Project instructions

- Once done, upload the .cpp files for each of the two labs; e.g.: `recipe.cpp` and `student.cpp`.
- Don't zip the source files.
- Don't zip the entire folder and upload that. I only want source files.

Additional features

- For this game, you can modify the foods and games that you can play with the pet and you can adjust the stat change numbers however you'd like. You can add more foods/games for the pet, or more features as you wish - though it won't result in extra credit.
- If you're familiar with functions/methods from a previous class, you could use functions to make the program cleaner. But, since we have not yet covered functions in this course, you won't get extra credit for it.

Grading breakdown

CS200

CS 200 – Project 1 – Virtual Pet			
Criteria	**Points possible**	**Earned points**	**Feedback**
Program works - Program builds - Program runs - Does not crash		50	
Clean code - Consistent indentation - Use of whitespace to separate areas - Comments as needed		10	
Variables declared - Stats (hunger/health/happiness) - Pet name - Menu choice		5	
User input - petName - menuChoice		5	
User interface - Easy to read - Display pet stats - Display text before menus - Display response to user choices		10	
Feature: Feed		5	
Feature: Play		5	
Feature: Vet		5	
Feature: Quit and Game Over		5	
Bonus			
Total	0	100	

Page 1

Project 1: Virtual Pet

About

For this project, we will use variables, input, output, if statements, and while loops to make a simple "virtual pet" game.

The player will choose a pet name, and the pet will have three main stats: **hunger**, **happiness**, and **health**. The user has several action options: **feed**, **play**, **visit the vet**. There are several options for foods you can give your pet and games you can play with your pet, and they each affect the stats (hunger/happines/health) differently. The game continues running until the player chooses to quit.

```
-------------------------------------------------------
Moose
     Hunger: 0%  Health: 100%    Happiness: 100%
-------------------------------------------------------
OPTIONS: 1. Feed  2. Play  3. Vet  4. Quit
> 1

FOODS: 1. Pizza  2. Broccoli  3. Tuna
> 1

You feed pizza to Moose

-------------------------------------------------------
Moose
     Hunger: 0%  Health: 99%     Happiness: 95%
-------------------------------------------------------
OPTIONS: 1. Feed  2. Play  3. Vet  4. Quit
> 2

GAMES: 1. Fetch  2. Tug-of-war  3. Videogame
> 1

You play fetch with Moose
```

About: Iterative design

Software projects are often large and cannot just be built from the ground-up 100% complete as-is. The software architects take the software's require-

ments and breaks it up into smaller chunks, sets of features, to be implemented a few at a time. This project's design doc is presented iteratively, so you're implementing a few features at a time, giving you the ability to build, run, and test for each iteration as you go.

Iteration 1: Game Initialization

For this iteration, you will need to do the following:

- Create the stat variables and name variable for the pet.
- Get the pet's name.
- Display the pet's stats.

There will not yet be a game loop; that will be handled later.

Empty program:
For the starter program, you will need to include the **iostream** library in order to use `cin` and `cout`, as well as the **string** library in order to use the `string` data type. Your basic empty program will look like this:

```
1 #include <iostream>      // imports cin / cout
2 #include <string>        // imports string types
3 using namespace std;     // standard library
4
5 int main()
6 {
7     return 0;
8 }
```

Note that copy-pasting out of this document may not work; since this is a PDF file it could have invisible characters that mess up the code when pasting it into an editor.

Variables:
First thing in your program, you will declare these variables that will be used during the game.

Variable name	Data type	Initial value	Description
hunger	integer (int)	0	The % hungry the pet is (0 = not hungry)
health	integer (int)	100	The % healthy the pet is (100 = totally healthy)
happiness	integer (int)	100	The % happy the pet is (100 = totally happy)
petName	string	`""` (empty string)	The pet's name, set by the player.

C++ naming conventions
In C++, is it standard to give variables camel case names, with the first letter being lower-case, like this:
`playerChoice`, `howManyBugsInABox`, `secretOfMonkeyIsland`

Percentages?
Generally if you were working with percents in a math-context, you would use decimal form (1.00 = 100%, 0.5 = 50%, 0.05 = 5%) with a float or double data type. There is not a "percentage" data type.

In this case, we're just representing a basic percentage that will be modified by whole numbers (we don't want 50.24% healthy) so integers will be fine. Don't use % signs when assigning values to these variables.

Getting the pet's name from the user

Next, we need to ask the user to enter the name of the pet. Use a `cout` statement to display a message to the screen like "Please enter your pet's name: " and then use the `cin` statement to get their input and store it in the `petName` variable.

Spaces in names?
Note that a basic `cin >>` statement will only allow single words to be entered; no spaces allowed in these names. We will cover entering lines of text another time.

Displaying pet stats
Finally for this iteration, we will display the pet's information to the screen like in the example output from before:

```
------------------------------------------------------
Moose
     Hunger: 0%  Health: 100%    Happiness: 100%
------------------------------------------------------
```

Displaying percentages?
For these percentages, I'm just displaying the variable value, followed by a percent sign in my cout statement...

```
cout << "\t Hunger: " << hunger << "%";
```

Your output doesn't have to look exactly like mine, but it should be easy enough to read and shouldn't take up the whole screen.

Testing Iteration 1

At this point, make sure to build and run your program. There should be no build errors, and your program should run without crashing (which shouldn't happen now anyway; I'm not sure how you'd even program something to crash at this point!)

Example output:

```
Please enter your pet's name: Moose

------------------------------------------------------
Moose
     Hunger: 0%  Health: 100%    Happiness: 100%
------------------------------------------------------
```

Iteration 2: The game loop and basic menu

Next, we will want to add a game loop - basically, the program should continue running until the user decides to quit. Additionally, we will display a menu to the player on things they can do with their pet. This iteration won't deal with the logic for each menu option, we're just going to establish the loop, show the menu, and get the user's input.

New variables:
These variables can be added with the other variable declarations, or after you enter the pet's name; they just need to be declared *before* they're used in the program.

Variable name	Data type	Initial value	Description
isDone	boolean (bool)	false	Whether the program is done
menuChoice	integer (int)	n/a	The menu option the user chose

We will use the boolean variable `isDone` to make sure the program continues running until the user selects the "quit" option in the menu. The `menuChoice` variable is where we will store inputs any time we give the user a menu of options to choose from.

Game loop:
After the user has entered their pet's name, you will create a game loop that continues looping *while the program is not done*, which will look like this:

```
1 while ( !isDone )   // While the game isn't done yet...
2 {
3 }
```

Everything within the loop will happen each cycle of the game. For example, in one cycle it will display the pet's stats, ask the player if they want to feed / play with the pet, visit the vet, or quit the game. This will continually happen until the user eventually decides to quit (or if they get a gameover - more on that later).

Pet stats:
Move the output from Iteration 1 where you display the pet's name and stats to right within this while loop - first thing that happens.

Each cycle through the loop, we want to display the updated stats to the player. If we left the stats output *before the loop*, it would only display once and the player wouldn't know anything changed each time they selected an action.

Action menu 1:
Next, you will display a set of menu options, along with number codes that select each one:

```
OPTIONS: 1. Feed  2. Play  3. Vet  4. Quit
>
```

In the example above, I have an extra `cout` statement displaying a simple ">" sign, as a kind of prompt to signal to the user that the program is waiting on their input. This is just a simple cout followed by a cin:

```
1 cout << "> ";
2 cin >> menuChoice;
```

At the moment, we're not going to respond to the user's selection. Next iteration, we will use **if/else if statements** to look at what their choice was and respond.

Stat updates:
At the end of the while loop, *before* the closing curly brace }, we will make some stat adjustments. Each cycle, the pet will get a little hungrier, as well as a little less happy (because they get bored!) If the pet is *really hungry*, then they lose happiness even faster.

Implement the following logic:

- Add 5 to `hunger` (remember that you have to store it back into the same variable like:
  `hunger = hunger + 5;`
  or
  `hunger += 5;`
- If hunger is greater than 50...
  - Subtract 10 from `happiness`
  - Subtract 10 from `health`
- Else...
  - Subtract 5 from `happiness`

Testing Iteration 2

Now make sure to build, run, and test again. The program will start, you'll enter a pet name, and you can type in a menu option but it doesn't change the outcome. However, each cycle, you'll see the pet's stats change.

```
--------------------------------------------------------
Moose
     Hunger: 0%  Health: 100%    Happiness: 100%
--------------------------------------------------------
OPTIONS: 1. Feed  2. Play  3. Vet  4. Quit
> 1

--------------------------------------------------------
Moose
     Hunger: 5%  Health: 100%    Happiness: 95%
--------------------------------------------------------
OPTIONS: 1. Feed  2. Play  3. Vet  4. Quit
> 1

--------------------------------------------------------
Moose
     Hunger: 10%     Health: 100%    Happiness: 90%
--------------------------------------------------------
OPTIONS: 1. Feed  2. Play  3. Vet  4. Quit
> 1

--------------------------------------------------------
Moose
     Hunger: 15%     Health: 100%    Happiness: 85%
--------------------------------------------------------
```

Typing the wrong type of data

At this point, the only way your program would crash is if you typed a symbol (like "$") or a letter ("a") into the game while it's waiting for you to enter a numeric menu option (1, 2, 3, 4).

Validating the right *data type* was inputted in C++ is actually kind of a pain, and would require more knowledge of casting data, so don't worry about it right now. Assume the player will always enter the correct **type** of data.

Iteration 3: Number bounding and game over state

For this iteration, we will do some simple clean up - our percentages shouldn't go lower than 0% and shouldn't go higher than 100% - and, we will add a "game over" if the health of the pet hits 0%.

These will just be a set of if/else if statements right at the end of the game-loop, after any other stat changes have occurred.

Conditions:

- Item range for `happiness`:
  - If `happiness` is less than 0, set `happiness` to 0.
  - Otherwise if `happiness` is greater than 100, set `happiness` to 100.
- Item range for `hunger`:
  - If `hunger` is less than 0, set `hunger` to 0.
  - Otherwise if `hunger` is greater than 100, set `hunger` to 100.
- Item range for `health`:
  - Otherwise if `health` is greater than 100, set `health` to 100.
  - If `health` is less than 0, set `health` to 0 AND it's game over!

Game over:
When the game-over state is hit (`health` is less than 0), the following will happen:

1. Display a message that the pet has been taken away from the player.
2. Set the `isDone` variable to `true`.

```
You haven't taken care of Moose!
Moose has been removed from your care.
```

Build and test! - Make sure that the game over state is triggered once the pet health falls too low.

Iteration 4: Vet visits and quitting

Now we're getting into implementing handling menu options:

```
OPTIONS: 1. Feed  2. Play  3. Vet  4. Quit
```

After the user enters their choice and it's stored in `menuChoice`. You can go ahead and add `if/else if statements` or `switch statements` for each of the options, but we're only going to implement the logic for options 3 (visit the vet) and options 4 (quit the game).

Option 3: Visit the vet
When visiting the vet, they will give you advice on caring for your pet based on the pet's current stats. These will each have their own **if statements** (not if/else if) because multiple suggestions might be displayed, depending on how the pet is doing.

- Happiness is low (below 50%):
  Output the message "Make sure to play with [petname] more."
- Hunger is high (above 50%):
  Output the message "Make sure to feed [petname]."
- Health is low (below 50%):
  Output the message "[petname] isn't looking healthy. Take better care of it!"
- Happiness is greater than or equal to 50% AND
  Hunger is less than or equal to 50% AND
  Health is greater than or equal to 50%:
  Output the message "[petname] is looking OK!"

```
OPTIONS: 1. Feed  2. Play  3. Vet  4. Quit
> 3

The vet says:
- Make sure you play with Moose
- Make sure to feed Moose!!
- Moose isn't looking healthy. Take better care of it!
```

Option 4: Quit

When the user selects quit, ask them "Are you sure you want to quit?". Let the user enter 1 to quit or 2 to cancel. Get their input, and if they choose 1, then set `isDone` to `true`.

```
Are you sure you want to quit?
1. QUIT      2. Don't quit
> 1
```

Build and test! - Make sure that all of the vet messages are displayed, and that the user is able to quit and change their mind on quitting.

Iteration 5: Feeding and playing with the pet

Finally, for options 1 and 2, we're going to add sub-menus to let the player choose *what* to feed their pet and *what* to play with their pet, each option affecting the stats differently.

Customization
I'm going to put in some example stat adjustments that I used when writing the game, but feel free to change up these numbers! You can also change the foods and games available, and can add more if you'd like.

Option 1: Feed

Option	Food	Stat changes
1	Pizza	health - 1 hunger - 15
2	Broccoli	health + 1 hunger - 10
3	Tuna	health + 2 hunger - 12

After the player selects "1. Feed", then you'll display another menu of foods available. Get the user's choice again, store it in `menuChoice`, and have another set of **if/else if statements** or **switch statements** to adjust stats.

```
------------------------------------------------------
Moose
     Hunger: 25%     Health: 100%    Happiness: 75%
------------------------------------------------------
OPTIONS: 1. Feed  2. Play  3. Vet  4. Quit
> 1

FOODS: 1. Pizza  2. Broccoli  3. Tuna
> 1

You feed pizza to Moose

------------------------------------------------------
Moose
     Hunger: 15%     Health: 99%     Happiness: 70%
------------------------------------------------------
```

Note that when you feed your pet, it will subtract some amount from hunger, but then hunger also always increases every game loop. That's why here, pizza subtracts 15 from hunger, but then hunger goes up by 5 as well.

Option 2: Play

Option	Game	Stat changes
1	Fetch	happiness + 8 health + 2
2	Tug-of-war	happiness + 9
3	Videogame	happiness + 10 health - 1

The code will be very similar here as with the feed option. Use a set of if/else if statements or switch statements, get the player's selection for what kind of game they want to play, and then adjust the pet stats accordingly.

Again, feel free to adjust stat numbers or change the name of game options.

```
------------------------------------------------------------
Moose
     Hunger: 30%     Health: 100%    Happiness: 70%
------------------------------------------------------------
OPTIONS: 1. Feed  2. Play  3. Vet  4. Quit
> 2

GAMES: 1. Fetch  2. Tug-of-war  3. Videogame
> 3

You play videogames with Moose

------------------------------------------------------------
Moose
     Hunger: 35%     Health: 99%     Happiness: 75%
------------------------------------------------------------
```

Build and test!

Example game output

```
Enter your pet's name (all one word): Moose


------------------------------------------------------
Moose
     Hunger: 0%  Health: 100%    Happiness: 100%
------------------------------------------------------
OPTIONS: 1. Feed  2. Play  3. Vet  4. Quit
> 3

The vet says:
Moose is looking OK.

------------------------------------------------------
Moose
     Hunger: 5%  Health: 100%    Happiness: 95%
------------------------------------------------------
OPTIONS: 1. Feed  2. Play  3. Vet  4. Quit
> 1

FOODS: 1. Pizza  2. Broccoli  3. Tuna
> 2

You feed broccoli to Moose
```

```
------------------------------------------------------
Moose
     Hunger: 55%     Health: 90%     Happiness: 55%
------------------------------------------------------
OPTIONS: 1. Feed  2. Play  3. Vet  4. Quit
> 3

The vet says:
- Make sure to feed Moose!!

------------------------------------------------------
Moose
     Hunger: 60%     Health: 80%     Happiness: 45%
------------------------------------------------------
OPTIONS: 1. Feed  2. Play  3. Vet  4. Quit
> 1

FOODS: 1. Pizza  2. Broccoli  3. Tuna
> 1

You feed pizza to Moose

------------------------------------------------------
Moose
     Hunger: 50%     Health: 79%     Happiness: 40%
------------------------------------------------------
OPTIONS: 1. Feed  2. Play  3. Vet  4. Quit
> 1

FOODS: 1. Pizza  2. Broccoli  3. Tuna
> 1

You feed pizza to Moose
```

```
-------------------------------------------------------
Moose
     Hunger: 40%     Health: 78%     Happiness: 35%
-------------------------------------------------------
OPTIONS: 1. Feed  2. Play  3. Vet  4. Quit
> 3

The vet says:
- Make sure you play with Moose

-------------------------------------------------------
Moose
     Hunger: 45%     Health: 78%     Happiness: 30%
-------------------------------------------------------
OPTIONS: 1. Feed  2. Play  3. Vet  4. Quit
> 2

GAMES: 1. Fetch  2. Tug-of-war  3. Videogame
> 1

You play fetch with Moose

-------------------------------------------------------
Moose
     Hunger: 50%     Health: 80%     Happiness: 33%
-------------------------------------------------------
OPTIONS: 1. Feed  2. Play  3. Vet  4. Quit
> 2

GAMES: 1. Fetch  2. Tug-of-war  3. Videogame
> 1

You play fetch with Moose
```