

```
1: // $Id: cbox.h,v 1.2 2014-02-27 17:46:13-08 - - $ //
2:
3: #ifndef __CBOX_H__
4: #define __CBOX_H__
5:
6: #include <stdbool.h>
7:
8: // NAME
9: //     cbox ADT
10: //
11: // DESCRIPTION
12: //     A simple ADT that permits the holding of an integer in a box
13: //     similar to the way Java uses an 'Integer' to box an 'int'.
14:
15: typedef struct cbox cbox;
16:     // Incomplete type defined in implementation file.
17:
18: cbox *new_cbox (void);
19:     // Constructor: create a new 'cbox' box initialized to 0.
20:     // Postcond:     new cbox box is returned.
21:
22: cbox *new_int_cbox (int value);
23:     // Constructor: create a new 'cbox' box initialized by caller.
24:     // Postcond:     new cbox box is returned.
25:
26: void free_cbox (cbox *this);
27:     // Destructor: destroys an allocated box
28:     // Precond:     box created by new_cbox/1.
29:     // Postcond:     this pointer is dangling.
30:
31: int get_cbox (cbox *this);
32:     // Accessor:     retrieves the integer from the box.
33:     // Precond:     valid handle to an cbox.
34:     // Postcond:     returns the value in the box.
35:
36: void put_cbox (cbox *this, int newvalue);
37:     // Mutator:     replaces the integer in the box with a new one.
38:     // Precond:     valid handle to an cbox.
39:     // Postcond:     old value is lost, new value is kept
40:
41: #endif
42:
43: //
44: // Notes:
45: //
46: // File guards protect the file from multiple inclusion.
47: //
48: // A header file specifies only the prototypes for functions,
49: // similar to the way an interface does in Java.  Everything in the
50: // header file is 'public'.
51: //
52: // Note that all function names are global and can not be
53: // overloaded.  So we name a function as in Java and suffix it with
54: // the last name of the 'module' that it belongs to.  Note that in
55: // the standard C library, there are often common prefixes, such as
56: // 'f-' for file-oriented functions, 'str-' for string functions, etc.
57: //
```

```
1: // $Id: cbox.c,v 1.1 2014-02-13 18:38:23-08 - - $
2:
3: #include <assert.h>
4: #include <stdio.h>
5: #include <stdlib.h>
6: #include <string.h>
7:
8: #include "cbox.h"
9:
10: struct cbox {
11:     int value;
12: };
13:
14: cbox *new_cbox (void) {
15:     return new_int_cbox (0);
16: }
17:
18: cbox *new_int_cbox (int value) {
19:     cbox *this = malloc (sizeof (struct cbox));
20:     assert (this != NULL);
21:     this->value = value;
22:     return this;
23: }
24:
25: void free_cbox (cbox *this) {
26:     free (this);
27: }
28:
29: int get_cbox (cbox *this) {
30:     return this->value;
31: }
32:
33: void put_cbox (cbox *this, int newvalue) {
34:     this->value = newvalue;
35: }
36:
37: //
38: // Notes that would normally not be put in the file:
39: //
40: // A '.c' file always includes its own header.
41: //
42: // The 'struct' definition itself is specified in the
43: // implementation file. Everything declared in the implementation
44: // file is 'private'. Never put field definitions in a header
45: // file.
46: //
47:
```

```
1: // $Id: main.c,v 1.12 2014-02-27 17:46:38-08 - - $
2:
3: //
4: // Silly main program which just creates an cbox box, puts a
5: // number in it, gets it back out, and deletes the box.
6: // Run with bcheck to verify no memory leaks.
7: //
8:
9: #include <errno.h>
10: #include <libgen.h>
11: #include <stdio.h>
12: #include <stdlib.h>
13: #include <string.h>
14:
15: #include "cbox.h"
16:
17: char *execname = NULL;
18:
19: int main (int argc, char **argv) {
20:     (void) argc; // warning: unused parameter 'argc'
21:     execname = basename (argv[0]);
22:
23:     // Declare the box and initialize it.
24:     cbox *box = new_cbox();
25:     printf ("box = %p\n", box);
26:
27:     // Perform a couple of operations on it.
28:     put_cbox (box, 1024);
29:     printf ("box value is %d\n", get_cbox (box));
30:
31:     // Free up the box.
32:     free_cbox (box);
33:
34:     return EXIT_SUCCESS;
35: }
36:
```

```
1: # $Id: Makefile,v 1.11 2014-02-13 18:53:02-08 - - $
2:
3: MKFILE      = Makefile
4: DEPSFILE    = ${MKFILE}.deps
5: NOINCLUDE   = ci clean spotless
6: NEEDINCL    = ${filter ${NOINCLUDE}, ${MAKECMDGOALS}}
7:
8: GCC          = gcc -g -O0 -Wall -Wextra -std=gnu99
9: MKDEPS      = gcc -MM
10: GRIND        = valgrind --leak-check=full
11:
12: CHEADER      = cbox.h
13: CSOURCE      = cbox.c main.c
14: OBJECTS      = ${CSOURCE:.c=.o}
15: EXECBIN      = cbox
16: SOURCES      = ${CHEADER} ${CSOURCE} ${MKFILE}
17: LISTING      = Listing.cbox.ps
18:
19: all : ${EXECBIN}
20:
21: ${EXECBIN} : ${OBJECTS}
22:             ${GCC} -o $@ ${OBJECTS}
23:
24: %.o : %.c
25:         cid + $<
26:         ${GCC} -c $<
27:
28: ci : ${SOURCES}
29:     cid + ${SOURCES}
30:
31: lis : ${SOURCES} test
32:     mkpspdf ${LISTING} ${SOURCES} test.lis
33:
34: clean :
35:     - rm ${OBJECTS} ${DEPSFILE} core test.lis
36:
37: spotless : clean
38:     - rm ${EXECBIN} ${LISTING:.ps=.p*} test.lis
39:
40: test : ${EXECBIN}
41:     ${GRIND} --log-file=test.log ${EXECBIN} >test.out 2>test.err
42:     more ${DEPSFILE} test.out test.err test.log >test.lis
43:     - rm test.out test.err test.log
44:
45: deps : ${CSOURCE} ${CHEADER}
46:     @ echo "# ${DEPSFILE} created `date`" >${DEPSFILE}
47:     ${MKDEPS} ${CSOURCE} >>${DEPSFILE}
48:
49: ${DEPSFILE} :
50:     @ touch ${DEPSFILE}
51:     ${MAKE} --no-print-directory deps
52:
53: ifeq ("${NEEDINCL}", "")
54: include ${DEPSFILE}
55: endif
56:
```

```
1: ::::::::::::::
2: Makefile.deps
3: ::::::::::::::
4: # Makefile.deps created Thu Feb 13 18:56:07 PST 2014
5: cbox.o: cbox.c cbox.h
6: main.o: main.c cbox.h
7: ::::::::::::::
8: test.out
9: ::::::::::::::
10: box = 0x4c28040
11: box value is 1024
12: ::::::::::::::
13: test.err
14: ::::::::::::::
15: ::::::::::::::
16: test.log
17: ::::::::::::::
18: ==12880== Memcheck, a memory error detector
19: ==12880== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al
.
20: ==12880== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright i
nfo
21: ==12880== Command: cbox
22: ==12880== Parent PID: 12879
23: ==12880==
24: ==12880==
25: ==12880== HEAP SUMMARY:
26: ==12880==      in use at exit: 0 bytes in 0 blocks
27: ==12880==    total heap usage: 1 allocs, 1 frees, 4 bytes allocated
28: ==12880==
29: ==12880== All heap blocks were freed -- no leaks are possible
30: ==12880==
31: ==12880== For counts of detected and suppressed errors, rerun with: -v
32: ==12880== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 6 from 6)
```