Question 1

```
/*  1 */ class MyClass {
/*  2 */   public:
/*  3 */     MyClass();
/*  4 */     MyClass *Copy();
/*  5 */     MyClass(const MyClass &mc);
/*  6 */     void A(Otherclass &s) const;
/*  7 */     void B(Otherclass s);
/*  8 */     void C(Otherclass &s);
/*  9 */     void D(const Otherclass &s);
/* 10 */     int F;
/* 11 */   protected:
/* 12 */     vector <int> v;
/* 13 */ };
```

Behold the declaration of the **MyClass** class to the right. Please answer the following true/false questions:

A. The copy constructor is declared on line 3.
B. The copy constructor is declared on line 4.
C. The copy constructor is declared on line 5.
D. The method **A()** cannot change **s**.
E. The method **B()** cannot change **s**.
F. The method **C()** cannot change **s**.
G. The method **D()** cannot change **s**.
H. The method **A()** cannot change **v**.
I. The method **B()** cannot change **v**.
J. The method **C()** cannot change **v**.
K. The method **D()** cannot change **v**.
L. When you call **A()**, you will call **Otherclass**' copy constructor.
M. When you call **B()**, you will call **Otherclass**' copy constructor.
N. When you call **A()**, you will call **Otherclass**' regular constructor.
O. When you call **B()**, you will call **Otherclass**' regular constructor.
P. When **A()** returns, it will call the destructor for **s**.
Q. When **B()** returns, it will call the destructor for **s**.
R. This header won't compile, because **F** should be protected.
S. If **x** and **y** are declared as **MyClass** variables, I'm allowed to say "**x=y**", even though I didn't declare an assignment overload.
T. There is a memory leak, because there is no destructor to free up the memory corresponding to the variable **v**.
U. If **Copy()** calls **new**, I should have a destructor that calls **delete**.

Answers and explanations are in https://web.eecs.utk.edu/~jplank/plank/classes/cs202/Tests/2019-Fall/t1/answers.html

CS140 Midterm Exam - October 15, 2019 Answers and Grading

James S. Plank

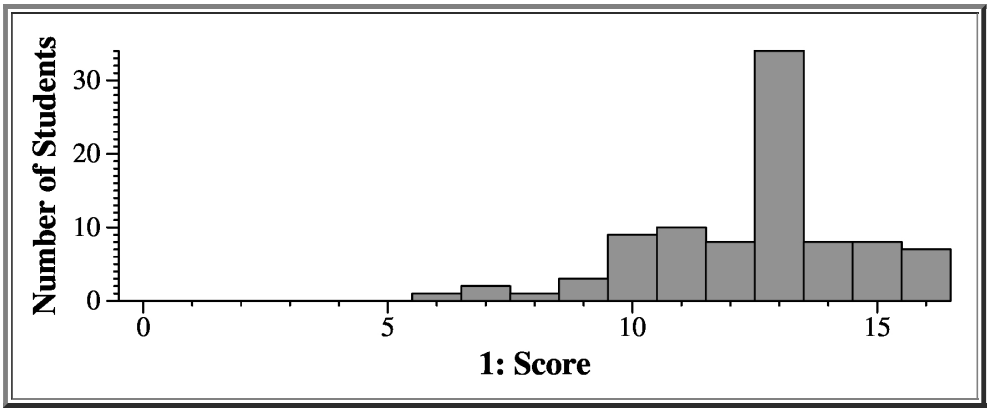
Question 1: 16 Points

- Part *A*: The copy constructor is on line 5. **False**
- Part *B*: The copy constructor is on line 5. **False**
- Part *C*: The copy constructor is on line 5. **True**
- Part *D*: The reference parameter is not declared **const**, so the method can do whatever it wants to *s*. **False**
- Part *E*: The parameter is not declared **const**, so the method can do whatever it wants to *s*. **False**
- Part *F*: The reference parameter is not declared **const**, so the method can do whatever it wants to *s*. **False**
- Part *G*: The reference parameter is declared **const**, so the method cannot change *s*. **True**
- Part *H*: The method is declared **const**, so it cannot change *v*. **True**
- Part *I*: The method is declared declared **const**, so it do whatever it wants to *v*. **False**
- Part *J*: The method is declared declared **const**, so it do whatever it wants to *v*. **False**
- Part *K*: The method is declared declared **const**, so it do whatever it wants to *v*. **False**
- Part *L*: Since **s** is a reference parameter, there is no constructor called when **A()** is called, as **A()** simply gets a reference to the caller's **s**: **False**
- Part *M*: Since **s** is not a reference parameter, it will make a copy of the caller's parameter, and that will use the copy constructor: **True**.
- Part *N*: See part *L*: No constructor is called: **False**
- Part *O*: See part *M*: The copy constructor is called: **False**
- Part *P*: Since nothing was constructed, nothing needs to be destructed. **False**
- Part *Q*: The copy needs to be destructed when the method returns: **True**
- Part *R*: Variables may be public: **False**
- Part *S*: This is fine -- it simply calls a default assignment operator, which will make copies of **F** and **v**: **True**
- Part *T*: Vectors and strings are destroyed automatically. **False**
- Part *U*: This is identical to the **Copy()** method in the bitmatrix lab. It is up to the caller to free the pointer. You'll note, there's nowhere to store the pointer in the class definition, so it can't delete anything: **False**

Grading:

0.75 points for parts *A* through *Q*, and parts *S* through *U*. 1 point for part *R*. Here are counts of correct answers (91 students):

```
A: 89 F      B: 44 F      C: 55 T      D: 68 F      E: 61 F
F: 86 F      G: 89 T      H: 72 T      I: 85 F      J: 86 F
K: 81 F      L: 68 F      M: 67 T      N: 60 F      O: 55 F
P: 78 F      Q: 77 T      R: 89 F      S: 64 T      T: 81 F
U: 73 T
```



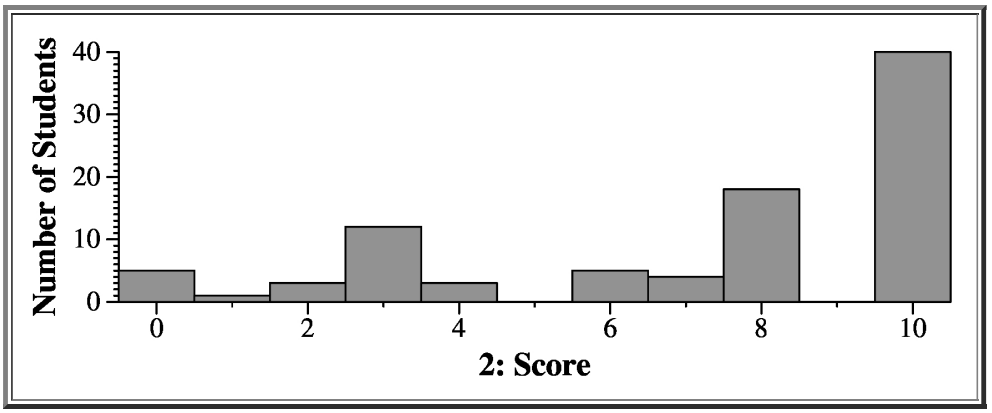
Question 2: 10 Points

I know, no one likes to trace through code, but how else to get you to demonstrate to me that you understand the basics of **try/catch**? Let's walk through it:

- At the first **cout** statement, *i* is 0, so: **0**.
- Since 0 is even, we don't throw, so *i* becomes 3 and we skip the catch clause: **3**.
- We add two to *i*, which becomes 5, and go back to the top of the while loop. It prints: **5**.
- Now, five is odd, so we throw it. The **catch** clause prints 5 again, and then adds 5 + 5 to set 5 to 10. After the **catch** clause, it prints: **10**.
- At the end of the while loop, it increments *i* by two again, to turn it into 12. The while loop exits, and we print: **12**.

So the answer is: `0, 3, 5, 10, 12`

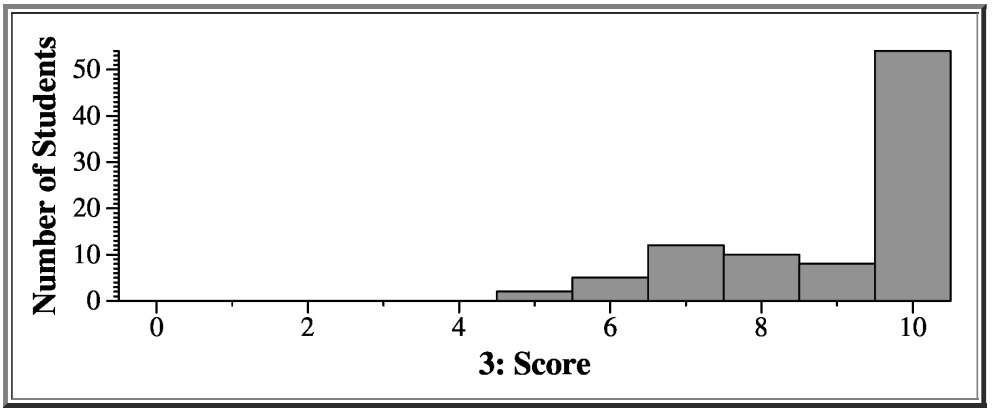
Grading: 2 points per correct answer, -0.75 per incorrect answer. Minimum = 0.



Question 3: 10 Points

- Part *A*: 512 + 16 + 4: 0010 0001 0100 = 0x214.
- Part *B*: Split it into the four-bit units: 010 0101 0010 = 0x252.
- Part *C*: 4*16 + 8 = 64+8 = 72.
- Part *D*: A four-bit shift simply shifts the hex digits: 0x480.

Grading: 2.5 points per answer.



Question 4: 16 Points

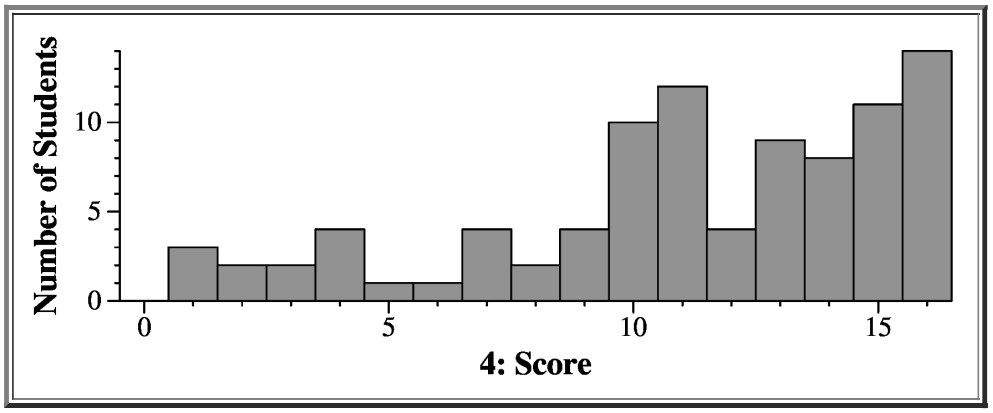
Since the table size is 16, the last hex digit is the hash index.

- Part *A*: There are 10/16 entries full, so the load factor is 10/16.
- Part *B*: Austin Prissy's index is 9, which is empty, so the answer is 9.
- Part *C*: Joshua Polonium's index is 5. Five is taken, so we check $5+1^2 = 6$. That is taken, so we check $5+2^2 = 9$. That is empty, so the answer is 9.
- Part *D*: Maya Paddy's index is 0xe = 14. 14 is taken, so we'll need to use the second hash value, which is 0xf = 15. This means that we'll be looking at successively smaller indices -- 13, 12, 11, 10, and finally 9.
- Part *E*: Noah Porous' index is 5. Five is taken, so we'll look at successively larger indices -- 6, which is taken, and then 7, which is empty. The answer is 7.
- Part *F*: Alexander Aeronautic's index is 0xe = 14, which is taken. $14+1^2 = 15$, which is taken. $14+2^2 = 2$, which is taken. $14+3^2 = 7$, which is empty -- the answer is 7.
- Part *G*: Riley Predecessor's index is 1. 1 is taken, so we'll need to use the second hash value, which is 4. 5 is taken, but 9 is empty, so the answer is nine.
- Part *H*: Since 16 is not a prime number, there will be values of the second hash function which won't test all of the indices. 8 is the easiest of these, because it only tests two indices. So, if the first hash function is 2, and the second is 8, then we'll only look at indices 2 and 10. So those are good examples. There are a lot of correct answers:
  - H1 = 2, 5, 6, 10, 13 or 14 and H2 = 8.
  - H1 = 2, 6, 10, or 14 and H2 = 4 or 12.

- Part *I*: Both Madison Willoughby and Aiden Nepal hash to 0xc = 12. Since Aiden Nepal is actually stored at index 12, that means index 12 was empty when he was inserted. Since Madison Willoughby is not stored at 12, it means that she was inserted after Aiden Nepal.

Grading:

- 1 point for part *A*.
- 1.5 points for each of parts *B* through *G*.
- 2 points for parts *H*. If you gave a value of 0 for H2 in part *H*, then you only received half credit, because we have to handle 0 in a special manner (remember your lab -- we turned zeros into ones).
- Four points for part *I*.



Question 5: 14 Points

This is a question that tests argc/argv, stringstreams and stdout/stderr. Here's an example answer:

```
#include <iostream>
#include <sstream>
using namespace std;

int main(int argc, char **argv)
{
  istringstream ss;
  int i, j;

  for (i = 1; i < argc; i++) {
    ss.clear();                 // You have to clear the stringstream between uses.
    ss.str(argv[i]);
    if (ss >> j) {              // The boolean is true only if the string represents an integer.
      cout << j << endl;
      return 0;
    }
  }
  cerr << "No integer" << endl; // If we exit the for loop, we haven't found an integer.
  return 0;
}
```

I didn't care if you started the loop at 0 or 1, since I didn't specify in the problem description.

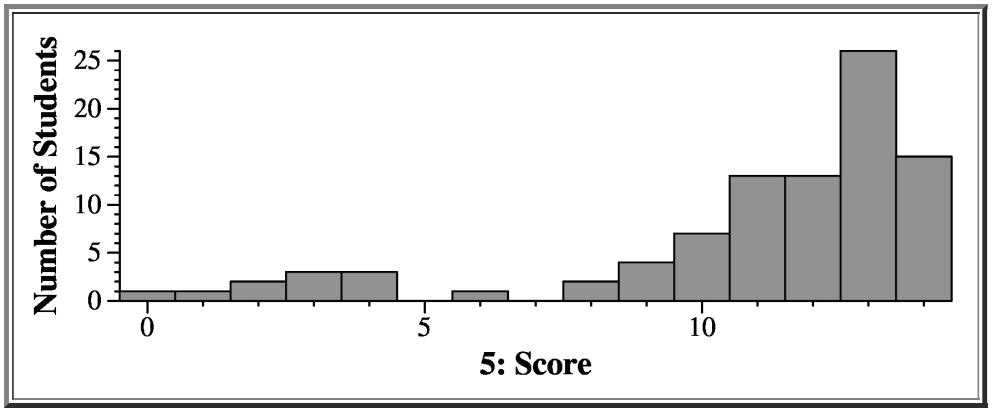
Grading:

The grading for this question (and most of the coding questions), uses the following rubric: Either you start with zero points (and your grading file will say "Please see the answer") and then you get points for things that are correct. Or, you start with 14 points, and then receive deductions for things that are incorrect.

Common deductions:

- Minorly flawed **main()** declaration: -0.5
- Majorly flawed (or omitted) **main()** declaration: -1
- No return or exit statement: -1
- Don't use stderr for "No integer": -0.7
- Don't clear the stringstream: -0.5
- Don't return after the first integer: -1.5
- Bad loop indexing: -1
- There are others.

Regardless, if you mixed code and variable declarations (with the exception of declaring an integer inside a for loop), you received a 1 point deduction.



Question 6: 14 Points

You can solve this in a variety of ways. Here are a few:

```
void vreverse(vector <int> &v)
{
  size_t i;
  int j;

  for (i = 0; i < v.size()/2; i++) {
    j = v[i];
    v[i] = v[v.size()-i-1];
    v[v.size()-i-1] = j;
  }
}
```

```
void vreverse(vector <int> &v)
{
  vector <int> v2;
  int i;

  for (i = v.size()-1; i >= 0; i==) {
    v2.push_back(v[i]);
  }
  v = v2;
}
```

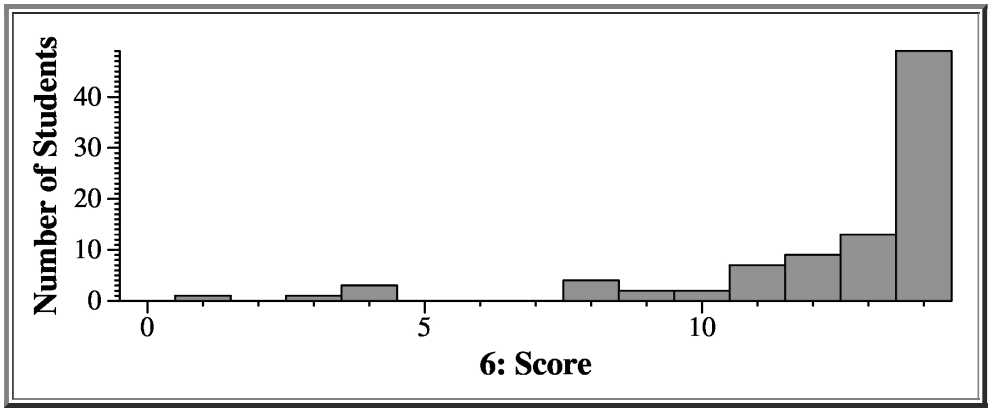
```
void vreverse(vector <int> &v)
{
  deque <int> q;
  int i;

  for (i = 0; i < v.size(); i++) {
    q.push_front(v[i]);
  }
  v.clear();
  for (i = 0; i < q.size(); i++) {
    v.push_back(q[i]);
  }
}
```

Personally, I think that the first is the best.

Grading:

The grading is like the previous question. Here are the major deductions -- there are others:

- Implemented swap incorrectly: -2
- Printed the reversed list rather than setting the reference parameter: -3
- Forget to set the parameter to the temporary vector at the end: -2
- Didn't reverse the vector, because you went through all of the vector rather than half: -3
- Implemented an n^2 algorithm: -4
- Off by one error (either in the swap or while indexing): -2
- Forget to set the parameter to the temporary vector at the end: -2
- Since you declared the variable inside the for loop, its value will always be zero: -3
- Segmentation violation: -2



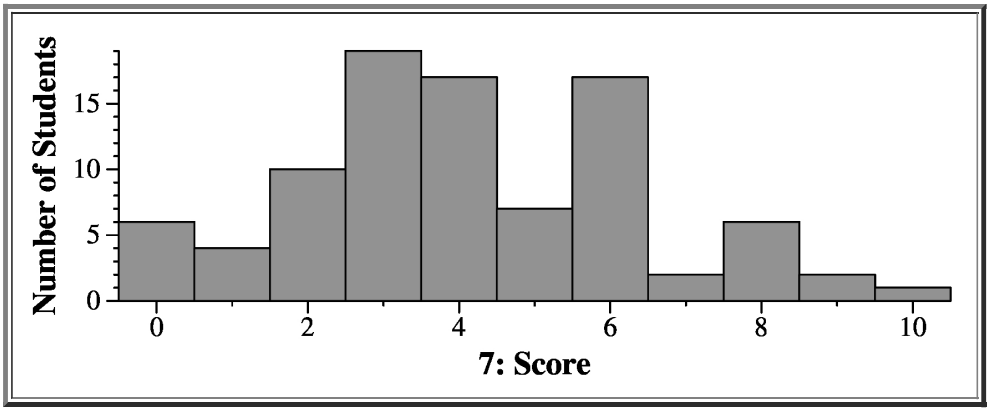
Question 7: 10 Points

Here's a correct answer: **qlist.cpp**

```
int dmiddle(list <string> *l)
{
  list <string>::const_iterator lit, lit2;
  size_t i;

  if (l->size() == 0) return 0;
  lit = l->begin();
  for (i = 0; i < l->size()/2; i++) lit++;
  if (l->size() % 2 == 0) {
    lit2 = lit;
    lit2--;
    l->erase(lit);
    l->erase(lit2);
    return 2;
  } else {
    l->erase(lit);
    return 1;
  }
}
```

The grading for this is like the last question. Here are the major deductions -- there were others.

- Bad return value on parameter declaration: -4
- Conflating iterators and integers: -4
- Assumed that there is a middle() method that returns the middle iterator of a list: -4
- Did not use iterators to find the middle of the list: -4
- Tried to use an iterator after erasing its element: -2



Question 8: 13 Points

```
/* Part A */
bool Data::Print_By_Decade(int decade)
{
  size_t i;

  if (decade < 0 || decade >= Decades.size()) return false;
  for (i = 0; i < Decades[decade].size(); i++) {
    cout << Decades[decade][i]->name << endl;
  }
  return true;
}

/* Part B -- you need to delete the things that you have allocated with new.
   Those are the People.  So, simply run through the All vector and delete
   the people. */

Data::~Data()
{
  size_t i;

  for (i = 0; i < All.size(); i++) delete All[i];
}
```

```
/*  Or you can do it with Decades, but that is more of a pain:

Data::~Data()
{
  size_t i, j;

  for (i = 0; i < Decades.size(); i++) {
    for (j = 0; j < Decades[i].size(); j++) delete Decades[i][j];
  }
}
*/
```

Part C: Since **Print_By_Decade()** doesn't change anything, you should declare it as **const**. So, change line 11 to "bool Print_By_Decade(int decade) const;"

Part D: You handle errors in constructors by throwing exceptions The best exceptions to throw are strings. You need to throw the exception before the **new** call, to avoid a memory leak. So, either before line 25 or 26, add "if (a < 0) throw ((string) "Data::Data() - Bad age.".

That's actually an incomplete answer. You should really run through **All** and delete all of the people, too. However, I accepted the above as an answer.

Grading:

Part A is like the other coding questions, with a maximum of 5 points.

Part B is worth three points. You received 1.8 points of you deleted the pointers twice (traversing **All** and **Decades**). You received 1 point if you deleted **Decades[i]**.

Part C is worth two points. Here are the answer keys and their values:

- OK: 2 points
- EC: Good answer, but extraneous consts: 1.5 points
- NC: Not correct: 0 points
- NA: No answer: 0 points

Part D was 3 points, and I treated it like the coding questions, but with deductions. The following were all one-point deductions:

- Continue instead of stopping the function
- Using exit instead of throw
- Memory leak -- they put the if statement after "new Person" and don't delete person.
- No print statement or string throw
- Only print something, but no return/exit/throw/continue
- Return/break instead of throwing an exception
- Putting the error on a weird or wrong line number, but handling the error correctly otherwise