Week 3 Quiz

LATEST SUBMISSION GRADE

42%

1. Which one of these is the only true statement about a class constructor.   **1 / 1 point**

    ○ A class must have at least one constructor declared for it.

    ○ A class can only have one constructor.

    ○ When declaring a constructor for a class, the return type of the constructor must be
       the type of the class.

    ◉ When declaring a constructor for a class, the name of the constructor must be the
       name of the class.

    ✓ **Correct**
       A constructor cannot be assigned a name other than that of the class itself.

2. Which of the following examples does NOT call a copy constructor at least once?   **0 / 1 point**

    (If you already have advanced knowledge of C++ that makes this seem like a trick
    question, then we'll also specify this: Assume that compiler optimizations are mostly
    disabled.)

    ○
    ```
    1    Cube a,b(10);
    2    a = b;
    ```

    ○
    ```
    1    Cube b(10);
    2    Cube a = b;
    ```

    ○
    ```
    1    // Function prototype for "intersect":
    2    Cube intersect(Cube &left, Cube &right);
    3    // ...
    4    Cube a(10),b(5);
    5    Cube c;
    6    c = intersect(a,b);
    7
    ```

    ○
    ```
    1    // Function prototype for "contains":
    2    int contains(Cube outer, Cube inner);
    3    // ...
    4    Cube a(10),b(5);
    5    int a_bounds_b = contains(a,b);
    6
    ```

> **❗ Incorrect**
> You didn't select an answer.

3.Recall that a custom assignment operator can be declared such that line 2 of the code below executes a user-defined function to perform the assignment.   **1 / 1 point**

```
1    Cube a,b(10);
2    a = b(10);
```

Which one of the following statements regarding the declaration of such a custom assignment operator allowing is true?

- ○ The custom assignment operator function is declared with two arguments: the source and target objects of the assignment.

- ○ The custom assignment operator is a function declared with the name "operator::assignment".

- ○ The type of the custom assignment operator function should be void.

- ◉ The custom assignment operator is a public member function of the class.

> **✓ Correct**
> Indeed, the assignment operator is a member function of the class of the target object, and this member function must be public in order for the assignment operator to be used in contexts outside of the class's own implementation as shown in the example in this question.

4.Consider the following class: **0.4 / 1 point**

```
 1    class Orange {
 2      public:
 3        Orange(double weight);
 4        ~Orange();
 5        double getWeight();
 6
 7      private:
 8        double weight_;
 9    };
10
```

Select **all** functions that are present in this class (including any automatic/implicit functions added by the compiler):

- ☑ Default constructor

    This should not be selected

!

☑ At least one custom, non-default constructor

     ✓ **Correct**

☐ Copy constructor

☐ Assignment operator

☑ Destructor

     ✓ **Correct**

5. Consider the following class: **0.8 / 1 point**

```
1    class Blue {
2      public:
3        double getValue();
4        void setValue(double value);
5
6      private:
7        double value_;
8    };
9
```

Select **all** functions that are present in this class (including any automatic/implicit functions added by the compiler):

☑ Default constructor

     ✓ **Correct**

☐ At least one custom, non-default constructor

☐ Copy constructor

☑ Assignment operator

     ✓ **Correct**

☑ Destructor

✓ **Correct**

You didn't select all the correct answers

6.Consider the following class: **0 / 1 point**

```
1    class Animal {
2      public:
3        Animal();
4        Animal(std::string name);
5        Animal(std::string name, int age);
6        Animal(std::string name, int age, double weight);
7
8        Animal(const Animal & other);
9
10       void setName(std::string name);
11       std::string getName();
12
13     private:
14       // ...
15    };
16
```

How many **explicit** (non-automatic) **constructors** are present in the class?

○ 2

○ 3

◉ 4

○ 5

○ 6

○ 7

! **Incorrect**

7. When you use the **new** operator to create a class object instance in heap memory,   **0 / 1 point**
    the **new** operator makes sure that memory is allocated in the heap for the object, and
    then it initializes the object instance by automatically calling the class constructor.

    After a class object instance has been created in heap memory with **new**, when is
    the *destructor* usually called?

    ○ The destructor is called automatically when the program returns from the function
       where the **new** operator was used to create the class object instance.

○ The destructor is called automatically when the **delete** operator is used with a pointer to the instance of the class.

◉ The programmer always needs to call the destructor manually in order to free up memory.

○ The destructor is called automatically when the variable goes out of scope.

> **! Incorrect**
>
> This is incorrect. There are some specific expert-level situations in C++ where the programmer may call the destructor manually, but it's beyond the scope of this course. In general, you should not call the destructor manually.

8. Consider the following program:

1 / 1 point

```
1    double magic(uiuc::Cube cube) {
2      cube.setLength(1);
3      return cube.getVolume();
4    }
5
6    int main() {
7      uiuc::Cube c(10);
8      magic(c);
9      return 0;
10   }
11
```

How many times is the **uiuc::Cube** 's copy constructor invoked?

○ Never

◉ Once

○ Twice

○ Three times

> **✓ Correct**

9. We have looked at examples where the assignment operator returned the value "*this". The variable "this" is available by default in most class member functions. What is the value of this built-in class variable "this"?

0 / 1 point

○ A reference to the current object.

◉ A pointer to a heap-memory copy of the current object.

○ An alias of the current object.

○ A pointer to the current object instance.

! **Incorrect**

C++ does not automatically create a heap-memory copy of an object, so "this"
always pertains to the current object, not a copy.

10. Consider the code below that includes a class that has a custom constructor and destructor and both utilize a   **0 / 1 point**
global variable (which has global scope and can be accessed anywhere and initialized before the function main
is executed).

```
1    int reference_count = 0;
2
3    class Track {
4    public:
5        Track() { reference_count++; }
6        ~Track() { reference_count--; }
7    };
```

Which one of the following procedures (void functions) properly ensures the deallocation of all the memory
allocated for objects of type Track so the memory can be re-used for something else after the procedure
returns?

For the correct answer, the variable reference_count should be zero after all calls to track_stuff() **and** all of the
memory should be deallocated properly. This will dependably occur after only one of the following procedures.

○
```
1    void track_stuff() {
2        Track *t = new Track;
3        // ...
4        t->~Track();
5        return;
6    }
```

○
```
1    void track_stuff() {
2        Track t;
3        Track *p = new Track;
4        // ...
5        delete p;
6        return;
7    }
```

◉
```
1    void track_stuff() {
2        Track t;
3        // ...
4        delete t;
5        return;
6    }
```

○
```
1    void track_stuff() {
2        Track t;
3        Track *p = &t;
4        // ...
5        delete p;
6        return;
7    }
```

! **Incorrect**

This will trigger a compile error. This procedure declares t as an object of class Track with its memory allocated on the stack. The delete keyword only deallocates memory on the heap, not the stack, and it must be used on a pointer.