```
#include <memory>
#include <iostream>
#include <vector>

using namespace std;

template<typename T>
class debug_allocator {
  public:
    using value_type = T;


    debug_allocator() noexcept {}

    template<typename U>
    debug_allocator(const debug_allocator<U>&) noexcept {}

    value_type* allocate(std::size_t n) {
      std::size_t size = n * sizeof(value_type);
      value_type* t = static_cast<value_type*>(::operator new(size));
      cout << "allocated " << size << " bytes at addr: " << t << "\n";
      return t;
    }

    void deallocate(value_type* p, std::size_t n) noexcept {
      std::size_t size = n * sizeof(value_type);
      ::operator delete(p);
      cout << "deallocated " << size << " bytes at addr: " << p << "\n";
    }

    template<typename U, typename ...Args>
    void construct(U* p, Args&& ...args) {
      ::new(p) U(std::forward<Args>(args)...);
      cout << "constructed new " << typeid(decltype(*p)).name() << " at addr: " << p <
< "\n";
    }

    template<typename U>
    void destroy(U* p) noexcept {
      p->~U();
      cout << "destroyed " << typeid(decltype(*p)).name() << " at addr: " << p << "\n"
;
    }

    template<typename U>
    bool operator==(const debug_allocator<U>&) noexcept {
      return true;
    }

    template<typename U>
    bool operator!=(const debug_allocator<U>& u) noexcept {
      return (*this == u);
    }
};

int main() {
  debug_allocator<int> d;
  vector<int, debug_allocator<int>> v(d);

  cout << "emplace 1\n";
  v.emplace(v.end(), 1);
  cout << "emplace 2\n";
  v.emplace(v.end(), 2);
  cout << "emplace 3\n";
  v.emplace(v.end(), 3);

  return 0;
}
```