## Chapter 2

```
// pe2-2.cpp

#include <iostream>

int main(void)
{
    using namespace std;
    cout << "Enter a distance in furlongs: ";
    double furlongs;
    cin >> furlongs;
    double feet;
    feet = 220 * furlongs;
    cout << furlongs << " furlongs = "
         << feet << " feet\n";

    return 0;
}


// pe2-3.cpp

#include <iostream>
using namespace std;

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

   return 0;
}

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

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


// pe2-4.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";

   return 0;
}

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

## Chapter 3

```
// pe3-1.cpp

#include <iostream>

const int Inch_Per_Foot = 12;

int main(void)
{
    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";
    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";
    return 0;
}
```

```
// pe3-5.cpp

#include <iostream>

int main(void)
{
    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";
    return 0;
}
```

```
// pe3-6.cpp

#include <iostream>

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

int main ( void )
{
    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";

    return 0;
}
```

## Chapter 4

```
// 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";
    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;
    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;

    return 0;
}
```

```
// pe4-7.ccp

#include <iostream>

const int Slen = 70;

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

int main(void)
{
```

```
    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";
    return 0;
}
```

## Chapter 5

```
// pe5-2.cpp

#include <iostream>

int main(void)
{
    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";
    return 0;
}


// pe5-4.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"};
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;
    return 0;
}



// pe5-6.cpp

#include <iostream>

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

int main(void)
{
    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;
    return 0;
}


// pe5-7.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";
    return 0;
}


// pe5-9.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;
    }

    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);
    }

    return 0;
}
```

```
// pe6-3.cpp

#include <iostream>

int main(void)
{
    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";
    }
    return 0;
}


// 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 your annual income in tvarps: ";
    cin >> income;

    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 << "You owe Neutronia " <<  tax << " tvarps in taxes.\n";
```

```
    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";

    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";
        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
    return 0;
}
```

## Chapter 7

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

#include <iostream>

double h_mean(double x, double y);

int main(void)
{
    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";
    return 0;
}

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


// 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(void)
{
    box carton = {"Bingo Boxer", 2, 3, 5}; // no volume provided
    setbox(&carton);
    showbox(carton);
    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";
    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;
}
// pe7-6.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);

    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;
    }
}

//pe7-9.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(void)
{
    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";
    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(void)
{
    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";
    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;
}
```

```
// pe8-4.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(void)
{
    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!");
    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;

    return 0;
}
```

# 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();
        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(void)
{
    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]);

    return 0;
}
```

## 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

    return 0;
}
```

## 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();
}
```

## PE10-4

```
// pe10-4.h
#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
#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();

    return 0;
}

// pe10-4b.cpp
#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

#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(void)
{
    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";
    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 10-8

```
// 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(void)
{
    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";
    return 0;
}

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

## Chapter 11

### PE 112

```
// pe11-2.h -- Vector class with <<, mode state
// modified implementation
#ifndef MODVECTOR_H_
#define MODVECTOR_H_
#include <iostream>
namespace VECTOR
{
    using std::ostream;
    class Vector
    {
```

```
    private:
        double x;          // horizontal value
        double y;          // vertical value
        char mode;         // 'r' = rectangular, 'p' = 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, char form = 'r');
        void set(double n1, double n2, char form = 'r');
        ~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 'p'
        void rect_mode();                     // set mode to 'r'
    // 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 ostream & operator<<(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::atan2;
using std::cout;

namespace VECTOR
{

    const double Rad_to_deg = 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 = 'r';
}

// 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, char form)
{
    mode = form;
    if (form == 'r')
     {
          x = n1;
          y = n2;
    }
    else if (form == 'p')
    {
          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 = 'r';
    }
}

// set vector from rectangular coordinates if form is r (the
// default) or else from polar coordinates if form is p
void Vector:: set(double n1, double n2, char form)
{
    mode = form;
    if (form == 'r')
     {
          x = n1;
          y = n2;
    }
    else if (form == 'p')
    {
          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 = 'r';
    }
}

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 = 'p';
}

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

// 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
ostream & operator<<(ostream & os, const Vector & v)
{
    if (v.mode == 'r')
         os << "(x,y) = (" << v.x << ", " << v.y << ")";
    else if (v.mode == 'p')
    {
         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 vect.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.set(dstep, direction, 'p');
            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.set(0.0, 0.0);
        cout << "Enter target distance (q to quit): ";
    }
    cout << "Bye!\n";

    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.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

// pe11-5.cpp
#include <iostream>
#include "pe11ston.h"
// link with pe11ston.cpp
int main(void)
{
     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;

     return 0;
}
```

**PE 11-7**

```
// 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";
    return 0;
}

// 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;
}
```

## Chapter 12

### 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";
    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 <iostream>
#include <cctype>
//#include "string2.h"
#include "pe12strg.h"  // alternative name
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 12-4

```
// 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(void)
{
    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";
    return 0;
}
```

**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";

   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>
using namespace std;
#include "classic.h"     // which will contain #include cd.h
void Bravo(const Cd & disk);
int main()
{
    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();

    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();
  return 0;
}
```

## Chapter 14

### PE 14-1

```
// pairs.h -- 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() {}
};

#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";
    return 0;
}
```

## PE 14 2

```
// pairs.h -- 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() {}
};

#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";
    return 0;
}
```

**PE 14-4**

```
// pe14-pg.h

#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() { 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(); }
};

#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\n";
}


// pe14-4.cpp
```

```
#include <iostream>
#include <cstring>
#include "pe14-4pg.h"
const int SIZE = 5;
int main(void)
{
    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";
    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, VCR};

    Tv(State s = Off, int mc = 100) : 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)? VCR : 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 VCR
    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" : "VCR\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(void)
{
    Tv s20;
    std::cout << "Initial settings for 20\" TV:\n";
    s20.settings();
    s20.onoff();
    s20.chanup();
    std::cout << "\nAdjusted settings for 20\" TV:\n";
    s20.settings();

    Remote grey;

    grey.set_chan(s20, 10);
    grey.volup(s20);
    grey.volup(s20);
    std::cout << "\n20\" settings after using remote\n";
    s20.settings();

    Tv s27(Tv::On);
    s27.set_mode();
    grey.set_chan(s27,28);
    std::cout << "\n27\" settings:\n";
    s27.settings();
    grey.show_style();    // check mode
    s27.rmode(grey);         // change mode
    grey.show_style();    // recheck mode
    s27.onoff();           // turn set off
    s27.rmode(grey);        // try changing mode again
    grey.show_style();     // check result

    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) throw(hmeanexcp);
double gmean(double a, double b) throw(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";
    return 0;
}

double hmean(double a, double b) throw(hmeanexcp)
{
    if (a == -b)
        throw hmeanexcp();
    return 2.0 * a * b / (a + b);
}

double gmean(double a, double b) throw(gmeanexcp)
{
    if (a < 0 || b < 0)
        throw gmeanexcp();
    return std::sqrt(a * b);
}
```

## 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";

    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 16-4

```
// 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);
    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 16-8

```
// 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);

    return 0;
}
```

## Chapter 17

### PE 17-1

```
// pe17-1.cpp
#include <iostream>

int main(void)
{
    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;
    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 17-5**

```
// 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);

    return 0;
}
    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);

    return 0;

}
```

## 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);

    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);
    }
}
```