**Exquisite Corpse**

In the game *Exquisite Corpse*, participants draw either a head, body, or feet of a creature. The players don’t know *how* their part of the body connects to the other two, until the drawing is finished and revealed.

For this project, you’ll write a program that mimics the Exquisite Corpse game. Using methods, you’ll be able to randomly combine different parts of ASCII characters so they create a new creature.

If you get stuck or would like to see an experienced programmer tackle this project, click “get help” for a walkthrough video.

**Tasks**

3/21 Complete

Mark the tasks as complete by checking them off

**Create a BuildACreature Method**

**1.**

Take a look at the program. You’ll notice that we’ve already defined nine methods: three creatures, broken up into three different parts (head, body, feet). We can call a combination of these methods to build various creatures.

Test this out by calling the following methods in the Main() method: MonsterHead(), GhostBody(), BugFeet().

To see the program in action: save the program, then in the console type the following command and press Enter on your keyboard: dotnet run.

Hint

To call a method, write the method name, followed by a set of parentheses:

// Call a method  
myMethod();

**2.**

What if we could automate this process? Rather than manually calling three methods for each part of the body, we can create *one* method that combines those calls for us. That way, we can easily swap out the different creatures and quickly create lots of different character combinations.

After the Main() method, create a method called BuildACreature() that has three string parameters, each representing the different parts of the body (head, body, and feet).

Hint

To create a method with parameters:

static void MyMethod(string name, int age)  
{  
  // function body  
}

**3.**

When we eventually call the BuildACreature() method, we’ll pass in the names of the creatures as arguments to tell the method which parts of which creatures to select. It will look something like this:

BuildACreature("bug", "monster", "ghost");  
// this will return an ASCII creature  
// with a bug head, monster body, ghost feet

Now, how does the method know which creature will be called for each part? We’ll use a series of switch statements.

In the BuildACreature() method, create three switch statements. Each switch statement should use the value of a parameter and the cases should call the creature methods.

For example, one switch statement will take in the value of the head parameter. If the value of head is ghost, it will output the method GhostHead().

Hint

To create a switch statement in C#:

string color;  
   
switch (color)  
{  
  case "blue":  
    BluePaint();  
    break;  
   
  case "red":  
    RedPaint();  
    break;  
}

**4.**

Let’s see our new method in action!

In the Main() method, delete the original three method calls.

Now, call the BuildACreature() method, passing in strings such as "ghost", "monster", and "bug" as arguments.

Hint

To call a method with arguments:

PickAPaintPalette("red", "green", "blue");

**5.**

It’s a good practice to save and run your code every few steps to make sure there are no bugs. In the console, type the following command and press Enter on your keyboard: dotnet run.

Hint

Executing this command in the terminal tells the computer to run the **Program.cs** program. The expected behavior is that it should print out whatever you have written inside of the Console.WriteLine() commands. For each Console.Read() commands, the terminal should pause until the user types something into the terminal and presses Enter.

If you are getting an error, make sure to debug (check your code for typos or mistakes) before repeating this task. Common errors include:

* Misspelling variable names or commands
* Not closing quotation marks
* Missing semicolons (this ain’t JavaScript, y’all)
* Not assigning a type to your variables

**Create a RandomMode Method**

**6.**

Now, we will work to automate the process of building a random creature. Rather than passing in arguments each time we want to build a creature, we will create a custom method that uses the built-in Random() class and it’s method .Next() to randomly assemble a body each time the program is run.

Start by creating a custom method named RandomMode() underneath the BuildACreature() method. It won’t have any parameters.

**7.**

Inside of RandomMode(), we need to create an instance of the Random() class. Don’t worry if you’re unfamiliar with classes - we’re just using this one so we can access some of it’s built-in methods.

We’ll create an instance and new Random object by adding the following line of code to the program:

Random randomNumber = new Random();

Hint

This line of code creates a new Random object for us to use, so we can use some of its built-in methods, like .Next(). We’ve done this with other objects, like String with .IndexOf().

Random randomNumber = new Random();

**8.**

Next, we want to use the Random class’s [Next() method](https://docs.microsoft.com/en-us/dotnet/api/system.random.next) to randomly pick a number between 1 and inclusive of 3. For that reason, the range we will use is (1, 4) where 1 is the minimum value and 4 is the maximum.

Since there are three things we want to randomly select, we’ll want to call the Next() method three times, each time saving it to a new variable. In total, there should be three variables with a random value, one of the head, body, and feet.

Hint

For example, to randomly select a value between 1 and 3 for the head, use the following code:

int head = randomNumber.Next(1, 4);

Now, each time we run the program, a new value between 1 and 4 will be saved to head.

**9.**

The next step is to do something similar to what we did in the BuildACreature() method, which is to use a switch statement to determine which creature methods to call. However, this time the program would switch based on an integer versus a string.

However, if we’re *basically* reusing the same switch statement twice, it makes more sense to turn it into its own function and refactor the code.

For now, leave the RandomMode() method. We’ll come back to it in a little.

**Use SwitchStatement Method to Refactor**

**10.**

To start refactoring our code, we’ll create a new custom method to store our switch statement. Call this method SwitchCase() and give it int parameters named head, body, and feet.

**11.**

Now, inside of the SwitchCase(), write three switch statements that use the parameter integer values to call the different creature methods.

For example, to select a head the program will either select the case with value 1 to the GhostHead() method, or value 2 to the BugHead(), or value 3 to the MonsterHead() method.

Hint

To create a switch statement in C# using integers:

string color;  
   
switch (color)  
{  
  case 1:  
    BluePaint();  
    break;  
   
  case 2:  
    RedPaint();  
    break;  
}

**12.**

Test out SwitchCase() to make sure it works by calling it in Main(). For example, SwitchCase(1,1,1) should print a complete ghost to the console.

**13.**

Now, return to the RandomMode() method. Call the SwitchCase() method after the int variable definitions.

Make sure to pass in the variables head, body, and feet as arguments to the SwitchCase() method. That way, each time we run the program, a new set of numbers will be passed to that method and the output will change.

Hint

To pass in a variable as an argument:

int shoes = 3;  
   
MyClothing(shoes);

**14.**

Call the RandomMode() method in the Main() method. Then, try running the program to see the random outputs!

**Create a TranslateToNumber Method**

**15.**

Now that we have the switch statement in the method, we want to reuse it in the BuildACreature() method. But we’ve got one big problem. The BuildACreature() method takes a string type as parameters, but the SwitchCase() function takes an int type.

In order to keep using strings, we need to create a method that *translates* the meaning of our string data into ints. This method will take in a creature name and translate it to an int that SwitchCase() can make sense of. For example, "ghost" would turn into 1.

Below the SwitchCase() method, create a new method called TranslateToNumber() that takes in creature as a string parameter.

**16.**

Inside of TranslateToNumber() we’re going to use (you guessed it) another switch statement.

Create a switch statement that takes in the value of creature. This time, instead of calling a function, we’re going to *return* an integer ([you can return values in switch statements, too!](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/switch)).

So, "ghost" should return 1, "bug" should return 2, and "monster" should return 3. Make sure to include a default case that returns 1.

Hint

Here’s an example of a switch statement that returns a value:

switch (color)  
{  
  case "red":  
    return 1;  
   
  case "green":  
     return 2;  
   
  case "blue":  
     return 3;  
   
  default:  
     return 1;  
}

Note that the break keyword is not needed.

**17.**

Now that we have our switch statement in a method, and we have a way to translate string values to int values, we can take out the switch statement in the BuildACreature() method. Delete it!

Hint

There shouldn’t be anything inside of the BuildACreature() method.

**18.**

Next, we need to take the string values that are passed into the BuildACreature() parameters and convert them to ints using TranslateToNumber().

Call TranslateToNumber() three times, one for each body part. Save your method calls to a set of int variables. You can call them headNum, etc.

Hint

To save a method call to a variable:

int blueColor = TranslateToNumber("red");  
int redColor = TranslateToNumber("green");  
int greenColor = TranslateToNumber("blue");

**19.**

Pass the new int variables into the SwitchCase() method as arguments. Congratulations, you just refactored your code using methods!

**Complete and Run the Program**

**20.**

Let’s run the final program! Save the program. Then, in the console, type the following command and press Enter on your keyboard: dotnet run

Hint

If you encounter an error when running **Program.cs**, make sure to double check your code for typos or mistakes. If the program ran successfully previously, then the error(s) is in the code you wrote after that point.

Common errors:

* Incorrect switch statement syntax.
* A mismatch in variable or method names.

**21.**

If you are feeling ambitious, here are some extensions:

* Extend the BuildACreature() method so that all of its parameters are optional. It should assign a random body part if a parameter is not specified.
* Recreate this program so that rather than outputting a creature immediately, it prompts a user to select which parts of each creature to use to build a new creature.
* Add a starting mode, so a user can select whether to randomly generate a creature or create one manually.
* Use this [ASCII art archive](https://www.asciiart.eu/animals) to add other body parts to the program.