```
 1: /* $Id: intx.h,v 1.4 2012-02-15 20:47:04-08 - - $ */
 2:
 3: #ifndef __INTX_H__
 4: #define __INTX_H__
 5:
 6: #include <stdbool.h>
 7:
 8: /*
 9: * NAME
10: *    intx ADT
11: *
12: * DESCRIPTION
13: *    A simple ADT that permits the holding of an integer in a box
14: *    similar to the way Java uses an 'Integer' to box an 'int'.
15: */
16:
17: typedef struct intx *intx_ref;
18:    /*
19:    * The handle pointing at the 'intx' box.
20:    */
21:
22: intx_ref new_intx (void);
23:    /*
24:    * Constructor: create a new 'intx' box initialized to 0.
25:    * Postcond:    new intx box is returned.
26:    */
27:
28: intx_ref new1_intx (int initvalue);
29:    /*
30:    * Constructor: create a new 'intx' box initialized by caller.
31:    * Postcond:    new intx box is returned.
32:    */
33:
34: void free_intx (intx_ref this);
35:    /*
36:    * Destructor: destroys an allocated box
37:    * Precond:    box created by new_intx/1.
38:    * Postcond:   this pointer is dangling.
39:    */
40:
41: int get_intx (intx_ref this);
42:    /*
43:    * Accessor:   retrieves the integer from the box.
44:    * Precond:    valid handle to an intx.
45:    * Postcond:   returns the value in the box.
46:    */
47:
48: void put_intx (intx_ref this, int newvalue);
49:    /*
50:    * Mutator:    replaces the integer in the box with a new one.
51:    * Precond:    valid handle to an intx.
52:    * Postcond:   old value is lost, new value is kept
53:    */
54:
55: bool is_intx (intx_ref this);
56:    /*
57:    * Accessor:   check to see if this is really an intx.
58:    */
59:
60: #endif
```

```
61:
62: /*
63: ********************************************************************
64:
65: Notes:
66:
67: File guards protect the file from multiple inclusion.
68:
69: A header file specifies only the prototypes for functions,
70: similar to the way an interface does in Java.  Everything in the
71: header file is ‘public’.
72:
73: As a standard, the handle type is defined as a pointer to a
74: structure whose fields are secret.
75:
76: Note that all function names are global and can not be
77: overloaded.  So we name a function as in Java and suffix it with
78: the last name of the ‘module’ that it belongs to.  Note that in
79: the standard C library, there are often common prefixes, such as
80: ‘f-’ for file-oriented functions, ‘str-’ for string functions,
81: etc.
82:
83: ********************************************************************
84: */
```

```
 1: /* $Id: main.c,v 1.4 2012-02-14 20:49:59-08 - - $ */
 2:
 3: /*
 4: * Silly main program which just creates an intx 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: #include <sys/time.h>
15: #include <time.h>
16:
17: #include "intx.h"
18:
19: char *execname = NULL;
20:
21: void say_when (char *when) {
22:    struct timeval timeval;
23:    int retcode = gettimeofday (&timeval, NULL);
24:    if (retcode != 0) {
25:       fprintf (stderr, "%s: gettimeofday: %s\n",
26:                execname, strerror (errno));
27:    }
28:    struct tm *tm_buffer = localtime (&timeval.tv_sec);
29:    char buffer1[64];
30:    char buffer2[64];
31:    strftime (buffer1, sizeof buffer1, "%a %b %e %T", tm_buffer);
32:    strftime (buffer2, sizeof buffer2, " %Z %Y", tm_buffer);
33:    printf ("%s: %s: %s.%06ld %s\n", execname, when,
34:            buffer1, timeval.tv_usec, buffer2);
35: }
36:
37: int main (int argc, char **argv) {
38:    argc = argc; // Avoid:16: warning: unused parameter 'argc'
39:    execname = basename (argv[0]);
40:    say_when ("starting");
41:
42:    /* Declare the box and initialize it.  */
43:    intx_ref box = new_intx();
44:    printf ("box = %p\n", box);
45:
46:    /* Perform a couple of operations on it.  */
47:    put_intx (box, 1024);
48:    printf ("box value is %d\n", get_intx (box));
49:
50:    /* Free up the box and set the handle to NULL to avoid a dangle. */
51:    free_intx (box);
52:    box = NULL;
53:
54:    /* OK! */
55:    say_when ("finished");
56:    return EXIT_SUCCESS;
57: }
58:
```

```
 1: /* $Id: intx.c,v 1.3 2012-02-15 20:47:04-08 - - $ */
 2:
 3: #include <assert.h>
 4: #include <stdio.h>
 5: #include <stdlib.h>
 6: #include <string.h>
 7:
 8: #include "intx.h"
 9:
10: static char *intx_tag = "struct intx";
11:
12: struct intx {
13:    char *tag;
14:    int value;
15: };
16:
17: intx_ref new_intx (void) {
18:    return new1_intx (0);
19: }
20:
21: intx_ref new1_intx (int initvalue) {
22:    intx_ref new = malloc (sizeof (struct intx));
23:    assert (new != NULL);
24:    new->tag = intx_tag;
25:    new->value = initvalue;
26:    return new;
27: }
28:
29: void free_intx (intx_ref this) {
30:    assert (is_intx (this));
31:    memset(this, 0, sizeof (struct intx));
32:    free (this);
33: }
34:
35: int get_intx (intx_ref this) {
36:    assert (is_intx (this));
37:    return this->value;
38: }
39:
40: void put_intx (intx_ref this, int newvalue) {
41:    assert (is_intx (this));
42:    this->value = newvalue;
43: }
44:
45: bool is_intx (intx_ref this) {
46:    // LINTED (warning: assignment of 32-bit integer to 8-bit integer)
47:    return this != NULL && this->tag == intx_tag;
48: }
```

```
49:
50: /*
51: ********************************************************************
52:
53: Notes that would normally not be put in the file:
54:
55: A '.c' file always includes its own header.
56:
57: The 'struct' definition itself is specified in the
58: implementation file.  Everything declared in the implementation
59: file is 'private'.  Never put field definitions in a header
60: file.
61:
62: A tag string is defined so that each structure can be identified
63: at runtime similar to the way that 'System.identityHashCode' in
64: Java can identify the type of the object.  It is also cleared
65: out when freed to permit checking of dangling pointers.
66:
67: The tag is static so it can't be accessed outside of this file.
68: Static variables work as in Java if one considers the '.c' file
69: to be a class.
70:
71: The function memset(3) is used before free(3C) in order to avoid
72: having pointers into the object remain live and also to prevent
73: a dangling pointer from verifying true for 'is_intx'.  It also
74: can be used for checking up on types when using 'void*' for
75: 'Object'.
76:
77: All functions assert is_intx as a precondition if appropriate.
78:
79: ********************************************************************
80: */
```

```
 1: # $Id: Makefile,v 1.4 2012-02-14 20:44:24-08 - - $
 2: MKFILE    = Makefile
 3: DEPSFILE  = ${MKFILE}.deps
 4: NOINCLUDE = ci clean spotless
 5: NEEDINCL  = ${filter ${NOINCLUDE}, ${MAKECMDGOALS}}
 6:
 7: GCC      = gcc -g -O0 -Wall -Wextra -std=gnu99
 8: MKDEPS   = gcc -MM
 9: LINT     = lint -Xa -fd -m -u -x -errchk=%all
10:
11: CHEADER   =        intx.h
12: CSOURCE   = main.c intx.c
13: OBJECTS   = ${CSOURCE:.c=.o}
14: EXECBIN   = intx
15: SOURCES   = ${CHEADER} ${CSOURCE} ${MKFILE}
16: LISTSRC   = ${SOURCES} ${DEPSFILE}
17: LISTING   = Listing.intx.ps
18:
19: all : ${EXECBIN}
20:
21: ${EXECBIN} : ${OBJECTS}
22:         ${GCC} -o $@ ${OBJECTS}
23:
24: %.o : %.c
25:         ${GCC} -c $<
26:
27: lint : ${CSOURCE}
28:         ${LINT} ${CSOURCE}
29:         checksource ${CSOURCE}
30:
31: ci : ${SOURCES}
32:         cid + ${SOURCES}
33:
34: ident : ${SOURCES}
35:         ident ${SOURCES}
36:
37: lis : ${SOURCES} test
38:         mkpspdf ${LISTING} ${LISTSRC} test*.lis
39:
40: clean :
41:         - rm ${OBJECTS} ${DEPSFILE} core test*.lis
42:
43: spotless : clean
44:         - rm ${EXECBIN}
45:
46: test : ${EXECBIN}
47:         runprogram.perl -x testa.lis ${EXECBIN}
48:         valgrind --leak-check=full ${EXECBIN} >testb.lis 2>&1
49:
50: deps : ${CSOURCE} ${CHEADER}
51:         @ echo "# ${DEPSFILE} created ‘date‘" >${DEPSFILE}
52:         ${MKDEPS} ${CSOURCE} >>${DEPSFILE}
53:
54: ${DEPSFILE} :
55:         @ touch ${DEPSFILE}
56:         ${MAKE} --no-print-directory deps
57:
58: again :
59:         gmake spotless deps ci lint all lis
60:
61: ifeq ("${NEEDINCL}","")
62: include ${DEPSFILE}
63: endif
64:
```

```
1: # Makefile.deps created Wed Feb 15 20:47:04 PST 2012
2: main.o: main.c intx.h
3: intx.o: intx.c intx.h
```

```
 1:
 2: ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 3: log: testa.log
 4: ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 5:
 6:      1  Script  : /afs/cats.ucsc.edu/courses/cmps012b-wm/bin/runprogram.perl
 7:      2  limit c :    0 max core file size (KB)
 8:      3  limit f : 4194303 max output file size (KB)
 9:      4  limit t : 4294967295 max CPU time (sec)
10:      5  stdin   : /dev/null
11:      6  stdout  : testa.out
12:      7  stderr  : testa.err
13:      8  log     : testa.log
14:      9  listing : testa.lis
15:     10  Command : intx
16:     11  starting: pid 28293: 20:47:05.00
17:     12  finished: pid 28293: 20:47:05.00, real 0.00, user 0.00, sys 0.00
18:     13  pstatus: 0x0000 EXIT STATUS = 0
19:
20: ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
21: stdin: /dev/null
22: ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
23:
24:
25: ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
26: stdout: testa.out
27: ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
28:
29:      1  intx: starting: Wed Feb 15 20:47:05.451253  PST 2012
30:      2  box = 0x1bc610d0
31:      3  box value is 1024
32:      4  intx: finished: Wed Feb 15 20:47:05.451419  PST 2012
33:
34: ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
35: stderr: testa.err
36: ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
37:
```

```
 1: ==28300== Memcheck, a memory error detector
 2: ==28300== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
 3: ==28300== Using Valgrind-3.5.0 and LibVEX; rerun with -h for copyright info
 4: ==28300== Command: intx
 5: ==28300==
 6: intx: starting: Wed Feb 15 20:47:05.694541  PST 2012
 7: box = 0x4c30d70
 8: box value is 1024
 9: intx: finished: Wed Feb 15 20:47:05.771954  PST 2012
10: ==28300==
11: ==28300== HEAP SUMMARY:
12: ==28300==     in use at exit: 0 bytes in 0 blocks
13: ==28300==   total heap usage: 17 allocs, 17 frees, 2,603 bytes allocated
14: ==28300==
15: ==28300== All heap blocks were freed -- no leaks are possible
16: ==28300==
17: ==28300== For counts of detected and suppressed errors, rerun with: -v
18: ==28300== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 4 from 4)
```