```
 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 1.5 2012-11-01 19:33:32-07 - - $ */
 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: int main (int argc, char **argv) {
22:    argc = argc; // Avoid:16: warning: unused parameter 'argc'
23:    execname = basename (argv[0]);
24:
25:    /* Declare the box and initialize it.  */
26:    intx_ref box = new_intx();
27:    printf ("box = %p\n", box);
28:
29:    /* Perform a couple of operations on it.  */
30:    put_intx (box, 1024);
31:    printf ("box value is %d\n", get_intx (box));
32:
33:    /* Free up the box and set the handle to NULL to avoid a dangle. */
34:    free_intx (box);
35:    box = NULL;
36:
37:    return EXIT_SUCCESS;
38: }
39:
```

```
 1: /* $Id: intx.c,v 1.4 2012-11-01 19:33:32-07 - - $ */
 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:    return this != NULL && this->tag == intx_tag;
47: }
```

```
48:
49: /*
50: ********************************************************************
51:
52: Notes that would normally not be put in the file:
53:
54: A '.c' file always includes its own header.
55:
56: The 'struct' definition itself is specified in the
57: implementation file.  Everything declared in the implementation
58: file is 'private'.  Never put field definitions in a header
59: file.
60:
61: A tag string is defined so that each structure can be identified
62: at runtime similar to the way that 'System.identityHashCode' in
63: Java can identify the type of the object.  It is also cleared
64: out when freed to permit checking of dangling pointers.
65:
66: The tag is static so it can't be accessed outside of this file.
67: Static variables work as in Java if one considers the '.c' file
68: to be a class.
69:
70: The function memset(3) is used before free(3C) in order to avoid
71: having pointers into the object remain live and also to prevent
72: a dangling pointer from verifying true for 'is_intx'.  It also
73: can be used for checking up on types when using 'void*' for
74: 'Object'.
75:
76: All functions assert is_intx as a precondition if appropriate.
77:
78: ********************************************************************
79: */
```

```
 1: # $Id: Makefile,v 1.5 2012-11-01 19:33:32-07 - - $
 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:
10: CHEADER   =        intx.h
11: CSOURCE   = main.c intx.c
12: OBJECTS   = ${CSOURCE:.c=.o}
13: EXECBIN   = intx
14: SOURCES   = ${CHEADER} ${CSOURCE} ${MKFILE}
15: LISTSRC   = ${SOURCES} ${DEPSFILE}
16: LISTING   = Listing.intx.ps
17:
18: all : ${EXECBIN}
19:
20: ${EXECBIN} : ${OBJECTS}
21:         ${GCC} -o $@ ${OBJECTS}
22:
23: %.o : %.c
24:         cid + $<
25:         ${GCC} -c $<
26:
27: ci : ${SOURCES}
28:         cid + ${SOURCES}
29:
30: ident : ${SOURCES}
31:         ident ${SOURCES}
32:
33: lis : ${SOURCES} test
34:         mkpspdf ${LISTING} ${LISTSRC} test*.lis
35:
36: clean :
37:         - rm ${OBJECTS} ${DEPSFILE} core test*.lis
38:
39: spotless : clean
40:         - rm ${EXECBIN} ${LISTING:.ps=.p*}
41:
42: test : ${EXECBIN}
43:         runprogram.perl -x testa.lis ${EXECBIN}
44:         valgrind --leak-check=full ${EXECBIN} >testb.lis 2>&1
45:
46: deps : ${CSOURCE} ${CHEADER}
47:         @ echo "# ${DEPSFILE} created `date`" >${DEPSFILE}
48:         ${MKDEPS} ${CSOURCE} >>${DEPSFILE}
49:
50: ${DEPSFILE} :
51:         @ touch ${DEPSFILE}
52:         ${MAKE} --no-print-directory deps
53:
54: again :
55:         gmake spotless deps ci all lis
56:
57: ifeq ("${NEEDINCL}","")
58: include ${DEPSFILE}
59: endif
60:
```

```
1: # Makefile.deps created Wed Nov  7 18:06:58 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 15691: 18:06:59.00
17:     12  finished: pid 15691: 18:06:59.00, 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  box = 0x637010
30:      2  box value is 1024
31:
32: ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
33: stderr: testa.err
34: ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
35:
```

```
 1: ==15698== Memcheck, a memory error detector
 2: ==15698== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
 3: ==15698== Using Valgrind-3.6.0 and LibVEX; rerun with -h for copyright info
 4: ==15698== Command: intx
 5: ==15698==
 6: box = 0x4c23040
 7: box value is 1024
 8: ==15698==
 9: ==15698== HEAP SUMMARY:
10: ==15698==     in use at exit: 0 bytes in 0 blocks
11: ==15698==   total heap usage: 1 allocs, 1 frees, 16 bytes allocated
12: ==15698==
13: ==15698== All heap blocks were freed -- no leaks are possible
14: ==15698==
15: ==15698== For counts of detected and suppressed errors, rerun with: -v
16: ==15698== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 6 from 6)
```