Memory Allocation & Linked Lists

Exercise 1:

- Create a function that allocates a n amount of integers for a delivered pointer.
- Create a function that frees a pointer.
- Create a function that frees a pointer AND sets it as `NULL`.

Exercise 2:

Create a function that reverses a linked list.

Exercise 3:

Create a function that returns the node at the middle of the linked list.

Exercise 4:

Create a function that returns the character with the largest frequency in a linked list, knowing the "values" in the list are characters instead of integers.

Exercise 5:

Create a function that merges two linked lists.
Note: `Merge(LL1, LL2)` appends `LL2` into `LL1` instead of a new list.

Exercise 6:

Create a function that detects whether a linked list contains a cycle (loop).
Hint: use the Slow & Fast Algorithm.

Exercise 7:

Create a function that removes all duplicate values from a sorted linked list, keeping only one occurrence of each value.

Exercise 8:

Create a function that splits a linked list into two halves. If the list has an odd number of nodes, the extra node goes into the first half.

Exercise 9:

Create a function that sorts a linked list using the bubble sort algorithm.

Exercise 10:

Create a function that returns the nth node from the end of a linked list (e.g., $n = 1$ returns the last node, $n = 2$ returns the second-to-last, etc.).

Exercise 11:

Create a function that inserts a new node at a given position (index) in a linked list. If the index is out of bounds, insert the node at the end.

Exercise 12:

Given a list `head` and a value `x`, create a function that moves all occurrences of said value to the right (end of the list.)