N8 SWALLEY 2/9/2022

SE463 – Spring 2022 Assignment #1

Boundary Value Analysis

Introduction

The first programming assignment in SE461 involved the design and implementation of a standard array class for characters, i.e., char types. Students were provided with a skeleton of an array class (.h file) and were challenged to complete the correct functionality for each method. Disclaimer: the use of any Standard Template Library (STL) or Standard C Library routines to implement the Array class were prohibited. This document outlines the evaluation of each function from Assignment #1 using Boundary Value Analysis and its four techniques: Normal, Robust, Worst-case, and Robust worst-case.

Array Class

```
/// Pointer to the actual data.

char * data_;

/// Current size of the array.

size_t cur_size_;

/// Maximum size of the array.

size_t max_size_;
```

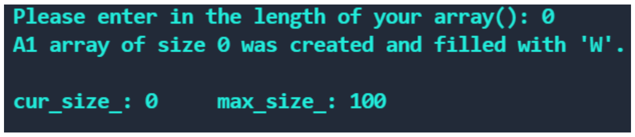
For testing purposes, the current size of the array has been set to 10 and the maximum size has been set to 100. The array has been filled with the character type 'W'. This allows us to use the four different techniques of boundary value analysis at the given boundaries.

Normal Boundary Value Analysis:

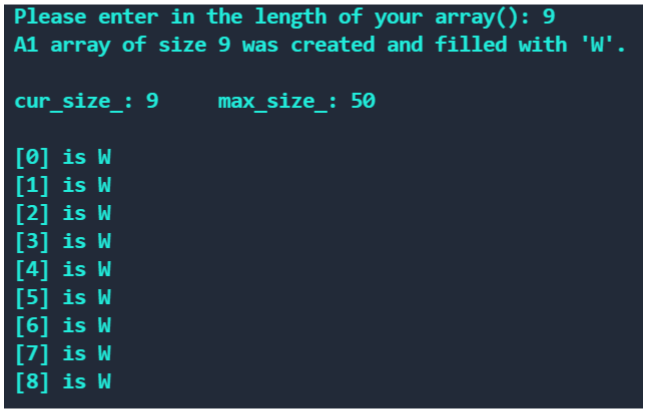
Boundary Value testing uses a “single fault” assumption which implies that faults are due to an incorrect value of a single variable. In this specific technique, each test case represents a variable being tested at its minimum, one value above the minimum, maximum, and one value below the maximum boundaries. In addition, there is one test case using nominal values for each variable. In total, there were 4n+1, or 13 test cases.

Cases:	Cur_size	Max_size	Data_	Outcome
1	0	50	[cur_size]	Valid
2	1	50	[cur_size]	Valid
3	5	50	[cur_size]	Valid
4	9	50	[cur_size]	Valid
5	10	50	[cur_size]	Valid
6	5	0	[cur_size]	Valid
7	5	1	[cur_size]	Valid
8	5	99	[cur_size]	Valid
9	5	100	[cur_size]	Valid
10	5	50	[0]	Valid
11	5	50	[1]	Valid
12	5	50	[cur_size-1]	Valid
13	5	50	[cur_size]	Valid

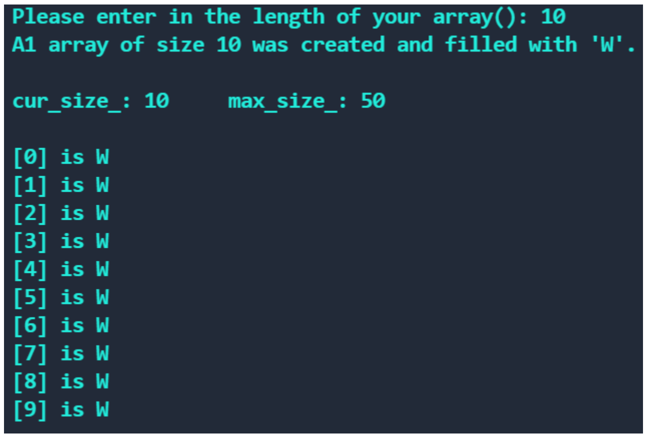
The following snippets demonstrate functionality of each test case highlighted in blue.



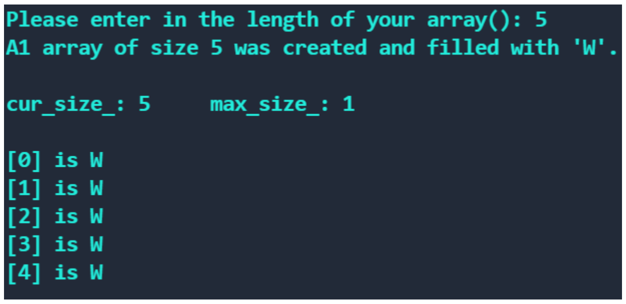
```
Please enter in the length of your array(): 0
A1 array of size 0 was created and filled with 'W'.

cur_size_: 0     max_size_: 100
```



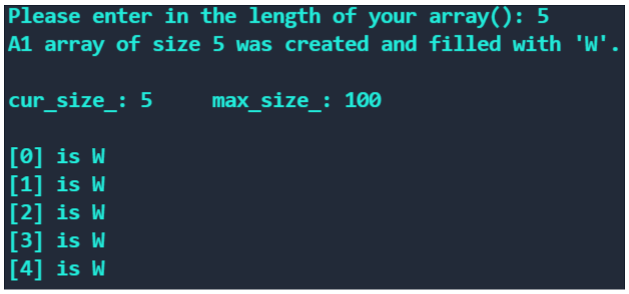
```
Please enter in the length of your array(): 9
A1 array of size 9 was created and filled with 'W'.

cur_size_: 9     max_size_: 50

[0] is W
[1] is W
[2] is W
[3] is W
[4] is W
[5] is W
[6] is W
[7] is W
[8] is W
```



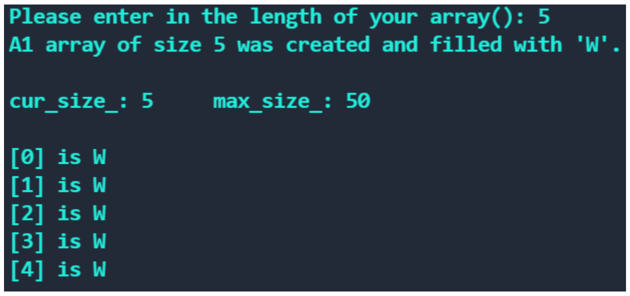
```
Please enter in the length of your array(): 10
A1 array of size 10 was created and filled with 'W'.

cur_size_: 10     max_size_: 50

[0] is W
[1] is W
[2] is W
[3] is W
[4] is W
[5] is W
[6] is W
[7] is W
[8] is W
[9] is W
```



```
Please enter in the length of your array(): 5
A1 array of size 5 was created and filled with 'W'.

cur_size_: 5     max_size_: 1

[0] is W
[1] is W
[2] is W
[3] is W
[4] is W
```



```
Please enter in the length of your array(): 5
A1 array of size 5 was created and filled with 'W'.

cur_size_: 5     max_size_: 100

[0] is W
[1] is W
[2] is W
[3] is W
[4] is W
```



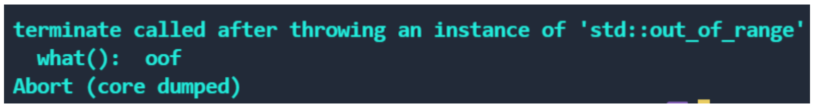
```
Please enter in the length of your array(): 5
A1 array of size 5 was created and filled with 'W'.

cur_size_: 5     max_size_: 50

[0] is W
[1] is W
[2] is W
[3] is W
[4] is W
```

All expected outcomes were reported successfully by Normal Boundary Value Testing.

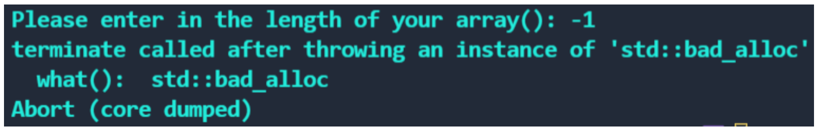
Robust Boundary Value Analysis:

In Robust Value Testing, input values are given just outside the set boundaries to test for invalid cases. The total number of test cases is the same as before, only with six extra cases for each variables invalid boundaries.

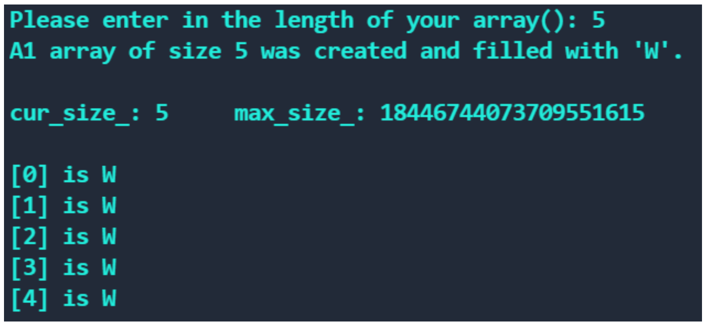
Cases:	Cur_size	Max_size	Data_	Outcome
1	0	50	[cur_size]	Valid
2	1	50	[cur_size]	Valid
3	5	50	[cur_size]	Valid
4	9	50	[cur_size]	Valid
5	10	50	[cur_size]	Valid
6	5	0	[cur_size]	Valid
7	5	1	[cur_size]	Valid
8	5	99	[cur_size]	Valid
9	5	100	[cur_size]	Valid
10	5	50	[0]	Valid
11	5	50	[1]	Valid
12	5	50	[cur_size-1]	Valid
13	5	50	[cur_size]	Valid
14	11	50	[cur_size]	Invalid
15	-1	50	[cur_size]	Invalid
16	5	-1	[cur_size]	Invalid
17	5	101	[cur_size]	Invalid
18	5	50	[cur_size+1]	Invalid
19	5	50	[-1]	Invalid



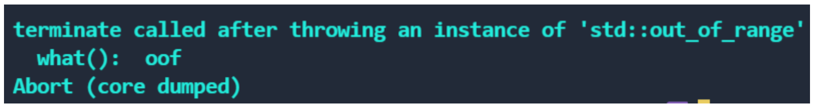
```
terminate called after throwing an instance of 'std::out_of_range'
  what():  oof
Abort (core dumped)
```



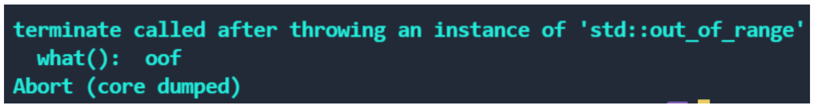
```
Please enter in the length of your array(): -1
terminate called after throwing an instance of 'std::bad_alloc'
  what():  std::bad_alloc
Abort (core dumped)
```



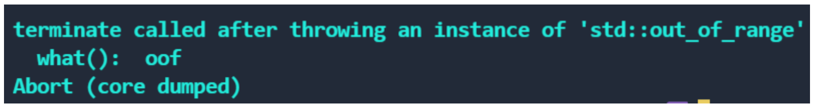
```
Please enter in the length of your array(): 5
A1 array of size 5 was created and filled with 'W'.

cur_size_: 5     max_size_: 18446744073709551615

[0] is W
[1] is W
[2] is W
[3] is W
[4] is W
```



```
terminate called after throwing an instance of 'std::out_of_range'
  what():  oof
Abort (core dumped)
```



```
terminate called after throwing an instance of 'std::out_of_range'
  what():  oof
Abort (core dumped)
```



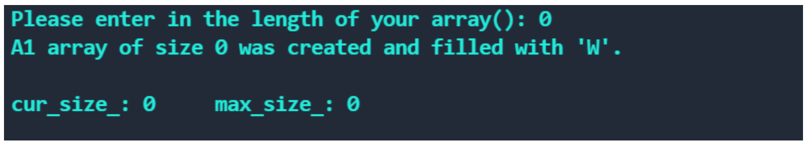
```
terminate called after throwing an instance of 'std::out_of_range'
  what():  oof
Abort (core dumped)
```

All expected outcomes were reported successfully by Robust Boundary Value Testing.

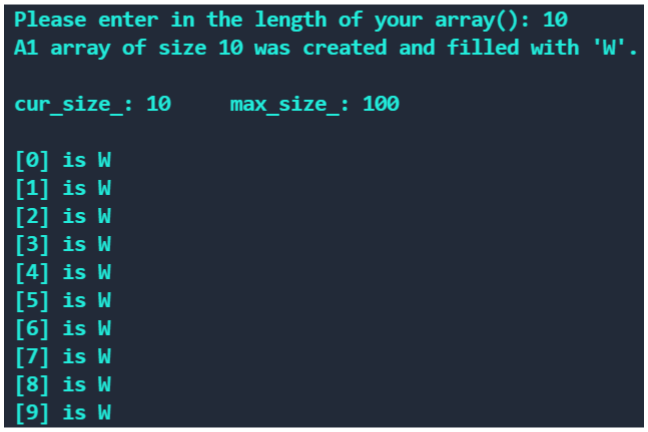
Worst-case Boundary Value Analysis:

In Worst-case Boundary Value Analysis, the total number of test cases is equal to 5^n, or 125. This is a "black box" testing technique that makes all combinations of each value of one variable with each value of another variable. To demonstrate the boundary tests, I focused on 4 specific cases: When Cur_size and Max_size were both near their minimum and maximum boundaries, when Cur_size was near the minimum boundary while Max_size was near the maximum boundary, and when Cur_size was near the maximum boundary while Max_size was near the minimum boundary.

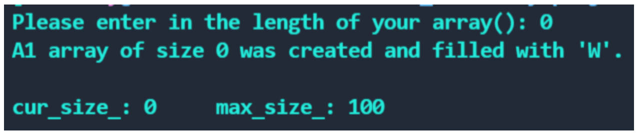
Cases:	Cur_size	Max_size	Data_	Outcome
1	0	0	[cur_size]	Valid
2	10	100	[cur_size]	Valid
3	0	100	[cur_size]	Valid
4	10	0	[cur_size]	Valid



```
Please enter in the length of your array(): 0
A1 array of size 0 was created and filled with 'W'.

cur_size_: 0     max_size_: 0
```



```
Please enter in the length of your array(): 10
A1 array of size 10 was created and filled with 'W'.

cur_size_: 10     max_size_: 100

[0] is W
[1] is W
[2] is W
[3] is W
[4] is W
[5] is W
[6] is W
[7] is W
[8] is W
[9] is W
```



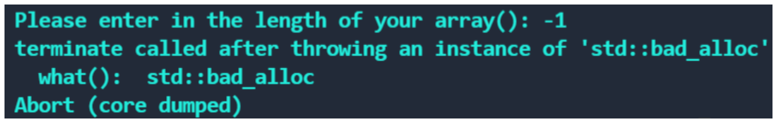
```
Please enter in the length of your array(): 0
A1 array of size 0 was created and filled with 'W'.

cur_size_: 0     max_size_: 100
```

All expected outcomes were reported successfully by Worst-case Boundary Value Testing.

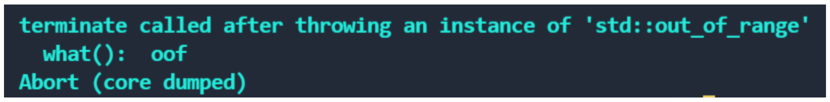
Robust Worst-case Boundary Value Analysis:

In Robust Worst-case Boundary Value Analysis, the total number of test cases is equal to 7^n, or 343. This technique is essentially combining all other techniques. To demonstrate the boundary tests, I modeled my test cases after the previous example except stretching the inputs beyond the acceptable boundaries.

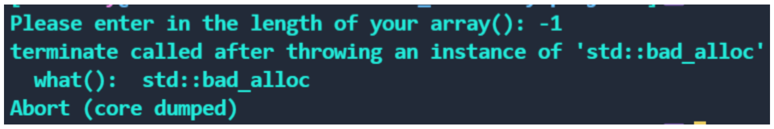
Cases:	Cur_size	Max_size	Data_	Outcome
1	-1	-1	[cur_size+1]	Invalid
2	11	101	[cur_size+1]	Invalid
3	-1	101	[cur_size+1]	Invalid
4	11	-1	[cur_size+1]	Invalid



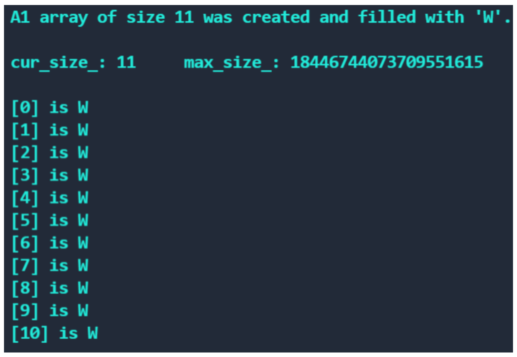
```
Please enter in the length of your array(): -1
terminate called after throwing an instance of 'std::bad_alloc'
  what():  std::bad_alloc
Abort (core dumped)
```



```
terminate called after throwing an instance of 'std::out_of_range'
  what():  oof
Abort (core dumped)
```



```
Please enter in the length of your array(): -1
terminate called after throwing an instance of 'std::bad_alloc'
  what():  std::bad_alloc
Abort (core dumped)
```



```
A1 array of size 11 was created and filled with 'W'.

cur_size_: 11     max_size_: 18446744073709551615

[0] is W
[1] is W
[2] is W
[3] is W
[4] is W
[5] is W
[6] is W
[7] is W
[8] is W
[9] is W
[10] is W
```

All expected outcomes were reported successfully by Robust Worst-case Boundary Value Testing.