**Quiz time**

**Question #1**

a) Write a class named Point2d. Point2d should contain two member variables of type double: m\_x, and m\_y, both defaulted to 0.0. Provide a constructor and a print function.

The following program should run:

#include <iostream>

int main()

{

Point2d first{};

Point2d second{ 3.0, 4.0 };

first.print();

second.print();

return 0;

}

COPY

This should print:

Point2d(0, 0)

Point2d(3, 4)

[Show Solution](javascript:void(0))

b) Now add a member function named distanceTo that takes another Point2d as a parameter, and calculates the distance between them. Given two points (x1, y1) and (x2, y2), the distance between them can be calculated as std::sqrt((x1 - x2) \* (x1 - x2) + (y1 - y2) \* (y1 - y2)). The std::sqrt function lives in header cmath.

The following program should run:

#include <iostream>

int main()

{

Point2d first{};

Point2d second{ 3.0, 4.0 };

first.print();

second.print();

std::cout << "Distance between two points: " << first.distanceTo(second) << '\n';

return 0;

}

COPY

This should print:

Point2d(0, 0)

Point2d(3, 4)

Distance between two points: 5

[Show Solution](javascript:void(0))

c) Change function distanceTo from a member function to a non-member friend function that takes two Points as parameters. Also rename it “distanceFrom”.

The following program should run:

#include <iostream>

int main()

{

Point2d first{};

Point2d second{ 3.0, 4.0 };

first.print();

second.print();

std::cout << "Distance between two points: " << distanceFrom(first, second) << '\n';

return 0;

}

COPY

This should print:

Point2d(0, 0)

Point2d(3, 4)

Distance between two points: 5

[Show Solution](javascript:void(0))

**Question #2**

Write a destructor for this class:

#include <iostream>

class HelloWorld

{

private:

char\* m\_data{};

public:

HelloWorld()

{

m\_data = new char[14];

const char\* init{ "Hello, World!" };

for (int i = 0; i < 14; ++i)

m\_data[i] = init[i];

}

~HelloWorld()

{

// replace this comment with your destructor implementation

}

void print() const

{

std::cout << m\_data << '\n';

}

};

int main()

{

HelloWorld hello{};

hello.print();

return 0;

}

COPY

[Show Solution](javascript:void(0))

**Question #3**

Let’s create a random monster generator. This one should be fun.

a) First, let’s create an enumeration of monster types named MonsterType. Include the following monster types: Dragon, Goblin, Ogre, Orc, Skeleton, Troll, Vampire, and Zombie. Add an additional max\_monster\_types enum so we can count how many enumerators there are.

[Show Solution](javascript:void(0))

b) Now, let’s create our Monster class. Our Monster will have 4 attributes (member variables): a type (MonsterType), a name (std::string), a roar (std::string), and the number of hit points (int). Create a Monster class that has these 4 member variables.

[Show Solution](javascript:void(0))

c) enum MonsterType is specific to Monster, so move the enum inside the class as a public declaration. When the enum is inside the class, “MonsterType” can be renamed “Type” since the context is already Monster.

[Show Solution](javascript:void(0))

d) Create a constructor that allows you to initialize all of the member variables.

The following program should compile:

int main()

{

Monster skeleton{ Monster::Type::skeleton, "Bones", "\*rattle\*", 4 };

return 0;

}

COPY

[Show Solution](javascript:void(0))

e) Now we want to be able to print our monster so we can validate it’s correct. To do that, we’re going to need to write a function that converts a Monster::Type into a string. Write that function (called getTypeString()), as well as a print() member function.

The following program should compile:

int main()

{

Monster skeleton{ Monster::Type::skeleton, "Bones", "\*rattle\*", 4 };

skeleton.print();

return 0;

}

COPY

and print:

Bones the skeleton has 4 hit points and says \*rattle\*

[Show Solution](javascript:void(0))

f) Now we can create a random monster generator. Let’s consider how our MonsterGenerator class will work. Ideally, we’ll ask it to give us a Monster, and it will create a random one for us. We don’t need more than one MonsterGenerator. This is a good candidate for a static class (one in which all functions are static). Create a static MonsterGenerator class. Create a static function named generateMonster(). This should return a Monster. For now, make it return anonymous Monster(Monster::Type::skeleton, "Bones", "\*rattle\*", 4);

The following program should compile:

int main()

{

Monster m{ MonsterGenerator::generateMonster() };

m.print();

return 0;

}

COPY

and print:

Bones the skeleton has 4 hit points and says \*rattle\*

[Show Solution](javascript:void(0))

g) Now, MonsterGenerator needs to generate some random attributes. To do that, we’ll need to make use of this handy function:

// Generate a random number between min and max (inclusive)

// Assumes srand() has already been called

static int getRandomNumber(int min, int max)

{

static constexpr double fraction{ 1.0 / (static\_cast<double>(RAND\_MAX) + 1.0) }; // static used for efficiency, so we only calculate this value once

// evenly distribute the random number across our range

return static\_cast<int>(std::rand() \* fraction \* (max - min + 1) + min);

}

COPY

However, because MonsterGenerator relies directly on this function, let’s put it inside the class, as a static function.

[Show Solution](javascript:void(0))

h) Now edit function generateMonster() to generate a random Monster::Type (between 0 and Monster::Type::max\_monster\_types-1) and a random hit points (between 1 and 100). This should be fairly straightforward. Once you’ve done that, define two static fixed arrays of size 6 inside the function (named s\_names and s\_roars) and initialize them with 6 names and 6 sounds of your choice. Pick a random name and roar from these arrays.

The following program should compile:

#include <ctime> // for time()

#include <cstdlib> // for rand() and srand()

int main()

{

std::srand(static\_cast<unsigned int>(std::time(nullptr))); // set initial seed value to system clock

std::rand(); // If using Visual Studio, discard first random value

Monster m{ MonsterGenerator::generateMonster() };

m.print();

return 0;

}

COPY

[Show Solution](javascript:void(0))

i) Why did we declare variables s\_names and s\_roars as static?