```
1  /*****************************************************************************
2   * AUTHOR       : JOSHUA SALZEDO, Chris Burrell
3   * LAB #8       : Array Train - Set # 1
4   * CLASS        : CS1A
5   * SECTION      : MW: 8AM
6   * DUE DATE     : 04/23/2018
7   *****************************************************************************/
8
9  #include "functions.h"
10
11 /*****************************************************************************
12  * FUNCTION SumIntArray
13  * ---------------------------------------------------------------------------
14  * Outputs the sum of all elements within an integer array
15  * ---------------------------------------------------------------------------
16  * PRE-CONDITIONS
17  *      The following parameters must have defined values:
18  *          ARR
19  *          arrLen
20  * POST-CONDITIONS
21  *      ==> returns sum
22  *****************************************************************************/
23 int SumIntArray(const int ARR[], // IN - integer array
24                 int arrLen)      // IN - Array length
25 {
26     int sum;
27     sum = 0;
28
29     // sanity check
30     if (arrLen > 0)
31     {
32         // loop over items in collection
33         for (int index = 0; index < arrLen; ++index)
34         {
35             // and accumulate them
36             sum += ARR[index];
37         }
38     }
39     return sum;
40 }
41 /*****************************************************************************
42  * FUNCTION AverageIntArray
43  * ---------------------------------------------------------------------------
44  * Outputs the average of the elements of the array
45  * ---------------------------------------------------------------------------
46  * PRE-CONDITIONS
47  *      The following parameters must have defined values:
48  *          ARR
49  *          arrLen
50  *          searchInt
51  * POST-CONDITIONS
52  *      ==> returns index: the position of the item's first occurrence
53  *****************************************************************************/
54 double AverageIntArray(const int ARR[], // IN - integer array
55                        int arrLen)      // IN - Array length
56 {
57     // define
58     int sum;
59     double average;
60
61     //initialize
```

```
 62     average = 0.0;
 63
 64     // sanity check
 65     if (arrLen > 0)
 66     {
 67         // loop over and add all elements to the accumulator `sum`
 68         sum = SumIntArray(ARR, arrLen);
 69         average = double(sum) / arrLen;
 70     }
 71     return average;
 72 }
 73 /*****************************************************************************
 74  * FUNCTION FindFirstInstance
 75  * ---------------------------------------------------------------------------
 76  * Outputs the index the first time input integer occurs within the array
 77  * ---------------------------------------------------------------------------
 78  * PRE-CONDITIONS
 79  *      The following parameters must have defined values:
 80  *          ARR
 81  *          arrLen
 82  *          searchInt
 83  * POST-CONDITIONS
 84  *      ==> returns index: the position of the item's first occurrence
 85  *****************************************************************************/
 86 int FindFirstInstance(const int ARR[], // IN - integer array
 87                       int arrLen,      // IN - length of array
 88                       int searchInt)   // IN - int to search for
 89 {
 90     int index;
 91     bool found;
 92
 93     found = false;
 94     index = 0;
 95
 96     while (!found && index < arrLen)
 97     {
 98         found = ARR[index] == searchInt;
 99         if (!found)
100         {
101             ++index;
102         }
103     }
104     return index;
105 }
106 /*****************************************************************************
107  * FUNCTION FindOccurrencesInt
108  * ---------------------------------------------------------------------------
109  * Outputs the number of times an int occurs within the input array
110  * ---------------------------------------------------------------------------
111  * PRE-CONDITIONS
112  *      The following parameters must have defined values:
113  *          ARR
114  *          arrLen
115  *          searchInt
116  * POST-CONDITIONS
117  *      ==> returns occurrences: the number of times an int occurs
118  *****************************************************************************/
119 int FindOccurrencesInt(const int ARR[], // IN - integer array
120                        int arrLen,      // IN - length of array
121                        int searchInt)   // IN - int to search for
122 {
```

```
123     int occurrences;
124     occurrences = 0;
125 
126     // loop over collection
127     for (int index = 0; index < arrLen; ++index)
128     {
129         // check if the item is equal to the search term
130         if (ARR[index] == searchInt)
131         {
132             ++occurrences;
133         }
134     }
135 
136     return occurrences;
137 
138 }
139 
140 /*************************************************************************
141  * FUNCTION FindString
142  * ----------------------------------------------------------------------
143  * Outputs the index of the searched string within an array
144  * ----------------------------------------------------------------------
145  * PRE-CONDITIONS
146  *      The following parameters must have defined values:
147  *          ARR
148  *          arrLen
149  *          SEARCH_STR
150  * POST-CONDITIONS
151  *      ==> returns Index: the index of the searched name
152  *************************************************************************/
153 int FindString(const string ARR[],             // IN - integer array
154                int arrLen,                  // IN - Array length
155                const string &string1)   // IN - string to search for
156 {
157     int index;
158     bool found;
159 
160     found = false;
161     index = 0;
162 
163     while (!found && index < arrLen)
164     {
165         found = ARR[index] == string1;
166         if (!found)
167         {
168             ++index;
169         }
170     }
171     return index;
172 }
173 /*************************************************************************
174  * FUNCTION FindLargestInt
175  * ----------------------------------------------------------------------
176  * Outputs the index of the largest element within the input array
177  * ----------------------------------------------------------------------
178  * PRE-CONDITIONS
179  *      The following parameters must have defined values:
180  *          ARR
181  *          arrLen
182  * POST-CONDITIONS
183  *      ==> returns minIndex: largest element's index
```

```
 ******************************************************************************/
int FindLargestInt(const int ARR[], // IN - integer array
                   int arrLen)      // IN - Array length
{
    int max;
    int maxIndex;
    int item;

    max = 0;
    maxIndex = arrLen;

    for (int index = 0; index < arrLen; ++index)
    {
        item = ARR[index];
        if (item > max)
        {
            max = item;
            maxIndex = index;
        }
    }
    return maxIndex;
}
/******************************************************************************
 * FUNCTION FindSmallestInt
 * ----------------------------------------------------------------------------
 * Outputs the index of the smallest element within the input array
 * ----------------------------------------------------------------------------
 * PRE-CONDITIONS
 *      The following parameters must have defined values:
 *          ARR
 *          arrLen
 * POST-CONDITIONS
 *      ==> returns minIndex: smallest element's index
 ******************************************************************************/
int FindSmallestInt(const int ARR[], // IN - integer array
                    int arrLen)      // IN - Array length
{
    // declare
    int min;
    int item;
    int minIndex;

    //Initialize
    min = 1000;
    minIndex = arrLen;

    for (int index = 0; index < arrLen; ++index)
    {
        item = ARR[index];
        if (item < min)
        {
            min = item;
            minIndex = index;
        }
    }
    return minIndex;
}
```