Project 2: Wedding Bells Are Ringing… Testing

There were 100 test cases. Each test was worth 1.3 points each; to run the test cases:

1. Remove the main routine from your `BirthdayParty.cpp` file.
2. Append the following text to the end of your `BirthdayParty.cpp` file and build the resulting program.
3. For any test case you wish to try, run the program, providing as input the test number.

```
#include <iostream>
#include <cstdlib>
#include <string>
#include <cassert>
#include <vector>
#include <type_traits>
#include "OnlineDating.h"

using namespace std;

bool lookAtMatches3type(bool (OnlineDating::*)(const
std::string&, const std::string&, OnlineType&) const) { return
true; }
bool lookAtMatches3type(bool (OnlineDating::*)(const
std::string&, const std::string&, OnlineType&)) { return false;
}
bool lookAtMatches3type(...) { return false; }
bool confirmMatch4type(bool (OnlineDating::*)(int, std::string&,
std::string&, OnlineType&) const) { return true; }
bool confirmMatch4type(bool (OnlineDating::*)(int, std::string&,
std::string&, OnlineType&)) { return false; }
bool confirmMatch4type(...) { return false; }

std::string SOMEFNAME = std::string("123");
std::string DEFAULTFNAME = std::string();
std::string ARRAYFNAME[6] = {
      std::string("10"), std::string("20"), std::string("30"),
      std::string("40"), std::string("50"), std::string("60")
};

std::string SOMELNAME = std::string("321");
std::string DEFAULTLNAME = std::string();
std::string ARRAYLNAME[6] = {
      std::string("11"), std::string("21"), std::string("31"),
      std::string("41"), std::string("51"), std::string("61")
};
```

```
OnlineType SOMEVALUE = "junk";
OnlineType DEFAULTV = OnlineType();
OnlineType ARRAYV[6] = {
      "able", "baker", "charlie", "delta", "echo", "foxtrot"
};

bool has(const OnlineDating& m, const std::string& firstName,
const std::string& lastName, const OnlineType& v)
{
      OnlineType v2 = DEFAULTV;
      m.lookAtMatches(firstName, lastName, v2);
      OnlineType v3 = SOMEVALUE;
      m.lookAtMatches(firstName, lastName, v3);
      return v2 == v && v3 == v;
}

void testone(int n)
{
      OnlineDating m;
      switch (n)
      {
      default: {
            cout << "Bad argument" << endl;
      } break; case  1: {
            assert((is_same<decltype(&OnlineDating::noMatches),
bool (OnlineDating::*)() const>::value));
      } break; case  2: {

      assert((is_same<decltype(&OnlineDating::howManyMatches),
int (OnlineDating::*)() const>::value));
      } break; case  3: {

      assert((is_same<decltype(&OnlineDating::someoneAmongMatche
s), bool (OnlineDating::*)(const std::string, const std::string)
const>::value) ||

      (is_same<decltype(&OnlineDating::someoneAmongMatches),
bool (OnlineDating::*)(const std::string&, const std::string&)
const>::value));
      } break; case  4: {

      assert(lookAtMatches3type(&OnlineDating::lookAtMatches));
      } break; case  5: {

      assert(confirmMatch4type(&OnlineDating::confirmMatch));
      } break; case  6: {
            assert(m.noMatches());
```

```
      } break; case  7: {
            assert(m.howManyMatches() == 0);
      } break; case  8: {
            assert(!m.transformMatch(DEFAULTFNAME, DEFAULTLNAME,
SOMEVALUE) && m.howManyMatches() == 0);
      } break; case  9: {
            assert(!m.blockPreviousMatch(DEFAULTFNAME,
DEFAULTLNAME) && m.howManyMatches() == 0);
      } break; case 10: {
            assert(!m.someoneAmongMatches(DEFAULTFNAME,
DEFAULTLNAME));
      } break; case 11: {
            OnlineType v = SOMEVALUE;
            assert(!m.lookAtMatches(DEFAULTFNAME, DEFAULTLNAME,
v) && v == SOMEVALUE);
      } break; case 12: {
            OnlineType v = SOMEVALUE;
            assert(!m.confirmMatch(0, DEFAULTFNAME,
DEFAULTLNAME, v) && v == SOMEVALUE);
      } break; case 13: {
            assert(m.makeMatch(SOMEFNAME, SOMELNAME,
SOMEVALUE));
      } break; case 14: {
            m.makeMatch(SOMEFNAME, SOMELNAME, SOMEVALUE);
            assert(!m.noMatches());
      } break; case 15: {
            m.makeMatch(SOMEFNAME, SOMELNAME, SOMEVALUE);
            assert(m.howManyMatches() == 1);
      } break; case 16: {
            m.makeMatch(SOMEFNAME, SOMELNAME, SOMEVALUE);
            assert(m.someoneAmongMatches(SOMEFNAME, SOMELNAME));
      } break; case 17: {
            m.makeMatch(SOMEFNAME, SOMELNAME, SOMEVALUE);
            OnlineType v = DEFAULTV;
            assert(m.lookAtMatches(SOMEFNAME, SOMELNAME, v));
      } break; case 18: {
            m.makeMatch(SOMEFNAME, SOMELNAME, SOMEVALUE);
            OnlineType v = DEFAULTV;
            m.lookAtMatches(SOMEFNAME, SOMELNAME, v);
            assert(v == SOMEVALUE);
      } break; case 19: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
SOMEVALUE);
            OnlineType v = DEFAULTV;
            assert(!m.lookAtMatches(ARRAYFNAME[1],
ARRAYLNAME[1], v));
      } break; case 20: {
```

```
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            OnlineType v = SOMEVALUE;
            m.lookAtMatches(ARRAYFNAME[1], ARRAYLNAME[1], v);
            assert(v == SOMEVALUE);
      } break; case 21: {
            m.makeMatch(SOMEFNAME, SOMELNAME, SOMEVALUE);
            std::string f = DEFAULTFNAME;
            std::string l = DEFAULTLNAME;
            OnlineType v = DEFAULTV;
            assert(m.confirmMatch(0, f, l, v));
      } break; case 22: {
            m.makeMatch(SOMEFNAME, SOMELNAME, SOMEVALUE);
            std::string f = DEFAULTFNAME;
            std::string l = DEFAULTLNAME;
            OnlineType v = DEFAULTV;
            m.confirmMatch(0, f, l, v);
            assert(f == SOMEFNAME && l == SOMELNAME && v ==
SOMEVALUE);
      } break; case 23: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            assert(!m.noMatches() && m.howManyMatches() == 2);
      } break; case 24: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            assert(m.someoneAmongMatches(ARRAYFNAME[0],
ARRAYLNAME[0]) && m.someoneAmongMatches(ARRAYFNAME[1],
ARRAYLNAME[1]));
      } break; case 25: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            assert(has(m, ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]) && has(m, ARRAYFNAME[1], ARRAYLNAME[1], ARRAYV[1]));
      } break; case 26: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
SOMEVALUE);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
SOMEVALUE);
            assert(has(m, ARRAYFNAME[0], ARRAYLNAME[0],
SOMEVALUE) && has(m, ARRAYFNAME[1], ARRAYLNAME[1], SOMEVALUE));
```

```
      } break; case 27: {
            assert(m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]));
            assert(m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]));
      } break; case 28: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[2]);
            assert(m.howManyMatches() == 2);
      } break; case 29: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[2]);
            assert(has(m, ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]) && has(m, ARRAYFNAME[1], ARRAYLNAME[1], ARRAYV[1]));
      } break; case 30: {
            assert(m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]));
            assert(m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]));
            assert(!m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[2]));
      } break; case 31: {
            assert(m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]));
            assert(m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]));
            assert(!m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]));
      } break; case 32: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            m.transformMatch(ARRAYFNAME[1], ARRAYLNAME[1],
SOMEVALUE);
            assert(m.howManyMatches() == 3 &&
m.someoneAmongMatches(ARRAYFNAME[0], ARRAYLNAME[0]) &&
```

```
                    m.someoneAmongMatches(ARRAYFNAME[1],
ARRAYLNAME[1]) && m.someoneAmongMatches(ARRAYFNAME[2],
ARRAYLNAME[2]));
      } break; case 33: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            m.transformMatch(ARRAYFNAME[1], ARRAYLNAME[1],
SOMEVALUE);
            assert(has(m, ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]) && has(m, ARRAYFNAME[1], ARRAYLNAME[1], SOMEVALUE) &&
                  has(m, ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]));
      } break; case 34: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            assert(m.transformMatch(ARRAYFNAME[1],
ARRAYLNAME[1], SOMEVALUE));
      } break; case 35: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.transformMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[0]);
            assert(m.howManyMatches() == 2 && has(m,
ARRAYFNAME[0], ARRAYLNAME[0], ARRAYV[0]) &&
                  has(m, ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]) && !m.someoneAmongMatches(ARRAYFNAME[2],
ARRAYLNAME[2]));
      } break; case 36: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            assert(!m.transformMatch(ARRAYFNAME[2],
ARRAYLNAME[2], ARRAYV[2]) && !m.transformMatch(ARRAYFNAME[3],
ARRAYLNAME[3], ARRAYV[0]));
      } break; case 37: {
```

```
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeOrTransform(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            assert(!m.noMatches() && m.howManyMatches() == 2);
      } break; case 38: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeOrTransform(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            assert(has(m, ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]) && has(m, ARRAYFNAME[1], ARRAYLNAME[1], ARRAYV[1]));
      } break; case 39: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
SOMEVALUE);
            m.makeOrTransform(ARRAYFNAME[1], ARRAYLNAME[1],
SOMEVALUE);
            assert(has(m, ARRAYFNAME[0], ARRAYLNAME[0],
SOMEVALUE) && has(m, ARRAYFNAME[1], ARRAYLNAME[1], SOMEVALUE));
      } break; case 40: {
            assert(m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]));
            assert(m.makeOrTransform(ARRAYFNAME[1],
ARRAYLNAME[1], ARRAYV[1]));
      } break; case 41: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeOrTransform(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[2]);
            assert(m.howManyMatches() == 2 && has(m,
ARRAYFNAME[0], ARRAYLNAME[0], ARRAYV[2]) &&
                  has(m, ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]));
      } break; case 42: {
            assert(m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]));
            assert(m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]));
            assert(m.makeOrTransform(ARRAYFNAME[0],
ARRAYLNAME[0], ARRAYV[2]));
      } break; case 43: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
```

```
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            m.makeOrTransform(ARRAYFNAME[1], ARRAYLNAME[1],
SOMEVALUE);
            assert(m.howManyMatches() == 3 && has(m,
ARRAYFNAME[0], ARRAYLNAME[0], ARRAYV[0]) &&
                  has(m, ARRAYFNAME[1], ARRAYLNAME[1],
SOMEVALUE) && has(m, ARRAYFNAME[2], ARRAYLNAME[2], ARRAYV[2]));
      } break; case 44: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            assert(m.makeOrTransform(ARRAYFNAME[1],
ARRAYLNAME[1], SOMEVALUE));
      } break; case 45: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeOrTransform(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[0]);
            assert(m.howManyMatches() == 3 && has(m,
ARRAYFNAME[0], ARRAYLNAME[0], ARRAYV[0]) &&
                  has(m, ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]) && has(m, ARRAYFNAME[2], ARRAYLNAME[2], ARRAYV[0]));
      } break; case 46: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            assert(m.makeOrTransform(ARRAYFNAME[2],
ARRAYLNAME[2], ARRAYV[2]));
      } break; case 47: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            assert(m.blockPreviousMatch(ARRAYFNAME[1],
ARRAYLNAME[1]));
      } break; case 48: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
```

```
            m.blockPreviousMatch(ARRAYFNAME[1], ARRAYLNAME[1]);
            assert(!m.noMatches() && m.howManyMatches() == 1 &&
has(m, ARRAYFNAME[0], ARRAYLNAME[0], ARRAYV[0]) &&
                  !m.someoneAmongMatches(ARRAYFNAME[1],
ARRAYLNAME[1]));
      } break; case 49: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.blockPreviousMatch(ARRAYFNAME[0], ARRAYLNAME[0]);
            assert(!m.noMatches() && m.howManyMatches() == 1 &&
has(m, ARRAYFNAME[1], ARRAYLNAME[1], ARRAYV[1]) &&
                  !m.someoneAmongMatches(ARRAYFNAME[0],
ARRAYLNAME[0]));
      } break; case 50: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.blockPreviousMatch(ARRAYFNAME[0], ARRAYLNAME[0]);
            m.blockPreviousMatch(ARRAYFNAME[1], ARRAYLNAME[1]);
            assert(m.howManyMatches() == 0);
      } break; case 51: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            m.blockPreviousMatch(ARRAYFNAME[1], ARRAYLNAME[1]);
            m.blockPreviousMatch(ARRAYFNAME[2], ARRAYLNAME[2]);
            m.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
            assert(m.howManyMatches() == 2 && has(m,
ARRAYFNAME[0], ARRAYLNAME[0], ARRAYV[0]) &&
                  has(m, ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]) && !m.someoneAmongMatches(ARRAYFNAME[1],
ARRAYLNAME[1]) &&
                  !m.someoneAmongMatches(ARRAYFNAME[2],
ARRAYLNAME[2]));
      } break; case 52: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
```

```
            assert(!m.blockPreviousMatch(ARRAYFNAME[2],
ARRAYLNAME[2]) && m.howManyMatches() == 2);
      } break; case 53: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            std::string f;
            std::string l;
            OnlineType v;
            assert(!m.confirmMatch(-1, f, l, v));
      } break; case 54: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            std::string f = SOMEFNAME;
            std::string l = SOMELNAME;
            OnlineType v = SOMEVALUE;
            m.confirmMatch(-1, f, l, v);
            assert(f == SOMEFNAME && l == SOMELNAME && v ==
SOMEVALUE);
      } break; case 55: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            std::string f, l;
            OnlineType v;
            assert(!m.confirmMatch(2, f, l, v));
      } break; case 56: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            std::string f = SOMEFNAME;
            std::string l = SOMELNAME;
            OnlineType v = SOMEVALUE;
            m.confirmMatch(2, f, l, v);
            assert(f == SOMEFNAME && l == SOMELNAME && v ==
SOMEVALUE);
      } break; case 57: {
            m.makeMatch(DEFAULTFNAME, DEFAULTLNAME, SOMEVALUE);
            assert(m.howManyMatches() == 1 && has(m,
DEFAULTFNAME, DEFAULTLNAME, SOMEVALUE));
      } break; case 58: {
```

```
            m.transformMatch(DEFAULTFNAME, DEFAULTLNAME,
SOMEVALUE);
            assert(m.howManyMatches() == 0 &&
!m.someoneAmongMatches(DEFAULTFNAME, DEFAULTLNAME));
      } break; case 59: {
            m.makeOrTransform(DEFAULTFNAME, DEFAULTLNAME,
SOMEVALUE);
            assert(m.howManyMatches() == 1 && has(m,
DEFAULTFNAME, DEFAULTLNAME, SOMEVALUE));
      } break; case 60: {
            m.makeMatch(DEFAULTFNAME, DEFAULTLNAME, SOMEVALUE);
            m.blockPreviousMatch(DEFAULTFNAME, DEFAULTLNAME);
            assert(m.howManyMatches() == 0 &&
!m.someoneAmongMatches(DEFAULTFNAME, DEFAULTLNAME));
      } break; case 61: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            {
                  OnlineDating m2;
                  m2.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
                  m2.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
                  m2.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
                  m.tradeMatches(m2);
                  assert(m.howManyMatches() == 3);
            }
      } break; case 62: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            {
                  OnlineDating m2;
                  m2.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
                  m2.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
                  m2.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
                  m.tradeMatches(m2);
                  assert(has(m, ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]) && has(m, ARRAYFNAME[2], ARRAYLNAME[2], ARRAYV[2]) &&
```

```
                        has(m, ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]) && !m.someoneAmongMatches(ARRAYFNAME[0],
ARRAYLNAME[0]));
            }
      } break; case 63: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            {
                  OnlineDating m2;
                  m2.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
                  m2.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
                  m2.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
                  m.tradeMatches(m2);
                  assert(m2.howManyMatches() == 2);
            }
      } break; case 64: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            {
                  OnlineDating m2;
                  m2.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
                  m2.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
                  m2.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
                  m.tradeMatches(m2);
                  assert(has(m2, ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]) && has(m2, ARRAYFNAME[1], ARRAYLNAME[1], ARRAYV[1])
&&
                        !m2.someoneAmongMatches(ARRAYFNAME[2],
ARRAYLNAME[2]) && !m2.someoneAmongMatches(ARRAYFNAME[3],
ARRAYLNAME[3]));
            }
      } break; case 65: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
```

```
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            m.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
            m.makeMatch(ARRAYFNAME[4], ARRAYLNAME[4],
ARRAYV[4]);
            {
                  OnlineDating m2;
                  m2.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
                  m2.makeMatch(ARRAYFNAME[4], ARRAYLNAME[4],
ARRAYV[4]);
                  m2.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
                  m.tradeMatches(m2);
                  assert(m.howManyMatches() == 3 &&
m2.howManyMatches() == 5);
            }
      } break; case 66: {
            {
                  OnlineDating m2;
                  m2.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
                  m2.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
                  assert(m2.howManyMatches() == 2 &&
m2.someoneAmongMatches(ARRAYFNAME[1], ARRAYLNAME[1]) &&
!m2.someoneAmongMatches(ARRAYFNAME[2], ARRAYLNAME[3]));
            }
      } break; case 67: {
            {
                  OnlineDating m2;
                  m2.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
                  m2.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
                  m2.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
                  m2.blockPreviousMatch(ARRAYFNAME[1],
ARRAYLNAME[1]);
                  m2.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
                  m2.blockPreviousMatch(ARRAYFNAME[2],
ARRAYLNAME[2]);
                  m2.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
```

```
                    m2.blockPreviousMatch(ARRAYFNAME[0],
ARRAYLNAME[0]);
                    m2.blockPreviousMatch(ARRAYFNAME[3],
ARRAYLNAME[3]);
                    m2.makeMatch(ARRAYFNAME[4], ARRAYLNAME[4],
ARRAYV[4]);
             }
             assert(true);  // no corruption so bad that
destruction failed
      } break; case 68: {
             {
                    OnlineDating m2;
                    m2.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
                    m2.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
                    OnlineDating m3(m2);
                    m3.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
                    m3.blockPreviousMatch(ARRAYFNAME[1],
ARRAYLNAME[1]);
                    m3.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
                    m3.blockPreviousMatch(ARRAYFNAME[2],
ARRAYLNAME[2]);
                    m3.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
                    m3.blockPreviousMatch(ARRAYFNAME[0],
ARRAYLNAME[0]);
                    m3.blockPreviousMatch(ARRAYFNAME[3],
ARRAYLNAME[3]);
                    m3.makeMatch(ARRAYFNAME[4], ARRAYLNAME[4],
ARRAYV[4]);
             }
             assert(true);  // no corruption so bad that
destruction failed
      } break; case 69: {
             {
                    OnlineDating m2;
                    m2.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
                    m2.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
                    m2.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
                    OnlineDating m3(m2);
                    assert(m3.howManyMatches() == 3);
```

```
            }
      } break; case 70: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            {
                  OnlineDating m2(m);
                  m2.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
                  assert(m2.howManyMatches() ==
m.howManyMatches() + 1);
            }
      } break; case 71: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            {
                  OnlineDating m2(m);
                  m2.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
                  assert(m2.howManyMatches() == 4 &&
m2.someoneAmongMatches(ARRAYFNAME[1], ARRAYLNAME[1]) &&
m2.someoneAmongMatches(ARRAYFNAME[3], ARRAYLNAME[3]));
            }
      } break; case 72: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            {
                  OnlineDating m2(m);
                  m2.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
                  assert(m2.howManyMatches() == 4 &&
m2.someoneAmongMatches(ARRAYFNAME[1], ARRAYLNAME[1]) &&
!m2.someoneAmongMatches(ARRAYFNAME[4], ARRAYLNAME[4]));
            }
      } break; case 73: {
            {
```

```
                    OnlineDating m2;
                    m2.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
                    m2.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
                    m2.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
                    OnlineDating m3;
                    m3.makeMatch(ARRAYFNAME[4], ARRAYLNAME[4],
ARRAYV[4]);
                    m3.makeMatch(ARRAYFNAME[5], ARRAYLNAME[5],
ARRAYV[5]);
                    m3 = m2;
                    assert(m3.howManyMatches() == 3 &&
m2.howManyMatches() == 3);
              }
       } break; case 74: {
              m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
              m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
              {
                    OnlineDating m2;
                    m2.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
                    m2.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
                    m2.makeMatch(ARRAYFNAME[4], ARRAYLNAME[4],
ARRAYV[4]);
                    m2 = m;
                    m2.makeMatch(ARRAYFNAME[5], ARRAYLNAME[5],
ARRAYV[5]);
                    assert(m2.howManyMatches() ==
m.howManyMatches() + 1);
              }
       } break; case 75: {
              m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
              m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
              {
                    OnlineDating m2;
                    m2.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
                    m2.makeMatch(ARRAYFNAME[4], ARRAYLNAME[4],
ARRAYV[4]);
```

```
                    m2.makeMatch(ARRAYFNAME[5], ARRAYLNAME[5],
ARRAYV[5]);
                    m2 = m;
                    m2.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
                    assert(m2.someoneAmongMatches(ARRAYFNAME[0],
ARRAYLNAME[0]) &&
                           m2.someoneAmongMatches(ARRAYFNAME[1],
ARRAYLNAME[1]) &&
                           m2.someoneAmongMatches(ARRAYFNAME[2],
ARRAYLNAME[2]) &&
                           !m2.someoneAmongMatches(ARRAYFNAME[3],
ARRAYLNAME[3]));
             }
      } break; case 76: {
             m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
             m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
             {
                    OnlineDating m2;
                    m2.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
                    m2.makeMatch(ARRAYFNAME[4], ARRAYLNAME[4],
ARRAYV[4]);
                    m2.makeMatch(ARRAYFNAME[5], ARRAYLNAME[5],
ARRAYV[5]);
                    m2 = m;
                    m2.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
                    assert(m.someoneAmongMatches(ARRAYFNAME[0],
ARRAYLNAME[0]) &&
                           m.someoneAmongMatches(ARRAYFNAME[1],
ARRAYLNAME[1]) &&
                           !m.someoneAmongMatches(ARRAYFNAME[2],
ARRAYLNAME[2]));
             }
      } break; case 77: {
             {
                    OnlineDating m2;
                    m2.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
                    m2.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
                    m2.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
                    m2 = m2;
```

```
                  assert(m2.howManyMatches() == 3);
                  assert(m2.someoneAmongMatches(ARRAYFNAME[0],
ARRAYLNAME[0]) &&
                        m2.someoneAmongMatches(ARRAYFNAME[1],
ARRAYLNAME[1]) &&
                        m2.someoneAmongMatches(ARRAYFNAME[2],
ARRAYLNAME[2]));
            }
            assert(true);  // no corruption so bad that
destruction failed
      } break; case 78: {
            {
                  OnlineDating m2;
                  m2.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
                  m2.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
                  m2.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
                  m2 = m2;
                  m2.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
                  assert(m2.someoneAmongMatches(ARRAYFNAME[0],
ARRAYLNAME[0]) &&
                        m2.someoneAmongMatches(ARRAYFNAME[1],
ARRAYLNAME[1]) &&
                        m2.someoneAmongMatches(ARRAYFNAME[2],
ARRAYLNAME[2]) &&
                        m2.someoneAmongMatches(ARRAYFNAME[3],
ARRAYLNAME[3]));
            }
      } break; case 79: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            OnlineDating m2;
            OnlineDating m3;
            mergeMatches(m, m2, m3);
            assert(m3.someoneAmongMatches(ARRAYFNAME[0],
ARRAYLNAME[0]) &&
                  m3.someoneAmongMatches(ARRAYFNAME[1],
ARRAYLNAME[1]) &&
                  m3.someoneAmongMatches(ARRAYFNAME[2],
ARRAYLNAME[2]));
```

```
      } break; case 80: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            OnlineDating m2;
            OnlineDating m3;
            mergeMatches(m2, m, m3);
            assert(m3.someoneAmongMatches(ARRAYFNAME[0],
ARRAYLNAME[0]) &&
                  m3.someoneAmongMatches(ARRAYFNAME[1],
ARRAYLNAME[1]) &&
                  m3.someoneAmongMatches(ARRAYFNAME[2],
ARRAYLNAME[2]));
      } break; case 81: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            OnlineDating m2;
            m2.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
            m2.makeMatch(ARRAYFNAME[4], ARRAYLNAME[4],
ARRAYV[4]);
            OnlineDating m3;
            mergeMatches(m, m2, m3);
            assert(m3.someoneAmongMatches(ARRAYFNAME[0],
ARRAYLNAME[0]) &&
                  m3.someoneAmongMatches(ARRAYFNAME[1],
ARRAYLNAME[1]) &&
                  m3.someoneAmongMatches(ARRAYFNAME[2],
ARRAYLNAME[2]) &&
                  m3.someoneAmongMatches(ARRAYFNAME[3],
ARRAYLNAME[3]) &&
                  m3.someoneAmongMatches(ARRAYFNAME[4],
ARRAYLNAME[4]));
      } break; case 82: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
```

```
            OnlineDating m2;
            m2.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
            m2.makeMatch(ARRAYFNAME[4], ARRAYLNAME[4],
ARRAYV[4]);
            OnlineDating m3;
            assert(mergeMatches(m, m2, m3));
      } break; case 83: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            OnlineDating m2;
            m2.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
            m2.makeMatch(ARRAYFNAME[4], ARRAYLNAME[4],
ARRAYV[4]);
            OnlineDating m3;
            m3.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[5]);
            mergeMatches(m, m2, m3);
            assert(m3.someoneAmongMatches(ARRAYFNAME[0],
ARRAYLNAME[0]) &&
                   m3.someoneAmongMatches(ARRAYFNAME[1],
ARRAYLNAME[1]) &&
                   m3.someoneAmongMatches(ARRAYFNAME[2],
ARRAYLNAME[2]) &&
                   m3.someoneAmongMatches(ARRAYFNAME[3],
ARRAYLNAME[3]) &&
                   m3.someoneAmongMatches(ARRAYFNAME[4],
ARRAYLNAME[4]) &&
                   has(m3, ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]) &&
                   !has(m3, ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[5]));
      } break; case 84: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            OnlineDating m2;
            m2.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
```

```
            m2.makeMatch(ARRAYFNAME[4], ARRAYLNAME[4],
ARRAYV[4]);
            OnlineDating m3;
            m3.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[5]);
            assert(mergeMatches(m, m2, m3));
      } break; case 85: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            OnlineDating m2;
            m2.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
            m2.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            OnlineDating m3;
            mergeMatches(m, m2, m3);
            assert(m3.someoneAmongMatches(ARRAYFNAME[0],
ARRAYLNAME[0]) &&
                  m3.someoneAmongMatches(ARRAYFNAME[1],
ARRAYLNAME[1]) &&
                  m3.someoneAmongMatches(ARRAYFNAME[2],
ARRAYLNAME[2]) &&
                  m3.someoneAmongMatches(ARRAYFNAME[3],
ARRAYLNAME[3]));
      } break; case 86: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            OnlineDating m2;
            m2.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
            m2.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            OnlineDating m3;
            assert(mergeMatches(m, m2, m3));
      } break; case 87: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
```

```
            m.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
            OnlineDating m2;
            m2.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            m2.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[4]);
            OnlineDating m3;
            mergeMatches(m, m2, m3);
            assert(m3.someoneAmongMatches(ARRAYFNAME[0],
ARRAYLNAME[0]) &&
                  m3.someoneAmongMatches(ARRAYFNAME[1],
ARRAYLNAME[1]) &&
                  m3.someoneAmongMatches(ARRAYFNAME[2],
ARRAYLNAME[2]));
      } break; case 88: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
            OnlineDating m2;
            m2.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            m2.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[4]);
            OnlineDating m3;
            assert(!mergeMatches(m, m2, m3));
      } break; case 89: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            OnlineDating m2;
            m2.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
            m2.makeMatch(ARRAYFNAME[4], ARRAYLNAME[4],
ARRAYV[4]);
            mergeMatches(m, m2, m);
            assert(m.someoneAmongMatches(ARRAYFNAME[0],
ARRAYLNAME[0]) &&
                  m.someoneAmongMatches(ARRAYFNAME[1],
ARRAYLNAME[1]) &&
```

```
                    m.someoneAmongMatches(ARRAYFNAME[2],
ARRAYLNAME[2]) &&
                    m.someoneAmongMatches(ARRAYFNAME[3],
ARRAYLNAME[3]) &&
                    m.someoneAmongMatches(ARRAYFNAME[4],
ARRAYLNAME[4]));
      } break; case 90: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            OnlineDating m2;
            m2.makeMatch(ARRAYFNAME[3], ARRAYLNAME[3],
ARRAYV[3]);
            m2.makeMatch(ARRAYFNAME[4], ARRAYLNAME[4],
ARRAYV[4]);
            mergeMatches(m, m2, m2);
            assert(m2.someoneAmongMatches(ARRAYFNAME[0],
ARRAYLNAME[0]) &&
                    m2.someoneAmongMatches(ARRAYFNAME[1],
ARRAYLNAME[1]) &&
                    m2.someoneAmongMatches(ARRAYFNAME[2],
ARRAYLNAME[2]) &&
                    m2.someoneAmongMatches(ARRAYFNAME[3],
ARRAYLNAME[3]) &&
                    m2.someoneAmongMatches(ARRAYFNAME[4],
ARRAYLNAME[4]));
      } break; case 91: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            OnlineDating m2;
            m2.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            authenticateMatches(ARRAYFNAME[1], ARRAYLNAME[1], m,
m2);
            assert(!m2.someoneAmongMatches(ARRAYFNAME[0],
ARRAYLNAME[0]) && m2.someoneAmongMatches(ARRAYFNAME[1],
ARRAYLNAME[1]));
      } break; case 92: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
```

```
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            OnlineDating m2;
            authenticateMatches(ARRAYFNAME[1], ARRAYLNAME[1], m,
m2);
            assert(!m2.someoneAmongMatches(ARRAYFNAME[0],
ARRAYLNAME[0]) && m2.someoneAmongMatches(ARRAYFNAME[1],
ARRAYLNAME[1]));
      } break; case 93: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            authenticateMatches(ARRAYFNAME[1], ARRAYLNAME[1], m,
m);
            assert(!m.someoneAmongMatches(ARRAYFNAME[0],
ARRAYLNAME[0]) && m.someoneAmongMatches(ARRAYFNAME[1],
ARRAYLNAME[1]));
      } break; case 94: {
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[2]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[1],
ARRAYV[3]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            OnlineDating m2;
            authenticateMatches("*", ARRAYLNAME[1], m, m2);
            assert(m2.someoneAmongMatches(ARRAYFNAME[0],
ARRAYLNAME[1]) &&
                  m2.someoneAmongMatches(ARRAYFNAME[1],
ARRAYLNAME[1]) &&
                  m2.someoneAmongMatches(ARRAYFNAME[2],
ARRAYLNAME[1]) &&
                  !m2.someoneAmongMatches(ARRAYFNAME[0],
ARRAYLNAME[0]) &&
                  !m2.someoneAmongMatches(ARRAYFNAME[2],
ARRAYLNAME[2]));
      } break; case 95: {
```

```
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[0],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[2]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[2],
ARRAYV[3]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            OnlineDating m2;
            authenticateMatches(ARRAYFNAME[1], "*", m, m2);
            assert(m2.someoneAmongMatches(ARRAYFNAME[1],
ARRAYLNAME[0]) &&
                  m2.someoneAmongMatches(ARRAYFNAME[1],
ARRAYLNAME[1]) &&
                  m2.someoneAmongMatches(ARRAYFNAME[1],
ARRAYLNAME[2]) &&
                  !m2.someoneAmongMatches(ARRAYFNAME[0],
ARRAYLNAME[0]) &&
                  !m2.someoneAmongMatches(ARRAYFNAME[2],
ARRAYLNAME[2]));
      } break; case 96: {
            string all = "*";
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[0],
ARRAYV[0]);
            m.makeMatch(ARRAYFNAME[0], ARRAYLNAME[1],
ARRAYV[1]);
            m.makeMatch(ARRAYFNAME[1], ARRAYLNAME[1],
ARRAYV[2]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[1],
ARRAYV[3]);
            m.makeMatch(ARRAYFNAME[2], ARRAYLNAME[2],
ARRAYV[2]);
            OnlineDating m2;
            authenticateMatches("*", "*", m, m2);
            assert(m2.someoneAmongMatches(ARRAYFNAME[0],
ARRAYLNAME[1]) &&
                  m2.someoneAmongMatches(ARRAYFNAME[1],
ARRAYLNAME[1]) &&
                  m2.someoneAmongMatches(ARRAYFNAME[2],
ARRAYLNAME[1]) &&
                  m2.someoneAmongMatches(ARRAYFNAME[0],
ARRAYLNAME[0]) &&
                  m2.someoneAmongMatches(ARRAYFNAME[2],
ARRAYLNAME[2]));
      } break; case 97: {
```

```
            OnlineDating m2 = m;
            OnlineDating m3;
            authenticateMatches("*", "*", m2, m3);
            assert(m3.howManyMatches() == m.howManyMatches());
      } break; case 98: {
            OnlineDating m2;
            OnlineDating m3(m);
            authenticateMatches("*", "*", m2, m3);
            assert(m3.noMatches());
      } break; case 99: {
            OnlineDating m2;
            authenticateMatches("*", "*", m2, m2);
            assert(m2.howManyMatches() == 0);
      } break; case 100: {
            const int NITEMS = 2000;
            for (int k = 0; k < NITEMS; k++)
                  assert(m.makeMatch(std::to_string(k),
std::to_string(k), SOMEVALUE));
            assert(m.howManyMatches() == NITEMS);
      }
      }
}

int main()
{
    cout << "Enter test number: ";
    int n;
    cin >> n;
    testone(n);
    cout << "Passed" << endl;
}
```