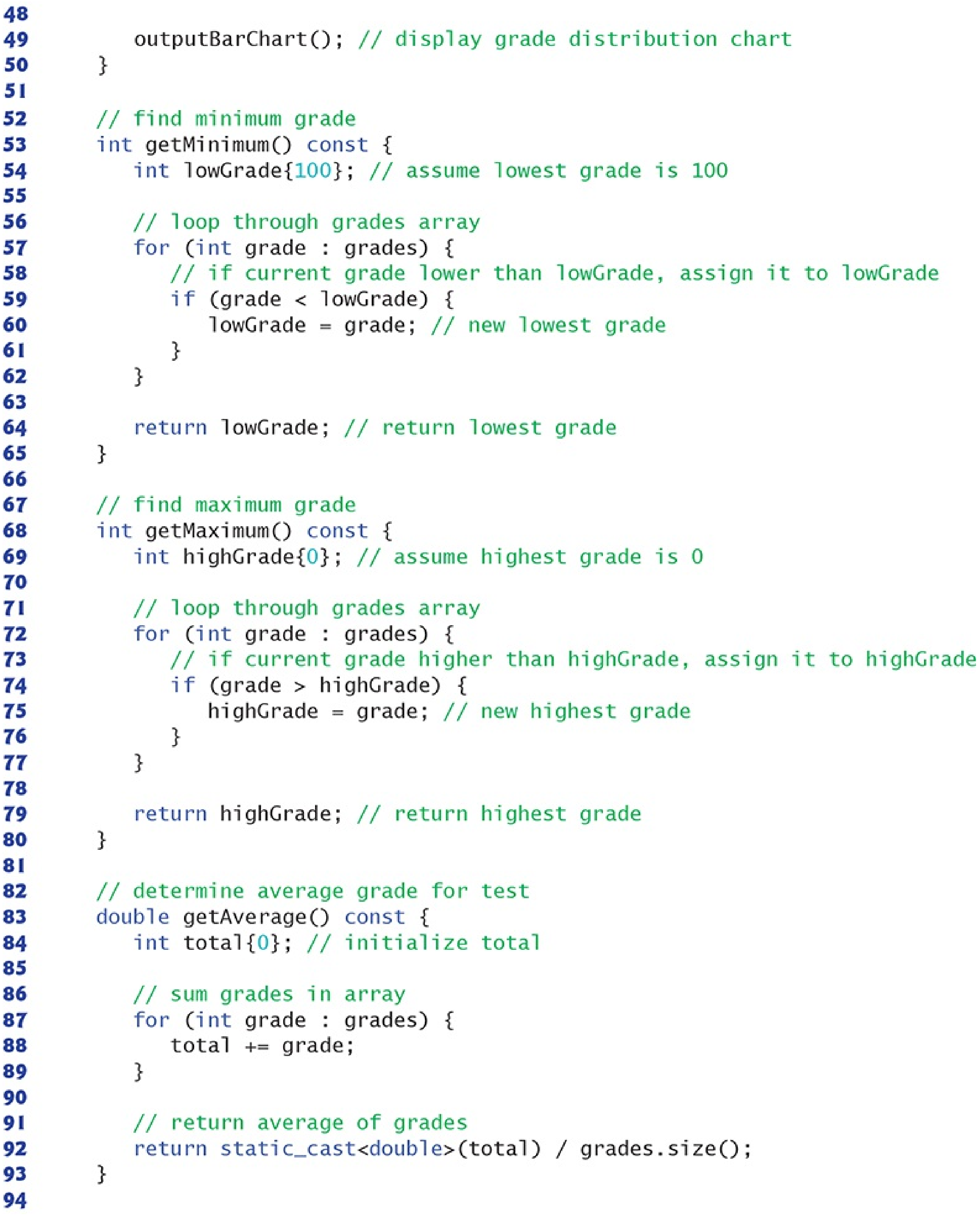
GradeBook we use static and const in student

```cpp
// Fig. 7.13: GradeBook.h
// Definition of class GradeBook that uses an array to store test grades.
#include <string>
#include <array>
#include <iostream>
#include <iomanip> // parameterized stream manipulators

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

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

   // function to retrieve the course name
   const std::string& getCourseName() const {
      return courseName;
   }

   // display a welcome message to the GradeBook user
   void displayMessage() const {
      // call getCourseName to get the name of this GradeBook's course
      std::cout << "Welcome to the grade book for\n" << getCourseName()
         << "!" << std::endl;
   }

   // perform various operations on the data (none modify the data)
   void processGrades() const {
      outputGrades(); // output grades array

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

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

      outputBarChart(); // display grade distribution chart
   }

   // find minimum grade
   int 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
         }
      }

      return lowGrade; // return lowest grade
   }

   // find maximum grade
   int 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
         }
      }

      return highGrade; // return highest grade
   }

   // determine average grade for test
   double 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();
   }

```



```

      outputBarChart(); // display grade distribution chart
   }

   // find minimum grade
   int 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
         }
      }

      return lowGrade; // return lowest grade
   }

   // find maximum grade
   int 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
         }
      }

      return highGrade; // return highest grade
   }

   // determine average grade for test
   double 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();
   }

```

the line was :

```
static const size_t students{10}; //note public data
```

1: understanding `const` in C++

- `const` makes a variable read-only after initialization. (can't modify)
- if a `const` variable is inside an object, each object has its own copy , and its value is set at **runtime**
- if a `const` variable is **static** , it now belongs to the class (not individual objects) and is a **compile-time constant** (if initialized properly)

```
class GradeBook {
public:
    const size_t students{10}; //NOT a compile-time constant
};
```

- even though `student = 10`, it is tied to an object and initialized when an object is created (runtime)
- so the compiler here does not treat it as a compile-time constant.
- so **cannot used for array sizes** because its value is determined at **runtime** and array sized need a variable **compile-time**

2: why `static const` is a true compile-time constant

```
class GradeBook {
public:
    static const size_t students{10};//Compile-time constant
};
```

- now `student` is shared among all objects (only one copy exists) belongs to a class itself not a specific object

- so its initialized at **compile-time** and can be used in array sizes

3: why `const size_t student{10};` is not always a compile-time constant

- event though `const size_t students{10};` looks like a compile-time constant, it is **inside an object**, meaning it is tied to an instance which is normally created at run-time so `students` can't be compile-time
- and also in case of `const size_t students;`, the constructor can change its value , so different objects can have different values

```
class GradeBook {
public:
    const size_t students;
    GradeBook(size_t s) : students{s} {}  //Cannot be used in compile-time expressions
};
```

all this make `students` a **runtime** constant , not a compile-time.

4: why removing `static` affects array declarations

- array require a **compile-time** sizes
- `std::array<int, students>` works only if `students` is **compile-time**
- without `static`, `students` is tied to an object, and the compiler cannot guarantee its value is fixed before run-time and this violates array rules for variables using to size it (MUST compile-time) because compiler need to allocate the appropriate sized in memory for array

> **Note**

as array need a **compile-time constant** for sizes, so best is using `static const` both together.