```
// Thanks to former CS 31 TA Andrew Forney for the set of final practice
problems and solutions.
// http://web.cs.ucla.edu/~forns/classes/fall-2014/cs-31/final-
review.html

/*
The following examples are designed to be open-ended in the sense that
they may contain runtime or compile time errors... or may work just
fine! See if you can figure them out.
*/

/* Q1:
Will the following code compile? Is there any undefined behavior? If
"No" to both questions, what will it print out?
*/
  #include <iostream>
  #include <string>
  #include <cstring>
  using namespace std;

  struct builder {
      int yearsExperience,
          yearsAtCompany;
      string name;

      // Interview questions were getting stale...
      char favoriteLetter;

      // Get it? Constructor?
      builder () {
          // Must have 2 years experience to apply
          int yearsExperience = 2;
          int yearsAtCompany = 0;
          string name = "Bob";
          favoriteLetter = 'B';
      }
  };

  int main () {
      builder bob;

      cout << bob.name << endl;
      cout << ((bob.yearsExperience < 2) ? "Veteran" : "Noobuilder") <<
endl;
```

```
  }

/* Q1-solution:
Undefined behavior! bob.yearsExperience is undefined. Look at the
constructor: all of these are local variables being declared, and do not
refer to the data members! Tricky!
*/

//====================================
/* Q2:
Will the following code compile? Is there any undefined behavior? If
"No" to both questions, what will it print out?
*/
  #include <iostream>
  #include <string>
  #include <cstring>
  using namespace std;

  struct builder {
      int yearsExperience,
          yearsAtCompany;
      string name;
      char favoriteLetter;

      builder (int experience, string name) {
          // Must have 2 years experience to apply
          int yearsExperience = experience;
          int yearsAtCompany = 0;
          string name = name;
          favoriteLetter = 'B';
      }

      builder (int atCompany, string name) {
          // Must have 2 years experience to apply
          int yearsExperience = 0;
          int yearsAtCompany = atCompany;
          string name = name;
          favoriteLetter = 'B';
      }
  };

  int main () {
      int experience = 5;
      string name = "Bob";
      builder bob(experience, name);
      cout << bob.name << endl;
      cout << ((bob.yearsExperience < 2) ? "Veteran" : "Noobuilder") <<
endl;
  }

/* Q2-solution:
Compile time error! We can't have two constructors with the same
parameter-types, -orders, and -counts. Our poor compilers will get
confused :(
*/

//====================================
/* Q3:
Find all of the syntax errors in the following code:
(hint: there are 4 errors)
*/
  #include <iostream>
  #include <string>
  #include <cstring>
  using namespace std;

  const int NAME_LEN = 100;

  class Spy {
      int agentId;
      char name[NAME_LEN];
      bool tellsTruth;
      Spy* target;
      Spy* contact;

      Spy (int id, char codeName[], bool truthiness) {
          agentId = id;
          name = codeName;
          tellsTruth = truthiness;
      }
  };

  int main () {
      Spy jamesBond(007, "Bond... James Bond", 1);

      cout << jamesBond.name << endl;
      cout << jamesBond.tellsTruth << endl;
  }
```

```
/* Q3-solution:
  // 1 ----------------------------------
  // Constructor is private! We can't
  // even make Spy objects
  // Fixed:
  public:
     Spy (int id, char codeName[], bool truthiness) { ... }

  // 2 ----------------------------------
  // In constructor: name and codeName
  // are cstrings, so we must use strcpy
  name = codeName;
  // Fixed:
  strcpy(name, codeName);

  // 3, 4 -------------------------------
  // Cannot access private members in the
  // cout statements in main!
  cout << jamesBond.name << endl;
  cout << jamesBond.tellsTruth << endl;
  // To fix it, you can either make the
  // data members public or make getter
  // functions that are public and
  // restrict access
*/

//====================================
/* Q4:
Will the following code compile? Does it involve any undefined behavior?
If "No" to both questions, what will it print out?
*/
  #include <iostream>
  #include <string>
  #include <cstring>
  using namespace std;

  struct dullExample {
      int i;

      dullExample (int j) {
          i = j;
      }
  };

  int main () {
      dullExample* arr[5];

      for (int i = 0; i < 5; i++) {
          arr[i] = new dullExample(i);
      }

      // What is ptr pointing to?
      dullExample* ptr = arr[0];

      for (int j = 0; j < 5; j++) {
          cout << ptr->i << endl;
          ptr++;
      }
  }

/* Q4-solution:
Undefined behavior. The problem is in our loop where we say:
  for (int j = 0; j < 5; j++) {
      cout << ptr->i << endl;

      // [X] Problem here: incrementing ptr is wrong because
      // each array element is a pointer to an object *in
      // the heap* (dynamic memory), which are NOT guaranteed
      // to be contiguous addresses
      ptr++;
  }
Additionally, we do not delete the dynamically allocated dullExample
objects. See problem below for the fixes.
*/

//====================================
/* Q5:
Fix the previous problem to print out the member i of each element of
arr. Also, resolve any memory leaks.
*/

/* Q5-solution:
We can fix the code by making sure we access the pointer at each index
in arr, and then delete it after we're done:
  for (int j = 0; j < 5; j++) {
      // [!] Now, ptr will point to the correct objects
      ptr = arr[j];
      cout << ptr->i << endl;

      // [!] Clean up our dynamic allocation along the way
      delete ptr;
  }
*/

//====================================
/* Q6:
The following code compiles and runs with no error. For each statement
in the main function, determine what is printed out, taking care to
determine what type of object is being printed.
*/
  #include <iostream>
  #include <string>
  #include <cstring>
  using namespace std;

  const int MAX_LENGTH = 50;
  char arr[][MAX_LENGTH] = {
      "ARR",
      "PIRATE",
      "RELATED",
      "JOKE",
      "BECAUSE",
      "ARRRRAY"
  };

  int main () {
      char* ptr1 = arr[0];
      cout << ptr1 << endl;               // #1

      char* ptr2 = ptr1 + 1;
      cout << ptr2 << endl;               // #2
      cout << ptr2[0] << endl;            // #3

      char* ptr3 = *(arr + 2);
      cout << *(ptr3 + 2) << endl;        // #4
      cout << (ptr3 < ptr2) << endl;      // #5
      cout << (ptr3[7] == '\0') << endl;  // #6

      strcpy(ptr1, ptr3 + 2);
      cout << ptr1 << endl;               // #7
      cout << *ptr1 << endl;              // #8
  }

/* Q6-solution:
  1. ARR
  2. RR
  3. R
  4. L
  5. 0
  6. 1
  7. LATED
  8. L
*/

//====================================
/* Q7:
The following function isDoubleString returns true if the input string
is a representation of a possibly white-space-surrounded decimal number
(dictated by the tests in the main function as follows). Someone tried
to solve it and did an insultingly poor job at it. Below is their
attempt to solve it. Your task is twofold:

1. Although this code DOES compile, find all of the errors that lead to
undefined behavior or other problems at runtime.
2. Fix these errors. Now, determine which test case this "working" code
will fail.
*/
  #include <iostream>
  #include <string>
  #include <cctype>
  #include <cassert>
  using namespace std;

  bool isDoubleString (string s);

  int main () {
      assert(  isDoubleString("1.23")); // True
      assert(  isDoubleString("1")); // True
      assert(  isDoubleString("  1.23  ")); // True
      assert(  isDoubleString(".23")); // True
      assert(! isDoubleString("  1.  23  ")); // False
      assert(! isDoubleString("1.2.3.4")); // False
      assert(! isDoubleString("I IZ DUBLE :DDD")); // False
      assert(! isDoubleString("1a.23")); // False
      assert(! isDoubleString("a1.23")); // False
      assert(! isDoubleString("")); // False

      cerr << "[!] ALL TESTS PASSED!" << endl;
  }
  bool isDoubleString (string s) {
      // Set up flags, which will see if we've found a digit
      // or decimal yet
      bool foundFirstDigit = false,
           foundDecimal = false;

      int i;
      while (i < s.length()) {
          // Case where we've seen a digit
          if (isdigit(s[i])) {
              foundFirstDigit = true;

          // Case where we've seen a decimal point
          } else if (s[i] == '.') {
              if (foundDecimal) {
                  // Case where we've seen 2 decimals
                  return false;
              }
              foundDecimal = true;

          // Case where we've seen a space
          } else if (s[i] == ' ') {
              continue;

          // Catch-all for bad chars
          } else {
              return false;
          }

          i++;
      }

      // If we reach this point, we have a legal double-string so long
as
      // at least one digit was found
      return foundFirstDigit;
  }
/* Q7-solution:
1. The programmer forgot to take care of his/her iterator! See portions
marked with [X] below:
  bool isDoubleString (string s) {
      // Set up flags, which will see if we've found a digit
      // or decimal yet
      bool foundFirstDigit = false,
           foundDecimal = false;
      // [X] Iterator i was uninitialized
      int i = 0;
      while (i < s.length()) {
          // Case where we've seen a digit
          if (isdigit(s[i])) {
              foundFirstDigit = true;

          // Case where we've seen a decimal point
          } else if (s[i] == '.') {
              if (foundDecimal) {
                  // Case where we've seen 2 decimals
                  return false;
              }
              foundDecimal = true;

          // Case where we've seen a space
          } else if (s[i] == ' ') {
              // [X] Continuing here without an iterator
              // increment will enter an infinite loop!
              i++;
              continue;

          // Catch-all for bad chars
          } else {
              return false;
          }

          i++;
      }

      // If we reach this point, we have a legal double-string so long
as
      // at least one digit was found
      return foundFirstDigit;
  }

2. Even with these fixes, the code will fail the test: assert(!
isDoubleString(" 1. 23 ")); // False because it assumes spaces should
simply be ignored.
*/

//====================================
/* Q8:
Your friend Yenrof was very tired while coding late into the night, and
```

```
attempted to create a function that reverses a substring in a cstring.
Remedy their blatant incompetence by finding their error (and proposing
a solution, which may modify any element of the code) below:
*/
  #include <iostream>
  #include <string>
  #include <cctype>
  #include <cassert>
  using namespace std;

  char* reverseSubstring (char c[], int start, int count);

  int main () {
      char c1[] = "art";
      assert(!strcmp(reverseSubstring(c1, 0, 1), "rat"));
      char c2[] = "pretty";
      assert(!strcmp(reverseSubstring(c2, 1, 1), "pertty"));
      char c3[] = "howdy!";
      assert(!strcmp(reverseSubstring(c3, 0, 1000), "!ydwoh"));
      cout << "[!] PASSED ALL UNIT TESTS!" << endl;
  }

  // Returns a pointer to the cstring c after all elements from
  // index start to (start + count) have been reversed
  char* reverseSubstring (char c[], int start, int count) {
      int endIndex = start + count,
          len = strlen(c);

      char* iter = &c[start];
      char* holder;

      // Put a ceiling on the endIndex such that it is no
      // greater than the cstring length
      if (endIndex >= len) {
          endIndex = len - 1;
      }

      // As long as our iterator has a smaller address than
      // the end index, we'll keep swapping elements
      while (iter < &c[endIndex]) {
          holder = &c[endIndex];
          *iter = c[endIndex--];
          *holder = *iter;
          iter++;
      }
      return c;
  }

/* Q8-solution:
Yenrof's error was in overwriting the char at *iter with the one at
c[endIndex] before swapping it into the holder position!
  // Returns a pointer to the cstring c after all elements from
  // index start to (start + count) have been reversed
  char* reverseSubstring (char c[], int start, int count) {
      int endIndex = start + count,
          len = strlen(c);

      char* iter = &c[start];

      // [!] Change holder to a char type, not char*
      char holder;

      // Put a ceiling on the endIndex such that it is no
      // greater than the cstring length
      if (endIndex >= len) {
          endIndex = len - 1;
      }

      // As long as our iterator has a smaller address than
      // the end index, we'll keep swapping elements
      while (iter < &c[endIndex]) {
          // [!] holder now preserves the iterator's element
          // while we swap the endIndex char with the one
          // at iter
          holder = *iter;
          *iter = c[endIndex];
          c[endIndex--] = holder;
          iter++;
      }

      return c;
  }
*/

//====================================
/*
Use the following over-elaborate class and description for exercise Q9 ~
Q16.
*/
```

```
/*
The British spy agency MI6 learned that you've nearly completed CS31 and
have
entrusted you with managing their spy tracking system. Your predecessor
left
you the following class for these purposes before he met his... untimely
termination.

The class tracks each agent / contact's system id, name, whether or not
they tell
the truth when asked their name, and any pointers to OTHER spys that
might be that
spy's agency contact or that spy's target.
*/
  #include <iostream>
  #include <string>
  #include <cstring>
  using namespace std;

  const int NAME_LEN = 100;

  class Spy {
      int agentId;
      char name[NAME_LEN];
      bool tellsTruth;
      Spy* target;
      Spy* contact;

  public:
      Spy (int id, char codeName[], bool truthiness) {
          agentId = id;
          strcpy(name, codeName);
          tellsTruth = truthiness;
          target = nullptr;
          contact = nullptr;
      }

      // Sets the contact of a given Spy. If that Spy
      // already has a contact, they betray them, setting
      // their previous contact as their new target!
      void setContact (Spy* con) {
          // Betrayal most foul!
          if (contact != nullptr) {
              target = contact;
          }
          contact = con;
      }

      // Calls setTarget for this Spy using the target
      // of this Spy's contact.
      void getTargetFromContact () {
          this->setTarget(contact->target);
      }

      // Sets the target of this spy to the given Spy
      // input. BUT if the input target is this Spy's
      // contact, we will flip their truth telling
      // status to the opposite of what it currently is.
      void setTarget (Spy* tar) {
          // Yet more betrayal!
          if (tar == contact) {
              tellsTruth = !tellsTruth;
          }
          target = tar;
      }

      // How the *caller* would reply if asked his / her name.
      // If they're not trustworthy, they will give the name of
      // the best James Bond actor of all time </sarcasm>
      void giveName () {
          cout << ((tellsTruth) ? name : "Timothy Dalton") << endl;
      }

      Spy* getTarget () {
          return target;
      }

      Spy* getContact () {
          return contact;
      }
  };

//************************************
/* Q9:
Using the above class definition, find all of the syntax / runtime
errors in the following main functions:
*/
  int main () {
      Spy jamesBond(007, "Bond... James Bond", 1);
      Spy blofeld(100, "Blofeld", 0);
      jamesBond.setTarget(blofeld);
      blofeld.setTarget(jamesBond);
  }
/* Q9-solution:
  int main () {
      Spy jamesBond(007, "Bond... James Bond", 1);
      Spy blofeld(100, "Blofeld", 0);

      // [X] setTarget takes *pointers* to Spy objects,
      // not Spy objects themselves
      jamesBond.setTarget(blofeld);
      blofeld.setTarget(jamesBond);
  }
*/

//=====================================
/* Q10:
Using the above class definition, find all of the syntax / runtime
errors in the following main functions:
*/
  int main () {
      Spy jamesBond(007, "Bond... James Bond", 1);
      Spy blofeld(100, "Blofeld", 0);

      jamesBond.setTarget(&blofeld);

      // Trust no one but yourself!
      jamesBond.setContact(&jamesBond);
      jamesBond.getTargetFromContact();

      jamesBond.getTarget().giveName();
  }
/* Q10-solution:
  int main () {
      Spy jamesBond(007, "Bond... James Bond", 1);
      Spy blofeld(100, "Blofeld", 0);

      jamesBond.setTarget(&blofeld);
      jamesBond.setContact(&jamesBond);
      jamesBond.getTargetFromContact();

      // [X] getTarget() returns a pointer to a Spy,
      // which means if we wanted that Spy to give their
      // name, we must use the -> notation, e.g.:
```

```
      // jamesBond.getTarget()->giveName();
      jamesBond.getTarget().giveName();
  }
*/

//====================================
/* Q11:
Using the above class definition, find all of the syntax / runtime
errors in the following main functions:
*/
  int main () {
      Spy jamesBond(007, "Bond... James Bond", 1);
      Spy blofeld(100, "Blofeld", 0);

      jamesBond.setTarget(&blofeld);
      blofeld.getTargetFromContact();
      blofeld.getTarget()->giveName();
  }
/* Q11-solution:
  int main () {
      Spy jamesBond(007, "Bond... James Bond", 1);
      Spy blofeld(100, "Blofeld", 0);

      jamesBond.setTarget(&blofeld);
      blofeld.getTargetFromContact();

      // [X] blofeld has no contact, so after his
      // getTargetFromContact(), target = nullptr,
      // meaning we're asking nullptr to give us its
      // name (runtime error)
      blofeld.getTarget()->giveName();
  }
*/

//====================================
/* Q12:
The following main function compiles and runs just fine; what will it
print out?
*/
  int main () {
      Spy jamesBond(007, "Bond... James Bond", 1);
      Spy leChiffre(100, "LeChiffre", 0);
      Spy Q('Q', "Q", 1);

      jamesBond.setContact(&Q);
      Q.setTarget(&leChiffre);
      jamesBond.getTargetFromContact();

      jamesBond.getContact()->giveName();
      jamesBond.getTarget()->giveName();
  }
/* Q12-solution:
  Q
  Timothy Dalton
*/

//====================================
/* Q13:
The following main function compiles and runs just fine; what will it
print out?
*/
  int main () {
      Spy jaws(0, "Jaws", 0);

      // He was never the smartest henchman...
      jaws.setTarget(&jaws);
      jaws.setContact(&jaws);
      jaws.getTargetFromContact();

      jaws.getTarget()->giveName();
  }
/* Q13-solution:
  Jaws
*/

//====================================
/* Q14:
The following main function compiles and runs just fine; what will it
print out?
*/
  int main () {
      Spy jamesBond(007, "Bond... James Bond", 1);
      Spy leChiffre(100, "LeChiffre", 0);
      Spy vesper(0, "Vesper", 0);

      // NB: I will not answer for any transgressions against
      // actual James Bond plots... this is my practice final dammit!
      jamesBond.setContact(&leChiffre);
      vesper.setContact(&jamesBond);
      vesper.setTarget(&leChiffre);
      leChiffre.setTarget(&jamesBond);
      jamesBond.setContact(&vesper);

      jamesBond.getTarget()->giveName();
      vesper.getTarget()->getTarget()->giveName();
  }
/* Q14-solution:
  Timothy Dalton
  Bond... James Bond
*/

//====================================
/* Q15: (If you can figure out this one... well done.)
The following main function compiles and runs just fine; what will it
print out?
*/
  int main () {
      Spy jamesBond(007, "Bond... James Bond", 1);
      Spy blofeld(100, "Blofeld", 0);
      Spy ninja1(0, "Ninja 1", 1);
      Spy ninja2(0, "Ninja 2", 0);

      // Ninjas hired...
      ninja1.setContact(&blofeld);
      ninja2.setContact(&blofeld);
      blofeld.setTarget(&jamesBond);

      // Betrayal!
      ninja1.setTarget(&blofeld);
      ninja2.setContact(&ninja1);
      ninja2.getTargetFromContact();

      ninja1.giveName();
      ninja2.giveName();
      ninja1.getTarget()->giveName();
      ninja2.getTarget()->giveName();
  }
/* Q15-solution:
  // (AKA: best James Bond cast ever)
  Timothy Dalton
  Timothy Dalton
  Timothy Dalton
  Timothy Dalton
*/
```

```
//====================================
/* Q16:
Using the Spy class definition we defined earlier in the exam, implement
a new class SpyRing that has the following interface:

- Private members:

Spy* m_ring[MAX_SPIES]; // the internal array of pointers to spies;
assume MAX_SPIES is a const int defined in scope (you may assign to it
whatever value you please).

Spy* leader; // the leader of the spy ring; the leader is also within
m_ring

int m_nSpies; // tracking count for the number of spies in the ring

- Public interface:

SpyRing (); // SpyRing default constructor that creates an empty SpyRing
with no leader and no spies.

~SpyRing (); // SpyRing destructor that makes sure to delete any
dynamically allocated Spies in the ring.

bool addSpy (char name[], bool truthy, bool isLeader); // create a new
Spy in the SpyRing with the name and truth-telling attributes of the
input. Set isLeader to true if this newly created Spy is the leader of
the SpyRing. Use m_nSpies to set the agentId of the Spy. Return true if
there is enough room in the SpyRing to insert the Spy, otherwise, return
false and do not modify the SpyRing.

void setContact (int agentId, Spy* contact); // sets the contact of the
SpyRing spy with the given agentId to the input Spy* contact. (assume
agentId corresponds with the m_ring index)

void setTarget (int agentId, Spy* target); // sets the target of the
SpyRing spy with the given agentId to the input Spy* target. (assume
agentId corresponds with the m_ring index)

void issueHit (); // sets each member of the spy ring's target to that
of the leader. Do nothing if there is no leader.

int findLoyal (); // returns the number of spies in the ring that have
the leader as their contact. Return -1 if there is no leader.
```

```
*/

/* Q16-solution:

// private member and class structure solution.

  const int MAX_SPIES = 10;

  class SpyRing {
  private:
      Spy* m_ring[MAX_SPIES];
      Spy* leader;
      int m_nSpies;
  };


// constructor / destructor solutions.

  class SpyRing {
  private:
      Spy* m_ring[MAX_SPIES];
      Spy* leader;
      int m_nSpies;

  public:
      SpyRing ();
      ~SpyRing ();
  };

  SpyRing::SpyRing () {
      leader = nullptr;
      m_nSpies = 0;
  }

  SpyRing::~SpyRing () {
      for (int i = 0; i < m_nSpies; i++) {
          delete m_ring[i];
      }
  }

// addSpy, setContact, setTarget solutions.

  // ... class definition omitted above

  bool SpyRing::addSpy (char name[], bool truthy, bool isLeader) {
      // Only add a spy if there's room
      if (m_nSpies < MAX_SPIES) {
          // Add that spy to the m_ring array
          m_ring[m_nSpies] = new Spy(m_nSpies, name, truthy);

          // Set them to the leadership status if necessary
          if (isLeader) {
              leader = m_ring[m_nSpies];
          }

          // Increase the spy count and return true, indicating
          // that we successfully added the spy
          m_nSpies++;
          return true;
      } else {
          return false;
      }
  }

  void SpyRing::setContact (int agentId, Spy* contact) {
      m_ring[agentId]->setContact(contact);
  }

  void SpyRing::setTarget (int agentId, Spy* target) {
      m_ring[agentId]->setTarget(target);
  }

// issueHit solution.

  void SpyRing::issueHit () {
      // If no leader, do nothing
      if (leader == nullptr) {
          return;
      }

      // Otherwise, we'll get the leader's target, and set
      // everyone else's target to that!
      Spy* target = leader->getTarget();
      for (int i = 0; i < m_nSpies; i++) {
          m_ring[i]->setTarget(target);
      }
  }

// findLoyal solution.
```

```
  // These spies aint loyal...
  int SpyRing::findLoyal () {
      // No leader? Return -1
      if (leader == nullptr) {
          return -1;
      }

      // Otherwise, compare the pointers of each Spy's contact
      // to see if it's equivalent to the leader; track how many
      // meet this criteria and return it at the end
      int loyal = 0;
      for (int i = 0; i < m_nSpies; i++) {
          if (m_ring[i]->getContact() == leader) {
              loyal++;
          }
      }

      return loyal;
  }
*/


//======================================
/* (END OF EXERCISES)
THANK YOU VERY MUCH!
GOOD LUCK FOR THE FINALS! (^-^)
*/
```