```
 1: // $Id: xvector.h,v 1.47 2016-07-21 19:04:26-07 - - $
 2:
 3: //
 4: // Vector explicitly managing memory
 5: // using allocate/deallocate/construct/destroy.
 6: // with iterator/const_iterator.
 7: //
 8:
 9: #ifndef __XVECTOR_H__
10: #define __XVECTOR_H__
11:
12: #include <initializer_list>
13: #include <iterator>
14:
15: template <typename Type> class xvector_base;
16: template <typename Type> class xvector;
17: template <typename Type> class xvector_iterator;
18:
19: // Base class guarantees all xvector ctors no uninit pointers.
20: // Manages allocation/deallocation of memory.
21: template <typename Type>
22: class xvector_base {
23:    friend class xvector<Type>;
24:
25: private:
26:    allocator<Type> alloc;
27:    Type* begin {nullptr};
28:    Type* end {nullptr};
29:    Type* limit {nullptr};
30:    xvector_base(): begin(nullptr), end(nullptr), limit(nullptr) {}
31:    xvector_base (const xvector_base&) = delete;
32:    xvector_base& operator= (const xvector_base&) = delete;
33:    xvector_base (xvector_base&& that);
34:    xvector_base& operator= (xvector_base&&);
35:    ˜xvector_base() { alloc.deallocate (begin, limit - begin); }
36:    explicit xvector_base (size_t capacity);
37: };
38:
```

```
39:
40: // Uses xvector_base to construct/destroy elements
41: // and provide access to elements.
42: template <typename Type>
43: class xvector {
44:
45: public:
46:     using value_type = Type;
47:     using reference = value_type&;
48:     using const_reference = const value_type&;
49:     using pointer = value_type*;
50:     using const_pointer = const value_type*;
51:     using difference_type = ptrdiff_t;
52:     using size_type = size_t;
53:     using iterator = xvector_iterator<value_type>;
54:     using const_iterator = xvector_iterator<const value_type>;
55:
56: private:
57:     static constexpr size_type MIN_RESERVE = 4;
58:     xvector_base<value_type> base;
59:     void deallocate_this();
60:     void copy_from_that (const xvector&);
61:
```

```
 62:
 63: public:
 64:     // Replace implicit members.
 65:     xvector(): base() {}
 66:     xvector (const xvector&);
 67:     xvector (xvector&&);
 68:     xvector& operator= (const xvector&);
 69:     xvector& operator= (xvector&&);
 70:     ~xvector() { resize (0); }
 71:
 72:     // More constructors.
 73:     explicit xvector (size_type size,
 74:                       const value_type& val = value_type{});
 75:     explicit xvector (initializer_list<value_type> ilist);
 76:
 77:     // Capacity.
 78:     size_type size() const      { return base.end - base.begin; }
 79:     size_type capacity() const { return base.limit - base.begin; }
 80:     bool empty() const          { return size() == 0; }
 81:     void reserve (size_type);
 82:     void resize (size_type, const value_type &val = value_type());
 83:
 84:     // Modifiers: push_back, pop_back.
 85:     void push_back (const value_type&);
 86:     void push_back (value_type&&);
 87:     void pop_back();
 88:
 89:     // Iterators: begin, cbegin, end, cend.
 90:     iterator begin()                  { return iterator (base.begin); }
 91:     const_iterator begin() const  { return iterator (base.begin); }
 92:     const_iterator cbegin() const { return iterator (base.begin); }
 93:     iterator end()                    { return iterator (base.end); }
 94:     const_iterator end() const    { return iterator (base.end); }
 95:     const_iterator cend() const   { return iterator (base.end); }
 96:
 97:     // Access: [], front, back.
 98:     reference operator[] (size_type pos) { return base.begin[pos]; }
 99:     const_reference operator[] (size_type pos) const
100:                                             { return base.begin[pos]; }
101:     reference front()                       { return base.begin[0]; }
102:     const_reference front() const           { return base.begin[0]; }
103:     reference back()                        { return base.end[-1]; }
104:     const_reference back() const            { return base.end[-1]; }
105: };
106:
```

```
107:
108: // Relational operators for comparing vectors lexicographically.
109: // Do not need to be members.
110: template <typename Type>
111: bool operator== (const xvector<Type>& lhs, const xvector<Type>& rhs);
112:
113: template <typename Type>
114: bool operator< (const xvector<Type>& lhs, const xvector<Type>& rhs);
115:
116: template <typename Type>
117: bool operator!= (const xvector<Type>& lhs, const xvector<Type>& rhs) {
118:     return not (lhs == rhs);
119: }
120:
121: template <typename Type>
122: bool operator> (const xvector<Type>& lhs, const xvector<Type>& rhs) {
123:     return rhs < lhs;
124: }
125:
126: template <typename Type>
127: bool operator<= (const xvector<Type>& lhs, const xvector<Type>& rhs) {
128:     return not (rhs < lhs);
129: }
130:
131: template <typename Type>
132: bool operator>= (const xvector<Type>& lhs, const xvector<Type>& rhs) {
133:     return not (lhs < rhs);
134: }
135:
```

```
136:
137: // xvector<Type>::iterator
138: template <typename Type>
139: class xvector_iterator {
140:     friend class xvector<Type>;
141:     template<typename> friend class xvector_iterator;
142:
143: public:
144:     using iterator_category = random_access_iterator_tag;
145:     using value_type = Type;
146:     using reference = value_type&;
147:     using pointer = value_type*;
148:     using const_reference = const value_type&;
149:     using const_pointer = const value_type*;
150:     using difference_type = ptrdiff_t;
151:
152: private:
153:     pointer base {};
154:     xvector_iterator (pointer base): base(base) {}
155:
156: public:
157:     xvector_iterator(): base(nullptr) {}
158:     // Other implicit members by default OK.
159:
160:     xvector_iterator& operator++()      { ++base; return *this; }
161:     xvector_iterator& operator--()      { --base; return *this; }
162:     xvector_iterator operator++ (int);
163:     xvector_iterator operator-- (int);
164:
165:     reference operator*()                            { return *base; }
166:     const_reference operator*() const                { return *base; }
167:     pointer operator->()                             { return base; }
168:     const_pointer operator->() const                 { return base; }
169:     reference operator[] (size_t pos)            { return base[pos]; }
170:     const_reference operator[] (size_t pos) const { return base[pos]; }
171:
172:     // Comparison and arithmetic operators.
173:     bool operator== (const xvector_iterator& that) const
174:                                     { return base == that.base; }
175:     bool operator<  (const xvector_iterator& that) const
176:                                     { return base < that.base; }
177:     xvector_iterator& operator+= (difference_type offset)
178:                                     { base += offset; return *this; }
179:     xvector_iterator& operator-= (difference_type offset)
180:                                     { base -= offset; return *this; }
181:     difference_type operator- (const xvector_iterator& that)
182:                                     { return base - that.base; }
183:     // Implicit conversion of iterator to const_iterator.
184:     operator xvector_iterator<const value_type>() const
185:     { return xvector_iterator<const value_type> (base); }
186:     operator bool() { return base != nullptr; }
187:
188: };
189:
```

```
190:
191: //
192: // XVECTOR-ITERATOR NON-MEMBER OPERATORS
193: //
194:
195: template <typename Type>
196: xvector_iterator<Type>
197: operator+ (typename xvector_iterator<Type>::difference_type offset,
198:           const xvector_iterator<Type>& itor) {
199:    xvector_iterator<Type> result {itor};
200:    return result += offset;
201: }
202:
203: template <typename Type>
204: xvector_iterator<Type>
205: operator+ (const xvector_iterator<Type>& itor,
206:           typename xvector_iterator<Type>::difference_type offset) {
207:    xvector_iterator<Type> result {itor};
208:    return result += offset;
209: }
210:
211: template <typename Type>
212: xvector_iterator<Type>
213: operator- (const xvector_iterator<Type>& itor,
214:           typename xvector_iterator<Type>::difference_type offset) {
215:    xvector_iterator<Type> result {itor};
216:    return result -= offset;
217: }
218:
219: template <typename Type>
220: bool operator!= (const xvector_iterator<Type>& one,
221:                  const xvector_iterator<Type>& two) {
222:    return not (one == two);
223: }
224:
225: template <typename Type>
226: bool operator> (const xvector_iterator<Type>& one,
227:                 const xvector_iterator<Type>& two) {
228:    return two < one;
229: }
230:
231: template <typename Type>
232: bool operator<= (const xvector_iterator<Type>& one,
233:                  const xvector_iterator<Type>& two) {
234:    return not (two < one);
235: }
236:
237: template <typename Type>
238: bool operator>= (const xvector_iterator<Type>& one,
239:                  const xvector_iterator<Type>& two) {
240:    return not (one < two);
241: }
242:
243: #include "xvector.tcc"
244:
245: #endif
246:
```

```
 1: // $Id: xvector.tcc,v 1.38 2016-07-21 19:04:26-07 - - $
 2:
 3: #include <memory>
 4: #include <utility>
 5:
 6: //
 7: // XVECTOR-BASE MEMBERS AND FUNCTIONS.
 8: //
 9:
10: template <typename Type>
11: xvector_base<Type>::xvector_base (xvector_base&& that):
12:    begin (that.begin),
13:    end (that.end),
14:    limit (that.limit) {
15:    that.begin = that.end = that.limit = nullptr;
16: }
17:
18: template <typename Type>
19: xvector_base<Type>&
20: xvector_base<Type>::operator= (xvector_base&& that) {
21:    if (this != &that) {
22:       if (begin) alloc.deallocate (begin, limit - begin);
23:       begin = that.begin;
24:       end = that.end;
25:       limit = that.limit;
26:       that.begin = that.end = that.limit = nullptr;
27:    }
28:    return *this;
29: }
30:
31: template <typename Type>
32: xvector_base<Type>::xvector_base (size_t capacity):
33:    begin (alloc.allocate (capacity)),
34:    end (begin),
35:    limit (&begin[capacity]) {
36: }
37:
```

```
38:
39: //
40: // XVECTOR CONSTRUCTORS, OPERATOR=, DESTRUCTOR.
41: //
42:
43: // Copy constructor.
44: template <typename Type>
45: xvector<Type>::xvector (const xvector& that): base (that.capacity()) {
46:    uninitialized_copy (that.cbegin(), that.cend(), base.begin);
47:    base.end = base.begin + that.size();
48: }
49:
50: // Copy operator=.
51: template <typename Type>
52: xvector<Type>& xvector<Type>::operator= (const xvector& that) {
53:    if (*this != that) {
54:       resize (0);
55:       reserve (that.size());
56:       uninitialized_copy (that.cbegin(), that.cend(), base.begin);
57:       base.end = base.begin + that.size();
58:    }
59:    return *this;
60: }
61:
62: // Move constructor.
63: template <typename Type>
64: xvector<Type>::xvector (xvector&& that):
65:    base (std::move (that.base)) {
66: }
67:
68: // Move operator=.
69: template <typename Type>
70: xvector<Type>& xvector<Type>::operator= (xvector&& that) {
71:    if (*this != that) base = std::move (that);
72:    return *this;
73: }
74:
75: // Fill constructor.
76: template <typename Type>
77: xvector<Type>::xvector (size_type fill_size, const value_type& val):
78:    base (fill_size) {
79:    base.end = &base.begin[fill_size];
80:    uninitialized_fill (base.begin, base.end, val);
81: }
82:
83: // Initializer list constructor.
84: template <typename Type>
85: xvector<Type>::xvector (initializer_list<value_type> list):
86:    base (list.size()) {
87:    uninitialized_copy (list.begin(), list.end(), base.begin);
88:    base.end = base.begin + list.size();
89: }
90:
```

```
 91:
 92: //
 93: // XVECTOR OTHER FUNCTION MEMBERS.
 94: //
 95:
 96: // Reserve minimum uninitialized space.
 97: template <typename Type>
 98: void xvector<Type>::reserve (size_type capacity_) {
 99:    if (capacity_ <= capacity()) return;
100:    if (capacity_ < MIN_RESERVE) capacity_ = MIN_RESERVE;
101:    if (capacity_ < 2 * capacity()) capacity_ = 2 * capacity();
102:    xvector_base<value_type> new_base (capacity_);
103:    new_base.end = &new_base.begin[size()];
104:    if (base.begin and size() > 0) {
105:       uninitialized_copy (begin(), end(), new_base.begin);
106:       resize (0);
107:    }
108:    base = std::move (new_base);
109: }
110:
111: // Increase or decrease size of vector.
112: template <typename Type>
113: void xvector<Type>::resize (size_type size_, const value_type &val) {
114:    while (size_ < size()) pop_back();
115:    while (size_ > size()) push_back (val);
116: }
117:
118: template <typename Type>
119: void xvector<Type>::push_back (const value_type& that) {
120:    reserve (size() + 1);
121:    base.alloc.construct (base.end++, that);
122: }
123:
124: template <typename Type>
125: void xvector<Type>::push_back (value_type&& that) {
126:    reserve (size() + 1);
127:    *base.end++ = std::move (that);
128: }
129:
130: template <typename Type>
131: void xvector<Type>::pop_back() {
132:    base.alloc.destroy (--base.end);
133: }
134:
```

```
135:
136: //
137: // XVECTOR RELATIONAL OPERATORS == and <
138: //
139: template <typename Type>
140: bool operator== (const xvector<Type>& lhs, const xvector<Type>& rhs) {
141:     if (lhs.size() != rhs.size()) return false;
142:     auto lhs_itor = lhs.cbegin();
143:     auto rhs_itor = rhs.cbegin();
144:     for (; lhs_itor != lhs.cend(); ++lhs_itor, ++rhs_itor) {
145:         if (*lhs_itor != *rhs_itor) return false;
146:     }
147:     return true;
148: }
149:
150: template <typename Type>
151: bool operator< (const xvector<Type>& lhs, const xvector<Type>& rhs) {
152:     auto lhs_itor = lhs.cbegin();
153:     auto rhs_itor = rhs.cbegin();
154:     for (; lhs_itor != lhs.cend(); ++lhs_itor, ++rhs_itor) {
155:         if (rhs_itor == rhs.cend()) return false;
156:         if (*lhs_itor < *rhs_itor) return true;
157:         if (*rhs_itor < *lhs_itor) return false;
158:     }
159:     return rhs_itor != rhs.cend();
160: }
161:
```

```
162:
163: //
164: // XVECTOR::ITERATOR FUNCTIONS.
165: //
166:
167: template <typename Type>
168: xvector_iterator<Type> xvector_iterator<Type>::operator++ (int) {
169:     xvector_iterator<Type> result {*this};
170:     ++base;
171:     return result;
172: }
173:
174: template <typename Type>
175: xvector_iterator<Type> xvector_iterator<Type>::operator-- (int) {
176:     xvector_iterator<Type> result {*this};
177:     --base;
178:     return result;
179: }
180:
```

```
 1: # $Id: Makefile,v 1.34 2016-07-21 19:04:26-07 - - $
 2:
 3: DEPFILE  = Makefile.dep
 4: NOINCL   = ci clean spotless
 5: NEEDINCL = ${filter ${NOINCL}, ${MAKECMDGOALS}}
 6: PROGRAMS = testbool.cpp testint.cpp testpointer.cpp \
 7:            testsort.cpp teststring.cpp testvector.cpp
 8: OBJFILES = ${PROGRAMS:.cpp=.o}
 9: BINARIES = ${PROGRAMS:.cpp=}
10: TESTRUNS = ${foreach file, ${PROGRAMS}, ${file} ${file:.cpp=.out}}
11: AUXFILES = xvector.h xvector.tcc Makefile
12: OUTPUTS  = ${PROGRAMS:.cpp=.out}
13: LISTING  = Listing.ps
14:
15: GPP      = g++ -g -O0 -Wall -Wextra -std=gnu++14
16: GRIND    = valgrind --leak-check=full --show-reachable=yes
17:
18: all : ${BINARIES}
19:
20: % : %.o
21:         ${GPP} $< -o $@
22:
23: %.o : %.cpp
24:         cpplint.py.perl $<
25:         ${GPP} $< -c
26:
27: ci : ${AUXFILES} ${PROGRAMS}
28:         cid + ${AUXFILES} ${PROGRAMS}
29:         checksource ${AUXFILES} ${PROGRAMS}
30:
31: out : ${OUTPUTS}
32:
33: %.out : %
34:         ${GRIND} $< >$@ 2>&1; pstatus >>$@
35:
36: lis : out
37:         pkill gv || exit 0
38:         mkpspdf ${LISTING} ${AUXFILES} ${TESTRUNS}
39:
40: ${DEPFILE} :
41:         ${GPP} -MM ${PROGRAMS} >${DEPFILE}
42:         cat ${DEPFILE}
43:
44: clean :
45:         - rm ${DEPFILE} ${OBJFILES}
46:
47: spotless : clean
48:         - rm ${BINARIES} ${OUTPUTS} ${LISTING} ${LISTING:.ps=.pdf}
49:
50: again :
51:         ${MAKE} spotless ci
52:         ${MAKE} all out lis
53:
54: ifeq (${NEEDINCL},)
55: include ${DEPFILE}
56: endif
57:
```

```cpp
 1: // $Id: testbool.cpp,v 1.6 2013-08-12 18:55:17-07 - - $
 2:
 3: //
 4: // Sieve of Eratosthenes.
 5: // To Koskinon 'Eratosthenous.
 6: //
 7:
 8: #include <iomanip>
 9: #include <iostream>
10:
11: using namespace std;
12:
13: #include "xvector.h"
14:
15: int main () {
16:    constexpr size_t rows {32};
17:    constexpr size_t columns {16};
18:    constexpr size_t num_width {4};
19:    xvector<bool> sieve (rows * columns, true);
20:    sieve[0] = sieve[1] = false;
21:
22:    for (size_t prime {2}; prime * prime < sieve.size(); ++prime) {
23:       if (sieve[prime]) {
24:          for (size_t itor {prime * prime};
25:                itor < sieve.size(); itor += prime) {
26:             sieve[itor] = false;
27:          }
28:       }
29:    }
30:
31:    size_t prime_count {0};
32:    size_t col_count {0};
33:    cout << "Sieve of Eratosthenes." << endl;
34:    cout << "To Koskinon 'Eratosthenous." << endl;
35:    for (size_t itor {0}; itor < rows * columns; ++itor) {
36:       cout << setw (num_width);
37:       if (sieve[itor]) { cout << itor; ++prime_count; }
38:                else { cout << "."; }
39:       if (++col_count % columns == 0) cout << endl;
40:    }
41:    cout << "Sieve size: " << sieve.size() << ".   ";
42:    cout << "Primes found: " << prime_count << "." << endl;
43:
44:    return 0;
45: }
46:
```

```
 1: ==23798== Memcheck, a memory error detector
 2: ==23798== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al
.
 3: ==23798== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright
info
 4: ==23798== Command: testbool
 5: ==23798==
 6: Sieve of Eratosthenes.
 7: To Koskinon 'Eratosthenous.
 8:   .   .   2   3   .   5   .   7   .   .   .  11   .  13   .   .
 9:   .  17   .  19   .   .   .  23   .   .   .   .  29   .  31
10:   .   .   .   .   .  37   .   .   .  41   .  43   .   .   .  47
11:   .   .   .   .   .  53   .   .   .   .   .  59   .  61   .   .
12:   .   .   .  67   .   .   .  71   .  73   .   .   .   .   .  79
13:   .   .   .  83   .   .   .   .   .  89   .   .   .   .   .   .
14:   .  97   .   .   . 101   . 103   .   .   . 107   . 109   .   .
15:   . 113   .   .   .   .   .   .   .   .   .   .   .   .   . 127
16:   .   .   . 131   .   .   .   .   . 137   . 139   .   .   .   .
17:   .   .   .   .   . 149   . 151   .   .   .   .   . 157   .   .
18:   .   .   . 163   .   .   . 167   .   .   .   .   . 173   .   .
19:   .   .   . 179   . 181   .   .   .   .   .   .   .   .   . 191
20:   . 193   .   .   . 197   . 199   .   .   .   .   .   .   .   .
21:   .   .   . 211   .   .   .   .   .   .   .   .   .   .   . 223
22:   .   .   . 227   . 229   .   .   . 233   .   .   .   .   . 239
23:   . 241   .   .   .   .   .   .   .   .   . 251   .   .   .   .
24:   . 257   .   .   .   . 263   .   .   .   .   . 269   . 271
25:   .   .   .   .   . 277   .   .   . 281   . 283   .   .   .   .
26:   .   .   .   .   . 293   .   .   .   .   .   .   .   .   .   .
27:   .   .   . 307   .   .   . 311   . 313   .   .   . 317   .   .
28:   .   .   .   .   .   .   .   .   .   .   . 331   .   .   .   .
29:   . 337   .   .   .   .   .   .   .   .   . 347   . 349   .   .
30:   . 353   .   .   .   .   . 359   .   .   .   .   .   .   . 367
31:   .   .   .   .   . 373   .   .   .   .   . 379   .   .   . 383
32:   .   .   .   .   . 389   .   .   .   .   .   .   . 397   .   .
33:   . 401   .   .   .   .   .   .   . 409   .   .   .   .   .   .
34:   .   .   . 419   . 421   .   .   .   .   .   .   .   .   . 431
35:   . 433   .   .   .   .   . 439   .   .   . 443   .   .   .   .
36:   . 449   .   .   .   .   .   .   . 457   .   .   . 461   . 463
37:   .   .   . 467   .   .   .   .   .   .   .   .   .   .   . 479
38:   .   .   .   .   .   .   .   .   . 487   .   .   . 491   .   .
39:   .   .   . 499   .   .   . 503   .   .   .   .   . 509   .   .
40: Sieve size: 512.  Primes found: 97.
41: ==23798==
42: ==23798== HEAP SUMMARY:
43: ==23798==     in use at exit: 0 bytes in 0 blocks
44: ==23798==   total heap usage: 1 allocs, 1 frees, 512 bytes allocated
45: ==23798==
46: ==23798== All heap blocks were freed -- no leaks are possible
47: ==23798==
48: ==23798== For counts of detected and suppressed errors, rerun with: -v
49: ==23798== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
50: pstatus: 0x0000 EXIT STATUS = 0
```

```cpp
 1: // $Id: testint.cpp,v 1.11 2014-07-09 13:00:06-07 - - $
 2:
 3: #include <iostream>
 4: #include <sstream>
 5: #include <string>
 6:
 7: using namespace std;
 8:
 9: #include "xvector.h"
10:
11: void printvec (const string& label, const xvector<int>& vec) {
12:     cout << label << ":";
13:     for (size_t i {0}; i < vec.size(); ++i) {
14:        cout << " [" << i << "]" << vec[i];
15:     }
16:     cout << endl;
17: }
18:
19: int main() {
20:     xvector<int> aa {10, 20, 30, 40, 50, 60, 70, 80, 90};
21:     xvector<int> va;
22:     cout << "sizeof aa = " << sizeof aa << endl;
23:     cout << "sizeof va = " << sizeof va << endl;
24:
25:     printvec ("loop1(aa)", aa);
26:     for (auto& i: aa) va.push_back (i);
27:
28:     for (auto i = va.cbegin(); i != va.cend(); ++i) {
29:        cout << "loop2: " << &*i << "->" << *i << endl;
30:     }
31:
32:     printvec ("loop3(va)", va);
33:
34:     xvector<int> vb (va);
35:     xvector<int>::iterator j {vb.begin()};
36:     xvector<int>::const_iterator cj {j};
37:
38:     ++cj;
39:     cout << &*cj << ": " << *j << " " << *cj << endl;
40:
41:     //xvector<int>::iterator k = cj;
42:     //error: conversion from 'xvector_iterator<const int>'
43:     //to non-scalar type 'xvector_iterator<int>' requested
44:
45:     xvector<int> bb {10, 20, 30, 45};
46:     printvec ("compare(aa)", aa);
47:     printvec ("compare(bb)", bb);
48:     cout << "aa < bb = " << boolalpha << (aa < bb) << endl;
49:
50:     return 0;
51: }
52:
```

```
 1: ==23802== Memcheck, a memory error detector
 2: ==23802== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al
.
 3: ==23802== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright
info
 4: ==23802== Command: testint
 5: ==23802==
 6: sizeof aa = 32
 7: sizeof va = 32
 8: loop1(aa): [0]10 [1]20 [2]30 [3]40 [4]50 [5]60 [6]70 [7]80 [8]90
 9: loop2: 0x5a241d0->10
10: loop2: 0x5a241d4->20
11: loop2: 0x5a241d8->30
12: loop2: 0x5a241dc->40
13: loop2: 0x5a241e0->50
14: loop2: 0x5a241e4->60
15: loop2: 0x5a241e8->70
16: loop2: 0x5a241ec->80
17: loop2: 0x5a241f0->90
18: loop3(va): [0]10 [1]20 [2]30 [3]40 [4]50 [5]60 [6]70 [7]80 [8]90
19: 0x5a242c4: 10 20
20: compare(aa): [0]10 [1]20 [2]30 [3]40 [4]50 [5]60 [6]70 [7]80 [8]90
21: compare(bb): [0]10 [1]20 [2]30 [3]45
22: aa < bb = true
23: ==23802==
24: ==23802== HEAP SUMMARY:
25: ==23802==     in use at exit: 0 bytes in 0 blocks
26: ==23802==   total heap usage: 10 allocs, 10 frees, 368 bytes allocated
27: ==23802==
28: ==23802== All heap blocks were freed -- no leaks are possible
29: ==23802==
30: ==23802== For counts of detected and suppressed errors, rerun with: -v
31: ==23802== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
32: pstatus: 0x0000 EXIT STATUS = 0
```

```cpp
 1: // $Id: testpointer.cpp,v 1.3 2015-02-03 13:06:33-08 - - $
 2:
 3: #include <iomanip>
 4: #include <iostream>
 5:
 6: using namespace std;
 7:
 8: #include "xvector.h"
 9:
10: struct node {
11:    int a;
12:    int b;
13:    node (int ai = 0, int bi = 0): a(ai), b(bi) {}
14: };
15:
16: int main() {
17:    xvector<int*> vecpi;
18:    for (int i = 0; i < 10; ++i) vecpi.push_back (new int (i));
19:    for (auto it = vecpi.begin(); it != vecpi.end(); ++it) **it *= **it;
20:    cout << "vecpi:";
21:    for (auto it = vecpi.begin(); it != vecpi.end(); ++it)
22:        cout << " " << **it;
23:    cout << endl;
24:    while (not vecpi.empty()) {
25:       int *ip = vecpi.back();
26:       vecpi.pop_back();
27:       delete ip;
28:    }
29:
30:    xvector<node> vecn;
31:    for (int i = 0; i < 10; ++i) vecn.push_back (node (i, i * i));
32:    cout << "vecn:";
33:    for (auto i = vecn.cbegin(); i != vecn.cend(); ++i) {
34:       cout << " (" << (*i).a << "," << i->b << ")";
35:    }
36:    cout << endl;
37:
38:    xvector<node*> vecpn;
39:    for (int i = 0; i < 10; ++i) vecpn.push_back (new node (i, i * i));
40:    for (auto i = vecpn.cbegin(); i != vecpn.cend(); ++i) {
41:       cout << "vecpn: " << *i << "->(" << (**i).a << "," << (*i)->b
42:            << ")" << endl;
43:    }
44:    while (not vecpn.empty()) {
45:       delete vecpn.back();
46:       vecpn.pop_back();
47:    }
48:
49:    return 0;
50: }
```

```
 1: ==23805== Memcheck, a memory error detector
 2: ==23805== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al
.
 3: ==23805== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright
info
 4: ==23805== Command: testpointer
 5: ==23805==
 6: vecpi: 0 1 4 9 16 25 36 49 64 81
 7: vecn: (0,0) (1,1) (2,4) (3,9) (4,16) (5,25) (6,36) (7,49) (8,64) (9,81)
 8: vecpn: 0x5a246a0->(0,0)
 9: vecpn: 0x5a24750->(1,1)
10: vecpn: 0x5a247a0->(2,4)
11: vecpn: 0x5a247f0->(3,9)
12: vecpn: 0x5a24840->(4,16)
13: vecpn: 0x5a24910->(5,25)
14: vecpn: 0x5a24960->(6,36)
15: vecpn: 0x5a249b0->(7,49)
16: vecpn: 0x5a24a00->(8,64)
17: vecpn: 0x5a24b10->(9,81)
18: ==23805==
19: ==23805== HEAP SUMMARY:
20: ==23805==     in use at exit: 0 bytes in 0 blocks
21: ==23805==   total heap usage: 29 allocs, 29 frees, 792 bytes allocated
22: ==23805==
23: ==23805== All heap blocks were freed -- no leaks are possible
24: ==23805==
25: ==23805== For counts of detected and suppressed errors, rerun with: -v
26: ==23805== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
27: pstatus: 0x0000 EXIT STATUS = 0
```

```cpp
 1: // $Id: testsort.cpp,v 1.2 2016-07-21 19:04:26-07 - - $
 2:
 3: #include <algorithm>
 4: #include <iomanip>
 5: #include <iostream>
 6: #include <string>
 7:
 8: using namespace std;
 9:
10: #include "xvector.h"
11:
12: xvector<string> vecvalues {
13:     "Hello", "World",
14:     "foo", "bar", "baz", "qux",
15:     "3.14159265358979",
16:     "1.61803398874989",
17:     "2.71828182845905",
18:     "!@#$%^&*()_+|",
19: };
20:
21: template <typename Iterator>
22: void print (const string& label, Iterator itor, const Iterator &end) {
23:     cout << label << ":" << endl;
24:     for (; itor != end; ++itor) {
25:         cout << "    " << *itor << endl;
26:     }
27: }
28:
29: int main () {
30:     xvector<string> v1 (vecvalues);
31:     sort (v1.begin(), v1.end());
32:     print ("Default sort", v1.cbegin(), v1.cend());
33:
34:     xvector<string> v2 (vecvalues);
35: //   sort (v2.begin(), v2.end(), greater<string>());
36:     print ("Greater sort", v2.cbegin(), v2.cend());
37:
38:     return 0;
39: }
40:
```

```
 1: ==23813== Memcheck, a memory error detector
 2: ==23813== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al
.
 3: ==23813== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright
info
 4: ==23813== Command: testsort
 5: ==23813==
 6: Default sort:
 7:    !@#$%^&*()_+|
 8:    1.61803398874989
 9:    2.71828182845905
10:    3.14159265358979
11:    Hello
12:    World
13:    bar
14:    baz
15:    foo
16:    qux
17: Greater sort:
18:    Hello
19:    World
20:    foo
21:    bar
22:    baz
23:    qux
24:    3.14159265358979
25:    1.61803398874989
26:    2.71828182845905
27:    !@#$%^&*()_+|
28: ==23813==
29: ==23813== HEAP SUMMARY:
30: ==23813==     in use at exit: 0 bytes in 0 blocks
31: ==23813==   total heap usage: 15 allocs, 15 frees, 647 bytes allocated
32: ==23813==
33: ==23813== All heap blocks were freed -- no leaks are possible
34: ==23813==
35: ==23813== For counts of detected and suppressed errors, rerun with: -v
36: ==23813== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
37: pstatus: 0x0000 EXIT STATUS = 0
```

```cpp
 1: // $Id: teststring.cpp,v 1.7 2013-08-13 14:53:57-07 - - $
 2:
 3: #include <iostream>
 4: #include <sstream>
 5: #include <string>
 6:
 7: using namespace std;
 8:
 9: #include "xvector.h"
10:
11: template <typename Iter>
12: void print (Iter itor, Iter end) {
13:    for (; itor != end; ++itor) cout << " " << *itor;
14: }
15:
16: int main() {
17:    xvector<string> vs {"hello", "world", "foo", "bar", "baz"};
18:    xvector<string> vt = vs;
19:    vs.resize (9, "six");
20:
21:    while (not vs.empty()) {
22:       cout << "vs.size = " << vs.size() << ", vs.back() = \""
23:            << vs.back() << "\"" << endl;
24:       vs.pop_back();
25:    }
26:
27:    cout << "second string vt:";
28:    for (auto i = vt.cbegin(); i != vt.cend(); ++i) {
29:       cout << " " << *i;
30:    }
31:    cout << endl;
32:
33:    cout << "template print:";
34:    print (vt.begin(), vt.end());
35:    xvector<string>::iterator j {vt.begin()};
36:    xvector<string>::const_iterator cj {j};
37:    ++cj;
38:    cout << " " << *cj << endl;
39:    cout << "END" << endl;
40:
41:    return 0;
42: }
43:
```

```
 1: ==23816== Memcheck, a memory error detector
 2: ==23816== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al
.
 3: ==23816== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright
info
 4: ==23816== Command: teststring
 5: ==23816==
 6: vs.size = 9, vs.back() = "six"
 7: vs.size = 8, vs.back() = "six"
 8: vs.size = 7, vs.back() = "six"
 9: vs.size = 6, vs.back() = "six"
10: vs.size = 5, vs.back() = "baz"
11: vs.size = 4, vs.back() = "bar"
12: vs.size = 3, vs.back() = "foo"
13: vs.size = 2, vs.back() = "world"
14: vs.size = 1, vs.back() = "hello"
15: second string vt: hello world foo bar baz
16: template print: hello world foo bar baz world
17: END
18: ==23816==
19: ==23816== HEAP SUMMARY:
20: ==23816==     in use at exit: 0 bytes in 0 blocks
21: ==23816==   total heap usage: 9 allocs, 9 frees, 332 bytes allocated
22: ==23816==
23: ==23816== All heap blocks were freed -- no leaks are possible
24: ==23816==
25: ==23816== For counts of detected and suppressed errors, rerun with: -v
26: ==23816== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
27: pstatus: 0x0000 EXIT STATUS = 0
```

```cpp
 1: // $Id: testvector.cpp,v 1.7 2014-05-29 19:02:56-07 - - $
 2:
 3: #include <iomanip>
 4: #include <iostream>
 5:
 6: using namespace std;
 7:
 8: #include "xvector.h"
 9:
10: using dvector = xvector<double>;
11: using matrix = xvector<dvector>;
12:
13: matrix outer_product (const dvector &v1, const dvector &v2) {
14:    matrix m (v1.size(), dvector (v2.size()));
15:    for (size_t i {0}; i < v1.size(); ++i) {
16:       for (size_t j {0}; j < v2.size(); ++j) {
17:          m[i][j] = v1[i] * v2[j];
18:       }
19:    }
20:    return m;
21: }
22:
23: void print (const matrix &m) {
24:    cout << fixed << setprecision(0);
25:    for (size_t i {0}; i < m.size(); ++i) {
26:       for (size_t j {0}; j < m[i].size(); ++j) {
27:          cout << setw(4) << m[i][j];
28:       }
29:       cout << endl;
30:    }
31: }
32:
33: int main() {
34:    dvector v1 {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29};
35:    dvector v2 {2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30};
36:    matrix m = outer_product (v1, v2);
37:    print (m);
38:    cout << "sizeof (dvector) = " << sizeof (dvector) << endl;
39:    cout << "sizeof (matrix)  = " << sizeof (matrix) << endl;
40:    return 0;
41: }
42:
```

```
 1: ==23839== Memcheck, a memory error detector
 2: ==23839== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al
.
 3: ==23839== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright
info
 4: ==23839== Command: testvector
 5: ==23839==
 6:     2    4    6    8   10   12   14   16   18   20   22   24   26   28   30
 7:     6   12   18   24   30   36   42   48   54   60   66   72   78   84   90
 8:    10   20   30   40   50   60   70   80   90  100  110  120  130  140  150
 9:    14   28   42   56   70   84   98  112  126  140  154  168  182  196  210
10:    18   36   54   72   90  108  126  144  162  180  198  216  234  252  270
11:    22   44   66   88  110  132  154  176  198  220  242  264  286  308  330
12:    26   52   78  104  130  156  182  208  234  260  286  312  338  364  390
13:    30   60   90  120  150  180  210  240  270  300  330  360  390  420  450
14:    34   68  102  136  170  204  238  272  306  340  374  408  442  476  510
15:    38   76  114  152  190  228  266  304  342  380  418  456  494  532  570
16:    42   84  126  168  210  252  294  336  378  420  462  504  546  588  630
17:    46   92  138  184  230  276  322  368  414  460  506  552  598  644  690
18:    50  100  150  200  250  300  350  400  450  500  550  600  650  700  750
19:    54  108  162  216  270  324  378  432  486  540  594  648  702  756  810
20:    58  116  174  232  290  348  406  464  522  580  638  696  754  812  870
21: sizeof (dvector) = 32
22: sizeof (matrix)  = 32
23: ==23839==
24: ==23839== HEAP SUMMARY:
25: ==23839==     in use at exit: 0 bytes in 0 blocks
26: ==23839==   total heap usage: 19 allocs, 19 frees, 2,640 bytes allocated
27: ==23839==
28: ==23839== All heap blocks were freed -- no leaks are possible
29: ==23839==
30: ==23839== For counts of detected and suppressed errors, rerun with: -v
31: ==23839== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
32: pstatus: 0x0000 EXIT STATUS = 0
```