```
 1: // $Id: xlist.h,v 1.3 2019-04-24 17:08:41-07 - - $
 2:
 3: #ifndef __XLIST_H__
 4: #define __XLIST_H__
 5:
 6: template <typename T>
 7: struct xlist {
 8:    struct node;
 9:    struct link {
10:       node* next;
11:       node* prev;
12:       explicit link (node* next_ = nullptr, node* prev_ = nullptr):
13:                      next(next_), prev(prev_) {}
14:       node* operator&() { return static_cast<node*> (this); }
15:    };
16:    struct node: link {
17:       T item;
18:       explicit node (node* next_ = nullptr, node* prev_ = nullptr,
19:                      const T& item_ = T()):
20:             link(next_, prev_), item(item_) {}
21:    };
22:    link base;
23:
24:    xlist(): base (&base, &base) {}
25:    xlist (const xlist&) = delete;
26:    xlist& operator= (const xlist&) = delete;
27:    ~xlist() { while (not empty()) pop_back(); }
28:
29:    bool empty() { return base.next == &base; }
30:    void push_back (const T&);
31:    void pop_back();
32:    T& back() { return base.prev->item; }
33:
34:    class iterator;
35:    iterator begin() { return iterator (base.next); }
36:    iterator end() { return iterator (&base); }
37: };
38:
```

```
39:
40: template <typename T>
41: struct xlist<T>::iterator {
42:    node* curr;
43:    explicit iterator (node* curr_ = nullptr): curr(curr_) {}
44:    T& operator*() { return curr->item; }
45:    iterator& operator++() { curr = curr->next; return *this; }
46:    iterator& operator--() { curr = curr->prev; return *this; }
47:    bool operator== (const iterator &that) { return curr == that.curr; }
48:    bool operator!= (const iterator &that) { return curr != that.curr; }
49: };
50:
51: template <typename T>
52: void xlist<T>::push_back (const T& item) {
53:    node* tmp = new node (&base, base.prev, item);
54:    base.prev = tmp;
55:    tmp->prev->next = tmp;
56: }
57:
58: template <typename T>
59: void xlist<T>::pop_back() {
60:    node* tmp = base.prev;
61:    base.prev = tmp->prev;
62:    base.prev->next = &base;
63:    delete tmp;
64: }
65:
66: #endif
67:
```

```
 1: // $Id: testxlist.cpp,v 1.1 2016-06-28 14:47:09-07 - - $
 2:
 3: #include <cxxabi.h>
 4: #include <iostream>
 5: #include <string>
 6: #include <typeinfo>
 7: using namespace std;
 8:
 9: #include "xlist.h"
10:
11: template <typename T>
12: ostream& show_node (typename xlist<T>::node* np) {
13:    cout << np << "->{next=" << np->next << ", prev=" << np->prev;
14:    return cout;
15: }
16:
17: template <typename T>
18: void show_list (const string &str, xlist<T>& thelist) {
19:    cout << str << ":" << endl;
20:    show_node<T> (&thelist.base) << "}" << endl;
21:    for (typename xlist<T>::iterator it = thelist.begin();
22:         it != thelist.end();
23:         ++it) {
24:       show_node<T> (it.curr) << ", item=" << *it << "}" << endl;
25:    }
26: }
27:
28: void test_int() {
29:    xlist<int> xli;
30:    cout << "sizeof (xlist) = " << sizeof (xli) << endl;
31:    cout << "sizeof (int) = " << sizeof (int) << endl;
32:    show_list ("After decl", xli);
33:    xli.push_back(3);
34:    xli.push_back(4);
35:    xli.push_back(5);
36:    xli.push_back(6);
37:    show_list ("After push_back", xli);
38:    cout << xli.back() << endl;
39:    xli.pop_back();
40:    cout << xli.back() << endl;
41:    xli.pop_back();
42:    show_list ("At end of test_int", xli);
43: }
44:
```

```
45:
46: void test_string() {
47:    xlist<string> xli;
48:    cout << "sizeof (xlist) = " << sizeof (xli) << endl;
49:    cout << "sizeof (string) = " << sizeof (string) << endl;
50:    show_list ("After decl", xli);
51:    xli.push_back("Hello");
52:    xli.push_back("World");
53:    xli.push_back("foo");
54:    xli.push_back("bar");
55:    show_list ("After push_back", xli);
56:    cout << xli.back() << endl;
57:    xli.pop_back();
58:    cout << xli.back() << endl;
59:    xli.pop_back();
60:    show_list ("At end of test_string", xli);
61: }
62:
63: int main() {
64:    string line = "------------------------------------------------";
65:    cout << line << endl;
66:    test_int();
67:    cout << line << endl;
68:    test_string();
69:    cout << line << endl;
70:    return 0;
71: }
72:
73: //TEST// testxlist >testxlist.out 2>&1
74: //TEST// mkpspdf testxlist.ps xlist.h testxlist.cpp testxlist.out
75:
```

```
 1: ------------------------------------------------
 2: sizeof (xlist) = 16
 3: sizeof (int) = 4
 4: After decl:
 5: 0x7ffdd5906d70->{next=0x7ffdd5906d70, prev=0x7ffdd5906d70}
 6: After push_back:
 7: 0x7ffdd5906d70->{next=0xfdd0a0, prev=0xfdd100}
 8: 0xfdd0a0->{next=0xfdd0c0, prev=0x7ffdd5906d70, item=3}
 9: 0xfdd0c0->{next=0xfdd0e0, prev=0xfdd0a0, item=4}
10: 0xfdd0e0->{next=0xfdd100, prev=0xfdd0c0, item=5}
11: 0xfdd100->{next=0x7ffdd5906d70, prev=0xfdd0e0, item=6}
12: 6
13: 5
14: At end of test_int:
15: 0x7ffdd5906d70->{next=0xfdd0a0, prev=0xfdd0c0}
16: 0xfdd0a0->{next=0xfdd0c0, prev=0x7ffdd5906d70, item=3}
17: 0xfdd0c0->{next=0x7ffdd5906d70, prev=0xfdd0a0, item=4}
18: ------------------------------------------------
19: sizeof (xlist) = 16
20: sizeof (string) = 8
21: After decl:
22: 0x7ffdd5906d40->{next=0x7ffdd5906d40, prev=0x7ffdd5906d40}
23: After push_back:
24: 0x7ffdd5906d40->{next=0xfdd0a0, prev=0xfdd100}
25: 0xfdd0a0->{next=0xfdd0c0, prev=0x7ffdd5906d40, item=Hello}
26: 0xfdd0c0->{next=0xfdd0e0, prev=0xfdd0a0, item=World}
27: 0xfdd0e0->{next=0xfdd100, prev=0xfdd0c0, item=foo}
28: 0xfdd100->{next=0x7ffdd5906d40, prev=0xfdd0e0, item=bar}
29: bar
30: foo
31: At end of test_string:
32: 0x7ffdd5906d40->{next=0xfdd0a0, prev=0xfdd0c0}
33: 0xfdd0a0->{next=0xfdd0c0, prev=0x7ffdd5906d40, item=Hello}
34: 0xfdd0c0->{next=0x7ffdd5906d40, prev=0xfdd0a0, item=World}
35: ------------------------------------------------
```