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
```

COS217 Precept 15
Week 9, Mon/Tue

stack.c (Page 1 of 2)

```
/*--------------------------------------------------------------------*/
/* stack.c (Version 3)                                                */
/* Author: Bob Dondero                                                */
/*--------------------------------------------------------------------*/

#include "stack.h"
#include <stdlib.h>
#include <assert.h>

/*--------------------------------------------------------------------*/

/* A Stack object consists of an array of items, and related data. */
struct Stack
{
   /* The array in which items are stored. */
   double *pdArray;

   /* The index one beyond the top element. */
   size_t uTop;

   /* The number of elements in the array. */
   size_t uPhysLength;
};

/*--------------------------------------------------------------------*/

/* Double the physical length of oStack.  Return 1 (TRUE) if
   successful, or 0 (FALSE) if insufficient memory is available. */

static int Stack_grow(Stack_T oStack)
{
   const size_t GROWTH_FACTOR = 2;

   size_t uNewPhysLength;
   double *pdNewArray;

   assert(oStack != NULL);

   uNewPhysLength = GROWTH_FACTOR * oStack->uPhysLength;
   pdNewArray = (double*)
      realloc(oStack->pdArray, sizeof(double) * uNewPhysLength);
   if (pdNewArray == NULL)
      return 0;
   oStack->uPhysLength = uNewPhysLength;
   oStack->pdArray = pdNewArray;

   return 1;
}

/*--------------------------------------------------------------------*/

Stack_T Stack_new(void)
{
   const size_t INITIAL_PHYS_LENGTH = 2;

   Stack_T oStack;

   oStack = (Stack_T)malloc(sizeof(struct Stack));
   if (oStack == NULL)
      return NULL;

   oStack->pdArray =
      (double*)calloc(INITIAL_PHYS_LENGTH, sizeof(double));
```

stack.c (Page 2 of 2)

```
   if (oStack->pdArray == NULL)
   {
      free(oStack);
      return NULL;
   }

   oStack->uTop = 0;
   oStack->uPhysLength = INITIAL_PHYS_LENGTH;
   return oStack;
}

/*--------------------------------------------------------------------*/

void Stack_free(Stack_T oStack)
{
   assert(oStack != NULL);
   free(oStack->pdArray);
   free(oStack);
}

/*--------------------------------------------------------------------*/

int Stack_push(Stack_T oStack, double dItem)
{
   assert(oStack != NULL);
   if (oStack->uTop == oStack->uPhysLength)
      if (! Stack_grow(oStack))
         return 0;
   (oStack->pdArray)[oStack->uTop] = dItem;
   (oStack->uTop)++;
   return 1;
}

/*--------------------------------------------------------------------*/

double Stack_pop(Stack_T oStack)
{
   assert(oStack != NULL);
   assert(oStack->uTop > 0);
   (oStack->uTop)--;
   return (oStack->pdArray)[oStack->uTop];
}

/*--------------------------------------------------------------------*/

int Stack_isEmpty(Stack_T oStack)
{
   assert(oStack != NULL);
   return oStack->uTop == 0;
}
```

teststack.c (Page 1 of 2)

```
/*--------------------------------------------------------------------*/
/* teststack.c (Version 3)                                            */
/* Author: Bob Dondero                                                */
/*--------------------------------------------------------------------*/

#include "stack.h"
#include <stdio.h>
#include <stdlib.h>

/*--------------------------------------------------------------------*/

/* Write an error message to stderr indicating that not enough memory
   is available.  Then exit with status EXIT_FAILURE. */

static void handleMemoryError(void)
{
   fprintf(stderr, "Insufficient memory\n");
   exit(EXIT_FAILURE);
}

/*--------------------------------------------------------------------*/

/* Test the Stack ADT.  Return 0, or EXIT_FAILURE if not enough memory
   is available. */

int main(void)
{
   Stack_T oStack1;
   Stack_T oStack2;
   int iSuccessful;

   /* Use a Stack object referenced by oStack1. */

   oStack1 = Stack_new();
   if (oStack1 == NULL) handleMemoryError();

   iSuccessful = Stack_push(oStack1, 1.1);
   if (! iSuccessful) handleMemoryError();

   iSuccessful = Stack_push(oStack1, 2.2);
   if (! iSuccessful) handleMemoryError();

   iSuccessful = Stack_push(oStack1, 3.3);
   if (! iSuccessful) handleMemoryError();

   while (! Stack_isEmpty(oStack1))
      printf("%g\n", Stack_pop(oStack1));

   Stack_free(oStack1);

   /* Use a Stack object referenced by oStack2. */

   oStack2 = Stack_new();
   if (oStack2 == NULL) handleMemoryError();

   iSuccessful = Stack_push(oStack2, 4.4);
   if (! iSuccessful) handleMemoryError();

   iSuccessful = Stack_push(oStack2, 5.5);
   if (! iSuccessful) handleMemoryError();

   iSuccessful = Stack_push(oStack2, 6.6);
   if (! iSuccessful) handleMemoryError();
```

teststack.c (Page 2 of 2)

```

   while (! Stack_isEmpty(oStack2))
      printf("%g\n", Stack_pop(oStack2));

   Stack_free(oStack2);

   return 0;
}

/*--------------------------------------------------------------------*/

/*
   Output:
   3.3
   2.2
   1.1
   6.6
   5.5
   4.4
*/
```

```
/*--------------------------------------------------------------------------*/
/* stackao.h                                                                */
/* Author: Bob Dondero                                                      */
/*--------------------------------------------------------------------------*/

#ifndef STACKAO_INCLUDED
#define STACKAO_INCLUDED

/* The Stack object is a last-in-first-out collection of doubles. */

/*--------------------------------------------------------------------------*/

/* Initialize the Stack object.  Return 1 (TRUE) if successful, or
   0 (FALSE) if insufficient memory is available. */

int Stack_init(void);

/*--------------------------------------------------------------------------*/

/* Free the resources consumed by the Stack object, and uninitialize
   it. */

void Stack_free(void);

/*--------------------------------------------------------------------------*/

/* Push dItem onto the Stack object.  Return 1 (TRUE) if successful,
   or 0 (FALSE) if insufficient memory is available. */

int Stack_push(double dItem);

/*--------------------------------------------------------------------------*/

/* Pop and return the top item of the Stack object. */

double Stack_pop(void);

/*--------------------------------------------------------------------------*/

/* Return 1 (TRUE) if the Stack object is empty, or 0 (FALSE)
   otherwise. */

int Stack_isEmpty(void);

#endif
```

stackao.c (Page 1 of 2)

```
/*--------------------------------------------------------------------*/
/* stackao.c                                                          */
/* Author: Bob Dondero                                                */
/*--------------------------------------------------------------------*/

#include "stackao.h"
#include <stdlib.h>
#include <assert.h>

/*--------------------------------------------------------------------*/

/* In lieu of a boolean data type. */
enum {FALSE, TRUE};

/*--------------------------------------------------------------------*/

/* The state of the Stack object. */

/* The array in which items are stored. */
static double *pdArray;

/* The index one beyond the top element. */
static size_t uTop;

/* The number of elements in the array. */
static size_t uPhysLength;

/* Is the Stack initialized? */
static int iInitialized = FALSE;

/*--------------------------------------------------------------------*/

/* Increase the physical length of the Stack object.  Return 1 (TRUE) if
   successful, or 0 (FALSE) if insufficient memory is available. */

static int Stack_grow(void)
{
   const size_t GROWTH_FACTOR = 2;

   size_t uNewPhysLength;
   double *pdNewArray;

   assert(iInitialized);

   uNewPhysLength = GROWTH_FACTOR * uPhysLength;
   pdNewArray = (double*)
      realloc(pdArray, sizeof(double) * uNewPhysLength);
   if (pdNewArray == NULL)
      return 0;

   uPhysLength = uNewPhysLength;
   pdArray = pdNewArray;

   return 1;
}

/*--------------------------------------------------------------------*/

int Stack_init(void)
{
   const size_t INITIAL_PHYS_LENGTH = 2;

   assert(! iInitialized);
```

stackao.c (Page 2 of 2)

```

   pdArray = (double*)calloc(INITIAL_PHYS_LENGTH, sizeof(double));
   if (pdArray == NULL)
      return 0;

   uTop = 0;
   uPhysLength = INITIAL_PHYS_LENGTH;
   iInitialized = TRUE;
   return 1;
}

/*--------------------------------------------------------------------*/

void Stack_free(void)
{
   assert(iInitialized);
   free(pdArray);
   pdArray = NULL;
   uPhysLength = 0;
   uTop = 0;
   iInitialized = FALSE;
}

/*--------------------------------------------------------------------*/

int Stack_push(double dItem)
{
   assert(iInitialized);
   if (uTop == uPhysLength)
      if (! Stack_grow())
         return 0;

   pdArray[uTop] = dItem;
   uTop++;
   return 1;
}

/*--------------------------------------------------------------------*/

double Stack_pop(void)
{
   assert(iInitialized);
   assert(uTop > 0);
   uTop--;
   return pdArray[uTop];
}

/*--------------------------------------------------------------------*/

int Stack_isEmpty(void)
{
   assert(iInitialized);
   return uTop == 0;
}
```

teststackao.c (Page 1 of 2)

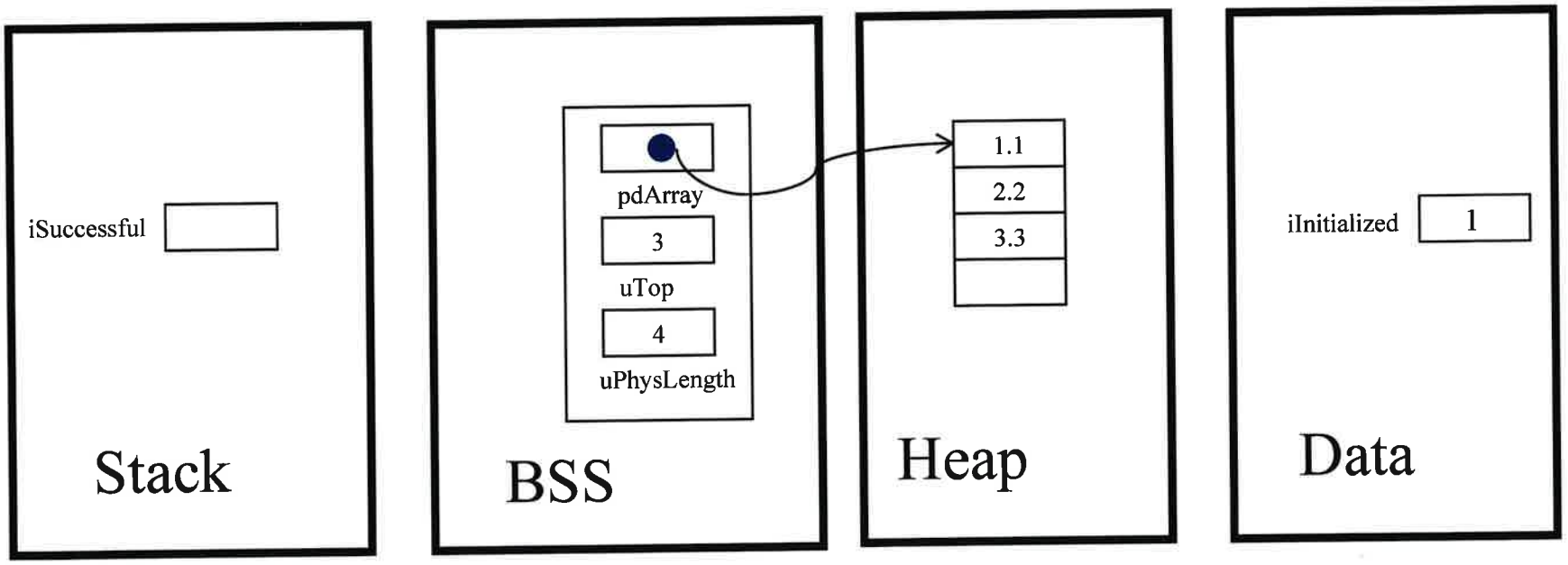
```
 1: /*--------------------------------------------------------------------*/
 2: /* teststackao.c                                                      */
 3: /* Author: Bob Dondero                                                */
 4: /*--------------------------------------------------------------------*/
 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)

```
/*--------------------------------------------------------------------*/
/* stackao.c (version with isValid checks)                            */
/* Author: Bob Dondero, Christopher Moretti                           */
/*--------------------------------------------------------------------*/

#include "stackao.h"
#include "stackaochecker.h"
#include <stdlib.h>
#include <assert.h>

/*--------------------------------------------------------------------*/

/* In lieu of a boolean data type. */
enum {FALSE, TRUE};

/*--------------------------------------------------------------------*/

/* The state of the Stack object. */

/* The array in which items are stored. */
static double *pdArray;

/* The index one beyond the top element. */
static size_t uTop;

/* The number of elements in the array. */
static size_t uPhysLength;

/* Is the Stack initialized? */
static int iInitialized = FALSE;

/*--------------------------------------------------------------------*/

/* Increase the physical length of the Stack object.  Return 1 (TRUE) if
   successful, or 0 (FALSE) if insufficient memory is available. */

static int Stack_grow(void)
{
   const size_t GROWTH_FACTOR = 2;

   size_t uNewPhysLength;
   double *pdNewArray;

   assert(iInitialized);

   uNewPhysLength = GROWTH_FACTOR * uPhysLength;
   pdNewArray = (double*)
      realloc(pdArray, sizeof(double) * uNewPhysLength);
   if (pdNewArray == NULL)
      return 0;

   uPhysLength = uNewPhysLength;
   pdArray = pdNewArray;

   return 1;
}

/*--------------------------------------------------------------------*/

int Stack_init(void)
{
   const size_t INITIAL_PHYS_LENGTH = 2;

   assert(! iInitialized);

```

stackaochecked.c (Page 2 of 2)

```
   pdArray = (double*)calloc(INITIAL_PHYS_LENGTH, sizeof(double));
   if (pdArray == NULL) {
      assert(isValid(pdArray, uTop, uPhysLength, iInitialized));
      return 0;
   }

   uTop = 0;
   uPhysLength = INITIAL_PHYS_LENGTH;
   iInitialized = TRUE;

   assert(isValid(pdArray, uTop, uPhysLength, iInitialized));
   return 1;
}

/*--------------------------------------------------------------------*/

void Stack_free(void)
{
   assert(iInitialized);
   free(pdArray);
   pdArray = NULL;
   uPhysLength = 0;
   uTop = 0;
   iInitialized = FALSE;
   assert(isValid(pdArray, uTop, uPhysLength, iInitialized));
}

/*--------------------------------------------------------------------*/

int Stack_push(double dItem)
{
   assert(iInitialized);
   if (uTop == uPhysLength)
      if (! Stack_grow()) {
         assert(isValid(pdArray, uTop, uPhysLength, iInitialized));
         return 0;
      }

   pdArray[uTop] = dItem;
   uTop++;

   assert(isValid(pdArray, uTop, uPhysLength, iInitialized));
   return 1;
}

/*--------------------------------------------------------------------*/

double Stack_pop(void)
{
   assert(iInitialized);
   assert(uTop > 0);
   uTop--;

   assert(isValid(pdArray, uTop, uPhysLength, iInitialized));
   return pdArray[uTop];
}

/*--------------------------------------------------------------------*/

int Stack_isEmpty(void)
{
   assert(iInitialized);
   return uTop == 0;
}
```

stackaocheckedbad.c (Page 1 of 2)

```
/*--------------------------------------------------------------------*/
/* stackaocheckedbad.c (version with isValid checks and a bug)        */
/* Author: Bob Dondero, Christopher Moretti, Donna Gabai              */
/*--------------------------------------------------------------------*/

#include "stackao.h"
#include "stackaochecker.h"
#include <stdlib.h>
#include <assert.h>

/*--------------------------------------------------------------------*/

/* In lieu of a boolean data type. */
enum {FALSE, TRUE};

/*--------------------------------------------------------------------*/

/* The state of the Stack object. */

/* The array in which items are stored. */
static double *pdArray;

/* The index one beyond the top element. */
static size_t uTop;

/* The number of elements in the array. */
static size_t uPhysLength;

/* Is the Stack initialized? */
static int iInitialized = FALSE;

/*--------------------------------------------------------------------*/

/* Increase the physical length of the Stack object.  Return 1 (TRUE) if
   successful, or 0 (FALSE) if insufficient memory is available. */

static int Stack_grow(void)
{
   const size_t GROWTH_FACTOR = 2;

   size_t uNewPhysLength;
   double *pdNewArray;

   assert(iInitialized);

   uNewPhysLength = GROWTH_FACTOR * uPhysLength;
   pdNewArray = (double*)
      realloc(pdArray, sizeof(double) * uNewPhysLength);
   if (pdNewArray == NULL)
      return 0;

   uPhysLength = uNewPhysLength;
   pdArray = pdNewArray;

   return 1;
}

/*--------------------------------------------------------------------*/

int Stack_init(void)
{
   const size_t INITIAL_PHYS_LENGTH = 2;

   assert(! iInitialized);

```

stackaocheckedbad.c (Page 2 of 2)

```
   pdArray = (double*)calloc(INITIAL_PHYS_LENGTH, sizeof(double));
   if (pdArray == NULL) {
      assert(isValid(pdArray, uTop, uPhysLength, iInitialized));
      return 0;
   }

   uTop = 0;
   uPhysLength = INITIAL_PHYS_LENGTH;
   iInitialized = TRUE;

   assert(isValid(pdArray, uTop, uPhysLength, iInitialized));
   return 1;
}

/*--------------------------------------------------------------------*/

void Stack_free(void)
{
   assert(iInitialized);
   free(pdArray);
   /*pdArray = NULL;*/
   uPhysLength = 0;
   uTop = 0;
   iInitialized = FALSE;
   assert(isValid(pdArray, uTop, uPhysLength, iInitialized));
}

/*--------------------------------------------------------------------*/

int Stack_push(double dItem)
{
   assert(iInitialized);
   if (uTop > uPhysLength)
      if (! Stack_grow()) {
         assert(isValid(pdArray, uTop, uPhysLength, iInitialized));
         return 0;
      }

   pdArray[uTop] = dItem;
   uTop++;

   assert(isValid(pdArray, uTop, uPhysLength, iInitialized));
   return 1;
}

/*--------------------------------------------------------------------*/

double Stack_pop(void)
{
   assert(iInitialized);
   assert(uTop > 0);
   uTop--;

   assert(isValid(pdArray, uTop, uPhysLength, iInitialized));
   return pdArray[uTop];
}

/*--------------------------------------------------------------------*/

int Stack_isEmpty(void)
{
   assert(iInitialized);
   return uTop == 0;
}
```

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)`