```
// $Id: thingstack.cpp,v 1.22 2018-06-27 16:51:39-07 - - $

#include <iostream>
#include <list>

using namespace std;

#include "iterstack.h"

int serial = 0;

#define PRINT(FUNC) print(FUNC, __LINE__)

struct thing {
   int ser;
   int val;
   explicit thing(int v = int());
   thing (const thing&);
   thing& operator= (const thing&);
   ~thing();
   void print (const char* name, int line);
};

thing::thing(int v): ser(++serial), val(v) {
   PRINT(__PRETTY_FUNCTION__);
}

thing::thing (const thing& that): ser(++serial), val(that.val) {
   PRINT(__PRETTY_FUNCTION__);
}

thing& thing::operator= (const thing& that) {
   if (this != &that) {
      val = that.val;
   }
   PRINT(__PRETTY_FUNCTION__);
   return *this;
}

thing::~thing() {
   PRINT(__PRETTY_FUNCTION__);
}

void thing::print (const char* name, int line) {
   cout << name << "[" << line << "]: " << this << "-> ser="
        << ser << ", val=" << val << endl;
}

```

```

#define SCOPE(X) cout << endl << X << " scope " << __LINE__ << endl

int main (int, char**) {
   iterstack<thing> stk;
   for (int i = 0; i < 3; ++i) {
      SCOPE("enter");
      thing t(i);
      cout << endl << "stk.push (t);" << endl;
      stk.push (t);
      SCOPE("leave");
   }
   while (not stk.empty()) {
      SCOPE("enter");
      thing t = stk.top();
      t.PRINT("stk.top()");
      cout << endl << "stk.pop();" << endl;
      stk.pop();
      SCOPE("leave");
   }
   return 0;
}

/*
//TEST// valgrind --leak-check=full --show-reachable=yes \
//TEST//       --log-file=thingstack.out.grind \
//TEST//       thingstack >thingstack.out 2>&1
//TEST// mkpspdf thingstack.ps thingstack.cpp* iterstack.h \
//TEST//       thingstack.out*
*/

```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: starting thingstack.cpp
checksource thingstack.cpp
ident thingstack.cpp
thingstack.cpp:
     $Id: thingstack.cpp,v 1.22 2018-06-27 16:51:39-07 - - $
cpplint.py.perl thingstack.cpp
Done processing thingstack.cpp
g++ -g -O0 -Wall -Wextra -Werror -Wpedantic -Wshadow -fdiagnostics-color=never -std=gnu++17 -Wold-style-cast thingstack.cpp -o thingstack -lm
rm -f thingstack.o
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: finished thingstack.cpp
```

```
// $Id: iterstack.h,v 1.5 2014-05-30 13:47:32-07 - - $

//
// The class std::stack does not provide an iterator, which is
// needed for this class.  So, like std::stack, class iterstack
// is implemented on top of a container.
//
// We use private inheritance because we want to restrict
// operations only to those few that are approved.  All functions
// are merely inherited from the container, with only ones needed
// being exported as public.
//
// No implementation file is needed because all functions are
// inherited, and the convenience functions that are added are
// trivial, and so can be inline.
//
// Any underlying container which supports the necessary operations
// could be used, such as vector, list, or deque.
//

#ifndef __ITERSTACK_H__
#define __ITERSTACK_H__

#include <vector>
using namespace std;

template <typename value_type>
class iterstack: private vector<value_type> {
   private:
      using vector<value_type>::crbegin;
      using vector<value_type>::crend;
      using vector<value_type>::push_back;
      using vector<value_type>::pop_back;
      using vector<value_type>::back;
      using const_iterator = typename
            vector<value_type>::const_reverse_iterator;
   public:
      using vector<value_type>::clear;
      using vector<value_type>::empty;
      using vector<value_type>::size;
      const_iterator begin() { return crbegin(); }
      const_iterator end() { return crend(); }
      void push (const value_type& value) { push_back (value); }
      void pop() { pop_back(); }
      const value_type& top() const { return back(); }
};

#endif

```

```

enter scope 55
thing::thing(int)[25]: 0x1ffefff488-> ser=1, val=0

stk.push (t);
thing::thing(const thing&)[29]: 0x5a23040-> ser=2, val=0

leave scope 59
thing::~thing()[41]: 0x1ffefff488-> ser=1, val=0

enter scope 55
thing::thing(int)[25]: 0x1ffefff488-> ser=3, val=1

stk.push (t);
thing::thing(const thing&)[29]: 0x5a23098-> ser=4, val=1
thing::thing(const thing&)[29]: 0x5a23090-> ser=5, val=0
thing::~thing()[41]: 0x5a23040-> ser=2, val=0

leave scope 59
thing::~thing()[41]: 0x1ffefff488-> ser=3, val=1

enter scope 55
thing::thing(int)[25]: 0x1ffefff488-> ser=6, val=2

stk.push (t);
thing::thing(const thing&)[29]: 0x5a230f0-> ser=7, val=2
thing::thing(const thing&)[29]: 0x5a230e0-> ser=8, val=0
thing::thing(const thing&)[29]: 0x5a230e8-> ser=9, val=1
thing::~thing()[41]: 0x5a23090-> ser=5, val=0
thing::~thing()[41]: 0x5a23098-> ser=4, val=1

leave scope 59
thing::~thing()[41]: 0x1ffefff488-> ser=6, val=2

enter scope 62
thing::thing(const thing&)[29]: 0x1ffefff480-> ser=10, val=2
stk.top()[64]: 0x1ffefff480-> ser=10, val=2

stk.pop();
thing::~thing()[41]: 0x5a230f0-> ser=7, val=2

leave scope 67
thing::~thing()[41]: 0x1ffefff480-> ser=10, val=2

enter scope 62
thing::thing(const thing&)[29]: 0x1ffefff480-> ser=11, val=1
stk.top()[64]: 0x1ffefff480-> ser=11, val=1

stk.pop();
thing::~thing()[41]: 0x5a230e8-> ser=9, val=1

leave scope 67
thing::~thing()[41]: 0x1ffefff480-> ser=11, val=1

enter scope 62
thing::thing(const thing&)[29]: 0x1ffefff480-> ser=12, val=0
stk.top()[64]: 0x1ffefff480-> ser=12, val=0

stk.pop();
thing::~thing()[41]: 0x5a230e0-> ser=8, val=0

leave scope 67
thing::~thing()[41]: 0x1ffefff480-> ser=12, val=0
```

```
==18909== Memcheck, a memory error detector
==18909== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==18909== Using Valgrind-3.14.0.GIT and LibVEX; rerun with -h for copyright info
==18909== Command: thingstack
==18909== Parent PID: 18908
==18909==
==18909==
==18909== HEAP SUMMARY:
==18909==     in use at exit: 0 bytes in 0 blocks
==18909==   total heap usage: 3 allocs, 3 frees, 56 bytes allocated
==18909==
==18909== All heap blocks were freed -- no leaks are possible
==18909==
==18909== For counts of detected and suppressed errors, rerun with: -v
==18909== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```