Introduction to Data Abstraction, Algorithms and Data Structures

With C++ and the STL

Solutions to Exercises

Spring 2016

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 official solution, 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 for help from another student or the 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

Abstraction

1.1 A Pay Calculator

There are no exercises in this section.

1.2 Design

There are no exercises in this section.

1.3 Names

There are no exercises in this section.

1.4 Implementation

1.4.5.

```
bool is_later_than(const Time & t1,
                   const Time & t2)
{
    return difference(t1, t2) > 0;
}
```

1.4.6.

```
void add_minutes(Time & t, int num_minutes)
{
    int t_in_minutes = t.hours * 60 + t.minutes;
    t_in_minutes += num_minutes;
    t_in_minutes %= 1440;  // 1440 = 24 * 60
    if (t_in_minutes < 0) t_in_minutes += 1440;
    t.hours = t_in_minutes / 60;
    t.minutes = t_in_minutes % 60;
}
```

1.4.7.

```
struct Date { int year, month, day; };

void initialize(Date & d)
{
    d.year = 1;
    d.month = 1;
    d.day = 2000;
}
```

```
void initialize(Date & d,
                int month,
                int day,
                int year)
{
    d.month = month;
    d.day = day;
    d.year = year;
}

void read(Date & d, istream & in)
{
    in >> d.month;
    in.get(); // '/'
    in >> d.day;
    in.get(); // '/'
    in >> d.year;
}

void print(const Date & d, ostream & out)
{
    out << d.month << '/' << d.day << '/'
        << d.year;
}
```

```
void print_in_words(const Date & d,
                    ostream & out)
{
    switch (d.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 << ' ' << d.day << ", " << d.year;
}
```

```
void add_days(Date & d, int num_days)
{
    int d_in_days =
            (d.year - 1) * 360
            + (d.month - 1) * 30
            + (d.day - 1) + num_days;
        // - 1 everywhere because there are 0's
    if (d_in_days >= 0) {
        d.year = d_in_days / 360 + 1;
        d_in_days %= 360;
            // days from start of year
        d.month = d_in_days / 30 + 1;
        d.day = d_in_days % 30 + 1;
    } else { // d_in_days < 0
        d.year = (d_in_days + 1) / 360 - 1;
            // + 1 to make 0 correspond to
            // 12/31/-1
        d_in_days = d_in_days - d.year * 360;
            // days from start of year
    }
    d.month = d_in_days / 30 + 1;
    d.day = d_in_days % 30 + 1;
}
```

1.4.8.

```
struct ThreeDVector { double x, y, z; };

void initialize(ThreeDVector & v)
{
    v.x = v.y = v.z = 0;
}
```

```
void initialize(ThreeDVector & v,
                double x0,
                double y0,
                double z0)
{
    v.x = x0;
    v.y = y0;
    v.z = z0;
}

void read(ThreeDVector & v, istream & in)
{
    char dummy;
    in >> dummy; // '('.
        // >> used instead of get() so
        // whitespace is skipped.
    in >> v.x;
    in >> dummy; // ','
    in >> v.y;
    in >> dummy; // ','
    in >> v.z;
    in >> dummy; // ')'
}

void print(const ThreeDVector & d,
           ostream & out)
{
    out << '(' << d.x << ", "<< d.y << ", "
        << d.z << ')';
}
```

```
ThreeDVector add(const ThreeDVector & v1,
                 const ThreeDVector & v2)
{
    ThreeDVector result;
    initialize(result, v1.x + v2.x, v1.y + v2.y,
               v1.z + v2.z);
    return result;
}
```

1.4.9.

```
struct Fraction { int a, b; };

void initialize(Fraction & r)
{
    r.a = 0;
    r.b = 1;
}

void initialize(Fraction & r, int a0)
{
    r.a = a0;
    r.b = 1;
}

void initialize(Fraction & r, int a0, int b0)
{
    r.a = a0;
    r.b = b0;
}
```

```
void read(Fraction & r, istream & in)
{
    in >> r.a;
    in.get(); // '/'
    in >> r.b;
}

void print(const Fraction & r, ostream & out)
{
    out << r.a << '/' << r.b;
}

void print_mixed(const Fraction & r,
                 ostream & out)
{
    int n;
    int a = r.a;
    int b = r.b;

    if (b < 0) { a = -a; b = -b; }

    n = a / b;
    a = a % b;

    out << n;
    if (a > 0)
        out << ' ' << a << '/' << b;
    else if (a < 0)
        out << ' ' << -a << '/' << b;
    // else (a == 0), don't print fraction
}
```

```
Fraction add(const Fraction & r1,
             const Fraction & r2)
{
    Fraction result;
    initialize(result,
               r1.a * r2.b + r2.a * r1.b,
               r1.b * r2.b);
    return result;
}

Fraction multiply(const Fraction & r1,
                  const Fraction & r2)
{
    Fraction result;
    initialize(result, r1.a * r2.a,
               r1.b * r2.b);
    return result;
}
```

1.4.10. Here is the code that was revised or created for each part.

a)

```
struct Time
{
    int hours, minutes;
    bool pm;
};

void initialize(Time & t)
{
    t.hours = t.minutes = 99;
    t.pm = false;
}
```

```
void read(Time & t, istream & in)
{
    in >> t.hours;
    in.get(); // colon
    in >> t.minutes;
    char c;
    in >> c;
    t.pm = (c == 'p');
    cin.ignore(3);
}

void print(const Time & t, ostream & out)
{
    out << t.hours << ':';
    if ( t.minutes < 10 ) out << '0';
    out << t.minutes << ' ';
    if (t.pm)
        out << "p.m.";
    else
        out << "a.m.";
}

double convert_to_hours(const Time & t)
{
    double hours = t.hours + t.minutes/60.0;
    if (t.hours == 12 && !t.pm)
        hours -= 12;
    else if (t.pm && t.hours < 12)
        hours += 12;
    return hours;
}
```

```
double difference(const Time & t1,
                  const Time & t2)
{
    return convert_to_hours(t1) -
                convert_to_hours(t2);
}
```

b)

```
const float kOvertimeFactor = 1.5;

double compute_pay(const Time & start,
                   const Time & stop)
{
    double hours_worked =
                difference(stop, start);
    if (hours_worked <= 8)
        return hours_worked * kPayRate;
    else
        return (8 + (hours_worked - 8)
                  * kOvertimeFactor)
               * kPayRate;
}

int main()
{
        ...
        double pay = compute_pay(
                          start_time,
                          stop_time);
        ...
}
```

1.5 Modularity and Abstraction

There are no exercises in this section.

Chapter 2

Data Abstraction

2.1 Introduction

There are no exercises in this section.

2.2 Classes to Enforce Data Abstraction

2.2.3. Here's the code that was revised or created for each part.

a)

```
class Time
{
    ...

private:
    int minutes;
        // total number of minutes since
        // midnight
};
```

```
void initialize(Time & t)
{
    t.minutes = -1;
}

void read(Time & t, istream & in)
{
    int hours = -1;
    int minutes = -1;
    in >> hours;
    in.get(); // colon
    in >> minutes;
    t.minutes = hours*60 + minutes;
}

void print(const Time & t, ostream & out)
{
    out << t.minutes/60 << ':';
    int minutes = t.minutes%60;
    if (minutes < 10) out << '0';
    out << minutes;
}

double difference(const Time & t1,
                  const Time & t2)
{
    return (t1.minutes - t2.minutes)/60.0;
}
```

b)

```
class Time
{
    ...

private:
    int hours, minutes, seconds;
};

void initialize(Time & t)
{
    t.hours = t.minutes = t.seconds = 99;
}

void read(Time & t, istream & in)
{
    in >> t.hours;
    in.get(); // colon
    in >> t.minutes;
    in.get(); // colon
    in >> t.seconds;
}

void print(const Time & t, ostream & out)
{
    out << t.hours << ':';
    if (t.minutes < 10) out << '0';
    out << t.minutes << ':';
    if (t.seconds < 10) out << '0';
    out << t.seconds;
}
```

```
double difference(const Time & t1,
                  const Time & t2)
{
    return (t1.hours + t1.minutes/60.0
                + t1.seconds/3600.0)
           - (t2.hours + t2.minutes/60.0
                + t2.seconds/3600.0);
}
```

2.3 Classes to Support Object-Oriented Programming

2.3.4.

```
bool is_later_than(const Time & t2)
{
    return minus(t2) > 0;
}
```

2.3.5.

```
void add_minutes(int num_minutes)
{
    int t_in_minutes = hours * 60 + minutes;
    t_in_minutes += num_minutes;
    t_in_minutes %= 1440;  // 1440 = 24 * 60
    if (t_in_minutes < 0) t_in_minutes += 1440;
    hours = t_in_minutes / 60;
    minutes = t_in_minutes % 60;
}
```

2.3.6.

```
class Date
{
public:
    void initialize()
    {
        initialize(1, 1, 2000);
    }

    void initialize(int month0, int day0,
                    int year0)
    {
        month = month0;
        day = day0;
        year = year0;
    }

    void read(istream & in)
    {
        in >> month;
        in.get(); // '/'
        in >> day;
        in.get(); // '/'
        in >> year;
    }

    void print(ostream & out)
    {
        out << month << '/' << day << '/'
            << year;
    }
```

```
void print_in_words(ostream & out)
{
    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;
}
```

```
    void add_days(int num_days)
    {
        int d_in_days =
                (year − 1) * 360
                + (month − 1) * 30
                + (day − 1) + num_days;
            // − 1 everywhere because there are
            // 0's
        if (d_in_days >= 0) {
            year = d_in_days / 360 + 1;
            d_in_days %= 360;
                // days from start of year
            month = d_in_days / 30 + 1;
            day = d_in_days % 30 + 1;
        } else { // d_in_days < 0
            year = (d_in_days + 1) / 360 − 1;
                // + 1 to make 0 correspond to
                // 12/31/−1
            d_in_days = d_in_days − year * 360;
                // days from start of year
        }
        month = d_in_days / 30 + 1;
        day = d_in_days % 30 + 1;
    }

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

2.3.7.

```
class ThreeDVector
{
public:
    void initialize() { x = y = z = 0; }

    void initialize(double x0, double y0,
                    double z0)
    {
        x = x0;
        y = y0;
        z = z0;
    }

    void read(istream & in)
    {
        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)
    {
        out << '(' << x << ", "<< y << ", "
            << z << ')';
    }

    ThreeDVector add(const ThreeDVector & v2)
    {
        ThreeDVector result;
        result.initialize(x + v2.x, y + v2.y,
                          z + v2.z);
        return result;
    }

private:
    double x, y, z;
};
```

2.3.8.

```
class Fraction
{
public:
    void initialize()
    {
        a = 0;
        b = 1;
    }
```

```
void initialize(int a0)
{
    a = a0;
    b = 1;
}

void initialize(int a0, int b0)
{
    a = a0;
    b = b0;
}

void read(istream & in)
{
    in >> a;
    in.get(); // '/'
    in >> b;
}

void print(ostream & out)
{
    out << a << '/' << b;
}
```

```
void print_mixed(ostream & out)
{
    int n;
    int a1 = a;
    int b1 = b;

    if (b1 < 0) { a1 = -a1; b1 = -b1; }

    n = a1 / b1;
    a1 = a1 % b1;

    out << n;
    if (a1 > 0)
        out << ' ' << a1 << '/' << b1;
    else if (a1 < 0)
        out << ' ' << -a1 << '/' << b1;
    // else (a1 == 0), don't print fraction
}

Fraction add(const Fraction & r2)
{
    Fraction result;
    result.initialize(a * r2.b + r2.a * b,
                      b * r2.b);
    return result;
}
```

```
    Fraction multiply(const Fraction & r2)
    {
        Fraction result;
        result.initialize(a * r2.a, b * r2.b);
        return result;
    }

private:
    int a, b;
};
```

2.4 Constant Methods

2.4.2. The method `is_later_than` should be declared constant.

2.4.3. The `print` and `print_in_words` methods of class `Date`, the `print` and `add` methods of class `ThreeDVector`, and the `print`, `print_mixed`, `add` and `multiply` methods of class `Fraction`.

2.5 Inline Methods

2.5.3.

```
class Date
{
public:
    void initialize() {
        year = month = day = 99;
    }
```

```
    void initialize(int month0, int day0,
                    int year0);

    void read(istream & in);
    void print(ostream & out) const {
        out << month << '/' << day << '/'
            << year;
    }
    void print_in_words(ostream & out) const;

    void add_days(int num_days);

private:
    int year, month, day;
};

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

```
inline void Date::read(istream & in)
{
    in >> month;
    in.get(); // '/'
    in >> day;
    in.get(); // '/'
    in >> 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;
}
```

```
void Date::add_days(int num_days)
{
    int d_in_days = (year - 1) * 360
                  + (month - 1) * 30
                  + (day - 1) + num_days;
        // - 1 everywhere because there are 0's
    if (d_in_days >= 0) {
        year = d_in_days / 360 + 1;
        d_in_days %= 360;
            // days from start of year
        month = d_in_days / 30 + 1;
        day = d_in_days % 30 + 1;
    } else { // d_in_days < 0
        year = (d_in_days + 1) / 360 - 1;
            // + 1 to make 0 correspond to
            // 12/31/-1
        d_in_days = d_in_days - year * 360;
            // days from start of year
    }
    month = d_in_days / 30 + 1;
    day = d_in_days % 30 + 1;
}

class ThreeDVector
{
public:
    void initialize() { x = y = z = 0; }
    void initialize(double x0, double y0,
                    double z0);

    void read(istream & in);
```

```
    void print(ostream & out) const {
        out << '(' << x << ", "<< y << ", "
            << z << ')';
    }

    ThreeDVector add(
        const ThreeDVector & v2) const;

private:
    double x, y, z;
};

inline void ThreeDVector::initialize(
        double x0, double y0, double z0)
{
    x = x0;
    y = y0;
    z = z0;
}
```

```
inline void ThreeDVector::read(istream & in)
{
    char dummy;
    in >> dummy; // '('.
        // >> used instead of get() so
        // whitespace is skipped.
    in >> x;
    in >> dummy; // ','
    in >> y;
    in >> dummy; // ','
    in >> z;
    in >> dummy; // ')'
}

inline ThreeDVector ThreeDVector::add(
        const ThreeDVector & v2) const
{
    ThreeDVector result;
    result.initialize(x + v2.x, y + v2.y,
                      z + v2.z);
    return result;
}

class Fraction
{
public:
    void initialize();
    void initialize(int a0);
    void initialize(int a0, int b0);

    void read(istream & in);
```

```
    void print(ostream & out) const {
        out << a << '/' << b;
    }
    void print_mixed(ostream & out) const;

    Fraction add(const Fraction & r2) const;
    Fraction multiply(const Fraction & r2) const;

private:
    int a, b;
};

inline void Fraction::initialize()
{
    a = 0;
    b = 1;
}

inline void Fraction::initialize(int a0)
{
    a = a0;
    b = 1;
}

inline void Fraction::initialize(int a0,
                                 int b0)
{
    a = a0;
    b = b0;
}
```

```
inline void Fraction::read(istream & in)
{
    in >> a;
    in.get(); // '/'
    in >> b;
}

void Fraction::print_mixed(
        ostream & out) const
{
    int n;
    int a1 = a;
    int b1 = b;

    if (b1 < 0) { a1 = -a1; b1 = -b1; }

    n = a1 / b1;
    a1 = a1 % b1;

    out << n;
    if (a1 > 0)
        out << ' ' << a1 << '/' << b1;
    else if (a1 < 0)
        out << ' ' << -a1 << '/' << b1;
    // else (a1 == 0), don't print fraction
}
```

```
inline Fraction Fraction::add(
        const Fraction & r2) const
{
    Fraction result;
    result.initialize(a * r2.b + r2.a * b,
                      b * r2.b);
    return result;
}

inline Fraction Fraction::multiply(
        const Fraction & r2) const
{
    Fraction result;
    result.initialize(a * r2.a, b * r2.b);
    return result;
}
```

2.6 Constructors

2.6.13.

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

    ...
};
```

```
inline Date::Date(int month0, int day0,
                  int year0 ) :
    month(month0), day(day0), year(year0) {}


class ThreeDVector {
public:
    ThreeDVector(): ThreeDVector(0, 0, 0) {}
    ThreeDVector(double x0, double y0,
                 double z0);

    ThreeDVector add(
            const ThreeDVector & v2) const
    {
        return ThreeDVector(x + v2.x, y + v2.y,
                            z + v2.z);
    }

    ...
};

inline ThreeDVector::ThreeDVector(double x0,
                                  double y0,
                                  double z0) :
    x(x0), y(y0), z(z0) {}


class Fraction
{
public:
    Fraction() : Fraction(0) {}
    Fraction(int a0) : Fraction(a0, 1) {}
    Fraction(int a0, int b0) : a(a0), b(b0) {}
```

```
    Fraction add(const Fraction & r2) const
    {
        return Fraction(a * r2.b + r2.a * b,
                        b * r2.b);
    }
    Fraction multiply(
            const Fraction & r2) const
    {
        return Fraction(a * r2.a, b * r2.b);
    }

    ...
};
```

2.7 Get and Set Methods

2.7.3.

```
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.";
}
```

2.7.4.

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

    void read(istream & in);
    void print(ostream & out) const;

    double minus(const Time & t2) 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_;
};

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

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

```
inline double Time::minus(
        const Time & t2) const
{
    return (total_minutes_ - t2.total_minutes_)
                / 60.0;
}

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_minut
{
    total_minutes_ = new_hours * 60
                          + new_minutes;
}
```

2.7.5.

```
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;
}
```

2.7.6.

```
class ThreeDVector
{
public:
    ...

    void read(istream & in);

    void print(ostream & out) const
    {
        out << '(' << x() << ", "<< y() << ", "
            << z() << ')';
    }

    void set(double x0, double y0, double z0);
    double x() const { return x_; }
    double y() const { return y_; }
    double z() const { return z_; }
```

```
    ThreeDVector add(
            const ThreeDVector & v2) const
    {
        return ThreeDVector(x() + v2.x(),
                            y() + v2.y(),
                            z() + v2.z());
    }

private:
    double x_, y_, z_;
};

inline ThreeDVector::ThreeDVector(double x0,
                                  double y0,
                                  double z0) :
    x_(x0), y_(y0), z_(z0) {}

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

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

2.7.7.

```
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;
}
```

2.8 Operators

2.8.5. It's better to put it outside so we can have implicit conversion on both operands.

```
inline bool operator<(const Time & t1,
                      const Time & t2)
{
    return t1 - t2 < 0;
}
```

2.8.6.

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

2.8.7.

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

    void operator+=(int num_days);

    ...
};
```

```
inline istream & operator>>(istream & in,
                            Date & d)
{
    in >> d.month;
    in.get(); // '/'
    in >> d.day;
    in.get(); // '/'
    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
}
```

2.8.8.

```
class ThreeDVector
{
public:
    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 get() so
        // whitespace is skipped.
    double x;
    in >> x;
    in >> dummy; // ','
    double y;
    in >> y;
    in >> dummy; // ','
    double z;
    in >> z;
    in >> dummy; // ')'
    v.set(x, y, z);
    return in;
}

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

2.8.9.

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

2.8.10.

```
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();
}
```

2.9 Compiling Large Programs

2.9.3. Here's what `Date.h` looks like:

```
// 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() { year = month = day = 99; }

    Date(int month0, int day0, int year0);

    void print_in_words(
        std::ostream & out) const;
    void set(int month0, int day0, int year0);

    void operator+=(int num_days);

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

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

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

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

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

#endif
```

Here's `Date.cpp`:

```
// Date.cpp

#include "Date.h"

#include <iostream>
using std::istream;
using std::ostream;

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

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

Here's a test driver:

```
// test_Date.cpp

#include <iostream>
using std::cout;
using std::endl;

#include "Date.h"

int main()
{
    Date date;
    cout << date << endl;
    date.set(1,22,2015);
    cout << date << endl;
    date += 9;
    cout << date << endl;

    return 0;
}
```

The source code of the classes `ThreeDVector` and `Fraction` can be reorganized in a very similar way.

2.10 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 << '\n';
}
```

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;
    while (i < n && source[i] != '\0') {
        // i equals the number of chars copied
        // so far
        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 << '\n';
}
```

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
```

3.3 I/O Streams

3.3.1.

```
cout << "Testing output operator.  Done.\n";
cout << "Writing a single 'x' to the screen: ";
cout.put('x');
cout << endl;
```

```
cout << "Enter an integer: ";
int n = 0;
cin >> n;
cout << "Here it is: " << n << endl;
cin.get(); // to flush out '\n'

cout << "Enter three characters: ";
cout << "First one: " << char(cin.get())
     << endl;
char c = ' ';
cin.get(c);
cout << "Second one: " << c << endl;
cout << "Third one: " << char(cin.peek())
     << endl;
cout << "Third one again: " << char(cin.get())
     << endl;
cin.putback(c);
cout << "Second one again: " << char(cin.get())
     << endl;

cout << "Create a file input.txt.\n"
     << "Write 53 on its first line.\n"
     << "Write abc on the second.\n";
ifstream in("input.txt");
ofstream out;
out.open("output.txt");
in >> n;
out << n << endl;
out.close();
cout << "File output.txt should contain 53.\n";

cout << "Input stream state should be good: "
     << (in.good() ? "yes" : "no") << endl;
```

```
in >> n;
cout << "Input stream state should be fail: "
     << (in.fail() ? "yes" : "no") << endl;

in.clear();
string s;
in >> s;
cout << "Input stream state should be good: "
     << (in.good() ? "yes" : "no") << endl;
in >> n;
cout << "Input stream state should be eof: "
     << (in.eof() ? "yes" : "no") << endl;
cout << "Input stream state should be fail: "
     << (in.fail() ? "yes" : "no") << endl;
```

3.3.2.

```
void read(char cs[], int n, istream & in)
{
    int i = 0;
    while (i < n - 1) {
        char c = in.peek();
        if (in.eof()) break;
        if (c == '\n') break;
        in.get(cs[i]);
        ++i;
    }
    cs[i] = '\0';
}
```

```
void readln(char cs[], int n, istream & in)
{
    int i = 0;
    while (i < n - 1) {
        char c = in.get();
        if (in.eof()) break;
        if (c == '\n') break;
        cs[i] = c;
        ++i;
    }
    cs[i] = '\0';
}
```

3.4 String Streams

3.4.1.

```
string line;
getline(cin, line);

istringstream iss(line);
int x;
for (int i = 0; i < 3; ++i) iss >> x;

if (!iss) {
    cout << "The line did not contain three "
         << "integers.";
} else {
    char c;
    iss >> c;
    if (iss)
        cout << "The line contained more than "
             << "just three integers.";
    else
        cout << "The line contained three "
             << "integers and nothing else.";
}
cout << endl;
```

Chapter 4

Error Checking

4.1 Introduction

4.1.3.

```
inline bool Date::is_valid() const
{
    if (year != 0 && month > 0 && month <= 12 &&
            day > 0 && day <= 30)
        return true;
    else
        return false;
}
```

```
inline bool Fraction::is_valid() const
{
    if (b > 0)
        return true;
    else
        return false;
}
```

4.2 Exceptions

4.2.5.

```
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");
    }
}
```

4.2.6.

```
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");
    }
}
```

4.3 Input Validation

Chapter 5

Vectors

5.1 A Simple File Viewer

There are no exercises in this section.

5.2 Vectors in the STL

5.2.6.

```
vector<int> v0;
cout << "An empty vector of integers: ";
for (int i = 0; i < v0.size(); ++i)
    cout << v0[i] << ' ';
cout << '\n';
```

```
vector<int> v1(3);
cout << "A vector with three (random) "
     << "integers: ";
for (int i = 0; i < v1.size(); ++i)
    cout << v1[i] << ' ';
cout << '\n';

vector<string> v2(3);
cout << "A vector with three empty strings: ";
for (int i = 0; i < v2.size(); ++i)
    cout << "'" << v2[i] << "' ";
cout << '\n';

vector<int> v3(3, 17);
cout << "A vector with three 17's: ";
for (int i = 0; i < v3.size(); ++i)
    cout << v3[i] << ' ';
cout << '\n';

vector<int> v4(v3);
cout << "A copy of the previous vector: ";
for (int i = 0; i < v4.size(); ++i)
    cout << v4[i] << ' ';
cout << '\n';

v4.front() = 1;
v4.back() = 23;
cout << "The last vector with its first and "
     << "last elements changed to 1 and 23: ";
for (int i = 0; i < v4.size(); ++i)
    cout << v4[i] << ' ';
cout << '\n';
```

```
    v4.resize(2);
    cout << "The last vector shrunk to size 2: ";
    for (int i = 0; i < v4.size(); ++i)
        cout << v4[i] << ' ';
    cout << '\n';

    v4.resize(4);
    cout << "The last vector grown to size 4: ";
    for (int i = 0; i < v4.size(); ++i)
        cout << v4[i] << ' ';
    cout << '\n';

    v4.resize(6, 42);
    cout << "The last vector grown to size 6 and "
         << "padded with 42's: ";
    for (int i = 0; i < v4.size(); ++i)
        cout << v4[i] << ' ';
    cout << '\n';

    v4.push_back(60);
    cout << "The last vector with a 60 added to "
         << "its back: ";
    for (int i = 0; i < v4.size(); ++i)
        cout << v4[i] << ' ';
    cout << '\n';

    v4.pop_back();
    cout << "The last vector with its back element "
         << "removed: ";
    for (int i = 0; i < v4.size(); ++i)
        cout << v4[i] << ' ';
    cout << '\n';
```

```
vector<int> v5;
vector<int> v6;
v5 = v6 = v4;
cout << "Two new copies of the last vector:\n";
for (int i = 0; i < v5.size(); ++i)
    cout << v5[i] << ' ';
cout << '\n';
for (int i = 0; i < v6.size(); ++i)
    cout << v6[i] << ' ';
cout << '\n';

v6.clear();
cout << "The last vector with all its elements "
     << "removed: ";
for (int i = 0; i < v6.size(); ++i)
    cout << v6[i] << ' ';
cout << '\n';

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

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

```
cout << "The maximum size of the last vector: "
     << v6.max_size() << '\n';

v6.assign(5, 12);
cout << "The last vector with five 12's: ";
for (int i = 0; i < v6.size(); ++i)
    cout << v6[i] << ' ';
cout << '\n';

cout << "The last two vectors:\n";
for (int i = 0; i < v5.size(); ++i)
    cout << v5[i] << ' ';
cout << '\n';
for (int i = 0; i < v6.size(); ++i)
    cout << v6[i] << ' ';
cout << '\n';
v5.swap(v6);
cout << "The same vectors with their contents "
     << "swapped:\n";
for (int i = 0; i < v5.size(); ++i)
    cout << v5[i] << ' ';
cout << '\n';
for (int i = 0; i < v6.size(); ++i)
    cout << v6[i] << ' ';
cout << '\n';
```

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

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

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] << '\n';
}
```

5.3 Design and Implementation of the File Viewer

5.4 Vectors and Exceptions

There are no exercises in this section.

5.5 Arrays

Chapter 6

Generic Algorithms

6.1 Introduction

6.1.3.

```
cout << "4 = " << max(3, 4) << '\n';
cout << "bob = "
     << max(string("alice"), string("bob"))
     << '\n';

vector<int> v1 = {37, 12, 23, 37, 60, 37};
cout << "3 = "
     << count(v1.begin(), v1.end(), 37) << '\n';
cout << "0 = "
     << count(v1.begin(), v1.end(), 5) << '\n';
sort(v1.begin(), v1.end());
for (int x : v1) cout << x << ' ';
cout << '\n';
```

```
vector<string> v2 = {"bob", "alice", "charlie",
                     "erik", "diane"};
cout << "1 = "
     << count(v2.begin(), v2.end(), "alice")
     << '\n';
cout << "0 = "
     << count(v2.begin(), v2.end(), "john")
     << '\n';
sort(v2.begin(), v2.end());
for (const string & s : v2) cout << s << ' ';
cout << '\n';
```

In this code, when `max` is called with string arguments, we first convert the literal strings into C++ strings because the operator < does not work properly on literal strings (or C strings). An alternative would have been to tell `max` to expect `string` arguments:

```
max<string>("alice", "bob")
```

This would force the implicit conversion of the literal strings into C++ strings.

6.2 Iterators

6.2.7.

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);
```

6.2.8.

```
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;
}
```

6.3 Iterator Types and Categories

6.4 Vectors and Iterators

6.5 Algorithms in the STL

6.5.4. 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`.

6.6 Implementing Generic Algorithms

6.6.4.

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

template <class Iterator>
Iterator max_element(Iterator start,
                     Iterator stop)
{
    if (start == stop) return stop;
    Iterator itr_max_so_far = start;
    for (Iterator itr = start;
            itr != stop;
            ++itr)
        if (*itr_max_so_far < *itr)
            itr_max_so_far = itr;
    return itr_max_so_far;
}
```

```
template <class Itr1, class Itr2>
Itr2 copy(Itr1 start1,
          Itr1 stop1,
          Itr2 start2)
{
    Itr2 itr2 = start2;
    for (Itr1 itr1 = start1; itr1 != stop1; ++itr
        *itr2 = *itr1;
        ++itr2;
    }
    return itr2;
}

template <class Itr1, class Itr2>
Itr2 copy_backward(Itr1 start1,
                   Itr1 stop1,
                   Itr2 stop2)
{
    Itr2 itr2 = stop2;
    Itr1 itr1 = stop2;
    while (itr1 != start1) {
        --itr1;
        --itr2;
        *itr2 = *itr1;
    }
    return itr2;
}
```

```
template <class Iterator>
void reverse(Iterator start, Iterator stop)
{
    Iterator first = start;
    Iterator second = stop;
    --second;
    while (first != second) {
        swap(*first, *second);
        ++first;
        if (first != second) --second;
    }
}
```

6.7 Initializer Lists

6.7.2.

```
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;
    }
}
```

6.8 Function as Arguments

6.8.2.

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

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

6.8.3.

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

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

6.8.4.

```
template <class Iterator, class 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 <class Iterator,
          class UnaryPredicate,
          class 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 <class Iterator, class 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 = start;
            itr != stop;
            ++itr)
        if (less_than(*itr_max_so_far, *itr))
            itr_max_so_far = itr;
    return itr_max_so_far;
}

template <class SourceItr, class DestItr,
          class 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 <class Iterator, class UnaryPredicate>
UnaryPredicate for_each(Iterator start,
                        Iterator stop,
                        UnaryPredicate f)
{
    for (Iterator itr = start;
            itr != stop;
            ++itr) {
        f(*itr);
    }
    return f;
}
```

6.9 Function Objects

6.9.2.

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

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

Chapter 7

Linked Lists

7.1 A Simple Text Editor

There are no exercises in this section.

7.2 Vector Version of the Text Editor

7.3 Vectors and Linked Lists

There are no exercises in this section.

7.4 Linked Lists in the STL

7.4.3.

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

7.4.4.

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

7.4.5.

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

7.4.6.

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

7.4.7.

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

7.5 List Version of the Text Editor

Chapter 8

Maps

8.1 A Phone Book

There are no exercises in this section.

8.2 Maps in the STL

8.2.6.

```
map<string, int> m_ages;
ifstream ifs_ages("ages.txt");
string name;
int age;
while (getline(ifs_ages, name)) {
    ifs_ages >> age;
    ifs_ages.get();
    m_ages[name] = age;
}
```

```
for (const auto & p : m_ages)
    cout << p.first << ": " << p.second << '\n';

for (const auto & p : m_ages)
    if (p.second < 21) cout << p.first << '\n';
```

8.2.7.

```
map<int, double> m_prices;
ifstream ifs_prices("prices.txt");
int number;
double price;
while (ifs_prices >> number) {
    ifs_prices >> price;
    m_prices[number] = price;
}

for (const auto & p : m_prices)
    if (p.second < 1) cout << p.first << ' ';
cout << '\n';

map<int, double> m_cheap;
for (const auto & p : m_prices)
    if (p.second < 1)
        m_cheap.insert(m_cheap.end(), p);
```

8.3 Design and Implementation of the Phone Book

Chapter 9

Object-Oriented Design

There are no exercises in this chapter.

Chapter 10

Dynamically Allocated Arrays

10.1 The Size of Ordinary Arrays

There are no exercises in this section.

10.2 The Dynamic Allocation of Arrays

There are no exercises in this section.

10.3 Programming with Dynamically Allocated Arrays

10.3.5.

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

10.3.6.

```
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;
}
```

10.3.7.

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 11

Implementation of Vectors

11.1 A Basic Class of Vectors

11.1.5. The constructor and the methods `empty`, `back` and `swap` can be implemented in the class:

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

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

```
Vector(int n, const T & e)
{
    buffer_ = get_new_buffer(n);
    std::fill(buffer_, buffer_ + n, e);
    size_ = n;
}

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

But, to take advantage of implicit conversions on both sides, it is 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);
}
```

11.2 Growing and Shrinking Vectors

11.2.1. 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 is 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 = get_new_buffer(size_ + 1);
    std::copy(cbegin(), cend(), new_buffer);
    new_buffer[size_] = e;
    delete [] buffer_;
    buffer_ = new_buffer;
    ++size_;
}

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

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

11.3 Destroying and Copying Vectors

11.3.7. The revised destructor:

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

A test driver:

```
int main() { Vector<int> v; }
```

11.3.8. The revised copy constructor:

```
template <class T>
Vector<T>::Vector( const Vector & v )
{
    buffer_ = get_new_buffer( 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;
}
```

11.3.9.

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

void assign(int n, const T & e)
{
    T * new_buffer = get_new_buffer(n);
    std::fill(new_buffer, new_buffer + n, e);

    // deallocate old buffer
    delete [] buffer_;

    // give new buffer to receiver
    buffer_ = new_buffer;
    size_ = n;
}
```

11.4 Growing and Shrinking Vectors Efficiently

Chapter 12

Implementation of Linked Lists

12.1 Nodes and Links

There are no exercises in this section.

12.2 Some Basic Methods

12.2.1. Here's the implementation of the `size` method:

```
int size() const { return size_; }
```

This assumes that a new private data member was added to the class:

```
int size_;
```

In addition, `size_` should be set to 0 in the default constructor, incremented by 1 in `push_back` and decremented by 1 in `pop_back`.

The following methods can be implemented in the class:

```
T & front() {
    return p_head_node->next->element;
}
const T & front() const {
    return p_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(p_head_node, ls.p_head_node);
    std::swap(size_, ls.size_);
}
```

```
template <class T>
inline void List<T>::push_front(
        const T & new_element)
{
    // set a pointer to the first node
    ListNode<T> * p_first_node =
        p_head_node->next;

    // create new node and set its contents
    ListNode<T> * p_new_node =
        new ListNode<T>(new_element,
                        p_first_node,
                        p_head_node);

    // finish linking new node to list
    p_head_node->next = p_new_node;
    p_first_node->previous = p_new_node;

    ++size_;
}
```

```
template <class T>
inline void List<T>::pop_front()
{
    // set pointers to first node and node that
    // will become first
    ListNode<T> * p_first_node =
        p_head_node->next;
    ListNode<T> * p_new_first_node =
        p_first_node->next;

    // modify the list to skip the first node
    p_head_node->next = p_new_first_node;
    p_new_first_node->previous = p_head_node;

    // deallocate first node
    p_first_node->next =
        p_first_node->previous = nullptr;
    delete p_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;
    ListNode<T> * p1 = ls1.p_head_node->next;
    ListNode<T> * p2 = ls2.p_head_node->next;
    while (p1 != ls1.p_head_node) {
        if (p1->element != p2->element)
            return false;
        p1 = p1->next;
        p2 = p2->next;
    }
    return true;
}
```

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

12.2.2.

```
template <class T>
List<T>::List(int n, const T & e = T())
{
    p_head_node = new ListNode<T>;
    ListNode<T> * p_last_node = p_head_node;

    for (int i = 0; i < n; ++i) {
        // create new node and set its element
        // and previous ptr
        ListNode<T> * p_new_node =
            new ListNode<T>;
        p_new_node->element = e;
        p_new_node->previous = p_last_node;

        // finish linking new node to last node
        p_last_node->next = p_new_node;

        p_last_node = p_new_node;
    }

    // link last node to head node
    p_last_node->next = p_head_node;
    p_head_node->previous = p_last_node;

    size_ = n;
}
```

12.3 Iterators, Insert and Erase

12.3.3.

```
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)
{
    const_iterator 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()
{
    ListNode<T> * p = p_head_node;
    do {
        std::swap(p->next, p->previous);
        p = p->previous;
    } while (p != p_head_node);
}
```

12.3.4.

```
template <class T>
void List<T>::erase(const_iterator start,
                    const_iterator stop)
{
    ListNode<T> * p_target_node =
        (ListNode<T> * )(start.p_current_node);
    ListNode<T> * p_node_before_start =
        p_target_node->previous;
    while (p_target_node != stop.p_current_node)
        p_target_node->previous = nullptr;
        p_target_node = p_target_node->next;

        // set next pointer of previous target
        // node to nullptr
        p_target_node->previous->next = nullptr;

        delete p_target_node->previous;
        --size_;
    }
    // link node that preceded start node to
    // stop node
    p_node_before_start->next = p_target_node;
    p_target_node->previous =
        p_node_before_start;
}
```

12.4 Destroying and Copying Linked Lists

12.4.1.

```
template <class T>
List<T>::~List()
{
    while (!empty()) pop_back();
    delete p_head_node;
}
```

12.4.2.

```
template <class T>
List<T>::~List()
{
    ListNode<T> * p = p_head_node->next;
    while (p != p_head_node) {
        p = p->next;
        delete p->previous;
    }
    delete p_head_node;
}
```

12.4.3.

```
template <class T>
List<T>::List(const List<T> & ls) : List<T>()
{
    // traverse argument and add elements at end
    // of receiver
    ListNode<T> * p_last_node = p_head_node;
    for (const T & e : ls) {
        ListNode<T> * p_new_node =
            new ListNode<T>;
        p_new_node->element = e;
        p_new_node->previous = p_last_node;
        p_last_node->next = p_new_node;

        p_last_node = p_new_node;
    }

    // link last node to head node
    p_last_node->next = p_head_node;
    p_head_node->previous = p_last_node;

    size_ = ls.size_;
}
```

Chapter 13

Analysis of Algorithms

13.1 Introduction

There are no exercises in this section.

13.2 Measuring Exact Running Times

There are no exercises in this section.

13.3 Analysis

There are no exercises in this section.

13.4 Asymptotic Running Times

13.4.6.

a) For the lower bound, it's always true that $n + 10 \geq n$. On the other hand, for the upper bound, $n + 10 \leq n + 10n = 11n$ if $n \geq 1$. Therefore, we get that $an \leq n + 10 \leq bn$ for every $n \geq n_0$ by setting $a = 1$, $b = 11$ and $n_0 = 1$.

Another solution comes from noticing that if $n \geq 10$, then $n + 10 \leq n + n = 2n$. Therefore, we also get that $an \leq n + 10 \leq bn$ for every $n \geq n_0$ by setting $a = 1$, $b = 2$ and $n_0 = 10$.

b) We have that $n^2 + n \geq n^2$ if $n \geq 0$. On the other hand, $n^2 + n \leq n^2 + n^2 = 2n^2$ if $n \geq 1$. Therefore, $an^2 \leq n^2 + n \leq bn^2$ for every $n \geq n_0$ when $a = 1$, $b = 2$ and $n_0 = 1$.

c) If $n \geq 0$, then $3n^2 - n \leq 3n^2$. On the other hand, if $n \geq 1$, $3n^2 - n \geq 2n^2$ because $n^2 \geq n$. Therefore, $an^2 \leq 3n^2 - n \leq bn^2$ for every $n \geq n_0$ when $a = 2$, $b = 3$ and $n_0 = 1$.

d) If $n \geq 10$, then $3n^2 - n + 10 \leq 3n^2$. On the other hand, still if $n \geq 10$, $3n^2 - n + 10 \geq 2n^2$ because $n^2 \geq 10n \geq 2n \geq n + 10$. Therefore, $an^2 \leq 3n^2 - n + 10 \leq bn^2$ for every $n \geq n_0$ when $a = 2$, $b = 3$ and $n_0 = 10$.

13.4.7. By definition, $n = d^{\log_d n}$. By taking $\log_c$ on both sides, and by using a well-known property of logarithms, we get that $\log_c n = \log_d n \log_c d$. Therefore, $\log_c n$ is a constant multiple of $\log_d n$, which implies that $\log_c n$ is $\Theta(\log_d n)$.

A more basic proof (one that doesn't use that property of logs) goes like this. By definition, $n = d^{\log_d n}$ and $d = c^{\log_c d}$. Therefore, $n = (c^{\log_c d})^{\log_d n} = c^{\log_c d \log_d n}$. By definition, this implies that $\log_c n = \log_c d \log_d n$. Therefore, $\log_c n$ is a constant multiple of $\log_d n$, which implies that $\log_c n$ is $\Theta(\log_d n)$.

13.5 Some Common Running Times

13.5.1.

n	10	10^3	10^6
$\log_2 n$ μs	3 μs	10 μs	20 μs
n μs	10 μs	1 ms	1 s
$n \log_2 n$ μs	33 μs	10 ms	20 s
n^2 μs	100 μs	1 s	12 days
n^3 μs	1 ms	17 min	32×10^3 years

n	10	20	40	60	80
n μs	10 μs	20 μs	40 μs	60 μs	80 μs
$n \log_2 n$ μs	33 μs	86 μs	210 μs	350 μs	510 μs
n^2 μs	0.1 ms	0.4 ms	1.6 ms	3.6 ms	6.4 ms
n^3 μs	1 ms	8 ms	64 ms	220 ms	510 ms
2^n μs	1 ms	1 s	13 days	37×10^3 years	38×10^9 years

13.5.2. $2^n > n$ when $n \geq 1$. $2^n > n^2$ when $n \geq 5$. $2^n > n^3$ when $n \geq 10$. $2^n > n^6$ when $n \geq 30$.

13.6 Basic Strategies

13.6.1.

a) The body of the inner loop runs in constant time c. The inner loop repeats $2n+1$ times. Therefore, the running time of the inner loop is $\Theta((2n+1)c) = \Theta(n)$. The inner loop always take the same amount of time and is repeated n times by the outer loop. Therefore, the running time of the outer loop is $\Theta(n^2)$.

b) The inner loop runs in time $\Theta(n)$ as we've seen before. The outer loop repeats 10 times. Therefore, the running time of the outer loop is $\Theta(10n) = \Theta(n)$.

c) The inner loop repeats 6 times. Therefore, since its body runs in constant time, the inner loop runs in constant time. The outer loop repeats the inner loop n times. Therefore, the running time of the outer loop is $\Theta(n)$.

d) The inner loop repeats $n-i+1$ times. Since i varies, this implies that the running time of the inner loop varies. Let $T(k)$ be the running time of the inner loop when it repeats k times. Therefore, the running time of the outer loop is

$$\Theta\left(\sum_{i=1}^{n} T(n-i+1)\right)$$

Note that

$$\sum_{i=1}^{n} T(n-i+1) = \sum_{i=1}^{n} T(i)$$

Then, since $T(k) = \Theta(k)$, there is a constant b such that

$$\sum_{i=1}^{n} T(i) \leq \sum_{i=1}^{n} bi = b\frac{n(n+1)}{2}$$

A similar argument also gives a quadratic lower bound. Therefore, the running time of the outer loop is $\Theta(n^2)$.

e) The inner loop repeats $2i+1$ times. Let $T(k)$ be the running time of the inner loop when it repeats k times. Therefore, the running time of the outer loop is

$$\Theta\left(\sum_{i=1}^{n} T(2i+1)\right)$$

Since $T(k) = \Theta(k)$, there is a constant b such that

$$\sum_{i=1}^{n} T(2i+1) \leq \sum_{i=1}^{n} b(2i+1) = 2b\sum_{i=1}^{n} i + bn = bn(n+1) + bn$$

A similar argument also gives a quadratic lower bound. Therefore, the running time of the outer loop is $\Theta(n^2)$.

f) The inner loop repeats i times. Let $T(k)$ be the running time of the inner loop when it repeats k times. Therefore, the running time of the outer loop is

$$\Theta\left(\sum_{i=1}^{n^2} T(i)\right)$$

(Note that the summation runs up to n^2.) Since $T(k) = \Theta(k)$, there is a constant b such that

$$\sum_{i=1}^{n^2} T(i) \leq b\sum_{i=1}^{n^2} i = b\frac{n^2(n^2+1)}{2} = b\frac{n^4+n^2}{2}$$

A similar argument also gives a lower bound with an n^4 dominant term. Therefore, the running time of the outer loop is $\Theta(n^4)$.

13.6.2. Since large values of n determine the asymptotic running time, we can ignore the *if* part of the **`if`** statement. Therefore, in all cases, the running time of this algorithm is $\Theta(T_A(n) + nT_C(n))$.

a) The running time is $\Theta(n) + n\Theta(\log n) = \Theta(n \log n)$ because $n \log n$ is the dominant term.

b) The running time is $\Theta(n^2) + n\Theta(\log n) = \Theta(n^2)$ because n^2 is the dominant term.

c) Same as (b).

13.7 Worst-Case and Average-Case Analysis

13.8 The Binary Search Algorithm

13.8.1.

```
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!
```

13.8.2. Suppose that the size n of the initial s is a power of 2. Say $n = 2^k$. The body of the loop now runs in time $\Theta(m)$ where m is the number of elements in `s`. Over the various iterations of the loop, m will have the values $2^k, 2^{k-1}, 2^{k-2}, \ldots, 1$. Ignoring low-order terms, we get that the total running time of the loop is

$$\Theta\left(\sum_{i=0}^{k} c2^i\right)$$

Now,

$$\sum_{i=0}^{k} c2^i = c\sum_{i=0}^{k} 2^i = c(2^{k+1}-1) = 2cn - c$$

Therefore, the running time of the loop is $\Theta(n)$ and the running time of the binary search is also $\Theta(n)$.

13.8.3. Right now, the middle element is the first one of the right half and when `e` equals the middle element, we go right. To find the first occurrence of `e` use the last element of the left half as middle element and when `e` equals the middle element, go left.

Chapter 14

Recursion

14.1 The Technique

14.1.3.

```
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;
    }
}
```

14.1.4.

```
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];
    }
}
```

14.1.5.

```
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;
    }
}
```

14.1.6.

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

14.1.7.

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

14.1.8.

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

14.2 When to Use Recursion

14.2.2. All of them.

14.3 Tail Recursion

14.3.3.

```
template <class T>
void display(const T a[], int start, int stop)
{
    while (start < stop) {
        cout << a[start] << ' ';
        ++start;
    }
}
```

```
template <class T>
int binary_search(const T a[], int start,
                  int stop, const T & e )
{
    while (stop - start >= 2) {
        int middle = (start + stop) / 2;
        if (e < a[middle])
            stop = middle;
        else
            start = middle;
    }
    if (stop - start == 1) {
        if (e == a[start])
            return start;
        else
            return -1;
    }
    // stop - start <= 0
    return -1;
}
```

14.3.4. The second `print` is the only one of those function that's tail recursive.

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

Chapter 15

Sorting

15.1 Selection Sort

15.1.1.

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

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

15.2 Insertion Sort

15.3 Mergesort

15.3.1.

```
[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]
```

15.3.2.

```
  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]
```

15.4 Quicksort

15.4.3.

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

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

15.4.4. If the array is already sorted, then the middle element of the array is the median, which implies that pivot will be the median element. If the elements are not reordered unnecessarily during the partitioning step, then the subarrays will again be ordered and the pivots will again be the median elements. This leads to partitions that are as even as possible.

Suppose that n is even. Consider the array that contains the following elements, in this order:

$$1, 3, 5, \ldots, n-1, 2, 4, 6, \ldots, n$$

The pivot will be 2 which will cause the right subarray to contain the following $n-2$ elements:

$$3, 5, \ldots, n-1, 4, 6, \ldots, n$$

If the elements are not reordered unnecessarily during the partitioning step, this will repeat, with the size of the right subarray decreasing by only 2 at every partition. (This would lead to a recurrence relation similar to that of selection sort and to a $\Theta(n^2)$ running time.)

If n is odd, the consider the following array:

$$2, 4, 6, \ldots, n-1, 1, 3, 5, \ldots, n$$

Note that 1 is the middle element of this array. The pivot will again be 2 and the right subarray will contain the following $n-2$ elements:

$$4, 6, \ldots, n-1, 3, 5, \ldots, n$$

The middle element is now 3 and the pattern repeats.

15.4.5. The best-case running time is $\Theta(n \log n)$. When the pivot is the middle element of the array, this is achieved on every array. When the pivot is the first element, this is achieved with an array whose elements follow this pattern:

$$n/2, n/4, 3n/4, n/8, 3n/8, 5n/8, 7n/8, \ldots$$

When the pivot is the median of the first, middle and last elements, a sorted array causes the pivot to be the median element, which leads to the optimal time. When the pivot is a random element, the best-case running time can be achieved on every array.

15.4.6. First, here's a simple implementation of the partitioning step. It does two scans of the array and uses a vector of the same size as the array as temporary storage. (Note that it's possible to do the partitioning with a single scan and no extra memory.)

```
int partition(T a[], int start, int stop, int pivot)
// Partitions elements in a about a pivot. Partition
// the elements in the range [start, stop). The
// argument pivot is the index of the pivot.
//
// PRECONDITION: The indices are valid, start occurs
// before stop and pivot is within the range [start,
// stop).
//
// POSTCONDITION: Returns an index new_pivot such
// that the elements in [start, new_pivot) are
// smaller than the pivot, the element at index
// new_pivot is equal to the pivot, and the
// elements in [new_pivot + 1, stop) are greater or
// equal to the pivot.
//
// ASSUMPTION ON TEMPLATE ARGUMENT: Values of type T
// can be compared using the < operator.
{
    std::swap(a[pivot], a[start]);
        // pivot is now at start
    std::vector<T> temp;
    temp.reserve(stop - start);
    for (int i = start + 1; i < stop; ++i)
        if (a[i] < a[start]) temp.push_back(a[i]);
    temp.push_back(a[start]);
    int new_pivot = start + temp.size() - 1;
    for (int i = start + 1; i < stop; ++i)
        if (!(a[i] < a[start]))
            temp.push_back(a[i]);
    std::copy(temp.begin(), temp.end(), a + start);
    return new_pivot;
}
```

Now, using this partition function, here's an implementation of the randomized version of quicksort.

```
template <class T>
void quicksort(T a[], int start, int stop)
// Sorts elements in a in increasing order using the
// mergesort algorithm.  Sorts elements in the range
// [start,stop).  Sorts according to the < operator.
//
// PRECONDITION: The indices are valid and start
// occurs before stop.
//
// ASSUMPTION ON TEMPLATE ARGUMENT: Values of type T
// can be compared using the < operator.
{
    if (stop - start > 1) {
        int pivot = start + rand() % (stop - start);

        pivot = partition(a, start, stop, pivot);
        quicksort(a, start, pivot);
        quicksort(a, pivot + 1, stop);
    }
}
```