CASE STUDY

8 Set Intersection Case Study

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 8A lists the variables needed.

Table 8A Variables Used in the Set Intersection Program

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 8B lists the functions used by the program.

Table 8B Functions Used in the Set Intersection Program

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 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 to `SET_SIZE-1`, where `SET_SIZE` is the number of elements in each of the sets. This variable is used as the subscript for `set1`. The inner loop also cycles a counter variable (`index2`) through the values 0 to `SET_SIZE-1`. 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 `SET_SIZE` 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 intCount = 0, index3 = 0;

   for (int index1 = 0; index1 < SET_SIZE; index1++)
   {
      for(int index2 = 0; index2 < SET_SIZE; index2++)
      {
         if (first[index1] == second[index2])
         {
            intersect[index3] = first[index1];
            index3++;
            intCount++;
         }
      }
   }
   return intCount;
}
```

Here is the source code for the completed program.

SetIntersect.cpp

```
// This program allows the user to enter two sets of numbers and then
// finds the intersection of the two sets (i.e., the set of numbers
// contained in both sets). The intersecting values are displayed.
#include <iostream>
using namespace std;

// Function prototypes
void getArrays(int [], int []);
int findIntersection(int [], int [], int []);
void displayIntValues(int [], int);

const int SET_SIZE = 10;

int main()
{
   int set1[SET_SIZE],          // First set
       set2[SET_SIZE],          // Second set
       intersection[SET_SIZE],  // Set containing intersection values
       numIntValues;            // Number of values in intersection set

   getArrays(set1, set2);
   numIntValues = findIntersection(set1, set2, intersection);
   displayIntValues(intersection, numIntValues);
   return 0;
}

```

```
27 /*****************************************************
28  *                      getArrays                    *
29  * This function receives two int arrays as arguments *
30  * and fills them with values input by the user.      *
31  *****************************************************/
32 void getArrays(int first[], int second[])
33 {
34    int index;
35
36    // Get values for first array.
37    cout << "Enter " << SET_SIZE << " values for the first set:\n";
38    for (index = 0; index < SET_SIZE; index++)
39       cin >> first[index];
40
41    // Get values for second array.
42    cout << "\nEnter " << SET_SIZE << " values for the second set:\n";
43    for (index = 0; index < SET_SIZE; index++)
44       cin >> second[index];
45 }
46
47 /*****************************************************
48  *                   findIntersection                *
49  * This function receives three int arrays as arguments.  *
50  * The first two arrays (first and second) are scanned,   *
51  * and all values appearing in both are stored in the     *
52  * third array (intersect). The number of values common   *
53  * to both arrays is returned.                            *
54  *****************************************************/
55 int findIntersection(int first[], int second[], int intersect[])
56 {
57    int intCount = 0, index3 = 0;
58
59    for (int index1 = 0; index1 < SET_SIZE; index1++)
60    {
61       for(int index2 = 0; index2 < SET_SIZE; index2++)
62       {
63          if (first[index1] == second[index2])
64          {
65             intersect[index3] = first[index1];
66             index3++;
67             intCount++;
68          }
69       }
70    }
71    return intCount;
72 }
73
```

```
74 /*********************************************************
75  *                       displayIntValues                *
76  * This function receives two arguments: an array of ints   *
77  * and an int indicating the number of valid elements in    *
78  * the array. These values are displayed, if there are any. *
79  *********************************************************/
80 void displayIntValues(int intersect[], int num)
81 {
82    if (!num)          // Same as saying if (num == 0)
83       cout << "\nThere are no intersecting values.\n";
84    else
85    {
86       cout << "\nHere is a list of the intersecting values:\n";
87       for (int index = 0; index < num; index++)
88          cout << intersect[index] << " ";
89       cout << endl;
90    }
91 }
```

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]
Enter 10 values for the second set:
2 4 8 12 14 20 25 28 30 32[Enter]
Here is a list of the intersecting values:
2 4 8
```