

```
1: // $Id: thingstack.cpp,v 1.22 2018-06-27 16:51:39-07 - - $
2:
3: #include <iostream>
4: #include <list>
5:
6: using namespace std;
7:
8: #include "iterstack.h"
9:
10: int serial = 0;
11:
12: #define PRINT(FUNC) print(FUNC, __LINE__)
13:
14: struct thing {
15:     int ser;
16:     int val;
17:     explicit thing(int v = int());
18:     thing (const thing&);
19:     thing& operator= (const thing&);
20:     ~thing();
21:     void print (const char* name, int line);
22: };
23:
24: thing::thing(int v): ser(++serial), val(v) {
25:     PRINT(__PRETTY_FUNCTION__);
26: }
27:
28: thing::thing (const thing& that): ser(++serial), val(that.val) {
29:     PRINT(__PRETTY_FUNCTION__);
30: }
31:
32: thing& thing::operator= (const thing& that) {
33:     if (this != &that) {
34:         val = that.val;
35:     }
36:     PRINT(__PRETTY_FUNCTION__);
37:     return *this;
38: }
39:
40: thing::~~thing() {
41:     PRINT(__PRETTY_FUNCTION__);
42: }
43:
44: void thing::print (const char* name, int line) {
45:     cout << name << "[" << line << "]: " << this << "-> ser="
46:         << ser << ", val=" << val << endl;
47: }
48:
```

```
49:
50: #define SCOPE(X) cout << endl << X << " scope " << __LINE__ << endl
51:
52: int main (int, char**) {
53:     iterstack<thing> stk;
54:     for (int i = 0; i < 3; ++i) {
55:         SCOPE("enter");
56:         thing t(i);
57:         cout << endl << "stk.push (t);" << endl;
58:         stk.push (t);
59:         SCOPE("leave");
60:     }
61:     while (not stk.empty()) {
62:         SCOPE("enter");
63:         thing t = stk.top();
64:         t.PRINT("stk.top()");
65:         cout << endl << "stk.pop();" << endl;
66:         stk.pop();
67:         SCOPE("leave");
68:     }
69:     return 0;
70: }
71:
72: /*
73: //TEST// valgrind --leak-check=full --show-reachable=yes \
74: //TEST//      --log-file=thingstack.out.grind \
75: //TEST//      thingstack >thingstack.out 2>&1
76: //TEST// mkpspdf thingstack.ps thingstack.cpp* iterstack.h \
77: //TEST//      thingstack.out*
78: */
79:
```

```
yle-cast thingstack.cpp -o thingstack -lm
```

```
1: // $Id: iterstack.h,v 1.5 2014-05-30 13:47:32-07 - - $
2:
3: //
4: // The class std::stack does not provide an iterator, which is
5: // needed for this class. So, like std::stack, class iterstack
6: // is implemented on top of a container.
7: //
8: // We use private inheritance because we want to restrict
9: // operations only to those few that are approved. All functions
10: // are merely inherited from the container, with only ones needed
11: // being exported as public.
12: //
13: // No implementation file is needed because all functions are
14: // inherited, and the convenience functions that are added are
15: // trivial, and so can be inline.
16: //
17: // Any underlying container which supports the necessary operations
18: // could be used, such as vector, list, or deque.
19: //
20:
21: #ifndef __ITERSTACK_H__
22: #define __ITERSTACK_H__
23:
24: #include <vector>
25: using namespace std;
26:
27: template <typename value_type>
28: class iterstack: private vector<value_type> {
29:     private:
30:         using vector<value_type>::crbegin;
31:         using vector<value_type>::crend;
32:         using vector<value_type>::push_back;
33:         using vector<value_type>::pop_back;
34:         using vector<value_type>::back;
35:         using const_iterator = typename
36:             vector<value_type>::const_reverse_iterator;
37:     public:
38:         using vector<value_type>::clear;
39:         using vector<value_type>::empty;
40:         using vector<value_type>::size;
41:         const_iterator begin() { return crbegin(); }
42:         const_iterator end() { return crend(); }
43:         void push (const value_type& value) { push_back (value); }
44:         void pop() { pop_back(); }
45:         const value_type& top() const { return back(); }
46: };
47:
48: #endif
49:
```

```
1:
2: enter scope 55
3: thing::thing(int)[25]: 0x1ffeffff678-> ser=1, val=0
4:
5: stk.push (t);
6: thing::thing(const thing&)[29]: 0x5a23040-> ser=2, val=0
7:
8: leave scope 59
9: thing::~~thing()[41]: 0x1ffeffff678-> ser=1, val=0
10:
11: enter scope 55
12: thing::thing(int)[25]: 0x1ffeffff678-> ser=3, val=1
13:
14: stk.push (t);
15: thing::thing(const thing&)[29]: 0x5a23098-> ser=4, val=1
16: thing::thing(const thing&)[29]: 0x5a23090-> ser=5, val=0
17: thing::~~thing()[41]: 0x5a23040-> ser=2, val=0
18:
19: leave scope 59
20: thing::~~thing()[41]: 0x1ffeffff678-> ser=3, val=1
21:
22: enter scope 55
23: thing::thing(int)[25]: 0x1ffeffff678-> ser=6, val=2
24:
25: stk.push (t);
26: thing::thing(const thing&)[29]: 0x5a230f0-> ser=7, val=2
27: thing::thing(const thing&)[29]: 0x5a230e0-> ser=8, val=0
28: thing::thing(const thing&)[29]: 0x5a230e8-> ser=9, val=1
29: thing::~~thing()[41]: 0x5a23090-> ser=5, val=0
30: thing::~~thing()[41]: 0x5a23098-> ser=4, val=1
31:
32: leave scope 59
33: thing::~~thing()[41]: 0x1ffeffff678-> ser=6, val=2
34:
35: enter scope 62
36: thing::thing(const thing&)[29]: 0x1ffeffff670-> ser=10, val=2
37: stk.top()[64]: 0x1ffeffff670-> ser=10, val=2
38:
39: stk.pop();
40: thing::~~thing()[41]: 0x5a230f0-> ser=7, val=2
41:
42: leave scope 67
43: thing::~~thing()[41]: 0x1ffeffff670-> ser=10, val=2
44:
45: enter scope 62
46: thing::thing(const thing&)[29]: 0x1ffeffff670-> ser=11, val=1
47: stk.top()[64]: 0x1ffeffff670-> ser=11, val=1
48:
49: stk.pop();
50: thing::~~thing()[41]: 0x5a230e8-> ser=9, val=1
51:
52: leave scope 67
53: thing::~~thing()[41]: 0x1ffeffff670-> ser=11, val=1
54:
55: enter scope 62
56: thing::thing(const thing&)[29]: 0x1ffeffff670-> ser=12, val=0
57: stk.top()[64]: 0x1ffeffff670-> ser=12, val=0
58:
```

```
59: stk.pop();  
60: thing::~~thing()[41]: 0x5a230e0-> ser=8, val=0  
61:  
62: leave scope 67  
63: thing::~~thing()[41]: 0x1ffeffff670-> ser=12, val=0
```

```
1: ==16568== Memcheck, a memory error detector
2: ==16568== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al
.
3: ==16568== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright
info
4: ==16568== Command: thingstack
5: ==16568== Parent PID: 16567
6: ==16568==
7: ==16568==
8: ==16568== HEAP SUMMARY:
9: ==16568==      in use at exit: 0 bytes in 0 blocks
10: ==16568==    total heap usage: 3 allocs, 3 frees, 56 bytes allocated
11: ==16568==
12: ==16568== All heap blocks were freed -- no leaks are possible
13: ==16568==
14: ==16568== For counts of detected and suppressed errors, rerun with: -v
15: ==16568== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```