
Don Cobb Art
Don Cobb Artwork
https://www.etsy.com/shop/DonCobbArtwork
Purple Martin Time

The Purple Martin

In this quest you get to create a Pet Store using the Pets you created in the last quest.

The name of the class you will implement is `Pet_Store`. In addition to implementing this class, you will have some fun creating some efficient (and not-so-efficient) search functions.

And along the way you'll pick up a new concept: Enumerated types (enums). Now is the time to read up about enums in your reference material of choice, before you start your quest.

Your first miniquest - Make a Store

You must implement the class Pet_Store containing an inner enumerated type called _SORT_ORDER which can have three possible values:

- BY_ID
- BY_NAME
- NONE

Define it as follows:

```
enum _SORT_ORDER {
    BY_ID,
    BY_NAME,
    NONE
};
```

Once the compiler sees the above declaration, you can assume that you have created a brand new type of your own (just like int or bool or the other types you're used to). The special thing about this new type you just created is that it can only take the three values named in its declaration.

It is important to make sure you retain the ordering of the enum elements I give above. The compiler will internally represent them using consecutive integer values starting at 0 for the first element.

The only private members of the `Pet_Store` class are the following:

1. a `std::vector` of `Pet` objects called `_pets`;
2. an enum of type `_SORT_ORDER` called `_sort_order`.

Your public constructor for making a Pet Store object is:

```
Pet_Store::Pet_Store(size_t n = 0);
```

It should correctly size its internal Pet vector and set the _sort order to NONE.

Your second miniquest - Get the size

Implement the public method

```
size_t Pet_Store::get_size() const;
```

I'll try and create many different pet stores of different sizes and make sure the sizes you report are the sizes I asked for.

Your third miniquest - Set the size

Implement the public method

```
void Pet_Store::set_size(size_t n);
```

This should resize your pet store as requested. It should leave the `_sort_order` untouched.

Your fourth miniquest - Clear the store

Implement the public method

```
void Pet_Store::clear();
```

This should simply call the `vector`'s `clear()` on the `_pets` vector. That's all.

Your fifth miniquest - Populate with a number of random pets

Implement the public method

```
void Pet_Store::populate_with_n_random_pets(size_t n);
```

Use static method you created in the previous quest in the `Pet` class, `Pet::get_n_pets()`, and populate the `_pets` vector with `n` Pets whose names are each 7 letters long.

Since `get_n_pets` assigns IDs in strictly increasing order, you can (and should) now set the sort order correctly to BY_ID.

Your sixth miniquest - Find a pet by ID using linear search

Implement the public method

```
bool Pet_Store::find_pet_by_id_lin(long id, Pet& pet);
```

I'm using the suffix, `_lin`, in the name of the method to imply *linear search*. In a subsequent miniquest, you will be implementing the same functionality, but using binary search.

First, note the signature and ponder/discuss why it is like that. Extra credit points may await **short** insightful posts.

When I invoke this method, I will supply it a long ID value and an overwritable `Pet` object by reference. This method should do a sequential scan of Pets in its _pets vector checking to see if any of the pets has the requested ID. If one is found, it should fill in the passed Pet object with the first such found Pet object's details. It should then return true.

If the requested ID is not found in the _pets vector, it should simply return false without touching the passed Pet object.

Your seventh miniquest - Find a pet by ID using binary search

Implement the public method

```
bool Pet_Store::find_pet_by_id_bin(long id, Pet& pet);
```

The outside-facing behavior of this method is identical to that of the previous one (linear search). But internally, this method should try to locate the requested ID using binary search, not a sequential scan.

It is critical that the `_pets` vector is sorted in non-descending order by `_id` before you begin your search. Thus, the first thing you must do here is check the `_sort_order` property of the store. If it isn't `BY_ID`, then you must invoke the helper method (supplied) `_sort_pets_by_id()`. You can see the body of this simple function in the starter code I'm giving you. Feel free to ask for clarifications and discuss things in the forums.

Implementation of binary search can be tricky if you're doing it for the first time. Pay special attention to corner cases (values of variables near loop boundaries, etc.).

I highly recommend you implement your search iteratively. Don't mess around with recursion just yet. Don't write additional helper methods.

This miniquest is worth a lot of loot. If done correctly, it will be absolutely worth it.

Since the interface of this method is identical to that of the previous one, I hope you are not tempted to use the same sequential scan logic in this method. If you do, it will fail my random test cases when I try to search LARGE vectors MANY times.

Your eighth miniquest - Find a pet by name using linear search

Implement the public method

```
bool Pet_Store::find_pet_by_name_lin(string name, Pet& pet);
```

This would be the same as its id-searching counterpart from before, except that it looks for the given name rather than an ID.

Your ninth miniquest - Find a pet by name using binary search

Implement the public method

```
bool Pet_Store::find_pet_by_name_bin(string name, Pet& pet);
```

I bet you don't need me to tell you what this needs to do. But just in case… this should do the same thing as the previous method except that it should employ binary rather than linear search, internally.

Just like in binary search for an ID, you must first check if the _sort_order of the store is BY_NAME. If not, you must invoke a corresponding helper function (supplied) before starting your search.

Your tenth miniquest - Serialization

Implement the public method

```
string Pet_Store::to_string(size_t n1, size_t n2);
```

When I invoke this method, I will supply two numbers n1 and n2. You should return to me a string of newline separated Pets at indices n1 through n2 (inclusive) provided such an index is valid (i.e. smaller than the size of your _pets vector). There should be exactly one newline character after each Pet object, including the last one.

Each `Pet` in the result string should itself stringified using its own `to_string()` method from your previous quest.

Important implementation note

Unfortunately, I have to impose the following constraint again. Terribly sorry about that.

You may not call `srand()` anywhere in your submitted code, and you have to adhere to the very strict guidelines about exactly where your code is allowed to call `rand()`.

Obviously you have to call these functions in your own testing code. Just make sure that they're not in the code you submit.

Finally, keep in mind that none of these methods is allowed to talk to the user. If you try to print something to the console or read from it in your submitted code, it will likely fail a whole bunch of test cases.

Starter code

You will submit four files, including the two from the previous quest): `Pet.h` and `Pet.cpp`. These files don't have to be identical to your submissions in the previous quest. You may have worked more on them, perhaps fixed more bugs or incorporated corrections from my model solution. The two new files are Pet_Store.h and Pet_Store.cpp. The `.h` file should contain the definition of your Pet_Store class. The `.cpp` file should contain its implementation.

Here is your `Pet_Store.h` starter code

```
// Student ID: 12345678
// TODO - Replace the number above with your actual student ID
//
//  Pet_Store.h
//
#ifndef Pet_Store_h
#define Pet_Store_h

#include <algorithm> // Needed for sort
#include <vector>

#include "Pet.h"

class Pet_Store {
private:
    std::vector<Pet> _pets;

    enum _SORT_ORDER { BY_ID, BY_NAME, NONE };
    _SORT_ORDER _sort_order = BY_ID;

    // Provided functions (no points)
    static bool _id_compare(const Pet& pet1, const Pet& pet2);
    static bool _name_compare(const Pet& pet1, const Pet& pet2);
    void _sort_pets_by_id();
    void _sort_pets_by_name();

public:
    Pet_Store(size_t n = 0);

    size_t get_size() const;
    void set_size(size_t n);
    void clear();

    void populate_with_n_random_pets(size_t n);
    bool find_pet_by_id_lin(long id, Pet& pet);
    bool find_pet_by_id_bin(long id, Pet& pet);
    bool find_pet_by_name_lin(string name, Pet& pet);
    bool find_pet_by_name_bin(string name, Pet& pet);

    // Return a string with the concatenated string reps of valid pets with
    // index n1 to n2-1 (inclusive)
    std::string to_string(size_t n1, size_t n2);

    // Don't remove the following line
    friend class Tests;
};

#endif /* Pet_Store_h */
```

And your Pet_Store.cpp file:

```
//  Pet_Store.cpp

#include <iostream>
#include <sstream>

#include <vector>

#include "Pet_Store.h"

using namespace std;

Pet_Store::Pet_Store(size_t n) {
    // TODO - Your code here
}

void Pet_Store::set_size(size_t n) {
    // TODO - Your code here
}

size_t Pet_Store::get_size() const {
    // TODO - Your code here
}

void Pet_Store::clear() {
    // TODO - Your code here
}

void Pet_Store::populate_with_n_random_pets(size_t n) {
    // TODO - Your code here
}

// These two functions can be conveniently made anonymous "lambda" functions
// defined within the scope of the functions where they're used (but only
// c++-11 on. For now we're just going to leave them here. It's straightforward
// to move them in. Just look up c++ lambda functions if you want. If you want
// to know but don't understand it, I'm happy to explain what they are. Ask me
//  in the forums. It's not necessary to know about it to ace this course.
//
// You are free to experiment by hacking the functions below, but restore their
// correct functionalities before submitting your lab.
//
bool Pet_Store::_id_compare(const Pet& p1, const Pet& p2) {
    return p1.get_id() < p2.get_id();
}
bool Pet_Store::_name_compare(const Pet& p1, const Pet& p2) {
    return p1.get_name() < p2.get_name();
}
void Pet_Store::_sort_pets_by_id() {
    std::sort(_pets.begin(), _pets.end(), Pet_Store::_id_compare);
    _sort_order = BY_ID;
}
void Pet_Store::_sort_pets_by_name() {
    std::sort(_pets.begin(), _pets.end(), Pet_Store::_name_compare);
    _sort_order = BY_NAME;
}
bool Pet_Store::find_pet_by_id_lin(long id, Pet& pet) {
    // TODO - Your code here
}
bool Pet_Store::find_pet_by_name_lin(string name, Pet& pet) {
    // TODO - Your code here
}
```

```
// When this method starts, the _pets[] vector must be sorted in
// non-descending order by _id. If it is not already, then it will be resorted.

bool Pet_Store::find_pet_by_id_bin(long id, Pet& pet) {
    // TODO - Your code here
}

// When this method is called, the _pets[] vector must be sorted in
// lexicographic non-descending order by _name. If it is not already,
// then it will be resorted.

bool Pet_Store::find_pet_by_name_bin(string name, Pet& pet) {
    // TODO - Your code here
}

// Return a string representation of the pets with indexes n1 through n2
// inclusive, exclusive of non-existent indices
// Each pet is on a line by itself.

string Pet_Store::to_string(size_t n1, size_t n2) {
    // TODO - Your code here
}
```

Testing your own code

You should test your functions using your own `main()` function in which you try and call your methods in many different ways and cross-check their return values against your expected results. But when you submit you must NOT submit your `main()` function. I will use my own and invoke your functions in many creative ways. Hopefully you've thought of all of them.

Submission

When you think you're happy with your code and it passes all your own tests, it is time to see if it will also pass mine.

1. Head over to https://quests.nonlinearmedia.org
1. Enter the secret password for this quest in the box.
2. Drag and drop your four `Pet*.*` files into the button and press it. (Make sure you're not submitting a `main()` function)
3. Wait for me to complete my tests and report back (usually a minute or less).

Points and Extra Credit Opportunities

I monitor the discussion forums closely and award extra credit points for well-thought out and helpful discussions.

May the best coders win. That may just be all of you.

Happy Hacking,

&