```
1   // Chapter 8 - Programming Challenge 20, Theatre
    Ticket Sales
2   // This program creates and uses a TicketManager
    class to handle ticket sales
3   // for a small theater with a single one-night
    show. It uses data read from the
4   // SeatAvailability.dat and SeatPrices.dat files,
    which are located on Moodle.
5   // A copy of the data files should be placed in
    the program's project directory.
6
7   #include <iostream>
8   #include <fstream>
9   #include <sstream>
10  #include <iomanip>
11  using namespace std;
12
13  const int NUM_ROWS = 15,
14  ROW_SIZE = 30;
15
16  class TicketManager
17  {
18  private:
19      struct SeatStructure
20      {
21          char   status;              // '#' =
    available; '*' = sold
22          double price;
23
24          SeatStructure()             // Default
    constructor
25          {
26              status = '#';
27              price = 0.0;
28          }
29      };
```

```
30      SeatStructure seat[NUM_ROWS][ROW_SIZE];
31
32      int seatsSold;
33      double totalRevenue;
34
35      string printTickets(int, int, int);
36
37  public:
38      TicketManager();              // Constructor
39      ~TicketManager();             // Destructor
40      string displaySeats();
41      string requestTickets(int, int, int, bool&);
42      string purchaseTickets(int, int, int);
43      string displaySalesReport();
44      string numToString(double num) const;
45  };
46
47  // TicketManager Class Function Implementation
    Section
48
49  /**********************************************
    *************
50   *
    TicketManager                          *
51   * Class constructor that reads seat information
    in from files.*
52   * and uses it to initialize the elements of the
    seat array.    *
53   * It also computes the number of seats sold so
    far and the    *
54   * total revenue to
    date.                                  *
55
    **********************************************
    *************/
56  TicketManager::TicketManager()
```

```
{
    double rowPrice;
    ifstream availFile, priceFile;

    // Open the data files for input
    availFile.open("SeatAvailability.dat");
    if (!availFile)
        cout << "ERROR OPENING SEAT AVAILABILLITY
DATA FILE \n";

    priceFile.open("SeatPrices.dat");
    if (!priceFile)
        cout << "ERROR OPENING PRICES DATA FILE
\n";

    seatsSold = 0;
    totalRevenue = 0.0;

    for (int row = 0; row < NUM_ROWS; row++)
    {
        // Read in the price for this row
        priceFile >> rowPrice;

        // For each seat in the
row                 Do the following:
        for (int col = 0; col < ROW_SIZE; col++)
        {
            availFile >> seat[row][col].status;
    // Read in the seat's status
            seat[row][col].price = rowPrice;
       // Set the seat's price
            if (seat[row][col].status == '*')
      // If seat has been sold,
            {
                seatsSold++;
                        // add to seatsSold and
```

```
totalRevenue
                totalRevenue += rowPrice;
            }
        }
    }
    // Close the files
    availFile.close();
    priceFile.close();
}
/*****************************************************
*************
 *
~TicketManager                              *
 * Class destructor that writes seat availability
information  *
 * back out to the
file.                                          *
*****************************************************
************/
TicketManager::~TicketManager()
{
    ofstream availFile;

    // Open the "avail" data file for output
    availFile.open("SeatAvailability.dat");
    if (!availFile)
        cout << "ERROR OPENING SEAT AVAILABILLITY
DATA FILE \n";

    // Write each seat's status (available or
taken) out to the file
    for (int row = 0; row < NUM_ROWS; row++)
    {
        for (int col = 0; col < ROW_SIZE; col++)
            availFile << seat[row][col].status;
```

```
113         availFile << endl;
114     }
115     // Close the file
116     availFile.close();
117 }
118
119 /*****************************************************
    ****
120  *
    displaySeats                    *
121  * Creates and returns a string that displays a
    chart *
122  * showing sold and available
    seats                          *
123
    *****************************************************
    ***/
124 string TicketManager::displaySeats()
125 {
126     string chart = "\n\n                          Seats
    \n\n";
127     chart += "
    12345678901234567890123456789\n";
128
129     for (int row = 0; row < NUM_ROWS; row++)
130     {
131         chart += "\nRow ";
132         if (row < 9)
133         {
134             chart += static_cast<char>('1' + row);
135             chart += "   ";
136         }
137         else
138         {
139             chart += "1";
140             chart += static_cast<char>('1' + row -
```

```
 10);
141             chart += "   ";
142         }
143
144         for (int col = 0; col < ROW_SIZE; col++)
145         {
146             chart += seat[row][col].status;
147         }
148     }
149
150     // Add the legend
151     chart += "\n\n\n\tLegend:     *  =  Sold";
152     chart += "\n                      #  =
Available";
153
154     return chart;
155 } // end displaySeats
156
157 /*************************************************
************
158  *
requestTickets                          *
159  * Returns a string holding an invoice if the
requested seats *
160  * are available. Returns an appropriate message
if the       *
161  * request is invalid or the seats are not
available.          *
162
*************************************************
***********/
163 string TicketManager::requestTickets(int
numSeatsRequested, int requestedRow, int startSeat
, bool & allOK)
164 {
165     int row = requestedRow - 1,                 //
```

```
Offset user request because array rows
166         firstSeatWanted = startSeat - 1,   // and
columns begin with 0, rather than 1.
167         lastSeatWanted = firstSeatWanted +
numSeatsRequested - 1;
168
169     allOK = true;
170
171     string returnString;
172     double pricePerSeat, totalPrice;
173
174     if (row < 0 || row >= NUM_ROWS)
175     {
176         returnString = "\nIllegal row request.
Rows are numbered 1 to 15.";
177         allOK = false;
178     }
179     else if (firstSeatWanted < 0 || lastSeatWanted
 >= ROW_SIZE)
180     {
181         returnString = "\nOne or more of the
seats you have requested do not exist.\nSeats in
each row are numbered 1 - 30.\n\n";
182         allOK = false;
183     }
184     else
185     {
186         for (int seatWanted = firstSeatWanted;
seatWanted <= lastSeatWanted; seatWanted++)
187         {
188             if (seat[row][seatWanted].status ==
'*')
189             {
190                 returnString = "\nOne or more of
the seats you have requested are already sold.\n";
191                 allOK = false;
```

```
192             }
193         }
194     }
195
196     // If we got this far with allOK still true,
    all requested seats exist and are available for
    purchase.
197     if (allOK)
198     {
199         //pricePerSeat =
    static_cast<int>(seat[row][firstSeatWanted].price);
200         pricePerSeat = seat[row][firstSeatWanted].
    price;
201         totalPrice = pricePerSeat *
    numSeatsRequested;
202
203         returnString = "\nThe seats you have
    requested are available for purchase.\n";
204         returnString += "The total purchase price
    is ";
205
206         returnString += numToString(
    numSeatsRequested);
207         returnString += " seats X $";
208         returnString += numToString(pricePerSeat);
209         returnString += " = $";
210         returnString += numToString(totalPrice);
211     }
212     return returnString;
213 }
214
215 /**************************************************
    **********************
216  *
    purchaseTickets                              *
217  * Handles ticket purchases for a previousy
```

```
validated ticket request.   *
218
**********************************************************
**********************/
219 string TicketManager::purchaseTickets(int numSeats
, int requestedRow, int startSeat)
220 {
221     int row = requestedRow - 1,              //
Offset user request because array rows
222         firstSeat = startSeat - 1,          // and
columns begin with 0, rather than 1.
223         lastSeat = firstSeat + numSeats - 1;
224
225     double pricePerSeat = seat[row][firstSeat].
price;
226     double totalPrice = pricePerSeat * numSeats;
227     double amtPaid,
228         changeDue;
229
230     string tickets;
231
232     // Collect money
233     cout << "\nThe price for the requested
tickets is $ " << totalPrice << endl
234         << "Please input amount paid: $";
235     cin >> amtPaid;
236
237     if (amtPaid >= totalPrice)   // If payment is
sufficient, sell the tickets
238     {
239         // Mark these seats taken
240         for (int purchase = firstSeat; purchase <=
 lastSeat; purchase++)
241             seat[row][purchase].status = '*';
242
243         // Add to seats sold and revenue
```

```
accumulators
244         seatsSold += numSeats;
245         totalRevenue += numSeats * pricePerSeat;
246 
247         // Call printTickets to create a string
holding the actual tickets
248         // Starting row and seat numbering should
be from 1 (not 0)
249         tickets = printTickets(requestedRow,
firstSeat + 1, lastSeat + 1);
250 
251         // Display a purchase summary for the
patron
252         changeDue = amtPaid - totalPrice;
253         cout << "\n\nTickets purchased : " <<
numSeats << endl
254              << "Payment           : $" << setw(6)
<< amtPaid << endl
255              << "Total ticket price: $" << setw(6)
<< totalPrice << endl
256              << "Change due        : $" << setw(6)
<< changeDue << endl << endl;
257     }
258     else
259     {
260         tickets = "Insufficient payment. \n";
261         tickets += "The sale has been cancelled
and your money is being returned.\n\n";
262     }
263     return tickets;
264 }
265 
266 /*************************************************
************
 *
267 printTickets                                    *
```

```
268     * Creates a string holding one ticket for each
        seat sold.    *
269
        ***********************************************
        ***********/
270     string TicketManager::printTickets(int rowNum, int
         firstSeatNum, int lastSeatNum)
271     {
272         double seatPrice = seat[rowNum - 1][
        firstSeatNum - 1].price;
273         string price = numToString(seatPrice);
274
275         string tickets = "";
276
277         for (int seat = firstSeatNum; seat <=
        lastSeatNum; seat++)
278         {
279             tickets +=
        "\n\n\n**********************************************
        ***\n";
280             tickets += "          Little Theater
        Production            \n\n";
281             tickets += "          Row ";
282             tickets += numToString(rowNum);
283             tickets += " ~ Seat ";
284             tickets += numToString(seat);
285             tickets += "\n\n          Price: $ ";
286             tickets += numToString(seatPrice);
287             tickets +=
        "\n**********************************************\
        n";
288         }
289         return tickets;
290     }
291
292     /***********************************************
```

```
**********
 *
displaySalesReport                          *
 * Creates and returns a string holding a ticket
sales report.*                                  *
*
**********************************************************
**********/
string TicketManager::displaySalesReport()
{
    string report = "\n\nLittle Theater Sales
Report\n";
    report += "_____________________________\n\n";
    report += "Seats sold:              ";
    report += numToString(seatsSold);
    report += "\nSeats available:        ";
    report += numToString(NUM_ROWS * ROW_SIZE -
seatsSold);
    report += "\nTotal revenue to date: $";
    report += numToString(totalRevenue);
    report += "\n\n\n";

    return report;
}

/**********************************************************
*******
 *
numToString                          *
 * This function, which converts and returns a
number in *
 * string form, works for both integer and
floating-point*
 * value. It uses an ostringstream object, which
requires*
 * the sstream header
```

```
file.                                   *
317
*****************************************************
*****/
318 string TicketManager::numToString(double num) const
319 {
320     ostringstream ss;
321     ss << num;
322     return ss.str();
323 }
324
325
326 // ****************** Client Program That Uses
    the Class ***************
327
328 // Function prototypes
329 void displayMenu();
330 int  getChoice(int);
331 void getUserRequest(int&, int&, int&);
332
333 const int MAX_MENU_CHOICE = 4;
334 int main()
335 {
336     TicketManager boxOffice;     // Create a
    TicketManager object.
337                                  // The
    constructor will correctly initialize each
338                                  // seat with its
    price and availability status.
339
340     int  choice,                 // User's menu
    choice
341         numSeats,                // Number of
    seats wanted
342         row,                     // Desired row
343         startSeat;               // Desired
```

```
starting seat number
344
345         string message;
346         bool  ticketRequestOK = true; // Are all
    requested tickets available for sale?
347         char  buy;                    // Buy these
    tickets? Y or N.
348
349         // Set print formats
350         cout << fixed << showpoint << setprecision(2);
351
352         do
353         {
354             displayMenu();
355             choice = getChoice(MAX_MENU_CHOICE);
356
357             switch (choice)
358             {
359             case 1:
360                 cout << boxOffice.displaySeats();
                                          // Class
    function call
361                 break;
362             case 2:
363                 getUserRequest(numSeats, row,
    startSeat);                          // Client
    module to get user input
364
365                 cout << boxOffice.requestTickets(
    numSeats, row, startSeat, ticketRequestOK);  //
    Class function call
366
367                 if (ticketRequestOK)
368                 {
369                     cout << "\nDo you wish to
    purchase these tickets (Y/N)? ";
```

```
370                 cin >> buy;
371                 if (toupper(buy) == 'Y')
372                     cout << boxOffice.
purchaseTickets(numSeats, row, startSeat);
// Class function call
373             }
374             break;
375         case 3:
376             cout << boxOffice.displaySalesReport
();                                      // Class
function call
377 
378         }// end switch
379     } while (choice != MAX_MENU_CHOICE);
380 
381     return 0;
382 }// end main
383 
384 /*************************************************
***********
385  *
displayMenu                             *
386  * Displays the menu of program
options                            *
387 
*************************************************
**********/
388 void displayMenu()
389 {
390     // Display menu of choices
391     cout << "\n\n\n\t\tC++ Theatre" << endl <<
endl;
392     cout << "\n\t1.  View Available Seats";
393     cout << "\n\t2.  Request Tickets";
394     cout << "\n\t3.  Display Theater Sales Report";
395     cout << "\n\t4.  Exit the Program\n";
```

```
}

/*************************************************************
 *
getChoice                                  *
 * Accepts, validates, and returns a user
choice.                *

*************************************************************/
int getChoice(int max)
{
    int choice;

    // Get and validate user's choice
    cin >> choice;
    while (choice < 1 || choice > max)
    {
        cout << "Choice must be between 1 and " <<
 max
            << ". Please re-enter: ";
        cin >> choice;
    }
    //Return the choice
    return choice;
}

/*************************************************************
 *
getUserRequest                                *
 * Accepts, validates, and returns user's seat
selection.     *

*************************************************************
```

```
**********/
void getUserRequest(int &numSeats, int &row, int &
startSeat)
{
    cout << "Number of seats desired (1 - 30): ";
    numSeats = getChoice(30);

    cout << "Desired row (1-15): ";
    row = getChoice(15);

    cout << "Desired starting seat number in the
row (1 - 30): ";
    startSeat = getChoice(30);
}

```