**Test Corrections**

Anja Sheppard AMS180001

7. Given the code segment below, which of the following statements would evoke the copy constructor of the class?

Points: 4/6

Correct answer comes from: MindTap 12-10c

The equals sign is overloaded to copy one DragonRider object to another using the copy constructor.

//function prototypes

void DragonRider(Viking);

Viking DragonTrainer();

int main()

{

Viking Astrid;

Viking \* Tuffnut;

//Which statement below would evoke the copy constructor?

return 0;

}

1. **Astrid = DragonTrainer();**
2. Tuffnut = &Astrid;
3. Tuffnut = new Viking;
4. DragonRider(Astrid);
5. **Viking Ruffnut = Astrid;**
6. Tuffnut = &Astrid;

DragonRider(\*Tuffnut);

12. Identify the errors in the following code segment.  Select all that apply.

Points: 2.5/3

Correct answer comes from: MindTap 7-1

You can’t have the same keyword in multiple enums as it gets confusing to the compiler which enum you are referring to.

enum Dragon{TOOTHLESS, STORMFLY, HOOKFANG, MEATLUG, BARF\_BELCH} Trained;

enum {BARF\_BELCH, SKULLCRUSHER, GRUMP, CLOUDJUMPER};

enum Valka{AUROARA, HOOKFANG, BEWILDERBEAST, WINDSHEAR};

Dragon Alpha = STORMFLY;

Trained = Alpha;

int X = WINDSHEAR;

1. declaring a Dragon variable named Trained
2. **Having BARF\_BELCH and HOOKFANG in multiple enums**
3. declaring an enum without a name
4. assigning STORMfly to a Dragon variable
5. assigning one Dragon variable to another Dragon variable
6. assigning WINDSHEAR to an integer variable

13. Overloaded operator functions can be overloaded. For example, there can be more than one += function in a class.

Points: 0/2

Correct answer comes from: MindTap 13-2i

This is true, for example we overload ++ for post-fix and pre-fix addition.

1. **True**
2. False

17. Based on the code segment below, which of the following items will create a syntax error.  Select all that apply.

Points: 0/3

Correct answer comes from: MindTap 10-7

They can’t be declared outside the class definition because there can’t be any conflicts between what the value is for different instantiations. Static variables stay the same for all instantiations of a class.

class Viking

{

public:

//hidden constructors, destructors, accessors and mutators

static int village\_population;

static void population\_growth(int x)

{ x += 5; village\_population = x; }

};

Viking::Viking = 0;

int main()

{

Viking::village\_population = 100;

Viking::population\_growth(Viking::village\_population);

Viking Hiccup;

Hiccup.population\_growth(20);

return 0;

}

1. defining a static class variable
2. the definition of the static class funtion
3. **declaring the static variable outside of the class definition**
4. assigning a value to the static variable in main
5. calling the static function in main without an object
6. calling the static function in main with an object

20. What is the output of the following code segment?  Assume all necessary libraries have been included.

Points: 0/3

Correct answer comes from: MindTap 11-1

Rider() isn’t virual, so it will always print Snotlout. Trainer() is virtual so it will print whatever is in the derived class (Astrid).

|  |
| --- |
| #include <iostream>  using namespace std;  class Dragon  {  public:  void Rider()  {  cout << "Snotlout";}  virtual void Trainer()  {  cout << "Fishlegs";}  };    class Toothless : public Dragon  {  public:  virtual void Rider()  {  cout << "Hiccup";}  void Trainer()  {  cout << "Astrid";}  };    int main()  {  Toothless T;  Dragon\* D = &T;  D->Rider();  cout << " ";  D->Trainer();  return 0;  } |

1. **Snotlout Astrid**
2. Hiccup Astrid
3. Snotlout Fishlegs
4. Hiccup Fishlegs

21. Identify the syntax errors in the following code segment.  Select all that apply.  Ignore the missing preprocessor directives.

Points: 2.667/4

Correct answer comes from: MindTap 10-1i, 10-1k

The class specifier Dragon:: goes in front of the function name, not in front of the return type. Also, it should return a de-referenced this pointer.

//header file

class Dragon

{

string \* diet = new string[3];

public:

    string feedDragon() const;

};

//source file

Dragon::string feedDragon() const

{

    diet[0] = "fish";

return this;

}

1. lack of specifier for the class variable
2. the feedDragon() prototype in the header file
3. **the feedDragon() function header in the source file**
4. assigning a value to the array in the feedDragon() function
5. **the return statement in the feedDragon() function**
6. No syntax errors