**Jayden Jardine**

**03/18/22**

**Professor C**

**CS - 116**

# Pet Shelter P2

**Purpose:**

*Learning objective*

This lab's purpose is to get a better grasp on three important concepts, object-oriented programming, separate compilation, and inheritance. OOP is a necessary subject in coding, in this lab, it will allow for easy code reusability and added security with its private data members. Separate compilation allows programmers to maintain code in a more organized fashion by separating the code into more manageable sections. It also allows for programmers to work on the same project simultaneously, and make development faster due to less compilation. This lab will use separate compilation but since the program is still relatively small, its main purpose will be to practice keeping our code clean and manageable. Lastly, a sub-topic of OOP, Inheritance, will be the main learning objective for this lab. Inheritance is a way to take an already existing class and create a new one that shares all of the same properties plus any added features the programmer wants. Inheritance will allow this lab to become more specific by building on the base Pet class and creating derived classes with more specialized variables that fit certain types of pets.

*Program description*

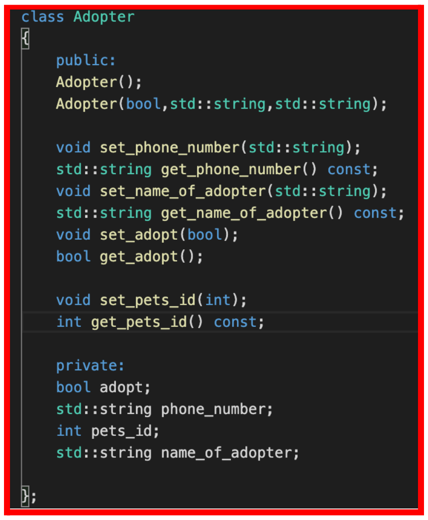
This program will share many of the same functionalities as the last lab. It will be based on a pet shelter that has a max capacity of twenty animals. The user will be able to either take existing inventory and run the program with that data or create the shelter from scratch. Each animal class will share many of the same variables because it will be inherited from the base Pet class. Each animal will be adopted after 10 days if they came in with a person willing to adopt the animal or after 15 days if they didn't. The program will print out when an animal gets adopted and then will remove it from the array. Each animal will have routine check-ups every 7 days and grooming every 1-3 days based on the animal's needs. It will also have the option to see what animal has been in the shelter the longest.  Lastly, the user will be able to save the data to a .txt file so that the data can be read in and used on the following day.

**Plan & Organization:**

In order to complete this lab to the best of my abilities, I will start by doing research on inheritance and other topics that I am currently learning. By getting a better grasp on these topics, I will be able to start coding with fewer questions thus allowing me to be more efficient. To prepare and plan my days accordingly I will create a timeline with important due dates. A timeline will allow me to break this lab into smaller more manageable chunks and make me stay on track. Another tool that will come in handy is the flow chart, it will have the program's flow and help guide me on the order in which this program will be developed. Since this lab builds off our last lab, many pieces will be reused. The first thing to do is make the old code functional with separate compilation, this is helpful to me the programmer, and is also a requirement. Since most of the code will be very similar, it seems like a good approach would be to try and build off the code from the last lab. This will allow me to focus on learning and implementing inherited classes and even some aspects of the user experience design. Since the basic outline is already completed, retrofitting the program will take up most of my time. First I will need to restructure my base class to hold the pets name as a string, the unique ID as a five-digit long int, the days it's been in the shelter as an int, the animal's weight as a double, and an estimate of the animal's age as an int. This type of data is shared between all animals, this is why it's in the base class. Next, I will create the derived classes for two types of pets, one being a dog, the other being a bird. The dog class will have two extra data members, the first being a string of its favorite type of toy, the second being the breed of dog. The Bird class will have data members of its color with a type string, and a bool on if it speaks or not. In addition to extra data members, I will need to create member functions that set and get the values. After creating and implementing the classes, I will need to create some functions that can filter through all of the data and perform operations like grooming, checkups, and adoption. Adoption will need multiple helper functions, one to read in from a .txt file, one to parse data into an array, one to resize arrays as needed, one to write to file. The grooming and checkup functions will also need functions but it seems like they will be fairly simple and will only need a for-loop. The input needed for this program will be provided by the user or from a .txt file, the types will consist of ints, doubles, strings, and bools. This program shouldn't need any global variables but could be used for the size of the shelter. It would help with maintaining the size of the array to prevent the user from putting too many animals into the shelter. Once the program is functional, I will try to make the user interface/experience more enjoyable by sectioning parts of the output off with boxes and providing the user with a helpful menu and guide. The project will be complete when the code runs smoothly without any bugs and is enjoyable for the end-user.

**Development Process:**

The developmental process was quite different than usual. My previous labs consisted of creating a program from scratch, whereas this program was built off of code I had already written. I had to review all of the code and become familiar with the way it functioned again. This part was easy because I had written the code only a few weeks earlier. Since this lab required the implementation of new concepts that I hadn't fully grasped like inheritance, virtual functions, and polymorphism, I made sure to do some research and create/use “mini” projects on CPP.sh to grasp how they worked. After figuring things out, I started to think about what functions needed to be virtual and which functions would be inherited. The flowchart allowed me to quickly decipher what needed to be added and where. The first lines of code added to the project were creating two derived classes. This needed to be done first because the rest of the code was going to change and depend on those classes. Working with the new classes went smoothly when it came to assigning data before it was entered into the base class array. Once the derived class pointers were entered into the array, getting any useful information from then again seemed impossible. After many hours of research and playing with the code I found that I had made a simple mistake and was not implementing the virtual functions correctly. This error will be expanded on further in the pitfalls section of my report. After both the classes and virtual functions were created and functioning properly I moved on to creating the logic for getting the user's information, creating the correct object, assigning the respective data members, and then finally pushing it to the base class array. To make this process easier, I split it into a few different functions. A detailed description of these functions can be found in the Product section of the lab report. The most important function created the object based on the user's input and then returned the pointer to the object so it could more easily be placed into the array. Once creating and assigning the object based on the user input was finished, I transitioned into taking that same information but from a .txt file. Since these two actions are similar, this was done in a short amount of time and really just built off of the previous lab's implementation of reading the data in from a .txt file, just with more data. I then created a new class for the potential adopters that would be aggregated by the base class see DP(1).



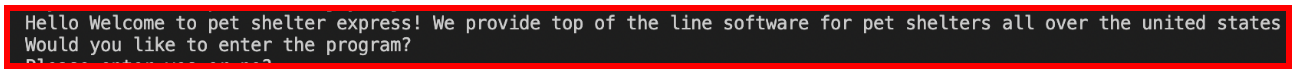
```
class Adopter
{
    public:
    Adopter();
    Adopter(bool,std::string,std::string);

    void set_phone_number(std::string);
    std::string get_phone_number() const;
    void set_name_of_adopter(std::string);
    std::string get_name_of_adopter() const;
    void set_adopt(bool);
    bool get_adopt();

    void set_pets_id(int);
    int get_pets_id() const;

    private:
    bool adopt;
    std::string phone_number;
    int pets_id;
    std::string name_of_adopter;

};
```

This data would be created in the same place where the pets would be added into the array but instead, assigned to a vector. I chose a vector to hold these objects so that deleting and resizing could be developed faster, and wanted to get a better understanding of how to use vectors and how they can be utilized. I wrote this data to a separate .txt file and read from it so that the shelter could easily interchange or edit the data but in hindsight, it was just added steps that were not necessary. Lastly, I had to create a delete function for both the array and the vector.

**Product:**

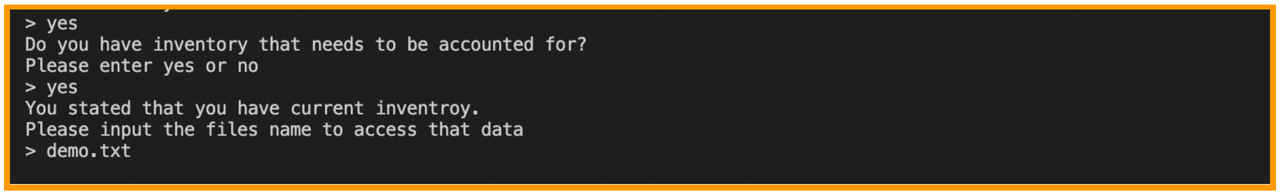
*Perfect case test case 1:*

The program is fully functional and runs smoothly. It starts by greeting the user and giving a little background as to what it is. The user is then asked if they would like to enter the program P(1).



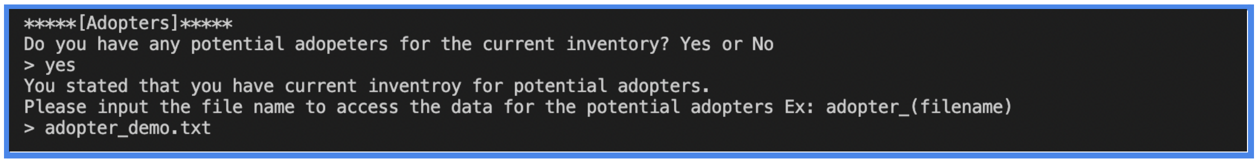
```
Hello Welcome to pet shelter express! We provide top of the line software for pet shelters all over the united states
Would you like to enter the program?
```

After entering in yes, the program asks if you have any files to load in. If the user does have files, the data will be read in and assigned accordingly P(2).



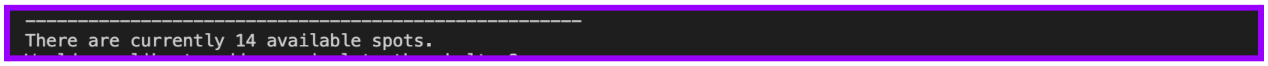
```
> yes
Do you have inventory that needs to be accounted for?
Please enter yes or no
> yes
You stated that you have current inventroy.
Please input the files name to access that data
> demo.txt
```

If the user enters provides a .txt file for the inventory then they most likely will also have data for potential adopters. This of course leads to taking in another.txt file P(3).



```
*****[Adopters]*****
Do you have any potential adopeters for the current inventory? Yes or No
> yes
You stated that you have current inventroy for potential adopters.
Please input the file name to access the data for the potential adopters Ex: adopter_(filename)
> adopter_demo.txt
```

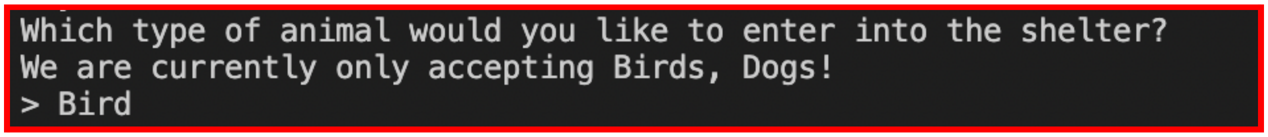
Once the data is populated into the array/vector or the user states they have no files to read in, the program will then display the number of open spaces P(4).



```
-----------------------------------------------------
There are currently 14 available spots.
```

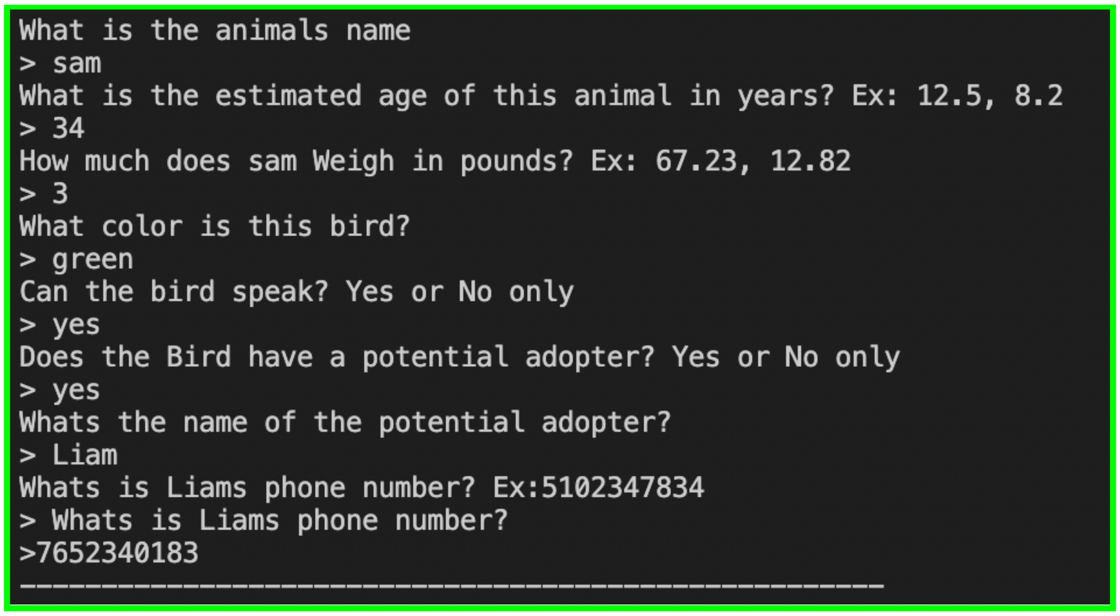
The user is asked if they would like to add a pet to the shelter, once the user confirms they would like to, they are asked what type of animal and are displayed all of the possible options

P(5).



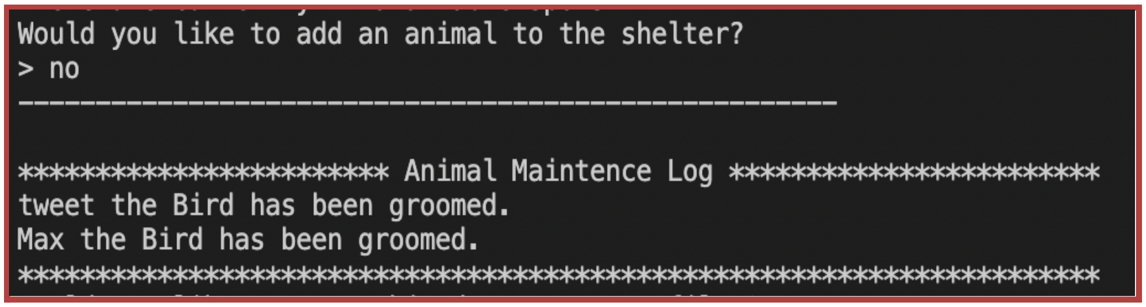
```
Which type of animal would you like to enter into the shelter?
We are currently only accepting Birds, Dogs!
> Bird
```

After deciding on which animal, they are asked a series of questions to fill the objects data members up P(6).



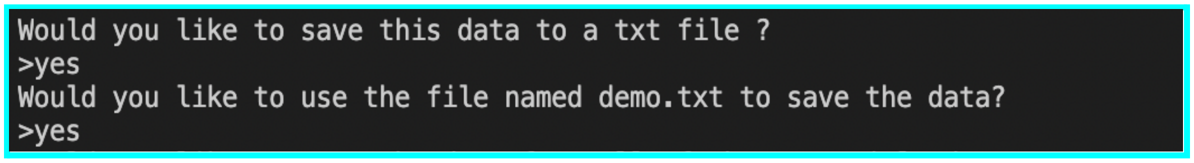
```
What is the animals name
> sam
What is the estimated age of this animal in years? Ex: 12.5, 8.2
> 34
How much does sam Weigh in pounds? Ex: 67.23, 12.82
> 3
What color is this bird?
> green
Can the bird speak? Yes or No only
> yes
Does the Bird have a potential adopter? Yes or No only
> yes
Whats the name of the potential adopter?
> Liam
Whats is Liams phone number? Ex:5102347834
> Whats is Liams phone number?
>7652340183
-----------------------------------------------------
```

This action of adding pets continues until the user decides to stop or the limit of the shelter is reached. Once the user stops adding the pets, a log of what animals were groomed and had their checkups are displayed P(7).



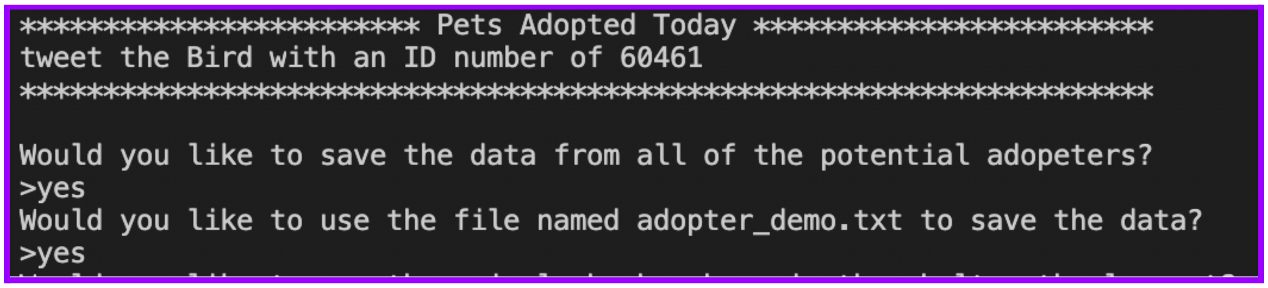
```
Would you like to add an animal to the shelter?
> no
---------------------------------------------------

*********************** Animal Maintence Log ***********************
tweet the Bird has been groomed.
Max the Bird has been groomed.
*******************************************************************
```

The user is asked if they would like to save the data, which is then followed up by asking for the names of the files, or by using the same filename that was used at the beginning of the program P(8).



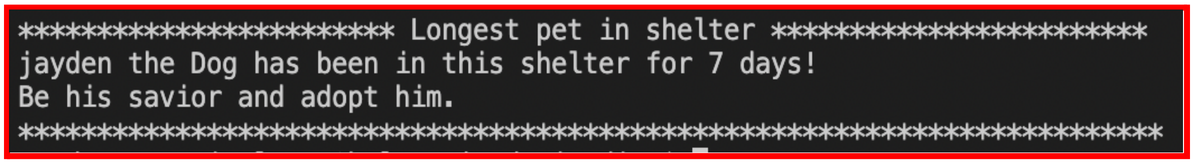
```
Would you like to save this data to a txt file ?
>yes
Would you like to use the file named demo.txt to save the data?
>yes
```

All of the animals that had been adopted are displayed and the user is asked if they would like to save the data from the potential adopter's vector P(9).



```
************************* Pets Adopted Today *************************
tweet the Bird with an ID number of 60461
**********************************************************************

Would you like to save the data from all of the potential adopeters?
>yes
Would you like to use the file named adopter_demo.txt to save the data?
>yes
```

Lastly, the program displays the pet that has been in the shelter the longest if the user desires P(10).

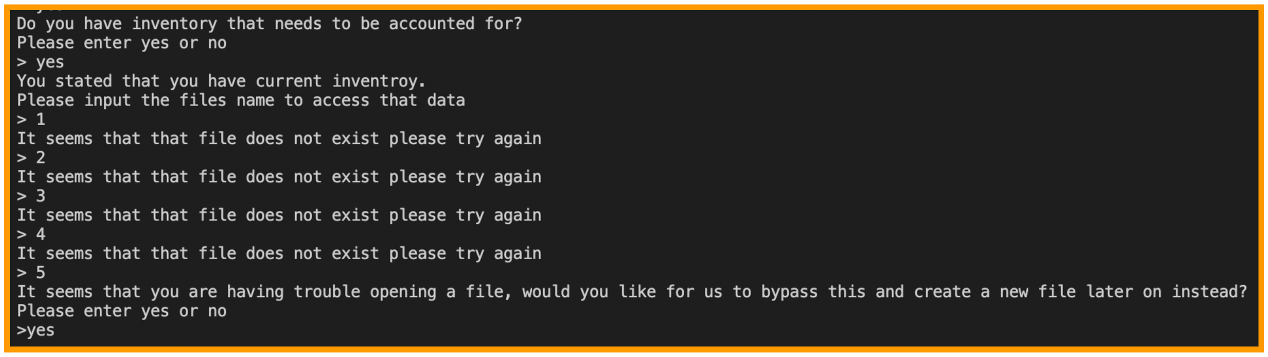


```
************************ Longest pet in shelter ************************
jayden the Dog has been in this shelter for 7 days!
Be his savior and adopt him.
************************************************************************
```

*Test case 2:*

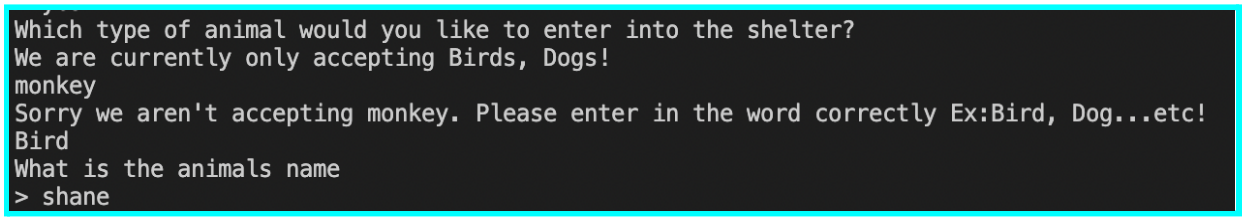
*Highlighted points:*

This execution will not be as detailed as the first because many points will be the same but instead will feature the capabilities of the program and the limitations. The first capability to be noted from the program is when the user says they have previous data that needs to be read in from a .txt file but can't enter due to an invalid file name, they are given five chances before the program asks if they would like to proceed without it HP(1).



```
Do you have inventory that needs to be accounted for?
Please enter yes or no
> yes
You stated that you have current inventroy.
Please input the files name to access that data
> 1
It seems that that file does not exist please try again
> 2
It seems that that file does not exist please try again
> 3
It seems that that file does not exist please try again
> 4
It seems that that file does not exist please try again
> 5
It seems that you are having trouble opening a file, would you like for us to bypass this and create a new file later on instead?
Please enter yes or no
>yes
```

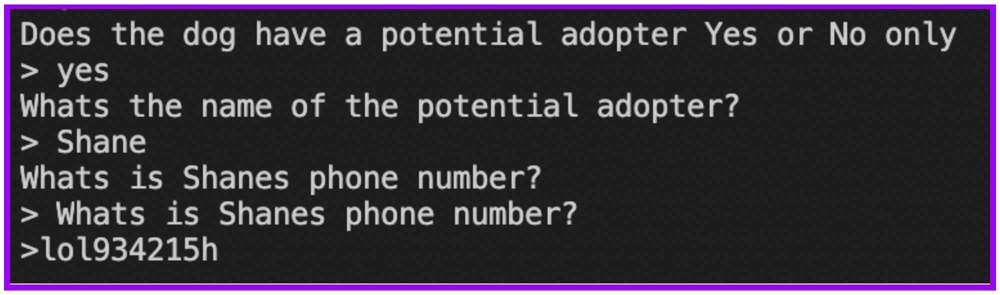
This is fail-safe so that the user isn't trapped in a loop because they can't seem to find the correct file. The next part of the program to be highlighted is when the user is asked what type of animal they would like to enter into the shelter. If the user enters the wrong type of pet then the program displays the options that are available and allows them to try again HP(2).



```
Which type of animal would you like to enter into the shelter?
We are currently only accepting Birds, Dogs!
monkey
Sorry we aren't accepting monkey. Please enter in the word correctly Ex:Bird, Dog...etc!
Bird
What is the animals name
> shane
```

*limitations:*

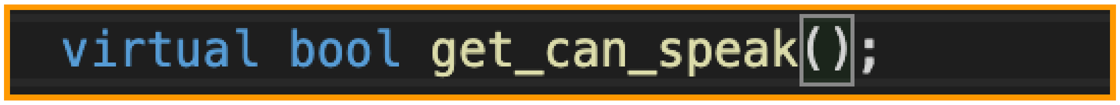
Lastly, a function that gets the user's phone number is called, this function has validation for the length of the string but doesn't verify that all characters are a number. Though this would work syntactically, it wouldn't make sense because you can't call a phone number that has a "j" in it L(1).



```
Does the dog have a potential adopter Yes or No only
> yes
Whats the name of the potential adopter?
> Shane
Whats is Shanes phone number?
> Whats is Shanes phone number?
>lol934215h
```

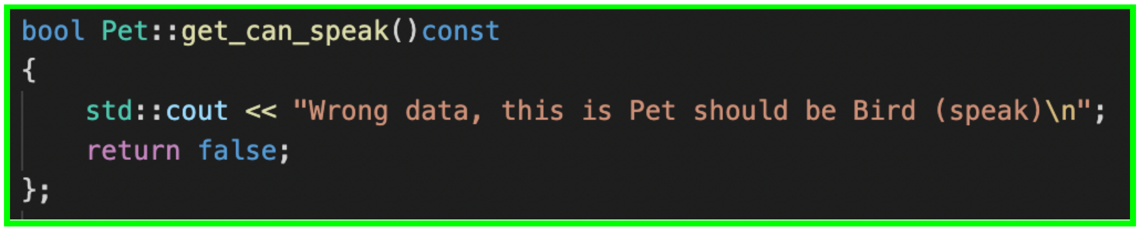
**Pitfalls:**

Since this was my first time using inheritance, It took quite a bit of research and playing around with the code to grasp exactly what was happening. I was running into an error with my virtual functions, I was not allowed to call them on my derived class objects. The implementation fit with what I was learning from both the book, StackOverflow, and youtube, but it still wasn't working. Finally, after hours of research, I found that I was forgetting to add one little word, “const”, in the derived class declaration see PI(2). Since the functions didn't match up, it was automatically calling the base class version of the function when it was supposed to be calling the derived class implementation. The first picture is the definition of the get_can_speak() function in the “Bird” class, where the second is the definition from the Pet base class PI(1). As you can see, one has const and the other doesn't, to the compiler these are treated as two totally different functions.



```
virtual bool get_can_speak();
```

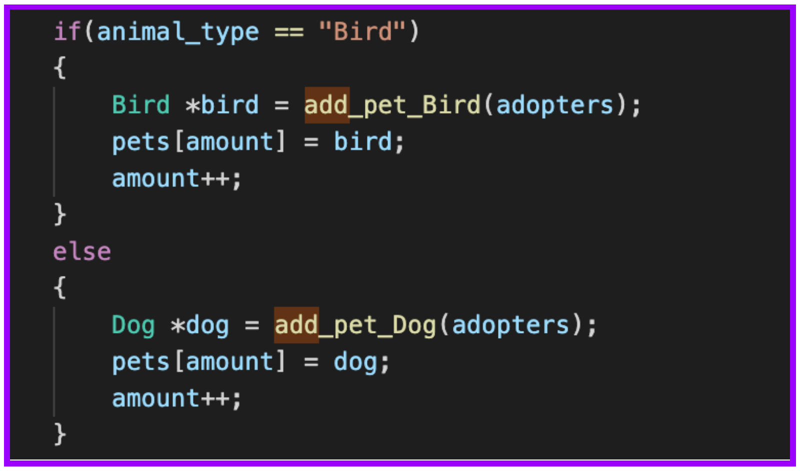
PI(1)



```
bool Pet::get_can_speak()const
{
    std::cout << "Wrong data, this is Pet should be Bird (speak)\n";
    return false;
};
```

- PI(2)

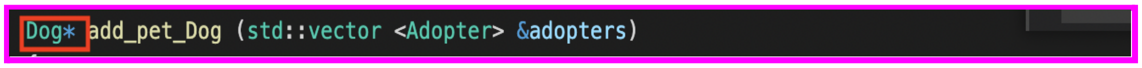
Another problem I had was creating the objects based on the user's input. I first tried to create and assign the data in the same function with an if, else statement but since this took many lines of code I had to find another implementation. Instead, I created two different functions that would create the object and return a pointer to it which would then be assigned to its respective place in the array PI(3).



```
if(animal_type == "Bird")
{
    Bird *bird = add_pet_Bird(adopters);
    pets[amount] = bird;
    amount++;
}
else
{
    Dog *dog = add_pet_Dog(adopters);
    pets[amount] = dog;
    amount++;
}
```

-PI(3)

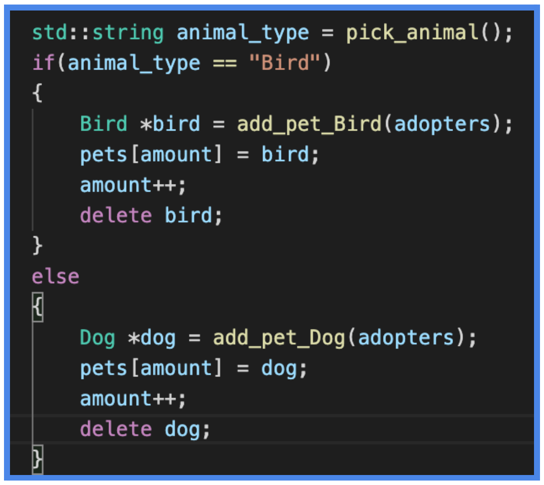
This was tricky at first because I had never made a function that returned a pointer to a derived class. At first, it was trying to assign the object directly to the array instead of a pointer to it which caused my program to not compile. This was easily fixed after changing the function declaration to return the pointer instead of the object PI(4).



```
Dog* add_pet_Dog (std::vector <Adopter> &adopters)
```

-PI(4)

Also, this function was dynamically creating the objects, I made sure to delete them after they were returned. This caused my program to stop working as intended. I had quite the time debugging this because It didn't throw any errors and was only two words worth of code. Finally, after tracing through the code countless times, I deleted the delete statement and the program was back to working as expected PI(5).



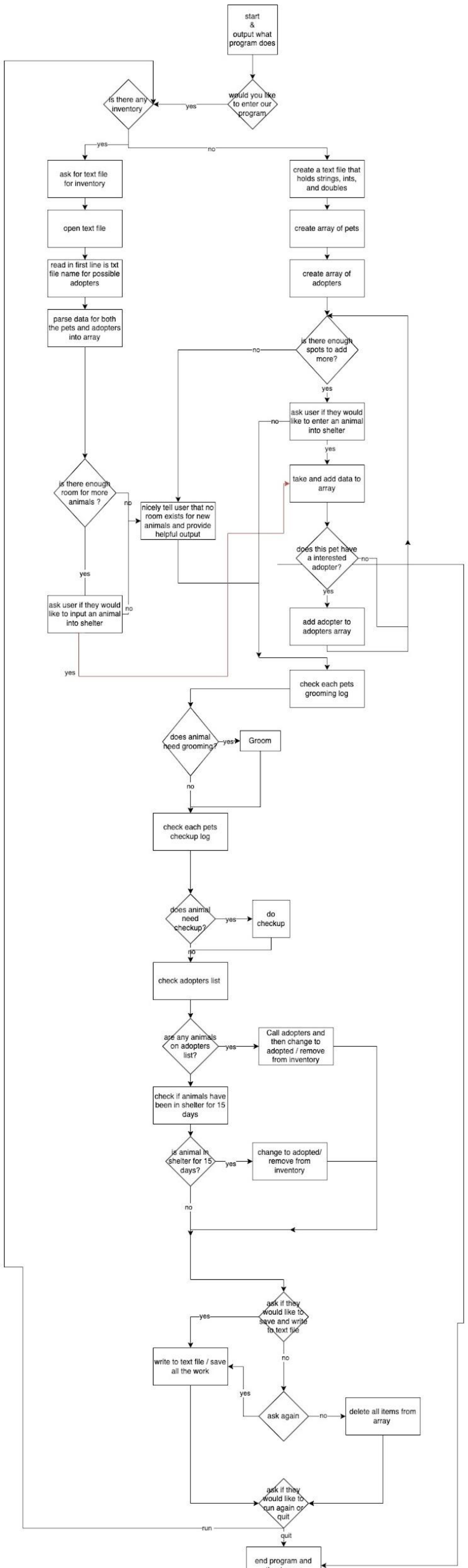
```
std::string animal_type = pick_animal();
if(animal_type == "Bird")
{
    Bird *bird = add_pet_Bird(adopters);
    pets[amount] = bird;
    amount++;
    delete bird;
}
else
{
    Dog *dog = add_pet_Dog(adopters);
    pets[amount] = dog;
    amount++;
    delete dog;
}
```

- PI(5)

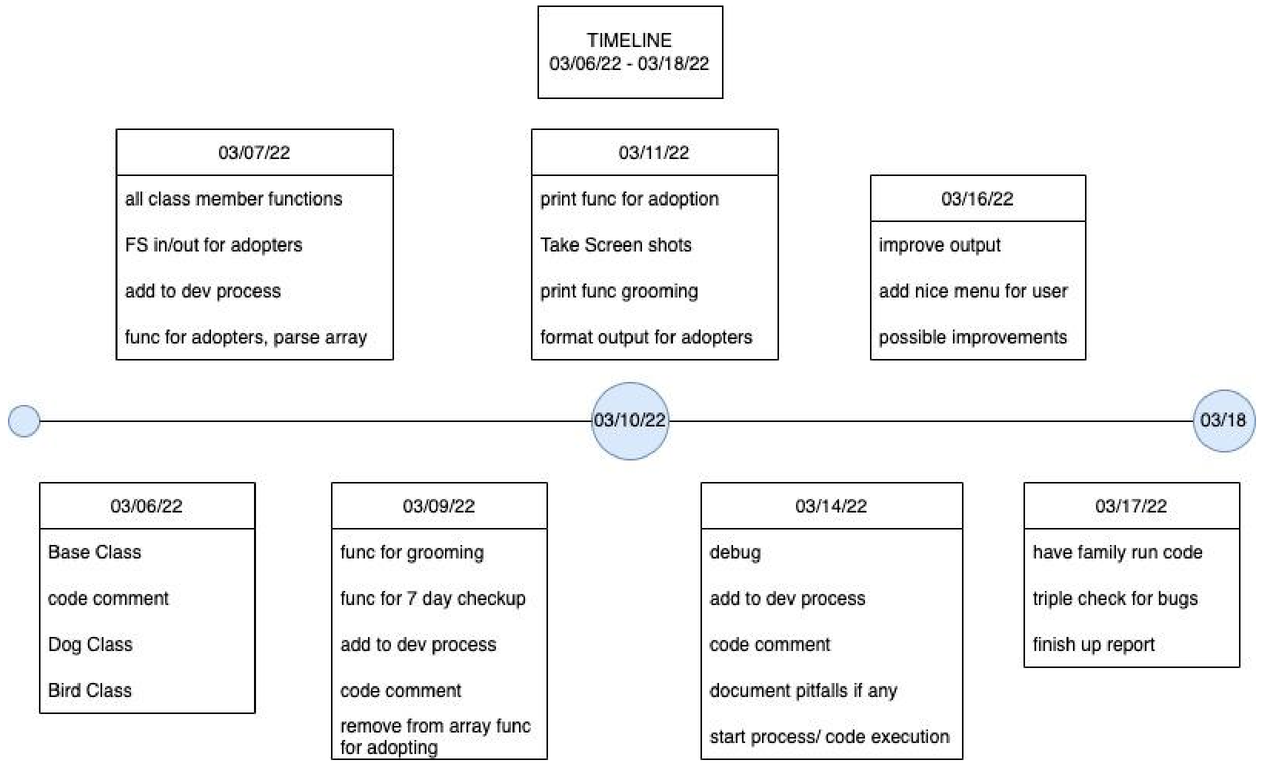
**Possible Improvements:**

For this lab specifically, I feel that a few areas of my code were repetitive and not needed. The biggest area was reading in data from two different files, though it may seem reasonable at first, one of the files contained all of the data that was needed, the other contained a copy of 3-4 data members that were also in the first file. Essentially I wrote a bunch of extra logic that was not needed. This is easy to see in hindsight but during the development of my code, I felt that it was a necessary part to allow the user to interchange different adopter files. This doesn't make sense because the data is already in one place. In the future to mitigate any mental errors like this, I will be sure to think through any code that seems similar to something already implemented. If something is eerily similar, most of the time you can modify it to be more general, thus allowing it to be more multi-purpose. A problem that I find myself running into a lot is not wanting to refactor functions that can be split into multiple functions. I fear that it will lead me to yet another bug, I don't necessarily fear bugs or debugging but I do fear not completing my assignment on time. To overcome this problem I need to remember that my code isn't due until the deadline and completing it early may not always be the best thing. Spending just an extra fifteen minutes refactoring my code will make things look a lot cleaner. To improve my skills as a programmer and for this assignment would be to make my functions more multi-purpose. For example, If in the future I wanted to add an extra type of pet to the shelter, It would be a lot of work. Yet if I made the functions, mainly the virtual ones, more general, it would allow me to quickly add a new type of pet in a fraction of the time. This wouldn't take much effort if I had started programming this project with this in mind, but since I came into it confused about virtual functions both syntactically and convention-wise, I was more worried about it running properly. A valuable lesson that I learned from this lab was to refactor my code when needed and to think about the purpose of the lab at a higher level so that adding to the program could be done more quickly.

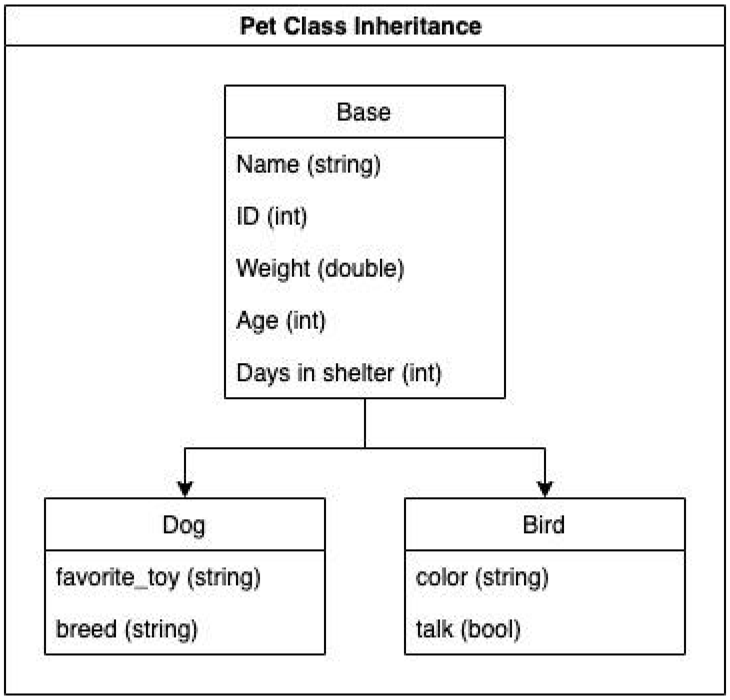
## FlowChart:


start & output what program does
would you like to enter our program
yes
is there any inventory
yes
no
ask for text file for inventory
open text file
read in first line is txt file name for possible adopters
parse data for both the pets and adopters into array
create a text file that holds strings, ints, and doubles
create array of pets
create array of adopters
is there enough spots to add more?
no
yes
ask user if they would like to enter an animal into shelter
no
yes
take and add data to array
does this pet have a interested adopter?
no
yes
add adopter to adopters array
is there enough room for more animals ?
no
nicely tell user that no room exists for new animals and provide helpful output
yes
ask user if they would like to input an animal into shelter
no
yes
check each pets grooming log
does animal need grooming?
yes
Groom
no
check each pets checkup log
does animal need checkup?
yes
do checkup
no
check adopters list
are any animals on adopters list?
yes
Call adopters and then change to adopted / remove from inventory
check if animals have been in shelter for 15 days
is animal in shelter for 15 days?
yes
change to adopted/ remove from inventory
no
ask if they would like to save and write to text file
yes
no
write to text file / save all the work
yes
ask again
no
delete all items from array
ask if they would like to run again or quit
run
quit
end program and

**Timeline:**


TIMELINE
03/06/22 - 03/18/22
03/07/22
all class member functions
FS in/out for adopters
add to dev process
func for adopters, parse array
03/11/22
print func for adoption
Take Screen shots
print func grooming
format output for adopters
03/16/22
improve output
add nice menu for user
possible improvements
03/10/22
03/18
03/06/22
Base Class
code comment
Dog Class
Bird Class
03/09/22
func for grooming
func for 7 day checkup
add to dev process
code comment
remove from array func for adopting
03/14/22
debug
add to dev process
code comment
document pitfalls if any
start process/ code execution
03/17/22
have family run code
triple check for bugs
finish up report

**Inheritance Chart:**


Pet Class Inheritance
Base
Name (string)
ID (int)
Weight (double)
Age (int)
Days in shelter (int)
Dog
favorite_toy (string)
breed (string)
Bird
color (string)
talk (bool)