Array-Based Stack Push Operation

Assume that we have the following lines of code:

```
Stack<int> stack1;  // Line 1

stack1.push(5);     // Line 2
stack1.push(8);     // Line 3
stack1.push(3);     // Line 4
stack1.push(6);     // Line 5
stack1.push(2);     // Line 6
```

The following sequence of diagrams shows how the `Stack` object and its associated dynamic storage changes as these lines are executed.

Figure 1: The new, empty `Stack` object `stack1` created in Line 1 of the code above. The `stk_array` pointer is `nullptr`, while `stk_size` and `stk_capacity` are both 0.

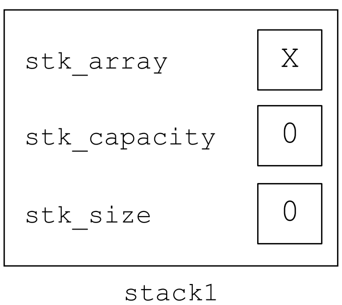


Figure 2: The `Stack` object following the execution of Line 2. Since `stkSize == stk_capacity`, the `push()` method will call the `reserve()` method to allocate a new dynamic array. Since the current `stk_capacity` is 0, the capacity requested for the new array will be 1. The contents of the existing array (if any) are copied to the new array (in this case, there's nothing to copy). The `stk_capacity` is updated to the capacity of the new array. The existing array is then deleted (in this case, there's nothing to delete) and the `stk_array` pointer is set to point to the new array. Finally, the value to insert is stored in the array at subscript `stkSize` (subscript 0) and then the `stk_size` is incremented to 1.

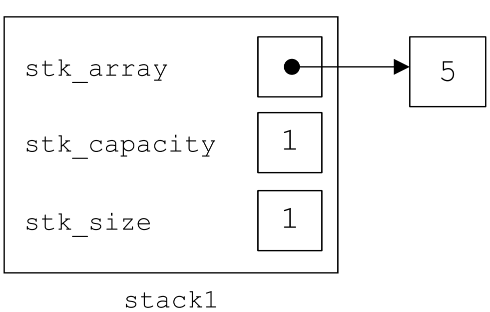


Figure 3: The `Stack` object following the execution of Line 3. Since `stk_size == stk_capacity`, the `push()` method will call the `reserve()` method to allocate a new dynamic array. Since the current `stk_capacity` is not 0, the capacity requested for the new array will be 2 (two times the current capacity of 1). The contents of the existing array are copied to the new array. The `stk_capacity` is updated to the capacity of the new array. The existing array is then deleted and the `stk_array` pointer is set to point to the new array. Finally, the value to insert is stored in the array at subscript `stk_size` (subscript 1) and then the `stk_size` is incremented to 2.

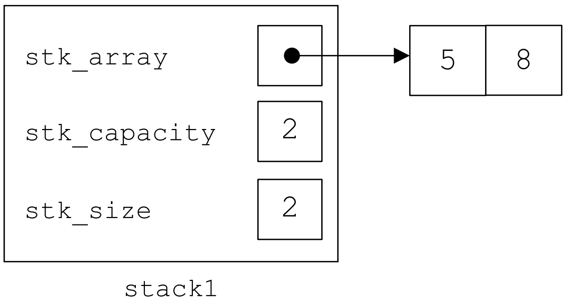


Figure 4: The `Stack` object following the execution of Line 4. Since `stk_size == stk_capacity`, the `push()` method will call the `reserve()` method to allocate a new dynamic array. Since the current `stk_capacity` is not 0, the capacity requested for the new array will be 4 (two times the current capacity of 2). The contents of the existing array are copied to the new array. The `stk_capacity` is updated to the capacity of the new array. The existing array is then deleted and the `stk_array` pointer is set to point to the new array. Finally, the value to insert is stored in the array at subscript `stk_size` (subscript 2) and then the `stk_size` is incremented to 3.

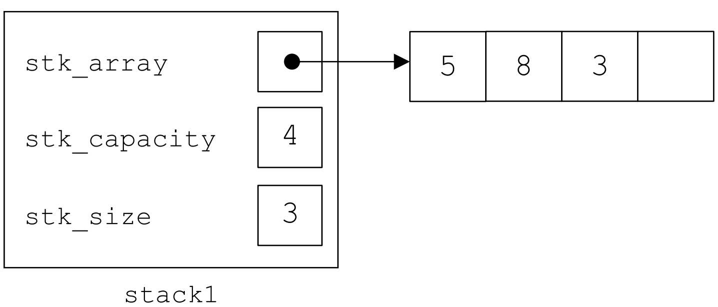


Figure 5: The `Stack` object following the execution of Line 5. Since `stk_size != stk_capacity`, the `push()` method does not call the `reserve()` method. The value to insert is simply stored in the array at subscript `stk_size` (subscript 3) and then the `stk_size` is incremented to 4.

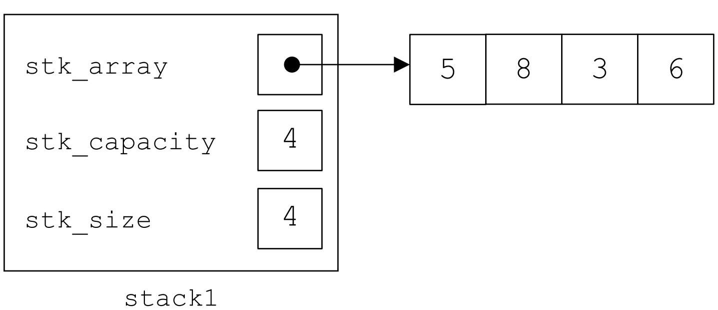
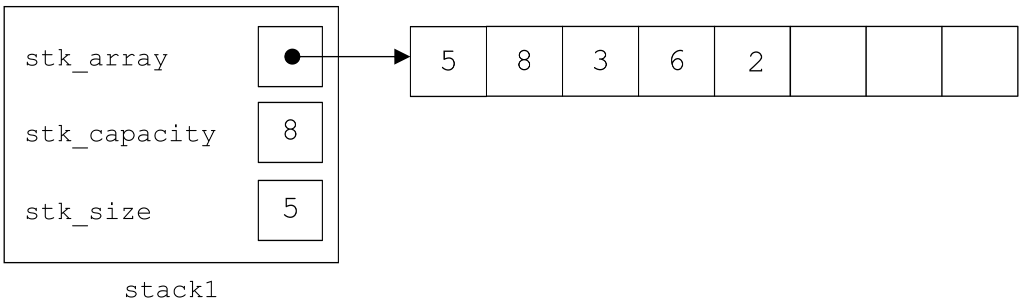


Figure 6: The `Stack` object following the execution of Line 6. Since `stk_size == stk_capacity`, the `push()` method will call the `reserve()` method to allocate a new dynamic array. Since the current `stk_capacity` is not 0, the capacity requested for the new array will be 8 (two times the current capacity of 4). The contents of the existing array are copied to the new array. The `stk_capacity` is updated to the capacity of the new array. The existing array is then deleted and the `stk_array` pointer is set to point to the new array. Finally, the value to insert is stored in the array at subscript `stk_size` (subscript 4) and then the `stk_size` is incremented to 5.



Array-Based Stack Pop Operation

Assume that we then add the following lines of code after the code listed above:

```
stack1.pop();        // Line 7
stack1.pop();        // Line 8
stack1.pop();        // Line 9
```

The following sequence of diagrams shows how the `Stack` object and its associated dynamic storage changes as these lines are executed.

Figure 7: The `Stack` object following the execution of Line 7. The `stk_size` is decremented to 4. That means that element 3 (the value 6) is now the top item in the stack, and element 4 (the value 2) is outside the boundaries of the stack. Effectively, it has been removed from the stack even though the value is technically still present in the array.

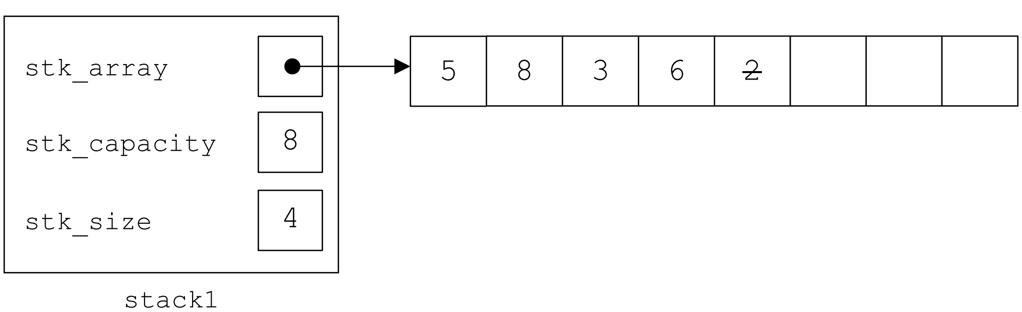


Figure 8: The `Stack` object following the execution of Line 8. The `stk_size` is decremented to 3. That means that element 2 (the value 3) is now the top item in the stack, and element 3 (the value 6) is now outside the boundaries of the stack.

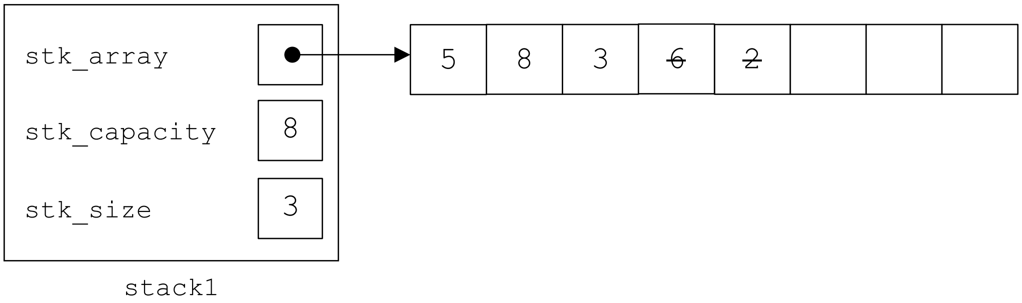
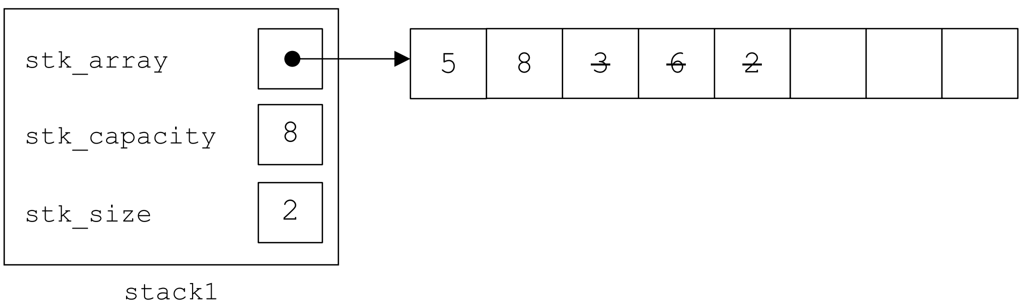


Figure 9: The `Stack` object following the execution of Line 9. The `stk_size` is decremented to 2. That means that element 1 (the value 8) is now the top item in the stack, and element 2 (the value 3) is now outside the boundaries of the stack.

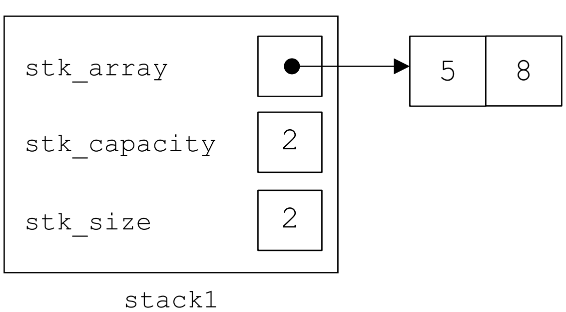


Note that the `pop()` method (or at least the version outlined in the notes) does **not** change the stack capacity.

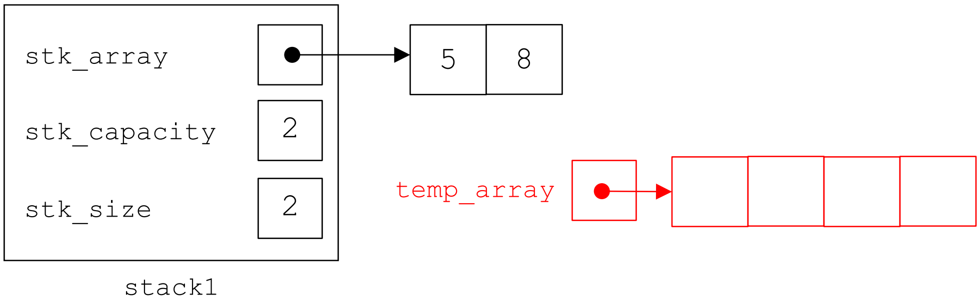
The `reserve()` Method

The following sequence of diagrams illustrate how the `reserve()` method works.

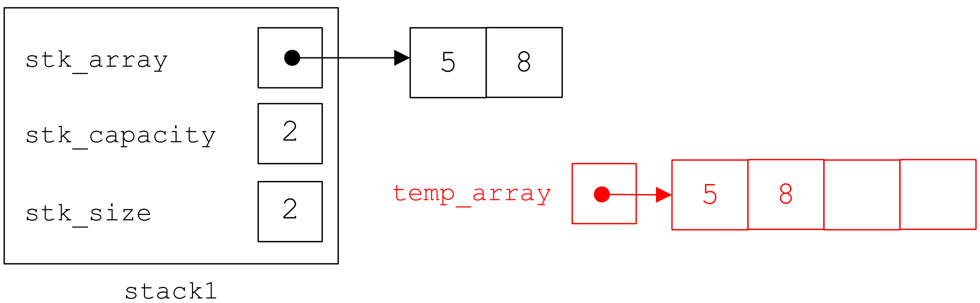
1. When `push()` is called and the `stk_size` is equal to the `stk_capacity` (i.e., the dynamic array is full, the reserve() method is called to allocate additional space to accommodate the new array element.



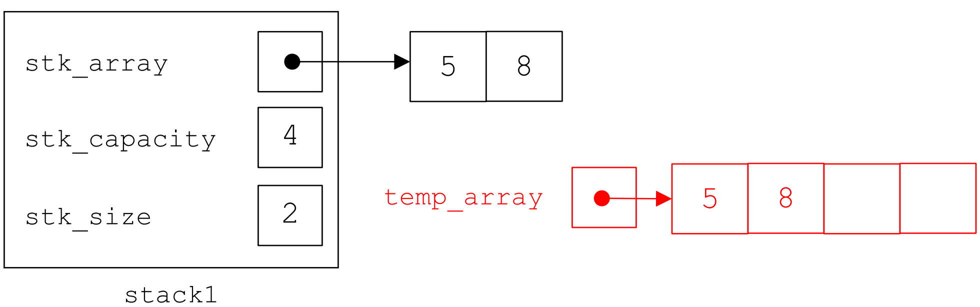
2. A temporary pointer (`temp_array`) is declared and used to allocate a new array with the requested capacity (in this case, a capacity of 4 has been requested).



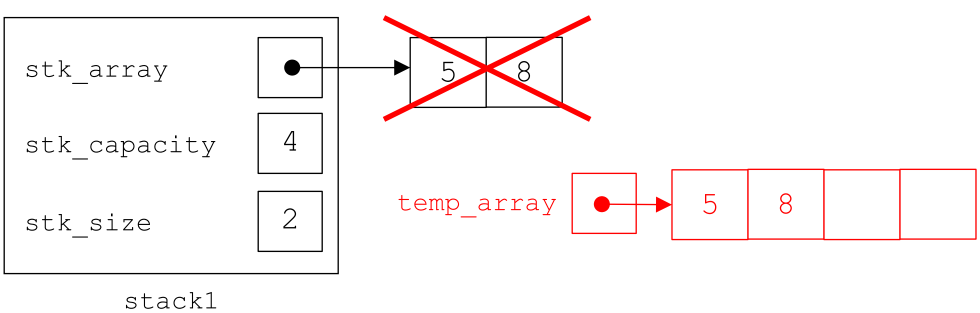
3. The contents of the existing array (if any) are copied into the new array.

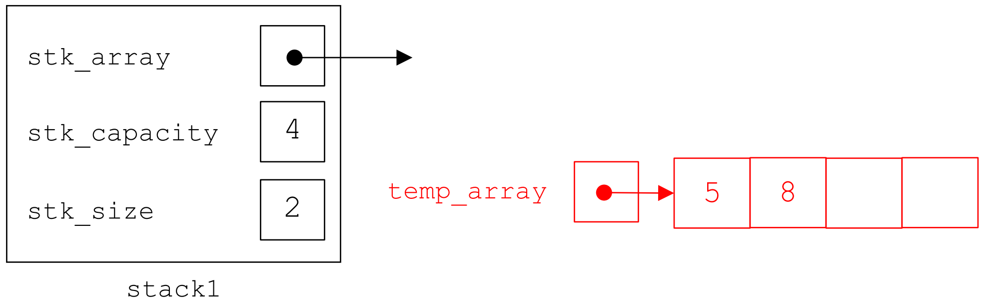


4. The stack capacity is updated to reflect the capacity of the new array.

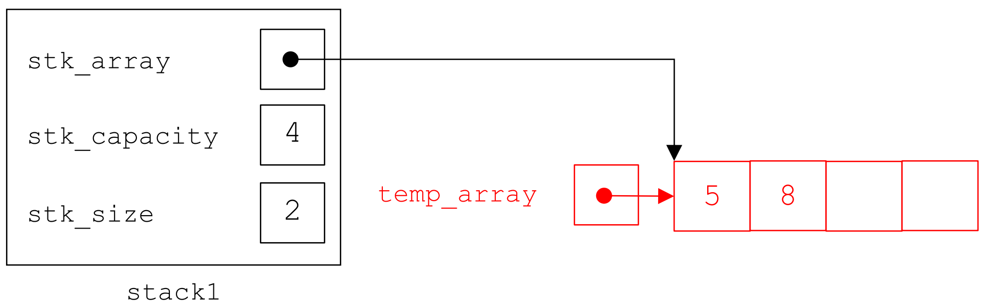


5. The existing array is deleted.

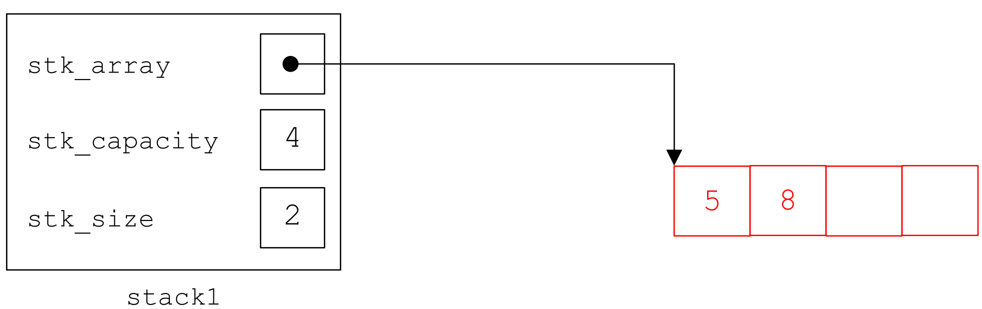




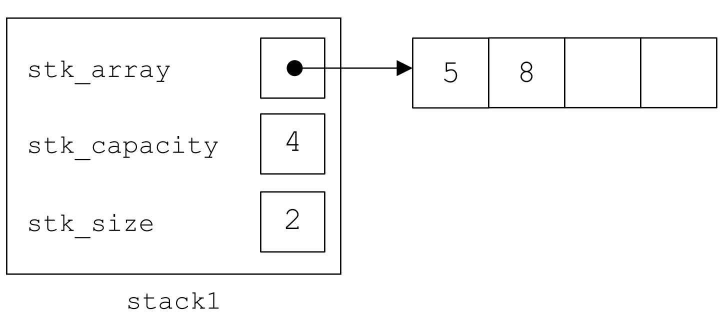
6. The address of the new array is copied into the pointer `stk_array`.



7. When the `reserve()` method ends, the temporary pointer `temp_array` ceases to exist (since it's a local variable).



The result is that the reserve() method has effectively increased the size of the stack array, providing enough room for the push() method to insert a new value.



8. The `push()` method can now insert the new value into the array at subscript `stk_size` and then increment `stk_size`.