CS 2B Midterm Exam

Due Feb 10 at 11:59pm **Points** 20 **Questions** 20
Available Feb 10 at 6am - Feb 10 at 11:59pm about 18 hours **Time Limit** 60 Minutes

Instructions

Finish this test before midnight. Once you begin, you will have 60 minutes to complete it. If you do not submit before that time, your incomplete exam will be automatically submitted as is. You can't take this test in multiple sessions, so do not start it unless you have protected time in which to take the test.

You can look at lectures or texts, or use IDEs, but not consult any other individuals or non-course help/ask sites for help. Reference sites are fine.

Each question is worth 1 point.

This quiz was locked Feb 10 at 11:59pm.

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	60 minutes	17.27 out of 20

ⓘ Correct answers will be available on Mar 24 at 12am.

Score for this quiz: **17.27** out of 20
Submitted Feb 10 at 2:09pm
This attempt took 60 minutes.

Question 1 — 1 / 1 pts

Consider the following 2-D array allocation:

```
double * myArray[3];
int k;

for (k = 0; k < 3; k++)
   myArray[k] = new double[5];
```

Assume that we deallocate in a sensible manner using simple loop structures that would work no matter how large the array was. Check *all* the true statements:

- [ ] It is a completely dynamic array having 3 rows and 5 columns.
- [ ] Memory freeing will require a double-nested loop.
- [ ] It is a completely dynamic array having 5 rows and 3 columns.
- [x] It is a partially dynamic array having 3 rows and 5 columns.
- [ ] It is a partially dynamic array having 5 rows and 3 columns.
- [x] Memory freeing will require one un-nested loop.

To be a completely dynamic 2-D array, (which it is not) it would have had to be declared as **double **myArray**.

It only requires one loop to de-allocate. In fact, even if it had been a completely dynamic array, it would still only require one loop (plus a final delete for the master column) but no a double nested loop.

Partial

Question 2

0.6 / 1 pts

Here are some class and method prototypes, labeled ONE, TWO, THREE and FOUR which are involved in four operator overloading, two as member operators, two as non-members:

```
class classA
{

public:
```

```
   string operator+( classA  other );  // method ONE
   int  operator+(int num);            // method TWO

};

string operator+(classA firstA, string secondString);   // m
ethod THREE
string operator+(string firstString, classA secondA);   // m
ethod FOUR
```

(Assume no other operators are defined which involve this class.)

Match each of the client (**main()**) operator invocations to the method that it calls, or choose **ILLEGAL/NOT SHOWN** if it is illegal or it is legal but attempts to call an operator which is not defined above. Variable names **firstA** and **secondA** are also assumed to be defined in **main()** as objects of **classA**:

METHOD TWO	cout << firstA + 5 << (
ILLEGAL/NOT SHOWN	cout << "Hello Foothil
METHOD ONE	cout << "Hello Foothil
METHOD THREE	cout << firstA + "Hellc
METHOD FOUR	cout << firstA + secor

"Hello Foothill" + 5 will compile but it is not going to call one of the four defined operators.

Question 3

1 / 1 pts

Assume you are working inside the method definition of some instance method of **class B** - you are writing the statements that define this method. In order to call an *instance method* of a **class A** from inside this instance method of a **class B** (where **B** and **A** are unrelated by inheritance to one-another), you:

- ... can call the method without any object used for dereferencing.
- ... must use an object of **class A** for dereferencing the method call.
- ... must use an object of **class B** for dereferencing the method call.
- ... must use an object of a third class, a base class of both class A and class B, for dereferencing.

By definition, an instance method of A requires and instantiated object of A to call it, unless we are already inside a method of the same class (which the question ruled out since B and A were unrelated).

Question 4

1 / 1 pts

A constructor is used to:

- ... initialize the instance members of a class.
- ... declare the member variables of a class.
- ... initialize both instance and static members of a class.
- ... initialize the static members of a class.

No class members can be declared in a constructor. Static members are initialized either outside the class at global scope or by a separate static method, which might be called by the constructor, but would not be repeatedly executed every time the constructor is called, since that would constantly and inadvisedly overwrite the value of the static member. Only instance members are initialized in a constructor.

Question 5

1 / 1 pts

In which type of class methods is it typically necessary to filter bad arguments (either directly or indirectly with the help of validators) in order to protect private data of your class?

- [ ] A **display()** (or **show()**) method, which takes no parameters, intended to output the data of the class for the display.
- [x] A Constructor that takes parameters.
- [x] A **set()** method (non-constructor mutator).
- [ ] A **toString()** method, which takes no parameters, intended to return some private data to the client in the form of a string.
- [ ] A **get()** method (non-constructor accessor).

set() methods and Constructors are forms of mutators and typically require that the arguments passed to them be filtered so that no bad data is assigned to the class's private members.

Display methods, accessors or mutators that don't take parameters, do not assign private data that originates from a client, so no filtering would be needed.

Partial

Question 6

0.67 / 1 pts

Consider the following code fragment (assumed to be in an otherwise correct program):

```
double *dubPtr;
double dubArray[100];

dubPtr = new double[50];
dubPtr = dubArray;
dubPtr[75] = 9;
delete[] dubPtr;
```

Assume there are no omitted statements within this fragment -- you see everything. Check the true statements (may be more than one):

- [x] Illegally attempts to delete non-dynamic memory.
- [ ] It will compile without error but probably crash (i.e., generate a run-time error).
- [ ] It has no compiler or runtime errors, although it is a little odd looking.
- [ ]

- [x] Some allocated memory becomes inaccessible, and therefore lost for further use in main(), while the program is still executing.

- [ ] It has an array bounds violation, possibly causing a fatal runtime error.

- [ ] It has a compiler error.

The program's errors are of the run-time variety, not detectable by a compiler. After the line **dubPtr = dubArray;**, the newly allocated 50 doubles is no longer reachable, and lost forever. However, **dubPtr[75]** is perfectly fine, since it refers to location 75 in **dubArray[]**, which has 100 locations. The **delete** statement, while fine if **dubPtr** had still pointed to its new-ed 50 doubles, is now illegal since **dubPtr** no longer points to those elements. Now it ended up pointing to **dubArray[]** which cannot be deleted. This illegally attempts to delete non-dynamic memory and will likely cause a runtime error (crash).

Question 7

1 / 1 pts

Assume MyClass defines two nested exception classes, BadNumber and BadString. Also, assume that MyClass's method doSomething() throws these exceptions under certain error conditions, and returns without throwing anything when it decides no error has occurred. We call doSomething four times in a try/catch block:

```
MyClass  a;

try
{
   a.doSomething();
```

```
    a.doSomething();
    a.doSomething();
    a.doSomething();
}
catch (MyClass::BadNumber)
{
    cout << "\n bad number \n"  << endl;
}
catch (MyClass::BadString)
{
    cout << "\n bad string \n"  << endl;
}
catch (...)
{
    cout << "\n just generally bad\n" << endl;
}
```

Check the true statements (there will be more than one correct answer).

[Ignore whether spaces or '\n' are displayed - it is the ***words*** of the **cout** statements that we are interested in.]

- [ ] It is possible ***both*** "bad number" and "bad string" is displayed on the screen after this block.
- [x] It is possible that **doSomething()** will return ***without error*** in ***one or more*** of these invocations, yet we ***still have one*** of "bad number", "bad string" or "just generally bad" displayed on the screen after this block.
- [x] It is possible that this block will be called without anything being displayed on the screen.
- [x] It is possible ***neither*** "bad number" ***nor*** "bad string" is displayed on the screen after this block.
- [x] It is possible ***none of*** "bad number", "bad string" or "just generally bad" is displayed on the screen after this block.

Until **doSomething()** throws an exception, it will continue to be called up to four times. So even if it fails at some point, it may return without error and later result in a throw, which prints our one of our error messages.

We can never get more than one error message, since the first throw will stop the method calls and go to a catch block.

If there is no error in any of the calls, no error message will be displayed.

Question 8

1 / 1 pts

If we operate ***bitwise*** on the following two ***decimal integers***, as indicated, what are the two resulting decimal answers?

9 bitwise-***or*** 7
9 bitwise-***and*** 7

- 12
  1
- 12
  2
- 15
  1
- 16
  1
- 15
  2

Represented in binary, 9 is 1001 and 7 is 0111.

1001 | 0111 = 1111 (or 15)

and

1001 & 1001 = 0001 (or 1)

Incorrect

Question 9

0 / 1 pts

The statements

```
Card *card1, *card2, *card3, myCard;
card1 = new Card;
card2 = card1;
```

will cause how many card objects to be instantiated? (only one correct choice):

- 2
- 0
- 4 (selected)
- 3
- 1

`myCard` is instantiated immediately, and a second **`Card`** object is instantiated in the **`new`** statement (but further pointers to that object do not cause additional instantiation).

Question 10

1 / 1 pts

Which of the following are good candidates for ***instance variables*** for a class that you would define?

- Variables (like loop counters or temporary variables) used by most or all of your instance methods (in an attempt to avoid redeclaring these variables multiple times as locals for each instance method).
- Constants used by the class and the client.
- Private data whose values could be different for different objects of the class.
- Variables whose values are shared by all class objects.

Only data whose values change from object to object would be a good candidate for instance members. Shared data and constants are always static, never instance. Local and helper variables should never be made instance variables simply to avoid declaring them in multiple methods. This would create very bad design and result in confusion between the methods.

Question 11

1 / 1 pts

A ***linear search algorithm*** is written (as in the modules, for example) which searches an array for some user-defined value, **clientData**. If **clientData** is stored in the array, it returns its array position, and if not found, it returns -1 (again, just like in the modules). Assume the array to be searched has 100 data elements in it. (Check *all* that apply):

[NOTE: due to common off-by-one interpretations when counting such things, if your predicted answer is within one (+1 or -1) of a posted option below, you can assume your prediction and the choice you are looking at are equivalent and check that option.]

- [x] It ***will never require more than 100*** comparisons of **data** before it returns.
- [ ] **It *will always* return with an answer *in 50 or fewer* comparisons of data.**
- [x] It ***may* require *as many as 100*** comparisons of **data** before it returns.
- [x] **It *might* return to the client with an answer after *only one* comparison of data.**
- [ ] **It *will always* return with an answer *in 10 or fewer* comparisons of data.**

The data may be in position 0, in which case it only requires one comparison (of data), so it may return after about one comparison. At the other extreme, the data may be in position 99, or not in the array at all. In those cases it will require about 100 comparisons (of data), but never more. So 100 is the most it would require.

Incorrect

Question 12

0 / 1 pts

`ClassA` has a method, `void methodA(ClassA aObj)`, which takes, as a parameter, an object of the same `ClassA`.

Check all the true statements. (Check *all* that apply):

- [x] **If `methodA()` modifies a private member of *the parameter object,* `aObj`, it will result in a simultaneous change of the corresponding private member in the *calling object* (the *this* object), even if `aObj` is a different object than the calling object.**

- [ ] If `methodA()` modifies a private member of its ***calling object*** (the ***this*** object), it will result in a simultaneous change of the corresponding private member in the parameter object, `aObj`, even if `aObj` is a different object than the calling object.

- [ ] **`methodA()` can access private data of `aObj` directly, as in `aObj.somePrivateMember = something`, without the need for a public mutator or accessor.**

- [ ] **`methodA()` can access private data of its *calling object* (the *this* object) directly, as in `somePrivateMember = something`, without the need for a public mutator or accessor and without the need to dereference anything.**

Member methods can access private data of objects of the same class. This applies to both the ***calling object*** as well as any objects that it may receive ***as a parameter***. In the case of the calling object, no dereference is necessary. However, the calling object and the parameter object ***may be different*** -- and usually are: there is no need to pass a calling object as a parameter. The problem tells you to consider that they are distinct, in which case changes to one do not affect changes in the other.

Question 13

1 / 1 pts

In the following numbers, we use (D) to denote ordinary decimal notation, and (H) to denote hexadecimal notation:

```
FF (H)
11 (H)
 5 (D)
11 (D)
```

Which of the following are the correct binary representation of these four numbers?

- ```
  11001100
  10001000
       101
      1101
  ```

- ```
  11111111
     10001
       101
      1011
  ```

```
11111111
   10001
   10000
    1011
```

```
11111111
   10001
     101
      11
```

```
11001100
   10001
     101
    1101
```

-

Question 14

1 / 1 pts

A program can be written in a messy and inefficient style and still compile without any errors and run flawlessly.

- (•) True
- ( ) False

Unfortunately, it is true that a poorly written and sloppily designed program can compile and run without errors. However, if and when that program has to be modified by another programmer, such a perfect "working" program shows its weakness, not to mention the weakness of the programmer.

Question 15

1 / 1 pts

A binary search of a pre-sorted array of 256 elements would take (at most) how many comparisons? (Assume the search key is the key on which the array is sorted).

[I allow for off-by-one errors in this problem, so if your prediction is within one of the posted choice below, choose it.]

- (●) 9
- ( ) 4
- ( ) 256
- ( ) 128
- ( ) 1

Each time a comparison is done in the ***binary search***, half the list is eliminated, thereby forcing a find by no more than the ninth comparison.

Question 16

1 / 1 pts

A ***binary search algorithm*** is written (as in the modules, for example) which searches a ***pre-sorted*** array for some user-defined value, **clientData**. If **clientData** is stored in the array, it returns its array position, and if not found, it returns -1 (again, just like in the modules). Assume the array to be searched has ***100 data elements*** in it. (Check *all* that apply):

[NOTE: due to common off-by-one interpretations when counting such things, if your predicted answer is within one (+1 or -1) of a posted option below, you can assume your prediction and the choice you are looking at are equivalent and check that option.]

- [ ] It ***may*** **require** ***as many as 99*** comparisons of **data** before it returns.
- [x] It ***will always*** return with an answer ***in 7 or fewer*** comparisons of **data.**
- [x] It ***might*** return to the client with an answer after ***only one*** comparison of **data.**
- [ ] It ***will always*** return with an answer ***in 3 or fewer*** comparisons of **data.**

Each recursive call throws away ***half of the elements*** it gets. The method gets 100 elements, but if **clientData** is not found in position 49 ***(first comparison)*** the method throws 51 of those elements away and calls itself. This inner call now has 49 elements to search. It tests element 24 against **clientData** ***(second comparison)***, and if it is not a match, throws about away half again, leaving 24. The ***next call/comparison***, if needed, throws away about 12. The ***next call***, if needed, throws away 6. The ***next call*** throws away 3. The ***next call*** throws away 2. Finally we only have ***one element to test***. ***Each recursive call does one comparison***. How many comparisons have we done, give or take one, (count above) in the worst case, assuming we have to go all the way down until we only have one element before we find the data we are searching for?

Question 17

1 / 1 pts

Check the ***true*** statement(s)
(There is at least one correct choice, possibly more.)

- [x] An ***instance method*** can access a ***static member*** (of the same class) inside its definition using ***only the member name***, no object or class name prepended (i.e., w**ithout ClassName::** or **someObj**. in front).

- [ ] A ***static method*** can access an ***instance member*** (of the same class) inside its definition using only the member name, no object or class name prepended (i.e., *without* **someObj.** in front).

- [x] An ***instance method*** can access an ***instance member*** (of the same class) inside its definition using ***only the member name***, no object or class name prepended (i.e., w**ithout ClassName::** or **someObj**. in front).

- [x] A ***static method*** can access a ***static member*** (of the same class) inside its definition using ***only the member name***, no object or class name prepended (i.e., w**ithout ClassName::** or **someObj**. in front).

Static methods cannot access instance members unless there is some explicit object inside that method to dereference. It makes to sense, otherwise, since the static method has no instance members. All other choices are true.

Question 18

1 / 1 pts

If **varA** and **varB** are two instantiated object variables of some class, then **varB = varA** copies all the (shallow) data members from **varA's** object over to **varB's** object. Assume that = has not been overloaded (i.e., has its default meaning).

- (•) True
- ( ) False

In C++, assigment statements between object variables result in a transference of the memer data from one to the other.

Question 19

1 / 1 pts

Because there is no illegal value for a particular class's ***instance*** member **xCoordinate** and ***static*** member **newOriginX**, both **xCoordinate** and

newOriginX are declared **public**, allowing the client to use them directly. Consider the client code (where the two objects are members of the class in question):

```
evilVillain_1.xCoordinate = 3;
evilVillain_1.newOriginX = -1.4;
evilVillain_2.xCoordinate = 5;
evilVillain_2.newOriginX = -2.2;
```

Immediately after this code, what are the values of the instance and static members?

- **evilVillain_1** has **xCoordiante** = 3
  evilVillain_2 has **xCoordiante** = 5
  the class's static, if accessed through **evilVillain_1.newOrigin**, = -2.2
  the class's static, if accessed through **evilVillain_2.newOrigin**, = -2.2

- **evilVillain_1** has **xCoordiante** = 5
  evilVillain_2 has **xCoordiante** = 5
  the class's static, if accessed through **evilVillain_1.newOrigin**, = -2.2
  the class's static, if accessed through **evilVillain_2.newOrigin**, = -2.2

- **evilVillain_1** has **xCoordiante** = 3
  evilVillain_2 has **xCoordiante** = 5
  the class's static, if accessed through **evilVillain_1.newOrigin**, = -1.4
  the class's static, if accessed through **evilVillain_2.newOrigin**, = -2.2

- **evilVillain_1** has **xCoordiante** = 3
  evilVillain_2 has **xCoordiante** = 5
  the class's static, if accessed through **evilVillain_1.newOrigin**, = -1.4
  the class's static, if accessed through **evilVillain_2.newOrigin**, = -1.4

Although it is not common, a client can use an object, rather than a class name, to get at a static member.

The instance members are distinct as assigned, but the static member only holds the most recent value.

Question 20

1 / 1 pts

If an instance method calls another instance method of the same class, an object must be used for dereferencing the call.

- True
- False

Once an instance method is entered from some client, all further calls to other instance members from within that method would not use any object to dereference the call. The implied "this" object is always used for nested method calls. In some situations, an instance method might declare or use extra, non-***this***, objects for use in lower-level calls, but this is not typical and it is not something that "must" be done all the time.

Quiz Score: **17.27** out of 20