```cpp
// Fig. 7.15: GradeBook.h
// Definition of class GradeBook that uses an array to store test grades.
// Member functions are defined in GradeBook.cpp
#include <string>
#include <array>

// GradeBook class definition
class GradeBook
{
public:
   // constant -- number of students who took the test
   static const size_t students = 10; // note public data

   // constructor initializes course name and array of grades
   GradeBook( const std::string &, const std::array< int, students > & );

```

Fig. 7.15 | Definition of class `GradeBook` that uses an array to store test grades. (Part 1 of 2.)

```
      void setCourseName( const std::string & ); // set the course name
      string getCourseName() const; // retrieve the course name
      void displayMessage() const; // display a welcome message
      void processGrades() const; // perform operations on the grade data
      int getMinimum() const; // find the minimum grade for the test
      int getMaximum() const; // find the maximum grade for the test
      double getAverage() const; // determine the average grade for the test
      void outputBarChart() const; // output bar chart of grade distribution
      void outputGrades() const; // output the contents of the grades array
   private:
      std::string courseName; // course name for this grade book
      std::array< int, students > grades; // array of student grades
   }; // end class GradeBook
```

Fig. 7.15 | Definition of class `GradeBook` that uses an array to store test grades. (Part 2 of 2.)

```cpp
// Fig. 7.16: GradeBook.cpp
// GradeBook class member functions manipulating
// an array of grades.
#include <iostream>
#include <iomanip>
#include "GradeBook.h" // GradeBook class definition
using namespace std;

// constructor initializes courseName and grades array
GradeBook::GradeBook( const string &name,
   const array< int, students > &gradesArray )
   : courseName( name ), grades( gradesArray )
{
} // end GradeBook constructor

// function to set the course name
void GradeBook::setCourseName( const string &name )
{
   courseName = name; // store the course name
} // end function setCourseName

```

Fig. 7.16 | GradeBook class member functions manipulating an array of grades. (Part 1 of 8.)

```cpp
// function to retrieve the course name
string GradeBook::getCourseName() const
{
   return courseName;
} // end function getCourseName

// display a welcome message to the GradeBook user
void GradeBook::displayMessage() const
{
   // this statement calls getCourseName to get the
   // name of the course this GradeBook represents
   cout << "Welcome to the grade book for\n" << getCourseName() << "!"
      << endl;
} // end function displayMessage

```

Fig. 7.16 | GradeBook class member functions manipulating an array of grades. (Part 2 of 8.)

```
// perform various operations on the data
void GradeBook::processGrades() const
{
   // output grades array
   outputGrades();

   // call function getAverage to calculate the average grade
   cout << setprecision( 2 ) << fixed;
   cout << "\nClass average is " << getAverage() << endl;

   // call functions getMinimum and getMaximum
   cout << "Lowest grade is " << getMinimum() << "\nHighest grade is "
      << getMaximum() << endl;

   // call function outputBarChart to print grade distribution chart
   outputBarChart();
} // end function processGrades

```

Fig. 7.16 | GradeBook class member functions manipulating an array of grades. (Part 3 of 8.)

```
// find minimum grade
int GradeBook::getMinimum() const
{
   int lowGrade = 100; // assume lowest grade is 100

   // loop through grades array
   for ( int grade : grades )
   {
      // if current grade lower than lowGrade, assign it to lowGrade
      if ( grade < lowGrade )
         lowGrade = grade; // new lowest grade
   } // end for

   return lowGrade; // return lowest grade
} // end function getMinimum

```

Fig. 7.16 | GradeBook class member functions manipulating an array of grades. (Part 4 of 8.)

```cpp
// find maximum grade
int GradeBook::getMaximum() const
{
   int highGrade = 0; // assume highest grade is 0

   // loop through grades array
   for ( int grade : grades )
   {
      // if current grade higher than highGrade, assign it to highGrade
      if ( grade > highGrade )
         highGrade = grade; // new highest grade
   } // end for

   return highGrade; // return highest grade
} // end function getMaximum

```

Fig. 7.16 | GradeBook class member functions manipulating an array of grades. (Part 5 of 8.)

```cpp
// determine average grade for test
double GradeBook::getAverage() const
{
   int total = 0; // initialize total

   // sum grades in array
   for ( int grade : grades )
      total += grade;

   // return average of grades
   return static_cast< double >( total ) / grades.size();
} // end function getAverage

// output bar chart displaying grade distribution
void GradeBook::outputBarChart() const
{
   cout << "\nGrade distribution:" << endl;

   // stores frequency of grades in each range of 10 grades
   const size_t frequencySize = 11;
   array< unsigned int, frequencySize > frequency = {}; // init to 0s

```

Fig. 7.16 | GradeBook class member functions manipulating an array of grades. (Part 6 of 8.)

```
   // for each grade, increment the appropriate frequency
   for ( int grade : grades )
      ++frequency[ grade / 10 ];

   // for each grade frequency, print bar in chart
   for ( size_t count = 0; count < frequencySize; ++count )
   {
      // output bar labels ("0-9:", ..., "90-99:", "100:" )
      if ( 0 == count )
         cout << "  0-9: ";
      else if ( 10 == count )
         cout << "  100: ";
      else
         cout << count * 10 << "-" << ( count * 10 ) + 9 << ": ";

      // print bar of asterisks
      for ( unsigned int stars = 0; stars < frequency[ count ]; ++stars )
         cout << '*';

      cout << endl; // start a new line of output
   } // end outer for
} // end function outputBarChart

```

Fig. 7.16 | GradeBook class member functions manipulating an array of grades. (Part 7 of 8.)

```
// output the contents of the grades array
void GradeBook::outputGrades() const
{
   cout << "\nThe grades are:\n\n";

   // output each student's grade
   for ( size_t student = 0; student < grades.size(); ++student )
      cout << "Student " << setw( 2 ) << student + 1 << ": " << setw( 3 )
         << grades[ student ] << endl;
} // end function outputGrades
```

Fig. 7.16 | GradeBook class member functions manipulating an array of grades. (Part 8 of 8.)

7.6 Case Study: Class `GradeBook` Using an Array to Store Grades (cont.)

- The size of the array is specified as a `public static const` data member `students`.
  - `public` so that it's accessible to the clients of the class.
  - `const` so that this data member is constant.
  - `static` so that the data member is shared by all objects of the class
- There are variables for which each object of a class does not have a *separate copy*.
- That's the case with `static` data members, which are also known as class variables.
- When objects of a class containing `static` data members are created, all the objects share one copy of the class's `static` data members.

7.6 Case Study: Class `GradeBook` Using an Array to Store Grades (cont.)

- A `static` data member can be accessed within the class definition and the member-function definitions like any other data member.
- A `public static` data member can also be accessed outside of the class, *even when no objects of the class exist*, using the class name followed by the binary scope resolution operator (`::`) and the name of the data member.

```cpp
// Fig. 7.17: fig07_17.cpp
// Creates GradeBook object using an array of grades.
#include <array>
#include "GradeBook.h" // GradeBook class definition
using namespace std;

// function main begins program execution
int main()
{
   // array of student grades
   const array< int, GradeBook::students > grades =
      { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
   string courseName = "CS101 Introduction to C++ Programming";

   GradeBook myGradeBook( courseName, grades );
   myGradeBook.displayMessage();
   myGradeBook.processGrades();
} // end main
```

Fig. 7.17 | Creates a `GradeBook` object' using an `array` of grades, then invokes member function `processGrades` to analyze them.

7.7 Sorting and Searching `array`s

- In this section, we use the built-in C++ Standard Library `sort` function to arrange the elements in an `array` into ascending order and the built-in `binary_search` function to determine whether a value is in the `array`.
- Sorting data—placing it into ascending or descending order—is one of the most important computing applications.

7.7 Sorting and Searching `arrays` (cont.)

- Often it may be necessary to determine whether an `array` contains a value that matches a certain key value.
  - Called searching.
- Figure 7.18 begins by creating an unsorted `array` of strings (lines 13–14) and displaying the contents of the `array` (lines 17–19).
- Next, line 21 uses C++ Standard Library function `sort` to sort the elements of the `array` colors into ascending order.
- Lines 24–26 display the contents of the sorted `array`.