# Exercise 1

Gorecki, Nicholas

Software Engineering Process II

Finished By: 9.20.2018 23:36

## DESCRIPTION OF SOLUTION

The functionality of my solution for the binary searching is as follows:

- Empty or negative size will return false
- The search will search the middle of the sub-selected array and check if that number is larger or smaller than the expected target
    - If it is smaller, it will shift the end to the center selected minus one
    - If it is larger, it will shift the home to the center selected plus one
- The search will continue to happen until the home is greater than the end or until the target is found. Only then it will exit.

## DESCRIPTION OF TESTS

1. The first test just searches as normal against a list of whole numbers, still stored as doubles, though.

```
//Normal Searching Whole Numbers
TEST (binarySearch, wholeNumbers)
{
  double wholeNumbers[9] = {8, 12, 67, 112, 456, 567, 890, 1001, 2011};
  EXPECT_EQ(true, contains(12, wholeNumbers, 9));
}
```

2. The second test searches normally against a list of potential non-whole numbers, testing the functionality of checking within the10e-6 value range.\

```
//Normal Searching NonWhole Numbers
TEST (binarySearch, nonWholeNumbers)
{
  double nonWholeNumbers[11] = {2.8, 4.6, 6, 8, 10, 12, 16, 18, 22.1, 78.3, 111.6};
  EXPECT_EQ(true, contains(78.3, nonWholeNumbers, 11));
}
```

3. The third test checks if numbers at the end of the array can be reached.

```
//Boundary Checking End Of Array Element
TEST (binarySearch, endArrayCheck)
{
  double nonWholeNumbers[11] = {2.8, 4.6, 6, 8, 10, 12, 16, 18, 22.1, 78.3, 111.6};
  EXPECT_EQ(true, contains(111.6, nonWholeNumbers, 11));
}
```

4. The fourth test checks if numbers at the beginning of the array can be reached.

```
//Boundary Checking Front Of Array Element
TEST (binarySearch, frontArrayCheck)
{
  double nonWholeNumbers[11] = {2.8, 4.6, 6, 8, 10, 12, 16, 18, 22.1, 78.3, 111.6};
  EXPECT_EQ(true, contains(2.8, nonWholeNumbers, 11));
}
```

5. The fifth test checks if the search can handle checking for a number not inside the array inside of its boundary values.

```
//Checking If An Array Doesn't Contain A Value Inside Boundary
TEST (binarySearch, arrayDoesntContainInside)
{
  double nonWholeNumbers[11] = {2.8, 4.6, 6, 8, 10, 12, 16, 18, 22.1, 78.3, 111.6};
  EXPECT_EQ(false, contains(8.7, nonWholeNumbers, 11));
}
```

6. The sixth test checks if the search can handle checking for a number not inside the array outside of its boundary values.

```
//Checking If An Array Doesn't Contain A Value Outside Boundary
TEST (binarySearch, arrayDoesntContainOutside)
{
  double nonWholeNumbers[11] = {2.8, 4.6, 6, 8, 10, 12, 16, 18, 22.1, 78.3, 111.6};
  //Lower End
  EXPECT_EQ(false, contains(-3.1, nonWholeNumbers, 11));
  //Upper End
  EXPECT_EQ(false, contains(556.2, nonWholeNumbers, 11));
}
```

7. The seventh test checks if the search can handle an array with one element.

```
//Checking If Search Can Handle One Element Array
TEST (binarySearch, oneElement)
{
  double number[1] = {5.4};
  EXPECT_EQ(true, contains(5.4, number, 1));
}
```

8. The eighth test checks if the search can handle an array with no elements.

```
//Checking if Search Can Handle No Element Array
TEST (binarySearch, noLengthArray)
{
  double emptyArray[0] = {};
  EXPECT_EQ(false, contains(1.2, emptyArray, 0));
}
```

9. The ninth test checks if the search doesn't crash when a length greater than the actual array size is input, causing the search to access memory outside of the array bounds.

```
//Checking If Search Can Handle Length Greater Than Actual Array Size
TEST (binarySearch, smallerThanExpectedDoesntContain)
{
  double smallArray[3] = {2.1, 3.4, 5.6};
  EXPECT_EQ(false, contains(9.7, smallArray, 6));
}
```

10. The tenth tests checks if the search can handle having an array input that contains the target, but doesn't actually have all elements filled in. Since an improper size is input here, it should return false.

```
//Checking If Search Can Handle An Array With A Max Size, But Not All
//Elements Are Filled In
TEST (binarySearch, unfinishedSize)
{
  double numbers[7] = {2.3, 5.6, 4.9};
  EXPECT_EQ(false, contains(5.6, numbers, 7));
}
```

**RUNNING OF ALL TESTS PASSING**

```
[----------] Global test environment tear-down
[==========] 10 tests from 1 test case ran. (8 ms total)
[  PASSED  ] 10 tests.
Removing intermediate container c6323b97b9bf
 ---> 3659c0de4439
Successfully built 3659c0de4439
```

EXAMPLE OUTPUT OF RUNNING A SEARCH:

```
[ RUN      ] binarySearch.endArrayCheck
Selected: 12
Result: 1
Selected: 22.1
Result: 1
Selected: 78.3
Result: 1
Selected: 111.6
Result: 0
[       OK ] binarySearch.endArrayCheck (0 ms)
```

The above snippet is an example of the endArrayCheck test, where it checks if the end element of the array can be reached when searched for. It shows every iteration of the process: When the result is one, that means the element selected is greater than the target value, meaning that the upper half of the sub-selection of the array must be chosen. If it is -1, then the lower half of the sub-selected array must be chosen. If the result is 0, then that means the selected element in the array is equal to the target and it should end there.

ALL TESTS RUNNING:

1. The search finds the selected target value, returning a result of zero and thus true

```
[ RUN      ] binarySearch.wholeNumbers
Selected: 456
Result: -1
Selected: 12
Result: 0
```

2. The search finds the selected target value, returning a result of zero and thus true

```
[ RUN      ] binarySearch.nonWholeNumbers
Selected: 12
Result: 1
Selected: 22.1
Result: 1
Selected: 78.3
Result: 0
[       OK ] binarySearch.nonWholeNumbers (0 ms)
```

3. The search finds the selected target value, returning a result off zero and thus true

```
[ RUN      ] binarySearch.endArrayCheck
Selected: 12
Result: 1
Selected: 22.1
Result: 1
Selected: 78.3
Result: 1
Selected: 111.6
Result: 0
[       OK ] binarySearch.endArrayCheck (0 ms)
```

4. The search finds the selected target value, returning a result of zero and thus true

```
[ RUN      ] binarySearch.frontArrayCheck
Selected: 12
Result: -1
Selected: 6
Result: -1
Selected: 2.8
Result: 0
[       OK ] binarySearch.frontArrayCheck (1 ms)
```

5. The search attempts to find the target, but the target is not inside the array. The result finally returns as nonzero and thus false

```
[ RUN      ] binarySearch.arrayDoesntContainInside
Selected: 12
Result: -1
Selected: 6
Result: 1
Selected: 8
Result: 1
Selected: 10
Result: -1
[       OK ] binarySearch.arrayDoesntContainInside (0 ms)
```

6. The search attempts to find the target, but the target is not inside the array. The result finally returns as nonzero and thus false. There are two tests ran in this unit, which would explain why there are two restarts of the number 12.

```
[ RUN      ] binarySearch.arrayDoesntContainOutside
Selected: 12
Result: -1
Selected: 6
Result: -1
Selected: 2.8
Result: -1
Selected: 12
Result: 1
Selected: 22.1
Result: 1
Selected: 78.3
Result: 1
Selected: 111.6
Result: 1
[       OK ] binarySearch.arrayDoesntContainOutside (1 ms)
```

7. The search find the target in the one element array and thus true

```
[ RUN      ] binarySearch.oneElement
Selected: 5.4
Result: 0
[       OK ] binarySearch.oneElement (0 ms)
```

8. The search checks if the array is greater than 0, which it isn't, so it returns false as an invalid array

```
[ RUN      ] binarySearch.noLengthArray
Invalid Array
[       OK ] binarySearch.noLengthArray (0 ms)
```

9. The search attempts to find a number against an array with all elements filled, but an invalid length greater than the arrays is input. This in turn returns a nonzero result, given that data outside of the array was accessed, and thus false

```
[ RUN      ] binarySearch.smallerThanExpectedDoesntContain
Selected: 5.6
Result: 1
Selected: 6.95272e-310
Result: 1
Selected: 0
Result: 1
[       OK ] binarySearch.smallerThanExpectedDoesntContain (0 ms)
```

10. The search attempts to find a number inside an array where not all elements are filled in, but the max size is input instead of the current size. This results in a nonzero result and thus false

```
[ RUN      ] binarySearch.unfinishedSize
Selected: 0
Result: 1
Selected: 0
Result: 1
Selected: 0
Result: 1
[       OK ] binarySearch.unfinishedSize (0 ms)
```

## CHANGES TO CODE TO BREAK TESTS

1. I swapped the yellow boxed numbers in the following snippet, of which failed the test case for the first test by inverting the search process, thus making it never reach the target value since its going the ‘opposite way’

```
if (selected >= target - 0.000001 && selected <= target + 0.000001)
{
  return 0;
}
else if (selected > target + 0.000001)
{
  return 1;
}
else
{
  return -1;
}
```

```
[ RUN      ] binarySearch.wholeNumbers
Selected: 456
Result: 1
Selected: 890
Result: 1
Selected: 1001
Result: 1
Selected: 2011
Result: 1
BinarySearchTest.cpp:8: Failure
      Expected: true
To be equal to: contains(12, wholeNumbers, 9)
      Which is: false
[  FAILED  ] binarySearch.wholeNumbers (0 ms)
```

2. This swap also failed the second test

```
[ RUN      ] binarySearch.nonWholeNumbers
Selected: 12
Result: -1
Selected: 6
Result: -1
Selected: 2.8
Result: -1
BinarySearchTest.cpp:15: Failure
      Expected: true
To be equal to: contains(78.3, nonWholeNumbers, 11)
      Which is: false
[  FAILED  ] binarySearch.nonWholeNumbers (0 ms)
```

3. It also failed the third test

```
[ RUN      ] binarySearch.endArrayCheck
Selected: 12
Result: -1
Selected: 6
Result: -1
Selected: 2.8
Result: -1
BinarySearchTest.cpp:22: Failure
      Expected: true
To be equal to: contains(111.6, nonWholeNumbers, 11)
      Which is: false
[  FAILED  ] binarySearch.endArrayCheck (0 ms)
```

4. It failed the fourth test

```
[ RUN      ] binarySearch.frontArrayCheck
Selected: 12
Result: 1
Selected: 22.1
Result: 1
Selected: 78.3
Result: 1
Selected: 111.6
Result: 1
BinarySearchTest.cpp:29: Failure
      Expected: true
To be equal to: contains(2.8, nonWholeNumbers, 11)
      Which is: false
[  FAILED  ] binarySearch.frontArrayCheck (0 ms)
```

5. I changed the following code in the yellow highlighted box from false to true, which fails the fifth test. This, in turn, makes it so the while loop is never entered, thus making the searching impossible. This gives back false feedback about if it is actually contained or not

```
//initialize variables
int home = 0;
int end = size - 1;
double selected;
int center;
bool found = true;
```

```
[ RUN      ] binarySearch.arrayDoesntContainInside
BinarySearchTest.cpp:36: Failure
      Expected: false
To be equal to: contains(8.7, nonWholeNumbers, 11)
      Which is: true
[  FAILED  ] binarySearch.arrayDoesntContainInside (0 ms)
```

6. This change also failed the sixth test

```
[ RUN      ] binarySearch.arrayDoesntContainOutside
BinarySearchTest.cpp:44: Failure
      Expected: false
To be equal to: contains(-3.1, nonWholeNumbers, 11)
      Which is: true
BinarySearchTest.cpp:46: Failure
      Expected: false
To be equal to: contains(556.2, nonWholeNumbers, 11)
      Which is: true
[  FAILED  ] binarySearch.arrayDoesntContainOutside (0 ms)
```

7. I changed the following code in the yellow box from size <= 0 to size <= 1, thus making one element arrays return false, even if they have the proper value inside

```
if (size <= 1)
  {
    cout << "Invalid Array" << endl;
    return false;
  }
```

```
[ RUN      ] binarySearch.oneElement
Invalid Array
BinarySearchTest.cpp:53: Failure
      Expected: true
To be equal to: contains(5.4, number, 1)
      Which is: false
[  FAILED  ] binarySearch.oneElement (0 ms)
```

8. This one is a stubborn one to try to fail. Even if I change the following code in the yellow box to size < 0, it will still go through and return false, given that the home will be zero and the end will be the size (which is zero) minus 1 (so -1) and the while loop condition will become false. I could always modify multiple lines of code to make this fail, sure, but I'm trying to keep it to subtle value/conditional changes and not out due it to try and make something fail (one mutant per attempt to fail code)

```
bool contains(double target, double numbers[], int size)
{
  if (size <= 0)
    {
      cout << "Invalid Array" << endl;
      return false;
    }
  //initialize variables
  int home = 0;
  int end = size - 1;
  double selected;
  int center;
  bool found = false;

  //conditional to loop until all items are searched
  while (end >= home && !found)
  {
```

9. I changed the following code in the yellow highlighted box from false to true, which fails the ninth test. This makes it so the search loop is never reached, thus making the feedback false

```
//initialize variables
int home = 0;
int end = size - 1;
double selected;
int center;
bool found = true;
```

```
[ RUN      ] binarySearch.smallerThanExpectedDoesntContain
BinarySearchTest.cpp:67: Failure
      Expected: false
To be equal to: contains(9.7, smallArray, 6)
      Which is: true
[  FAILED  ] binarySearch.smallerThanExpectedDoesntContain (0 ms)
[ RUN      ] binarySearch.unfinishedSize
BinarySearchTest.cpp:75: Failure
      Expected: false
To be equal to: contains(5.6, numbers, 7)
      Which is: true
[  FAILED  ] binarySearch.unfinishedSize (0 ms)
```

10. I swapped the yellow boxed numbers in the following snippet, of which failed the test case for the tenth test. This basically inverts the searching process, thus making it so the search never reaches the unfilled space allocated for the array

```
if (selected >= target - 0.000001 && selected <= target + 0.000001)
{
  return 0;
}
else if (selected > target + 0.000001)
{
  return 1;
}
else
{
  return -1;
}
```

```
[ RUN      ] binarySearch.unfinishedSize
Selected: 0
Result: -1
Selected: 5.6
Result: 0
BinarySearchTest.cpp:75: Failure
      Expected: false
To be equal to: contains(5.6, numbers, 7)
      Which is: true
[  FAILED  ] binarySearch.unfinishedSize (0 ms)
```