    while (target != stop) {
       target = ls.erase(target);
    }
    return stop;
}
```

6.5 List Version of the Text Editor

No solutions are available for this section.

Chapter 7

Maps

7.1 A Phone Book

There are no exercises in this section.

7.2 Maps in the STL

7.2.6.

```
using HometownMap = map<string, string>;

void read_hometowns(HometownMap& m,
                    const string& file_name)
{
    ifstream in(file_name);
    string name;
    string town;

    while (getline(in, name)) {
        getline(in, town);
        m[name] = town;
    }
}

void print_hometowns(const HometownMap& m)
{
    for (const auto& p : m)
        cout << p.first << ": " << p.second
             << endl;
}
```

```
void print_local(const HometownMap& m,
                 const string& town)
{
    for (const auto& p : m)
        if (p.second == town)
            cout << p.first << endl;
}
```

7.2.7.

```
using AppointmentMap = map<Time, string>;

void read_appointments(AppointmentMap& m,
                       const string& file_name)
{
    ifstream in(file_name);
    Time t;
    string appt;

    while (in >> t) {
        in.get(); // the space between the time
            // and the description
        getline(in, appt);
        m[t] = appt;
    }
}
```

```
void print_appointments(const AppointmentMap& m)
{
    for (const auto& p : m)
        cout << p.first << ": " << p.second
             << endl;
}
```

A simple version of `print_upcoming`:

```
void print_upcoming(const AppointmentMap& m,
                    const Time& t)
{
    for (const auto& p : m)
        if (p.first >= t)
            cout << p.first << ": " << p.second
                 << endl;
}
```

A more efficient solution because it avoids traversing the entire map:

```
void print_upcoming(const AppointmentMap& m,
                    const Time& t)
{
    auto itr = m.lower_bound(t);
    for (; itr != m.end(); ++itr)
        cout << itr->first << ": "
             << itr->second << endl;
}
```

A test driver:

```
int main()
{
    AppointmentMap m;
    read_appointments(m, "appointments.txt");
    print_appointments(m);
    print_upcoming(m, {14, 30});
    return 0;
}
```

7.3 Design and Implementation of the Phone Book

7.3.4.

```
struct Product
{
    int number;
    string name;
    double price;
};
```

```
istream & operator>>(istream& in, Product& p)
{
    in >> p.number;
    in.get();
    getline(in, p.name);
    in >> p.price;
    in.get();
    return in;
}

using ProductMap = map<int, Product>;

void read_products(ProductMap& m,
                   const string& file_name)
{
    ifstream in(file_name);
    Product p;

    while (in >> p)
        m[p.number] = p;
}

void print_cheap_products(const ProductMap& m,
                          double price)
{
    for (const auto& data : m)
        if (data.second.price < price)
            cout << data.second.name << endl;
}
```

```
void copy_cheap_products(const ProductMap& m1,
                         double price,
                         ProductMap& m2)
{
    for (const auto& data : m1)
        if (data.second.price < price)
            m2.insert(m2.end(), data);
}
```

Chapter 8

Dynamically Allocated Arrays

8.1 The Size of Ordinary Arrays

There are no exercises in this section.

8.2 The Dynamic Allocation of Arrays

There are no exercises in this section.

8.3 Programming with Dynamically Allocated Arrays

8.3.5.

```
template <class T>
T* copy(const T* a, int n)
{
    T* b = new T[n];
    std::copy(a, a + n, b);
    return b;
}
```

8.3.6.

```
template <class T>
T* concatenate(const T* a, int n,
               const T* b, int m)
{
    T* c = new T[n + m];
    std::copy(a, a + n, c);
    std::copy(b, b + m, c + n);
    return c;
}
```

8.3.7.

a) Does not compile because an array cannot be assigned a value by the assignment operator.

b) Same thing.

c) Makes `c` point to the array `b` so that `b` and `c` refer to the same array.

d) Makes `c` point to the array that `d` points to so that `c` and `d` point to the same array.

Chapter 9

Implementation of Vectors

9.1 A Basic Class of Vectors

9.1.4.

```
vector(int n, const T& e)
{
    if (n == 0) {
        buffer_ = nullptr;
        size_ = 0;
    }
    else {
        buffer_ = new T[n];
        std::fill(buffer_, buffer_ + n, e);
        size_ = n;
    }
}
```

```
bool empty() const { return (size_ == 0); }

T& back() { return buffer_[size_ - 1]; }
const T& back() const
{
    return buffer_[size_ - 1];
}

void clear()
{
    delete[] buffer_;
    buffer_ = nullptr;
    size_ = 0;
}

void swap(vector<T>& v)
{
    std::swap(buffer_, v.buffer_);
    std::swap(size_, v.size_);
}
```

To take advantage of possible implicit conversions on both sides, it's better to implement the operators == and != as standalone functions:

```
template <class T>
inline bool operator==(const vector<T>& v1,
                       const vector<T>& v2)
{
    if (v1.size() != v2.size())
        return false;
    else
        return std::equal(v1.begin(), v1.end(),
                          v2.begin());
}

template <class T>
inline bool operator!=(const vector<T>& v1,
                       const vector<T>& v2)
{
    return !(v1 == v2);
}
```

9.1.5. The revised destructor:

```
~vector()
{
    delete[] buffer_;
    cout << "The destructor of class vector "
         << "was executed.\n";
}
```

A test driver:

```
int main()
{
    vector<int> v;
    return 0;
}
```

9.2 Iterators, Insert and Erase

9.2.1. The easiest way to implement `push_back`, `pop_back` and `resize` is to use `erase` and `insert`:

```
void push_back(const T& e)
{
    insert(end(), e);
}

void pop_back() { erase(end() - 1); }

void resize(int n, const T& e = T{})
{
    while (size_ > n)
        pop_back();
    while (size_ < n)
        push_back(e);
}
```

However, it's more efficient to implement these methods directly, by adapting the implementations of `erase` and `insert`. In the case of `push_back` and `pop_back`, the gain in efficiency is minimal. But in the case of `resize`, the difference is significant since this avoids repeated reallocations and the associated copying.

```
void push_back(const T& e)
{
    T* new_buffer = new T[size_ + 1];
    std::copy(cbegin(), cend(), new_buffer);
    new_buffer[size_] = e;
    delete[] buffer_;
    buffer_ = new_buffer;
    ++size_;
}
```

```
void pop_back()
{
    if (size_ == 1) {
        delete [] buffer_;
        buffer_ = nullptr;
        size_ = 0;
    }
    else {
         T* new_buffer = new T[size_ - 1];
         std::copy(cbegin(), cend() - 1,
                   new_buffer);
         delete[] buffer_;
         buffer_ = new_buffer;
         --size_;
    }
}
```

```
void resize(int n, const T& e = T{})
{
    if (n == 0) {
        buffer_ = nullptr;
        size_ = 0;
    }
    else {
        T* new_buffer = new T[n];
        if (n <= size_) {
            std::copy(cbegin(), cbegin() + n,
                      new_buffer);
        }
        else {
            std::copy(cbegin(), cend(),
                      new_buffer);
            std::fill(new_buffer + size_,
                      new_buffer + n,
                      e);
        }
        delete[] buffer_;
        buffer_ = new_buffer;
        size_ = n;
    }
}
```

9.3 Copying Vectors

9.3.6. The revised copy constructor:

```
template <class T>
vector<T>::vector(const vector& v)
{
    if (v.size() == 0) {
        buffer_ = nullptr;
        size_ = 0;
    }
    else {
        buffer_ = new T[v.size()];
        std::copy(v.begin(), v.end(), buffer_);
        size_ = v.size();
    }
    cout << "The copy constructor of class "
         << "vector was executed.\n";
}
```

A test driver:

```
vector<int> f(vector<int> v)
{
    return v;
}

int main()
{
    vector<int> v;
    vector<int> v2 = v;

    f(v);

    return 0;
}
```

9.3.7.

```
void assign(int n, const T& e)
{
    if (n == 0) {
        delete [] buffer_;
        buffer_ = nullptr;
        size_ = 0;
    }
    else {
        T * new_buffer = new T[n];
        std::fill(new_buffer, new_buffer + n,
                  e);
        delete [] buffer_;
        buffer_ = new_buffer;
        size_ = n;
    }
}
```

Chapter 10

Implementation of Linked Lists

10.1 Nodes and Links

There are no exercises in this section.

10.2 Some Basic Methods

10.2.1. The following methods can be implemented in the class:

```
bool empty() const { return (size_ == 0); }
```

```
T& front()
{
    return head_node_->next->element;
}
const T& front() const
{
    return head_node_->next->element;
}
```

The following methods are probably best implemented outside the class:

```
template <class T>
inline void list<T>::swap(list<T>& ls)
{
    std::swap(head_node_, ls.head_node_);
    std::swap(size_, ls.size_);
}
```

```
template <class T>
void list<T>::push_front(
        const T& new_element)
{
    auto first_node = head_node_->next;

    auto new_node =
        new ListNode<T>(new_element,
                        first_node,
                        head_node_);

    head_node_->next = new_node;
    first_node->previous = new_node;
    ++size_;
}

template <class T>
void list<T>::pop_front()
{
    auto first_node = head_node_->next;
    auto new_first_node = first_node->next;

    // modify the list to skip the first node
    head_node_->next = new_first_node;
    new_first_node->previous = head_node_;

    delete first_node;
    --size_;
}
```

```
template <class T>
list<T>::list(int n, const T& e = T()) :
        list<T>()
{
    for (int i = 0; i < n; ++i)
        push_back(e);
}

template <class T>
list<T>::list(
        const std::initializer_list<T>&
        init_list) : list<T>()
{
    for (const T& e : init_list)
        push_back(e);
}

template <class T>
void list<T>::clear()
{
    while (!empty())
        pop_back();
}
```

```
template <class T>
bool operator==(const list<T>& ls1,
                const list<T>& ls2)
{
    if (ls1.size() != ls2.size())
        return false;
    auto node1 = ls1.head_node_->next;
    auto node2 = ls2.head_node_->next;
    while (node1 != ls1.head_node_) {
        if (node1->element != node2->element)
            return false;
        node1 = node1->next;
        node2 = node2->next;
    }
    return true;
}

template <class T>
inline bool operator!=(const list<T>& ls1,
                       const list<T>& ls2)
{
    return !(ls1 == ls2);
}
```

10.2.2.

```
template <class T>
list<T>::list(int n, const T& e = T())
{
    head_node_ = new ListNode<T>;
    auto last_node = head_node_;

    for (int i = 0; i < n; ++i) {
        auto last_node->next =
            new ListNode<T>(e, nullptr,
                            last_node);
        last_node = last_node->next;
    }

    last_node->next = head_node_;
    head_node_->previous = last_node;

    size_ = n;
}
```

10.2.3.

```
template <class T>
list<T>::~list()
{
    auto node = head_node_->next;
    while (node != head_node_) {
        node = node->next;
        delete node->previous;
    }
    delete head_node_;
}
```

10.3 Iterators, Insert and Erase

10.3.3.

```
template <class T>
bool list<T>::operator==(const list<T>& ls2)
{
    if (size() != ls2.size())
        return false;
    auto itr1 = begin();
    auto itr2 = ls2.begin();
    while (itr1 != end()) {
        if (*itr1 != *itr2)
            return false;
        ++itr1;
        ++itr2;
    }
    return true;
}
```

```
template <class T>
void list<T>::remove(const T& e)
{
    auto itr = begin();
    while (itr != end()) {
        if (*itr == e)
            itr = erase(itr);
        else
            ++itr;
    }
}

template <class T>
void list<T>::erase(const_iterator start,
                    const_iterator stop)
{
    auto itr = start;
    while (itr != stop)
        itr = erase(itr);
}
```

The requirement that `reverse` not invalidate any iterators implies that elements cannot be moved. So we'll have to move pointers instead:

```
template <class T>
void list<T>::reverse()
{
    auto node = head_node_;
    do {
        std::swap(node->next, node->previous);
        node = node->previous;
    } while (node != head_node_);
}
```

10.3.4.

```
template <class T>
void list<T>::erase(const_iterator start,
                    const_iterator stop)
{
    auto target_node =
        const_cast<ListNode<T>*>(
            start.current_node);
    auto node_before_start =
        target_node->previous;
    auto stop_node =
        stop.current_node;

    while (target_node != stop_node) {
        target_node->previous = nullptr;
        target_node = target_node->next;
        target_node->previous->next = nullptr;

        delete target_node->previous;
        --size_;
    }

    // link node that preceded start node to
    // stop node
    node_before_start->next = stop_node;
    stop_node->previous = node_before_start;
}
```

10.4 Copying Linked Lists

10.4.1.

```
template <class T>
list<T>::list(const list<T>& ls) : list<T>()
{
    // traverse argument and add elements at end
    // of receiver
    auto last_node = head_node_;
    for (const T& e : ls) {
        auto last_node->next =
            new ListNode<T>(e, nullptr,
                            last_node);
        last_node = last_node->next;
    }

    last_node->next = head_node_;
    head_node_->previous = last_node;
    size_ = ls.size_;
}
```

Chapter 11

Recursion

11.1 The Technique

11.1.3.

```
template <class T>
int count(const T a[], int start, int stop,
          const T& e)
{
    if (start < stop) {
        int count_in_rest =
                count(a, start+1, stop, e);
        if (a[start] == e)
            return  count_in_rest + 1;
        else
            return count_in_rest;
    }
    else {
        return 0;
    }
}
```

11.1.4.

```
template <class T>
const T& max(const T a[], int start, int stop)
{
    if (stop - start > 1) {
        const T& max_in_rest =
            max(a, start+1, stop);
        if (max_in_rest > a[start])
            return max_in_rest;
        else
            return a[start];
    }
    else {
        return a[start];
    }
}
```

11.1.5.

```
template <class T>
int max(const T a[], int start, int stop)
{
    if (stop - start > 1) {
        int i_max_in_rest =
                max(a, start+1, stop);
        if (a[i_max_in_rest] > a[start])
            return i_max_in_rest;
        else
            return start;
    }
    else {
        return start;
    }
}
```

11.1.6.

```
void print(int n)
{
    if (n > 0) {
        cout << ' ' << n;
        print(n-1);
    }
}
```

11.1.7.

```
void print(int n)
{
    if (n > 0) {
        print(n-1);
        cout << ' ' << n;
    }
}
```

11.1.8.

```
void print(int n)
{
    if (n > 1) {
        cout << ' ' << n;
        print(n-1);
        cout << ' ' << n;
    }
    else if (n == 1) {
        cout << ' ' << 1;
    }
}
```

11.2 When to Use Recursion

11.2.2. All of them.

11.3 Tail Recursion

11.3.4.

```
display(a, i, j)
{
    while (j > i)
        print a[i]
        ++i
}
```

11.3.5. The first `print` is the only one of those functions that's tail recursive.

```
void print(int n)
{
    while (n > 0) {
        cout << ' ' << n;
        --n;
    }
}
```

Chapter 12

Sorting

12.1 The Binary Search Algorithm

12.1.1.

```
e = 42
s = [11  27  28  30  36  42  58  65]   middle = 36
                    [36  42  58  65]            58
                    [36  42]                    42
                        [42]
Found!

e = 30
s = [11  27  28  30  36  42  58  65]   middle = 36
    [11  27  28  30]                            28
             [28  30]                           30
                 [30]
Found!
```

12.1.2. Right now, the middle element is the first one of the right half and when $\in$ equals the middle element, we go right. To find the first occurrence of $\in$ use the last element of the left half as middle element and when $\in$ equals the middle element, go left.

12.1.3. In this case, when n is a power of 2, the number of iterations would still be $\log n$. But now each iteration would require comparing two strings of length n and this takes time proportional to n. So the running time of each iteration is now cn instead of a. This implies that the running time of the binary search algorithm is now $cn \log n + b$ instead of $a \log n + b$. When n is large, the term $cn \log n$ dominates so the running time is essentially proportional to $n \log n$ and we say that it is $\Theta(n \log n)$.

Under the same conditions, in the worst case, a sequential search would have to compare every string in the array to given string. Since each comparison takes time cn, those n comparisons would take a total time of cn^2, which is $\Theta(n^2)$. Therefore, a binary search would still run faster than a sequential search.

12.2 Selection Sort

12.2.1.

```
[12  37  25  60  16  42  38]
[12  37  25  38  16  42] 60
[12  37  25  38  16] 42  60
[12  37  25  16] 38  42  60
[12  16  25] 37  38  42  60
[12  16] 25  37  38  42  60
[12] 16  25  37  38  42  60
 12  16  25  37  38  42  60
```

12.2.2.

```
template <class Iterator>
void selection_sort(Iterator start, Iterator stop)
{
    int n = std::distance(start, stop);
    while (n > 1) {
        auto itr_max =
            std::max_element(start, stop);
        std::swap(*itr_max, *(std::prev(stop)));
        --stop;
        --n;
    }
}
```

12.3 Mergesort

12.3.1.

```
[22  37  25  60  16  42  38  46  19]
[22  37  25  60  16][42  38  46  19]
[16  22  25  37  60][19  38  42  46]
[16  19  22  25  37  38  42  46  60]

[22  37  25  60  16  42  38  46  19]
[22  37  25  60  16][42  38  46  19]
[22  37  25][60  16][42  38][46  19]
[22  37][25][60][16][42][38][46][19]
[22][37][25][60][16][42][38][46][19]
[22  37][25][60][16][42][38][46][19]
[22  25  37][16  60][38  42][19  46]
[16  22  25  37  60][19  38  42  46]
[16  19  22  25  37  38  42  46  60]
```

12.3.2.

```
  First array    Second array  Resulting array
[16 22 25 37 60] [19 38 42 46] []
   [22 25 37 60] [19 38 42 46] [16]
   [22 25 37 60]    [38 42 46] [16 19]
      [25 37 60]    [38 42 46] [16 19 22]
         [37 60]    [38 42 46] [16 19 22 25]
            [60]    [38 42 46] [16 19 22 25 37]
            [60]       [42 46] [16 19 22 25 37 38]
            [60]          [46] [16 19 22 25 37 38 42]
            [60]            [] [16 19 22 25 37 38 42 46]
              []          [] [16 19 22 25 37 38 42 46 60]
```

Chapter 13

Error Checking

13.1 Introduction

There are no exercises in this section.

13.2 Exceptions

13.2.5.

```
class TimeError
{
public:
    TimeError(const std::string& d)
        : description(d) {}
    const std::string& what() const
    {
        return description;
    }
private:
    std::string description;
};

void set_hours(int new_hours)
{
    if (new_hours >= 0 && new_hours < 24) {
        hours_ = new_hours;
    }
    else {
        throw TimeError("Invalid argument " +
            "given to Time::set_hours");
    }
}
```

```
void set_minutes(int new_minutes)
{
    if (new_minutes >= 0 && new_minutes < 60) {
        minutes_ = new_minutes;
    }
    else {
        throw TimeError("Invalid argument " +
            "given to Time::set_minutes");
    }
}
```

13.2.6.

```
class DateError
{
public:
    DateError(const std::string& d)
        : description(d) {}
    const std::string& what() const
    {
        return description;
    }
private:
    std::string description;
};
```

```
inline void Date::set(int month0, int day0,
                      int year0)
{
    if (year0 != 0 &&
            month0 > 0 && month0 <= 12 &&
            day0 > 0 && day0 <= 30) {
        month = month0;
        day = day0;
        year = year0;
    }
    else {
        throw DateError("Invalid argument " +
            "given to Date::set");
    }
}

class FractionError
{
public:
    FractionError(const std::string& d)
        : description(d) {}
    const std::string& what() const
    {
        return description;
    }
private:
    std::string description;
};
```

```
inline void Fraction::set(int a0, int b0) {
    if (b0 > 0) {
        a = a0;
        b = b0;
    }
    else {
        throw FractionError("Invalid " +
            "argument given to Fraction::set");
    }
}
```

13.3 STL Containers and Exceptions

There are no exercises in this section.

13.4 Input Validation