```
#include<iostream>
#include<vector>
#include<stdlib.h>

/** People sometimes get the false impression that because STL and
iterators
 *  are fairly new additions to C++, they do sensible, commonplace
things like
 *  bounds checking.
 *
 *  This is sadly not the case, since preserving backwards-
compatibility with C
 *  means preserving the loaded-gun-pointed-at-your-foot aspects, too.
 *
 *  A semi-common related problem is to have doubles take on
impossibly tiny
 *  values in somewhere in your code. Tiny doubles are usually the
result of
 *  reinterpreting the ghost of an int as a double -- see end.
 */

// YEAH!
class Awesome
{
  public:
    int a;
    double b;
    std::string c;
    Awesome() : a(5), b(42.0), c("woot") { }
};

int main(int argc, char* argv[])
{
  // how many doubles to dereference?
  int n;
  if(argc > 1)
    n = atoi(argv[1]);
  else
    n = 10;

  // soil up the memory space
  std::vector<Awesome*> foo;
  for(unsigned i = 0; i < 10 * n; i++)
    foo.push_back(new Awesome());

  for(unsigned i = 0; i < 10 * n; i++)
    delete foo[i];

  // think iterators are smart? think again.
  std::vector<double> b(1);
  std::vector<double>::iterator it;
  // walk right off the end. a segfault is the best thing that could
happen,
  // since at least we'd know something went wrong.
  for(it = b.begin(); it < b.end() + n; it++)
    std::cout << *it << " ";

  std::cout << "\n";

  // ints interpreted as doubles = tiny number
  int fooInt = 42;
  double *fooDouble = reinterpret_cast<double*>(&fooInt);
  std::cout << "fooInt = " << fooInt << std::endl
            << "fooDouble = " << *fooDouble << std::endl;

}
```