Some programming environments run programs in a window that closes automatically when the program terminates. The solutions below include code as comments to keep the window open until a key is struck. In most cases this extra code consists of one or two `cin.get()` statements, but some program require more elaborate code.

Chapter 2

```
// pe2-1.cpp

#include <iostream>

int main()
{
    using namespace std;
    cout << "Glandville Gibbons\n";
    cout << "8234 Springle Road\n";
    cout << "Bright Rock, CA 94888\n";
    //cin.get();
    return 0;
}


// pe2-2.cpp

#include <iostream>

int main()
{
    using namespace std;
    cout << "Enter a distance in furlongs: ";
    double furlongs;
    cin >> furlongs;
    double feet;
    feet = 220 * furlongs;
    cout << furlongs << " furlongs = "
         << feet << " feet\n";
    //cin.get();
    //cin.get();
    return 0;
}


// pe2-3.cpp

#include <iostream>
using namespace std;

void mice();
void run();
int main()
{
   mice();
   mice();
   run();
   run();
```

```
   //cin.get();
   return 0;
}

void mice()
{
   cout << "Three blind mice\n";
}

void run()
{
   cout << "See how they run\n";
}


// pe2-4.cpp -- displays age in months
#include <iostream>
int main()
{
    using namespace std;

    cout << "Enter your age: ";
    int years;
    cin >> years;
    cout << "Your age in months is "
         << 12 * years << "." << endl;
   //cin.get();
   //cin.get();
    return 0;
}


// pe2-5.cpp

#include <iostream>

double C_to_F(double);
int main()
{
    using namespace std;
    cout << "Enter a temperature in Celsius: ";
    double C;
    cin >> C;
    double F;
    F = C_to_F(C);
    cout << C << " degrees Celsius = "
         << F << " degrees Fahrenheit\n";
   //cin.get();
   //cin.get();

   return 0;
}

double C_to_F(double temp)
{
    return 1.8 * temp + 32.0;
}
```

```
// pe2-6.cpp
#include <iostream>


double ly_to_au(double);
int main()
{
    using namespace std;
    double light_years;
    double astr_units;
    cout << "Enter the number of light years: ";
    cin >> light_years;
    astr_units = ly_to_au(light_years);
    cout << light_years << " light years = ";
    cout << astr_units << " astronomical units.\n";
   //cin.get();
   //cin.get();
    return 0;
}

double ly_to_au(double ly)
{
    return 63240.0 * ly;
}

// pe2-7.cpp -- displays hours and minutes
#include <iostream>
void showtime(int, int);     // function prototype
int main()
{
    using namespace std;

    cout << "Enter the number of hours: ";
    int hours;
    cin >> hours;
    cout << "Enter the number of minutes: ";
    int minutes;
    cin >> minutes;
    showtime(hours, minutes);
   //cin.get();
   //cin.get();
    return 0;
}

void showtime(int hrs, int mins)
{
    using namespace std;
    cout << "Time: " << hrs <<":" << mins << endl;
}
```

Chapter 3

```
// pe3-1.cpp
```

```
#include <iostream>

const int Inch_Per_Foot = 12;

int main()
{
    using namespace std;
// Note: some environments don't support the backspace character
    cout << "Please enter your height in inches: ___\b\b\b";
    int ht_inch;
    cin >> ht_inch;
    int ht_feet = ht_inch / Inch_Per_Foot;
    int rm_inch = ht_inch % Inch_Per_Foot;
    cout << "Your height is " << ht_feet << " feet, ";
    cout << rm_inch << " inch(es).\n";
   //cin.get();
   //cin.get();
    return 0;
}


// pe3-2.cpp

#include <iostream>

int main()
{
    using namespace std;
    const float INCHES_PER_FOOT = 12;
    const float METERS_PER_INCH = 0.0254;
    const float KG_PER_LB = 2.2;  // for standard Earth gravity
    cout << "This program calculates your Body Mass Index (BMI).\n";
    cout << "Enter your height in feet and inches.\n";
    cout << "First, enter the feet: ";
    float feet;
    cin >> feet;
    cout << "Now enter the inches: ";
    float inches;
    cin >> inches;
    cout << "Next, enter your weight in pounds: ";
    float pounds;
    cin >> pounds;
    float total_inches = INCHES_PER_FOOT * feet + inches;
    float meters = total_inches * METERS_PER_INCH;
    float kilograms = pounds / KG_PER_LB;
    float bmi = kilograms / (meters * meters);
    cout << "Your BMI = " << bmi << endl;
   //cin.get();
   //cin.get();
    return 0;
}

// pe3-3.cpp
#include <iostream>
const double MINS_PER_DEG = 60.0;
const double SECS_PER_MIN = 60.0;
int main()
```

```
{
    using namespace std;

    int degrees;
    int minutes;
    int seconds;
    double latitude;

    cout << "Enter a latitude in degrees, minutes, and seconds:\n";
    cout << "First, enter the degrees: ";
    cin >> degrees;
    cout << "Next, enter the minutes of arc: ";
    cin >> minutes;
    cout << "Finally, enter the seconds of arc: ";
    cin >> seconds;
    latitude = degrees + (minutes + seconds / SECS_PER_MIN)/MINS_PER_DEG;
    cout << degrees << " degrees, " << minutes << " minutes, "
         << seconds << " seconds = " << latitude << " degrees\n";
   //cin.get();
   //cin.get();
    return 0;
}


// pe3-4.cpp
#include <iostream>
const int HRS_PER_DAY = 24;
const int MINS_PER_HR = 60;
const int SECS_PER_MIN = 60;
int main()
{
    using namespace std;
    long time_in_sec;
    int days;
    int hours;
    int minutes;
    int seconds;

    cout << "Enter the number of seconds: ";
    cin >> time_in_sec;
    seconds = time_in_sec % SECS_PER_MIN;
    minutes = time_in_sec / SECS_PER_MIN;
    hours = minutes / MINS_PER_HR;
    minutes = minutes % MINS_PER_HR;
    days = hours / HRS_PER_DAY;
    hours = hours % HRS_PER_DAY;
    cout << time_in_sec << " seconds = " << days << " days, "
         << hours << " hours, "
         << minutes << " minutes, " << seconds << " seconds\n";
   //cin.get();
   //cin.get();
    return 0;
}

// pe3-5.cpp
#include <iostream>
int main()
```

```
{
   using namespace std;
   long long world_pop;
   long long us_pop;

   cout << "Enter the world's population: ";
   cin >> world_pop;
   cout << "Enter the population of the US: ";
   cin >> us_pop;
    double per_cent = double(us_pop)/double(world_pop) * 100;
    cout << "The population of the US is " << per_cent
         << "% of the world population.\n";
   //cin.get();
   //cin.get();
   return 0;
}

// pe3-6.cpp

#include <iostream>

int main()
{
    using namespace std;
    cout << "How many miles have you driven your car? ";
    float miles;
    cin >> miles;
    cout << "How many gallons of gasoline did the car use? ";
    float gallons;
    cin >> gallons;
    cout << "Your car got " << miles / gallons;
    cout << " miles per gallon.\n";
   //cin.get();
   //cin.get();
    return 0;
}


// pe3-7.cpp

#include <iostream>

const double KM100_TO_MILES = 62.14;
const double LITERS_PER_GALLON = 3.875;

int main()
{
    using namespace std;
    double euro_rating;
    double us_rating;
    cout << "Enter fuel consumption in liters per 100 km: ";
    cin >> euro_rating;
    // divide by LITER_PER_GALLON to get gallons per 100-km
    // divide by KM100_TO_MILES to get gallons per mile
    // invert result to get miles per gallon
    us_rating = (LITERS_PER_GALLON * KM100_TO_MILES) / euro_rating;
    cout << euro_rating << " liters per 100 km is ";
```

```
    cout << us_rating << " miles per gallon.\n";
    //cin.get();
    //cin.get();
    return 0;
}
```

Chapter 4

```
// pe4-1.cpp

#include <iostream>

const int Arsize = 20;

int main()
{
    using namespace std;
    char fname[Arsize];
    char lname[Arsize];
    char grade;
    int age;

    cout << "What is your first name? ";
    cin.getline(fname, Arsize);
    cout << "What is your last name? ";
    cin >> lname;
    cout << "What letter grade do you deserve? ";
    cin >> grade;
    cout << "What is your age? ";
    cin >> age;
    cout << "Name: " << lname << ", " << fname << "\n";
    grade = grade + 1;
    cout << "Grade: " << grade << "\n";
    // note that using << grade + 1 wouldn't work correctly
    cout << "Age: " << age << "\n";
   //cin.get();
   //cin.get();
    return 0;
}

// pe4-2.cpp -- storing strings in string objects
#include <iostream>
#include <string>
int main()
{
    using namespace std;
    string name;
    string dessert;

    cout << "Enter your name:\n";
    getline(cin, name);  // reads through newline
    cout << "Enter your favorite dessert:\n";
    getline(cin, dessert);
    cout << "I have some delicious " << dessert;
    cout << " for you, " << name << ".\n";
   //cin.get();
```

```
    return 0;
}


// pe4-3.cpp -- storing strings in char arrays
#include <iostream>
#include <cstring>
const int SIZE = 20;
int main()
{
    using namespace std;
    char firstName[SIZE];
    char lastName[SIZE];
    char fullName[2*SIZE + 1];

    cout << "Enter your first name: ";
    cin >> firstName;
    cout << "Enter your last name: ";
    cin >> lastName;
    strncpy(fullName,lastName,SIZE);
    strcat(fullName, ", ");
    strncat(fullName, firstName, SIZE);
    fullName[SIZE - 1] = '\0';
    cout << "Here's the information in a single string: "
         << fullName << endl;
   //cin.get();
   //cin.get();
    return 0;
}


// pe4-4.cpp -- storing strings in string objects
#include <iostream>
#include <string>
int main()
{
    using namespace std;
    string firstName;
    string lastName;
    string fullName;

    cout << "Enter your first name: ";
    cin >> firstName;
    cout << "Enter your last name: ";
    cin >> lastName;
    fullName = lastName + ", " + firstName;
    cout << "Here's the information in a single string: "
         << fullName << endl;
   //cin.get();
   //cin.get();
    return 0;
}


// pe4-5.cpp
// a candybar  structure
struct CandyBar {
```

```
    char brand[40];
    double weight;
    int calories;
};

#include <iostream>

int main()
{
    using namespace std;  //introduces namespace std
    CandyBar snack = { "Mocha Munch", 2.3, 350 };

    cout << "Brand name: " << snack.brand << endl;
    cout << "Weight: " << snack.weight << endl;
    cout << "Calories: " << snack.calories << endl;
    //cin.get();
    return 0;
}


// pe4-6.cpp
// an array of candybars

struct CandyBar
{
    char brand[40];
    double weight;
    int calories;
};

#include <iostream>

int main()
{
    using namespace std;  //introduces namespace std
    CandyBar snacks[3] =
    {
         { "Mocha Munch", 2.3, 350 },
         { "Cocoa Brittle", 1.8, 320},
         { "Pluto Bar", 1.2, 280}
    };

    cout << "Bar #1:\n";
    cout << "Brand name: " << snacks[0].brand << endl;
    cout << "Weight: " << snacks[0].weight << endl;
    cout << "Calories: " << snacks[0].calories << endl;
    cout << "Bar #2:\n";
    cout << "Brand name: " << snacks[1].brand << endl;
    cout << "Weight: " << snacks[1].weight << endl;
    cout << "Calories: " << snacks[1].calories << endl;
    cout << "Bar #3:\n";
    cout << "Brand name: " << snacks[2].brand << endl;
    cout << "Weight: " << snacks[2].weight << endl;
    cout << "Calories: " << snacks[2].calories << endl;
   //cin.get();
    return 0;
}
```

```
// pe4-7.ccp

#include <iostream>

const int Slen = 70;

struct pizza {
    char name[Slen];
    float diameter;
    float weight;
};

int main()
{
    using namespace std;
    pizza pie;
    cout << "What is the name of the pizza company? ";
    cin.getline(pie.name, Slen);
    cout << "What is the diameter of the pizza in inches? ";
    cin >> pie.diameter;
    cout << "How much does the pizza weigh in ounces? ";
    cin >> pie.weight;
    cout << "Company: " << pie.name << "\n";
    cout << "Diameter: " << pie.diameter << " inches\n";
    cout << "Weight: " << pie.weight << " ounces\n";
   //cin.get();
   //cin.get();
    return 0;
}


// pe4-8.ccp

#include <iostream>

const int Slen = 70;

struct pizza {
    char name[Slen];
    float diameter;
    float weight;
};

int main()
{
    using namespace std;
    pizza *ptr = new pizza;
    cout << "What is the diameter of the pizza in inches? ";
    cin >> ptr->diameter;
    while (cin.get() != '\n')
        ; // get rid of rest of line before reading a string
    cout << "What is the name of the pizza company? ";
    cin.getline(ptr->name, Slen);
    cout << "How much does the pizza weigh in ounces? ";
    cin >> ptr->weight;
```

```
    cout << "Company: " << ptr->name << "\n";
    cout << "Diameter: " << ptr->diameter << " inches\n";
    cout << "Weight: " << ptr->weight << " ounces\n";
    delete ptr;
   //cin.get();
   //cin.get();
    return 0;
}


// pe4-9.cpp
// an array of candybars

struct CandyBar
{
    char brand[40];
    double weight;
    int calories;
};

#include <iostream>

int main()
{
    using namespace std;  //introduces namespace std
    CandyBar  * snacks = new CandyBar[3];

    strcpy(snacks[0].brand, "Mocha Munch");
    snacks[0].weight = 2.3;
    snacks[0].calories = 350;
    strcpy(snacks[1].brand, "Cocoa Brittle");
    snacks[1].weight = 1.8;
    snacks[1].calories = 320;
    strcpy(snacks[2].brand, "Pluto Bar");
    snacks[2].weight = 1.2;
    snacks[2].calories = 280;

    cout << "Bar #1:\n";
    cout << "Brand name: " << snacks[0].brand << endl;
    cout << "Weight: " << snacks[0].weight << endl;
    cout << "Calories: " << snacks[0].calories << endl;
    cout << "Bar #2:\n";
    cout << "Brand name: " << snacks[1].brand << endl;
    cout << "Weight: " << snacks[1].weight << endl;
    cout << "Calories: " << snacks[1].calories << endl;
    cout << "Bar #3:\n";
    cout << "Brand name: " << snacks[2].brand << endl;
    cout << "Weight: " << snacks[2].weight << endl;
    cout << "Calories: " << snacks[2].calories << endl;

    delete [] snacks;
   //cin.get();
    return 0;
}

//pe4-10.cpp
#include <iostream>
```

```
#include <array>
int main()
{
    using namespace std;
    array<double, 3> t40;
// or double t40[3]; if array not available
    cout << "Enter the first 40-yd dash time: ";
    cin >> t40[0];
    cout << "Enter the second 40-yd dash time: ";
    cin >> t40[1];
    cout << "Enter the third 40-yd dash time: ";
    cin >> t40[2];
    cout << "Time 1:  " << t40[0] << endl;
    cout << "Time 2:  " << t40[1] << endl;
    cout << "Time 3:  " << t40[2] << endl;
    double average = (t40[0] + t40[1] + t40[2])/3.0;
    cout << "Average: " << average << endl;
   //cin.get();
   //cin.get();
    return 0;
}
```

Chapter 5

```
// pe5-1.cpp

#include <iostream>

int main()
{
    using namespace std;
    int start;
    cout << "Enter the starting integer: ";
    cin >> start;

    int end;
    cout << "Enter the ending integer: ";
    cin >> end;

    int sum = 0;

    for (int i = start; i <= end; i++)
        sum += i;

    cout << "The sum of the digits " << start
            << " through " << end << " is "
            << sum << ".\n";
    //cin.get();
    //cin.get();
    return 0;
}

// pe5-2.cpp -- using array object
#include <iostream>
```

```
#include <array>
const int ArSize = 101;      // example of external declaration
int main()
{
    std::array<long double, ArSize> factorials;
    factorials[1] = factorials[0] = 1.0L;
    for (int i = 2; i < ArSize; i++)
        factorials[i] = i * factorials[i-1];
    for (int i = 0; i < ArSize; i++)
        std::cout << i << "! = " << factorials[i] << std::endl;
    //std::cin.get();
    return 0;
}


// pe5-3.cpp

#include <iostream>

int main()
{
    using namespace std;
    double sum = 0.0;
    double in;
    cout << "Enter a number (0 to terminate) : ";
    cin >> in;
    while (in != 0) {
        sum += in;
        cout << "Running total = " << sum << "\n";
        cout << "Enter next number (0 to terminate) : ";
        cin >> in;
    }
    cout << "Bye!\n";
    //cin.get();
    //cin.get();
    return 0;
}


// pe5-4.cpp
// daphne and cleo
#include <iostream>

const double daphneRate = 0.10;
const double cleoRate = 0.05;
const double invest = 100.0;
int main()
{
    using namespace std;  //introduces namespace std
    double daphne = invest;
    double cleo = invest;
    int year = 0;

    while (cleo <= daphne)
    {
        daphne += daphneRate * invest;
        cleo += cleoRate * cleo;
```

```
        year++;
    }

    cout << "Accounts after " << year << " years:\n";
    cout << "Cleo: $" << cleo << endl;
    cout << "Daphne: $" << daphne << endl;
    //cin.get();
    return 0;
}


// pe5-5.cpp
// book sales
#include <iostream>

const int MONTHS = 12;
const char * months[MONTHS] = {"January", "February", "March", "April",
                           "May", "June", "July", "August", "September",
                            "October", "November", "December"};
// or include <string> and use const std::string months[MONTHS]
int main()
{
    using namespace std;  //introduces namespace std
    int sales[MONTHS];
    int month;

    cout << "Enter the monthly sales for \"C++ for Fools\":\n";
    for (month = 0; month < MONTHS; month++)
    {
        cout << "Sales for " << months[month] << ": ";
        cin >> sales[month];
    }

    double total = 0.0;
    for (month = 0; month < MONTHS; month++)
        total += sales[month];

    cout << "Total sales: " << total << endl;
    //cin.get();
    //cin.get();
    return 0;
}


// pe5-6.cpp
// book sales for three years

#include <iostream>
#include <string>

const int MONTHS = 12;
const int YEARS = 3;
const std::string months[MONTHS] = {"January", "February", "March", "April",
                           "May", "June", "July", "August", "September",
                            "October", "November", "December"};
// or use const char * months[MONTHS]
int main()
```

```
{
    using namespace std;  //introduces namespace std
    int sales[YEARS][MONTHS];
    int month;
    int year;

    cout << "Enter three years of monthly sales for"
         << " \"C++ for Fools\":\n";
    for (year = 0; year < YEARS; year++)
    {
        cout << "Year " << year + 1 << ":\n";
        for (month = 0; month < MONTHS; month++)
        {
            cout << "Sales for " << months[month] << ": ";
            cin >> sales[year][month];
        }
    }

    double total;
    double grandtotal = 0.0;
    for (year = 0; year < YEARS; year++)
    {
        total = 0.0;

        for (month = 0; month < MONTHS; month++)
            total += sales[year][month];
        cout << "Sales for year " << year + 1 << ": " << total << endl;
        grandtotal += total;
    }
    cout << "Total sales: " << grandtotal << endl;
    //cin.get();
    //cin.get();
    return 0;
}


// pe5-7.cpp

#include <iostream>

struct car { char name[20]; int year;};

int main()
{
    using namespace std;
    int n;
    cout << "How many cars do you wish to catalog?: ";

    cin >> n;

    while(cin.get() != '\n')    // get rid of rest of line
        ;

    car * pc = new car [n];

    int i;
    for (i = 0; i < n; i++)
```

```
    {
        cout << "Car #" << (i + 1) << ":\n";
        cout << "Please enter the make: ";
        cin.getline(pc[i].name,20);
        cout << "Please enter the year made: ";
        cin >> pc[i].year;
        while(cin.get() != '\n')    // get rid of rest of line
            ;
    }
    cout << "Here is your collection:\n";
    for (i = 0; i < n; i++)
        cout << pc[i].year << " " << pc[i].name << "\n";

    delete [] pc;
    //cin.get();
    return 0;
}


// pe5-8.cpp -- count words using C-style string

#include <iostream>
#include <cstring>     // prototype for strcmp()
const int STR_LIM = 50;
int main()
{
    using namespace std;
    char word[STR_LIM];
    int count = 0;

    cout << "Enter words (to stop, type the word done):\n";

    while (cin >> word && strcmp("done", word))
        ++count;

    cout << "You entered a total of " << count << " words.\n";
/*
    while (cin.get() != '\n')
         continue;
    cin.get();
*/
    return 0;
}


// pe5-9.cpp -- count words using string class

#include <iostream>
#include <string>     // string class
int main()
{
    using namespace std;
    string word;
    int count = 0;

    cout << "Enter words (to stop, type the word done):\n";
```

```
    while (cin >> word && word != "done")
        ++count;

    cout << "You entered a total of " << count << " words.\n";
/*
    while (cin.get() != '\n')
         continue;
    cin.get();
*/
    return 0;
}


// pe5-10.cpp
//nested loops

#include <iostream>

int main()
{
    using namespace std;  //introduces namespace std
    int rows;
    int row;
    int col;
    int periods;

    cout << "Enter number of rows: ";
    cin >> rows;

    for (row = 1; row <= rows; row++)
    {
        periods = rows - row;
        for (col = 1; col <= periods; col++)
            cout << '.';
        // col already has correct value for next loop
        for (   ; col <= rows; col++)
            cout << '*';
        cout << endl;
    }
    //cin.get();
    return 0;
}
```

Chapter 6

```
// pe6-1.cpp
#include <iostream>
#include <cctype>
int main( )
{
    using namespace std;    //introduces namespace std
    char ch;

    cin.get(ch);
    while(ch != '@')
```

```
    {
        if (!isdigit(ch))
        {
            if (isupper(ch))
                ch = tolower(ch);
            else if (islower(ch))
                ch = toupper(ch);
            cout << ch;
        }
        cin.get(ch);
    }
/*
    while (cin.get() != '\n')
         continue;
    cin.get();
*/
    return 0;
}


// pe6-2.cpp --- non-numeric input terminates loop

#include <iostream>

const int Max = 10;

int main(void)
{
    using namespace std;
    double donations[Max];

    cout <<  "Please enter the donations.\n\n";
    cout <<  "You may enter up to " << Max
            << " donations <q to terminate>.\n\n";
    cout << "donation #1: ";
    int i = 0;
    while (i < Max && cin >> donations[i])
    {
        if (++i < Max)
            cout << "donation #" << (i+1) << ": ";
    }
    double total = 0.0;
    int j;
    for (j = 0; j < i; j++)
        total += donations[j];

    if (i == 0)
            cout << "No donations\n\n";
    else
    {
            double average = total / i;
            cout << "$" << average << " = average of "
                            << i << " donations\n";
            int above = 0;
            for (j = 0; j < i; j++)
            {
                    if (donations[j] > average)
```

```
                            ++above;
            }
        cout << above <<  " contributions above average\n";
    }
/* keep window open
    if (!cin)  // input terminated by non-numeric response
    {
        cin.clear();  // reset input
        while (cin.get() != '\n')
            continue;   // read rest of line
    }
    else
        cin.get();      // read end of line after last input
    cin.get();          // wait before closing window
*/
    return 0;
}


// pe6-3.cpp

#include <iostream>

int main()
{
    using namespace std;
    cout << "Please enter one of the following choices:\n";
    cout << "c) carnivore        p) pianist\n"
         << "t) tree             g) game\n";
    char ch;
    cin >> ch;
    while (ch != 'c' && ch != 'p' && ch != 't' && ch != 'g')
    {
        cout << "Please enter a c, p, t, or g: ";
        cin >> ch;
    }
    switch (ch)
    {
        case 'c' :  cout << "A cat is a carnivore.\n";
                    break;
        case 'p' :  cout << "Radu Lupu is a pianist.\n";
                    break;
        case 't' :  cout << "A maple is a tree.\n";
                    break;
        case 'g' :  cout << "Golf is a game.\n";
                    break;
        default  :  cout <<  "The program shouldn't get here!\n";
    }
    //cin.get();
    //cin.get();
    return 0;
}


// pe6-4.cpp

#include <iostream>
```

```
const int strsize = 40;
const int bopsize = 5;

// Benevolent Order of Programmers name structure
struct bop {
            char fullname[strsize]; // wordly name
            char title[strsize];    // job title
            char bopname[strsize];  // secret BOP name
            int preference;         // 0 = fullname, 1 = title, 2 = bopname
    };

void showa(bop ar[], int n);
void showb(bop ar[], int n);
void showc(bop ar[], int n);
void showd(bop ar[], int n);

int main()
{
    using namespace std;
    bop team[bopsize] = {
        {"Wimp Macho", "Senior Programmer", "UNIXMAN", 0},
        {"Raki Rhodes", "Junior Programmer", "ESATA", 1},
        {"Celia Laiter",  "Junior Analyst", "MIPS", 2},
        {"Hoppy Hipman", "Analyst Trainee", "THUNDERBOLT", 1},
        {"Pat Hand", "Junior Programmer", "LOOPY" ,2 } };

    cout << "Benevolent Order of Programmers Report\n";
    cout << "a. display by name     b. display by title\n"
         << "c. display by bopname  d. display by preference\n"
         << "q. quit\n";
    cout << "Enter your choice: ";
    char choice;
    cin >> choice;
    while (choice != 'q')
    {
        switch(choice)
        {
            case 'a'    : showa(team, bopsize); break;
            case 'b'    : showb(team, bopsize); break;
            case 'c'    : showc(team, bopsize); break;
            case 'd'    : showd(team, bopsize); break;
            default     : cout << "Enter only a, b, c, d, or q.\n";
        }
        cout <<  """Next choice: """;
        cin >> choice;
    }
    cout << "Bye!\n";
    //cin.get();
    //cin.get();
    return 0;
}


void showa(bop ar[], int n)
{
    using namespace std;
```

```
    for(int i = 0; i < n; i++)
        cout << ar[i].fullname << "\n";
}

void showb(bop ar[], int n)
{
    using namespace std;
    for(int i = 0; i < n; i++)
        cout << ar[i].title << "\n";
}

void showc(bop ar[], int n)
{
    using namespace std;
    for(int i = 0; i < n; i++)
        cout << ar[i].bopname << "\n";
}

void showd(bop ar[], int n)
{
    using namespace std;
    for(int i = 0; i < n; i++)
        if (ar[i].preference == 0)
            cout << ar[i].fullname << "\n";
        else if (ar[i].preference == 1)
            cout << ar[i].title << "\n";
        else if (ar[i].preference == 2)
            cout << ar[i].bopname << "\n";
        else
            cout << "oops\n";
}


// pe6-5.cpp
// Neutronia taxation
#include <iostream>
const double LEV1 = 5000;
const double LEV2 = 15000;
const double LEV3 = 35000;
const double RATE1 = 0.10;
const double RATE2 = 0.15;
const double RATE3 = 0.20;
int main( )
{
    using namespace std;
    double income;
    double tax;

    cout << "Enter an annual income in tvarps: ";
    while (cin >> income && income >= 0)
   {
      if (income <= LEV1)
            tax = 0;
        else if (income <= LEV2)
            tax = (income - LEV1) * RATE1;
        else if (income <= LEV3)
            tax = RATE1 * (LEV2 - LEV1) + RATE2 * (income - LEV2);
```

```
        else
            tax = RATE1 * (LEV2 - LEV1) + RATE2 * (LEV3 - LEV2)
            + RATE3 * (income - LEV3);

        cout << "Neutronia is owed " <<  tax << " tvarps in taxes.\n";
       cout << "Enter an annual income in tvarps: ";
   }
   cout << "Done!\n";
/* keep window open
    if (!cin)  // input terminated by non-numeric response
    {
        cin.clear();  // reset input
        while (cin.get() != '\n')
            continue;   // read rest of line
    }
    else
        cin.get();      // read end of line after last input
    cin.get();          // wait before closing window
*/
    return 0;
}



// pe6-6.cpp
#include <iostream>

struct Contributor
{
    char name[80];
    double amount;
};

int main()
{
    using namespace std;
    int contributors;
    cout << "Enter the number of contributors: ";
    cin >> contributors;
    while (cin.get() != '\n')
        continue;
    Contributor * pc = new Contributor[contributors];
    int i;
    for (i = 0; i < contributors; i++)
    {
        cout << "Enter contributor's name: ";
        if (!cin.getline(pc[i].name, 80))
            break;
        cout << "Enter amount of contribution: $";
        if (!(cin >> pc[i].amount))
            break;
        while (cin.get() != '\n')
            continue;
    }
    int total = i;
    cout << "Grand Patrons:\n";
    int ct = 0;
```

```
    for (i = 0; i < total; i++)
    {
        if (pc[i].amount >= 10000.0)
        {
            ct++;
            cout << pc[i].name << ": $" << pc[i].amount << endl;
        }
    }
    if (ct == 0)
        cout << "None\n";
    cout << "Patrons:\n";
    ct = 0;
    for (i = 0; i < total; i++)
    {
        if (pc[i].amount < 10000.0)
        {
            ct++;
            cout << pc[i].name << ": $" << pc[i].amount << endl;
        }
    }
    if (ct == 0)
        cout << "None\n";
    delete [] pc;

    cout << "Done.\n";
/* keep window open
    if (!cin)  // input terminated by non-numeric response
    {
        cin.clear();  // reset input
        while (cin.get() != '\n')
            continue;   // read rest of line
    }
    cin.get();          // wait before closing window
*/
    return 0;
}


// pe6-7.cpp
#include <iostream>
#include <string>
int main()
{
    using namespace std;
    string word;
    char ch;
    int vowel = 0;
    int consonant = 0;
    int other = 0;
    cout << "Enter words (q to quit):\n";
    cin >> word;
    while ( word != "q")
    {
        ch = tolower(word[0]);
        if (isalpha(ch))
        {
            if (ch == 'a' || ch == 'e' || ch ==  'i' || ch == 'o'
```

```
                        || ch == 'u')
                vowel++;
            else
                consonant++;
        }
        else
            other++;
        cin >> word;
    }
    cout << vowel <<" words beginning with vowels\n";
    cout << consonant << " words beginning with consonants\n";
    cout << other << " others\n";
    //cin.get();
    //cin.get();
    return 0;
}


// pe6-8.cpp -- counting characters
#include <iostream>
#include <fstream>          // file I/O suppport
#include <cstdlib>          // support for exit()
const int SIZE = 60;
int main()
{
    using namespace std;
    char filename[SIZE];
    char ch;
    ifstream inFile;        // object for handling file input

    cout << "Enter name of data file: ";
    cin.getline(filename, SIZE);
    inFile.open(filename);  // associate inFile with a file
    if (!inFile.is_open())  // failed to open file
    {
        cout << "Could not open the file " << filename << endl;
        cout << "Program terminating.\n";
        //cin.get();
        exit(EXIT_FAILURE);
    }
    int count = 0;          // number of items read

    inFile >> ch;        // get first value
    while (inFile.good())   // while input good and not at EOF
    {
        count++;            // one more item read
        inFile >> ch;    // get next value
    }

    cout << count << " characters in " << filename << endl;

    inFile.close();         // finished with the file
    //cin.get();
    return 0;
}
```

```
// pe6-9.cpp
#include <iostream>
#include <fstream>
#include <cstdlib>

struct Contributor
{
    char name[80];
    double amount;
};
const int SIZE = 60;
int main()
{
    using namespace std;
    char filename[SIZE];
    char ch;
    ifstream inFile;        // object for handling file input

    cout << "Enter name of data file: ";
    cin.getline(filename, SIZE);
    inFile.open(filename);  // associate inFile with a file
    if (!inFile.is_open())  // failed to open file
    {
        cout << "Could not open the file " << filename << endl;
        cout << "Program terminating.\n";
        //cin.get();
        exit(EXIT_FAILURE);
    }
    int contributors;
    if (!(inFile >> contributors))
    {
        cout << "Data mismatch for contributor count.\n";
        //cin.get();
        exit(EXIT_FAILURE);
    }
    while (inFile.get(ch)&& ch != '\n')
        continue;
    if (!inFile)
    {
        cout << "Problems reading file.\n";
        //cin.get();
        exit(EXIT_FAILURE);
    }

    Contributor * pc = new Contributor[contributors];
    int i = 0;
    while (inFile && i < contributors)
    {
        if ( !inFile.getline(pc[i].name, 80))
            break;
        if (!(inFile >> pc[i].amount))
            break;
        i++;
        while (inFile.get(ch)&& ch != '\n')
            continue;
    }
    int total = i;
```

```
    cout << "Grand Patrons:\n";
    int ct = 0;
    for (i = 0; i < total; i++)
    {
        if (pc[i].amount >= 10000.0)
        {
            ct++;
            cout << pc[i].name << ": $" << pc[i].amount << endl;
        }
    }
    if (ct == 0)
        cout << "None\n";
    cout << "Patrons:\n";
    ct = 0;
    for (i = 0; i < total; i++)
    {
        if (pc[i].amount < 10000.0)
        {
            ct++;
            cout << pc[i].name << ": $" << pc[i].amount << endl;
        }
    }
    if (ct == 0)
        cout << "None\n";
    delete [] pc;
    inFile.close();
    cout << "Done.\n";
    //cin.get();
    return 0;
}
```

Chapter 7

```
//pe7-1.cpp -- harmonic mean

#include <iostream>

double h_mean(double x, double y);

int main()
{
    using namespace std;
    double x,y;

    cout << "Enter two numbers (a 0 terminates): ";
    while (cin >> x >> y && x * y != 0)
        cout << "harmonic mean of " << x << " and "
             << y << " = " << h_mean(x,y) << "\n";
/* or do the reading and testing in two parts:
    while (cin >> x && x != 0)
    {
        cin >> y;
        if (y == 0)
```

```
            break;
            ...
*/
    cout << "Bye\n";
    //cin.get();
    //cin.get();
    return 0;
}

double h_mean(double x, double y)
{
    return 2.0 * x * y / (x + y);
}


// pe7-2.cpp

#include <iostream>

int getgolf(int ar[], int n);
void showgolf(const int ar[], int n);
double ave(const int ar[] , int n);

const int Scores = 10;

int main()
{
    using namespace std;
    int golfscores[Scores];

    int games = getgolf(golfscores, Scores);
    showgolf(golfscores, games);
    if (games > 0)
        cout << ave(golfscores, games) << " = average\n";
    else
        cout << "No scores!\n";
    //cin.get();
    //cin.get();
    return 0;
}

int getgolf(int ar[], int n)
{
    using namespace std;
    cout << "Enter up to " << n << " scores <q to quit>:\n";
    int i;
    for (i = 0; i < n; i++)
    {
        if (!(cin >> ar[i]))
        {
            cin.clear(); // in case more input needed later
            while (cin.get() != '\n')
                continue;
            break;
        }
    }
    return i;
```

```
}

void showgolf(const int ar[], int n)
{
    using namespace std;
    for (int i = 0; i < n; i++)
        cout << ar[i] << " ";
    cout << "\n";
}

double ave(const int ar[] , int n)
{
    double tot = 0.0;
    for (int i = 0; i < n; i++)
        tot += ar[i];
    return tot / n;
}


// pe7-3.cpp

#include <iostream>

struct box {
    char maker[40];
    float height;
    float width;
    float length;
    float volume;
};

void showbox(box b);
void setbox(box * pb);

int main()
{
    box carton = {"Bingo Boxer", 2, 3, 5}; // no volume provided
    setbox(&carton);
    showbox(carton);
    //std::cin.get();
    return 0;
}

void showbox(box b)
{
    using namespace std;
    cout << "Box maker: " << b.maker
         << "\nheight: " << b.height
         << "\nlwidth: " << b.width
         << "\nlength: " << b.length
         << "\nvolume: " << b.volume << "\n";
}

void setbox(box * pb)
{
    pb->volume = pb->height * pb->width * pb->length;
}
```

```
// pe7-4.cpp -- probability of winning
#include <iostream>
long double probability(unsigned numbers, unsigned picks);

int main()
{
    using namespace std;
    double total, choices;
    double mtotal;
    double probability1, probability2;
    cout << "Enter total number of game card choices and\n"
            "number of picks allowed for the field:\n";
    while ((cin >> total >> choices) && choices <= total)
    {
        cout << "Enter total number of game card choices "
                "for the mega number:\n";
        if (!(cin >> mtotal))
            break;
        cout << "The chances of getting all " << choices << " picks is one in
"
             << (probability1 = probability(total, choices) ) << ".\n";
        cout << "The chances of getting the megaspot is one in "
             << (probability2 = probability(mtotal, 1) ) << ".\n";
        cout << "You have one chance in ";
        cout << probability1 * probability2;      // compute the probability
        cout << " of winning.\n";
        cout << "Next set of numbers (q to quit): ";
    }
    cout << "bye\n";
/* keep window open
    if (!cin)  // input terminated by non-numeric response
    {
        cin.clear();  // reset input
        while (cin.get() != '\n')
            continue;   // read rest of line
    }
    else
        cin.get();      // read end of line after last input
    cin.get();          // wait before closing window
*/
    return 0;
}

// the following function calculates the probability of picking picks
// numbers correctly from numbers choices
long double probability(unsigned numbers, unsigned picks)
{
    long double result = 1.0;  // here come some local variables
    long double n;
    unsigned p;

    for (n = numbers, p = picks; p > 0; n--, p--)
        result = result * n / p ;
    return result;
}
```

```
// pe7-5.cpp

#include <iostream>

double rfact(int n);

int main()
{
    using namespace std;
    int num;
    cout << "Enter an integer (< 0 to quit): ";
    while(cin >> num && num >= 0)
    {
        cout << num << " factorial = " << rfact(num) << "\n";
        cout << "Enter next value (<0 to quit): ";
    }
    cout << "Bye!\n";
/* keep window open
    if (!cin)  // input terminated by non-numeric response
    {
        cin.clear();  // reset input
        while (cin.get() != '\n')
            continue;   // read rest of line
    }
    else
        cin.get();      // read end of line after last input
    cin.get();          // wait before closing window
*/
   return 0;
}

double rfact(int n)
{
    if (n <= 1)
        return 1;
    else
        return n * rfact(n-1);
}

// pe7-6.cpp
#include <iostream>
int Fill_array(double ar[], int size);
void Show_array(const double ar[], int size);
void Reverse_array(double ar[], int size);
const int LIMIT = 10;

int main( )
{
    using namespace std;
    double values[LIMIT];

    int entries = Fill_array(values, LIMIT);
    cout << "Array values:\n";
    Show_array(values, entries);
    cout << "Array reversed:\n";
```

```
    Reverse_array(values, entries);
    Show_array(values, entries);
    cout << "All but end values reversed:\n";
    Reverse_array(values + 1, entries - 2);
    Show_array(values, entries);
/* keep window open
    if (!cin)  // input terminated by non-numeric response
    {
        cin.clear();  // reset input
        while (cin.get() != '\n')
            continue;   // read rest of line
    }
    else
        cin.get();      // read end of line after last input
    cin.get();          // wait before closing window
*/

    return 0;
}

int Fill_array(double ar[], int size)
{
    using namespace std;
    int n;
    cout << "Enter up to " << size << " values (q to quit):\n";
    for (n = 0; n < size; n++)
    {
        cin >> ar[n];
        if (!cin)
            break;
    }
    return n;
}

void Show_array(const double ar[], int size)
{
    using namespace std;
    int n;
    for (n = 0; n < size; n++)
    {
        cout << ar[n];
        if (n % 8 == 7)
            cout << endl;
        else
            cout << ' ';
    }
    if (n % 8 != 0)
        cout << endl;
}
void Reverse_array(double ar[], int size)
{
    int i, j;
    double temp;

    for (i = 0, j = size - 1; i < j; i++, j--)
    {
        temp = ar[i];
```

```
        ar[i] = ar[j];
        ar[j] = temp;
    }
}


// pe7-7.cpp -- array functions using ranges
#include <iostream>
const int LIMIT = 10;

// function prototypes
double * Fill_array(double * beg, double * end);
void Show_array(const double * beg, const double * end);  // don't change
data
void Reverse_array(double * beg, double * end);

int main()
{
    using namespace std;
    double values[LIMIT];
    double * true_end;

    true_end = Fill_array(values, values + LIMIT);
    cout << "Array values:\n";
    Show_array(values, true_end);
    cout << "Array reversed:\n";
    Reverse_array(values, true_end);
    Show_array(values, true_end);
    cout << "All but end values reversed:\n";
    Reverse_array(values + 1,true_end - 1);
    Show_array(values, true_end);
    cout << "Done.\n";
    //cin.get();
    return 0;
}

double * Fill_array(double * beg, double * end)
{
    using namespace std;
    double temp;
    double * pt;
    int i;

    for (pt = beg, i = 0; pt != end; pt++, i++)
    {
        cout << "Enter value #" << (i + 1) << ": ";
        cin >> temp;
        if (!cin)    // bad input
        {
            cin.clear();
            while (cin.get() != '\n')
                continue;
           cout << "Bad input; input process terminated.\n";
           break;
        }
        else if (temp < 0)     // signal to terminate
            break;
```

```
        *pt = temp;
    }
    return pt;
}

// the following function can use, but not alter,
// the array whose address is ar
void Show_array(const double * beg, const double * end)
{
    using namespace std;
    const double * pt;
    int count = 0;

    for (pt = beg; pt != end; pt++, count++)
    {
        cout << *pt << " ";
        if (count % 8 == 7)
            cout << endl;
    }
    if (count % 8 != 0)
        cout << endl;
}

// reverses array contents
void Reverse_array(double * beg, double * end)
{
    double * ps;
    double * pe;
    double temp;
    for (ps = beg, pe = end - 1; ps < pe; ps++, pe--)
    {
        temp = *ps;
        *ps = *pe;
        *pe = temp;
    }
}

//pe7-8a.cpp
#include <iostream>
// constant data
const int Seasons = 4;
const char * Snames[Seasons] =
    {"Spring", "Summer", "Fall", "Winter"};

// function to modify array
void fill(double ar[], int n);
// function that uses array without modifying it
void show(const double ar[], int n);
int main()
{
    double expenses[Seasons];
    fill(expenses, Seasons);
    show(expenses, Seasons);

    // std::cin.get();
    // std::cin.get();
    return 0;
```

```
}

void fill(double ar[], int n)
{
    using namespace std;
    for (int i = 0; i < Seasons; i++)
    {
        cout << "Enter " << Snames[i] << " expenses: ";
        cin >> ar[i];
    }
}

void show(const double ar[], int n)
{
    using namespace std;
    double total = 0.0;
    cout << "\nEXPENSES\n";
    for (int i = 0; i < Seasons; i++)
    {
        cout << Snames[i] << ": $" << ar[i] << endl;
        total += ar[i];
    }
    cout << "Total Expenses: $" << total << endl;
}

//pe7-8b.cpp
#include <iostream>
// constant data
const int Seasons = 4;
const char * Snames[Seasons] =
    {"Spring", "Summer", "Fall", "Winter"};

struct hold_ar {
    double values[Seasons];
};

// function to modify structure
void fill(struct hold_ar * pha);
// function that uses structure without modifying it
void show(const struct hold_ar ha);
int main()
{
    hold_ar expenses;
    fill(&expenses);
    show(expenses);
    //std::cin.get();
    //std::cin.get();
    return 0;
}

void fill(struct hold_ar * pha)
{
    using namespace std;
    for (int i = 0; i < Seasons; i++)
    {
        cout << "Enter " << Snames[i] << " expenses: ";
        cin >> pha->values[i];
```

```
    }
}

void show(const struct hold_ar ha)
{
    using namespace std;
    double total = 0.0;
    cout << "\nEXPENSES\n";
    for (int i = 0; i < Seasons; i++)
    {
        cout << Snames[i] << ": $" << ha.values[i] << endl;
        total += ha.values[i];
    }
    cout << "Total Expenses: $" << total << endl;
}


// pe7-9.cpp

#include <iostream>
using namespace std;

const int SLEN = 30;

struct student {
    char fullname[SLEN];
    char hobby[SLEN];
    int ooplevel;
};

// getinfo() has two arguments: a pointer to the first
// element of an array of student structures and an int
// representing the number of elements of the array. The
// function solicits and stores data about students. It
// terminates input upon filling the array or upon
// encountering a blank line for the student name. The
// function returns the actual number of array elements
// filled.
int getinfo(student pa[], int n);

// display1() takes a student structure as an argument
// and displays its contents
void display1(student st);

// display2() takes the address of student structure as an
// argument and displays the structure's contents
void display2(const student * ps);

// display3() takes (1) the address of the first element of
// an array of student structures and (2) the number of
// array elements as arguments and displays the contents
// of the structures
void display3( const student pa[], int n);

int main()
{
    cout << "Enter class size: ";
```

```
    int class_size;
    cin >> class_size;
    while (cin.get() != '\n')
        continue;

    student * ptr_stu = new student[class_size];
    int entered = getinfo(ptr_stu, class_size);
    for (int i = 0; i < entered; i++)
    {
        display1(ptr_stu[i]);
        display2(&ptr_stu[i]);
    }
    display3(ptr_stu, entered);
    cout << "Done\n";
    //cin.get();
    return 0;
}

int getinfo(student pa[], int n)
{
    int i;
    for (i = 0; i < n; i++)
    {
        cout << "Enter student name: ";
        cin.getline(pa[i].fullname, SLEN);
        if (pa[i].fullname[0] == '\0')
            break;
        cout << "Enter student hobby: ";
        cin.getline(pa[i].hobby, SLEN);
        cout << "Enter student oop level: ";
        cin >> pa[i].ooplevel;
        while (cin.get() != '\n')
            continue;
    }
    return i;
}

void display1(student st)
{
    cout << st.fullname << ": " << st.hobby << ": "
         << st.ooplevel << "\n";
}

void display2(const student * ps)
{
    cout << ps->fullname << ": " << ps->hobby << ": "
         << ps->ooplevel << "\n";
}

void display3( const student pa[], int n)
{
    for (int i = 0; i < n; i++)
        cout << pa[i].fullname << ": " << pa[i].hobby << ": "
             << pa[i].ooplevel << "\n";
}
```

```
//pe7-10.cpp

#include <iostream>

double calculate(double x, double y, double (*pf)(double, double));
double add(double x, double y);
double sub(double x, double y);
double mean(double x, double y);

int main()
{
    using namespace std;
    double (*pf[3])(double,double) = {add, sub, mean};
    char * op[3] = {"sum", "difference", "mean"};
    double a, b;
    cout << "Enter pairs of numbers (q to quit): ";
    int i;
    while (cin >> a >> b)
    {
        // using function names
        cout << calculate(a, b, add) << " = sum\n";
        cout << calculate(a, b, mean) << " = mean\n";
        // using pointers
        for (i = 0; i < 3; i++)
            cout << calculate(a, b, pf[i]) << " = "
                 << op[i] << "\n";
    }
    cout << "Done!\n";
/*
   cin.clear();
    while (cin.get() != '\n')
        continue;
    cin.get();
*/
    return 0;
}

double calculate(double x, double y, double (*pf)(double, double))
{
    return (*pf)(x, y);
}

double add(double x, double y)
{
    return x + y;
}

double sub(double x, double y)
{
    return x - y;
}

double mean(double x, double y)
{
    return (x + y) / 2.0;
}
```

Chapter 8

```
// pe8-1.cpp
#include <iostream>
void silly(const char * s, int n = 0);
int main()
{
    using namespace std;
    char * p1 = "Why me?\n";

    silly(p1);
    for (int i = 0; i < 3; i++)
    {
        cout << i << " = i\n";
        silly(p1, i);
    }
    cout << "Done\n";
    // cin.get();
    return 0;
}

void silly(const char * s, int n)
{
    using namespace std;
    static int uses = 0;

    int lim = ++uses;
    if (n == 0)
        lim = 1;
    for (int i = 0; i < lim; i++)
        cout << s;
}
```

```
// pe8-2.cpp
#include <iostream>
#include <cstring>

struct CandyBar
{
    char brand[40];
    double weight;
    int calories;
};

void SetData(CandyBar & cb, const char * b = "Millenium Munch",
             double wt = 2.85, int c = 350);
void ShowData(const CandyBar & cb);
int main( )
{
    CandyBar one;
    CandyBar two;
```

```
    SetData(one);
    SetData(two, "Chockochunks", 3.5, 400);
    ShowData(one);
    ShowData(two);
    //std::cin.get();
    return 0;
}

void SetData(CandyBar & cb, const char * b, double wt, int c)
{
    using namespace std;  // for strcpy()
    strcpy(cb.brand, b);
    cb.weight = wt;
    cb.calories = c;
}

void ShowData(const CandyBar & cb)
{
    using namespace std;
    cout << "Brand: " << cb.brand << endl;
    cout << "Weight: " << cb.weight << endl;
    cout << "Calories: " << cb.calories << endl;
}


// pe8-3.cpp -- convert a string object to uppercase
#include <iostream>
#include <cctype>
#include <string>
using namespace std;

void upper_str(string & s);
int main()
{
    string input;
    cout << "Enter a string (q to quit): ";
    while (getline(cin, input) && input != "q")
    {
        upper_str(input);
        cout << input << endl;
        cout << "Next string (q to quit): ";
    }
    cout << "Bye.\n";
    //cin.get();
    return 0;
}
void upper_str(string & s)
{
    for (int i = 0; i < s.size(); i++)
    {
        s[i] = toupper(s[i]);
    }

}
```

```
// pe8-4.cpp
#include <iostream>
#include <cstring>    // for strlen(), strcpy()
using namespace std;

struct stringy {
    char * str;         // points to a string
    int ct;        // length of string (not counting '\0')
    };


void show(const char *str, int cnt =  1);
void show(const stringy & bny, int cnt = 1);
void set(stringy & bny, const char * str);

int main()
{
    stringy beany;
    char testing[] = "Reality isn't what it used to be.";

    set(beany, testing);    // first argument is a reference,
            // allocates space to hold copy of testing,
            // sets str member of beany to point to the
            // new block, copies testing to new block,
            // and sets ct member of beany
    show(beany);        // prints member string once
    show(beany, 2);    // prints member string twice
    testing[0] = 'D';
    testing[1] = 'u';
    show(testing);    // prints testing string once
    show(testing, 3);    // prints testing string thrice
    show("Done!");
    //std::cin.get();
    return 0;
}

void show(const char *str, int cnt)
{
    while(cnt-- > 0)
    {
            cout << str << endl;
    }
}

void show(const stringy & bny, int cnt)
{
    while(cnt-- > 0)
    {
            cout << bny.str << endl;
    }
}

void set(stringy & bny, const char * str)
{
    bny.ct = strlen(str);
    bny.str = new char[bny.ct+1];
```

```
    strcpy(bny.str, str);
}



// pe8-5.cpp
#include <iostream>

template <class T>
T max5(T ar[])
{     int n;
    T max = ar[0];
    for (n = 1; n < 5; n++)
        if (ar[n] > max)
            max = ar[n];
    return max;
}

const int LIMIT = 5;
int main( )
{
    using namespace std;
    double ard[LIMIT] = { -3.4, 8.1, -76.4, 34.4, 2.4};
    int ari[LIMIT] = {2, 3, 8, 1, 9};
    double md;
    int mi;

    md = max5(ard);
    mi = max5(ari);

    cout << "md = " << md << endl;
    cout << "mi = " << mi << endl;
    //cin.get();
    return 0;
}


// pe8-6.cpp

#include <iostream>
#include <cstring>

template <class T>
T maxn(T ar[], int size)
{
    int n;
    T max = ar[0];
    for (n = 1; n < size; n++)
        if (ar[n] > max)
            max = ar[n];
    return max;
}

template < > const char * maxn(const char * ar[], int size);
int main( )
{
    using namespace std;
```

```
    double ard[4] = { -3.4, 58.1, -76.4, 34.4};
    int ari[6] = {2, 3, 81, 1, 9, 22};
    const char * strs[5] = {"Here", "is", "a", "test", "sequence"};
    double md;
    int mi;
    const char * longest;

    md = maxn(ard, 4);
    mi = maxn(ari, 6);
    longest = maxn(strs, 5);

    cout << "md = " << md << endl;
    cout << "mi = " << mi << endl;
    cout << "The longest string: " << longest << endl;
    //cin.get();
    return 0;

}

template < > const char * maxn(const char * ar[], int size)
{
    int n;
    const char * max = ar[0];
    for (n = 1; n < size; n++)
        if (std::strlen(ar[n]) > std::strlen(max))
            max = ar[n];
    return max;
}


// pe8-7.cpp -- template overloading
#include <iostream>

template <typename T>            // template A
T SumArray(T arr[], int n);

template <typename T>            // template B
T SumArray(T * arr[], int n);

struct debts
{
    char name[50];
    double amount;
};

int main()
{
    using namespace std;
    int things[6] = {13, 31, 103, 301, 310, 130};
    struct debts mr_E[3] =
    {
        {"Ima Wolfe", 2400.0},
        {"Ura Foxe", 1300.0},
        {"Iby Stout", 1800.0}
    };
    double * pd[3];
```

```
    for (int i = 0; i < 3; i++)
        pd[i] = &mr_E[i].amount;

    cout << "Total of Mr. E's things:\n";
    cout << SumArray(things, 6) << endl;
    cout << "Sum of Mr. E's debts:\n";
    cout << SumArray(pd, 3)  << endl;
    //cin.get();
    return 0;
}

template <typename T>
T SumArray(T arr[], int n)
{
    T total = 0;
    std::cout << "template A\n";
    for (int i = 0; i < n; i++)
        total += arr[i];
    return total;
}

template <typename  T >
T SumArray(T * arr[], int n)
{
    T total = 0;
    std::cout << "template B\n";
    for (int i = 0; i < n; i++)
        total += *arr[i];
    return total;
}
```

Chapter 9

PE 9-1

```
// pe9-golf.h - for pe9-1.cpp
const int Len = 40;
struct golf
{
    char fullname[Len];
    int handicap;
};

// non-interactive version
// function sets golf structure to provided name, handicap
// using values passed as arguments to the function
void setgolf(golf & g, const char * name, int hc);

// interactive version
// function solicits name and handicap from user
// and sets the members of g to the values entered
// returns 1 if name is entered, 0 if name is empty string
int setgolf(golf & g);
```

```
// function resets handicap to new value
void handicap(golf & g, int hc);

// function displays contents of golf structure
void showgolf(const golf & g);


// pe9-golf.cpp - for pe9-1.cpp
#include <iostream>
#include "pe9-golf.h"
#include <cstring>

// function solicits name and handicap from user
// returns 1 if name is entered, 0 if name is empty string
int setgolf(golf & g)
{
    std::cout << "Please enter golfer's full name: ";
    std::cin.getline(g.fullname, Len);
    if (g.fullname[0] == '\0')
        return 0;               // premature termination
    std::cout << "Please enter handicap for " << g.fullname << ": ";
    while (!(std::cin >> g.handicap))
    {
        std::cin.clear();
        while (std::cin.get() !='\n')
            continue;
        std::cout << "Please enter an integer: ";
    }
    while (std::cin.get() != '\n')
        continue;
    return 1;
}

// function sets golf structure to provided name, handicap
void setgolf(golf & g, const char * name, int hc)
{
    std::strcpy(g.fullname, name);
    g.handicap = hc;
}

// function resets handicap to new value
void handicap(golf & g, int hc)
{
    g.handicap = hc;
}

// function displays contents of golf structure
void showgolf(const golf & g)
{
    std::cout << "Golfer:   " << g.fullname << "\n";
    std::cout << "Handicap: " << g.handicap << "\n\n";
}



// pe9-1.cpp
#include <iostream>
```

```
#include "pe9-golf.h"
// link with pe9-golf.cpp
const int Mems = 5;
int main()
{
    using namespace std;
    golf team[Mems];

    cout << "Enter up to " << Mems << " golf team members:\n";
    int i;
    for (i = 0; i < Mems; i++)
        if (setgolf(team[i]) == 0)
            break;
    for (int j = 0; j < i; j++)
        showgolf(team[j]);
    setgolf(team[0], "Fred Norman", 5);
    showgolf(team[0]);
    handicap(team[0], 3);
    showgolf(team[0]);
    //cin.get();
    return 0;
}
```

PE 9-2

```
// pe9-2.cpp -- using a static local variable
#include <iostream>
#include <string>
// constants

// function prototype
void strcount(const std::string & s);

int main()
{
    using namespace std;
    string input;

    cout << "Enter a line:\n";
    getline(cin, input);
    while (cin && input != "")
    {
        strcount(input);
        cout << "Enter next line (empty line to quit):\n";
        getline(cin, input);
    }
    cout << "Bye\n";
   //cin.get();
    return 0;
}

void strcount(const std::string & s)
{
    using namespace std;
    static int total = 0;        // static local variable
    int count = 0;               // automatic local variable
```

```
    cout << "\"" << s <<"\" contains ";
    count = s.size();
    total += count;
    cout << count << " characters\n";
    cout << total << " characters total\n";
}
```

PE 9-3

```
//pe9-3.cpp -- using placement new
#include <iostream>
#include <new>
#include <cstring>
struct chaff
{
    char dross[20];
    int slag;
};

// char buffer[500]; // option 1
int main()
{
    using std::cout;
    using std::endl;
    chaff *p1;
    int i;
    char * buffer = new char [500];  // option 2
    p1 = new (buffer) chaff[2];    // place structures in buffer
    std::strcpy(p1[0].dross, "Horse Feathers");
    p1[0].slag = 13;
    std::strcpy(p1[1].dross, "Piffle");
    p1[1].slag = -39;

    for (i = 0; i < 2; i++)
        cout << p1[i].dross << ": " << p1[i].slag << endl;
    delete [] buffer;  // option 2
   //std::cin.get();
    return 0;
}
```

PE 9-4

```
// pe9-4.h
#ifndef SALES__
#define SALES__

namespace SALES
{
    const int QUARTERS = 4;
    struct Sales
    {
        double sales[QUARTERS];
        double average;
        double max;
```

```
        double min;
    };

    // copies the lesser of 4 or n items from the array ar
    // to the sales member of s and computes and stores the
    // average, maximum, and minimum values of the entered items;
    // remaining elements of sales, if any, set to 0
    void setSales(Sales & s, const double ar[], int n);

    // gathers sales for 4 quarters interactively, stores them
    // in the sales member of s and computes and stores the
    // average, maximum, and minumum values
    void setSales(Sales & s);

    // display all information in structure s
    void showSales(const Sales & s);
}
#endif


// pe9-4a.cpp
#include <iostream>
#include "pe9-4.h"

int main()
{
    using SALES::Sales;
    using SALES::showSales;
    using SALES::setSales;

    Sales forFiji;
    double vals[3] = {2000, 3000, 5000};
    setSales(forFiji, vals, 3);
    showSales(forFiji);

    Sales red;
    setSales(red);
    showSales(red);
    //std::cin.get();
   //std::cin.get();
    return 0;
}


// pe9-4b.cpp
#include <iostream>
#include "pe9-4.h"

namespace SALES
{
    using std::cin;
    using std::cout;
    using std::endl;
    void setSales(Sales & s, const double ar[], int n)
    {
        if (n < 0)
            n = 0;
```

```
        int limit = n < QUARTERS ? n : QUARTERS;
        double total = 0;
        s.min = 0;
        s.max = 0;
        s.average = 0;
        if (limit > 0)
            s.min = s.max = ar[0];
        int i;
        for (i = 0; i < limit; i++)
        {
            s.sales[i] = ar[i];
            total += ar[i];
            if (ar[i] > s.max)
                s.max = ar[i];
            else if (ar[i] < s.min)
                s.min = ar[i];
        }
        for (i = limit; i < QUARTERS; i++)
            s.sales[i] = 0;
        if (limit > 0)
            s.average = total / limit;
    }

    void setSales(Sales & s)
    {
        int i;
        for (i = 0; i < QUARTERS; i++)
        {
            cout << "Enter sales for quarter " << i + 1 << ": ";
            cin >> s.sales[i];
        }

        double total = 0;
        s.min = s.max = s.sales[0];

        for (i = 0; i < QUARTERS; i++)
        {
            total += s.sales[i];
            if (s.sales[i] > s.max)
                s.max = s.sales[i];
            else if (s.sales[i] < s.min)
                s.min = s.sales[i];
        }
        s.average = total / QUARTERS;
    }

    void showSales(const Sales & s)
    {
        cout << "Sales:\n";
        for (int i = 0; i < QUARTERS; i++)
            cout << "Quarter " << i + 1 << ": $"
                 << s.sales[i] << endl;
        cout << "Average: $" << s.average << endl;
        cout << "Minimum: $" << s.min << endl;
        cout << "Maximum: $" << s.max << endl;
    }
}
```

Chapter 10

PE 10-1

```
// pe10-1.cpp
#include <iostream>
#include <cstring>

// class declaration
class BankAccount
{
private:
    char name[40];
    char acctnum[25];
    double balance;
public:
    BankAccount(char * client = "no one", char * num = "0",
                double bal = 0.0);     void show(void) const;
    void deposit(double cash);     void withdraw(double cash);
};

// method definitions
BankAccount::BankAccount(char * client, char * num, double bal)
{
    std::strncpy(name, client, 39);
    name[39] = '\0';
    std::strncpy(acctnum, num, 24);
    acctnum[24] = '\0';
    balance = bal;
}

void BankAccount::show(void) const
{
    using std::cout;
    using std:: endl;
    cout << "Client: " << name << endl;
    cout << "Account Number: " << acctnum << endl;
    cout << "Balance: " << balance << endl;
}

void BankAccount::deposit(double cash)
{
    if (cash >= 0)
        balance += cash;
    else
        std::cout << "Illegal transaction attempted";
}

void BankAccount::withdraw(double cash)
{
    if (cash < 0)
        std::cout << "Illegal transaction attempted";
    else if (cash <= balance)
        balance -=cash;
```

```
    else
        std::cout << "Request denied due to insufficient funds.\n";
}

// sample use

int main()
{
    BankAccount bird;
    BankAccount frog("Kermit", "croak322", 123.00);

    bird.show();
    frog.show();
    bird = BankAccount("Chipper", "peep8282", 214.00);
    bird.show();
    frog.deposit(20);
    frog.show();
    frog.withdraw(4000);
    frog.show();
    frog.withdraw(50);
    frog.show();
    //std::cin.get();
    return 0;
}
```

PE 10-2

```
// pe10-2.cpp
#include <iostream>
#include <string>
#include <cstring>

class Person {
private:
    static const int LIMIT = 25;
    std::string lname;       // Person's last name
    char fname[LIMIT];  // Person's first name
public:
    Person() {lname = ""; fname[0] = '\0';  }
    Person(const std::string & ln, const char * fn = "Heyyou");
// the following methods display lname and fname
    void Show() const;        // firstname lastname format
    void FormalShow() const;  // lastname, firstname format
};

Person::Person(const std::string & ln, const char * fn)
{
    lname = ln;
    std::strncpy(fname, fn, LIMIT - 1);
    fname[LIMIT - 1] = '\0';
}

void Person::Show()const
{
    std::cout << fname << ' ' << lname;
```

```
}

void Person::FormalShow() const
{
    std::cout << lname << ", " << fname;
}

int main()
{
    using std::cout;
    using std::endl;
    Person one;
    Person two("Smythecraft");
    Person three("Dimwiddy", "Sam");
    one.Show();
    cout << endl;
    one.FormalShow();
    cout << endl;
    two.Show();
    cout << endl;
    two.FormalShow();
    cout << endl;
    three.Show();
    cout << endl;
    three.FormalShow();
    cout << endl;
    //std::cin.get();
    return 0;
}
```

PE 10-3

```
// pe10-golf.h - for pe10-3.cpp

#ifndef PE10_GOLF_H_
#define PE10_GOLF_H_

const int Len = 40;
class golf
{
private:
    char fullname[Len];
    int handicap_;
public:
// constructor sets golf structure to provided name, handicap
    golf(const char * name = "no one", int hc = -100);
//  method solicits name and handicap from user
//  returns 1 if name is entered, 0 if name is empty string
    int setgolf();
// reset handicap to new value
    void handicap(int hc);
// display contents of golf objcet
    void showgolf() const;
};
```

```
#endif


// pe10-golf.cpp - for pe10-3.cpp
#include <iostream>
#include "pe10-golf.h"
#include <cstring>

golf::golf(const char * name, int hc)
{
    std::strcpy(fullname, name);
    handicap_ = hc;
}

int golf::setgolf()
{
    char fname[Len];
    std::cout << "Please enter golfer's full name: ";
    std::cin.getline(fname, Len);
    if (fname[0] == '\0')
        return 0;                // premature termination
    std::cout << "Please enter handicap for " << fname << ": ";
    int hc;
    while (!(std::cin >> hc))
    {
        std::cin.clear();
        while (std::cin.get() !='\n')
            continue;
        std::cout << "Please enter an integer: ";
    }
    while (std::cin.get() != '\n')
        continue;
    *this = golf(fname, hc); // use constructor
    return 1;
}


void golf::handicap(int hc)
{
    handicap_ = hc;
}

void golf::showgolf() const
{
    std::cout << "Golfer:   " << fullname << "\n";
    std::cout << "Handicap: " << handicap_ << "\n\n";
}


// pe10-3.cpp
#include <iostream>
#include "pe10-golf.h"
// link with pe10-golf.cpp
const int Mems = 5;
int main()
{
    golf team[Mems];
```

```
    std::cout << "Enter up to " << Mems << " golf team members:\n";
    int i;
    for (i = 0; i < Mems; i++)
        if (team[i].setgolf() == 0)
            break;
    for (int j = 0; j < i; j++)
        team[j].showgolf();
    team[0] = golf("Fred Norman", 5);
    team[0].showgolf();
    team[0].handicap(3);
    team[0].showgolf();
    //std::cin.get();
    return 0;
}
```

PE10-4

```
// pe10-4.h -- define Sales class
#ifndef SALES__
#define SALES__

namespace SALES
{
    const int QUARTERS = 4;
    class Sales
    {
    private:
        double sales[QUARTERS];
        double average;
        double max;
        double min;
    public:
    // default constructor
        Sales();

    // copies the lesser of 4 or n items from the array ar
    // to the sales member and computes and stores the
    // average, maximum, and minimum values of the entered items;
    // remaining elements of sales, if any, set to 0
        Sales(const double ar[], int n);

    // gathers sales for 4 quarters interactively, stores them
    // in the sales member of object and computes and stores the
    // average, maximum, and minumum values
        void setSales();

    // display all information in object
        void showSales();
    };

}

#endif
```

```
// pe10-4a.cpp  -- use Sales class
#include <iostream>
#include "pe10-4.h"

int main()
{
    using SALES::Sales;

    double vals[3] = {2000, 3000, 5000};
    Sales forFiji(vals, 3);
    forFiji.showSales();

    Sales red;
    red.showSales();
    red.setSales();
    red.showSales();
    //std::cin.get();
    //std::cin.get();
    return 0;
}


// pe10-4b.cpp -- implement Sales class
#include <iostream>
#include "pe10-4.h"

namespace SALES
{
    using std::cin;
    using std::cout;
    using std::endl;
    Sales::Sales(const double ar[], int n)
    {
        if (n < 0)
            n = 0;
        int limit = n < QUARTERS ? n : QUARTERS;
        double total = 0;
        min = 0;
        max = 0;
        average = 0;
        if (limit > 0)
            min = max = ar[0];
        int i;
        for (i = 0; i < limit; i++)
        {
            sales[i] = ar[i];
            total += ar[i];
            if (ar[i] > max)
                max = ar[i];
            else if (ar[i] < min)
                min = ar[i];
        }
        for (i = limit; i < QUARTERS; i++)
            sales[i] = 0;
        if (limit > 0)
```

```
            average = total / limit;
    }

    Sales::Sales()
    {

        min = 0;
        max = 0;
        average = 0;
        for (int i = 0; i < QUARTERS; i++)
            sales[i] =0;
    }

    void Sales::setSales()
    {
        double sa[QUARTERS];
        int i;
        for (i = 0; i < QUARTERS; i++)
        {
            cout << "Enter sales for quarter " << i + 1 << ": ";
            cin >> sa[i];
        }

        // create temporary object, copy to invoking object
        *this = Sales(sa, QUARTERS);
    }

    void Sales::showSales()
    {
        cout << "Sales:\n";
        for (int i = 0; i < QUARTERS; i++)
            cout << "Quarter " << i + 1 << ": $"
                 << sales[i] << endl;
        cout << "Average: $" << average << endl;
        cout << "Minimum: $" << min << endl;
        cout << "Maximum: $" << max << endl;
    }
}
```

PE 10-5

```
// pe10stack.h -- class definition for the stack ADT
// for use with pe10-5.cpp
#ifndef _STACK_H_
#define _STACK_H_

struct customer {
    char fullname[35];
    double payment;
};

typedef customer Item;

class Stack
{
```

```
private:
    enum {MAX = 10};    // constant specific to class
    Item items[MAX];    // holds stack items
    int top;        // index for top stack item
public:
    Stack();
    bool isempty() const;
    bool isfull() const;
    // push() returns false if stack already is full, true otherwise
    bool push(const Item & item);    // add item to stack
    // pop() returns false if stack already is empty, true otherwise
    bool pop(Item & item);    // pop top into item
};
#endif


// pe10stack.cpp -- Stack member functions
// for use with pe10-5.cpp
// exactly the same as stack.cpp in the text
// except that it includes pe10stack.h

#include "pe10stack.h"
Stack::Stack()    // create an empty stack
{
    top = 0;
}

bool Stack::isempty() const
{
    return top == 0;
}

bool Stack::isfull() const
{
    return top == MAX;
}

bool Stack::push(const Item & item)
{
    if (top < MAX)
    {
        items[top++] = item;
        return true;
    }
    else
        return false;
}

bool Stack::pop(Item & item)
{
    if (top > 0)
    {
        item = items[--top];
        return true;
    }
    else
        return false;
```

```
}


// pe10-5.cpp

#include <iostream>
#include <cctype>
#include "pe10stack.h"     // modified to define customer structure
// link with pe10stack.cpp
void get_customer(customer & cu);
int main()
{
    using namespace std;
    Stack st; // create a stack of customer structures
    customer temp;
    double payments = 0;
    char c;

    cout << "Please enter A to add a customer,\n"
         << "P to process a customer, and Q to quit.\n";
    while (cin >> c && (c = toupper(c)) != 'Q')
    {
        while (cin.get() != '\n')
            continue;
        if (c != 'A' && c != 'P')
        {
            cout << "Please respond with A, P, or Q: ";
            continue;
        }
        switch (c)
        {
            case 'A'    :    if (st.isfull())
                                cout << "stack already full\n";
                             else
                             {
                                 get_customer(temp);
                                 st.push(temp);
                             }
                             break;
            case 'P'    :    if (st.isempty())
                                cout << "stack already empty\n";
                             else {
                                 st.pop(temp);
                                 payments += temp.payment;
                                 cout << temp.fullname << " processed. ";
                                 cout << "Payments now total $"
                                      << payments << "\n";
                             }
                             break;
            default     :    cout << "Whoops! Programming error!\n";
        }
        cout << "Please enter A to add a customer,\n"
                << "P to process a customer, and Q to quit.\n";
    }
    cout << "Done!\n";
    //cin.get();
    //cin.get();
```

```
    return 0;
}

void get_customer(customer & cu)
{
   using namespace std;
    cout << "Enter customer name: ";
    cin.getline(cu.fullname,35);
    cout << "Enter customer payment: ";
    cin >> cu.payment;
    while (cin.get() != '\n')
        continue;
}
```

PE 10-6

```
//pe10-6.cpp
#include <iostream>

class Move{
private:
    double x;
    double y;
public:
    Move(double a = 0, double b = 0) {x = a; y = b;};
    void show() { std::cout << "(x,y) = (" << x << ','
                            << y << ")\n";}
    void reset(double a = 0, double b = 0) {x = a; y = b;};
    Move add(Move & m);
};

Move Move::add(Move & m)
{
    double nx = x + m.x;
    double ny = y + m.y;
    Move xy(nx, ny);
    return xy;
}

int main()
{
    Move a(10.0, 20.0);
    Move b(2.5, 3.5);
    a.show();
    b.show();
    a.add(b).show();
    //std::cin.get();
    return 0;
}
```

PE 10-7

```
// pe10-7.cpp
```

```
#include <iostream>
#include <cstring>
using namespace std;

class Plorg
{
private:
    char name[20];
    int CI;
public:
    Plorg(const char * str = "Plorga");
    void alterCI(int n);
    void report()const;
};

Plorg::Plorg(const char * str)
{
    strncpy(name, str, 19);
    name[19] = '\0';
    CI = 50;    // All Plorgs are created equal!
}

void Plorg::alterCI(int n)
{
    CI = n;
}

void Plorg::report() const
{
    cout << "I am a Plorg! My name is " << name
         << " and my CI is " << CI << ".\n";
}

// sample use

int main()
{
    Plorg p;
    Plorg vp("Vargul Proplorg");

    p.report();
    vp.report();
    vp.alterCI(83);
    vp.report();
    //std::cin.get();
    return 0;
}
```

PE 10-8

```
// pe10-8arr.h -- header file for a simple list class

#ifndef SIMPLEST_
#define SIMPLEST_
```

```
// program-specific declarations
const int TSIZE = 45;      // size of array to hold title
struct film
{
    char title[TSIZE];
    int rating;
};

// general type definitions
typedef struct film Item;

const int MAXLIST = 10;
class simplist
{
private:
    Item items[MAXLIST];
    int count;
public:
    simplist(void);
 bool isempty(void);
 bool isfull(void);
    int itemcount();
 bool additem(Item item);
    void transverse( void (*pfun)(Item item));
};

#endif


// pe10-8arr.cpp -- functions supporting simple list operations
#include "pe10-8arr.h"

simplist::simplist(void)
{
    count = 0;
}

bool simplist::isempty(void)
{
    return count == 0;
}

bool simplist::isfull(void)
{
    return count == MAXLIST;
}

int simplist::itemcount()
{
    return count;
}
bool simplist::additem(Item item)
{
    if (count == MAXLIST)
        return false;
    else
        items[count++] = item;
```

```
    return true;
}

void simplist::transverse( void (*pfun)(Item item))
{
    for (int i = 0; i < count; i++)
        (*pfun)(items[i]);
}


// pe10-8.cpp -- using a class definition

#include <iostream>
#include <cstdlib>         // prototype for exit()
#include "pe10-8arr.h"     // simple list class declaration
                                 // array version
void showmovies(Item item); // to be used by transverse()

int main()
{
    using namespace std;
    simplist movies;     // creates an empty list
    Item temp;

    if (movies.isfull())    // invokes isfull() member function
    {
        cout << "No more room in list! Bye!\n";
        exit(1);
    }
    cout << "Enter first movie title:\n";
    while (cin.getline(temp.title,TSIZE) && temp.title[0] != '\0')
    {
        cout << "Enter your rating <0-10>: ";
        cin >> temp.rating;
        while(cin.get() != '\n')
            continue;
        if (movies.additem(temp) == false)
        {
            cout << "List already is full!\n";
            break;
        }
        if (movies.isfull())
        {
            cout << "You have filled the list.\n";
            break;
        }
        cout << "Enter next movie title (empty line to stop):\n";
    }
    if (movies.isempty())
        cout << "No data entered. ";
    else
    {
        cout << "Here is the movie list:\n";
        movies.transverse(showmovies);
    }
    cout << "Bye!\n";
    //cin.get();
```

```
    return 0;
}

void showmovies(Item item)
{
        std::cout << "Movie: " << item.title << "  Rating: "
             << item.rating << std::endl;
}
```

Chapter 11

PE 11-1

```
// vect.h -- Vector class with <<, mode state
#ifndef VECTOR_H_
#define VECTOR_H_
#include <iostream>
namespace VECTOR
{
    class Vector
    {
    public:
        enum Mode {RECT, POL};
    // RECT for rectangular, POL for Polar modes
    private:
        double x;          // horizontal value
        double y;          // vertical value
        double mag;        // length of vector
        double ang;        // direction of vector in degrees
        Mode mode;         // RECT or POL
    // private methods for setting values
        void set_mag();
        void set_ang();
        void set_x();
        void set_y();
    public:
       Vector();
        Vector(double n1, double n2, Mode form = RECT);
        void reset(double n1, double n2, Mode form = RECT);
        ~Vector();
        double xval() const {return x;}      // report x value
        double yval() const {return y;}      // report y value
        double magval() const {return mag;}  // report magnitude
        double angval() const {return ang;}  // report angle
        void polar_mode();                   // set mode to POL
        void rect_mode();                    // set mode to RECT
    // operator overloading
        Vector operator+(const Vector & b) const;
        Vector operator-(const Vector & b) const;
        Vector operator-() const;
        Vector operator*(double n) const;
    // friends
        friend Vector operator*(double n, const Vector & a);
        friend std::ostream & operator<<(std::ostream & os,
                                         const Vector & v);
```

```
    };

}   // end namespace VECTOR
#endif

// vect.cpp -- methods for the Vector class
#include <cmath>
#include "vect.h"   // includes <iostream>
using std::sqrt;
using std::sin;
using std::cos;
using std::atan;
using std::atan2;
using std::cout;

namespace VECTOR
{
    // compute degrees in one radian
    const double Rad_to_deg = 45.0 / atan(1.0);
    // should be about 57.2957795130823

    // private methods
    // calculates magnitude from x and y
    void Vector::set_mag()
    {
        mag = sqrt(x * x + y * y);
    }

    void Vector::set_ang()
    {
        if (x == 0.0 && y == 0.0)
            ang = 0.0;
        else
            ang = atan2(y, x);
    }

    // set x from polar coordinate
    void Vector::set_x()
    {
        x = mag * cos(ang);
    }

    // set y from polar coordinate
    void Vector::set_y()
    {
        y = mag * sin(ang);
    }

    // public methods
    Vector::Vector()             // default constructor
    {
        x = y = mag = ang = 0.0;
        mode = RECT;
    }

    // construct vector from rectangular coordinates if form is r
    // (the default) or else from polar coordinates if form is p
```

```
Vector::Vector(double n1, double n2, Mode form)
{
    mode = form;
    if (form == RECT)
     {
         x = n1;
         y = n2;
         set_mag();
         set_ang();
    }
    else if (form == POL)
    {
         mag = n1;
         ang = n2 / Rad_to_deg;
         set_x();
         set_y();
    }
    else
    {
         cout << "Incorrect 3rd argument to Vector() -- ";
         cout << "vector set to 0\n";
         x = y = mag = ang = 0.0;
         mode = RECT;
    }
}

// reset vector from rectangular coordinates if form is
// RECT (the default) or else from polar coordinates if
// form is POL
void Vector:: reset(double n1, double n2, Mode form)
{
    mode = form;
    if (form == RECT)
     {
         x = n1;
         y = n2;
         set_mag();
         set_ang();
    }
    else if (form == POL)
    {
         mag = n1;
         ang = n2 / Rad_to_deg;
         set_x();
         set_y();
    }
    else
    {
         cout << "Incorrect 3rd argument to Vector() -- ";
         cout << "vector set to 0\n";
         x = y = mag = ang = 0.0;
         mode = RECT;
    }
}

Vector::~Vector()    // destructor
{
```

```
}

void Vector::polar_mode()    // set to polar mode
{
    mode = POL;
}

void Vector::rect_mode()     // set to rectangular mode
{
    mode = RECT;
}

// operator overloading
// add two Vectors
Vector Vector::operator+(const Vector & b) const
{
    return Vector(x + b.x, y + b.y);
}

// subtract Vector b from a
Vector Vector::operator-(const Vector & b) const
{
    return Vector(x - b.x, y - b.y);
}

// reverse sign of Vector
Vector Vector::operator-() const
{
    return Vector(-x, -y);
}

// multiply vector by n
Vector Vector::operator*(double n) const
{
    return Vector(n * x, n * y);
}

// friend methods
// multiply n by Vector a
Vector operator*(double n, const Vector & a)
{
    return a * n;
}

// display rectangular coordinates if mode is RECT,
// else display polar coordinates if mode is POL
std::ostream & operator<<(std::ostream & os, const Vector & v)
{
    if (v.mode == Vector::RECT)
         os << "(x,y) = (" << v.x << ", " << v.y << ")";
    else if (v.mode == Vector::POL)
    {
         os << "(m,a) = (" << v.mag << ", "
             << v.ang * Rad_to_deg << ")";
    }
    else
         os << "Vector object mode is invalid";
```

```
        return os;
    }
}  // end namespace VECTOR


// pe11-1.cpp -- use the Vector class
// compile with the vect.cpp file
#include <iostream>
#include <fstream>
#include <cstdlib>      // rand(), srand() prototypes
#include <ctime>        // time() prototype
#include "vect.h"
int main()
{
    using namespace std;
    using VECTOR::Vector;
    srand(time(0));     // seed random-number generator
    double direction;
    Vector step;
    Vector result(0.0, 0.0);
    unsigned long steps = 0;
    double target;
    double dstep;
    ofstream fout;
    fout.open("thewalk.txt");
    if (!fout.is_open())
    {
        cerr << "Can't open output file. Bye.\n";
        exit(EXIT_FAILURE);
    }
    cout << "Enter target distance (q to quit): ";
    while (cin >> target)
    {
        cout << "Enter step length: ";
        if (!(cin >> dstep))
            break;
        fout << "Target Distance: " << target
            << ", Step Size: " << dstep << endl;
        fout << steps << ": " << result << endl;
        while (result.magval() < target)
        {
            direction = rand() % 360;
            step.reset(dstep, direction, Vector::POL);
            result = result + step;
            steps++;
            fout << steps << ": " << result << endl;
        }
        cout << "After " << steps << " steps, the subject "
            "has the following location:\n";
        cout << result << endl;
        fout << "After " << steps << " steps, the subject "
            "has the following location:\n";
        fout << result << endl;
        result.polar_mode();
        cout << " or\n" << result << endl;
        cout << "Average outward distance per step = "
            << result.magval()/steps << endl;
```

```
        fout << " or\n" << result << endl;
        fout << "Average outward distance per step = "
            << result.magval()/steps << endl << endl;
        steps = 0;
        result.reset(0.0, 0.0);
        cout << "Enter target distance (q to quit): ";
    }
    cout << "Bye!\n";
    fout.close();
/* keep window open
    cin.clear();
    while (cin.get() != '\n')
        continue;
    cin.get();
*/
    return 0;
}
```

PE 11-2

```
// pe11-2.h -- Vector class with <<, mode state
// modified implementation
#ifndef MODVECTOR_H_
#define MODVECTOR_H_
#include <iostream>
namespace VECTOR
{
    class Vector
    {
    public:
        enum Mode {RECT, POL};
    // RECT for rectangular, POL for Polar modes
    private:
        double x;          // horizontal value
        double y;          // vertical value
        Mode mode;         // RECT = rectangular, POL = polar
    // private methods for setting values
        void set_mag();
        void set_ang();
        void set_x(double, double);
        void set_y(double, double);
    public:
        Vector();
        Vector(double n1, double n2, Mode form = RECT);
        void reset(double n1, double n2, Mode form = RECT);
        ~Vector();
        double xval() const {return x;}    // report x value
        double yval() const {return y;}    // report y value
        double magval() const;             // report magnitude
        double angval() const;             // report angle
        void polar_mode();                 // set mode to POL
        void rect_mode();                  // set mode to RECT
    // operator overloading
        Vector operator+(const Vector & b) const;
        Vector operator-(const Vector & b) const;
```

```
        Vector operator-() const;
        Vector operator*(double n) const;
    // friends
        friend Vector operator*(double n, const Vector & a);
        friend std::ostream & operator<<(std::ostream & os,
                                         const Vector & v);
    };

}   // end namespace VECTOR
#endif


// pe11-2.cpp -- modified methods for Vector class
#include <cmath>
#include "pe11-2.h"   // includes <iostream>
using std::sqrt;
using std::sin;
using std::cos;
using std::atan;
using std::atan2;
using std::cout;

namespace VECTOR
{
    // compute degrees in one radian
    const double Rad_to_deg = 45.0 / atan(1.0);
    // should be about 57.2957795130823

    // private methods
    // calculates magnitude from x and y


    // set x from polar coordinate
    void Vector::set_x(double mag, double ang)
    {
        x = mag * cos(ang);
    }

    // set y from polar coordinate
    void Vector::set_y(double mag, double ang)
    {
        y = mag * sin(ang);
    }

    // public methods
    Vector::Vector()             // default constructor
    {
        x = y  = 0.0;
        mode = RECT;
    }

    // construct vector from rectangular coordinates if form is r
    // (the default) or else from polar coordinates if form is p
    Vector::Vector(double n1, double n2, Mode form)
    {
        mode = form;
        if (form == RECT)
```

```
     {
          x = n1;
          y = n2;
     }
     else if (form == POL)
     {
          set_x(n1, n2 / Rad_to_deg);
          set_y(n1, n2 / Rad_to_deg);
     }
     else
     {
          cout << "Incorrect 3rd argument to Vector() -- ";
          cout << "vector set to 0\n";
          x = y = 0.0;
          mode = RECT;
     }
}

// set vector from rectangular coordinates if form is RECT (the
// default) or else from polar coordinates if form is POL
void Vector::reset(double n1, double n2, Mode form)
{
     mode = form;
     if (form == RECT)
      {
          x = n1;
          y = n2;
     }
     else if (form ==POL)
     {
          set_x(n1, n2 / Rad_to_deg);
          set_y(n1, n2 / Rad_to_deg);
     }
     else
     {
          cout << "Incorrect 3rd argument to Vector() -- ";
          cout << "vector set to 0\n";
          x = y = 0.0;
          mode = RECT;
     }
}

Vector::~Vector()    // destructor
{
}

double Vector::magval() const            // report magnitude
{
     return sqrt(x*x +y*y);
}

double Vector::angval() const              // report angle
{
     if (x == 0.0 && y == 0.0)
         return 0;
     else
         return atan2(y, x);
```

```
}

void Vector::polar_mode()    // set to polar mode
{
    mode = POL;
}

void Vector::rect_mode()     // set to rectangular mode
{
    mode = RECT;
}

// operator overloading
// add two Vectors
Vector Vector::operator+(const Vector & b) const
{
    return Vector(x + b.x, y + b.y);
}

// subtract Vector b from a
Vector Vector::operator-(const Vector & b) const
{
    return Vector(x - b.x, y - b.y);
}

// reverse sign of Vector
Vector Vector::operator-() const
{
    return Vector(-x, -y);
}

// multiple vector by n
Vector Vector::operator*(double n) const
{
    return Vector(n * x, n * y);
}

// friend methods
// multiply n by Vector a
Vector operator*(double n, const Vector & a)
{
    return a * n;
}

// display rectangular coordinates if mode is r,
// else display polar coordinates if mode is p
std::ostream & operator<<(std::ostream & os, const Vector & v)
{
    if (v.mode == Vector::RECT)
         os << "(x,y) = (" << v.x << ", " << v.y << ")";
    else if (v.mode == Vector::POL)
    {
         os << "(m,a) = (" << v.magval() << ", "
             << v.angval() * Rad_to_deg << ")";
    }
    else
         os << "Vector object mode is invalid";
```

```
        return os;
    }

}  // end namespace VECTOR


// pe11-2walk.cpp -- use the modified Vector class
// compile with the pe11-2.cpp file
#include <iostream>
#include <cstdlib>      // rand(), srand() prototypes
#include <ctime>        // time() prototype
#include "pe11-2.h"

int main()
{
    using namespace std;
    using VECTOR::Vector;
    srand(time(0));     // seed random-number generator
    double direction;
    Vector step;
    Vector result(0.0, 0.0);
    unsigned long steps = 0;
    double target;
    double dstep;
    cout << "Enter target distance (q to quit): ";
    while (cin >> target)
    {
        cout << "Enter step length: ";
        if (!(cin >> dstep))
            break;

        while (result.magval() < target)
        {
            direction = rand() % 360;
            step.reset(dstep, direction, Vector::POL);
            result = result + step;
            steps++;
        }
        cout << "After " << steps << " steps, the subject "
            "has the following location:\n";
        cout << result << endl;
        result.polar_mode();
        cout << " or\n" << result << endl;
        cout << "Average outward distance per step = "
            << result.magval()/steps << endl;
        steps = 0;
        result.reset(0.0, 0.0);
        cout << "Enter target distance (q to quit): ";
    }
    cout << "Bye!\n";
/* keep window open
    cin.clear();
    while (cin.get() != '\n')
        continue;
    cin.get();
*/
    return 0;
```

```
}
```

PE 11-3

See PE 11-1 for vect.h, vect.cpp

```
// pe11-3.cpp -- use the Vector class
// compile with the vect.cpp file
// place original calculation inside a loop

#include <iostream>
#include <ctime>
#include <stdlib.h>     // rand(), srand() prototypes
#include "vect.h"
using namespace VECTOR;

int main()
{
    using namespace std;
    srand(time(0));        // seed random-number generator
    cout << "Hello!\n";
    double direction;
    Vector step;
    Vector result(0.0, 0.0);
    unsigned long steps = 0;
    double target;
    double dstep;
    unsigned long trials;
    unsigned long min, max, average;
    cout << "Enter target distance (q to quit): ";
    while (cin >> target)
    {
        cout << "Enter step length: ";
        if (!(cin >> dstep))
            break;
        cout << "Enter number of trials: ";
        if (!(cin >> trials))
            break;
        average = 0;
        for (int n = 0; n < trials; n++)    // new loop
        {
            while (result.magval() < target)
            {
                direction = rand() % 360;
                step.reset(dstep, direction, Vector::POL);
                result = result + step;
                steps++;
            }
            if ( n == 0)
                min = max = steps;
            else
            {
                if (steps > max) max = steps;
                if (steps < min) min = steps;
            }
```

```
            average += steps;
            steps = 0;
            result.reset(0.0, 0.0);
        }                           // end new loop
        if (trials < 1)
        {
            cout << "No trials\n";
            continue;
        }
        average /= trials;

        cout << "Trials: " << trials << '\n';
        cout << "Average number of steps: " << average << '\n';
        cout << "Minimum number of steps: " << min << '\n';
        cout << "Maximum number of steps: " << max << '\n';
        steps = 0;
        result.reset(0.0, 0.0);
        cout << "Enter target distance (q to quit): ";
    }
    cout << "Bye!\n";
/* keep window open
   cin.clear();
    while (cin.get() != '\n')
        continue;
    cin.get();
*/
    return 0;
}
```

PE 11-4

```
// pe11-4tm.h -- Time class with many friends
#ifndef PE11_4_H_
#define PE11_4_H_
#include <iostream>

class Time
{
private:
    int hours;
    int minutes;
public:
    Time();
    Time(int h, int m = 0);
    void AddMin(int m);
    void AddHr(int h);
    void Reset(int h = 0, int m = 0);
    friend Time operator+(const Time & t1,const Time & t2);
    friend Time operator-(const Time & t1,const Time & t2);
    friend Time operator*(const Time & t, double n);
    friend Time operator*(double m, const Time & t)
           { return t * m; }   // inline definition
    friend std::ostream & operator<<(std::ostream & os, const Time & t);
};
```

```
#endif

// pe11-4tm.cpp  -- implement Time methods
#include "pe11-4tm.h"

Time::Time()
{
    hours = minutes = 0;
}

Time::Time(int h, int m )
{
    hours = h;
    minutes = m;
}

void Time::AddMin(int m)
{
    minutes += m;
    hours += minutes / 60;
    minutes %= 60;
}
void Time::AddHr(int h)
{
    hours += h;
}

void Time::Reset(int h, int m)
{
    hours = h;
    minutes = m;
}

Time operator+(const Time & t1, const Time & t2)
{
    Time sum;
    sum.minutes = t1.minutes + t2.minutes;
    sum.hours = t1.hours + t2.hours + sum.minutes / 60;
    sum.minutes %= 60;
    return sum;
}

Time operator-(const Time & t1,const Time & t2)
{
    Time diff;
    int tot1, tot2;
    tot1 = t1.minutes + 60 * t1.hours;
    tot2 = t2.minutes + 60 * t2.hours;
    diff.minutes = (tot2 - tot1) % 60;
    diff.hours = (tot2 - tot1) / 60;
    return diff;
}

Time operator*(const Time & t, double mult)
{
    Time result;
    long totalminutes = t.hours * mult * 60 + t.minutes * mult;
```

```
    result.hours = totalminutes / 60;
    result.minutes = totalminutes % 60;
    return result;
}

std::ostream & operator<<(std::ostream & os, const Time & t)
{
    os << t.hours << " hours, " << t.minutes << " minutes";
    return os;
}

// pe11-4.cpp -- use fifth draft of Time class
// compile pe11-4.cpp and pe11-4tm.cpp together
#include <iostream>
#include "pe11-4tm.h"

int main()
{
    using std::cout;
    using std::endl;
    Time A;
    Time B(5, 40);
    Time C(2, 55);

    cout << "A, B, and C:\n";
    cout << A <<"; " << B << ": " << C << endl;
    A = B + C;     // operator+()
    cout << "B + C: " << A << endl;
    A = B * 2.75;  // member operator*()
    cout << "B * 2.75: " << A << endl;
    cout << "10 * B: " << 10 * B << endl;
    /std::cin.get();
    return 0;
}
```

PE 11-5

```
// pe11ston.h -- definition for Stonewt class (for pe 11-5)
#ifndef PE11STONEWT_H_
#define PE11STONEWT_H_
#include <iostream>
class Stonewt
{
private:
    enum {Lbs_per_stn = 14};    // pounds per stone
    int stone;       // whole stones
    double pds_left; // fractional pounds
    double pounds;   // entire weight in pounds
    char mode;       // display mode for weight
                     // 's' = stone, 'f' = float, 'w' = whole pounds
public:
    Stonewt(double lbs);    // constructor for double pounds
    Stonewt(int stn, double lbs);    // constructor for stone, lbs
    Stonewt();    // default constructor
    ~Stonewt();
    void set_mode(char m) {mode = m; }
```

```
    Stonewt operator+(const Stonewt & sw) const;
    Stonewt operator-(const Stonewt & sw) const;
    Stonewt operator*(double m) const;
    friend Stonewt operator*(double m, const Stonewt & sw)
        { return sw * m; }
    friend std::ostream & operator<<(std::ostream & os, const Stonewt & sw);
};
#endif


// pe11ston.cpp -- Stonewt class methods (for pe 11-5)
#include <iostream>
#include "pe11ston.h"

// construct Stonewt object from double value
Stonewt::Stonewt(double lbs)
{
    stone = int (lbs) / Lbs_per_stn;    // integer division
    pds_left = int (lbs) % Lbs_per_stn + lbs - int(lbs);
    pounds = lbs;
    mode = 'f';
}

// construct Stonewt object from stone, double values
Stonewt::Stonewt(int stn, double lbs)
{
    stone = stn;
    pds_left = lbs;
    pounds =  stn * Lbs_per_stn +lbs;
    mode = 's';
}

Stonewt::Stonewt()    // default constructor, wt = 0
{
    stone = pounds = pds_left = 0;
    mode = 's';
}

Stonewt::~Stonewt()    // destructor
{
}

std::ostream & operator<<(std::ostream & os, const Stonewt & sw)
{
// show weight in stones
    if (sw.mode == 's')
        os << sw.stone << " stone, " << sw.pds_left << " pounds\n";
// show weight in pounds
    else if (sw.mode == 'f')
        os << sw.pounds << " pounds\n";
// show weight in whole pounds
    else if (sw.mode == 'w')
        os << (int) sw.pounds << " pounds\n";
    else
        os << "Programming flaw in operator<<()\n";
    return os;
}
```

```
Stonewt Stonewt::operator+(const Stonewt & sw) const
{
    double wt = pounds + sw.pounds;
    Stonewt temp(wt);
    return temp;
}

Stonewt Stonewt::operator-(const Stonewt & sw) const
{
    double wt = pounds - sw.pounds;
    Stonewt temp(wt);
    return temp;
}

Stonewt Stonewt::operator*(double m) const
{
    double wt = m * pounds;
    Stonewt temp(wt);
    return temp;
}


// pe11-5.cpp
#include <iostream>
#include "pe11ston.h"
// link with pe11ston.cpp
int main()
{
     using std::cout;
     Stonewt fullback(245.5);
     Stonewt cornerback(13, 5.2);
     cout << fullback;
     cout << cornerback;
     cornerback.set_mode('w');
     cout << cornerback;
     Stonewt lump;
     lump = fullback + cornerback;
     cout << lump;
     fullback = fullback * 1.1;
     cout << fullback;
     lump = lump - fullback;
     cout << lump;
     lump = 1.3 * lump;
     lump.set_mode('s');
     cout << lump;
     //std::cin.get();
     return 0;
}
```

PE 11-6

```
// pe11stn6.h -- definition for Stonewt class
#ifndef _STONEWT_H_
#define _STONEWT_H_
```

```
class Stonewt
{
private:
    enum {Lbs_per_stn = 14};      // pounds per stone
    int stone;                    // whole stones
    double pds_left;              // fractional pounds
    double pounds;                // entire weight in pounds
public:
    Stonewt(double lbs);          // constructor for double pounds
    Stonewt(int stn, double lbs); // constructor for stone, lbs
    Stonewt();                    // default constructor
    ~Stonewt();
    void show_lbs() const;        // show weight in pounds format
    void show_stn() const;        // show weight in stone format
    bool operator>(const Stonewt &s) const
                                  { return pounds > s.pounds; }
    bool operator==(const Stonewt &s) const
                                  { return pounds == s.pounds; }
// for consistency, define rest in terms of > and ==
    bool operator<(const Stonewt &s) const { return s > *this; }
    bool operator!=(const Stonewt &s) const { return !(s == *this); }
    bool operator<=(const Stonewt &s) const { return !(*this > s); }
    bool operator>=(const Stonewt &s) const { return !(s > *this); }
};
#endif

// pe11stn6.cpp
#include <iostream>
#include "pe11stn6.h"

// construct Stonewt object from double value
Stonewt::Stonewt(double lbs)
{
    stone = int (lbs) / Lbs_per_stn;    // integer division
    pds_left = int (lbs) % Lbs_per_stn + lbs - int(lbs);
    pounds = lbs;
}

// construct Stonewt object from stone, double values
Stonewt::Stonewt(int stn, double lbs)
{
    stone = stn;
    pds_left = lbs;
    pounds =  stn * Lbs_per_stn +lbs;
}

Stonewt::Stonewt()       // default constructor, wt = 0
{
    stone = pounds = pds_left = 0;
}

Stonewt::~Stonewt()      // destructor
{
}

// show weight in stones
void Stonewt::show_stn() const
```

```
{
    std::cout << stone << " stone, " << pds_left << " pounds\n";
}

// show weight in pounds
void Stonewt::show_lbs() const
{
    std::cout << pounds << " pounds\n";
}


// pe11-6.cpp
#include <iostream>
#include "pe11stn6.h"
// compile with pe11stn6.cpp
const int STAFF = 6;
int main()
{
    using std::cout;
    using std::cin;
    using std::endl;

    Stonewt sales[STAFF]=
    {
        Stonewt(12, 4),
        Stonewt(10,6),
        Stonewt(9, 4)
    };
    double pounds;
    cout << "Enter the weight, in pounds, of the next "
         << "3 staff members:\n";

    int i;
    for (i = 3; i < STAFF; i++)
    {
        cout << "#" << i-2 << ": ";
        cin >> pounds;
        sales[i] = pounds;
    }
    cout << "Staff weights:\n";
    for (i = 0; i < STAFF; i++)
        sales[i].show_stn();
    Stonewt min = sales[0];
    Stonewt max = sales[0];
    Stonewt ref11(11,0);
    int ct11 = 0;
    for (i = 0; i < STAFF; i++)
    {
        if (max < sales[i])
            max = sales[i];
        if (min > sales[i])
            max = sales[i];
        if (sales[i] >= ref11)
            ct11++;
    }
    cout << "Largest and smallest weights:\n";
    max.show_stn();
```

```
    min.show_stn();
    cout << "11 stone and over: " << ct11 << endl;
    //cin.get();
    //cin.get();
    return 0;
}
```

PE 11-7

```
// complex0.h
#ifndef COMPLEX0_H_
#define COMPLEX0_H_
#include <iostream>

class complex
{
private:
    double r;
    double i;
public:
    complex();
    complex(double real);
    complex(double real, double imag);
    double magnitude();
    complex operator+(const complex & z) const;
    complex operator-(const complex & z) const;
    complex operator~() const;
    friend complex square(const complex & z);
    friend complex operator*(const complex & z, const complex & w);
    friend std::ostream & operator<<(std::ostream & os, const complex & z);
    friend std::istream & operator>>(std::istream & is, complex & z);
};
#endif

// complex0.cpp
#include <iostream>
#include <cmath>
#include "complex0.h"

complex::complex()
{
    r = i = 0.0;
}

complex::complex(double real)
{
    r = real;
    i = 0.0;
}

complex::complex(double real, double imag)
{
    r = real;
    i = imag;
}
```

```
double complex::magnitude()
{
    return std::sqrt(r*r + i*i);
}
complex complex::operator+(const complex & z) const
{
    complex sum;
    sum.r = r + z.r;
    sum.i = i + z.i;
    return sum;
}

complex complex::operator-(const complex & z) const
{
    complex sum;
    sum.r = r + z.r;
    sum.i = i + z.i;
    return sum;
}

complex complex::operator~() const
{
    complex conjugate;
    conjugate.r = r;
    conjugate.i = -i;
    return conjugate;
}
complex square (const complex & z)
{
    complex sq;
    sq.r = z.r * z.r - z.i * z.i;
    sq.i = 2.0 * z.r * z.i;
    return sq;
}

complex operator*(const complex & z, const complex & w)
{
    complex sq;
    sq.r = w.r * z.r - w.i * z.i;
    sq.i = w.r * z.i + w.i * z.r;
    return sq;
}


std::ostream & operator<<(std::ostream & os, const complex & z)
{
    os << '(' << z.r << ',' << z.i << "i)";
    return os;
}

std::istream & operator>>(std::istream & is, complex & z)
{
    std::cout << "real: ";
    if (is >> z.r)
    {
        std::cout << "imaginary: ";
```

```
        is >> z.i;
    }
    return is;
}

// pe11-7.cpp
#include <iostream>
#include "complex0.h"  // to avoid confusion with complex.h
int main()
{
    using std::cout;
    using std::endl;
    using std::cin;

    complex a(3.0, 4.0);   // initialize to (3,4i)
    complex c;
    cout << "Enter a complex number (q to quit):\n";
    while (cin >> c)
    {
        cout << "c is " << c << endl;
        cout << "complex conjugate is " << ~c << endl;
        cout << "a is " << a << endl;
        cout << "a + c is " << a + c << endl;
        cout << "a - c is " << a - c << endl;
        cout << "a * c is " << a * c << endl;
        cout << "2 * c is " << 2 * c << endl;
        cout << "Enter a complex number (q to quit):\n";
    }
    cout << "Done!\n";
/* to keep window open
    cin.clear();
    while (cin.get() != '\n')
        continue;
    cin.get();
 */
     return 0;
}
```

Chapter 12

PE 12-1

```
// pe12-1.cpp
#include <iostream>
#include <cstring>
using namespace std;

class Cow
{
private:

    char name[20];
    char * hobby;
    double weight;
public:
```

```
    Cow();
    Cow(const char * nm, const char * ho, double wt);
    Cow(const Cow & c);
    ~Cow();
    Cow & operator=(const Cow & c);
    void ShowCow();  // display Cow data
};

Cow::Cow()
{
    strcpy(name, "Bossie");
    hobby = new char [10];
    strcpy(hobby, "gamboling");
    weight = 925;
}

Cow::Cow(const char * nm, const char * ho, double wt)
{
    strcpy(name, nm);
    hobby = new char [strlen(ho) + 1];
    strcpy(hobby, ho);
    weight = wt;
}

Cow::Cow(const Cow & c)
{
    strcpy(name, c.name);
    hobby = new char [strlen(c.hobby) + 1];
    strcpy(hobby, c.hobby);
    weight = c.weight;
}

Cow::~Cow()
{
    delete [] hobby;
}

Cow & Cow::operator=(const Cow & c)
{
    if (this == &c)
        return *this;
    delete [] hobby;
    strcpy(name, c.name);
    hobby = new char [strlen(c.hobby) + 1];
    strcpy(hobby, c.hobby);
    weight = c.weight;
    return *this;
}

void Cow::ShowCow()
{
    cout << "Name of cow: " << name << endl;
    cout << "Hobby: " << hobby << endl;
    cout << "Weight: " << weight << endl;
}

int main()
```

```
{
    Cow hostest;                                    // 1st constructor
    Cow guest("Divine", "chewing cud", 880);    // 2nd constructor
    hostest.ShowCow();
    guest.ShowCow();
    Cow copy(guest);                                // copy constructor
    guest = hostest;                                // operator=()
    guest.ShowCow();
    copy.ShowCow();
    //std::cin.get();
    return 0;
}
```

PE 12-2

```
// pe12-2.cpp
#include <iostream>
//#include "string2.h"
#include "pe12strg.h"    // alternative name
int main()
{
    using std::cout;
    using std::cin;
    String s1(" and I am a C++ student.");
    String s2 = "Please enter your name: ";
    String s3;
    cout << s2;                 // overloaded << operator
    cin >> s3;                  // overloaded >> operator
    s2 = "My name is " + s3;    // overloaded =, + operators
    cout << s2 << ".\n";
    s2 = s2 + s1;
    s2.stringup();              // converts string to uppercase
    cout << "The string\n" << s2 << "\ncontains " << s2.has('A')
            << " 'A' characters in it.\n";
    s1 = "red";     // String(const char *),
                    // then String & operator=(const String&)
    String rgb[3] = { String(s1), String("green"), String("blue")};
    cout << "Enter the name of a primary color for mixing light: ";
    String ans;
    bool success = false;
    while (cin >> ans)
    {
        ans.stringlow();        // converts string to lowercase
        for (int i = 0; i < 3; i++)
        {
            if (ans == rgb[i])  // overloaded == operator
            {
                cout << "That's right!\n";
                success = true;
                break;
            }
        }
        if (success)
            break;
        else
```

```
            cout << "Try again!\n";
    }
    cout << "Bye\n";
    //cin.get();
    return 0;
}


// pe12strg.h

#ifndef PE12STRG_H_
#define PE12STRG_H_
#include <iostream>

class String {
private:
    char * str;            // pointer to a string
    int chars;             // number of characters
    static int strings;    // total number of strings
public:
    String();
    String(const char * ps); // converts C++ string to String
    String(const String & s);
    ~String();
    int numstrings();
    int len();
    void stringup();
    void stringlow();
    int has(char ch);
    String & operator=(const String & s);
    friend std::ostream & operator<<(std::ostream & os, const String & s);
    friend std::istream & operator>>(std::istream & os, String & s);
    friend String operator+(const String & s1, const String & s2);
    friend int operator==(const String & s1, const String & s2);
    friend int operator<(const String & s1, const String & s2);
    friend int operator>(const String & s1, const String & s2);
};

#endif

// pe12strg.cpp
#include <cctype>

#include "pe12strg.h"
int String::strings = 0;

String::String()
{
    str = NULL;
    chars = 0;
    strings++;
}

String::String(const char * ps)
{
    chars = std::strlen(ps);
    str = new char [chars + 1];
```

```
    std::strcpy(str, ps);
    strings++;
}

String::String(const String & s)
{
    chars = s.chars;
    str = new char [chars + 1];
    std::strcpy(str, s.str);
    strings++;
}

String::~String()
{
    strings--;
    delete [] str;
}

int String::numstrings()
{
    return strings;
}

int String::len()
{
    return chars;
}

void String::stringup()
{
    for (int i = 0; i < chars; i++)
        str[i] = std::toupper(str[i]);
}

void String::stringlow()
{
    for (int i = 0; i < chars; i++)
        str[i] = std::tolower(str[i]);
}

String & String::operator=(const String & s) // allows chaining
{
    if (this == &s)     // assignment to self
        return * this;
    delete [] str;            // free old contents, if any
    chars = s.chars;
    str = new char [chars + 1];
    std::strcpy(str, s.str);
    return * this;
}

std::ostream & operator<<(std::ostream & os, const String & s)
{
    os << s.str;
    return os;
}
```

```
std::istream & operator>>(std::istream & is, String & s)
{
    char temp[80];
    is.getline(temp,80);
    s = temp;
    return is;
}

String operator+(const String & s1, const String & s2)
{
    int len = s1.chars + s2.chars;
    char * ps = new char [len + 1];
    std::strcpy(ps, s1.str);
    std::strcat(ps, s2.str);
    String temp(ps);
    return temp;
}

int String::has(char ch)
{
    int ct = 0;
    char * ps = str;
    while (*ps)
    {
        if (*ps++ == ch)
            ++ct;
    }
    return ct;
}

int operator==(const String & s1, const String & s2)
{
    if (s1.chars != s2.chars)
        return 0;
    else if (std::strcmp(s1.str, s2.str) == 0)
        return 1;
    else
        return 0;
}

int operator<(const String & s1, const String & s2)
{
    if (std::strcmp(s1.str, s2.str) < 0)
        return 1;
    else
        return 0;
}
int operator>(const String & s1, const String & s2)
{
    if (std::strcmp(s1.str, s2.str) > 0)
        return 1;
    else
        return 0;
}
```

PE 12-3

```
// pe12-3.cpp -- use the Stock class with dynamic memory
// link with pe12stok.cpp
#include <iostream>
#include "pe12stok.h"

const int STKS = 4;
int main(void)
{
    using std::cout;
    using std::ios_base;
// create an array of initialized objects
    Stock stocks[STKS] = {
        Stock("NanoSmart", 12, 20.0),
        Stock("Boffo Objects", 200, 2.0),
        Stock("Monolithic Obelisks", 130, 3.25),
        Stock("Fleep Enterprises", 60, 6.5)
        };

    cout.precision(2);              // #.## format
    cout.setf(ios_base::fixed, ios_base::floatfield); // #.## format
    cout.setf(ios_base::showpoint); // #.## format

    cout << "Stock holdings:\n";
    int st;
    for (st = 0; st < STKS; st++)
        cout << stocks[st];

    Stock top = stocks[0];
    for (st = 1; st < STKS; st++)
        top = top.topval(stocks[st]);
    cout << "\nMost valuable holding:\n";
    cout << top;
    //std::cin.get();
    return 0;
}

// pe12stok.h
#ifndef PE12STOK_H_
#define PE12STOK_H_

#include <iostream>

class Stock
{
private:
    char * company;
    int shares;
    double share_val;
    double total_val;
    void set_tot() { total_val = shares * share_val; }
public:
    Stock();        // default constructor
    Stock(const char * co, int n, double pr);
    Stock(const Stock & st);    // copy constructor
```

```
    ~Stock() { delete [] company; }
    Stock & operator=(const Stock & st); // assignment
    void buy(int num, double price);
    void sell(int num, double price);
    void update(double price);
    friend std::ostream & operator<<(std::ostream & os, const Stock & st);
    const Stock & topval(const Stock & s) const;
};

#endif

// pe12stok.cpp     // Stock class methods

#include <iostream>
#include <cstring>  // for strcpy()

#include <stdlib.h>    // for exit()

#include "pe12stok.h"

// constructors
Stock::Stock()
{
    company = new char [std::strlen("no name") + 1];
    std::strcpy(company, "no name");
    shares = 0;
    share_val = 0.0;
    total_val = 0.0;
}

Stock::Stock(const char * co, int n, double pr)
{
    company = new char [std::strlen(co) + 1];
    std::strcpy(company, co);
    shares = n;
    share_val = pr;
    set_tot();
}

Stock::Stock(const Stock & st)
{
    company = new char [std::strlen(st.company) + 1];
    std::strcpy(company, st.company);
    shares = st.shares;
    share_val = st.share_val;
    set_tot();
}

Stock & Stock::operator=(const Stock & st)
{
    if (this == &st)
        return *this;
    delete [] company;
    company = new char [std::strlen(st.company) + 1];
    std::strcpy(company, st.company);
    shares = st.shares;
    share_val = st.share_val;
```

```
    set_tot();
    return *this;
}

void Stock::buy(int num, double price)
{
    shares += num;
    share_val = price;
    set_tot();
}

void Stock::sell(int num, double price)
{
    if (num > shares)
    {
        std::cerr << "You can't sell more than you have!\n";
        std::exit(1);
    }
    shares -= num;
    share_val = price;
    set_tot();
}

void Stock::update(double price)
{
    share_val = price;
    set_tot();
}

std::ostream & operator<<(std::ostream & os, const Stock & st)
{
    os << "Company: " << st.company
        << "  Shares: " << st.shares << '\n'
        << "  Share Price: $" << st.share_val
        << "  Total Worth: $" << st.total_val << '\n';
    return os;
}

const Stock & Stock::topval(const Stock & s) const
{
    if (s.total_val > total_val)
        return s;
    else
        return *this;
}
```

PE 12-4

```
// pe12stak.h -- class definition for the stack ADT
#ifndef PE12STAK_H_
#define PE12STAK_H_

typedef unsigned long Item;

class Stack
{
```

```
private:
    enum {MAX = 10};    // constant specific to class
    Item * pitems;    // holds stack items
    int size;    // max number of elements in stack
    int top;        // index for top stack item
    Stack(const Stack & st) { } // no copying of stacks
    Stack & operator=(const Stack & st) { return *this; } // no assignment
public:
    Stack(int n = MAX);
    ~Stack();
    bool isempty() const;
    bool isfull() const;
    // push() returns false if stack already is full, true otherwise
    bool push(const Item & item);    // add item to stack
    // pop() returns false if stack already is empty, true otherwise
    bool pop(Item & item);    // pop top into item
};
#endif

// pe12stak.cpp -- Stack member functions
#include "pe12stak.h"
Stack::Stack(int n)    // create an empty stack
{
    size = n;
    pitems = new Item [size];
    top = 0;
}
Stack::~Stack() { delete [] pitems; }

bool Stack::isempty() const
{
    return top == 0 ? true: false;
}

bool Stack::isfull() const
{
    return top == size ? true: false;
}

bool Stack::push(const Item & item)
{
    if (top < size)
    {
        pitems[top++] = item;
        return true;
    }
    else
        return false;
}

bool Stack::pop(Item & item)
{
    if (top > 0)
    {
        item = pitems[--top];
        return true;
    }
```

```
    else
        return false;
}

// pe12-4.cpp
#include <iostream>
#include <cctype>
#include "pe12stak.h"     // modified to define customer structure
// link with pe12stak.cpp
int main()
{
    using namespace std;
    Stack st(3); // create a stack of po numbers
    unsigned long temp;
    char c;

    cout << "Please enter A to add a PO,\n"
         << "P to process a PO, and Q to quit.\n";
    while (cin >> c && (c = toupper(c)) != 'Q')
    {
        while (cin.get() != '\n')
            continue;
        if (c != 'A' && c != 'P')
        {
            cout << "Please respond with A, P, or Q: ";
            continue;
        }
        switch (c)
        {
            case 'A': if (st.isfull())
                          cout << "stack already full\n";
                      else
                      {
                          cout << "Enter PO number: ";
                          cin >> temp;
                          st.push(temp);
                      }
                      break;
            case 'P': if (st.isempty())
                          cout << "stack already empty\n";
                      else {
                          st.pop(temp);
                          cout << "Processing PO " << temp << '\n';
                      }
                      break;
            default: cout << "Whoops! Programming error!\n";
        }
        cout << "Please enter A to add a customer,\n"
        << "P to process a customer, and Q to quit.\n";
    }
    cout << "Done!\n";
    //cin.get();
    //cin.get();
    return 0;
}
```

PE 12-5

```
// pe12-5.cpp -- use the Queue interface
// link to pe12que.cpp
// modify Listing 12.10 to put calculation in a loop to
// make it easier to test different values for customers
// per hour
#include <iostream>
#include <ctime>    // for time()
#include <cstdlib>    // for rand() and srand()

#include "pe12que.h"

const long MIN_PER_HR = 60L;

bool newcustomer(double x);        // is there a new customer?

int main(void)
{
    using std::cin;
    using std::cout;
    using std::endl;
    using std::ios_base;


// setting things up
    std::srand(std::time(0));  //  random initializing of rand()

    cout << "Case Study: Bank of Heather Automatic Teller\n";
    cout << "Enter maximum size of queue: ";
    int qs;
    cin >> qs;
    Queue line(qs);            // line queue holds up to qs people

    cout << "Enter the number of simulation hours: ";
    int hours;                //  hours of simulation
    cin >> hours;
    // simulation will run 1 cycle per minute
    long cyclelimit = MIN_PER_HR * hours; // # of cycles
    Item temp;            //  new customer data
    long turnaways;        //  turned away by full queue
    long customers;         //  joined the queue
    long served;        //  served during the simulation
    long sum_line;        //  cumulative line length
    int wait_time;        //  time until autoteller is free
    long line_wait;        //  cumulative time in line
    double min_per_cust;    //  average time between arrivals

    cout << "Enter the average number of customers per hour: ";
    double perhour;         //  average # of arrival per hour
    cin >> perhour;
    while ( perhour > 0 ) // begin new loop
    {
     min_per_cust = MIN_PER_HR / perhour;
     turnaways = 0;
     customers = 0;
```

```
      served = 0;
      sum_line = 0;
      wait_time = 0;
      line_wait = 0;

// running the simulation
      for (long cycle = 0; cycle < cyclelimit; cycle++)
      {
         if (newcustomer(min_per_cust))    // have newcomer
         {
             if (line.isfull())
                 turnaways++;
             else
             {
                 customers++;
                 temp.set(cycle);    // cycle = time of arrival
                 line.enqueue(temp);    // add newcomer to line
             }
         }
         if (wait_time <= 0 && !line.isempty())
         {
             line.dequeue (temp);      // attend next customer
             wait_time = temp.ptime(); // for wait_time minutes
             line_wait += cycle - temp.when();
             served++;
         }
         if (wait_time > 0)
             wait_time--;
         sum_line += line.queuecount();
      }

// reporting results
      if (customers > 0)
      {
         cout << "customers accepted: " << customers << '\n';
         cout << "  customers served: " << served << '\n';
         cout << "         turnaways: " << turnaways << '\n';
         cout << "average queue size: ";
         cout.precision(2);
         cout.setf(ios_base::fixed, ios_base::floatfield);
         cout.setf(ios_base::showpoint);
         cout << (double) sum_line / cyclelimit << '\n';
         cout << " average wait time: "
              << (double) line_wait / served << " minutes\n";
      }
      else
         cout << "No customers!\n";
      // clear queue
      while (!line.isempty())
          line.dequeue(temp);

      cout << "Enter new value for customers per hour (0 to quit): ";
      cin >> perhour;
     } // end of new loop
     cout << "Bye\n";
     //cin.get();
     //cin.get();
```

```
    return 0;
}

//  x = average time, in minutes, between customers
//  return value is true if customer shows up this minute
bool newcustomer(double x)
{
    if (std::rand() * x / RAND_MAX < 1)
        return true;
    else
        return false;
}
```

PE 12-6

```
// pe12que.h -- interface for a queue
#ifndef _QUEUE_H_
#define _QUEUE_H_
// This queue will contain Customer items
class Customer
{
private:
    long arrive;        // arrival time for customer
    int processtime;    // processing time for customer
public:
    Customer() { arrive = processtime = 0; }
    void set(long when);
    long when() const { return arrive; }
    int ptime() const { return processtime; }
};

typedef Customer Item;

class Queue
{
private:
// class scope definitions
    // Node is a nested structure definition local to this class
    struct Node { Item item; struct Node * next;};
    enum {Q_SIZE = 10};
// private class members
    Node * front;       // pointer to front of Queue
    Node * rear;        // pointer to rear of Queue
    int items;          // current number of items in Queue
    const int qsize;    // maximum number of items in Queue
    // preemptive definitions to prevent public copying
    Queue(const Queue & q) : qsize(0) { }
    Queue & operator=(const Queue & q) { return *this;}
public:
    Queue(int qs = Q_SIZE); // create queue with a qs limit
    ~Queue();
    bool isempty() const;
    bool isfull() const;
    int queuecount() const;
    bool enqueue(const Item &item); // add item to end
    bool dequeue(Item &item);       // remove item from front
```

```
};
#endif

// pe12que.cpp -- Queue and Customer methods
#include "pe12que.h"
#include <cstdlib>         // (or stdlib.h) for rand()
using std::rand;

// Queue methods
Queue::Queue(int qs) : qsize(qs)
{
    front = rear = NULL;
    items = 0;
}

Queue::~Queue()
{
    Node * temp;
    while (front != NULL)   // while queue is not yet empty
    {
        temp = front;       // save address of front item
        front = front->next;// reset pointer to next item
        delete temp;        // delete former front
    }
}

bool Queue::isempty() const
{
    return items == 0;
}

bool Queue::isfull() const
{
    return items == qsize;
}

int Queue::queuecount() const
{
    return items;
}

// Add item to queue
bool Queue::enqueue(const Item & item)
{
    if (isfull())
        return false;
    Node * add = new Node;  // create node
    if (add == NULL)
        return false;       // quit if none available
    add->item = item;       // set node pointers
    add->next = NULL;
    items++;
    if (front == NULL)      // if queue is empty,
        front = add;        // place item at front
    else
        rear->next = add;   // else place at rear
    rear = add;             // have rear point to new node
```

```
    return true;
}

// Place front item into item variable and remove from queue
bool Queue::dequeue(Item & item)
{
    if (front == NULL)
        return false;
    item = front->item;     // set item to first item in queue
    items--;
    Node * temp = front;    // save location of first item
    front = front->next;    // reset front to next item
    delete temp;            // delete former first item
    if (items == 0)
        rear = NULL;
    return true;
}

// customer method

// when is the time at which the customer arrives
// the arrival time is set to when and the processing
// time set to a random value in the range 1 - 3
void Customer::set(long when)
{
    processtime = std::rand() % 3 + 1;
    arrive = when;
}

// pe12-6.cpp -- use the Queue interface
// link to pe12que.cpp
// modify Listing 12.10 by adding a second queue
#include <iostream>
#include <ctime>    // for time()
#include <cstdlib>    // for rand() and srand()
#include "pe12que.h"

const long MIN_PER_HR = 60L;

bool newcustomer(double x);        // is there a new customer?

int main(void)
{
    using std::cin;
    using std::cout;
    using std::endl;
    using std::ios_base;

// setting things up
    std::srand(std::time(0));     //  random initializing of rand()

    cout << "Case Study: Bank of Heather Automatic Teller\n";
    cout << "Enter maximum size of each queue: ";
    int qs;
    cin >> qs;
    Queue line1(qs);            // line queue holds up to qs people
    Queue line2(qs);            // second queue
```

```
    cout << "Enter the number of simulation hours: ";
    int hours;                //  hours of simulation
    cin >> hours;
    // simulation will run 1 cycle per minute
    long cyclelimit = MIN_PER_HR * hours; // # of cycles
    Item temp;            //  new customer data
    long turnaways;        //  turned away by full queue
    long customers;         //  joined the queue
    long served;        //  served during the simulation
    long sum_line;        //  cumulative line length
    int wait_time1;        //  time until autoteller1 is free
    int wait_time2;        //  time until autoteller2 is free
    long line_wait;        //  cumulative time in line
    double min_per_cust;    //  average time between arrivals

    cout << "Enter the average number of customers per hour: ";
    double perhour;         //  average # of arrival per hour
    cin >> perhour;
    while ( perhour > 0 ) // begin new loop
    {
     min_per_cust = MIN_PER_HR / perhour;
     turnaways = 0;
     customers = 0;
     served = 0;
     sum_line = 0;
     wait_time1 = wait_time2 = 0;
     line_wait = 0;

// running the simulation
     for (long cycle = 0; cycle < cyclelimit; cycle++)
     {
        if (newcustomer(min_per_cust))    // have newcomer
        {
            if (line1.isfull() && line2.isfull())
                turnaways++;
            else // at least one line is not full
            {
                customers++;
                temp.set(cycle);    // cycle = time of arrival
// add customer to shorter line
                if (line1.queuecount() <= line2.queuecount())
                    line1.enqueue(temp);    // add newcomer to line1
                else
                    line2.enqueue(temp);    // add newcomer to line2
                }
        }
// process customers in first queue
        if (wait_time1 <= 0 && !line1.isempty())
        {
            line1.dequeue (temp);      // attend next customer
            wait_time1 = temp.ptime(); // for wait_time minutes
            line_wait += cycle - temp.when();
            served++;
        }
        if (wait_time1 > 0)
            wait_time1--;
```

```
        sum_line += line1.queuecount();
// process customers in second queue
        if (wait_time2 <= 0 && !line2.isempty())
        {
            line2.dequeue (temp);      // attend next customer
            wait_time2 = temp.ptime(); // for wait_time minutes
            line_wait += cycle - temp.when();
            served++;
        }
        if (wait_time2 > 0)
            wait_time2--;
        sum_line += line2.queuecount();
     }
// reporting results
     if (customers > 0)
     {
        cout << "customers accepted: " << customers << '\n';
        cout << "  customers served: " << served << '\n';
        cout << "         turnaways: " << turnaways << '\n';
        cout << "average queue size: ";
        cout.precision(2);
        cout.setf(ios_base::fixed, ios_base::floatfield);
        cout.setf(ios_base::showpoint);
        cout << (double) sum_line / cyclelimit << '\n';
        cout << " average wait time: "
             << (double) line_wait / served << " minutes\n";
     }
     else
        cout << "No customers!\n";
     // clear queues
     while (!line1.isempty())
        line1.dequeue(temp);
     while (!line2.isempty())
        line2.dequeue(temp);

     cout << "Enter new value for customers per hour (0 to quit): ";
     cin >> perhour;
    } // end of new loop
    cout << "Bye\n";
    //cin.get();
    //cin.get();

    return 0;
}

//  x = average time, in minutes, between customers
//  return value is true if customer shows up this minute
bool newcustomer(double x)
{
    if (std::rand() * x / RAND_MAX < 1)
        return true;
    else
        return false;
```

```
}
```

Chapter 13

PE 13-1

```
// cd.h -- base class

#ifndef CD_H_
#define CD_H_

class Cd {  // represents a CD disk
private:
    char performers[50];
    char label[20];
    int selections;   // number of selections
    double playtime;  // playing time in minutes
public:
    Cd(const char * s1, const char * s2, int n, double x);
//    Cd(const Cd & d);    // default version is fine
    Cd();
    virtual ~Cd() {}
    virtual void Report() const;  // reports all CD data
//    Cd & operator=(const Cd & d); // default version is fine
};

#endif

// pe13-1cd.cpp -- cd methods
#include <iostream>
#include <cstring>
#include "cd.h"

Cd::Cd(const char * s1, const char * s2, int n, double x)
{
    std::strncpy(performers, s1, 49);
    performers[49] = '\0';
    std::strncpy(label, s2, 19);
    label[19] = '\0';
    selections = n;
    playtime = x;
}

Cd::Cd()
{
    performers[0] = '\0';
    label[0] = '\0';
    selections = 0;
    playtime = 0.0;
}

void Cd::Report() const
{
    using std::cout;
    using std::endl;
    cout << "Performer(s): " << performers << endl;
```

```
    cout << "Label: " << label << endl;
    cout << "Number of selections: " << selections << endl;
    cout << "Play time: " << playtime << endl;
}

// classic.h
// derived class

#ifndef CLASSIC_H_
#define CLASSIC_H_

#include "cd.h"

class Classic : public Cd
{
private:
    char primarywork[50];
public:
    Classic(const char * pw, const char * s1, const char * s2,
             int n, double x);
    Classic();
    void Report() const;    // redefine to report primary work
};

#endif

// pe13-1cl.cpp
// Classic methods
#include <iostream>
#include <cstring>
#include "classic.h"

Classic::Classic(const char * pw, const char * s1,
                 const char * s2, int n, double x)
    : Cd(s1, s2, n, x)
{
    std::strncpy(primarywork, pw, 49);
    primarywork[49] = '\0';
}
Classic::Classic() : Cd()
{
    primarywork[0] = '\0';
}
void Classic::Report() const
{
    std::cout << "Primary work: " << primarywork << std::endl;
    Cd::Report();
}

// pe13-1.cpp
#include <iostream>
#include "classic.h"     // which will contain #include cd.h
void Bravo(const Cd & disk);
int main()
{
    using std::cout;
    Cd c1("Beatles", "Capitol", 14, 35.5);
```

```
    Classic c2 = Classic("Piano Sonata in B flat, Fantasia in C",
                    "Alfred Brendel", "Philips", 2, 57.17);
    Cd *pcd = &c1;

    cout << "Using object directly:\n";
    c1.Report();    // use Cd method
    c2.Report();    // use Classic method

    cout << "Using type cd * pointer to objects:\n";
    pcd->Report();  // use Cd method for cd object
    pcd = &c2;
    pcd->Report();  // use Classic method for classic object

    cout << "Calling a function with a Cd reference argument:\n";
    Bravo(c1);
    Bravo(c2);

    cout << "Testing assignment: ";
    Classic copy;
    copy = c2;
    copy.Report();
    //std::cin.get();
    return 0;
}

void Bravo(const Cd & disk)
{
    disk.Report();
}
```

PE 13-2

```
 // pe13cd.h  -- Cd and Classic classes
#ifndef PE13CD_H_
#define PE13CD_H_

class Cd {     // represents a CD disk
private:
    char * performers;
    char * label;
    int selections;
    double playtime;
public:
    Cd(char * s1, char * s2, int n, double x);
    Cd(const Cd & d);
    Cd();
    virtual ~Cd();
    virtual void Report() const;    // reports CD data
    Cd & operator=(const Cd & d);
};

class Classic : public Cd {
private:
    char * works;
public:
    Classic();
```

```
    Classic(char * w, char * s1, char * s2, int n, double x);
    Classic(const Classic & cl);
    ~Classic();
    virtual void Report() const;
    Classic & operator=(const Classic & cl);
};

#endif


// pe13cd.cpp -- Cd and Classic methods

#include <iostream>
#include <cstring>

#include "pe13cd.h"

Cd::Cd (char * s1, char * s2, int n, double x)
{
    performers = new char [std::strlen(s1) + 1];
    std::strcpy(performers, s1);
    label = new char [std::strlen(s2) + 1];
    std::strcpy(label, s2);
    selections = n;
    playtime = x;
}

Cd::Cd(const Cd & d)
{
    performers = new char [std::strlen(d.performers) + 1];
    std::strcpy(performers, d.performers);
    label = new char [ strlen(d.label) + 1];
    std::strcpy(label, d.label);
    selections = d.selections;
    playtime = d.playtime;
}

Cd::Cd ()
{
    performers = new char [8];
    std::strcpy(performers, "Unknown");
    label = new char [8];
    std::strcpy(label, "Unknown");
    selections = 0;
    playtime = 0.0;
}

Cd::~Cd()
{
    delete [] performers;
    delete [] label;
}

void Cd::Report() const
{
    std::cout << "Performance by " << performers << ": Label = "
            << label << '\n' << selections
```

```
            << " selections, playing time = " << playtime << '\n';
}

Cd & Cd::operator=(const Cd & d)
{
    if (this == &d)
        return *this;
    delete [] performers;
    performers = new char [std::strlen(d.performers) + 1];
    std::strcpy(performers, d.performers);
    delete [] label;
    label = new char [std::strlen(d.label) + 1];
    std::strcpy(label, d.label);
    selections = d.selections;
    playtime = d.playtime;
    return *this;
}

Classic::Classic(char * w, char * s1, char * s2, int n, double x) :
          Cd(s1, s2, n, x)
{
    works = new char [std::strlen(w) + 1];
    std::strcpy(works, w);
}

Classic::Classic() : Cd()
{
    works = new char [5];
    std::strcpy(works, "None");
}

Classic::Classic(const Classic & cl) : Cd(cl)
{
    works = new char [std::strlen(cl.works) + 1];
    std::strcpy(works, cl.works);
}

Classic::~Classic()
{
    delete [] works;
}

Classic & Classic::operator=(const Classic & cl)
{
    if (this == &cl)
        return *this;
    Cd::operator=(cl);    // base-portion assignment
    delete [] works;
    works = new char [std::strlen(cl.works) + 1];
    std::strcpy(works, cl.works);
    return *this;
}

void Classic::Report()  const
{
    Cd::Report();
    std::cout << "Works:\n" << works << '\n';
```

```
}


// pe13-2.cpp

#include <iostream>
#include "pe13cd.h"     // Cd and Classic classes
// compile with pe13cd.cpp

void Bravo(const Cd & disk);
int main()
{
    using std::cout;
    Cd c1("Beatles", "Capitol", 14, 35.5);
    Classic c2 = Classic("Piano Sonata in B flat, Fantasia in C",
                    "Alfred Brendel", "Philips", 2, 57.17);
    Cd *pcd = &c1;

    cout << "Using object directly:\n";
    c1.Report();    // use Cd method
    c2.Report();    // use Classic method

    cout << "Using type cd * pointer to objects:\n";
    pcd->Report();  // use Cd method for cd object
    pcd = &c2;
    pcd->Report();  // use Classic method for classic object

    cout << "Calling a function with a Cd reference argument:\n";
    Bravo(c1);
    Bravo(c2);

    cout << "Testing assignment: ";
    Classic copy;
    copy = c2;
    copy.Report();
    //std::cin.get();
    return 0;
}

void Bravo(const Cd & disk)
{
    disk.Report();
}
```

PE 13-3

```
// pe13dma.h  -- inheritance and dynamic memory allocation

#ifndef DMA_H_
#define DMA_H_
#include <iostream>

// Abstract Base Class
class ABC
{
```

```
private:
    char * label;
    int rating;
public:
    ABC(const char * l = "null", int r = 0);
    ABC(const ABC & rs);
    virtual ~ABC() = 0;
    virtual ABC & operator*()  { return *this; }
    ABC & operator=(const ABC & rs);
    virtual void View() const;
    friend std::ostream & operator<<( std::ostream & os, const ABC & rs);
};


//  Former Base Class Using DMA
class baseDMA : public ABC
{
private:

public:
    baseDMA(const char * l = "null", int r = 0);
};


// derived class without DMA
// no destructor needed
// uses implicit copy constructor
// uses implicit assignment operator
class lacksDMA :public ABC
{
private:
    char color[40];
public:
    lacksDMA(const char * c = "blank", const char * l = "null",
             int r = 0);
    lacksDMA(const char * c, const ABC & rs);
    void View() const;

};

// derived class with DMA
class hasDMA :public ABC
{
private:
    char * style;
public:
    hasDMA(const char * s = "none", const char * l = "null",
             int r = 0);
    hasDMA(const char * s, const ABC & rs);
    hasDMA(const hasDMA & hs);
    ~hasDMA();
    hasDMA & operator=(const hasDMA & rs);
    void View() const;
};

#endif
```

```
// pe13dma.cpp --dma class methods

#include "pe13dma.h"
#include <cstring>

// ABC methods
ABC::ABC(const char * l, int r)
{
    label = new char[std::strlen(l) + 1];
    std::strcpy(label, l);
    rating = r;
}

ABC::ABC(const ABC & rs)
{
    label = new char[std::strlen(rs.label) + 1];
    std::strcpy(label, rs.label);
    rating = rs.rating;
}

ABC::~ABC()
{
    delete [] label;
}

ABC & ABC::operator=(const ABC & rs)
{
    if (this == &rs)
        return *this;
    delete [] label;
    label = new char[std::strlen(rs.label) + 1];
    std::strcpy(label, rs.label);
    rating = rs.rating;
    return *this;
}

void ABC::View() const
{
    std::cout << "Label: " << label << std::endl;
    std::cout << "Rating: " << rating << std::endl;
}

std::ostream & operator<<(std::ostream & os, const ABC & rs)
{
    rs.View();
    return os;
}


// baseDMA methods
baseDMA::baseDMA(const char * l, int r) : ABC(l,r)
{
}

// lacksDMA methods
lacksDMA::lacksDMA(const char * c, const char * l, int r)
    : ABC(l, r)
```

```
{
    std::strncpy(color, c, 39);
    color[39] = '\0';
}

lacksDMA::lacksDMA(const char * c, const ABC & rs)
    : ABC(rs)
{
    std::strncpy(color, c, 39);
    color[39] = '\0';
}

void lacksDMA::View() const
{
    ABC::View();
    std::cout << "Color: " << color << std::endl;
}


// hasDMA methods
hasDMA::hasDMA(const char * s, const char * l, int r)
         : ABC(l, r)
{
    style = new char[std::strlen(s) + 1];
    std::strcpy(style, s);
}

hasDMA::hasDMA(const char * s, const ABC & rs)
         : ABC(rs)
{
    style = new char[std::strlen(s) + 1];
    std::strcpy(style, s);
}

hasDMA::hasDMA(const hasDMA & hs)
         : ABC(hs)  // invoke base class copy constructor
{
    style = new char[std::strlen(hs.style) + 1];
    std::strcpy(style, hs.style);
}

hasDMA::~hasDMA()
{
    delete [] style;
}

hasDMA & hasDMA::operator=(const hasDMA & hs)
{
    if (this == &hs)
        return *this;
    ABC::operator=(hs);  // copy base portion
    style = new char[std::strlen(hs.style) + 1];
    std::strcpy(style, hs.style);
    return *this;
}

void hasDMA::View() const
```

```
{
    ABC::View();
    std::cout << "Style: " << style << std::endl;
}

// pe13-3.cpp -- inheritance, friends, and DMA
// compile with pe13dma.cpp
#include <iostream>
#include "pe13dma.h"
int main()
{
    using std::cout;
    using std::endl;
    baseDMA shirt("Portabelly", 8);
    lacksDMA balloon("red", "Blimpo", 4);
    hasDMA map("Mercator", "Buffalo Keys", 5);
    cout << shirt << endl;
    cout << balloon << endl;
    cout << map << endl;
    lacksDMA balloon2(balloon);
    hasDMA map2;
    map2 = map;
    cout << balloon2 << endl;
    cout << map2 << endl;

    ABC * pts[3];
    pts[0] = &shirt;
    pts[1] = &balloon;
    pts[2] = &map;

    for (int i = 0; i < 3; i++)
        cout << *pts[i] << endl;
    for (int i = 0; i < 3; i++)
        pts[i]->View();
    //std::cin.get();
    return 0;
}
```

PE 13-4

```
// pe13cd.h  -- Cd and Classic classes
#ifndef PE13CD_H_
#define PE13CD_H_

class Cd {     // represents a CD disk
private:
    char * performers;
    char * label;
    int selections;
    double playtime;
public:
    Cd(char * s1, char * s2, int n, double x);
    Cd(const Cd & d);
    Cd();
    virtual ~Cd();
    virtual void Report() const;    // reports CD data
    Cd & operator=(const Cd & d);
```

```
};

class Classic : public Cd {
private:
    char * works;
public:
    Classic();
    Classic(char * w, char * s1, char * s2, int n, double x);
    Classic(const Classic & cl);
    ~Classic();
    virtual void Report() const;
    Classic & operator=(const Classic & cl);
};

#endif

// pe13port.cpp -- Port and VintagePort methods

#include "pe13port.h"
#include <cstring>
using std::strcpy;
using std::strncpy;
using std::strlen;
using std::ostream;
using std::cout;
using std::endl;

Port::Port( const char * br, const char * st, int b)
{
    brand = new char[ strlen( br) + 1];
    strcpy( brand, br);
    strncpy( style, st, 19);
    style[19] = '\0';
    bottles = b;
}

Port::Port( const Port & p)
{
    brand = new char[ strlen(p.brand) + 1];
    strcpy( brand, p.brand);
    strcpy( style, p.style);
    bottles = p.bottles;
}

Port & Port::operator=( const Port & p)
{
    if (this==&p) return *this;

    delete [] brand;
    brand = new char[ strlen( p.brand) + 1];
    strcpy(brand, p.brand);
    strcpy(style, p.style);
    bottles = p.bottles;
    return *this;
}

Port & Port::operator +=( int b)
```

```
{
    bottles += b;
    return *this;
}

Port & Port::operator -=(int b)
{
    if (bottles>=b)
        bottles -= b;
    else
        cout << "Inventory too low.  Removal failed.\n\n";

    return *this;
}

void Port::Show() const
{
    cout << "Brand: " << brand << endl;
    cout << "Style: " << style << endl;
    cout << "Bottles: " << bottles << endl;
}

ostream & operator<<(ostream & os, const Port & p)
{
    os << p.brand << ", " << p.style << ", " << p.bottles;
    return os;
}

VintagePort::VintagePort()
    :   Port("none", "Vintage")  // set style to "Vintage"
{
    //  Set defaults.
    nickname = new char[5];
    strcpy( nickname, "none");
    year = 0;
}

VintagePort::VintagePort(const char * brand, int bottles, const char *nname,
int yr)
    :   Port( brand, "Vintage", bottles)
{
    //  Set values.
    nickname = new char[ strlen( nname) + 1];

    strcpy( nickname, nname);
    year = yr;
}

VintagePort::VintagePort(const VintagePort & vp)
:   Port( vp)
{
    nickname = new char[ strlen(vp.nickname) + 1];
    strcpy( nickname, vp.nickname);
    year = vp.year;
}

VintagePort & VintagePort::operator =(const VintagePort & vp)
```

```
{
    if (this == &vp) return *this;

    Port::operator =(vp);

    delete [] nickname;
    nickname = new char[ strlen( vp.nickname) + 1];
    strcpy( nickname, vp.nickname);
    year = vp.year;

    return *this;
}

void VintagePort::Show() const
{
    Port::Show();
    cout << "Nickname: " << nickname << endl;
    cout << "Year: " << year << endl;
}

ostream & operator << ( ostream & os, const VintagePort & vp)
{
    os << (const Port &) vp ;   // use Port friend version of operator<<()
    os << ", " << vp.nickname << ", " << vp.year;
    return os;
}


// pe13-4.cpp
#include <iostream>
#include "pe13port.h"
// compile with pe13port.cpp

int main()
{
    using std::cout;
   using std::endl;
    Port usual("Mr. Porto", "Ruby", 100);
    VintagePort special("Bainscotts", 14, "The Bold", 1963);

    cout << usual << endl;
    cout << special << endl;
    special -= 2;
    cout << special << endl;
    //std::cin.get();
    return 0;
}
```

Chapter 14

PE 14-1

```
// pairs -- define a Pair template
#ifndef PAIRS_H_
#define PAIRS_H_
```

```
template<class T1, class T2>
class Pair
{
private:
    T1 a;
    T2 b;
public:
    T1 & first();
    T2 & second();
    T1 first() const { return a; }
    T2 second() const { return b; }
    Pair(const T1 & aval, const T2 & bval) : a(aval), b(bval) { }
    Pair() {}
};

template<class T1, class T2>
T1 & Pair<T1,T2>::first()
{
    return a;
}
template<class T1, class T2>
T2 & Pair<T1,T2>::second()
{
    return b;
}
#endif


// winec.h -- wine class using containment
#ifndef WINEC_H_
#define WINEC_H_

#include <iostream>
#include <string>
#include <valarray>
#include "pairs.h"

class Wine
{
private:
    typedef std::valarray<int> ArrayInt;
    typedef Pair<ArrayInt, ArrayInt> PairArray;
    std::string label;      // wine brandname
    int years;              // number of years
    PairArray data;

public:
    Wine() : label("none"), years(0), data(ArrayInt(),ArrayInt()) {}
    Wine(const char * l, int y, const int yr[], const int bot[]);
    Wine(const char * l, const ArrayInt & yr, const ArrayInt & bot);
    Wine(const char * l, const PairArray & yr_bot);
    Wine(const char * l, int y);
    void GetBottles();
    void Show() const;
    const std::string & Label() { return label; }
    int sum() const { return data.second().sum(); }
};
```

```
#endif

// winec.cpp  -- Wine class with containment
#include <iostream>
#include "winec.h"

using std::cin;
using std::cout;
using std::cerr;
using std::endl;

Wine::Wine(const char * l, int y, const int yr[], const int bot[])
  : label(l), years(y), data(ArrayInt(yr,y),ArrayInt(bot,y) )
{
}

Wine::Wine(const char * l, const ArrayInt & yr, const ArrayInt & bot)
  : label(l), years(yr.size()), data(ArrayInt(yr), ArrayInt(yr))
{
    if (yr.size() != bot.size())
    {
        cerr << "Year data, bottle data mismatch, array set to 0 size.\n";
        years = 0;
        data = PairArray(ArrayInt(),ArrayInt());
    }
    else
    {
        data.first() = yr;
        data.second() = bot;
    }
}

Wine::Wine(const char * l, const PairArray & yr_bot)
: label(l), years(yr_bot.first().size()), data(yr_bot)  { }

Wine::Wine(const char * l, int y) : label(l), years(y),
    data(ArrayInt(0,y),ArrayInt(0,y))
{}

void Wine::GetBottles()
{
    if (years < 1)
    {
        cout << "No space allocated for data\n";
        return;
    }

    cout << "Enter " << label <<
            " data for " << years << " year(s):\n";
    for (int i = 0; i < years; i++)
    {
        cout << "Enter year: ";
        cin >> data.first()[i];
        cout << "Enter bottles for that year: ";
        cin >> data.second()[i];
    }
```

```
}

void Wine::Show() const
{
    cout << "Wine: " << label << endl;
    cout << "\tYear\tBottles\n";
    for (int i = 0; i < years; i++)
        cout << '\t' << data.first()[i]
             << '\t' << data.second()[i] << endl;
}

// pe14-1.cpp  -- using Wine class with containment

#include <iostream>
#include "winec.h"

int main ( void )
{
    using std::cin;
    using std::cout;
    using std::endl;

    cout << "Enter name of wine: ";
    char lab[50];
    cin.getline(lab, 50);
    cout << "Enter number of years: ";
    int yrs;
    cin >> yrs;

    Wine holding(lab, yrs); // store label, years, give arrays yrs elements
    holding.GetBottles();   // solicit input for year, bottle count
    holding.Show();         // display object contents

    const int YRS = 3;
    int y[YRS] = {1993, 1995, 1998};
    int b[YRS] = { 48, 60, 72};
    // create new object, initialize using data in arrays y and b
    Wine more("Gushing Grape Red",YRS, y, b);
    more.Show();
    cout << "Total bottles for " << more.Label() // use Label() method
         << ": " << more.sum() << endl;          // use sum() method
    cout << "Bye\n";
    //cin.get();
    //cin.get();
    return 0;
}
```

PE 14-2

```
// pairs -- define a Pair template
#ifndef PAIRS_H_
#define PAIRS_H_

template<class T1, class T2>
class Pair
{
```

```
private:
    T1 a;
    T2 b;
public:
    T1 & first();
    T2 & second();
    T1 first() const { return a; }
    T2 second() const { return b; }
    Pair(const T1 & aval, const T2 & bval) : a(aval), b(bval) { }
    Pair() {}
};

template<class T1, class T2>
T1 & Pair<T1,T2>::first()
{
    return a;
}
template<class T1, class T2>
T2 & Pair<T1,T2>::second()
{
    return b;
}
#endif


// winei.h -- wine class using private inheritance
#ifndef WINEC_H_
#define WINEC_H_

#include <iostream>
#include <string>
#include <valarray>
#include "pairs.h"

class Wine: private std::string,
            private Pair<std::valarray<int>, std::valarray<int> >
{
private:
    typedef std::valarray<int> ArrayInt;
    typedef Pair<ArrayInt, ArrayInt> PairArray;
    int years;              // number of years

public:
    Wine() : std::string("none"), years(0), PairArray(ArrayInt(),
             ArrayInt()) { }
    Wine(const char * l, int y, const int yr[], const int bot[]);
    Wine(const char * l, const ArrayInt & yr, const ArrayInt & bot);
    Wine(const char * l, const PairArray & yr_bot);
    Wine(const char * l, int y);
    void GetBottles();
    void Show() const;
    const std::string & Label()const
               {return (const std::string &) (*this);}
    int sum() const { return PairArray::second().sum(); }
};

#endif
```

```
// winei.cpp  -- Wine class with private inheritance
#include <iostream>
#include "winei.h"

using std::cin;
using std::cout;
using std::cerr;
using std::endl;
using std::string;

Wine::Wine(const char * l, int y, const int yr[], const int bot[])
  : string(l), years(y), PairArray(ArrayInt(yr,y),ArrayInt(bot,y) )
{
}

Wine::Wine(const char * l, const ArrayInt & yr, const ArrayInt & bot)
  : string(l), years(yr.size()), PairArray(ArrayInt(yr), ArrayInt(yr))
{
    if (yr.size() != bot.size())
    {
        cerr << "Year data, bottle data mismatch, array set to 0 size.\n";
        years = 0;
        PairArray::operator=(PairArray(ArrayInt(),ArrayInt()));
    }
    else
    {
        PairArray::first() = yr;
        PairArray::second() = bot;
    }
}

Wine::Wine(const char * l, const PairArray & yr_bot)
: string(l), years(yr_bot.first().size()), PairArray(yr_bot)  { }


Wine::Wine(const char * l, int y) : string(l), years(y),
    PairArray(ArrayInt(0,y),ArrayInt(0,y))
{}

void Wine::GetBottles()
{
    if (years < 1)
    {
        cout << "No space allocated for data\n";
        return;
    }

    cout << "Enter " << Label() <<
            " data for " << years << " year(s):\n";
    for (int i = 0; i < years; i++)
    {
        cout << "Enter year: ";
        cin >> PairArray::first()[i];
        cout << "Enter bottles for that year: ";
        cin >> PairArray::second()[i];
    }
```

```
}

void Wine::Show() const
{
    cout << "Wine: " << Label() << endl;
    cout << "\tYear\tBottles\n";
    for (int i = 0; i < years; i++)
        cout << '\t' << PairArray::first()[i]
             << '\t' << PairArray::second()[i] << endl;
}

// pe14-2.cpp  -- using Wine class with private inheritance
#include <iostream>
#include "winei.h"

int main ( void )
{
    using std::cin;
    using std::cout;
    using std::endl;

    cout << "Enter name of wine: ";
    char lab[50];
    cin.getline(lab, 50);
    cout << "Enter number of years: ";
    int yrs;
    cin >> yrs;

    Wine holding(lab, yrs); // store label, years, give arrays yrs elements
    holding.GetBottles();   // solicit input for year, bottle count
    holding.Show();         // display object contents

    const int YRS = 3;
    int y[YRS] = {1993, 1995, 1998};
    int b[YRS] = { 48, 60, 72};
    // create new object, initialize using data in arrays y and b
    Wine more("Gushing Grape Red",YRS, y, b);
    more.Show();
    cout << "Total bottles for " << more.Label() // use Label() method
         << ": " << more.sum() << endl;          // use sum() method
    cout << "Bye\n";
    //cin.get();
    //cin.get();
    return 0;
}
```

PE 14-3

```
// queuetp.h -- interface for a queue template
#ifndef _QUEUETP_H_
#define _QUEUETP_H_

template <class Item, int n>
class QueueTP
{
// class scope definitions
```

```
    // Node is a nested structure definition local to this class
    struct Node { Item item; struct Node * next;};
private:
    Node * front;        // pointer to front of Queue
    Node * rear;        // pointer to rear of Queue
    int items;        // current number of items in Queue
    const int qsize;    // maximum number of items in Queue
    QueueTP(const QueueTP & q) : qsize(0) { }    // preemptive definition
    QueueTP & operator=(const QueueTP & q) { return *this;}
public:
    QueueTP();
    ~QueueTP();
    bool isempty() const;
    bool isfull() const;
    int queuecount() const;
    bool enqueue(const Item &item);    // add item to end
    bool dequeue(Item &item);        // remove item from front
};

// QueueTP methods
template <class Item, int n>
QueueTP<Item,n>::QueueTP() : qsize(n)
{
    front = rear = NULL;
    items = 0;
}

template <class Item, int n>
QueueTP<Item,n>::~QueueTP()
{
    Node * temp;
    while (front != NULL)    // while queue is not yet empty
    {
        temp = front;        // save address of front item
        front = front->next; // reset pointer to next item
        delete temp;         // delete former front
    }
}

template <class Item, int n>
bool QueueTP<Item,n>::isempty() const
{
    return items == 0 ? true : false;
}

template <class Item, int n>
bool QueueTP<Item,n>::isfull() const
{
    return items == qsize ? true : false;
}

template <class Item, int n>
int QueueTP<Item,n>::queuecount() const
{
    return items;
}
```

```
// Add item to queue
template <class Item, int n>
bool QueueTP<Item,n>::enqueue(const Item & item)
{
    if (isfull())
        return false;
    Node * add = new Node;  // create node
    if (add == NULL)
        return false;       // quit if none available
    add->item = item;       // set node pointers
    add->next = NULL;
    items++;
    if (front == NULL)      // if queue is empty,
        front = add;        // place item at front
    else
        rear->next = add;   // else place at rear
    rear = add;             // have rear point to new node
    return true;
}

// Place front item into item variable and remove from queue
template <class Item, int n>
bool QueueTP<Item,n>::dequeue(Item & item)
{
    if (front == NULL)
        return false;
    item = front->item;     // set item to first item in queue
    items--;
    Node * temp = front;    // save location of first item
    front = front->next;    // reset front to next item
    delete temp;            // delete former first item
    if (items == 0)
        rear = NULL;
    return true;
}
#endif


// workermi.h  -- working classes with MI
#ifndef WORKERMI_H_
#define WORKERMI_H_

#include <string>

class Worker   // an abstract base class
{
private:
    std::string fullname;
    long id;
protected:
    virtual void Data() const;
    virtual void Get();
public:
    Worker() : fullname("no one"), id(0L) {}
    Worker(const std::string & s, long n)
            : fullname(s), id(n) {}
    virtual ~Worker() = 0; // pure virtual function
```

```
    virtual void Set() = 0;
    virtual void Show() const = 0;
};

class Waiter : virtual public Worker
{
private:
    int panache;
protected:
    void Data() const;
    void Get();
public:
    Waiter() : Worker(), panache(0) {}
    Waiter(const std::string & s, long n, int p = 0)
            : Worker(s, n), panache(p) {}
    Waiter(const Worker & wk, int p = 0)
            : Worker(wk), panache(p) {}
    void Set();
    void Show() const;
};

class Singer : virtual public Worker
{
protected:
enum {other, alto, contralto, soprano,
                   bass, baritone, tenor};
    enum {Vtypes = 7};
    void Data() const;
    void Get();
private:
    static char *pv[Vtypes];    // string equivs of voice types
    int voice;
public:
    Singer() : Worker(), voice(other) {}
    Singer(const std::string & s, long n, int v = other)
            : Worker(s, n), voice(v) {}
    Singer(const Worker & wk, int v = other)
            : Worker(wk), voice(v) {}
    void Set();
    void Show() const;
};

// multiple inheritance
class SingingWaiter : public Singer, public Waiter
{
protected:
    void Data() const;
    void Get();
public:
    SingingWaiter()  {}
    SingingWaiter(const std::string & s, long n, int p = 0,
                          int v = other)
            : Worker(s,n), Waiter(s, n, p), Singer(s, n, v) {}
    SingingWaiter(const Worker & wk, int p = 0, int v = other)
            : Worker(wk), Waiter(wk,p), Singer(wk,v) {}
    SingingWaiter(const Waiter & wt, int v = other)
            : Worker(wt),Waiter(wt), Singer(wt,v) {}
```

```
    SingingWaiter(const Singer & wt, int p = 0)
            : Worker(wt),Waiter(wt,p), Singer(wt) {}
    void Set();
    void Show() const;
};

#endif

// workermi.cpp -- working class methods with MI
#include "workermi.h"
#include <iostream>
using std::cout;
using std::cin;
using std::endl;
// Worker methods
Worker::~Worker() { }

// protected methods
void Worker::Data() const
{
    cout << "Name: " << fullname << endl;
    cout << "Employee ID: " << id << endl;
}

void Worker::Get()
{
    getline(cin, fullname);
    cout << "Enter worker's ID: ";
    cin >> id;
    while (cin.get() != '\n')
        continue;
}

// Waiter methods
void Waiter::Set()
{
    cout << "Enter waiter's name: ";
    Worker::Get();
    Get();
}

void Waiter::Show() const
{
    cout << "Category: waiter\n";
    Worker::Data();
    Data();
}

// protected methods
void Waiter::Data() const
{
    cout << "Panache rating: " << panache << endl;
}

void Waiter::Get()
{
    cout << "Enter waiter's panache rating: ";
```

```
    cin >> panache;
    while (cin.get() != '\n')
        continue;
}

// Singer methods

char * Singer::pv[Singer::Vtypes] = {"other", "alto", "contralto",
            "soprano", "bass", "baritone", "tenor"};

void Singer::Set()
{
    cout << "Enter singer's name: ";
    Worker::Get();
    Get();
}

void Singer::Show() const
{
    cout << "Category: singer\n";
    Worker::Data();
    Data();
}

// protected methods
void Singer::Data() const
{
    cout << "Vocal range: " << pv[voice] << endl;
}

void Singer::Get()
{
    cout << "Enter number for singer's vocal range:\n";
    int i;
    for (i = 0; i < Vtypes; i++)
    {
        cout << i << ": " << pv[i] << "   ";
        if ( i % 4 == 3)
            cout << endl;
    }
    if (i % 4 != 0)
        cout << '\n';
    cin >>  voice;
    while (cin.get() != '\n')
        continue;
}

// SingingWaiter methods
void SingingWaiter::Data() const
{
    Singer::Data();
    Waiter::Data();
}

void SingingWaiter::Get()
{
    Waiter::Get();
```

```
    Singer::Get();
}

void SingingWaiter::Set()
{
    cout << "Enter singing waiter's name: ";
    Worker::Get();
    Get();
}

void SingingWaiter::Show() const
{
    cout << "Category: singing waiter\n";
    Worker::Data();
    Data();
}

// pe14-3.cpp -- multiple inheritance
// compile with workermi.cpp

#include <iostream>
#include <cstring>
#include "workermi.h"
#include "queuetp.h"
const int SIZE = 5;
int main()
{
    QueueTP<Worker *, (int) SIZE> lolas;
    Worker * ptemp;

    while (!lolas.isfull() )
    {
        char choice;
        std::cout << "Enter the employee category:\n"
            << "w: waiter  s: singer  "
            << "t: singing waiter  q: quit\n";
        std::cin >> choice;
        while (std::strchr("ewstq", choice) == NULL)
        {
            std::cout << "Please enter a w, s, t, or q: ";
            std::cin >> choice;
        }
        if (choice == 'q')
            break;
        switch(choice)
        {
            case 'w': ptemp = new Waiter;
                      break;
            case 's': ptemp = new Singer;
                      break;
            case 't': ptemp = new SingingWaiter;
                      break;
        }
        std::cin.get();
        ptemp->Set();
        lolas.enqueue(ptemp);
    }
```

```
    std::cout << "\nHere is your staff:\n";
    while (!lolas.isempty())
    {
        lolas.dequeue(ptemp);
        std::cout << std::endl;
        ptemp->Show();
        delete ptemp;
    }
    //std::cin.get();
    //std::cin.get();
    return 0;
}
```

PE 14-4

```
// pe14-pg.h
#ifndef PGPPBD__
#define PGPPDB__

#include <iostream>
#include <cstring>
#include <cstdlib>

const int Len = 20;
class Person
{
private:
    char fname[Len];
    char lname[Len];
public:
    Person() { fname[0] = lname[0] = '\0'; }
    Person (const char *fn, const char * ln);
    virtual ~Person() {}
    virtual void Show() const { std::cout << fname << " " << lname; }
    virtual void Set();
};

class Gunslinger : virtual public Person
{
private:
    double drawtime;
    int notches;
public:
    Gunslinger() : Person("Joe", "Doe"), drawtime(0.0),
                        notches(0) { }
    Gunslinger(const char *fn, const char *ln,
                    double d = 1.0, int n = 0) : Person (fn, ln),
                    drawtime(d), notches(n) { }
    Gunslinger(const Person & p, double d = 1.0, int n = 0) :
                    Person(p), drawtime(d), notches(n) { }
    virtual ~Gunslinger() {}
// Person(p) is the default copy constructor
    double Draw() const { return drawtime; }
    void Show () const;
    void Set();
```

```
};

class PokerPlayer : virtual public Person
{
public:
    PokerPlayer() : Person("Busted", "Strait") {}
    PokerPlayer(const char *fn, const char *ln) : Person(fn, ln) {}
    PokerPlayer(const Person & p) : Person(p) {}
    virtual ~PokerPlayer() {}
    int Draw() const { return std::rand() % 52 + 1; }
};

class BadDude : public Gunslinger, public PokerPlayer
{
public:
    BadDude() : Person("Bad", "Dude"), Gunslinger() {}
    BadDude(const char *fn, const char *ln,
                    double d = 1.0, int n = 0) : Person (fn, ln),
                    Gunslinger(fn, ln, d, n) { }
    BadDude(const Person & p, double d = 1.0, int n = 0) :
                    Person(p), Gunslinger(p, d, n) { }
    double Gdraw() const { return Gunslinger::Draw(); }
    int Cdraw() const { return PokerPlayer::Draw(); }
    void Show() const { Gunslinger::Show(); }
    void Set() { Gunslinger::Set(); }
};

#endif


//pe14-4pg.cpp
#include <iostream>
#include <cstring>
#include "pe14-4pg.h"

Person::Person (const char *fn, const char * ln)
{
    std::strncpy(fname,fn, Len - 1);
    fname[Len - 1] = '\0';
    std::strncpy(lname,ln, Len - 1);
    lname[Len - 1] = '\0';
}

void Person::Set()
{
    std::cout << "Enter first name: ";
    std::cin.getline(fname, Len);
    std::cout << "Enter last name: ";
    std::cin.getline(lname, Len);
}
void Gunslinger::Set()
{
    Person::Set();
    std::cout << "Enter draw time: ";
    std::cin >> drawtime;
    std::cout << "Enter number of notches: ";
    std::cin >> notches;
```

```
}

void Gunslinger::Show() const
{
    Person::Show();
    std::cout << ": " << drawtime << " drawtime, " << notches
          << " notches";
}


// pe14-4.cpp
#include <iostream>
#include <cstring>
#include "pe14-4pg.h"
const int SIZE = 5;
int main()
{
    using namespace std;
    int ct, i;
    Person * gang[SIZE];
    for (ct = 0; ct < SIZE; ct++)
    {
        char choice;
        cout << "Enter the gang category:\n"
            << "o: ordinary person  g: gunslinger  "
            << "p: pokerplayer  b: bad dude  q: quit\n";
        cin >> choice;
        while (strchr("ogpbq", choice) == NULL)
        {
            cout << "Please enter an o, g, p, b, or q: ";
            cin >> choice;
        }
        if (choice == 'q')
            break;
        switch(choice)
        {
            case 'o':   gang[ct] = new Person;
                        break;
            case 'g':   gang[ct] = new Gunslinger;
                        break;
            case 'p':   gang[ct] = new PokerPlayer;
                        break;
            case 'b':   gang[ct] = new BadDude;
                        break;
        }
        cin.get();
        gang[ct]->Set();
    }

    cout << "\nHere is your gang:\n";
    for (i = 0; i < ct; i++)
    {
        cout << '\n';
        gang[i]->Show();
    }
    for (i = 0; i < ct; i++)
        delete gang[i];
```

```
    cout << "\nBye!\n";
    //cin.get();
    //cin.get();
    return 0;
}
```

PE 14-5

```
// emp.h -- header file for abstr_emp class and children

#include <iostream>
#include <string>

class abstr_emp
{
private:
    std::string fname;    // abstr_emp's first name
    std::string lname;    // abstr_emp's last name
    std::string job;
public:
    abstr_emp();
    abstr_emp(const std::string & fn, const std::string &  ln,
              const std::string &  j);
    virtual void ShowAll() const;    // labels and shows all data
    virtual void SetAll();        // prompts user for values
    friend std::ostream & operator<<(std::ostream & os, const abstr_emp & e);
    // just displays first and last name
    virtual ~abstr_emp() = 0;         // virtual base class
};

class employee : public abstr_emp
{
public:
    employee();
    employee(const std::string & fn, const std::string &  ln,
              const std::string &  j);
    virtual void ShowAll() const;
    virtual void SetAll();
};

class manager:  virtual public abstr_emp
{
private:
    int inchargeof;        // number of abstr_emps managed
protected:
    int InChargeOf() const { return inchargeof; } // output
    int & InChargeOf(){ return inchargeof; }      // input
public:
    manager();
    manager(const std::string & fn, const std::string & ln,
            const std::string & j, int ico = 0);
    manager(const abstr_emp & e, int ico);
    manager(const manager & m);
    virtual void ShowAll() const;
    virtual void SetAll();
```

```
};

class fink: virtual public abstr_emp
{
private:
    std::string reportsto;        // to whom fink reports
protected:
    const std::string ReportsTo() const { return reportsto; }
    std::string & ReportsTo(){ return reportsto; }
public:
    fink();
    fink(const std::string & fn, const std::string & ln,
         const std::string & j, const std::string & rpo);
    fink(const abstr_emp & e, const std::string & rpo);
    fink(const fink & e);
    virtual void ShowAll() const;
    virtual void SetAll();
};

class highfink: public manager, public fink // management fink
{
public:
    highfink();
    highfink(const std::string & fn, const std::string & ln,
             const std::string & j, const std::string & rpo,
             int ico);
    highfink(const abstr_emp & e, const std::string & rpo, int ico);
    highfink(const fink & f, int ico);
    highfink(const manager & m, const std::string & rpo);
    highfink(const highfink & h);
    virtual void ShowAll() const;
    virtual void SetAll();
};

// emp.cpp -- abstr_emp class and children
#include "emp.h"
using std::string;
using std::cout;
using std::ostream;
using std::endl;
using std::cin;

abstr_emp::abstr_emp()
{
    fname = "";
    lname = "";
    job = "";
}

abstr_emp::abstr_emp(const string & fn, const string &  ln,
                  const string &  j)
          : fname(fn), lname(ln), job(j)
{
}

abstr_emp::~abstr_emp()
{
```

```
}

void abstr_emp::ShowAll() const
{
    cout << "Name: " << fname << " " << lname << endl;
    cout << "Job:  " << job << endl;
}

void abstr_emp::SetAll()
{
    cout << "First name: ";
    getline(cin, fname);
    cout << "Last name: ";
    getline(cin, lname);
    cout << "Job: ";
    getline(cin, job);
}

ostream & operator<<(ostream & os, const abstr_emp & e)
{
    os << e.fname << " " << e.lname;
    return os;
}

employee::employee(const std::string & fn, const std::string &  ln,
                   const std::string &  j)
    : abstr_emp(fn, ln, j)
{
}

void employee::ShowAll() const
{
    cout << "Employee:\n";
    abstr_emp::ShowAll();
}

void employee::SetAll()
{
    cout << "Employee input:\n";
    abstr_emp::SetAll();
}

manager::manager()
{
    inchargeof = 0;
}

manager::manager(const string & fn, const string &  ln,
                 const string &  j, int ico)
    : abstr_emp(fn, ln, j)
{
    inchargeof = ico;
}

manager::manager(const abstr_emp & e, int ico)
    : abstr_emp(e)
{
```

```
    inchargeof = ico;
}

manager::manager(const manager & m) : abstr_emp(m)
{
    inchargeof = m.inchargeof;
}

void manager::ShowAll() const
{
    cout << "Manager:\n";
    abstr_emp::ShowAll();
    cout << "In charge of " << inchargeof << endl;
}

void manager::SetAll()
{
    cout << "Manager input:\n";
    abstr_emp::SetAll();
    cout << "Number in charge of: ";
    cin >> inchargeof;
    while (cin.get() != '\n') continue;
}

fink::fink()
{
    reportsto = "";
}

fink::fink(const string & fn, const string &  ln,
                   const string &  j, const string & rpo)
    : abstr_emp(fn, ln, j), reportsto(rpo)
{
}

fink::fink(const abstr_emp & e, const string & rpo)
    : abstr_emp(e), reportsto(rpo)
{
 }

fink::fink(const fink & m) : abstr_emp(m), reportsto(m.reportsto)
{
}

void fink::ShowAll() const
{
    cout << "Fink:\n";
    abstr_emp::ShowAll();
    cout << "Reports to: " << reportsto << endl;
}

void fink::SetAll()
{
    cout << "Fink input: ";
    abstr_emp::SetAll();
    cout << "Reports to: ";
    getline(cin,reportsto);
```

```
}

highfink::highfink()
{
}

highfink::highfink(const string & fn, const string &  ln,
                   const string &  j, const string & rpo, int ico)
    : abstr_emp(fn, ln, j), manager(fn, ln, j, ico), fink(fn, ln, j, rpo)
{
}

highfink::highfink(const abstr_emp & e, const string & rpo, int ico)
    : abstr_emp(e), manager(e, ico), fink(e, rpo)
{
}

highfink::highfink(const fink & f, int ico)
    : abstr_emp(f), fink(f), manager(f, ico)
{
}

highfink::highfink(const manager & m, const string & rpo)
    : abstr_emp(m), manager(m), fink(m, rpo)
{
}

highfink::highfink(const highfink & h)
    : abstr_emp(h), manager(h), fink(h)
{
}

void highfink::ShowAll() const
{
    cout << "High Fink:\n";
    abstr_emp::ShowAll();
    cout << "In charge of " << InChargeOf() << endl;
    cout << "Reports to: " << ReportsTo() << endl;
}

void highfink::SetAll()
{
    cout << "High Fink input:\n";
    abstr_emp::SetAll();
    cout << "Reports to: ";
    getline(cin,ReportsTo());
    cout << "Number in charge of: ";
    cin >> InChargeOf();
}

// pe14-5.cpp
// useemp1.cpp -- using the abstr_emp classes

#include <iostream>
using namespace std;
#include "emp.h"
```

```
int main()
{
    employee em("Trip", "Harris", "Thumper");
    cout << em << endl;
    em.ShowAll();

    manager ma("Amorphia", "Spindragon", "Nuancer", 5);
    cout << ma << endl;
    ma.ShowAll();


    fink fi("Matt", "Oggs", "Oiler", "Juno Barr");
    cout << fi << endl;
    fi.ShowAll();
    highfink hf(ma, "Curly Kew");  // recruitment?
    hf.ShowAll();
    cout << "Press a key for next phase:\n";
    cin.get();
    highfink hf2;
    hf2.SetAll();

    cout << "Using an abstr_emp * pointer:\n";
    abstr_emp  * tri[4] = {&em, &fi, &hf, &hf2};
    for (int i = 0; i < 4; i++)
        tri[i]->ShowAll();
    //cin.get();
    //cin.get();
    return 0;
}
```

Chapter 15

PE 15-1

```
// pe15tv.h -- Tv and Remote classes

#ifndef PE15TV_H_
#define PE15TV_H_

class Tv
{
public:
    friend class Remote;    // Remote can access Tv private parts
    enum State{Off, On};
    enum {MinVal,MaxVal = 20};
    enum {Antenna, Cable};
    enum {TV, DVD};

    Tv(State s = Off, int mc = 125) : state(s), volume(5),
        maxchannel(mc), channel(2), mode(Cable), input(TV) {}
    void onoff() {state = (state == On)? Off : On;}
    bool ison() {return state == On ? true : false;}
    bool volup();
    bool voldown();
    void chanup();
```

```
    void chandown();
    void set_mode() {mode = (mode == Antenna)? Cable : Antenna;}
    void set_input() {input = (input == TV)? DVD : TV;}
    void settings();
    void rmode(Remote & r);
private:
    State state;
    int volume;
    int maxchannel;
    int channel;
    int mode;
    int input;
};

class Remote
{
friend class Tv;

public:
    enum Style {Normal, Interactive};
    Remote(int m = Tv::TV, int s = Normal) :
                mode(m), style(s) {}
    bool volup(Tv & t) { return t.volup();}
    bool voldown(Tv & t) { return t.voldown();}
    void onoff(Tv & t) { t.onoff(); }
    void chanup(Tv & t) {t.chanup();}
    void chandown(Tv & t) {t.chandown();}
    void set_chan(Tv & t, int c) {t.channel = c;}
    void set_mode(Tv & t) {t.set_mode();}
    void set_input(Tv & t) {t.set_input();}
    void show_style();
private:
    int mode;     // TV or DVD
    int style;  // Normal or Interactive
};

// place definition here where both Tv and Remote
// class declarations are known
inline void Tv::rmode(Remote & r)
{
    if(state == Off)
        return;
    if (r.style == Remote::Normal)
        r.style = Remote::Interactive;
    else r.style = Remote::Normal;
}
#endif


// pe15tv.cpp

#include <iostream>
#include "pe15tv.h"

bool Tv::volup()
{
    if (volume < MaxVal)
```

```
    {
        volume++;
        return true;
    }
    else
        return false;
}
bool Tv::voldown()
{
    if (volume > MinVal)
    {
        volume--;
        return true;
    }
    else
        return false;
}

void Tv::chanup()
{
    if (channel < maxchannel)
        channel++;
    else
        channel = 1;
}

void Tv::chandown()
{
    if (channel > 1)
        channel--;
    else
        channel = maxchannel;
}

void Tv::settings()
{
    using std::cout;
    cout << "TV is " << (state == Off? "Off\n" : "On\n");
    if (state == On)
    {
        cout << "Volume setting = " << volume << "\n";
        cout << "Channel setting = " << channel << "\n";
        cout << "Mode = "
            << (mode == Antenna? "antenna\n" : "cable\n");
        cout << "Input = "
            << (input == TV? "TV\n" : "DVD\n");
    }
}

void Remote::show_style()
{
    if (style == Normal)
        std::cout << "Remote in Normal mode\n";
    else
        std::cout << "Remote in Interactive mode\n";
}
```

```
// pe15-1.cpp
// link with pe15tv.cpp

#include <iostream>
#include "pe15tv.h"

int main()
{
    Tv s42;
    std::cout << "Initial settings for 42\" TV:\n";
    s42.settings();
    s42.onoff();
    s42.chanup();
    std::cout << "\nAdjusted settings for 42\" TV:\n";
    s42.settings();

    Remote grey;

    grey.set_chan(s42, 10);
    grey.volup(s42);
    grey.volup(s42);
    std::cout << "\n42\" settings after using remote\n";
    s42.settings();

    Tv s58(Tv::On);
    s58.set_mode();
    grey.set_chan(s58,28);
    std::cout << "\n58\" settings:\n";
    s58.settings();
    grey.show_style();    // check mode
    s58.rmode(grey);      // change mode
    grey.show_style();    // recheck mode
    s58.onoff();          // turn set off
    s58.rmode(grey);      // try changing mode again
    grey.show_style();    // check result
    //std::cin.get();
    return 0;
}
```

PE 15-2

```
// pe15-2.h  -- exception classes for hmean(), gmean()
#ifndef PE15_2_H_
#define PE15_2_H_

#include <iostream>
#include <stdexcept>

class hmeanexcp : public std::logic_error
{
public:
    hmeanexcp()
     : std::logic_error("hmean() invalid arguments: a = -b\n")
     {
```

```
     }
};

class gmeanexcp : public std::logic_error
{
public:
    gmeanexcp()
     : std::logic_error("gmean()  arguments should be >= 0\n")
     {

     }
};
#endif

//pe15-2.cpp
#include <iostream>
#include <cmath> // or math.h, unix users may need -lm flag
#include "pe15-2.h"
// function prototypes
double hmean(double a, double b);  // throws hmeanexcp
double gmean(double a, double b);  // throws gmeanexcp
int main()
{
    using std::cout;
    using std::cin;
    using std::endl;

    double x, y, z;

    cout << "Enter two numbers: ";
    while (cin >> x >> y)
    {
        try {                   // start of try block
            z = hmean(x,y);
            cout << "Harmonic mean of " << x << " and " << y
                << " is " << z << endl;
            cout << "Geometric mean of " << x << " and " << y
                << " is " << gmean(x,y) << endl;
            cout << "Enter next set of numbers <q to quit>: ";
        }// end of try block
        catch (hmeanexcp & bg)    // start of catch block
        {
            cout << bg.what();
            cout << "Try again.\n";
            continue;
        }
        catch (gmeanexcp & bh)
        {
               cout << bh.what();
            cout << "Sorry, you don't get to play any more.\n";
            break;
        } // end of catch block
    }
    cout << "Bye!\n";
/* to keep window open
    cin.clear();
```

```
    while (cin.get() != '\n')
        continue;
    cin.get();
*/
    return 0;
}

double hmean(double a, double b) // throws hmeanexcp
{
    if (a == -b)
        throw hmeanexcp();
    return 2.0 * a * b / (a + b);
}

double gmean(double a, double b) // throws gmeanexcp
{
    if (a < 0 || b < 0)
        throw gmeanexcp();
    return std::sqrt(a * b);
}
```

PE 15-3

```
// pe15-3.h  -- exception classes for hmean(), gmean()
#ifndef PE15_3_H_
#define PE15_3_H_

#include <iostream>
#include <stdexcept>

class bad_args : public std::logic_error
{
private:
    double a;
    double b;
public:
    bad_args(const char * s, double aa, double bb)
     : std::logic_error(s), a(aa), b(bb)
     {
     }
     virtual void values() const;
};

void bad_args::values() const
{

    std::cout << "argument values: " << a << ", "
              << b << std::endl;
}

class hmeanexcp : public bad_args
{
public:
    explicit hmeanexcp(double aa = 0, double bb = 0)
     : bad_args("hmean() invalid arguments: a = -b\n", aa, bb)
```

```
     {
     }

};

class gmeanexcp : public bad_args
{
public:
    explicit gmeanexcp(double aa = 0, double bb = 0)
     : bad_args("gmean()  arguments should be >= 0\n", aa, bb)

     {
     }

};

#endif

//pe15-3.cpp
#include <iostream>
#include <cmath> // or math.h, unix users may need -lm flag
#include "pe15-3.h"
// function prototypes
double hmean(double a, double b); // throws hmeanexcp
double gmean(double a, double b); // throws gmeanexcp
int main()
{
    using std::cout;
    using std::cin;
    using std::endl;

    double x, y, z;

    cout << "Enter two numbers: ";
    while (cin >> x >> y)
    {
        try {                   // start of try block
            z = hmean(x,y);
            cout << "Harmonic mean of " << x << " and " << y
                << " is " << z << endl;
            cout << "Geometric mean of " << x << " and " << y
                << " is " << gmean(x,y) << endl;
            cout << "Enter next set of numbers <q to quit>: ";
        }// end of try block
        catch (bad_args & ba)    // start of catch block
        {
            cout << ba.what();
            ba.values();
            cout << "Sorry, you don't get to play any more.\n";
            break;
        } // end of catch block
    }
    cout << "Bye!\n";
/* to keep window open
    cin.clear();
    while (cin.get() != '\n')
```

```
        continue;
    cin.get();
*/
    return 0;
}

double hmean(double a, double b) // throws hmeanexcp
{
    if (a == -b)
        throw hmeanexcp(a,b);
    return 2.0 * a * b / (a + b);
}

double gmean(double a, double b) // throws gmeanexcp
{
    if (a < 0 || b < 0)
        throw gmeanexcp(a,b);
    return std::sqrt(a * b);
}
```

PE 15-4

```
// pe15sales.h  -- exceptions and inheritance
#include <stdexcept>
#include <cstring>

class Sales
{
public:
    enum {MONTHS = 12};   // could be a static const
    class bad_index : public std::logic_error
    {
    private:
        int bi;  // bad index value
    public:
        explicit bad_index(int ix,
            const char * s = "Index error in Sales object\n");
        int bi_val() const {return bi;}
        virtual ~bad_index() throw() {}
    };
    explicit Sales(int yy = 0);
    Sales(int yy, const double * gr, int n);
    virtual ~Sales() { }
    int Year() const { return year; }
    virtual double operator[](int i) const;
    virtual double & operator[](int i);
private:
    double gross[MONTHS];
    int year;
};

class LabeledSales : public Sales
{
  public:
    static const int STRLEN = 50;  // could be an enum
```

```
    class nbad_index : public Sales::bad_index
    {
    private:
        char lbl[STRLEN];
    public:
        nbad_index(const char * lb, int ix,
           const char * s = "Index error in LabeledSales object\n");
        const char * label_val() {return lbl;}
        virtual ~nbad_index() throw() {}
    };
    explicit LabeledSales(const char * lb = "none", int yy = 0);
    LabeledSales(const char * lb, int yy, const double * gr, int n);
    virtual ~LabeledSales() { }
    const char * Label() const {return label;}
    virtual double operator[](int i) const;
    virtual double & operator[](int i);
private:
    char label[STRLEN];
};


// pe15sales.cpp -- Sales implementation
#include "pe15sales.h"

Sales::bad_index::bad_index(int ix, const char * s )
    : std::logic_error(s), bi(ix)
{
}

Sales::Sales(int yy)
{
    year = yy;
    for (int i = 0; i < MONTHS; ++i)
        gross[i] = 0;
}

Sales::Sales(int yy, const double * gr, int n)
{
    year = yy;
    int lim = (n < MONTHS)? n : MONTHS;
    int i;
    for (i = 0; i < lim; ++i)
        gross[i] = gr[i];
    // for i > n and i < MONTHS
    for ( ; i < MONTHS; ++i)
        gross[i] = 0;
}

double Sales::operator[](int i) const
{
    if(i < 0 || i >= MONTHS)
        throw bad_index(i);
    return gross[i];
}

double & Sales::operator[](int i)
{
```

```
    if(i < 0 || i >= MONTHS)
        throw bad_index(i);
    return gross[i];
}

LabeledSales::nbad_index::nbad_index(const char * lb, int ix,
           const char * s ) : Sales::bad_index(ix, s)
{
    std::strcpy(lbl, lb);
}

LabeledSales::LabeledSales(const char * lb, int yy)
         : Sales(yy)
{
    std::strcpy(label, lb);
}

LabeledSales::LabeledSales(const char * lb, int yy, const double * gr, int n)
        : Sales(yy, gr, n)
{
    std::strcpy(label, lb);
}

double LabeledSales::operator[](int i) const
{
    if(i < 0 || i >= MONTHS)
        throw nbad_index(Label(), i);
    return Sales::operator[](i);
}

double & LabeledSales::operator[](int i)
{
    if(i < 0 || i >= MONTHS)
        throw nbad_index(Label(), i);
    return Sales::operator[](i);
}


// pe15-4.cpp
#include <iostream>
#include "pe15sales.h"
// function prototypes
int main()
{
    using std::cout;
    using std::cin;
    using std::endl;

    double vals1[12] =
    {
        1220, 1100, 1122, 2212, 1232, 2334,
        2884, 2393, 3302, 2922, 3002, 3544
    };

    double vals2[12] =
    {
        12, 11, 22, 21, 32, 34,
```

```
        28, 29, 33, 29, 32, 35
    };

    Sales one(2004, vals1, 12);
    LabeledSales two("Blogstar",2005, vals2, 12 );

    cout << "First try block:\n";
    try
    {
        int i;
        cout << "Year = " << one.Year() << endl;
        for (i = 0; i < 12; ++i)
        {

            cout << one[i] << ' ';
            if (i % 6 == 5)
                cout << endl;
        }
        cout << endl;
        cout << "Year = " << two.Year() << endl;
        cout << "Label = " << two.Label() << endl;
        for (i = 0; i <= 12; ++i)
        {

            cout << two[i] << ' ';
            if (i % 6 == 5)
                cout << endl;
        }
        cout << "End of try block 1.\n";
    }
    catch(Sales::bad_index & bad)
    {
        cout << bad.what();
        LabeledSales::nbad_index * pn =
            dynamic_cast<LabeledSales::nbad_index *>(&bad);
        if (pn)
            cout << "Company: " << pn->label_val() << endl;
        cout << "bad index: " << bad.bi_val() << endl;

    }
    cout << "Next try block:\n";
    try
     {
        two[2] = 37.5;
        one[20] = 23345;
        cout << "End of try block 2.\n";
    }
    catch(Sales::bad_index & bad)
    {
        cout << bad.what();
        LabeledSales::nbad_index * pn =
            dynamic_cast<LabeledSales::nbad_index *>(&bad);
        if (pn)
            cout << "Company: " << pn->label_val() << endl;
        cout << "bad index: " << bad.bi_val() << endl;

    }
```

```
   cout << "done\n";
   //cin.get();
   return 0;
}
```

Chapter 16

PE 16-1

```
// pe16-1.cpp -- one of many possible solutions
#include <iostream>
#include <string>

bool isPal(const std::string & s);

int main()
{
    std::string input;

    std::cout << "Enter a string (empty string to quit):\n";
    std::getline(std::cin,input);
    while (std::cin && input.size() > 0)
    {
        if (isPal(input))
            std::cout << "That was a palindrome!\n";
        else
            std::cout << "That was not a palindrome!\n";
        std::cout << "Enter a string (empty string to quit):\n";
        std::getline(std::cin,input);
    }
    std::cout << "Bye!\n";
    //std::cin.get();
    return 0;
}

bool isPal(const std::string & s)
{
    std::string rev(s.rbegin(), s.rend()); // construct reversed string

    // some older compilers don't implement the above constructor
    // another approach is this
    // std::string rev(s);    // rev same size as s
    // copy(s.rbegin(), s.rend(), rev.begin());

    return (rev == s);
}
```

PE 16-2

```
//pe16-2.cpp
#include <iostream>
#include <algorithm>
#include <string>
```

```
#include <cctype>

bool isPal(const std::string & s);
std::string reduce(const std::string & s);
std::string toLower(const std::string & s);
bool reject(char ch);
char tolow(char ch);

int main()
{
    std::string input, reduced;

    std::cout << "Enter a string (empty string to quit):\n";
    std::getline(std::cin,input);
    while (std::cin && input.size() > 0)
    {
        reduced = reduce(input);
        if (isPal(reduced))
            std::cout << "That was a palindrome!\n";
        else
            std::cout << "That was not a palindrome!\n";
        std::cout << "Enter a string (empty string to quit):\n";
        std::getline(std::cin,input);
    }
    std::cout << "Bye!\n";
    std::cin.get();
    return 0;
}

std::string reduce(const std::string & s)
{
    std::string rd = toLower(s);
    std::string::iterator newend =
        std::remove_if(rd.begin(), rd.end(), reject);
    return std::string(rd.begin(), newend);
}

bool isPal(const std::string & s)
{
    std::string rev(s.rbegin(), s.rend());
    std::cout << s <<  " : " << rev << std::endl;
    return (rev == s);
}

char tolow(char ch) { return std::tolower(ch); }

std::string toLower(const std::string & s)
{
    std::string low(s);
    std::transform(s.begin(), s.end(), low.begin(), tolow);
    return low;
}

bool reject(char ch)
{
    return !std::isalpha(ch);
}
```

PE 16-3

```
// hangman.cpp -- some string methods
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <cstdlib>
#include <ctime>
#include <cctype>

int main()
{
    using std::cout;
    using std::cin;
    using std::tolower;
    using std::endl;
    using std::ifstream;
    using std::string;
    using std::vector;

    ifstream fin;
    fin.open("words.txt");
    if(fin.is_open() == false)
    {
        std::cerr << "Failed to open word file; bye\n";
        cin.get();
        std::exit(EXIT_FAILURE);
    }
    vector<string> wordlist;
    string temp;
    while (fin >> temp)
        wordlist.push_back(temp);
    int num = wordlist.size();
    std::srand(std::time(0));
    char play;
    cout << "Will you play a word game? <y/n> ";
    cin >> play;
    play = tolower(play);
    while (play == 'y')
    {
        string target = wordlist[std::rand() % num];
        int length = target.length();
        string attempt(length, '-');
        string badchars;
        int guesses = 6;
        cout << "Guess my secret word. It has " << length
            << " letters, and you guess\n"
            << "one letter at a time. You get " << guesses
            << " wrong guesses.\n";
        cout << "Your word: " << attempt << endl;
        while (guesses > 0 && attempt != target)
        {
```

```
            char letter;
            cout << "Guess a letter: ";
            cin >> letter;
            if (badchars.find(letter) != string::npos
                || attempt.find(letter) != string::npos)
            {
                cout << "You already guessed that. Try again.\n";
                    continue;
            }
            int loc = target.find(letter);
            if (loc == string::npos)
            {
                cout << "Oh, bad guess!\n";
                --guesses;
                badchars += letter; // add to string
            }
            else
            {
                cout << "Good guess!\n";
                attempt[loc]=letter;
                // check if letter appears again
                loc = target.find(letter, loc + 1);
                while (loc != string::npos)
                {
                    attempt[loc]=letter;
                    loc = target.find(letter, loc + 1);
                }
           }
            cout << "Your word: " << attempt << endl;
            if (attempt != target)
            {
                if (badchars.length() > 0)
                    cout << "Bad choices: " << badchars << endl;
                cout << guesses << " bad guesses left\n";
            }
        }
        if (guesses > 0)
            cout << "That's right!\n";
        else
            cout << "Sorry, the word is " << target << ".\n";

        cout << "Will you play another? <y/n> ";
        cin >> play;
        play = tolower(play);
    }
    fin.close();
    cout << "Bye\n";
    //cin.get();
    //cin.get();
    return 0;
}
```

PE 16-4

```
// pe16-4.cpp -- one possibility
#include <iostream>
```

```
#include <algorithm>
#define MAX 10

int reduce(long ar[], int n);
void show(const long ar[], int n);

int main()
{
    long myarray[MAX] = {12, 12 ,5, 6, 11, 5, 6, 77, 11,12};

    show(myarray, MAX);

    int newsize = reduce(myarray,MAX);
    show(myarray, newsize);
    //std::cin.get();
    return (0);
}

int reduce(long ar[], int n)
{
    // or one could copy to a list and use list methods
    // or copy to a set; in either case, copy results
    // back to array
    std::sort(ar, ar + n);
    long * past_end;
    past_end = std::unique(ar, ar + n);
    return past_end - ar;
}

void show(const long ar[], int n)
{
    for (int i = 0; i < n; i++)
        std::cout << ar[i] << ' ';
    std::cout << std::endl;
}
```

PE 16-5

```
// pe16-5.cpp
#include <iostream>
#include <algorithm>
#include <string>

#define MAX 10
#define MAXSTR 5
template <class T>
int reduce(T ar[], int n);

void show(const long ar[], int n);
void show(const std::string ar[], int n);

int main()
{
    long myarray[MAX] = {12, 12 ,5, 6, 11, 5, 6, 77, 11,12};
    std::string msgs[MAXSTR] = {
```

```
        "Hello, there",
        "Any number will do",
        "Hello, there",
        "Zoo day is tomorrow",
        "Hello, there"
    };
    show(myarray, MAX);
    show(msgs, MAXSTR);
    int newsize1 = reduce(myarray,MAX);
    show(myarray, newsize1);

    int newsize2 = reduce(msgs,MAXSTR);
    show(msgs, newsize2);
    //std::cin.get();
    return 0;
}

template <class T>
int reduce(T ar[], int n)
{
    std::sort(ar, ar + n);
    T * past_end;
    past_end = std::unique(ar, ar + n);
    return past_end - ar;
}

void show(const long ar[], int n)
{
    for (int i = 0; i < n; i++)
        std::cout << ar[i] << ' ';
    std::cout << std::endl;
}

void show(const std::string ar[], int n)
{
    for (int i = 0; i < n; i++)
        std::cout << ar[i] << std::endl;
}
```

PE 16-6

```
//customer.h for pe16-6

#ifndef CUSTOMER_H_
#define CUSTOMER_H_

class Customer
{
private:
    long arrive;
    int processtime;
public:
    //Customer() {arrive = processtime = 0; }
    void set(long when);
    long when() const {return arrive;}
    int ptime() const {return processtime;}
```

```
};

#endif

// customer.cpp for pe16-6

#include "customer.h"
#include <cstdlib>
void Customer::set(long when)
{
    processtime = std::rand() % 3 + 1;
    arrive = when;
}

// pe16-6.cpp

#include <iostream>
#include <queue>
#include <cstdlib>
#include <ctime>
#include "customer.h"

const int MIN_PER_HR = 60;

bool newcustomer(double x);

int main()
{
    using std::cout;
    using std::cin;
    using std::endl;
    using std::ios_base;

    std::srand(std::time(0));
    cout << "Case study" << endl;
    cout << "Enter max size of queue : ";
    int qs;
    cin >> qs;
    std::queue<Customer> line;

    cout << "enter number of simulation hours : ";
    int hours;
    cin >> hours;

    long cyclelimit = MIN_PER_HR * hours;

    cout << "Enter number of customers per hour : ";
    double perhour;
    cin >> perhour;
    double min_per_cust;
    min_per_cust = MIN_PER_HR / perhour;

    Customer temp;
    long turnaways = 0;
    long customers = 0;
    long served = 0;
    long sum_line = 0;
```

```
    int wait_time = 0;
    long line_wait = 0;

    for (int cycle = 0; cycle < cyclelimit; cycle++)
    {
        if (newcustomer(min_per_cust))
        {
            if (line.size() >= qs)
                turnaways++;
            else
            {
                customers++;
                temp.set(cycle);
                line.push(temp);
            }
        }
        if (wait_time <= 0 && !line.empty())
        {
            temp = line.front(); // recover element
            line.pop();          // remove from queue
            wait_time = temp.ptime();
            line_wait += cycle -temp.when();
            served++;
        }
        if (wait_time > 0)
            wait_time--;
        sum_line += line.size();
    }

    if (customers > 0)
    {
        cout << "customers accepted: " << customers << endl;
        cout << "  customers served: " << served << endl;
        cout << "         turnaways: " << turnaways << endl;
        cout << "average queue size: ";
        cout.precision(2);
        cout.setf(ios_base::fixed, ios_base::floatfield);
        cout.setf(ios_base::showpoint);
        cout << (double)sum_line / cyclelimit << endl;
        cout << " average wait time: "
            << (double)line_wait / served << " minutes\n";
    }
    else
        cout << "No customers!\n";
    //cin.get();
    //cin.get();
    return(0);
}

bool newcustomer(double x)
{
    return(std::rand() * x / RAND_MAX < 1);
}
```

PE 16-7

```
// pe16-7.cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

vector<int> Lotto(int, int);
void Show(int);

int main()
{
    vector<int> results;
    int spots;
    int picks;
    cout << "Enter spots and picks: ";
    while (cin >> spots >> picks)
    {
        if (spots < picks)
        {
            cout << "Number of spots cannot be less than number\n"
                 << "of picks. Try again.\n";
            continue;
        }
        results = Lotto(spots, picks);
        for_each(results.begin(), results.end(), Show);
        cout << endl;
        cout << "Enter spots and picks (q to quit): ";
   }
    cout << "Done\n";
/* to keep window open
    cin.clear();
    while (cin.get() != '\n')
        continue;
    cin.get();
*/
    return 0;
}

void Show(int n)
{
    cout << n << ' ';
}

vector<int> Lotto(int sp, int ps)
{
    vector<int> choices(sp);
    for (int i = 0; i < sp; i++)
        choices[i] = i + 1;

    random_shuffle(choices.begin(), choices.end());
    vector<int> picks(ps);
    copy(choices.begin(), choices.begin() + ps, picks.begin());
    sort(picks.begin(), picks.end());
    return picks;
}
```

PE 16-8

```
// pe16-8.cpp

#include <iostream>
#include <set>
#include <algorithm>
#include <iterator>
#include <cstdlib>
#include <string>

int main()
{
    using namespace std;
    string temp;

    set<string> mats;
    cout << "Enter Mat's guest list (empty line to quit):\n";
    while (getline(cin,temp) && temp.size() > 0)
        mats.insert(temp);
    ostream_iterator<string,char> out (cout, "\n");
    cout << "Mat's guest list:\n";
    copy(mats.begin(), mats.end(), out);

    set<string> pats;
    cout << "Enter Pat's guest list (empty line to quit):\n";
    while (getline(cin,temp) && temp.size() > 0)
        pats.insert(temp);
    cout << "\nPat's guest list:\n";
    copy(pats.begin(), pats.end(), out);

    set<string> both;
    set_union(mats.begin(), mats.end(), pats.begin(), pats.end(),
        insert_iterator<set<string> >(both, both.begin()));
    cout << "\nMerged guest list:\n";
    copy(both.begin(), both.end(), out);
    //std::cin.get();
    return 0;
}
```

PE 16-9

```
// pe16-9.cpp
#include <iostream>
#include <vector>
#include <list>
#include <cmath>
#include <ctime>
#include <cstdlib>
#include <algorithm>

int main()
```

```
{
   using namespace std;
   int n_elem;
   cout << "Enter number of elements: ";
   cin >> n_elem;
   vector<int> vi0(n_elem);
   for (auto pv = vi0.begin(); pv != vi0.end(); pv++)
      *pv  = rand();
   vector<int> vi(vi0);

   clock_t start = clock();
   sort(vi.begin(), vi.end());
   clock_t end = clock();
   cout.precision(3);
   cout << (double)(end - start)/CLOCKS_PER_SEC;
   cout << " = elapsed time for sorting vector\n";
   list<int> li(vi.size());
   copy(vi0.begin(), vi0.end(), li.begin());
   list<int> licp(li);

   start = clock();
   li.sort();
   end = clock();
   cout << (double)(end - start)/CLOCKS_PER_SEC;
   cout << " = elapsed time for list sort\n";

   start = clock();
   copy(licp.begin(), licp.end(), vi.begin());
   sort(vi.begin(), vi.end());
   copy(vi.begin(), vi.end(), licp.begin());
   end = clock();
   cout << (double)(end - start)/CLOCKS_PER_SEC;
   cout << " = elapsed time to copy list to vector,"
       << "sort, and copy result back to list\n";
   cout << "done\n";
    //cin.get();
    //cin.get();
   return 0;
}
```

PE 16-10

```
// pe16-10.cpp -- using STL functions
#include <iostream>
#include <iomanip>   // see Chapter 17
#include <cctype>
#include <string>
#include <vector>
#include <algorithm>
#include <memory>

struct Review {
    std::string title;
    int rating;
    float price;
};
```

```
typedef std::shared_ptr<Review> spr;
bool operator<(const spr & r1, const spr & r2);
bool worseThan(const spr & r1, const spr & r2);
bool cheaperThan(const spr & r1, const spr & r2);
bool FillReview(Review & rr);
void ShowReview(const spr & rr);
void Heading();
char getchoice();
int main()
{
    using namespace std;

    vector<spr> books;
    Review temp;
    while (FillReview(temp))
        books.push_back(shared_ptr<Review> (new Review (temp)));
    cout << "Thank you. You entered the following "
         << books.size() << " ratings:\n";
    Heading();
    for_each(books.begin(), books.end(), ShowReview);

// create arrays of sorted pointers
    vector<spr> byName(books);
    sort(byName.begin(), byName.end());
    vector<spr> byRating(books);
    sort(byRating.begin(), byRating.end(), worseThan);
    vector<spr> byPrice(books);
    sort(byPrice.begin(), byPrice.end(), cheaperThan);

    char choice;
    while ((choice = getchoice()) != 'q')
    {
        switch(choice)
        {
            case 'o' : cout << "Original order:\n";
                       Heading();
                       for_each(books.begin(), books.end(),
                                ShowReview);
                       break;
            case 's' : cout << "Sorted by title:\n";
                       Heading();
                       for_each(byName.begin(), byName.end(),
                                ShowReview);
                       break;
            case 'i' : cout << "By increasing ratings:\n";
                       Heading();
                       for_each(byRating.begin(), byRating.end(),
                                ShowReview);
                       break;
            case 'd' : cout << "By decreasing ratings:\n";
                       Heading();
                       for_each(byRating.rbegin(), byRating.rend(),
                                ShowReview);
                       break;
            case 'h' : cout << "From lower to higher prices:\n";
                       Heading();
```

```
                        for_each(byPrice.begin(), byPrice.end(),
                                 ShowReview);
                        break;
                case 'l' : cout << "From higher to lower prices:\n";
                        Heading();
                        for_each(byPrice.rbegin(), byPrice.rend(),
                                 ShowReview);
                        break;
                default  : cout << "PROGRAMMING ERROR\n!";
                        break;
            }
        }

        cout << "Bye.\n";
        //cin.get();
        return 0;
}

bool operator<(const spr & r1, const spr & r2)
{
    if (r1->title < r2->title)
        return true;
    else if (r1->title == r2->title && r1->rating < r2->rating)
        return true;
    else
        return false;
}

bool worseThan(const spr & r1, const spr & r2)
{
    if (r1->rating < r2->rating)
        return true;
    else
        return false;
}

bool cheaperThan(const spr & r1, const spr & r2)
{
    if (r1->price < r2->price)
        return true;
    else
        return false;
}

bool FillReview(Review & rr)
{
    std::cout << "Enter book title (quit to quit): ";
    std::getline(std::cin,rr.title);
    if (rr.title == "quit")
        return false;
    std::cout << "Enter book rating: ";
    std::cin >> rr.rating;
    if (!std::cin)
        return false;
    std::cin.get();
    std::cout << "Enter book price: ";
    std::cin >> rr.price;
```

```
    if (!std::cin)
        return false;
    while (std::cin.get() != '\n')
        continue;
    return true;
}

void ShowReview(const spr & rr)
{
    using std::cout;
    using std::setw;
    cout << std::fixed << std::setprecision(2);
    cout << std::left << setw(7) << rr->rating
         << setw(40) << rr->title
         << std::right << setw(3) << "$"
         << setw(6) << rr->price << '\n';
}

void Heading()
{
    using namespace std;
    cout << setw(7) << left << "Rating"
         << setw(40) << " Book"
         << setw(8) << right << "Price" << endl;
}

char getchoice()
{
    using std::cin;
    using std::cout;
    static std::string responses = "osidhlq";
    char ch;
    do
    {
        cout << "Choose wisely:\n";
        cout << "o -- original order           s -- sorted by title\n"
             << "i -- by increasing rating     d -- by decreasing rating\n"
             << "h -- by increasing price      l -- by decreasing price\n";
        ch = tolower(cin.get());
        while (cin.get() != '\n')
            continue;
    } while (responses.find(ch) == std::string::npos);
    return ch;
}
```

Chapter 17

PE 17-1

```
// pe17-1.cpp
#include <iostream>

int main()
{
```

```
    using namespace std;
    char ch;
    int count = 0;

    while (cin.get(ch) && ch != '$')
        count++;
    if (ch == '$')
        cin.putback(ch);
    else
        cout << "End of input was reached\n";
    cout << count << " characters read\n";
    cin.get(ch);
    cout << "Then next input character is " << ch << endl;
/* keeping window open
    while (cin.get()  != '\n')
        continue;
    cin.get();
*/
    return 0;
}
```

PE 17-2

```
// pe17-2.cpp
#include <iostream>
#include <fstream>
#include <cstdlib>

// if your system doesn't support command-line arguments,
// prompt the user to enter a file name
int main(int argc, char * argv[])
{
    using namespace std;

    if (argc < 2)
    {
        cerr << "Usage: " << argv[0] << " filename\n";
        exit(EXIT_FAILURE);
    }
    ofstream fout(argv[1]);
    char ch;
    while (cin.get(ch))
        fout << ch;
    fout.close();
    cout << "Input copied to " << argv[1] << endl;
    return 0;
}
```

PE 17-3

```
// pe17-3.cpp
#include <iostream>
#include <fstream>
#include <cstdlib>
```

```
int main(int argc, char * argv[])
{
    using namespace std;
    if (argc < 3)
    {
        cerr << "Usage: " << argv[0]
            << " source-file target-file\n";
        exit(EXIT_FAILURE);
    }
    ifstream fin(argv[1]);
    if (!fin)
    {
        cerr << "Can't open " << argv[1] << " for input\n";
        exit(EXIT_FAILURE);
    }
    ofstream fout(argv[2]);
    if (!fout)
    {
        cerr << "Can't open " << argv[2] << " for output\n";
        exit(EXIT_FAILURE);
    }
    char ch;
    while (fin.get(ch))
        fout << ch;
    cout << "Contents of " << argv[1] << " copied to "
         << argv[2] << endl;
    fin.close();
    fout.close();
    return 0;
}
```

PE 17-4

```
// pe17-4.cpp
// THIS CODE ASSUMES ALL LINES IN THE TEXT FILES
// TERMINATE IN A NEW LINE CHARACTER. SOME EDITORS (INCLUDING
// THE CODEWARRIOR EDITOR) DON'T AUTOMATICALLY APPEND A NEWLINE
// TO THE LAST LINE OF A FILE. FOR SUCH EDITORS, YOU CAN PRESS
// THE RETURN KEY AT THE END OF THE LAST LINE TO ADD A NEWLINE.
#include <iostream>
#include <fstream>
#include <cstdlib>

int main()
{
    using namespace std;
    ifstream f1("file1");
    if (!f1)
    {
        cerr << "Can't open file1.\n";
        exit(EXIT_FAILURE);
    }
    ifstream f2("file2");
    if (!f2)
    {
```

```
    cerr << "Can't open file2.\n";
    exit(EXIT_FAILURE);
}
ofstream fout("outfile");
if (!fout)
{
    cerr << "Can't open outfile.\n";
    exit(EXIT_FAILURE);
}
char ch1, ch2;

f1.get(ch1);
f2.get(ch2);
while(f1 && f2)
{
    fout << ch1;
    f1.get(ch1);
    while (f1 && ch1 != '\n')
    {
        fout << ch1;
        f1.get(ch1);
    }
    if (f1)
    {
        fout << ' ';
        f1.get(ch1);
    }
    fout << ch2;
    f2.get(ch2);
    while (f2 && ch2 != '\n')
    {
        fout << ch2;
        f2.get(ch2);
    }
    if (f2)
    {
        fout << '\n';
        f2.get(ch2);
    }
}

while (f1)
{
    fout << ch1;
    f1.get(ch1);
}

while (f2)
{
    fout << ch2;
    f2.get(ch2);
}

f1.close();
f2.close();
fout.close();
cout << "\ndone\n";
```

```
    return 0;
}
```

PE 17-5

```
// pe17-5.cpp
#include <iostream>
#include <fstream>
#include <set>
#include <algorithm>
#include <iterator>
#include <cstdlib>
#include <string>

int main()
{
    using namespace std;
    ifstream mat("mat.dat");
    if (!mat.is_open())
    {
        cerr << "Can't open mat.dat.\n";
        exit(1);
    }
    ifstream pat("pat.dat");
    if (!pat.is_open())
    {
        cerr << "Can't open pat.dat.\n";
        exit(1);
    }

    ofstream matnpat("matnpat.dat");
    if (!matnpat.is_open())
    {
        cerr << "Can't open pat.dat.\n";
        exit(1);
    }

    string temp;

    set<string> mats;
    while (getline(mat,temp))
        mats.insert(temp);
    ostream_iterator<string,char> out (cout, "\n");
    cout << "Mat's guest list:\n";
    copy(mats.begin(), mats.end(), out);

    set<string> pats;
    while (getline(pat,temp))
        pats.insert(temp);
    cout << "\nPat's guest list:\n";
    copy(pats.begin(), pats.end(), out);

    ostream_iterator<string,char> fout (matnpat, "\n");
    set<string> both;
    set_union(mats.begin(), mats.end(), pats.begin(), pats.end(),
```

```
        insert_iterator<set<string> >(both, both.begin()));
    cout << "\nMerged guest list:\n";
    copy(both.begin(), both.end(), out);
    copy(both.begin(), both.end(), fout);
    //cin.get();
    return 0;
}
```

PE 17-6

```
// pe17empf.h -- header file for employee class and children
#include <iostream>
#include <string>
using std::ostream;
using std::istream;
enum classkind{Employee, Manager, Fink, Highfink};

class abstr_emp
{
private:
    std::string fname;    // abstr_emp's first name
    std::string lname;    // abstr_emp's last name
    std::string job;
public:
    abstr_emp();
    abstr_emp(const std::string & fn, const std::string &  ln,
              const std::string &  j);
    virtual void ShowAll() const;    // labels and shows all data
    virtual void SetAll();           // prompts user for values
    virtual ostream & WriteAll(ostream & of) const;  // write to file
    virtual istream & ReadAll(istream & ifs);        // read from file
    friend std::ostream & operator<<(std::ostream & os, const abstr_emp & e);
    // just displays first and last name
    virtual ~abstr_emp() = 0;         // virtual base class
};

class employee : public abstr_emp
{
public:
    employee();
    employee(const std::string & fn, const std::string &  ln,
             const std::string &  j);
    virtual void ShowAll() const;
    virtual void SetAll();
    virtual ostream & WriteAll(ostream & of) const;
    virtual istream & ReadAll(istream & ifs);
};

class manager:  virtual public abstr_emp
{
private:
    int inchargeof;        // number of abstr_emps managed
protected:
    int InChargeOf() const { return inchargeof; } // output
    int & InChargeOf(){ return inchargeof; }      // input
public:
```

```
    manager();
    manager(const std::string & fn, const std::string & ln,
            const std::string & j, int ico = 0);
    manager(const abstr_emp & e, int ico);
    manager(const manager & m);
    virtual void ShowAll() const;
    virtual void SetAll();
    virtual ostream & WriteAll(ostream & of) const;
    virtual istream & ReadAll(istream & ifs);
};

class fink: virtual public abstr_emp
{
private:
    std::string reportsto;        // to whom fink reports
protected:
    const std::string ReportsTo() const { return reportsto; }
    std::string & ReportsTo(){ return reportsto; }
public:
    fink();
    fink(const std::string & fn, const std::string & ln,
         const std::string & j, const std::string & rpo);
    fink(const abstr_emp & e, const std::string & rpo);
    fink(const fink & e);
    virtual void ShowAll() const;
    virtual void SetAll();
    virtual ostream & WriteAll(ostream & of) const;
    virtual istream & ReadAll(istream & ifs);
};

class highfink: public manager, public fink // management fink
{
public:
    highfink();
    highfink(const std::string & fn, const std::string & ln,
             const std::string & j, const std::string & rpo,
             int ico);
    highfink(const abstr_emp & e, const std::string & rpo, int ico);
    highfink(const fink & f, int ico);
    highfink(const manager & m, const std::string & rpo);
    highfink(const highfink & h);
    virtual void ShowAll() const;
    virtual void SetAll();
    virtual ostream & WriteAll(ostream & of) const;
    virtual istream & ReadAll(istream & ifs);
};

// pe17emp.cpp -- employee class and children
#include "pe17emp.h"
using std::string;
using std::cout;
using std::ostream;
using std::istream;
using std::endl;
using std::cin;
using std::getline;
```

```
abstr_emp::abstr_emp()
{
    fname = "";
    lname = "";
    job = "";
}

abstr_emp::abstr_emp(const string & fn, const string & ln,
                     const string & j)
           : fname(fn), lname(ln), job(j)
{
}

abstr_emp::~abstr_emp()
{
}

void abstr_emp::ShowAll() const
{
    cout << "Name: " << fname << " " << lname << endl;
    cout << "Job:  " << job << endl;
}

void abstr_emp::SetAll()
{
    cout << "First name: ";
    getline(cin, fname);
    cout << " Last name: ";
    getline(cin, lname);
    cout << "       Job: ";
    getline(cin, job);
}

ostream & abstr_emp::WriteAll(ostream & of) const
{
    of << fname << endl;
    of << lname << endl;
    of << job << endl;
    return of;
}

istream & abstr_emp::ReadAll(istream & ifs)
{
    getline(ifs, fname);
    getline(ifs, lname);
    getline(ifs, job);
    return ifs;
}

ostream & operator << (ostream & os, const abstr_emp & e)
{
    os << e.fname << " " << e.lname;
    return os;
}

employee::employee() : abstr_emp() {}
```

```
employee::employee(const std::string & fn, const std::string & ln,
                   const std::string & j) : abstr_emp(fn, ln, j)
{
}


void employee::ShowAll() const
{
    cout << "Employee:\n";
    abstr_emp::ShowAll();
}

void employee::SetAll()
{
    cout << "Employee input:\n";
    abstr_emp::SetAll();
}

ostream & employee::WriteAll(ostream & of) const
{
    int kind = Employee;
    of << kind << endl;
    abstr_emp::WriteAll(of);
    return of;
}

istream & employee::ReadAll(istream & ifs)
{
    abstr_emp::ReadAll(ifs);
    return ifs;
}

manager::manager()
{
    inchargeof = 0;
}

manager::manager(const string & fn, const string & ln,
                 const string & j, int ico)
         : abstr_emp(fn, ln, j)
{
    inchargeof = ico;
}

manager::manager(const abstr_emp & e, int ico) : abstr_emp(e)
{
    inchargeof = ico;
}

manager::manager(const manager & m) : abstr_emp(m)
{
    inchargeof = m.inchargeof;
}

void manager::ShowAll() const
{
    cout << "Manager:\n";
```

```
    abstr_emp::ShowAll();
    cout << "In charge of " << inchargeof << endl;
}

void manager::SetAll()
{
    cout << "Manager input:\n";
    abstr_emp::SetAll();
    cout << "Number in charge of: ";
    cin >> inchargeof;
    while (cin.get() != '\n') continue;
}

ostream & manager::WriteAll(ostream & of) const
{
    int kind = Manager;
    of << kind << endl;
    abstr_emp::WriteAll(of);
    of << inchargeof << endl;
    return of;
}

istream & manager::ReadAll(istream & ifs)
{
    abstr_emp::ReadAll(ifs);
    ifs >> inchargeof;
    while (ifs.get() != '\n') continue;
    return ifs;
}

fink::fink()
{
    reportsto = "";
}

fink::fink(const string & fn, const string & ln, const string & j,
           const string    & rpo)
      : abstr_emp(fn, ln, j), reportsto(rpo)
{
}

fink::fink(const abstr_emp & e, const string & rpo)
      : abstr_emp(e), reportsto(rpo)
{
}

fink::fink(const fink & m) : abstr_emp(m), reportsto(m.reportsto)
{
}

void fink::ShowAll() const
{
    cout << "Fink:\n";
    abstr_emp::ShowAll();
    cout << "Reports to: " << reportsto << endl;
}
```

```
void fink::SetAll()
{
    cout << "Fink input:\n";
    abstr_emp::SetAll();
    cout << "Reports to: ";
    getline(cin, reportsto);
}

ostream & fink::WriteAll(ostream & of) const
{
    int kind = Fink;
    of << kind << endl;
    abstr_emp::WriteAll(of);
    of << reportsto << endl;
    return of;
}

istream & fink::ReadAll(istream & ifs)
{
    abstr_emp::ReadAll(ifs);
    getline(ifs, reportsto);
    return ifs;
}

highfink::highfink()
{
}

highfink::highfink(const string & fn, const string & ln,
                   const string & j, const string & rpo, int ico)
          : abstr_emp(fn, ln, j),manager(fn, ln, j, ico),
            fink(fn, ln, j, rpo)
{
}

highfink::highfink(const abstr_emp & e, const string & rpo, int ico)
          : abstr_emp(e), manager(e, ico), fink(e, rpo)
{
}

highfink::highfink(const fink & f, int ico)
          : abstr_emp(f), fink(f), manager(f, ico)
{
}

highfink::highfink(const manager & m, const string & rpo)
          : abstr_emp(m), manager(m), fink(m, rpo)
{
}

highfink::highfink(const highfink & h)
          : abstr_emp(h), manager(h), fink(h)
{
}

void highfink::ShowAll() const
{
```

```
    cout << "High Fink:\n";
    abstr_emp::ShowAll();
    cout << "In charge of " << InChargeOf() << endl;
    cout << "Reports to: " << ReportsTo() << endl;
}

void highfink::SetAll()
{
    cout << "High Fink input:\n";
    abstr_emp::SetAll();
    cout << "Reports to: ";
    getline(cin, ReportsTo());
    cout << "Number in charge of: ";
    cin >> InChargeOf();
}

ostream & highfink::WriteAll(ostream & of) const
{
    int kind = Highfink;
    of << kind << endl;
    abstr_emp::WriteAll(of);
    of << InChargeOf() << endl;
    of << ReportsTo() << endl;
    return of;
}

istream & highfink::ReadAll(istream & ifs)
{
    abstr_emp::ReadAll(ifs);
    ifs >> InChargeOf() ;
    while (ifs.get() != '\n') continue;
    getline(ifs, ReportsTo());
    return ifs;
}


// pe17-6.cpp -- use employee classes
// link with pe17emp.cpp

#include <iostream>
#include <fstream>
#include <cstdlib>
#include "pe17emp.h"

const char * myfile = "emp.dat";
const int MAX = 10;
char menu();

int main()
{
    using namespace std;
    abstr_emp * pc[MAX];
    int index = 0;
    int choice;
    int classtype;
    char ch;
```

```
int i;

ifstream fin;
fin.open(myfile);
if (fin.good())
{
    cout << "Here are the current contents of the "
        << myfile << " file:\n";
    while((fin >> classtype).get(ch))
    {
        switch(classtype)
        {
            case Employee    :
                pc[index] = new employee;
                break;
            case Manager    :
                pc[index] = new manager;
                break;
            case Fink    :
                pc[index] = new fink;
                break;
            case Highfink    :
                pc[index] = new highfink;
                break;
            default            :
                cerr << "Switch problem\n";
                break;
        }
        pc[index++]->ReadAll(fin);
        if (!fin.good())
            break;
    }
    for (i = 0; i < index; i++)
        pc[i]->ShowAll();
}

while (index < MAX)
{
    choice = menu();
    if (choice == 'q')
        break;
    switch(choice)
    {
    case 'e'    :    pc[index] = new employee;
                    break;
    case 'm'    :    pc[index] = new manager;
                    break;
    case 'f'    :    pc[index] = new fink;
                    break;
    case 'h'    :    pc[index] = new highfink;
                    break;
    }
    pc[index++]->SetAll();
}
cout << "Finished with data entry.\n";

if (index == MAX)
```

```
        cout << "File is full.\n";
    fin.close();
    cout << "Recapitulating:\n";
    for (i = 0; i < index; i++)
        pc[i]->ShowAll();
    ofstream fout(myfile, ios::out);
    if(!fout)
    {
        cerr << "Can't open file for writing\n";
        exit(2);
    }
    for (i = 0; i < index; i++)
    {
        pc[i]->WriteAll(fout);
    }
    for (i = 0; i < index; i++)
    {
         delete pc[i];;
    }

    cout << "Bye!\n";
    //cin.get();
    return 0;
}

char menu()
{
    using namespace std;
    cout << "Please make a choice as to what to add:\n";
    cout << "e) employee        m) manager\n";
    cout << "f) fink            h) highfink\n";
    cout << "q) quit\n";
    char ch;
    while (cin >> ch && ch != 'e' && ch != 'm' && ch != 'f'
            && ch != 'h' && ch != 'q')
        cout << "Try again!\n";
    while (cin.get() != '\n') continue;
    return ch;
}
```

PE 17-7

```
// pe17-7.cpp
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <algorithm>
#include <cstdlib>

void ShowStr(const std::string & s);
void GetStrs(std::istream & is, std::vector<std::string> & vs);

class Store
{
```

```
public:
    std::ostream & os;
    Store (std::ostream & o) : os(o) {}
    void operator()(const std::string &s);
};

int main()
{
    using namespace std;
    vector<string> vostr;
    string temp;

// acquire strings
    cout << "Enter strings (empty line to quit):\n";
    while (getline(cin,temp) && temp[0] != '\0')
        vostr.push_back(temp);
    cout << "Here is your input.\n";
    for_each(vostr.begin(), vostr.end(), ShowStr);

// store in a file
    ofstream fout("strings.dat", ios_base::out | ios_base::binary);
    for_each(vostr.begin(), vostr.end(), Store(fout));
    fout.close();

// recover file contents
    vector<string> vistr;
    ifstream fin("strings.dat", ios_base::in | ios_base::binary);
    if (!fin.is_open())
    {
        cerr << "Could not open file for input.\n";
        exit(EXIT_FAILURE);
    }
    GetStrs(fin, vistr);
    cout << "\nHere are the strings read from the file:\n";
    for_each(vistr.begin(), vistr.end(), ShowStr);
    //cin.get();
    return 0;
}

void ShowStr(const std::string & s)
{
    std::cout << s << std::endl;
}

void Store::operator()(const std::string &s)
{
    std::size_t len = s.size();
    os.write((char *)&len, sizeof(std::size_t));
    os.write(s.data(), len);
}

void GetStrs(std::istream & is, std::vector<std::string> & vs)
{
    std::string temp;
    size_t len;

    while (is.read((char *) &len, sizeof(size_t)) && len > 0)
```

```
    {
        char ch;
        temp = "";
        for (int j = 0; j < len; j++)
        {
            if (is.read(&ch, 1))
            {
                temp += ch;
            }
            else
                break;
        }
        if (is)
            vs.push_back(temp);
    }
}
```

Chapter 18

PE 18-1

```
// MS Visual C++ 2010 doesn't support list initialization
// g++ 4.5 does support list initialization
// pe18-1.cpp
#include <iostream>
#include <string>
#include <initializer_list>

template <typename T>
  T average_list(const std::initializer_list<T> & il);

int main()
{
    using namespace std;

    auto q = average_list( {15.4, 10.7, 9.0});
    cout << q << endl;
    cout << average_list( {20, 30, 19, 17, 45, 38} ) << endl;

    return 0;
}

template <typename T>
  T average_list(const std::initializer_list<T> & il)
{
    using namespace std;
    T tot = 0;
    int sz = il.size();
    for (auto p = il.begin(); p !=il.end(); p++)
        tot += *p;
    if (sz > 0)
        return tot/sz;
    else
        return 0;
}
```

PE 18-2

```
// g++ prior to 4.6 does not support nullptr
//pe18-2.cpp
#include <iostream>
#include <string>
// followed def for g++ 4.5.0
//#define nullptr 0
class Cpmv
{
public:
    struct Info
    {
        std::string qcode;
        std::string zcode;
    };
private:
    Info *pi;
public:
    Cpmv();
    Cpmv(std::string q, std::string z);
    Cpmv(const Cpmv & cp);
    Cpmv(Cpmv && mv);
    ~Cpmv();
    Cpmv & operator=(const Cpmv & cp);
    Cpmv & operator=(Cpmv && mv);
    Cpmv operator+(const Cpmv & obj) const;
    void Display() const;
};

int main()
{
    {
    Cpmv m1("qx56", "z370");
    m1.Display();
    Cpmv m2 = Cpmv("qogahog", "zatabat");
    m2.Display();
    Cpmv m3(m2);
    m3.Display();
    m1 = m2;
    m1.Display();
    m1 = Cpmv("qu22uq", "za88az");
    m1.Display();
    m1 = m2 + m1;
    m1.Display();
    Cpmv m4(m2 + m3);
    m4.Display();
    }  // all local variables expire here, destructors called
    std::cin.get(); // to keep output window open
    return 0;
}

Cpmv::Cpmv()
{
    pi = new Info;
```

```
    pi->qcode = "Q";
    pi->zcode = "Z";
    std::cout << "Cpmv(): Qcode = Q, Zcode = Z\n";
}

Cpmv::Cpmv(std::string q, std::string z)
{
    pi = new Info;
    pi->qcode =q;
    pi->zcode = z;
    std::cout << "Cpmv(string,string): Qcode = " << pi->qcode
              <<  ", Zcode = " << pi->zcode <<'\n';
}


Cpmv::Cpmv(const Cpmv & cp)
{
    pi = new Info;
    pi->qcode = cp.pi->qcode;
    pi->zcode = cp.pi->zcode;
    std::cout << "Cpmv(&): Qcode = " << pi->qcode
              <<  ", Zcode = " << pi->zcode << '\n';
}

Cpmv::Cpmv(Cpmv && mv)
{
    pi = mv.pi;
    mv.pi = nullptr;
    std::cout << "Cpmv(&&): Qcode = " << pi->qcode
              <<  ", Zcode = " << pi->zcode << '\n';
}

Cpmv & Cpmv::operator=(const Cpmv & cp)
{
    if (this == &cp)
        return *this;
    delete  pi;
    pi = new Info;
    pi->qcode = cp.pi->qcode;
    pi->zcode = cp.pi->zcode;
    std::cout << "=(&): Qcode = " << pi->qcode
              <<  ", Zcode = " << pi->zcode << '\n';
    return *this;
}

Cpmv & Cpmv::operator=(Cpmv && mv)
{
    delete pi;
    pi = mv.pi;
    mv.pi = nullptr;
    std::cout << "=(&&): Qcode = " << pi->qcode
              <<  ", Zcode = " << pi->zcode << '\n';
    return *this;
}

Cpmv Cpmv::operator+(const Cpmv & obj) const
{
```

```
    std::cout << "+(): ";
    Cpmv temp;
    temp.pi = new Info;
    temp.pi->qcode = pi->qcode + obj.pi->qcode;
    temp.pi->zcode = pi->zcode + obj.pi->zcode;
    std::cout << "Qcode = " << temp.pi->qcode
              <<  ", Zcode = " << temp.pi->zcode << '\n';
    return temp;
}

Cpmv::~Cpmv()
{
    std::cout << "~(): ";
    if (pi)
        std::cout << "Qcode = " << pi->qcode
                  <<  ", Zcode = " << pi->zcode << '\n';
    else
        std::cout << "no data\n";
}

void Cpmv:: Display() const
{
    std::cout << "Qcode: " << pi->qcode
               <<  ", Zcode: " << pi->zcode << '\n';
}
```

PE 18-3

```
// MS VC++ 2010 does not support variadic templates
// pe18-3.cpp
#include <iostream>
#include <string>

// definition for 0 parameters
long double sum_values() { return 0;}

// definition for 1 parameter
template<typename T>
long double sum_values(const T& value)
{
    return value;
}

// definition for 2 or more parameters
template<typename T, typename... Args>
long double sum_values(const T& value, const Args&... args)
{

    return value + sum_values(args...);
}

int main()
{
    using namespace std;
    int n = 14;
```

```
    double x = 2.71828;
    char ch = 'A';
    cout << sum_values(n, x, ch, 7+6) << endl;
    cout << sum_values() << endl;
    cout << sum_values(88) << endl;
    return 0;
}
```

PE 18-4

```
// MS VC++ 2010 and g++ 4.5 both support lambdas
// pe18-4.cpp
#include <iostream>
#include <list>
#include <iterator>
#include <algorithm>

int main()
{
    using std::list;
    using std::cout;
    using std::endl;
    using std::for_each;

    int vals[10] = {50, 100, 90, 180, 60, 210, 415, 88, 188, 201};
    list<int> yadayada(vals, vals + 10); // range constructor
    list<int> etcetera(vals, vals + 10);

    cout << "Original lists:\n";
    auto outint = [](int n){std::cout << n <<' ';};
    std::for_each(yadayada.begin(), yadayada.end(), outint);
    cout << endl;
    for_each(etcetera.begin(), etcetera.end(), outint);
    cout << endl;
    yadayada.remove_if([](const int & v) {return v > 100;});
    etcetera.remove_if([](const int & v) {return v > 200;});
    cout <<"Trimmed lists:\n";
    for_each(yadayada.begin(), yadayada.end(), outint);
    cout << endl;
    for_each(etcetera.begin(), etcetera.end(), outint);
    cout << endl;
    //std::cin.get();
    return 0;
}
```