The distance function expressed in terms of the `pair` template class from `<utility>`

The distance function calculates the distance between two points, p1= (x1, y1), and p2 = (x2, y2) in the Cartesian Plane. In traditional math notation, that formula looks like this:

$$d = \sqrt{(\Delta x)^2 + (\Delta y)^2} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

Section 2.6 describes a `pair<T1,T2>` class that is part of the Standard Template Library. To use this class, we have the following context:

```
#include <utility>
using namespace std;
```

In practice, in the expression `pair<T1,T2>`, the variables `T1` and `T2` are actually replaced with types—either predefined types such as `int`, `float`, `double`, `char`, or user-defined classes such as `Student`, `Course`, `Queue`, etc.

For example, we can use a instance of `pair<double,double>` to represent a point (x,y), and declare a function prototype as follows:

```
double distanceBetween(pair<double,double> p1, pair <double,double> p2);
```

In each instance of a `pair<double,double>`, there are two members, `.first` and `.second`. These can be used to represent x and y, respectively.

With this information, you should be able to write the definition of the function `distanceBetween` declared above.

CS32 F15 H06 Handout—Page 2

Consider the code from Figure 12.2 of the textbook (reproduced below).

Here is a table that shows how to trace through this code on a problem where we are search for the number 72 in a sorted array:
`int a[11]={13,21, 34, 41, 55, 66, 72, 86, 94, 107, 118};`

We show the computation for each recursive call, with the values of **first** and **size** passed in, the value of **middle** computed for that step, and the values of **first** and **size** passed to the next recursive call.

target=72															
step	values passed in		middle (first + size/2)	array values											values passed to next step	
	first passed in	size passed in		a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	first = first, or middle+1	size = size/2, or (size-1)/2
step 1	0	11	5 = 0 + (11/2)	13	21	34	41	55	66	72	86	94	107	118	6 = 5 + 1	5 = (11-1)/2
step 2	6	5	8 = 6 + (5/2)							72	86	94	107	118	6 = 6	2 = (5)/2
step 3	6	2	7 = 6 + (2/2)							72	86				6 = 6	1 = (2)/2
step 4	6	1	6=6+(1/2)							72						

FIGURE 12.2 The Binary Search Function

A Function Implementation

```
void search(
    const int a[ ],
    size_t first,
    size_t size,
    int target,
    bool& found,
    size_t& location
)
// Precondition: The array segment starting at a[first] and containing size elements is sorted
// from smallest to largest.
// Postcondition: The array segment starting at a[first] and containing size elements has been
// searched for the target. If the target was present, then found is true, and location is set so
// that target == a[location]. Otherwise, found is set to false.
// Library facilities used: cstdlib (provides size_t from namespace std)
{
    size_t middle;

    if (size == 0)
        found = false;
    else
    {
        middle = first + size/2;
        if (target == a[middle])
        {
            location = middle;
            found = true;
        }
        else if (target < a[middle])
            // The target is less than a[middle], so search before the middle.
            search(a, first, size/2, target, found, location);
        else
            // The target must be greater than a[middle], so search after the middle.
            search(a, middle+1, (size-1)/2, target, found, location);
    }
}
```