```
 1: // $Id: allocvec.cpp,v 1.1 2013-03-14 14:11:50-07 - - $
 2:
 3: //
 4: // allocvec
 5: // example shows how an allocator is used to manage a vector.
 6: //
 7: // Container elements must be default-constructable, copyable,
 8: // assignable, and destructible.
 9: //
10: // Simple functions are inlined for efficiency.
11: //
12:
13: #include <cassert>
14: #include <iostream>
15: #include <memory>
16: #include <stdexcept>
17: #include <string>
18:
19: using namespace std;
20:
21: template <typename value_t, class alloc_t = allocator<value_t>>
22: class allocvec {
23:    public:
24:       typedef value_t *iterator;
25:       typedef const value_t *const_iterator;
26:    private:
27:       static const size_t MIN_RESERVE = 8;
28:       alloc_t alloc;
29:       size_t size_;
30:       size_t capacity_;
31:       iterator begin_;
32:    public:
33:       // Inline members
34:       inline iterator begin() {return begin_;}
35:       inline iterator end() {return &begin_[size_];}
36:       inline const_iterator begin() const {return begin_;}
37:       inline const_iterator end() const {return &begin_[size_];}
38:       inline size_t size() const {return size_;}
39:       inline size_t capacity() const {return capacity_;}
40:       inline bool empty() const {return size_ == 0;}
41:       inline value_t &operator[] (size_t pos) {return begin_[pos];}
42:       inline const value_t &operator[] (size_t pos) const
43:                                              {return begin_[pos];}
44:       inline value_t &back() {return begin_[size_ - 1];}
45:       inline const value_t &back() const {return begin_[size_ - 1];}
46:
47:       // Other members
48:       void push_back (const value_t &value);
49:       void pop_back();
50:       explicit allocvec (size_t size = 0,
51:                          const value_t &init = value_t(),
52:                          alloc_t alloc = allocator<value_t>());
53:       allocvec (const allocvec &);
54:       allocvec &operator= (const allocvec &);
55:       ~allocvec();
56:       void reserve (size_t size);
57: };
58:
```

```
 59:
 60: template <typename value_t, class alloc_t>
 61: allocvec<value_t,alloc_t>::allocvec (size_t size,
 62:                const value_t &init, alloc_t alloc):
 63:                alloc(alloc), size_(size), capacity_(size),
 64:                begin_(NULL) {
 65:    if (size > 0) {
 66:       begin_ = alloc.allocate (capacity_);
 67:       for (iterator itor = begin(); itor != end(); ++itor) {
 68:          alloc.construct (itor, init);
 69:       }
 70:    }
 71: }
 72:
 73: template <typename value_t, class alloc_t>
 74: allocvec<value_t,alloc_t>::allocvec (const allocvec &that):
 75:                alloc(that.alloc), size_(that.size_),
 76:                capacity_(that.size_), begin_(NULL) {
 77:    if (size_ > 0) {
 78:       begin_ = alloc.allocate (capacity_);
 79:       iterator thisitor = begin_;
 80:       iterator thatitor = that.begin();
 81:       while (that.itor != that.end()) {
 82:          alloc.construct (*thisitor++, *thatitor++);
 83:       }
 84:    }
 85: }
 86:
 87: template <typename value_t, class alloc_t>
 88: allocvec<value_t,alloc_t> &
 89: allocvec<value_t,alloc_t>::operator= (const allocvec &that) {
 90:    if (this == &that) return *this;
 91:    for (iterator itor = begin(); itor != end(); ++itor) {
 92:       alloc.destroy (itor);
 93:    }
 94:    if (capacity_ < that.size()) {
 95:       alloc.deallocate (begin_, capacity_);
 96:       size_ = capacity_ = that.size();
 97:       begin_ = alloc.allocate (capacity_);
 98:    }
 99:    iterator thisitor = begin_;
100:    iterator thatitor = that.begin();
101:    while (that.itor != that.end()) {
102:       alloc.construct (*thisitor++, *thatitor++);
103:    }
104: }
105:
106: template <typename value_t, class alloc_t>
107: allocvec<value_t,alloc_t>::~allocvec() {
108:    for (iterator itor = begin(); itor != end(); ++itor) {
109:       alloc.destroy (itor);
110:    }
111:    alloc.deallocate (begin_, capacity_);
112: }
113:
```

```
114:
115: template <typename value_t, class alloc_t>
116: void allocvec<value_t,alloc_t>::reserve (size_t capacity) {
117:    if (capacity < MIN_RESERVE) capacity = MIN_RESERVE;
118:    if (capacity <= capacity_) return;
119:    iterator newarray = alloc.allocate (capacity);
120:    iterator newitor = newarray;
121:    for (iterator itor = begin(); itor != end(); ++itor) {
122:       alloc.construct (newitor++, *itor);
123:       alloc.destroy (itor);
124:    }
125:    alloc.deallocate (begin_, capacity_);
126:    capacity_ = capacity;
127:    begin_ = newarray;
128: }
129:
130: template <typename value_t, class alloc_t>
131: void allocvec<value_t,alloc_t>::push_back (const value_t &value) {
132:    if (size_ == capacity_) reserve (size_ * 2);
133:    alloc.construct (&begin_[size_++], value);
134: }
135:
136: template <typename value_t, class alloc_t>
137: void allocvec<value_t,alloc_t>::pop_back() {
138:    alloc.destroy (&begin_[--size_]);
139: }
140:
```

```
141:
142: int main (int argc, char **argv) {
143:    allocvec<string> vec;
144:    cout << "sizeof(allocvec) = " << sizeof vec << endl;
145:    for (char **arg = &argv[1]; arg < &argv[argc]; ++arg) {
146:       vec.push_back (*arg);
147:    }
148:    for (auto itor = vec.begin(); itor != vec.end(); ++itor) {
149:       cout << "vector: " << " " << *itor << endl;
150:    }
151:    cout << "vec.size() = " << vec.size() << endl;
152:    for (size_t count = 0; count <= vec.size() / 2; ++count) {
153:       cout << "half: " << vec.back() << endl ;
154:       vec.pop_back();
155:    }
156:    cout << "vec.size() = " << vec.size() << endl;
157:    return EXIT_SUCCESS;
158: }
159:
160: /*
161: //TEST// valgrind --leak-check=full --show-reachable=yes allocvec \
162: //TEST//          This is a simple test of allocvec. \
163: //TEST//          >allocvec.out 2>&1
164: //TEST// mkpspdf allocvec.ps allocvec.cpp* allocvec.out
165: */
166:
167:
```

```
1: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: starting allocvec.cpp
2: allocvec.cpp: $Id: allocvec.cpp,v 1.1 2013-03-14 14:11:50-07 - - $
3: g++ -g -O0 -Wall -Wextra -std=gnu++0x allocvec.cpp -o allocvec -lm
4: rm -f allocvec.o
5: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: finished allocvec.cpp
```

```
 1: ==1358== Memcheck, a memory error detector
 2: ==1358== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
 3: ==1358== Using Valgrind-3.6.0 and LibVEX; rerun with -h for copyright info
 4: ==1358== Command: allocvec This is a simple test of allocvec.
 5: ==1358==
 6: sizeof(allocvec) = 32
 7: vector:  This
 8: vector:  is
 9: vector:  a
10: vector:  simple
11: vector:  test
12: vector:  of
13: vector:  allocvec.
14: vec.size() = 7
15: half: allocvec.
16: half: of
17: half: test
18: vec.size() = 4
19: ==1358==
20: ==1358== HEAP SUMMARY:
21: ==1358==     in use at exit: 0 bytes in 0 blocks
22: ==1358==   total heap usage: 8 allocs, 8 frees, 267 bytes allocated
23: ==1358==
24: ==1358== All heap blocks were freed -- no leaks are possible
25: ==1358==
26: ==1358== For counts of detected and suppressed errors, rerun with: -v
27: ==1358== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 6 from 6)
```