```
 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: // $Id: teststack.cpp,v 1.13 2016-06-13 13:44:55-07 - - $
 2:
 3: #include <iostream>
 4: #include <string>
 5: #include <vector>
 6: using namespace std;
 7:
 8: #include "iterstack.h"
 9:
10: int main (int argc, char** argv) {
11:    vector<string> args (&argv[1], &argv[argc]);
12:
13:    iterstack<string> stk;
14:    for (const auto& arg: args) {
15:       cout << "Pushing: " << arg << endl;
16:       stk.push (arg);
17:    }
18:    for (const auto& elt: stk) cout << "Iteration: " << elt << endl;
19:
20:    while (not stk.empty()) {
21:       cout << "Popping: " << stk.top() << endl;
22:       stk.pop();
23:    }
24:    return 0;
25: }
26:
27: /*
28: //TEST// valgrind --leak-check=full --show-reachable=yes \
29: //TEST//       --log-file=teststack.out.grind \
30: //TEST//       teststack foo bar baz qux quux >teststack.out 2>&1
31: //TEST// mkpspdf teststack.ps iterstack.h teststack.cpp* teststack.out*
32: */
33:
```

```
1: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: starting teststack.cpp
2: teststack.cpp:
3:      $Id: teststack.cpp,v 1.13 2016-06-13 13:44:55-07 - - $
4: g++ -g -O0 -Wall -Wextra -rdynamic -std=gnu++14 teststack.cpp
5:          -o teststack -lglut -lGLU -lGL -lX11 -lrt -lm
6: rm -f teststack.o
7: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: finished teststack.cpp
```

```
 1: Pushing: foo
 2: Pushing: bar
 3: Pushing: baz
 4: Pushing: qux
 5: Pushing: quux
 6: Iteration: quux
 7: Iteration: qux
 8: Iteration: baz
 9: Iteration: bar
10: Iteration: foo
11: Popping: quux
12: Popping: qux
13: Popping: baz
14: Popping: bar
15: Popping: foo
```

```
    1: ==10899== Memcheck, a memory error detector
    2: ==10899== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al
.
    3: ==10899== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright
info
    4: ==10899== Command: teststack foo bar baz qux quux
    5: ==10899== Parent PID: 10898
    6: ==10899==
    7: ==10899==
    8: ==10899== HEAP SUMMARY:
    9: ==10899==     in use at exit: 0 bytes in 0 blocks
   10: ==10899==   total heap usage: 11 allocs, 11 frees, 317 bytes allocated
   11: ==10899==
   12: ==10899== All heap blocks were freed -- no leaks are possible
   13: ==10899==
   14: ==10899== For counts of detected and suppressed errors, rerun with: -v
   15: ==10899== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 1 from 1)
```