```c
// $Id: cbox.h,v 1.2 2014-02-27 17:46:13-08 - - $ //

#ifndef __CBOX_H__
#define __CBOX_H__

#include <stdbool.h>

// NAME
//    cbox ADT
//
// DESCRIPTION
//    A simple ADT that permits the holding of an integer in a box
//    similar to the way Java uses an `Integer' to box an `int'.

typedef struct cbox cbox;
   // Incomplete type defined in implementation file.

cbox *new_cbox (void);
   // Constructor: create a new `cbox' box initialized to 0.
   // Postcond:    new cbox box is returned.

cbox *new_int_cbox (int value);
   // Constructor: create a new `cbox' box initialized by caller.
   // Postcond:    new cbox box is returned.

void free_cbox (cbox *this);
   // Destructor: destroys an allocated box
   // Precond:    box created by new_cbox/1.
   // Postcond:   this pointer is dangling.

int get_cbox (cbox *this);
   // Accessor:   retrieves the integer from the box.
   // Precond:    valid handle to an cbox.
   // Postcond:   returns the value in the box.

void put_cbox (cbox *this, int newvalue);
   // Mutator:    replaces the integer in the box with a new one.
   // Precond:    valid handle to an cbox.
   // Postcond:   old value is lost, new value is kept

#endif

//
// Notes:
//
// File guards protect the file from multiple inclusion.
//
// A header file specifies only the prototypes for functions,
// similar to the way an interface does in Java.  Everything in the
// header file is `public'.
//
// Note that all function names are global and can not be
// overloaded.  So we name a function as in Java and suffix it with
// the last name of the `module' that it belongs to.  Note that in
// the standard C library, there are often common prefixes, such as
// `f-' for file-oriented functions, `str-' for string functions, etc.
//
```

```c
// $Id: cbox.c,v 1.1 2014-02-13 18:38:23-08 - - $

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "cbox.h"

struct cbox {
   int value;
};

cbox *new_cbox (void) {
   return new_int_cbox (0);
}

cbox *new_int_cbox (int value) {
   cbox *this = malloc (sizeof (struct cbox));
   assert (this != NULL);
   this->value = value;
   return this;
}

void free_cbox (cbox *this) {
   free (this);
}

int get_cbox (cbox *this) {
   return this->value;
}

void put_cbox (cbox *this, int newvalue) {
   this->value = newvalue;
}

//
// Notes that would normally not be put in the file:
//
// A '.c' file always includes its own header.
//
// The 'struct' definition itself is specified in the
// implementation file.  Everything declared in the implementation
// file is 'private'.  Never put field definitions in a header
// file.
//

```

```c
// $Id: main.c,v 1.12 2014-02-27 17:46:38-08 - - $

//
// Silly main program which just creates an cbox box, puts a
// number in it, gets it back out, and deletes the box.
// Run with bcheck to verify no memory leaks.
//

#include <errno.h>
#include <libgen.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "cbox.h"

char *execname = NULL;

int main (int argc, char **argv) {
   (void) argc; // warning: unused parameter 'argc'
   execname = basename (argv[0]);

   // Declare the box and initialize it.
   cbox *box = new_cbox();
   printf ("box = %p\n", box);

   // Perform a couple of operations on it.
   put_cbox (box, 1024);
   printf ("box value is %d\n", get_cbox (box));

   // Free up the box.
   free_cbox (box);

   return EXIT_SUCCESS;
}

```

```
# $Id: Makefile,v 1.12 2015-01-30 17:41:46-08 - - $

MKFILE    = Makefile
DEPSFILE  = ${MKFILE}.deps
NOINCLUDE = ci clean spotless
NEEDINCL  = ${filter ${NOINCLUDE}, ${MAKECMDGOALS}}

GCC      = gcc -g -O0 -Wall -Wextra -std=gnu99
MKDEPS   = gcc -MM
GRIND    = valgrind --leak-check=full

CHEADER   = cbox.h
CSOURCE   = cbox.c main.c
OBJECTS   = ${CSOURCE:.c=.o}
EXECBIN   = cbox
SOURCES   = ${CHEADER} ${CSOURCE} ${MKFILE}
LISTING   = Listing.cbox.ps

all : ${EXECBIN}

${EXECBIN} : ${OBJECTS}
        ${GCC} -o $@ ${OBJECTS}

%.o : %.c
        cid + $<
        ${GCC} -c $<

ci : ${SOURCES}
        cid + ${SOURCES}

lis : ${SOURCES} test
        mkpspdf ${LISTING} ${SOURCES} test.lis

clean :
        - rm ${OBJECTS} ${DEPSFILE} core test.lis

spotless : clean
        - rm ${EXECBIN} ${LISTING:.ps=.p*} test.lis

test : ${EXECBIN}
        ${GRIND} --log-file=test.log ${EXECBIN} >test.out 2>test.err
        more ${DEPSFILE} test.out test.err test.log >test.lis
        - rm test.out test.err test.log

deps : ${CSOURCE} ${CHEADER}
        @ echo "# ${DEPSFILE} created `date`" >${DEPSFILE}
        ${MKDEPS} ${CSOURCE} >>${DEPSFILE}

${DEPSFILE} :
        @ touch ${DEPSFILE}
        ${MAKE} --no-print-directory deps

ifeq ("${NEEDINCL}","")
include ${DEPSFILE}
endif

```

```
::::::::::::::
Makefile.deps
::::::::::::::
# Makefile.deps created Fri Jan 30 17:41:56 PST 2015
cbox.o: cbox.c cbox.h
main.o: main.c cbox.h
::::::::::::::
test.out
::::::::::::::
box = 0x51c3040
box value is 1024
::::::::::::::
test.err
::::::::::::::
::::::::::::::
test.log
::::::::::::::
==7199== Memcheck, a memory error detector
==7199== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==7199== Using Valgrind-3.9.0 and LibVEX; rerun with -h for copyright info
==7199== Command: cbox
==7199== Parent PID: 7198
==7199== 
==7199== 
==7199== HEAP SUMMARY:
==7199==     in use at exit: 0 bytes in 0 blocks
==7199==   total heap usage: 1 allocs, 1 frees, 4 bytes allocated
==7199== 
==7199== All heap blocks were freed -- no leaks are possible
==7199== 
==7199== For counts of detected and suppressed errors, rerun with: -v
==7199== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 6 from 6)
```