Transparency Masters

Section 1.1: A First C++ Program

```
// This program outputs the message
//
//    C++: one small step for the program,
//    one giant leap for the programmer
//
// to the video display
#include <iostream>
using namespace std;

int main() {
   cout << "C++: one small step for the program," << endl
        << "one giant leap for the programmer" << endl;
   return 0;
}
```

Section 1.4: Conversion of Lengths

```
//   This program reads a length in feet from
//   the keyboard, converts the length to inches,
//   and outputs the converted length to the
//   video display. It is assumed that the length
//   is a nonnegative integer.
#include <iostream>
using namespace std;
int main() {
   int foot, inch;

   cout << "Enter length: ";
   cin >> foot;

   inch = 12 * foot;
   cout << foot << " ft" << endl
        << inch << " in" << endl;
   return 0;
}
```

Section 1.8: A Math Puzzle

```
// This program prompts the user to enter
// the values of pi (= 3.1415...) and
// e (= 2.71828...). It then computes
// e to the pi and pi to the e so that
// the user can determine which is larger.
#include <iostream>
#include <cmath>
using namespace std;
main() {
   double pi, e;
   cout << "Enter pi: ";
   cin >> pi;
   cout << "Enter e: ";
   cin >> e;
   cout << "e to the pi = " << pow( e, pi ) << endl;
   cout << "pi to the e = " << pow( pi, e ) << endl;
   return 0;
}
```

Section 2.2: Computing Taxes

```
// This program reads an income. Income under
// 6000 greenbacks is taxed at 30 percent, and
// income greater than or equal to 6000 greenbacks
// is taxed at 60 percent. After reading the
// income, the program prints the income and tax.
#include <iostream>
using namespace std;
const int cutoff = 6000;
const float rate1 = 0.3;
const float rate2 = 0.6;
int main() {
   int income, tax;
   cout << "Income? ";
   cin >> income;
   if ( income < cutoff )
      tax = rate1 * income;
   else
      tax = rate2 * income;
   cout << "Income = " << income
        << " greenbacks" << endl
        << "Tax = " << tax
        << " greenbacks" << endl;
   return 0;
}
```

Section 2.4: Computing Taxes Revisited

```
// This program repeatedly reads an income. A
// negative number signals the end of input. Income
// under 6000 greenbacks is taxed at 30 percent, and
// income greater than or equal to 6000 greenbacks
// is taxed at 60 percent. After reading each
// income, the program prints the income and tax.
#include <iostream>
using namespace std;
const int cutoff = 6000;
const float rate1 = 0.3;
const float rate2 = 0.6;
int main() {
   int income, tax;
   cout << "Income? ";
   cin >> income;
   while ( income >= 0 ) {
      if ( income < cutoff )
         tax = rate1 * income;
      else
         tax = rate2 * income;
      cout << "Income = " << income
           << " greenbacks" << endl
           << "Tax = " << tax
           << " greenbacks" << endl
           << "Income? ";
      cin >> income;
   }
   return 0;
}
```

Example 2.5.4: Reading From and Writing to Files

```
// This program repeatedly reads an income from
// the file income.in until end-of-file.
// Income under 6000 greenbacks is taxed at 30
// percent, and income greater than or equal to
// 6000 greenbacks is taxed at 60 percent.
// After reading each income, the program
// outputs the income and tax to the file tax.out.
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;
const int cutoff = 6000;
const float rate1 = 0.3;
const float rate2 = 0.6;
int main() {
   ifstream infile;
   ofstream outfile;
   infile.open( "income.in" );
   outfile.open( "tax.out" );
   if ( !infile ) {
      cout << "Unable to open income.in" << endl;
      exit( 0 );
   }
   if ( !outfile ) {
      cout << "Unable to open tax.out" << endl;
      exit( 0 );
   }
   int income, tax;
   while ( infile >> income ) {
      if ( income < cutoff )
         tax = rate1 * income;
      else
         tax = rate2 * income;
```

```
      outfile << "Income = " << income
              << " greenbacks" << endl
              << "Tax = " << tax
              << " greenbacks" << endl;
   }
   infile.close();
   outfile.close();
   return 0;
}
```

Section 2.9: Statistical Measures

```
#include <iostream>
#include <cmath>
#include <cfloat>
using namespace std;
/** The program reads floating-point numbers from the
    standard input and prints a statistical summary that
    includes the largest value input, the smallest value
    input, the sum of all values input, the mean,
    population variance, and standard deviation. **/
int main() {
   float next,   // next value from standard input
         min,    // minimum value
         max,    // maximum value
         mean,   // average of all values
         sum,    // sum of all values
         sumSqr, // sum of squares of all values,
         var;    // variance of all values
   unsigned count = 0;
   min = FLT_MAX;
   max = FLT_MIN;
   sum = sumSqr = 0.0F;
   // loop until standard input is empty
   while ( cin >> next ) {
      if ( next > max ) // new maximum?
        max = next;
      if ( next < min ) // new minimum?
        min = next;
      sum += next;           // running sum
      sumSqr += next * next; // sum of squares
      count++;
   }
   mean = sum / count;
   var = sumSqr / count - mean * mean;
```

```
   // output results
   cout << "Statistical summary:" << endl;
   cout << '\t' << count << " numbers read." << endl;
   cout << '\t' << "  Maximum:  " << max << endl;
   cout << '\t' << "  Minimum:  " << min << endl;
   cout << '\t' << "  Sum:      " << sum << endl;
   cout << '\t' << "  Mean:     " << mean << endl;
   cout << '\t' << "  Variance: " << var << endl;
   cout << '\t' << "  StdDev:   " << sqrt( var ) << endl;
   return 0;
}
```

Section 2.11: Generating Prime Numbers

```
#include <iostream>
using namespace std;
/*** Print positive primes <= a user-supplied value ***/
/*** Selected lines are numbered for later reference. ***/
int main() {
        unsigned posPrime, // possible prime
                 posDiv,   // possible divisor
                 n;        // user supplied limit
        // get nonzero limit
        do {
          cout << "Limit? (> 0, please) ";
          cin >> n;
        } while ( n <= 0 );

        cout << "Primes <= " << n << endl;
/* 1 */ for ( posPrime = 2; posPrime <= n; posPrime++ ) {
           // look for a divisor of a possible prime
/* 2 */    for ( posDiv = 2; posDiv < posPrime; posDiv++ )
             // if the divisor divides the candidate with
             // zero remainder, the candidate is not prime
/* 3 */      if ( 0 == posPrime % posDiv )
               break; // stop looking -- not a prime
           // no divisors, hence we have a prime
/* 4 */    if ( posDiv == posPrime )
             cout << posPrime << endl;
        }
        return 0;
}
```

Example 3.1.1: Assigning a Grade

```
#include <iostream>
using namespace std;
const float cutoff = 7.0;
char grade( int exam1, int exam2, float exam_weight );

int main() {
   int mid_term, final;
   float weight;
   char ans, letter_grade;

   do {
      cout << "\n\nCompute another grade (Y)es, (N)o? ";
      cin >> ans;
      if ( ans == 'Y' ) {
         cout << "Enter mid term, final, weight: ";
         cin >> mid_term >> final >> weight;
         letter_grade = grade( mid_term, final, weight );
         cout << "Grade is: " << letter_grade << endl;
      }
   } while ( ans == 'Y');
   return 0;
}

char grade( int exam1, int exam2, float exam1_weight ) {
   float average;
   average = exam1_weight * exam1
            + ( 1.0 - exam1_weight ) * exam2;
   if ( average > cutoff )
      return 'P';
   else
      return 'F';
}
```

Section 3.4: The Monty Hall Problem

```
#include <iostream>
#include <cstdlib>
using namespace std;
int get_strat();
int get_no_games();
char get_each_game_flag();
int play_games( int strat,
                int no_games,
                char each_game_flag );

int main() {
   int strat, no_games, won;
   char each_game_flag;
   strat = get_strat();
   no_games = get_no_games();
   each_game_flag = get_each_game_flag();
   won = play_games( strat, no_games, each_game_flag );
   cout << "Games played = " << no_games << endl;
   cout << "Games won = " << won << endl;
   cout << "Percent won = " <<
           100.0 * won / no_games << endl;
   return 0;
}

int get_strat() {
   int strat;
   do {
      cout << "1--never switch" << endl
           << "2--always switch" << endl
           << "3--randomly switch" << endl
           << "Your choice? ";
      cin >> strat;
   } while ( strat < 1 && strat > 3 );
```

```
   return strat;
}

int get_no_games() {
   int no_games;
   cout << "Play how many games? ";
   cin >> no_games;
   return no_games;
}

char get_each_game_flag() {
   char each_game_flag;
   cout << "Display results of each game [Y]es, [N]o? ";
   cin >> each_game_flag;
   return each_game_flag;
}

int play_games( int strat,
                int no_games,
                char each_game_flag ) {
   int i, car, host, sw_flag, person, win;
   for ( win = 0, i = 0; i < no_games; i++ ) {
      if ( each_game_flag == 'Y' )
         cout << "\n\nBeginning a new game\n";
      // pick person's door
      person = rand() % 3;
      if ( each_game_flag == 'Y')
         cout << "Person picks door " << person << endl;
      // place car behind a door
      car = rand() % 3;
      if ( each_game_flag == 'Y' )
         cout << "Car is behind door " << car << endl;
      // if person has picked the car, the host
      // has a random choice of doors to open
```

```
if ( person == car )
   host = ( 1 + person + rand() % 2 ) % 3;
// otherwise the host's door is the door
// other than the person's that does not
// contain the car
else {
   host = ( person + 1 ) % 3;
   if ( host == car )
      host = ( host + 1 ) % 3;
}
if ( each_game_flag == 'Y' )
   cout << "Host opens door " << host << endl;
switch ( strat ) {
case 1: // never switch
   sw_flag = 0;
   break;
case 2: // always switch
   sw_flag = 1;
   break;
case 3: // randomly switch
   sw_flag = rand() % 2;
   break;
}
if ( sw_flag ) {
   // the person switches to the door
   // the host did not open
   person = ( person + 1 ) % 3;
   if ( person == host )
      person = ( person + 1 ) % 3;
   if ( each_game_flag == 'Y' )
      cout << "Switch to door " << person<< endl;
}
if ( car == person ) {
   if ( each_game_flag == 'Y' )
      cout << "Win" << endl;
```

```
         win++;
      }
      else if ( each_game_flag == 'Y' )
         cout << "Loss" << endl;
   }
   return win;
}
```

Section 3.6: Printing a Calendar

```
// This program prints a calendar for a year.
// The user enters the year, after which
// the year and calendar are printed to the
// file calendar.dat.

#include <iostream>
#include <iomanip>
#include <fstream>
#include <cstdlib>
using namespace std;
void open_or_die( ofstream& out );
void print_calendar( int year, ofstream& out );
int get_day_code( int year );
bool get_leap_code( int year );

int main() {
   ofstream outfile;
   open_or_die( outfile );
   int year;
   cout << "Enter the year: ";
   cin >> year;
   print_calendar( year, outfile );
   return 0;
}

void open_or_die( ofstream& out ) {
   out.open( "calendar.dat" );
   if ( !out ) {
      cerr << "Unable to open calendar.dat" << endl;
      exit( EXIT_FAILURE );
   }
}
```

```
// print_calendar prints the calendar specified
// by parameter year to the output file specified
// by parameter out.
void print_calendar( int year, ofstream& out ) {
   int day_code, month, day, days_in_month;
   bool leap_code;
   out << year;

   // get_day_code returns the day on which January
   // 1 of year begins. The return value is coded
   // as: 0 is Sunday, 1 is Monday, and so on.
   day_code = get_day_code( year );

   // get_leap_code returns true, if year is a
   // leap year; and false, otherwise.
   leap_code = get_leap_code( year );

   // month is coded as 1 for Jan, 2 for Feb,
   // and so on.
   for ( month = 1; month <= 12; month++ ) {
      // print name and set days_in_month
      switch ( month ) {
      case 1:
         out << "\n\nJanuary";
         days_in_month = 31;
         break;
      case 2:
         out << "\n\nFebruary";
         if ( leap_code )
            days_in_month = 29;
         else
            days_in_month = 28;
         break;
      case 3:
         out << "\n\nMarch";
```

```
        days_in_month = 31;
        break;
    case 4:
        out << "\n\nApril";
        days_in_month = 30;
        break;
    case 5:
        out << "\n\nMay";
        days_in_month = 31;
        break;
    case 6:
        out << "\n\nJune";
        days_in_month = 30;
        break;
    case 7:
        out << "\n\nJuly";
        days_in_month = 31;
        break;
    case 8:
        out << "\n\nAugust";
        days_in_month = 31;
        break;
    case 9:
        out << "\n\nSeptember";
        days_in_month = 30;
        break;
    case 10:
        out << "\n\nOctober";
        days_in_month = 31;
        break;
    case 11:
        out << "\n\nNovember";
        days_in_month = 30;
        break;
    case 12:
```

```
         out << "\n\nDecember";
         days_in_month = 31;
         break;
      }

      out << "\n\nSun  Mon  Tue  Wed  Thu  Fri  Sat\n";

      // advance to correct position for first day
      for ( day = 1; day <= day_code * 5; day++ )
         out << " ";

      // print the dates for one month
      for ( day = 1; day <= days_in_month; day++ ) {
         // field width is 3 because day names
         // are 3 characters.
         out << setw( 3 ) << day;
         if ( ( day + day_code ) % 7 > 0 ) // before Sat?
            // move 2 spaces to next day in same week.
            // (2 spaces separate the day names.)
            out << "  ";
         else  // skip to next line to start with Sun
            out << endl;
      }

      // set day_code for next month to begin
      day_code = ( day_code + days_in_month ) % 7;
   }
}

// get_day_code returns the day on which January
// 1 of the specified year begins. The return
// value is coded as: 0 is Sunday, 1 is Monday,
// and so on.
int get_day_code( int year ) {
   return ( year + ( year - 1 ) / 4 - ( year - 1 ) / 100
```

```
            + ( year - 1 ) / 400 ) % 7;
}

// get_leap_code returns true, if the specified
// year is a leap year; and false, otherwise.
bool get_leap_code( int year ) {
   return year % 4 == 0 && year % 100 != 0
          || year % 400 == 0;
}
```

Example 3.8.1: The Factorial Function

```
#include <iostream>
using namespace std;
int fact( int num );

int main() {
   int num;
   cout << "Please enter a number: ";
   cin >> num;
   if ( num < 0 )
      cout << "ERROR--number must be >= 0." << endl;
   else
      cout << num << "! = " << fact( num ) << endl;
   return 0;
}

int fact( int num ) {
   if ( num <= 1 )  // base cases
      return 1;
   // recursive call
   return num * fact( num - 1 );
}
```

Example 3.8.2: The Tower of Hanoi

```
#include <iostream>
using namespace std;
void hanoi( char peg1, char peg2, char peg3, int how_many );

int main() {
   char peg1 = 'A',   // origin
        peg2 = 'B',   // destination
        peg3 = 'C';   // spare
   int how_many;  // number of disks initially on the origin
   // Prompt for and warn about
   // the number of disks to be moved.
   cout << "\nHow many disks initially on peg A?"
        << "\nIf more than 7, the solution may seem to "
        << "take forever! ";
   cin >> how_many;
   cout << endl << endl;
   // Anything to move?
   if ( how_many < 1 )
      cout << "There's nothing to move!" << endl;
   // Otherwise solve with:
   //   -- peg1 as the origin
   //   -- peg2 as the destination
   //   -- peg3 as the spare
   else
      hanoi( peg1, peg2, peg3, how_many );
   return 0;
}
// p1 -- origin
// p2 -- destination
// p3 -- spare
// how_many -- number of disks to move
void hanoi( char p1, char p2, char p3, int how_many ) {
   // If there is only 1 disk on p1, then move it to p2
```

```
   // and quit as the problem is solved.
   if ( how_many == 1 ) {
      cout << "Move top disk from peg " << p1
           << " to peg " << p2 << endl;
      return;
   }
   // Otherwise:
   //    (1) Move how_many - 1 disks from p1 to p3:
   //        p1 is the origin, p3 is the destination,
   //        and p2 is the spare.
   //
   //    (2) Move the top disk from p1 to p2.
   //
   //    (3) Move how_many - 1 disks from p3 to p2:
   //        p3 is the origin, p2 is the destination,
   //        and p1 is the spare.
   hanoi( p1, p3, p2, how_many - 1 );
   cout << "Move top disk from peg " << p1
        << " to peg " << p2 << endl;
   hanoi( p3, p2, p1, how_many - 1 );
}
```

Section 4.4: Totaling Votes

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

int main() {
   ifstream infile;      // data file
   const unsigned n = 4; // number of candidate birds
   unsigned counts[ n ] = { 0 }; // ballot totals per bird
   unsigned precinct, bird, votes;
   // Read counts from input file and update totals
   infile.open( "votes.dat" );
   while ( infile >> precinct >> bird >> votes )
     counts[ bird ] += votes; // update vote count
   infile.close();
   // Print vote totals to the standard output
   cout << "Vote totals are:" << endl;
   for ( unsigned i = 0; i < n; i++ ) {
     switch ( i ) {
     case 0: // pigeon
       cout << "Pigeon:  "; break;
     case 1: // mourning dove
       cout << "Dove:    "; break;
     case 2: // falcon
       cout << "Falcon:  "; break;
     case 3: // robin
       cout << "Robin:   "; break;
     }
     cout << counts[ i ] << endl;
   }
   return 0;
}
```

Section 4.8: Merging Files

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;
const unsigned MaxRecLen = 80;
const unsigned MaxFileName = 200;
int main() {
   ifstream if1, if2;
   ofstream of;
   char infile1[ MaxFileName + 1 ],
        infile2[ MaxFileName + 1 ],
        outfile[ MaxFileName + 1 ];
   char rec1[ MaxRecLen ],
        rec2[ MaxRecLen ];

   // Prompt for and read file names.
   cout << "Input file 1: ";
   cin >> infile1;
   cout << "Input file 2: ";
   cin >> infile2;
   cout << "Output file: ";
   cin >> outfile;

   // Open the files.
   if1.open( infile1 );
   if2.open( infile2 );
   of.open( outfile );

   // Merge the files until both are empty. An empty record
   // is one whose length as a string is zero. We assume
   // that records are terminated by a newline character
   // and that the input files are sorted.
   //*** loop prolog
```

```
if1.getline( rec1, MaxRecLen );
if2.getline( rec2, MaxRecLen );
//*** While both input files still contain records,
//    compare the records and output the one
//    that lexicographically precedes the other.
while ( strlen( rec1 ) && strlen( rec2 ) ) {
  // if rec1 < rec2, output rec1 and get another
  if ( strcmp( rec1, rec2 ) < 0 ) {
    of << rec1 << endl;
    if1.getline( rec1, MaxRecLen );
  }
  // rec2 <= rec1, so output rec2 and get another
  else {
    of << rec2 << endl;
    if2.getline( rec2, MaxRecLen );
  }
}

//*** If the first file still contains records but the
//    the second does not, output the remaining records
//    from the first file.
while ( strlen( rec1 ) ) {
    of << rec1 << endl;
    if1.getline( rec1, MaxRecLen );
}

//*** If the second file still contains records but the
//    first does not, output the remaining records from
//    the second file.
while ( strlen( rec2 ) ) {
   of << rec2 << endl;
   if2.getline( rec2, MaxRecLen );
}

// Close the input and output files and exit.
```

```
  if1.close();
  if2.close();
  of.close();
  return 0;
}
```

Section 4.10: Maintaining an Address Book

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cstdlib>
#include <cstring>
using namespace std;
const int MaxNo = 30; // max numb of names and addresses
const int MaxFld = 81; // max length of one record
const char filename[ ] = "address.dat";
const int RespSize = 30; // max length of user response
void show( char list[ ][ MaxFld ], int no );
bool add( char name[ ], char list[ ][ MaxFld ], int& no );
bool del( int i, char list[ ][ MaxFld ], int& no );
void init( char list[ ][ MaxFld ], int& no );
void quit( char list[ ][ MaxFld ], int no );

int main() {
   char addr[ MaxNo ][ MaxFld ];
   char name[ MaxFld ];
   char resp[ RespSize ];
   int number;
   int index;
   // read in names from a file if it exists
   // and set number to the number of records
   init( addr, number );
   do {
      cout << "\n\n[S]how names and addresses\n"
           << "[A]dd a name and address\n"
           << "[D]elete a name and address\n"
           << "[Q]uit\n"
           << "Your choice? ";
      cin.getline( resp, RespSize );
      switch ( resp[ 0 ] ) {
```

```
      case 'S':
      case 's':
         show( addr, number );
         break;
      case 'A':
      case 'a':
         cout << "Name and address to add\n";
         cin.getline( name, MaxFld );
         if ( !add( name, addr, number ) )
            cout << "\a*** Out of room, unable to add: "
                 << name << endl;
         break;
      case 'D':
      case 'd':
         cout << "Number of name to delete? ";
         cin.getline( resp, RespSize );
         index = atoi( resp );
         if ( !del( index, addr, number ) )
            cout << "\a*** Unable to delete number: "
                 << index << endl;
         break;
      case 'Q':
      case 'q':
         quit( addr, number );
         break;
      default:
         cout << "\a*** Illegal choice; try again\n";
         break;
      }
   } while ( resp[ 0 ] != 'Q' && resp[ 0 ] != 'q' );
   return 0;
}

// show prints the records and numbers them
// starting with zero. The numbers are printed
```

```
// in a field of width 3, which is why
// setw( 3 ) is used.

// show also prints PerScreen (20) records per screen.
void show( char list[ ][ MaxFld ], int no ) {
   int i;
   char resp[ RespSize ];
   const int PerScreen = 20;
   for ( i = 0; i < no; i++ ) {
      cout << setw( 3 ) << i << ' '
           << list[ i ] << endl;
      if ( ( i + 1 ) % PerScreen == 0 ) {
         cout << "Hit RETURN to continue: ";
         cin.getline( resp, RespSize );
      }
   }
}

// add adds a record if possible and
// updates the number of records
bool add( char name[ ], char list[ ][ MaxFld ], int& no ) {
   int i;
   // out of room?
   if ( no >= MaxNo )
      return false;
   // find correct position for name
   for ( i = no - 1;
         i >= 0 && strcmp( name, list[ i ] ) < 0;
         i-- )
      strcpy( list[ i + 1 ], list[ i ] );
   strcpy( list[ i + 1 ], name );
   no++;
   return true;
}
```

```
// del deletes the record at index i
// and updates the number of records
bool del( int i, char list[ ][ MaxFld ], int& no ) {
   int j;
   // is i in bounds?
   if ( i < 0 || i >= no )
      return false;
   // move names down to delete entry i
   for ( j = i; j < no - 1; j++ )
      strcpy( list[ j ], list[ j + 1 ] );
   no--;
   return true;
}

// init reads names from the file address.dat
// and sets no to the number of records read.
// If address.dat does not exist, init simply sets
// no to zero and returns.
void init( char list[ ][ MaxFld ], int& no ) {
   ifstream in;
   in.open( filename );
   no = 0;
   // check if file exists; if not, return.
   if ( !in )
      return;
   // read records until out of room or end-of-file
   while ( no < MaxNo ) {
      in.getline( list[ no ], MaxFld );
      if ( !strlen( list[ no ] ) )
         break;
      no++;
   }
   in.close();
}
```

```
// quit writes the records to address.dat
void quit( char list[ ][ MaxFld ], int no ) {
   ofstream out;
   out.open( filename );
   int i;
   // write records to address.dat
   for ( i = 0; i < no; i++ )
      out << list[ i ] << endl;
   out.close();
}
```

Section 5.3: A Clock Class

```
class Clock {
public:
   void set( unsigned long t = 0 ) { ticks = t; }
   void tick() { ticks++; }
   void print() { cout << ticks << endl; }
private:
   unsigned long ticks;
};
```

Section 5.4: A Time Stamp Class

```
#include <iostream>
#include <ctime>
#include <cstring>
using namespace std;
class TimeStamp {
public:
   void        set( long s = 0 );
   time_t      get();
   const char* getAsString();
   const char* getYear();
   const char* getMonth();
   const char* getDay();
   const char* getHour();
   const char* getMinute();
   const char* getSecond();
private:
   const char* extract( int, int );
   time_t stamp;
   char   string[ 30 ]; // holds ctime's return string
};

void TimeStamp::set( long s ) {
   if ( s <= 0 )
     stamp = time( 0 );
   else
     stamp = s;
}
time_t TimeStamp::get() {
   return stamp;
}
const char* TimeStamp::getAsString() {
   return ctime( &stamp );
}
```

```
const char* TimeStamp::extract( int offset, int count ) {
   char temp[ 30 ];
   strcpy( temp, ctime( &stamp ) );
   strncpy( string, temp + offset, count );
   string[ count ] = '\0';  // ensure a string
   return string;
}
const char* TimeStamp::getYear() {
   return extract( 20, 4 );
}
const char* TimeStamp::getMonth() {
   return extract( 4, 3 );
}
const char* TimeStamp::getDay() {
   return extract( 0, 3 );
}
const char* TimeStamp::getHour() {
   return extract( 11, 2 );
}
const char* TimeStamp::getMinute() {
   return extract( 14, 2 );
}
const char* TimeStamp::getSecond() {
   return extract( 17, 2 );
}
```

Section 5.6: A Task Class

```
#include <iostream>
#include <ctime>
#include <cstring>
#include <fstream>
using namespace std;
class Task {
public:
   // constructors-destructor
   Task( const char id[ ] );
  ~Task();
   // set-get methods
   void        setST( time_t st = 0 );
   time_t      getST();
   const char* getStrST();
   void        setFT( time_t ft = 0 );
   time_t      getFT();
   const char* getStrFT();
   void        setID( const char ID[ ] ) {
     strncpy( id, ID, MaxId );
     id[ MaxId ] = '\0';
   }
   const char* getID() { return id; }
   double      getDU();
   void        logToFile();
private:
   enum { MaxId = 50, MaxLogFile = 20 };
   Task(); // default constructor disallowed
   TimeStamp  st;
   TimeStamp  ft;
   char       id[ MaxId + 1 ];
   char       logFile[ MaxLogFile + 1 ];
};
```

```
Task::Task( const char ID[ ] ) {
   strncpy( id, ID, MaxId );
   id[ MaxId ] = '\0';
   strcpy( logFile, "log.dat" );
   setST();
   ft = st; // so far, no duration
}
Task::~Task() {
   logToFile(); // if client forgets
}
void Task::logToFile() {
   // set finish if duration still 0
   if ( getFT() == getST() )
     setFT();
   // log the Task's vital statistics
   ofstream outfile( logFile, ios::app );
   outfile << endl << "ID: " << id << endl;
   outfile << "  ST: " << getStrST();
   outfile << "  FT: " << getStrFT();
   outfile << "  DU: " << getDU();
   outfile << endl;
   outfile.close();
}
void Task::setST( time_t ST ) {
   st.set( ST );
}
time_t Task::getST() {
   return st.get();
}
const char* Task::getStrST() {
   return st.getAsString();
}
void Task::setFT( time_t FT ) {
   ft.set( FT );
}
```

```
time_t Task::getFT() {
   return ft.get();
}
const char* Task::getStrFT() {
   return ft.getAsString();
}
double Task::getDU() {
   return difftime( getFT(), getST() );
}
```

Section 5.8: Using the Class CString from MFC

```
#include <afx.h>
#include <iostream>
using namespace std;
int main() {
   // create two CStrings
   CString singer( " Ella  " );
   CString blower( "Miles Davis" );

   // trim white space from singer
   singer.TrimRight(); // " Ella"
   singer.TrimLeft();  // "Ella"
   // append a last name to singer
   singer += " Fitzgerald"; // "Ella Fitzgerald"
   // reverse the string
   singer.MakeReverse();    // "dlaregztiF allE"
   // reverse it back again
   singer.MakeReverse();    // "Ella Fitzgerald"
   // check for characters in a CString
   if ( singer.FindOneOf( "aZ" ) ) // true
     cout << "There's an 'a' or 'Z' in singer." << endl;
   // determine lexicographical order of CStrings
   if ( singer < blower ) // true
     cout << "The singer comes first!" << endl;
   else
     cout << "The blower comes first!" << endl;
   // change case for blower
   blower.MakeLower(); // "miles davis"
   return 0;
}
```

Section 6.3: Tracking Films

```
#include <iostream>
#include <cstring>
using namespace std;
const int asize = 512;

class Film {
public:
   void store_title( const char[ ] );
   void store_director( const char[ ] );
   void store_time( int );
   void store_quality( int );
   void output();
private:
   char title[ asize ];
   char director[ asize ];
   int time; // in minutes
   int quality; // 0 (bad) to 4 (tops)
};

void Film::store_title( const char t[ ] ) {
   strcpy( title, t );
}

void Film::store_director( const char d[ ] ) {
   strcpy( director, d );
}

void Film::store_time( int t ) {
   time = t;
}

void Film::store_quality( int q ) {
   quality = q;
```

```
}

void Film::output() {
   cout << "Title: " << title << endl;
   cout << "Director: " << director << endl;
   cout << "Time: " << time << " mins" << endl;
   cout << "Quality: ";
   for ( int i = 0; i < quality; i++ )
      cout << '*';
   cout << endl;
}

class DirectorCut : public Film {
public:
   void store_rev_time( int );
   void store_changes( const char [ ] );
   void output();
private:
   int rev_time;
   char changes[ asize ];
};

void DirectorCut::store_rev_time( int t ) {
   rev_time = t;
}

void DirectorCut::store_changes( const char s[ ] ) {
   strcpy( changes, s );
}

void DirectorCut::output() {
   Film::output();
   cout << "Revised time: " << rev_time << " mins" << endl;
   cout << "Changes: " << changes << endl;
}
```

```
class ForeignFilm : public Film {
public:
   void store_language( const char [ ] );
   void output();
private:
   char language[ asize ];
};

void ForeignFilm::store_language( const char l[ ] ) {
   strcpy( language, l );
}

void ForeignFilm::output() {
   Film::output();
   cout << "Language: " << language << endl;
}
```

Section 6.6: A Sequence Hierarchy

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <cstring>
using namespace std;
const int MaxStr = 50;
const int MaxSize = 80;

class Sequence {
public:
   bool addS( int, char [ ] );
   bool del( int );
   void output();
   Sequence();
   Sequence( char [ ] );
   ~Sequence();
protected:
   char s[ MaxStr ][ MaxSize ];
   char filename[ MaxSize ];
   int last;
   ifstream in;
   ofstream out;
};

bool Sequence::addS( int pos, char entry[ ] ) {
   if ( last == MaxStr - 1
        || pos < 0
        || pos > last + 1 )
      return false;
   for ( int i = last; i >= pos; i-- )
      strcpy( s[ i + 1 ], s[ i ] );
   strcpy( s[ pos ], entry );
   last++;
```

```
   return true;
}

bool Sequence::del( int pos ) {
   if ( pos < 0 || pos > last )
      return false;

   for ( int i = pos; i < last; i++ )
      strcpy( s[ i ], s[ i + 1 ] );
   last--;
   return true;
}

void Sequence::output() {
   for ( int i = 0; i <= last; i++ )
      cout << i << "  " << s[ i ] << endl;
}

Sequence::Sequence() {
   last = -1;
   filename[ 0 ] = '\0';
}

Sequence::Sequence( char fname[ ] ) {
   last = -1;
   strcpy( filename, fname );
   in.open( filename );
   if ( !in )
      return;
   while ( last < MaxStr - 1 ) {
      in.getline( s[ last + 1 ], MaxSize );
      if ( !strlen( s[ last + 1 ] ) )
         break;
      last++;
   }
```

```
   in.close();
}

Sequence::~Sequence() {
   if ( filename[ 0 ] == '\0' )
      return;
   out.open( filename );
   for ( int i = 0; i <= last; i++ )
      out << s[ i ] << endl;
   out.close();
}

class SortedSeq : public Sequence {
public:
   bool addSS( char [ ] );
   SortedSeq();
   SortedSeq( char [ ] );
protected:
   void sort();
};

void SortedSeq::sort() {
   char temp[ MaxSize ];
   for ( int i = 0; i <= last - 1; i++ ) {
      strcpy( temp, s[ i + 1 ] );
      int j;
      for ( j = i; j >= 0; j-- )
         if ( strcmp( temp, s[ j ] ) < 0 )
            strcpy( s[ j + 1 ], s[ j ] );
         else
            break;
      strcpy( s[ j + 1 ], temp );
   }
}
```

```
bool SortedSeq::addSS( char entry[ ] ) {
   int i;
   for ( i = 0; i <= last; i++ )
      if ( strcmp( entry, s[ i ] ) <= 0 )
         break;
   return addS( i, entry );
}

SortedSeq::SortedSeq() : Sequence() { }

SortedSeq::SortedSeq( char fname[ ] ) : Sequence( fname ) {
   sort();
}
```

Section 7.2: Tracking Films Revisited

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

const int asize = 512;
class Film {
public:
   Film();
   void store_title( const char[ ] = "" );
   void store_director( const char[ ] = "" );
   void store_time( int = 0 );
   void store_quality( int = 0 );
   virtual void output();
   virtual void input( ifstream& );
   static bool read_input( const char[ ], Film*[ ], int );
private:
   char title[ asize ];
   char director[ asize ];
   int time;   // in minutes
   int quality; // 0 (bad) to 4 (tops)
};

Film::Film() {
   store_title();
   store_director();
   store_time();
   store_quality();
}

void Film::store_title( const char t[ ] ) {
   strcpy( title, t );
```

```
}

void Film::store_director( const char d[ ] ) {
   strcpy( director, d );
}

void Film::store_time( int t ) {
   time = t;
}

void Film::store_quality( int q ) {
   quality = q;
}

// Reads title, director, time, and quality.
void Film::input( ifstream& fin ) {
   const unsigned n = 200;
   char inbuff[ n ];
   fin.getline( inbuff, n );
   store_title( inbuff );
   fin.getline( inbuff, n );
   store_director( inbuff );
   fin.getline( inbuff, n );
   store_time( atoi( inbuff ) );
   fin.getline( inbuff, n );
   store_quality( atoi( inbuff ) );
}

// Writes title, director, time, and quality.
void Film::output() {
   cout << "Title: " << title << endl;
   cout << "Director: " << director << endl;
   cout << "Time: " << time << " mins" << endl;
   cout << "Quality: ";
   for ( int i = 0; i < quality; i++ )
```

```
      cout << '*';
   cout << endl;
}

class DirectorCut : public Film {
public:
   DirectorCut();
   void store_rev_time( int = 0 );
   void store_changes( const char [ ] = "" );
   virtual void output();
   virtual void input( ifstream& );
private:
   int rev_time;
   char changes[ asize ];
};

DirectorCut::DirectorCut() {
   store_rev_time();
   store_changes();
}

void DirectorCut::store_rev_time( int t ) {
   rev_time = t;
}

void DirectorCut::store_changes( const char s[ ] ) {
   strcpy( changes, s );
}

// Reads revised time and changes.
void DirectorCut::input( ifstream& fin ) {
   Film::input( fin );
   const unsigned n = 200;
   char inbuff[ n ];
   fin.getline( inbuff, n );
```

```
   store_rev_time( atoi( inbuff ) );
   fin.getline( inbuff, n );
   store_changes( inbuff );
}

// Writes revised time and changes.
void DirectorCut::output() {
   Film::output();
   cout << "Revised time: " << rev_time << endl;
   cout << "Changes: " << changes << endl;
}

class ForeignFilm : public Film {
public:
   ForeignFilm();
   void store_language( const char[ ] = "" );
   virtual void output();
   virtual void input( ifstream& );
private:
   char language[ asize ];
};

ForeignFilm::ForeignFilm() {
   store_language();
}

void ForeignFilm::store_language( const char l[ ] ) {
   strcpy( language, l );
}

// Reads language.
void ForeignFilm::input( ifstream& fin ) {
   Film::input( fin );
   const unsigned n = 200;
   char inbuff[ n ];
```

```
   fin.getline( inbuff, n );
   store_language( inbuff );
}

// Writes language.
void ForeignFilm::output() {
   Film::output();
   cout << "Language: " << language << endl;
}

// class method: Film::read_input
// Reads data from an input file, dynamically creating the
// appropriate Film object for each record group. For
// instance, a ForeignFilm object is dynamically created if
// the data represent a foreign film rather than a regular
// film or a director's cut. Pointers to dynamically created
// objects are stored in the array films of size n. Returns
// true to signal success and false to signal failure.
bool Film::read_input( const char file[ ],
                       Film* films[ ],
                       int n ) {
   const int buffSize = 200;
   char inbuff[ buffSize ];
   ifstream fin( file );
   if ( !fin )  // opened successfully?
     return false; // if not, return false

   // Read until end-of-file. Records fall into
   // groups. 1st record in each group is a string
   // that represents a Film type:
   //   "Film", "ForeignFilm", "DirectorCut", etc.
   // After reading type record, dynamically create
   // an object of the type (e.g., a ForeignFilm object),
   // place it in the array films, and invoke its
   // input method.
```

```
   int next = 0;
   while ( fin.getline( inbuff, buffSize ) && next < n ) {
      if ( strcmp( inbuff, "Film" ) == 0 )
        films[ next ] = new Film();         // regular film
      else if ( strcmp( inbuff, "ForeignFilm" ) == 0 )
        films[ next ] = new ForeignFilm(); // foreign film
      else if ( strcmp( inbuff, "DirectorCut" ) == 0 )
        films[ next ] = new DirectorCut(); // director's cut
      else //**** error condition: unrecognized film type
        continue;
      films[ next++ ]->input( fin ); // polymorphic method
   }
   fin.close();
   return true;
}
```

Section 8.2: A Complex Number Class

```
class Complex {
public:
   Complex();                    // default
   Complex( double );            // real given
   Complex( double, double ); // both given

   void write() const;
   // operator methods
   Complex operator+( const Complex& ) const;
   Complex operator-( const Complex& ) const;
   Complex operator*( const Complex& ) const;
   Complex operator/( const Complex& ) const;
private:
   double real;
   double imag;
};

// default constructor
Complex::Complex() {
   real = imag = 0.0;
}

// constructor -- real given but not imag
Complex::Complex( double re ) {
   real = re;
   imag = 0.0;
}

// constructor -- real and imag given
Complex::Complex( double re, double im ) {
   real = re;
   imag = im;
}
```

```
void Complex::write() const {
   cout << real << " + " << imag << 'i';
}

// Complex + as binary operator
Complex Complex::operator+( const Complex& u ) const {
   Complex v( real + u.real,
              imag + u.imag );
   return v;
}

// Complex - as binary operator
Complex Complex::operator-( const Complex& u ) const {
   Complex v( real - u.real,
              imag - u.imag );
   return v;
}

// Complex * as binary operator
Complex Complex::operator*( const Complex& u ) const {
   Complex v( real * u.real - imag * u.imag,
              imag * u.real + real * u.imag );
   return v;
}

// Complex / as binary operator
Complex Complex::operator/( const Complex& u ) const {
   double abs_sq = u.real * u.real + u.imag * u.imag;
   Complex v( ( real * u.real + imag * u.imag ) / abs_sq,
              ( imag * u.real - real * u.imag ) / abs_sq );
   return v;
}
```

Section 8.7: An Associative Array

```
const int MaxWord = 100;
const int MaxEntries = 100;

class Entry {
public:
   Entry();
   void add( const char*, const char* );
   bool match( const char* ) const;
   void operator=( const char* );
   friend ostream& operator<<( ostream&, const Entry& );
   bool& valid() { return flag; }
private:
   char word[ MaxWord + 1 ];
   char def[ MaxWord + 1 ];
   bool flag;
};

void Entry::operator=( const char* str ) {
   strcpy( def, str );
   flag = true;
}

Entry::Entry() {
   word[ 0 ] = '\0';
   def[ 0 ] = '\0';
   flag = false;
}

ostream& operator<<( ostream& out, const Entry& e ) {
   out << e.word << " defined as: "
       << e.def;
   return out;
}
```

```
void Entry::add( const char* w, const char* d ) {
   strcpy( word, w );
   strcpy( def, d );
}

bool Entry::match( const char* key ) const {
   return strcmp( key, word ) == 0;
}

class Dict {
public:
   friend ostream& operator<<( ostream&, Dict& );
   Entry& operator[ ]( const char* );
private:
   Entry entries[ MaxEntries + 1 ];
};

ostream& operator<<( ostream& out, Dict& d ) {
   for ( int i = 0; i < MaxEntries; i++ )
      if ( d.entries[ i ].valid() )
         out << d.entries[ i ] << endl;
   return out;
}

Entry& Dict::operator[ ]( const char* k ) {
   for ( int i = 0;
         i < MaxEntries && entries[ i ].valid();
         i++ )
      if ( entries[ i ].match( k ) )
         return entries[ i ];
   entries[ i ].add( k , "*** not in dictionary" );
   return entries[ i ];
}
```

Section 9.4: Sorting and Searching

```
#include <iostream>
#include <cstdlib>
using namespace std;
// callback for qsort and bsearch
int comp( const void*, const void* );
unsigned promptAndRead( char prompt[ ] );
unsigned* allocate( unsigned );
void fillVector( unsigned*, unsigned, unsigned );

int main() {
   unsigned* nums; // vector of random integers
   int n;          // its size
   unsigned* keysNotFound; // vector of keys not in nums
   int m;                  // its size
   unsigned range; // of integers in vectors
   int i, j;

   // Prompt for vector size.
   n = promptAndRead( "Vector size? " );
   // Prompt user for range of random numbers as
   // an integer multiple of array size (e.g., if array
   // size is 100, range could be 100 * 2, 100 * 3,
   // 100 * 4, etc.).
   range = promptAndRead( "Multiplier for range (>= 2)? " );
   range *= n;
   range++;
   // Prompt for size of vector of keys not found in nums.
   m = promptAndRead( "Keys-not-found size? " );

   // Allocate heap storage for the vector,
   nums = allocate( n );
   keysNotFound = allocate( m );
```

```
// Fill array with random numbers.
fillVector( nums, n, range );

// Sort it using library function qsort.
qsort( nums,                   // vector to sort
       n,                      // its size
       sizeof( nums[ 0 ] ),    // element size in bytes
       comp );                 // comparison function

// Print sorted array.
for ( i = 0; i < n; i++ )
   cout << nums[ i ] << endl;

// Generate random numbers as keys to be
// searched for in the array, halting when
// m such keys are not in the array. Save
// these keys into an array of their own for
// later printing. Library function bsearch used
// for searching.
unsigned key, count = 0;
j = 0; // index into keysNotFound
do {
   key = rand() % range; // generate random key
   count++;              // count it
   // if key not in vector, add it to keysNotFound
   if ( !bsearch( &key,                 // key's address
                  nums,                 // search vector
                  n,                    // its size
                  sizeof( nums[ 0 ] ),  // element size
                  comp ) )              // callback
     keysNotFound[ j++ ] = key; // add it to vector
} while ( j < m );

// Print keys not found in array, after sorting.
qsort( keysNotFound, m, sizeof( *keysNotFound ), comp );
```

```
   cout << "\n**** " << count
        << " tries to find " << m
        << " keys not in the array." << endl;
   for ( j = 0; j < m; j++ )
      cout << keysNotFound[ j ] << endl;
   return 0;
}

// qsort and bsearch callback:
// returns one of three integer values:
//    < 0 means that *p1 < *p2
//    > 0 means that *p1 > *p2
//      0 means that *p1 == *p2
int comp( const void* p1, const void* p2 ) {
   return *static_cast<const unsigned*>(p1) -
          *static_cast<const unsigned*>(p2);
}

// Prompts user for size of arrays and for range.
unsigned promptAndRead( char prompt[ ] ) {
   int temp;
   // Prompt user for vector size.
   do {
     cout << prompt;
     cin >> temp;
   } while ( temp <= 0 );
   return temp;
}

unsigned* allocate( unsigned n ) {
   return new unsigned[ n ];
}

void fillVector( unsigned* v, unsigned n, unsigned r ) {
   for ( unsigned i = 0; i < n; i++ )
```

```
    v[ i ] = rand() % r;
}
```

Section 10.2: A Template Stack Class

```
#include <iostream>
#include <cstdlib>
//#define NDEBUG //**** enable/disable assertions
#include <cassert>
using namespace std;
template< class T >
class Stack {
public:
   enum { DefaultStack = 10, EmptyStack = -1 };
   Stack();
   Stack( int );
   ~Stack();
   void push( const T& );
   T pop();
   T topNoPop() const;
   bool empty() const;
   bool full() const;
private:
   T* elements;
   int top;
   int size;
   void allocate() {
     elements = new T[ size ];
     top = EmptyStack;
   }
   void msg( const char m[ ] ) const {
     cout << "*** " << m << " ***" << endl;
   }
   friend ostream& operator<<( ostream&, const Stack< T >& );
};

template< class T >
Stack< T >::Stack() {
```

```
   size = DefaultStack;
   allocate();
}

template< class T >
Stack< T >::Stack( int s ) {
   if ( s < 0 )       // negative size?
     s *= -1;
   else if ( 0 == s ) // zero size?
     s = DefaultStack;
   size = s;
   allocate();
}

template< class T >
Stack< T >::~Stack() {
   delete[ ] elements;
}

template< class T >
void Stack< T >::push( const T& e ) {
   assert( !full() );
   if ( !full() )
     elements[ ++top ] = e;
   else
     msg( "Stack full!" );
}

template< class T >
T Stack< T >::pop() {
   assert( !empty() );
   if ( !empty() )
     return elements[ top-- ];
   else {
     msg( "Stack empty!" );
```

```
    T dummy_value;
    return dummy_value; // return arbitrary value
  }
}

template< class T >
T Stack< T >::topNoPop() const {
  assert( top > EmptyStack );
  if ( !empty() )
    return elements[ top ];
  else {
    msg( "Stack empty!" );
    T dummy_value;
    return dummy_value;
  }
}

template< class T >
bool Stack< T >::empty() const {
  return top <= EmptyStack;
}

template< class T >
bool Stack< T >::full() const {
  return top + 1 >= size;
}

template< class T >
ostream& operator<<( ostream& os, const Stack< T >& s ) {
  s.msg( "Stack contents:" );
  int t = s.top;
  while ( t > s.EmptyStack )
    cout << s.elements[ t-- ] << endl;
  return os;
}
```

Section 10.5: Stock Performance Reports

```
#include <iostream>
#include <fstream>
#include <deque>
#include <algorithm>
#include <cstring>
using namespace std;

//*** file names and miscellaneous globals
const char inFile[ ] = "stockData.dat";
const unsigned MaxSymbol = 4;
const char Unknown[ ] = "????";

//*** objects generated from input records
class Stock {
public:
   Stock() {
      strcpy( symbol, Unknown );
      open = close = gainLoss = volume = 0;
   }
   Stock( const char s[ ],      // symbol
          double o,             // opening price
          double c,             // closing price
          unsigned long v ) {   // volume traded
      strncpy( symbol, s, MaxSymbol );
      symbol[ MaxSymbol ] = '\0';
      open = o;
      close = c;
      volume = v;
      gainLoss = ( close - open ) / open;
   }
   const char* getSymbol() const {
      return symbol;
   }
```

```
   double getOpen() const {
      return open;
   }
   double getClose() const {
      return close;
   }
   unsigned long getVolume() const {
      return volume;
   }
   double getGainLoss() const {
      return gainLoss;
   }
private:
   char          symbol[ MaxSymbol + 1 ];
   double        open;     // opening price
   double        close;    // closing price
   double        gainLoss; // gain or loss fraction
   unsigned long volume;   // shares traded
};

//*** Sort comparison: gains in descending order
template< class T >
struct winCmp {
   bool operator()( T& t1, T& t2 ) const  {
      return t1.getGainLoss() > t2.getGainLoss();
   }
};

//*** Sort comparison: volume in descending order
template< class T >
struct volCmp {
   bool operator()( T& t1, T& t2 ) const {
      return t1.getVolume() > t2.getVolume();
   }
};
```

```
//*** invoked by function objects to do output
void output( bool volFlag,
             const char name[ ],
             const char openLabel[ ],  double open,
             const char closeLabel[ ], double close,
             const char gainLabel[ ],  double gain,
             const char volLabel[ ],   unsigned long vol ) {
   cout << "*** " << name << endl;
   if ( volFlag ) // if true, volume comes first
     cout << '\t' << volLabel << vol << endl;
   cout << '\t' << gainLabel  << gain  << endl
        << '\t' << openLabel  << open  << endl
        << '\t' << closeLabel << close << endl;
   if ( !volFlag ) // if false, volume comes last
     cout << '\t' << volLabel << vol << endl;
}

//*** Write Stocks sorted by gain-loss to standard output.
template< class T >
struct winPr {
   void operator()( Stock& s ) {
      output( false,
              s.getSymbol(),
              "Opening Price: ", s.getOpen(),
              "Closing Price: ", s.getClose(),
              "% Changed:     ", s.getGainLoss() * 100,
              "Volume:        ", s.getVolume() );
   }
};

//*** Write Stocks sorted by volume to standard output.
template< class T >
struct volPr {
   void operator()( Stock& s ) {
```

```
      output( true,
              s.getSymbol(),
              "Opening Price: ", s.getOpen(),
              "Closing Price: ", s.getClose(),
              "% Changed:     ", s.getGainLoss() * 100,
              "Volume:        ", s.getVolume() );
   }
};

void herald( const char[ ] );
void input( deque< Stock >& );
int main() {
   deque< Stock > stocks;
   //*** Input stocks and separate into vectors for
   //    winners, losers, and break-evens.
   input( stocks );
   //*** Sort winners in descending order and output.
   herald( "Gainers in descending order: " );
   sort( stocks.begin(), stocks.end(), winCmp< Stock >() );
   for_each( stocks.begin(), stocks.end(), winPr< Stock >() );
   //*** Output losers in ascending order.
   herald( "Losers in ascending order: " );
   for_each( stocks.rbegin(),
             stocks.rend(),
             winPr< Stock >() );
   //*** Sort volume in descending order and output
   herald( "Volume in descending order: " );
   sort( stocks.begin(), stocks.end(), volCmp< Stock >() );
   for_each( stocks.begin(), stocks.end(), volPr< Stock >() );
   return 0;
}

void input( deque< Stock >& d ) {
   char s[ MaxSymbol + 1 ];
   double o, c, v;
```

```
   ifstream input( inFile );
   //*** Read data until end-of-file,
   // creating a Stock object per input record
   while ( input >> s >> o >> c >> v )
     d.insert( d.end(), Stock( s, o, c, v ) );
   input.close();
}

void herald( const char s[ ] ) {
   cout << endl << "******* " << s << endl;
}
```