```
// Assignment 9
#include <iostream>
#include <string>
#include <sstream>

using namespace std;

class Student
{
private:
   string lastName;
   string firstName;
   int totalPoints;
   static int sortKey;

public:
   static const string DEFAULT_NAME;
   static const int DEFAULT_POINTS = 0;
   static const int MAX_POINTS = 1000;
   static const int SORT_BY_FIRST = 88;
   static const int SORT_BY_LAST = 98;
   static const int SORT_BY_POINTS = 108;

public:
   Student(string last, string first, long points);

   string getLastName() { return lastName; }
   string getFirstName() { return firstName; }
   int getTotalPoints() { return totalPoints; }
   static int getSortKey() { return sortKey; }

   bool setLastName(string last);
   bool setFirstName(string first);
   bool setPoints(int pts);
   static bool setSortKey(int key);

   static int compareTwoStudents(Student firstStud, Student secondStud);
   string toString();

private:
   static bool validString(string testStr);
   static bool validPoints(int testPoints);
};
class StudentArrayUtilities
{
public:
   static string toString(string title, Student data[], int arraySize);
   static void arraySort(Student array[], int arraySize);
   static double getMedianDestructive(Student array[], int arraySize);
   static void printArray(string title, Student data[], int arraySize);

private:
   static bool floatLargestToTop(Student data[], int top);
   static void mySwap(Student &a, Student &b);
};
const string Student::DEFAULT_NAME = "Unknown";
int Student::sortKey = SORT_BY_LAST;

int main()
{
   Student evenStudents[] =
   {
```

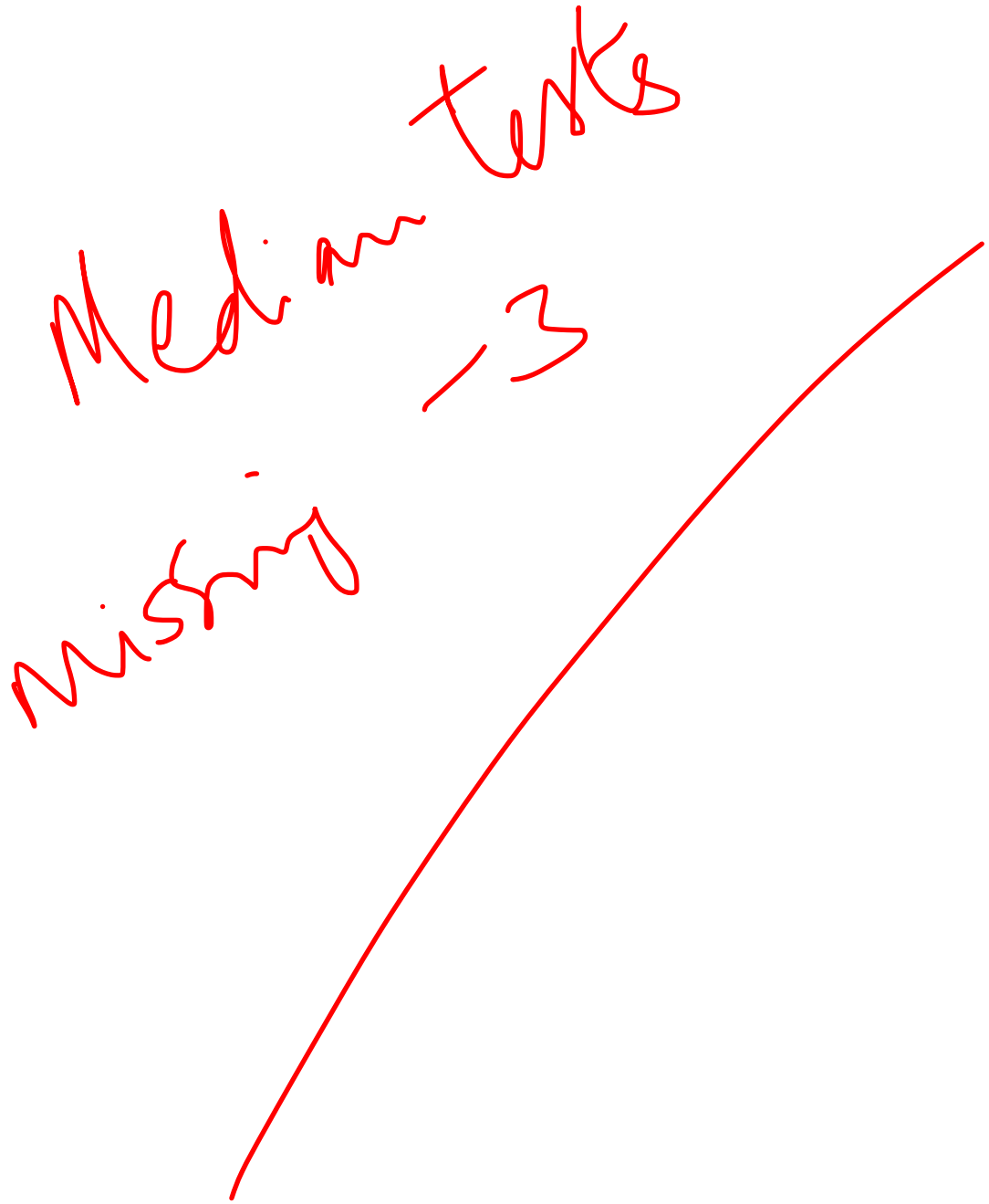
```
        Student("Loq", "Yuna", 150), Student("Delta", "Lance", 98),
        Student("Simpson", "Homer", 5), Student("Mate", "Coffee", 198),
        Student("Lee", "Tiff", 200), Student("Pickles", "Dill", 120),
        Student("Shortman", "Arnold", 160), Student("High", "Hau", 180),
        Student("Chilipeppa", "Lorena", 190), Student("Sky", "Sora", 165),
        Student("Ryder", "Flint", 155), Student("Charming", "Prince", 175),
        Student("Gieser", "Kai", 184), Student("Dixon", "Amanda", 168),
        Student("Moore", "Randi", 188), Student("Dalin", "Jenn", 192)
    };
    Student oddStudents[] =
    {
        Student("Loq", "Yuna", 150), Student("Delta", "Lance", 98),
        Student("Mate", "Coffee", 198), Student("Lee", "Tiff", 200),
        Student("Pickles", "Dill", 120), Student("Shortman", "Arnold", 160),
        Student("High", "Hau", 180), Student("Chilipeppa", "Lorena", 190),
        Student("Sky", "Sora", 165), Student("Ryder", "Flint", 155),
        Student("Charming", "Prince", 175), Student("Gieser", "Kai", 184),
        Student("Dixon", "Amanda", 168), Student("Moore", "Randi", 188),
        Student("Dalin", "Jenn", 192)
    };

    Student singleStudents[] =
    {
        Student("Loq", "Yuna", 150)
    };

    short evenArraySize = sizeof(evenStudents) / sizeof(evenStudents[0]);
    short oddArraySize = sizeof(oddStudents) / sizeof(oddStudents[0]);
    short singleArraySize = sizeof(singleStudents) / sizeof(singleStudents[0]);

    // Kept in to test if everything was working
    StudentArrayUtilities::printArray("Before: ", evenStudents, evenArraySize);
    StudentArrayUtilities::arraySort(evenStudents, evenArraySize);
    StudentArrayUtilities::printArray("After: ", evenStudents, evenArraySize);

    // Was trying to figure out how to call toString method. Not Successful. Trie
d to get help at STEM Center.
    /*
    Student s = evenStudents[];
    string resultString;
    resultString = s.toString();
    cout << resultString;
    */


}
Student::Student(string last, string first, long points)
{
    if (!setLastName(last))
        lastName = DEFAULT_NAME;
    if (!setFirstName(first))
        firstName = DEFAULT_NAME;
    if (!setPoints(points))
        totalPoints = DEFAULT_POINTS;
}

bool Student::setLastName(string last)
{
    if (!validString(last))
        return false;
    lastName = last;
    return true;
```

```
}

bool Student::setFirstName(string first)
{
   if (!validString(first))
      return false;
   firstName = first;
   return true;
}

bool Student::setPoints(int pts)
{
   if (!validPoints(pts))
      return false;
   totalPoints = pts;
   return true;
}

bool Student::setSortKey(int key)
{
   if (key != SORT_BY_LAST && key !=  SORT_BY_FIRST && key != SORT_BY_POINTS)
      return false;
   sortKey = key;
      return true;
}

bool Student::validString(string testStr)
{
   if (testStr.length() > 0 && isalpha(testStr[0]))
      return true;
   return false;
}

bool Student::validPoints(int testPoints)
{
   if (testPoints >= 0 && testPoints <= MAX_POINTS)
      return true;
   return false;
}
string Student::toString()
{
   string resultString;
   ostringstream cnvrtFirst, cnvrtLast, cnvrtPoints;

   cnvrtFirst << firstName;
   cnvrtLast << lastName;
   cnvrtPoints << totalPoints;

   resultString = " " + cnvrtLast.str()
      + ", " + cnvrtFirst.str()
      + " points: " + cnvrtPoints.str()
      + "\n";
   return resultString;
}
int Student::compareTwoStudents(Student firstStud, Student secondStud)
{
   int sortKey = getSortKey();
   switch (getSortKey())
   {
   case SORT_BY_FIRST:
      return firstStud.firstName.compare(secondStud.firstName);
      break;
```

```
   case SORT_BY_LAST:
      return firstStud.lastName.compare(secondStud.lastName);
      break;
   case SORT_BY_POINTS:
      return firstStud.getTotalPoints() - secondStud.getTotalPoints();
      break;
   }
   return 0;
}
// Method that was Suppose to replace printArray.
/*
string StudentArrayUtilities::toString(string title, Student data[], int arraySi
ze)
{
   string output = "";

   cout << title << endl;

   for (int k = 0; k < arraySize; k++)
      output += " " + data[k].toString();

   return output;
}
*/
void StudentArrayUtilities::printArray(string title, Student data[], int arraySi
ze)
{
   string output = "";

   cout << title << endl;

   // build the output string from the individual Students:
   for (int k = 0; k < arraySize; k++)
      output += " " + data[k].toString();

   cout << output << endl;
}
void StudentArrayUtilities::arraySort(Student array[], int arraySize)
{
   for (int k = 0; k < arraySize; k++)
      if (!floatLargestToTop(array, arraySize - 1 - k))
         return;
}

bool StudentArrayUtilities::floatLargestToTop(Student data[], int top)
{
   bool changed = false;

   for (int k = 0; k < top; k++)
      if (Student::compareTwoStudents(data[k], data[k + 1]) > 0)
      {
         mySwap(data[k], data[k + 1]);
         changed = true;
      }
   return changed;
}

void StudentArrayUtilities::mySwap(Student &a, Student &b)
{
   Student temp("", "", 0);

   temp = a;
```

```
   a = b;
   b = temp;
}

// Used to calculate Median.
double StudentArrayUtilities::getMedianDestructive(Student array[], int arraySiz
e)
{
   int srtKey = Student::getSortKey();
   Student::setSortKey(Student::SORT_BY_POINTS);

   arraySort(array, arraySize);

   Student::setSortKey(srtKey);

   if (arraySize >= 2 && arraySize % 2 == 0)
      return(array[arraySize / 2 - 1].getTotalPoints()
      + array[arraySize / 2].getTotalPoints()) / 2;
   if (arraySize >= 1 && arraySize % 2 == 1)
      return array[arraySize / 2].getTotalPoints();

   return 0;
}

/* ------------------------------ run ----------------------------

Before:
Loq, Yuna points: 150
Delta, Lance points: 98
Simpson, Homer points: 5
Mate, Coffee points: 198
Lee, Tiff points: 200
Pickles, Dill points: 120
Shortman, Arnold points: 160
High, Hau points: 180
Chilipeppa, Lorena points: 190
Sky, Sora points: 165
Ryder, Flint points: 155
Charming, Prince points: 175
Gieser, Kai points: 184
Dixon, Amanda points: 168
Moore, Randi points: 188
Dalin, Jenn points: 192

After:
Charming, Prince points: 175
Chilipeppa, Lorena points: 190
Dalin, Jenn points: 192
Delta, Lance points: 98
Dixon, Amanda points: 168
Gieser, Kai points: 184
High, Hau points: 180
Lee, Tiff points: 200
Loq, Yuna points: 150
Mate, Coffee points: 198
Moore, Randi points: 188
Pickles, Dill points: 120
Ryder, Flint points: 155
Shortman, Arnold points: 160
Simpson, Homer points: 5
Sky, Sora points: 165
```

```
Press any key to continue . . .
---------------------------------------------------------------- */
```



```
// Lab 09 - Instructor Solution:
// Original - Prof. Loceff, Updates, Edits, Annotations: &
//
//Notes:
//- Correct access qualifiers (private/public)
//- Correct use of getters/setters
//- Correct use of global consts
//- Use of symbolic consts rather than literals (magics)
//- No output in interior methods
//- sortKey correctly preserved
//- appropriate and thorough testing
//-
//- Faithfulness to spec

#include <string>
#include <iostream>
#include <sstream>
using namespace std;

// ------------------------------------------------------------------------
class Student {
private:
    string lastName;
    string firstName;
    int totalPoints;

public:
    static const string DEFAULT_NAME;
    static const int DEFAULT_POINTS = 0;
    static const int MAX_POINTS = 1000;

public:
    Student(string last, string first, int pts);

    // accessors and mutators
    string getLastName() { return lastName; }
    string getFirstName() { return firstName; }
    int getTotalPoints() { return totalPoints; }

    bool setLastName(string last);
    bool setFirstName(string first);
    bool setPoints(int pts);

    static int compareTwoStudents(Student first, Student second);
    string toString();

private:
    static bool validString(string testStr);
    static bool validPoints(int testPoints);

    // sort and ordering support
public:
    static  const int SORT_BY_FIRST = 88;
```

```
    static  const int SORT_BY_LAST = 98;
    static  const int SORT_BY_POINTS = 108;

private:
    static int sortKey;

public:
    static bool setSortKey(int key);
    static  int getSortKey() { return sortKey; }
};

// Defaults. Note that non-numeric statics can't be inited inline
int Student::sortKey = Student::SORT_BY_LAST;
const string Student::DEFAULT_NAME = "zz-error";


// end of class Student ------------------------------------------------------

class StudentArrayUtilities
{
public:
    static string toString(string title, Student data[], int arraySize);
    static void arraySort(Student array[], int arraySize);
    static double getMedianDestructive(Student array[], int arraySize);

private:
    static bool floatLargestToTop(Student data[], int top);
    static void mySwap(Student &a, Student &b);
};

// end of class Student ------------------------------------------------------

int main() {
    Student  evenClass[] = {
        Student("smith","fred", 95),
        Student("bauer","jack",123),
        Student("jacobs","carrie", 195),  Student("renquist","abe",148),
        Student("3ackson","trevor", 108),  Student("perry","fred",225),
        Student("loceff","fred", 44),  Student("stollings","pamela",452),
        Student("charters","rodney", 295),  Student("cassar","john",321)
    };

    Student  oddClass[] = {
        Student("smith","fred", 95),
        Student("bauer","jack",123),
        Student("jacobs","carrie", 195),  Student("renquist","abe",148),
        Student("3ackson","trevor", 108),  Student("perry","fred",225),
        Student("loceff","fred", 44),  Student("stollings","pamela",452),
        Student("charters","rodney", 295),  Student("cassar","john",321),
        Student( "odd", "manout", 100 )
    };

    Student  smallClass[] = {
```

```
    Student( "smith", "fred", 95 )
};

Student emptyClass[] = {};

// Note that the correct way to compute the size is to use sizeof(Student)
// in the denom, not sizeof(array[0]) 'cos it may be empty
int arraySize = sizeof(evenClass) / sizeof(Student);

cout << StudentArrayUtilities::toString("Before: ", evenClass, arraySize)
     << endl;

StudentArrayUtilities::arraySort(evenClass, arraySize);
cout << StudentArrayUtilities::toString("After default sort: ",
                                        evenClass, arraySize)
     << endl;

Student::setSortKey(Student::SORT_BY_FIRST);
StudentArrayUtilities::arraySort(evenClass, arraySize);
cout << StudentArrayUtilities::toString("After sort by first: ",
                                        evenClass, arraySize)
     << endl;

Student::setSortKey(Student::SORT_BY_POINTS);
StudentArrayUtilities::arraySort(evenClass, arraySize);
cout << StudentArrayUtilities::toString("After sort by points: ",
                                        evenClass, arraySize)
     << endl;

// test median
Student::setSortKey(Student::SORT_BY_LAST);

cout << "Median of evenClass = "
     << StudentArrayUtilities::getMedianDestructive(evenClass, arraySize)
     << endl;

cout << "Sort key successfully preserved? "
     << (Student::getSortKey() == Student::SORT_BY_LAST ? "YES" : "NO")
     << endl;

// test odd class
arraySize = sizeof(oddClass) / sizeof(Student);
cout << "Median of oddClass = "
     << StudentArrayUtilities::getMedianDestructive(oddClass, arraySize)
     << endl;

// test one-student class
arraySize = sizeof(smallClass) / sizeof(Student);
cout << "Median of smallClass = "
<< StudentArrayUtilities::getMedianDestructive(smallClass, arraySize)
<< endl;

// test empty class
```

```
    arraySize = sizeof(emptyClass) / sizeof(Student);
    cout << "Median of emptyClass = "
         << StudentArrayUtilities::getMedianDestructive(emptyClass, arraySize)
         << endl;
}

// Student method implementations -----------------------------------------

Student::Student(string last, string first, int points) {
    if (!setLastName(last))
        lastName = DEFAULT_NAME;
    if (!setFirstName(first))
        firstName = DEFAULT_NAME;
    if (!setPoints(points))
        totalPoints = DEFAULT_POINTS;
}

bool Student::setLastName(string last) {
    if (!validString(last))
        return false;
    lastName = last;
    return true;
}

bool Student::setFirstName(string first) {
    if ( !validString(first) )
        return false;
    firstName = first;
    return true;
}

bool Student::setPoints(int pts) {
    if ( !validPoints(pts) )
        return false;
    totalPoints = pts;
    return true;
}

string Student::toString() {
    return lastName + ", " + firstName + ". Points: " + to_string(totalPoints);
}

bool Student::validString(string testStr) {
    return testStr.length() > 0 && isalpha(testStr[0]);
}

bool Student::validPoints(int testPoints) {
    return testPoints >= 0 && testPoints <= MAX_POINTS;
}

bool Student::setSortKey(int key) {
    switch (key) {
        case SORT_BY_FIRST:
```

```
        case SORT_BY_LAST:
        case SORT_BY_POINTS:
            sortKey = key;
            return true;
        default:
            return false;
    }
}

int Student::compareTwoStudents(Student firstStud, Student secondStud) {
    switch (sortKey) {
        case SORT_BY_FIRST:
            return firstStud.firstName.compare(secondStud.firstName);
        case SORT_BY_LAST:
            return firstStud.lastName.compare(secondStud.lastName);
        case SORT_BY_POINTS:
            return firstStud.totalPoints - secondStud.totalPoints;
        default:
            return 0;
    }
}
// end of Student method definitions  ----------------------------------------

string StudentArrayUtilities::toString(string title, Student data[],
                                       int arraySize) {
    string output =  title + "\n";

    for (int k = 0; k < arraySize; k++)
        output += " " + data[k].toString() + "\n";

    return output;
}

void StudentArrayUtilities::arraySort(Student array[], int arraySize) {
    for (int k = 0; k < arraySize; k++)
        // compare with method def to see where inner loop stops
        if (!floatLargestToTop(array, arraySize-1-k))
            return;
}

// returns true if a modification was made to the array
bool StudentArrayUtilities::floatLargestToTop(Student data[], int top) {
    bool changed = false;

    // compare with client call to see where the loop stops
    for (int k =0; k < top; k++) {
        if (Student::compareTwoStudents(data[k], data[k+1]) > 0) {
            mySwap(data[k], data[k+1]);
            changed = true;
        }
    }
    return changed;
}
```

```
void StudentArrayUtilities::mySwap(Student &a, Student &b) {
    Student temp("", "", 0);

    temp = a;
    a = b;
    b = temp;
}

// median -- this method is allowed to return with array in new order
double StudentArrayUtilities::getMedianDestructive(Student arr[], int arrSize) {
    if (arrSize <= 0) return 0;

    int saveSortKey;
    double retVal = 0.0;

    // preserve the client's sortKey and set it to sort by points
    saveSortKey = Student::getSortKey();
    Student::setSortKey( Student::SORT_BY_POINTS );

    // Now sort by points to prep for computing median
    arraySort(arr, arrSize);

    if (arrSize % 2 == 0) {
        int n1 = (arrSize / 2 - 1);
        int n2 = arrSize / 2;
        retVal = (arr[n1].getTotalPoints() + arr[n2].getTotalPoints())/2.0;
    }
    else
        retVal = arr[arrSize / 2].getTotalPoints();

    // restore the sort key to what it was before the call
    Student::setSortKey(saveSortKey);

    return retVal;
}
// end of StudentArrayUtilities method definitions  -----------

/* -------------------------- run ----------------------------
 Before:
 smith, fred. Points: 95
 bauer, jack. Points: 123
 jacobs, carrie. Points: 195
 renquist, abe. Points: 148
 zz-error, trevor. Points: 108
 perry, fred. Points: 225
 loceff, fred. Points: 44
 stollings, pamela. Points: 452
 charters, rodney. Points: 295
 cassar, john. Points: 321

 After default sort:
 bauer, jack. Points: 123
```

```
cassar, john. Points: 321
charters, rodney. Points: 295
jacobs, carrie. Points: 195
loceff, fred. Points: 44
perry, fred. Points: 225
renquist, abe. Points: 148
smith, fred. Points: 95
stollings, pamela. Points: 452
zz-error, trevor. Points: 108

After sort by first:
renquist, abe. Points: 148
jacobs, carrie. Points: 195
loceff, fred. Points: 44
perry, fred. Points: 225
smith, fred. Points: 95
bauer, jack. Points: 123
cassar, john. Points: 321
stollings, pamela. Points: 452
charters, rodney. Points: 295
zz-error, trevor. Points: 108

After sort by points:
loceff, fred. Points: 44
smith, fred. Points: 95
zz-error, trevor. Points: 108
bauer, jack. Points: 123
renquist, abe. Points: 148
jacobs, carrie. Points: 195
perry, fred. Points: 225
charters, rodney. Points: 295
cassar, john. Points: 321
stollings, pamela. Points: 452

Median of evenClass = 171.5
Sort key successfully preserved? YES
Median of oddClass = 148
Median of smallClass = 95
Median of emptyClass = 0
Program ended with exit code: 0

------------------------------------------------------------ */
```