

stack.h (Page 1 of 1)

```
1: /*-----*/
2: /* stack.h (Version 3) */
3: /* Author: Bob Dondero */
4: /*-----*/
5:
6: #ifndef STACK_INCLUDED
7: #define STACK_INCLUDED
8:
9: /* A Stack_T object is a last-in-first-out collection of doubles. */
10:
11: typedef struct Stack *Stack_T;
12:
13: /*-----*/
14:
15: /* Return a new Stack_T object, or NULL if insufficient memory is
16:    available. */
17:
18: Stack_T Stack_new(void);
19:
20: /*-----*/
21:
22: /* Free oStack. */
23:
24: void Stack_free(Stack_T oStack);
25:
26: /*-----*/
27:
28: /* Push dItem onto oStack. Return 1 (TRUE) if successful, or 0
29:    (FALSE) if insufficient memory is available. */
30:
31: int Stack_push(Stack_T oStack, double dItem);
32:
33: /*-----*/
34:
35: /* Pop and return the top item of oStack. */
36:
37: double Stack_pop(Stack_T oStack);
38:
39: /*-----*/
40:
41: /* Return 1 (TRUE) if oStack is empty, or 0 (FALSE) otherwise. */
42:
43: int Stack_isEmpty(Stack_T oStack);
44:
45: #endif
```

COS 217 Precept 15
Week 9, Mon/Tue

①

stack.c (Page 1 of 2)

```

1:  /*-----*/
2:  /* stack.c (Version 3) */
3:  /* Author: Bob Dondero */
4:  /*-----*/
5:
6:  #include "stack.h"
7:  #include <stdlib.h>
8:  #include <assert.h>
9:
10: /*-----*/
11:
12: /* A Stack object consists of an array of items, and related data. */
13: struct Stack
14: {
15:     /* The array in which items are stored. */
16:     double *pdArray;
17:
18:     /* The index one beyond the top element. */
19:     size_t uTop;
20:
21:     /* The number of elements in the array. */
22:     size_t uPhysLength;
23: };
24:
25: /*-----*/
26:
27: /* Double the physical length of oStack. Return 1 (TRUE) if
28:    successful, or 0 (FALSE) if insufficient memory is available. */
29:
30: static int Stack_grow(Stack_T oStack)
31: {
32:     const size_t GROWTH_FACTOR = 2;
33:
34:     size_t uNewPhysLength;
35:     double *pdNewArray;
36:
37:     assert(oStack != NULL);
38:
39:     uNewPhysLength = GROWTH_FACTOR * oStack->uPhysLength;
40:     pdNewArray = (double*)
41:         realloc(oStack->pdArray, sizeof(double) * uNewPhysLength);
42:     if (pdNewArray == NULL)
43:         return 0;
44:     oStack->uPhysLength = uNewPhysLength;
45:     oStack->pdArray = pdNewArray;
46:
47:     return 1;
48: }
49:
50: /*-----*/
51:
52: Stack_T Stack_new(void)
53: {
54:     const size_t INITIAL_PHYS_LENGTH = 2;
55:
56:     Stack_T oStack;
57:
58:     oStack = (Stack_T)malloc(sizeof(struct Stack));
59:     if (oStack == NULL)
60:         return NULL;
61:
62:     oStack->pdArray =
63:         (double*)calloc(INITIAL_PHYS_LENGTH, sizeof(double));

```

stack.c (Page 2 of 2)

```

64:     if (oStack->pdArray == NULL)
65:     {
66:         free(oStack);
67:         return NULL;
68:     }
69:
70:     oStack->uTop = 0;
71:     oStack->uPhysLength = INITIAL_PHYS_LENGTH;
72:     return oStack;
73: }
74:
75: /*-----*/
76:
77: void Stack_free(Stack_T oStack)
78: {
79:     assert(oStack != NULL);
80:     free(oStack->pdArray);
81:     free(oStack);
82: }
83:
84: /*-----*/
85:
86: int Stack_push(Stack_T oStack, double dItem)
87: {
88:     assert(oStack != NULL);
89:     if (oStack->uTop == oStack->uPhysLength)
90:         if (! Stack_grow(oStack))
91:             return 0;
92:     (oStack->pdArray)[oStack->uTop] = dItem;
93:     (oStack->uTop)++;
94:     return 1;
95: }
96:
97: /*-----*/
98:
99: double Stack_pop(Stack_T oStack)
100: {
101:     assert(oStack != NULL);
102:     assert(oStack->uTop > 0);
103:     (oStack->uTop)--;
104:     return (oStack->pdArray)[oStack->uTop];
105: }
106:
107: /*-----*/
108:
109: int Stack_isEmpty(Stack_T oStack)
110: {
111:     assert(oStack != NULL);
112:     return oStack->uTop == 0;
113: }

```

teststack.c (Page 1 of 2)

```

1: /*-----*/
2: /* teststack.c (Version 3) */
3: /* Author: Bob Dondero */
4: /*-----*/
5:
6: #include "stack.h"
7: #include <stdio.h>
8: #include <stdlib.h>
9:
10: /*-----*/
11:
12: /* Write an error message to stderr indicating that not enough memory
13:  is available. Then exit with status EXIT_FAILURE. */
14:
15: static void handleMemoryError(void)
16: {
17:     fprintf(stderr, "Insufficient memory\n");
18:     exit(EXIT_FAILURE);
19: }
20:
21: /*-----*/
22:
23: /* Test the Stack ADT. Return 0, or EXIT_FAILURE if not enough memory
24:  is available. */
25:
26: int main(void)
27: {
28:     Stack_T oStack1;
29:     Stack_T oStack2;
30:     int iSuccessful;
31:
32:     /* Use a Stack object referenced by oStack1. */
33:
34:     oStack1 = Stack_new();
35:     if (oStack1 == NULL) handleMemoryError();
36:
37:     iSuccessful = Stack_push(oStack1, 1.1);
38:     if (! iSuccessful) handleMemoryError();
39:
40:     iSuccessful = Stack_push(oStack1, 2.2);
41:     if (! iSuccessful) handleMemoryError();
42:
43:     iSuccessful = Stack_push(oStack1, 3.3);
44:     if (! iSuccessful) handleMemoryError();
45:
46:     while (! Stack_isEmpty(oStack1))
47:         printf("%g\n", Stack_pop(oStack1));
48:
49:     Stack_free(oStack1);
50:
51:     /* Use a Stack object referenced by oStack2. */
52:
53:     oStack2 = Stack_new();
54:     if (oStack2 == NULL) handleMemoryError();
55:
56:     iSuccessful = Stack_push(oStack2, 4.4);
57:     if (! iSuccessful) handleMemoryError();
58:
59:     iSuccessful = Stack_push(oStack2, 5.5);
60:     if (! iSuccessful) handleMemoryError();
61:
62:     iSuccessful = Stack_push(oStack2, 6.6);
63:     if (! iSuccessful) handleMemoryError();

```

teststack.c (Page 2 of 2)

```

64:
65:     while (! Stack_isEmpty(oStack2))
66:         printf("%g\n", Stack_pop(oStack2));
67:
68:     Stack_free(oStack2);
69:
70:     return 0;
71: }
72:
73: /*-----*/
74:
75: /*
76:  Output:
77:  3.3
78:  2.2
79:  1.1
80:  6.6
81:  5.5
82:  4.4
83: */

```

(w)



stackao.h (Page 1 of 1)

```
1: /*-----*/
2: /* stackao.h */
3: /* Author: Bob Dondero */
4: /*-----*/
5:
6: #ifndef STACKAO_INCLUDED
7: #define STACKAO_INCLUDED
8:
9: /* The Stack object is a last-in-first-out collection of doubles. */
10:
11: /*-----*/
12:
13: /* Initialize the Stack object. Return 1 (TRUE) if successful, or
14:    0 (FALSE) if insufficient memory is available. */
15:
16: int Stack_init(void);
17:
18: /*-----*/
19:
20: /* Free the resources consumed by the Stack object, and uninitialized
21:    it. */
22:
23: void Stack_free(void);
24:
25: /*-----*/
26:
27: /* Push dItem onto the Stack object. Return 1 (TRUE) if successful,
28:    or 0 (FALSE) if insufficient memory is available. */
29:
30: int Stack_push(double dItem);
31:
32: /*-----*/
33:
34: /* Pop and return the top item of the Stack object. */
35:
36: double Stack_pop(void);
37:
38: /*-----*/
39:
40: /* Return 1 (TRUE) if the Stack object is empty, or 0 (FALSE)
41:    otherwise. */
42:
43: int Stack_isEmpty(void);
44:
45: #endif
```

stackao.c (Page 1 of 2)

```

1: /*-----*/
2: /* stackao.c */
3: /* Author: Bob Dondero */
4: /*-----*/
5:
6: #include "stackao.h"
7: #include <stdlib.h>
8: #include <assert.h>
9:
10: /*-----*/
11:
12: /* In lieu of a boolean data type. */
13: enum {FALSE, TRUE};
14:
15: /*-----*/
16:
17: /* The state of the Stack object. */
18:
19: /* The array in which items are stored. */
20: static double *pdArray;
21:
22: /* The index one beyond the top element. */
23: static size_t uTop;
24:
25: /* The number of elements in the array. */
26: static size_t uPhysLength;
27:
28: /* Is the Stack initialized? */
29: static int iInitialized = FALSE;
30:
31: /*-----*/
32:
33: /* Increase the physical length of the Stack object. Return 1 (TRUE) if
34:    successful, or 0 (FALSE) if insufficient memory is available. */
35:
36: static int Stack_grow(void)
37: {
38:     const size_t GROWTH_FACTOR = 2;
39:
40:     size_t uNewPhysLength;
41:     double *pdNewArray;
42:
43:     assert(iInitialized);
44:
45:     uNewPhysLength = GROWTH_FACTOR * uPhysLength;
46:     pdNewArray = (double*)
47:         realloc(pdArray, sizeof(double) * uNewPhysLength);
48:     if (pdNewArray == NULL)
49:         return 0;
50:
51:     uPhysLength = uNewPhysLength;
52:     pdArray = pdNewArray;
53:
54:     return 1;
55: }
56:
57: /*-----*/
58:
59: int Stack_init(void)
60: {
61:     const size_t INITIAL_PHYS_LENGTH = 2;
62:
63:     assert(! iInitialized);

```

stackao.c (Page 2 of 2)

```

64:
65:     pdArray = (double*)calloc(INITIAL_PHYS_LENGTH, sizeof(double));
66:     if (pdArray == NULL)
67:         return 0;
68:
69:     uTop = 0;
70:     uPhysLength = INITIAL_PHYS_LENGTH;
71:     iInitialized = TRUE;
72:     return 1;
73: }
74:
75: /*-----*/
76:
77: void Stack_free(void)
78: {
79:     assert(iInitialized);
80:     free(pdArray);
81:     pdArray = NULL;
82:     uPhysLength = 0;
83:     uTop = 0;
84:     iInitialized = FALSE;
85: }
86:
87: /*-----*/
88:
89: int Stack_push(double dItem)
90: {
91:     assert(iInitialized);
92:     if (uTop == uPhysLength)
93:         if (! Stack_grow())
94:             return 0;
95:
96:     pdArray[uTop] = dItem;
97:     uTop++;
98:     return 1;
99: }
100:
101: /*-----*/
102:
103: double Stack_pop(void)
104: {
105:     assert(iInitialized);
106:     assert(uTop > 0);
107:     uTop--;
108:     return pdArray[uTop];
109: }
110:
111: /*-----*/
112:
113: int Stack_isEmpty(void)
114: {
115:     assert(iInitialized);
116:     return uTop == 0;
117: }

```

teststackao.c (Page 1 of 2)

```

1:  /*-----*/
2:  /* teststackao.c */
3:  /* Author: Bob Dondero */
4:  /*-----*/
5:
6:  #include "stackao.h"
7:  #include <stdio.h>
8:  #include <stdlib.h>
9:
10: /*-----*/
11:
12: /* Write an error message to stderr indicating that not enough memory
13:    is available. Then exit with status EXIT_FAILURE. */
14:
15: static void handleMemoryError(void)
16: {
17:     fprintf(stderr, "Insufficient memory\n");
18:     exit(EXIT_FAILURE);
19: }
20:
21: /*-----*/
22:
23: /* Test the Stack abstract object. Return 0, or EXIT_FAILURE if not
24:    enough memory is available. */
25:
26: int main(void)
27: {
28:     int iSuccessful;
29:
30:     iSuccessful = Stack_init();
31:     if (! iSuccessful) handleMemoryError();
32:
33:     iSuccessful = Stack_push(1.1);
34:     if (! iSuccessful) handleMemoryError();
35:
36:     iSuccessful = Stack_push(2.2);
37:     if (! iSuccessful) handleMemoryError();
38:
39:     iSuccessful = Stack_push(3.3);
40:     if (! iSuccessful) handleMemoryError();
41:
42:     while (! Stack_isEmpty())
43:         printf("%g\n", Stack_pop());
44:
45:     Stack_free();
46:
47:     iSuccessful = Stack_init();
48:     if (! iSuccessful) handleMemoryError();
49:
50:     iSuccessful = Stack_push(4.4);
51:     if (! iSuccessful) handleMemoryError();
52:
53:     iSuccessful = Stack_push(5.5);
54:     if (! iSuccessful) handleMemoryError();
55:
56:     iSuccessful = Stack_push(6.6);
57:     if (! iSuccessful) handleMemoryError();
58:
59:     while (! Stack_isEmpty())
60:         printf("%g\n", Stack_pop());
61:
62:     Stack_free();
63:

```

teststackao.c (Page 2 of 2)

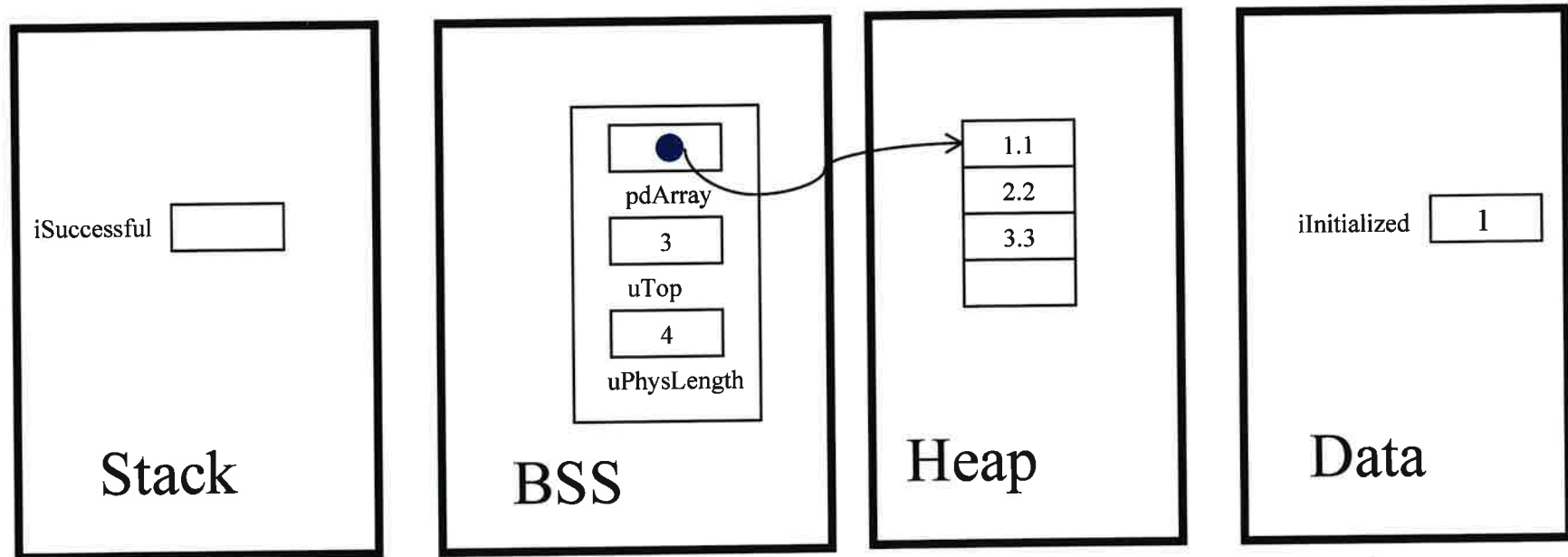
```

64:     return 0;
65: }
66:
67: /*-----*/
68:
69: /*
70:    Output:
71:    3.3
72:    2.2
73:    1.1
74:    6.6
75:    5.5
76:    4.4
77: */

```

Princeton University
COS 217: Introduction to Programming Systems
Trace of teststackao

```
int iSuccessful;  
iSuccessful = Stack_init();  
iSuccessful = Stack_push(1.1);  
iSuccessful = Stack_push(2.2);  
iSuccessful = Stack_push(3.3);
```





stackaochecker.h (Page 1 of 1)

```
1: /*-----*/
2: /* stackaochecker.h */
3: /* Author: Christopher Moretti */
4: /*-----*/
5:
6: #ifndef STACKAOCHECKER_INCLUDED
7: #define STACKAOCHECKER_INCLUDED
8:
9: #include <stddef.h>
10:
11: /* validate the Stackao object, based on internal state values
12:    pdArray (the storage array), uTop (the location of the next push),
13:    uPhysLength (the allocated length of pdArray), and iInitialized
14:    (a flag indicating the Stack is in the initialized state).
15:    Returns 1 if valid, 0 if invalid.
16:    Prints error description to stderr if invalid.
17: */
18:
19: int isValid(double *pdArray, size_t uTop, size_t uPhysLength,
20:            int iInitialized);
21:
22: #endif
```

stackaochecker.c (Page 1 of 1)

```
1: /*-----*/
2: /* stackaochecker.c */
3: /* Author: Christopher Moretti */
4: /*-----*/
5:
6: #include "stackaochecker.h"
7: #include <stdio.h>
8:
9: int isValid(double *pdArray, size_t uTop, size_t uPhysLength,
10:            int iInitialized)
11: {
12:     if(!iInitialized &&
13:        (pdArray != NULL || uTop != 0 || uPhysLength != 0))
14:     {
15:         fprintf(stderr, "Not initialized, but state is not NULL/0\n");
16:         return 0;
17:     }
18:
19:     if(iInitialized &&
20:        (pdArray == NULL || uPhysLength == 0))
21:     {
22:         fprintf(stderr, "Initialized, but storage array is not ready\n");
23:         return 0;
24:     }
25:
26:     if(uTop > uPhysLength)
27:     {
28:         fprintf(stderr, "Next push would be OOB of storage array\n");
29:         return 0;
30:     }
31:
32:     return 1;
33: }
```


stackaochecked.c (Page 1 of 2)

```

1: /*-----*/
2: /* stackao.c (version with isValid checks) */
3: /* Author: Bob Dondero, Christopher Moretti */
4: /*-----*/
5:
6: #include "stackao.h"
7: #include "stackaochecker.h"
8: #include <stdlib.h>
9: #include <assert.h>
10:
11: /*-----*/
12:
13: /* In lieu of a boolean data type. */
14: enum {FALSE, TRUE};
15:
16: /*-----*/
17:
18: /* The state of the Stack object. */
19:
20: /* The array in which items are stored. */
21: static double *pdArray;
22:
23: /* The index one beyond the top element. */
24: static size_t uTop;
25:
26: /* The number of elements in the array. */
27: static size_t uPhysLength;
28:
29: /* Is the Stack initialized? */
30: static int iInitialized = FALSE;
31:
32: /*-----*/
33:
34: /* Increase the physical length of the Stack object. Return 1 (TRUE) if
35:    successful, or 0 (FALSE) if insufficient memory is available. */
36:
37: static int Stack_grow(void)
38: {
39:     const size_t GROWTH_FACTOR = 2;
40:
41:     size_t uNewPhysLength;
42:     double *pdNewArray;
43:
44:     assert(iInitialized);
45:
46:     uNewPhysLength = GROWTH_FACTOR * uPhysLength;
47:     pdNewArray = (double*)
48:         realloc(pdArray, sizeof(double) * uNewPhysLength);
49:     if (pdNewArray == NULL)
50:         return 0;
51:
52:     uPhysLength = uNewPhysLength;
53:     pdArray = pdNewArray;
54:
55:     return 1;
56: }
57:
58: /*-----*/
59:
60: int Stack_init(void)
61: {
62:     const size_t INITIAL_PHYS_LENGTH = 2;
63:
64:     assert(! iInitialized);
65:

```

stackaochecked.c (Page 2 of 2)

```

66:     pdArray = (double*)calloc(INITIAL_PHYS_LENGTH, sizeof(double));
67:     if (pdArray == NULL) {
68:         assert(isValid(pdArray, uTop, uPhysLength, iInitialized));
69:         return 0;
70:     }
71:
72:     uTop = 0;
73:     uPhysLength = INITIAL_PHYS_LENGTH;
74:     iInitialized = TRUE;
75:
76:     assert(isValid(pdArray, uTop, uPhysLength, iInitialized));
77:     return 1;
78: }
79:
80: /*-----*/
81:
82: void Stack_free(void)
83: {
84:     assert(iInitialized);
85:     free(pdArray);
86:     pdArray = NULL;
87:     uPhysLength = 0;
88:     uTop = 0;
89:     iInitialized = FALSE;
90:     assert(isValid(pdArray, uTop, uPhysLength, iInitialized));
91: }
92:
93: /*-----*/
94:
95: int Stack_push(double dItem)
96: {
97:     assert(iInitialized);
98:     if (uTop == uPhysLength)
99:         if (! Stack_grow()) {
100:             assert(isValid(pdArray, uTop, uPhysLength, iInitialized));
101:             return 0;
102:         }
103:
104:     pdArray[uTop] = dItem;
105:     uTop++;
106:
107:     assert(isValid(pdArray, uTop, uPhysLength, iInitialized));
108:     return 1;
109: }
110:
111: /*-----*/
112:
113: double Stack_pop(void)
114: {
115:     assert(iInitialized);
116:     assert(uTop > 0);
117:     uTop--;
118:
119:     assert(isValid(pdArray, uTop, uPhysLength, iInitialized));
120:     return pdArray[uTop];
121: }
122:
123: /*-----*/
124:
125: int Stack_isEmpty(void)
126: {
127:     assert(iInitialized);
128:     return uTop == 0;
129: }

```

stackaocheckedbad.c (Page 1 of 2)

```

1: /*-----*/
2: /* stackaocheckedbad.c (version with isValid checks and a bug) */
3: /* Author: Bob Dondero, Christopher Moretti, Donna Gabai */
4: /*-----*/
5:
6: #include "stackao.h"
7: #include "stackaochecker.h"
8: #include <stdlib.h>
9: #include <assert.h>
10:
11: /*-----*/
12:
13: /* In lieu of a boolean data type. */
14: enum {FALSE, TRUE};
15:
16: /*-----*/
17:
18: /* The state of the Stack object. */
19:
20: /* The array in which items are stored. */
21: static double *pdArray;
22:
23: /* The index one beyond the top element. */
24: static size_t uTop;
25:
26: /* The number of elements in the array. */
27: static size_t uPhysLength;
28:
29: /* Is the Stack initialized? */
30: static int iInitialized = FALSE;
31:
32: /*-----*/
33:
34: /* Increase the physical length of the Stack object. Return 1 (TRUE) if
35:    successful, or 0 (FALSE) if insufficient memory is available. */
36:
37: static int Stack_grow(void)
38: {
39:     const size_t GROWTH_FACTOR = 2;
40:
41:     size_t uNewPhysLength;
42:     double *pdNewArray;
43:
44:     assert(iInitialized);
45:
46:     uNewPhysLength = GROWTH_FACTOR * uPhysLength;
47:     pdNewArray = (double*)
48:         realloc(pdArray, sizeof(double) * uNewPhysLength);
49:     if (pdNewArray == NULL)
50:         return 0;
51:
52:     uPhysLength = uNewPhysLength;
53:     pdArray = pdNewArray;
54:
55:     return 1;
56: }
57:
58: /*-----*/
59:
60: int Stack_init(void)
61: {
62:     const size_t INITIAL_PHYS_LENGTH = 2;
63:
64:     assert(! iInitialized);
65:

```

stackaocheckedbad.c (Page 2 of 2)

```

66:     pdArray = (double*)calloc(INITIAL_PHYS_LENGTH, sizeof(double));
67:     if (pdArray == NULL) {
68:         assert(isValid(pdArray, uTop, uPhysLength, iInitialized));
69:         return 0;
70:     }
71:
72:     uTop = 0;
73:     uPhysLength = INITIAL_PHYS_LENGTH;
74:     iInitialized = TRUE;
75:
76:     assert(isValid(pdArray, uTop, uPhysLength, iInitialized));
77:     return 1;
78: }
79:
80: /*-----*/
81:
82: void Stack_free(void)
83: {
84:     assert(iInitialized);
85:     free(pdArray);
86:     /*pdArray = NULL;*/
87:     uPhysLength = 0;
88:     uTop = 0;
89:     iInitialized = FALSE;
90:     assert(isValid(pdArray, uTop, uPhysLength, iInitialized));
91: }
92:
93: /*-----*/
94:
95: int Stack_push(double dItem)
96: {
97:     assert(iInitialized);
98:     if (uTop > uPhysLength)
99:         if (! Stack_grow()) {
100:             assert(isValid(pdArray, uTop, uPhysLength, iInitialized));
101:             return 0;
102:         }
103:
104:     pdArray[uTop] = dItem;
105:     uTop++;
106:
107:     assert(isValid(pdArray, uTop, uPhysLength, iInitialized));
108:     return 1;
109: }
110:
111: /*-----*/
112:
113: double Stack_pop(void)
114: {
115:     assert(iInitialized);
116:     assert(uTop > 0);
117:     uTop--;
118:
119:     assert(isValid(pdArray, uTop, uPhysLength, iInitialized));
120:     return pdArray[uTop];
121: }
122:
123: /*-----*/
124:
125: int Stack_isEmpty(void)
126: {
127:     assert(iInitialized);
128:     return uTop == 0;
129: }

```



Precept Activity Instructions

Work in small groups:

- 1) If you were to design a validation module for `Stackao`, what invariants would you have to check? (Invariants are things that you know must be true. That is, they cannot vary or something is wrong. In the context of a validation module, invariants are constraints on values of individual state variables or relationships between state variables.)
- 2) Review the structure of the validation module (`stackaochecker.h` and `stackaochecker.c`). Are all the invariants from part 1) present?
- 3) Review how the validation module is called from the implementation code in a version with hooks to do so (`stackaochecked.c`) – notice how validation happens on every return path from every mutator function.
- 4) Log into armlab and copy all files into your directory on armlab:
`cp -r /u/cos217/PreceptDemos/stackao .`
Build and run the correct (`stackaochecked.c`) and buggy (`stackaocheckedbad.c`) hooked versions. Compare the outputs. Find the bug in `stackaocheckedbad.c`.

Precept Activity Answers

1) The following cases indicate the state of the data structure is invalid:

Case 1. `iInitialized` is 0, but `pdArray` is not NULL, or `uTop` is not 0, or `uPhysLength` is not 0.

Case 2. `iInitialized` is 1, but `pdArray` is NULL, or `uPhysLength` is 0.

Case 3. `uTop` > `uPhysLength`.

2) They should be.

3) The checker function is called before every return statement in every non-static function that changes the state of the abstract object.

4) `stackaochecked` output:

```
3.3
2.2
1.1
6.6
5.5
4.4
```

`stackaocheckedbad` output:

```
Next push would be OOB of storage array
teststackbad: stackaocheckedbad.c:107: Stack_push: Assertion
`isValid(pdArray, uTop, uPhysLength, iInitialized)' failed.
Aborted (core dumped)
```

Why do we get this error message?

Because, in `Stack_push()` the condition for calling the `Stack_grow()` function is incorrect. Should be `if (uTop == uPhysicalLength)`