Ryan Filgas
CS 163
Karla Fant
Program 3 Design Writeup – Animals!

My program will implement a hash table ADT which imports information from an external data file, or takes it in from the user. The hash table will be a function that distributes data into a pointer array where each index is a pointer to a node in a linear linked list. Each node will contain a class object of type animal that manages information about breed, name, location, story, and something useful. The client program will be able to add an animal, load the hash table from an external data file, remove an item by the hash table, retrieve information about a particular animal by reference, display all information for a matching animal and location, as well as remove all animals. A separate ADT will manage a circular linked list of favorite animals. The ADT will be able to initiate the queue, check if the queue is empty, add an animal, remove an animal, peek an animal, and display all.

The public section of the animal_hash_table class will include:

Constructor – The constructor will initiate the hash table to null, and set the hash table size to 0.

Destructor – The destructor will delete the attached LLL of each element in the array, calling the destructor for each animal in the process. Once that's finished it will delete the entire array, and set the hash table to zero.

add animal – add animal is a function that will take in an animal class object by reference from the user, and then using the hash function it will be stored into an index of an array of pointers, where it's added to the beginning of an LLL where the data is then copied to an animal class object via a function in the animal class. It will return success or failure.

load from – load from is a function that reads data from an external data file, parses the information of each animal, stores that into an animal class object, and uses the add_animal function to take the data from there into the hash table. It will return success or failure.

remove animal – remove animal uses the hash table to remove an animal by breed, location, and name. It will use the breed and location to locate the animal, and the name to make sure it doesn't delete others. If the client would like to remove all, they can use the word "all" as the argument for the name.

retrieve animal – retrieve animal uses the hash function to retrieve an animal by breed, location, and name without changing the list. I The function will return the animal by reference using the animal class to copy the data. If the client would like to retrieve all, they can use the word "all" as the argument for the name.

display animal – is the same as retrieve animal, but it will display every animal with matching breed and location.
remove all – remove all will call the destructor on the hash table.

Ryan Filgas
CS 163
Karla Fant
Program 3 Design Writeup – Animals!

The private section of the animal hash table class will include:

Hash table pointer – This is a pointer to the first element of an array where each element is a pointer to a node containing an animal object.

Hash table size – This is an integer to keep track of the size of the hash table.

The public section of the favorite_list class will include:

Constructor – The constructor will set all values to NULL.
Destructor – The destructor will delete every item in the list, setting the rear pointer to NULL.
Initiate queue – Initiate queue will initiate a circular linked list with only one animal node.
Is empty – Is empty will return true if the list is empty.
Is Full – //Sometimes this is needed in a queue, but not here.
enqueue – enqueue will add a node to the beginning of a CLL by inserting after rear.
dequeue - dequeue will remove a node from the end of a cll by traversing to the end and deleting rear.
display all – Will display each node in a CLL.
Peek – This will display the first node without changing the list.

The animal_node struct will include:

An object of type animal. – This object contains char arrays holding breed, name, location, story, and something else useful
A pointer to the next node.

The animal class will include:

Display entry – displays data from each member.
Input entry – takes in breed, name, location, story, and something else useful as arguments, and copies into the class object.
copy entry – Copy entry takes an animal class object in by reference, and fills it with the current objects data.
Is match – is match checks for name first, breed second, and location last. Since the hash function will find the approximate location, the name is going to be the most unique piece of data in the list of objects, followed by the breed, and location. This means less time confirming a search if there isn't an immediate match.

Ryan Filgas
CS 163
Karla Fant
Program 3 Design Writeup – Animals!

Design Considerations

The main design considerations in this project will be the implementation of a hash table, and the implementation of a queue under a separate adt. The hash table will have to focus on efficiency, which balances the needs of the program for memory, and speed to access data. This means the hash function will need to use prime numbers both in the hash function itself, and for the size of the ADT, so that the data is distributed evenly. When implementing the ADTs other member functions, I'll have to compare data, and traverse linked lists to find the matching data that the function will look for. In addition to this I'll have to be vigilant about remaining within the definition of an ADT, and making sure that the ADT abstracts the data as much as possible. No communication will happen between the ADT or the user, and the client won't have to know the underlying structure of the program. The client also won't have direct access to any of the data in the ADT. Beyond all of this, dynamic memory is also a concern that will need to be addressed throughout implementation. All input passed in from the client will be converted into dynamic arrays before residing within the internal data structure. One of the larger challenges involved in an implementation like this is going to be keeping track of how to access different parts of the structure. To manage this, the program will be broken into different classes, and sub functions whenever possible. Lastly, every piece of this program will need to be well tested, and debugged before completion.