```
   void setCourseName( const std::string & ); // set the course name
   std::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 in the grade book
   int getMaximum() const; // find the maximum grade in the grade book
   double getAverage( const std::array< int, tests > & ) const;
   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< std::array< int, tests >, students > grades; // 2D array
}; // end class GradeBook
```

Fig. 7.22 | Definition of class `GradeBook` that uses a two-dimensional `array` to store test grades. (Part 2 of 2.)

```cpp
// Fig. 7.23: GradeBook.cpp
// Member-function definitions for class GradeBook that
// uses a two-dimensional array to store grades.
#include <iostream>
#include <iomanip> // parameterized stream manipulators
using namespace std;

// include definition of class GradeBook from GradeBook.h
#include "GradeBook.h" // GradeBook class definition

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

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

```

Fig. 7.23 | Member-function definitions for class GradeBook that uses a two-dimensional array to store grades. (Part 1 of 9.)

```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

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

```

Fig. 7.23 | Member-function definitions for class `GradeBook` that uses a two-dimensional `array` to store grades. (Part 2 of 9.)

```cpp
    // call functions getMinimum and getMaximum
    cout << "\nLowest grade in the grade book is " << getMinimum()
       << "\nHighest grade in the grade book is " << getMaximum() << endl;

    // output grade distribution chart of all grades on all tests
    outputBarChart();
} // end function processGrades

// find minimum grade in the entire gradebook
int GradeBook::getMinimum() const
{
    int lowGrade = 100; // assume lowest grade is 100

    // loop through rows of grades array
    for ( auto const &student : grades )
    {
       // loop through columns of current row
       for ( auto const &grade : student )
       {
          // if current grade less than lowGrade, assign it to lowGrade
          if ( grade < lowGrade )
             lowGrade = grade; // new lowest grade
       } // end inner for
    } // end outer for
```

Fig. 7.23 | Member-function definitions for class GradeBook that uses a two-dimensional array to store grades. (Part 3 of 9.)

```

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

// find maximum grade in the entire gradebook
int GradeBook::getMaximum() const
{
   int highGrade = 0; // assume highest grade is 0

   // loop through rows of grades array
   for ( auto const &student : grades )
   {
      // loop through columns of current row
      for ( auto const &grade : student )
      {
         // if current grade greater than highGrade, assign to highGrade
         if ( grade > highGrade )
            highGrade = grade; // new highest grade
      } // end inner for
   } // end outer for

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

Fig. 7.23 | Member-function definitions for class `GradeBook` that uses a two-dimensional `array` to store grades. (Part 4 of 9.)

```

// determine average grade for particular set of grades
double GradeBook::getAverage( const array<int, tests> &setOfGrades ) const
{
   int total = 0; // initialize total

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

```

Fig. 7.23 | Member-function definitions for class GradeBook that uses a two-dimensional array to store grades. (Part 5 of 9.)

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

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

   // for each grade, increment the appropriate frequency
   for ( auto const &student : grades )
      for ( auto const &test : student )
         ++frequency[ test / 10 ];

```

Fig. 7.23 | Member-function definitions for class GradeBook that uses a two-dimensional array to store grades. (Part 6 of 9.)

```
   // for each grade frequency, print bar in chart
   for ( size_t count = 0; count < frequencySize; ++count )
   {
      // output bar label ("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. 7.23 | Member-function definitions for class GradeBook that uses a two-dimensional array to store grades. (Part 7 of 9.)

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

   // create a column heading for each of the tests
   for ( size_t test = 0; test < tests; ++test )
      cout << "Test " << test + 1 << "  ";

   cout << "Average" << endl; // student average column heading

```

Fig. 7.23 | Member-function definitions for class GradeBook that uses a two-dimensional array to store grades. (Part 8 of 9.)

```
   // create rows/columns of text representing array grades
   for ( size_t student = 0; student < grades.size(); ++student )
   {
      cout << "Student " << setw( 2 ) << student + 1;

      // output student's grades
      for ( size_t test = 0; test < grades[ student ].size(); ++test )
         cout << setw( 8 ) << grades[ student ][ test ];

      // call member function getAverage to calculate student's average;
      // pass row of grades as the argument
      double average = getAverage( grades[ student ] );
      cout << setw( 9 ) << setprecision( 2 ) << fixed << average << endl;
   } // end outer for
} // end function outputGrades
```

Fig. 7.23 | Member-function definitions for class `GradeBook` that uses a two-dimensional `array` to store grades. (Part 9 of 9.)

```cpp
// Fig. 7.24: fig07_24.cpp
// Creates GradeBook object using a two-dimensional array of grades.
#include <array>
#include "GradeBook.h" // GradeBook class definition
using namespace std;

```

Fig. 7.24 | Creates a GradeBook object using a two-dimensional array of grades, then invokes member function processGrades to analyze them. (Part 1 of 2.)

```
// function main begins program execution
int main()
{
   // two-dimensional array of student grades
   array< array< int, GradeBook::tests >, GradeBook::students > grades =
      { 87, 96, 70,
        68, 87, 90,
        94, 100, 90,
        100, 81, 82,
        83, 65, 85,
        78, 87, 65,
        85, 75, 83,
        91, 94, 100,
        76, 72, 84,
        87, 93, 73 };

   GradeBook myGradeBook(
      "CS101 Introduction to C++ Programming", grades );
   myGradeBook.displayMessage();
   myGradeBook.processGrades();
} // end main
```

Fig. 7.24 | Creates a `GradeBook` object using a two-dimensional `array` of grades, then invokes member function `processGrades` to analyze them. (Part 2 of 2.)

7.10 Introduction to C++ Standard Library Class Template `vector`

- C++ Standard Library class template `vector` is similar to class template `array`, but also supports dynamic resizing.
- Except for the features that modify a `vector`, the other features shown in Fig. 7.25 also work for `array`s.
- Standard class template `vector` is defined in header `<vector>` (line 5) and belongs to namespace `std`.

```cpp
// Fig. 7.25: fig07_25.cpp
// Demonstrating C++ Standard Library class template vector.
#include <iostream>
#include <iomanip>
#include <vector>
#include <stdexcept>
using namespace std;

void outputVector( const vector< int > & ); // display the vector
void inputVector( vector< int > & ); // input values into the vector

int main()
{
   vector< int > integers1( 7 ); // 7-element vector< int >
   vector< int > integers2( 10 ); // 10-element vector< int >

   // print integers1 size and contents
   cout << "Size of vector integers1 is " << integers1.size()
      << "\nvector after initialization:" << endl;
   outputVector( integers1 );

```

Fig. 7.25 | Demonstrating C++ Standard Library class template `vector`. (Part 1 of 7.)

```cpp
    // print integers2 size and contents
    cout << "\nSize of vector integers2 is " << integers2.size()
       << "\nvector after initialization:" << endl;
    outputVector( integers2 );

    // input and print integers1 and integers2
    cout << "\nEnter 17 integers:" << endl;
    inputVector( integers1 );
    inputVector( integers2 );

    cout << "\nAfter input, the vectors contain:\n"
       << "integers1:" << endl;
    outputVector( integers1 );
    cout << "integers2:" << endl;
    outputVector( integers2 );

    // use inequality (!=) operator with vector objects
    cout << "\nEvaluating: integers1 != integers2" << endl;

    if ( integers1 != integers2 )
       cout << "integers1 and integers2 are not equal" << endl;

```

Fig. 7.25 | Demonstrating C++ Standard Library class template `vector`. (Part 2 of 7.)