Case Study: Intersection of Sets

In algebra, the intersection of two sets is defined as a new set that contains all the values common to both sets. For instance, consider the following two sets, *A* and *B*:

$A = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
$B = \{2, 4, 8, 12, 14, 20, 25, 28, 30, 32\}$

The only values common to both sets are 2, 4, and 8. The intersection of *A* and *B*, which is denoted as $A \cap B$, is

$A \cap B = \{2, 4, 8\}$

This case study illustrates how array contents can be processed to perform an operation such as finding the intersection of two sets. The program will ask the user to enter two sets of values, each stored in an array. Then it will scan the two arrays looking for values common to both. The common values will be stored in a third array, whose contents are displayed on the screen.

Variables

Table 1 lists the variables needed.

Table 1

Variable	Description
`set1`	An array of 10 integers to hold the first set
`set2`	An array of 10 integers to hold the second set
`intersection`	An array of 10 integers to hold the intersection of set1 and set2
`numIntValues`	An integer holding the number of intersecting values

Functions

Table 2 lists the functions used by the program.

Table 2

Function	Description
`getArray`	`set1` and `set2` are passed into the function. It prompts the user to enter 10 values for each array.
`findIntersection`	`set1`, `set2`, and `intersection` are passed into the function. It scans the arrays for values that appear in both. The intersecting values are stored in `intersection`. This function returns the number of intersecting values found.
`displayIntValue`	The `intersection` array and the `numIntValues` variable are passed into this function. If `numIntValues` contains a number greater than zero, the function displays that many elements in the `intersection` array. If there are no intersecting values, the function displays a a message indicating so.

The `findIntersection` Function

The `findIntersection` function uses two array parameters, `first` and `second`. The arrays `set1` and `set2` are passed into these parameters. The function uses nested loops to find the values that appear in both arrays. Here is the pseudocode.

```
For each element in the first array
   For each element in the second array
      Compare the selected elements in both arrays.
      If they contain the same value
         Store the value in the intersect array.
         Increment the count of intersecting values.
      End If.
   End For.
End For.
Return the count of intersecting values.
```

In the actual code, the outer loop cycles a counter variable (`index1`) through the values 0 through 9. This variable is used as the subscript for `set1`. The inner loop also cycles a counter variable (`index2`) through the values 0 through 9. This variable is used as a subscript for `set2`. For each iteration of the outer loop, the inner loop goes through all its iterations. An `if` statement in the inner loop compares `first[index1]` to `second[index2]`. Because the inner loop iterates 10 times for each iteration of the outer loop, the function will compare each individual element of the `first` array to every element of the `second` array. Here is the C++ code for the function.

```
int findIntersection(int first[], int second[], int intersect[], int size)
{
   int intCount = 0,                // Number of intersecting values
       index3 = 0;                  // Subscript variable for intersect array

   for (int index1 = 0; index1 < size; index1++)
   {
      for(int index2 = 0; index2 < size; index2++)
      {
         if (first[index1] == second[index2])
         {
            intersect[index3] = first[index1];
            index3++;
            intCount++;
         }
      }
   }
   return intCount;                  // Return the number of intersecting values.
}
```

Program 7-28 shows the entire program's source code.

Program 7-28

```
 1 // This program allows the user to enter two sets of numbers.
 2 // It finds the intersection of the two sets (which is the
 3 // set of numbers contained in both sets). The intersecting
 4 // values are displayed.
 5 #include <iostream>
 6 using namespace std;
 7
 8 // Function Prototypes
 9 void getArrays(int [], int [], int);
10 int findIntersection(int [], int [], int [], int);
11 void displayIntValues(int [], int);
12
13 int main()
14 {
15    const int NUM_VALUES = 10;    // Number of values in each array
16    int set1[NUM_VALUES],         // First set
17        set2[NUM_VALUES],         // Second set
18        intersection[NUM_VALUES], // Set containing intersection values
19        numIntValues;             // number of values in intersection
20
21    // Get values for the sets.
22    getArrays(set1, set2, NUM_VALUES);
23
24    // Find the intersection of the two sets
25    numIntValues = findIntersection(set1, set2,
26                        intersection, NUM_VALUES);
27
```

(program continues)

Program 7-28 *(continued)*

```
28    // Display the intersecting values
29    displayIntValues(intersection, numIntValues);
30    return 0;
31 }
32
33 //*********************************************************
34 // Definition of function getArrays                       *
35 // This function accepts two int arrays as arguments.     *
36 // It prompts the user to enter 10 values for each array *
37 //*********************************************************
38
39 void getArrays(int first[], int second[], int size)
40 {
41    // Get values for first array.
42    cout << "Enter 10 values for the first set:\n";
43    for (int index = 0; index < size; index++)
44       cin >> first[index];
45
46    // Get values for second array.
47    cout << "Enter 10 values for the second set:\n";
48    for (index = 0; index < size; index++)
49       cin >> second[index];
50 }
51
52
53 //*************************************************************
54 // Definition of function findIntersection                    *
55 // This functon accepts three arrays as arguments.            *
56 // The first two arrays (first and second) are scanned,       *
57 // and all values appearing in both are stored in the         *
58 // third array (intersect). The number of values that appear *
59 // in both arrays is returned.                                *
60 //*************************************************************
61
62 int findIntersection(int first[], int second[], int intersect[], int size)
63 {
64    int intCount = 0, // Number of intersecting values
65       index3 = 0;    // Subscript variable for intersect array
66
67    for (int index1 = 0; index1 < size; index1++)
68    {
69       for(int index2 = 0; index2 < size; index2++)
70       {
```

(program continues)

Program 7-28 *(continued)*

```
71          if (first[index1] == second[index2])
72          {
73             intersect[index3] = first[index1];
74             index3++;
75             intCount++;
76          }
77       }
78    }
79    return intCount;  // Return the number of intersecting values.
80 }
81
82 //*******************************************************
83 // Definition of function displayIntValues              *
84 // This function accepts two arguments: an array of ints *
85 // and an int. The second argument is the number of     *
86 // valid elements contained in the array.               *
87 // These values are displayed, if there are any.        *
88 //*******************************************************
89
90 void displayIntValues(int intersect[], int num)
91 {
92    if (!num)
93       cout << "There are no intersecting values.\n";
94    else
95    {
96       cout << "Here is a list of the intersecting values:\n";
97       for (int index = 0; index < num; index++)
98          cout << intersect[index] << " ";
99       cout << endl;
100   }
101 }
```

Program Output with Example Input Shown in Bold

```
Enter 10 values for the first set:
1 2 3 4 5 6 7 8 9 10
Enter 10 values for the second set:
2 4 8 12 14 20 25 28 30 32
Here is a list of the intersecting values:
2 4 8
```