California State University, Sacramento
College of Engineering and Computer Science

Computer Science 130: Data Structures and Algorithm Analysis

Spring 2026 – Project #1 – A Basic Linked List

Revision 3

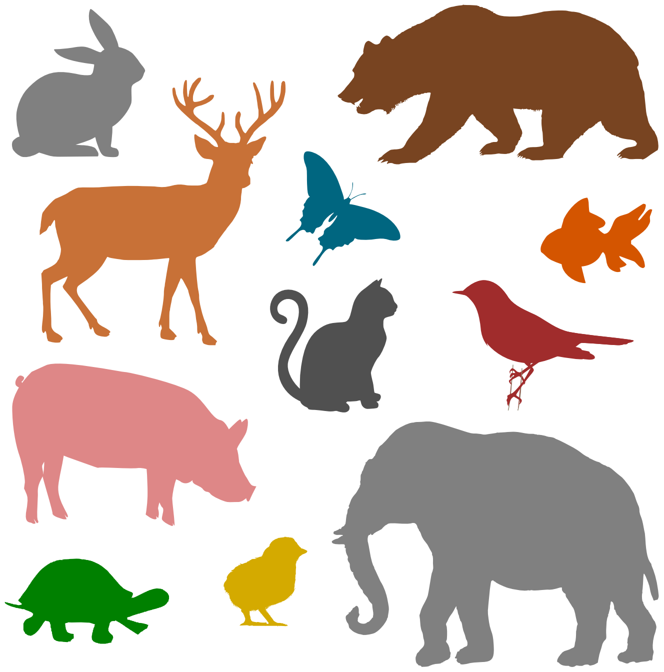
Overview

For this semester, we are going to create a basic program that helps organize a zoo.

Zoos can be wonderful places where children and adults can learn to appreciate all the creatures of this world All animals are beautiful in their own ways.

For this first assignment, you are going to create the main class that we are going to maintain this semester. In addition, you will also write a well-written, efficient, and versatile Linked-List class. You will use this in a future assignment to create an equally efficient Stack and Queue.

Given you already have developed excellent programming skills in CSc 20, this shouldn't be a difficult assignment.



Part 1: Animals

Category Enumeration

The zoo needs to keep track of the category of the animal. The categories are as follows:

- `Bug` – Arachnids and insects
- `Bird`
- `Aquatic` – Fish, mollusks, etc...
- `Mammal`
- `Ectotherm` – Reptiles and amphibians.
- `Other`

Obviously, this is not a proper zoological classification of animals. Any teacher from the Biology Department will, definitely, object to this. But, for *our* zoo, this is how we need to house each animal. For example, most zoos have a "reptile house" that contains all the snakes, lizards, and frogs. All these animals are cold-blooded (ectotherms) and need an environment that has air-conditioning. Likewise, starfish (which are not fish) live in the same aquatic environment as fish. So, they will be housed together.

Diet Enumeration

Our zoo also needs to keep track of the type of food each animal eats. Naturally, we don't want to house the herbivores anywhere near the carnivores. Our list will just have three items – which is a simplification of how nature works.

- **`Herbivore`**
- **`Carnivore`**
- **`Omnivore`**

Animal Class

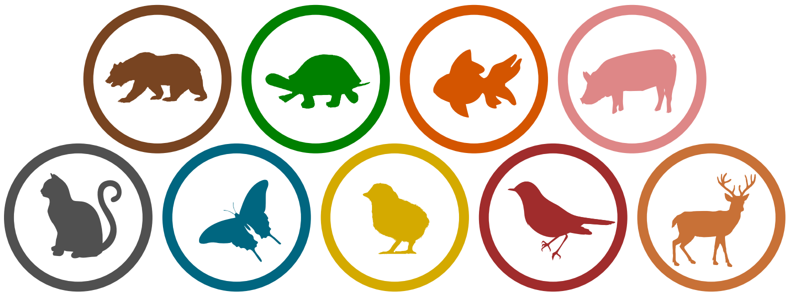
The animal class will keep track of seven different attributes. We will use these in different ways as the semester progresses. We might add some more later.

`Animal Class Attributes`		
`int`	**`ID`**	Each animal is assigned a unique number. Think of it as the animals Zoo ID.
`string`	**`name`**	At a zoo, all the animals are given names by the staff.
`string`	**`species`**	Species of the animal – such as horse, elephant, etc...
`Category`	**`category`**	The category does the animal belong to.
`Diet`	**`diet`**	The animal's diet – meat, vegetables, etc..
`int`	**`cost`**	How much it costs to feed the animal every month.

Constructors

We are just beginning to create our class, so we won't have that many methods (at least for now). Our basic constructor will contain the six fields above.

`Animal Class Constructors`
`Animal(int, string, string, Category, Diet, int)`



Methods

We won't have that many methods (at least for now).

`Animal Class Methods`		
`String`	**`nickname()`**	Returns a string with the animal's name + "the" + it's species. For example, if the animal's name is "Herky" and the species is "Hornet", the function will return "Herky the Hornet".
`String`	**`description()`**	Returns a string describing the animal. All the attributes will be used to construct the string. Please see below.
`String`	**`toString()`**	If you are using Java, override the "toString()" method to return **`nickname()`**. This is just one line of code.

description() function

This function will construct a string using the attributes listed above. I've made the text a different color for each attribute (just to make it easier to read).

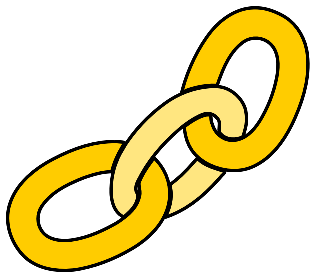
The ringtail named Scruffy is a mammal whose omnivore diet costs $180 dollars a month.

You might want to create functions to convert the Diet and Category enumerations into strings. I know that Java does this automatically, but creating functions will give you a bit more control.

In case you are curious: this is a Ringtail (aka a Miner's Cat).

Part 2: Linked List Nodes

You are going to create a very, very basic node class. Make sure to come up with a nice, well-documented, and well-written solution. We may build upon it, later, to create more advanced data structures.



This class must be encapsulated (i.e. hidden) within the LinkedList class. In other words, **make it private**.

Node Class

The Node Class will be hidden inside the Linked List class. As a result, you can make these fields "public" and not worry about assessors and mutators (get and set).

`Node Class Attributes`		
`Node`	**`Next`**	The next node in the chain.
`Animal`	**`Value`**	The contents of the node

Node Class Constructors

Node Class Constructors	
Node(animal)	Next is set to null
Node(animal, node)	Next is set to the next in the series.

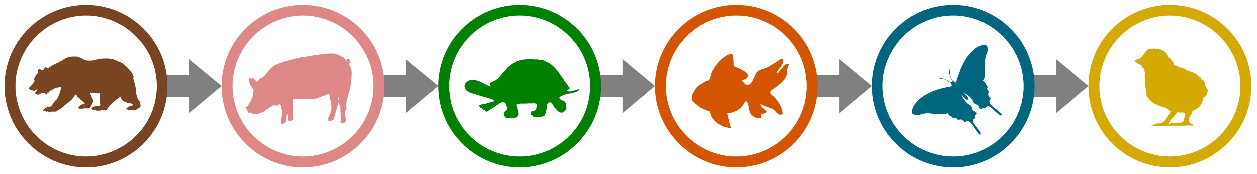
Part 3: Linked List Class

You are going to write a singly linked-list class. You will use this again in the future. So, do a good job on it, or you will hurt your future assignments. The LinkedList Class should maintain two private fields to maintain the list. In particular, it needs to maintain the link to the head and tail.

LinkedList Private Fields	
Node	**head**
Node	**tail**

Your methods **must** be O(1) for all adds. So, you **must** maintain a reference (link) to the tail. The following is the required interface (public methods – all classes define an interface)..

LinkedList Class		
string	**About()**	Returns text about you – the author of this class.
void	**AddHead(Animal)**	Adds the node to the head of the list. **This must be O(1).**
void	**AddTail(Animal)**	Adds the node to the tail of the list. **This must be O(1).**
boolean	**IsEmpty()**	Returns true if the linked list is empty. In other words, the head is null.
Animal	**RemoveHead()**	Remove the head from the linked list.
string	**ToList()**	Returns the contents of the linked list, from first to last, with each item separated by a new line. Call the description() method.



Part 4: Testing

You need to test the logic of your LinkedList class. In the **`main()`** function of your Tester Class (or whatever you call it – it doesn't matter to me), create at least **ten (10)** instances of the Animal class. Add them to the linked list – in various ways and print the **`ToList()`** method. Do **not** read values from a file or the keyboard.

Try removing the items from the list, outputting them, and printing the list afterwards.

Allowed Programming Languages

You may use any of the following programming languages.

- C#
- C++ (not recommended)
- Java

The following **cannot** be used:

- Groovy
- JavaScript
- Kotlin
- Lua
- Nim
- Python
- Ruby
- Scala
- Swift
- TypeScript
- Visual Basic .NET

Due Date & Submission

Due **March 6, 2026** by **11:59 pm.**

E-Mail your source code files to **dcook@csus.edu.** Please zip your source code files into a single .zip or .tar file. Please take note of the following. If you send your files incorrectly, I either won't receive them or won't be able to grade them.

1. Send **ONLY** **`.cpp`**, **`.java`**, or **`.cs`** files. The e-mail server will delete all attachments that have file extensions it deems dangerous. This includes **`.jar`**, **`.exe`**, and **`.class`**.
2. Do **NOT** simply zip an entire folder. The server will delete it.
3. Do **NOT** send a cloud link. I cannot open or grade these. Send an attachment.
4. Do **NOT** send it to canvas. I will not read nor grade Canvas e-mails.

Requirements

The following are the requirements for this assignment:

- This **must** be completely all your code. If you share your solution with another student or re-use code from another class, you will receive a zero.
- Do **not** use any built-in collection classes such as ArrayList. Many programming languages have nice collection classes available. Do not use any of them. **If you use any of these, you will receive a zero. No exceptions. No resubmissions.**
- You **must** write your own Linked-List. It must all be **O(1)** for all adds.
- You may use any of the programming languages listed above.
- Create some excellent testing for your class.
- Proper style (see below).
- Do **not** put your `main()` method inside the LinkedList class. This will cause major issues going forward.

Grading

1	Animal Class	20%
2	Node class is encapsulated in the LinkedList class (private)	10%
3	All Linked List add functions are O(1)	20%
4	Correct interfaces on the classes	10%
5	Proper Style	10%
6	Proper testing (see above).	20%
7	`main()` is in a testing class (not in Animal, Node or LinkedList)	10%

Beware of Static

The `static` keyword in Java indicates something that will have just 1 copy and it is not associated with an instance.

- If you define a static class, then you cannot create instances of it. Java will also force you to make every function static as well.

- A static function is associated with the class, but not an instance. Normally, your program has a static function called **`main()`**. This should be the **only** static definition in your program.

- If you declare fields in a class as static, there is only one copy that is shared by all instance. This can be disastrous in your code. For example, if you declare `head` and `tail` as static, you probably won't get weird side effects if you create **only one** instance of your class. However, if you create more than one instance, **they will fail** (they are using each other's data).

Proper Style

Points will be deducted if your program doesn't adhere to basic programming style guidelines.

1. If programming C++, Java, or C#, I don't care where you put the starting curly bracket. Just be consistent.

2. Indentation must be used.

3. Indentation must be consistent. Three or four spaces works. **Beware of the tab character!** It might not appear correctly on my computer (tabs are inconsistent in size).

4. Proper commenting. Not every line needs a comment, but sections that contain logic often do. Add a comment before every section of code – such as a loop or If Statement. Any complex idea, such as setting a link, must have a comment.

Well-formatted code

The following code is well formatted and commented.

```
void foo()
{
    int x;

    //Print off the list
    x = 0;
    while (x < this.count)
    {
       items.Add(this[x]);  //Add the item to the temporary list
       x++;
    }
}
```

Poorly-formatted code

The following code is poorly formatted and documented.

```
void foo()
{
    int x;

x = 0;
while (x < this.count)   {
     items.Add(this[x]);  //Call add on items.
       x++;
}
}
```

Some Helpful Pseudocode

AddHead Pseudocode

```
procedure AddHead(Animal animal)
    node = new Node(animal)          ... Create a new node.

    if the list is empty             ... is head null?
        this.head = node             ... Link both head and tail to the new node
        this.tail = node
    else
        node.next = this.head        ... Link the new node to the head
        this.head = node             ... The head is now the new node
    end if
end procedure
```

AddTail Pseudocode

```
procedure AddTail(Animal animal)
    node = new Node(animal)         ... Create a new node.

    if the list is empty            ... is head/tail null?
        this.head = node
        this.tail = node
    else
        this.tail.next = node       ... Link the old tail to the new node
        this.tail = node            ... The new node is the tail
    end if
end procedure
```

RemoveHead Pseudocode

```
function RemoveHead() returns Animal

    if the list is empty                  ... is head null?
        result = null                     ... We could also throw an error
    else if (this.head == this.tail)      ... Just 1 node. Set head/tail to null
        result = this.head.value
        this.head = null                  ... Deference both
        this.tail = null
    else                                  ... 2 or more nodes.
        result = this.head.value
        this.head = this.head next;       ... Move head to the next node
    end if

    return result
end function
```