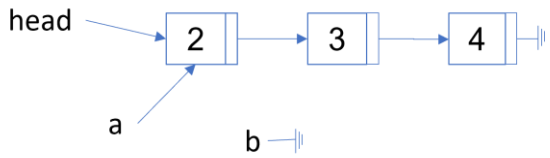
Drawing a Linked List

Consider this starting C++ data structure:



Draw the data structure that results after running this code fragment on the starting data structure:

```
Node* c = a->next;
a->next = b;
b = a;
a = c;
```

Make sure your drawing has everything: all variables, values, and pointers. Make it clear and easy to read.

Nodes have this type:

```
struct Node {
   int data;
   Node* next;
};
```

The variables `head`, `a`, and `b` are all of type `Node*`.

Linked Lists: Tail Deletion

Suppose you have a C++ program with a **singly-linked list** based on this:

```
struct Node {
   int data;
   Node* next;
};

Node* head = nullptr;  // head points to the first node on the list
```

`head` is a global variable that points to the first element of the list. If `head` is `nullptr`, then the list is empty.

Question

Using **detailed C++-like pseudocode**, write a function called `remove_last()` that:

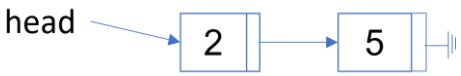
- Returns a copy of the data value of the last element in the list.
- Removes the last element, leaving a valid list with the other elements in the same order.
- Traverses the list at most one time.
- **Doesn't** use recursion.

You can assume that the list is **not** empty.

For example, suppose your program creates a list like this:

```
head = new Node{4, nullptr};
head = new Node{5, head};
head = new Node{2, head};
```

When you call `remove_last()` it returns 4, and the last node is deleted from the list:



Your `remove_last()` should work for *any* valid singly-linked list, not just this example. Your answer should be efficient, not use any unnecessary memory, and have **no memory leaks** or other errors.

Don't use arrays or vectors or other such data structures in your answer.

Linked Lists: Copying

Suppose you have a C++ program with a **singly-linked list** based on this:

```
struct Node {
   int data;
   Node* next;
};

Node* head = nullptr;  // head points to the first node on this list
```

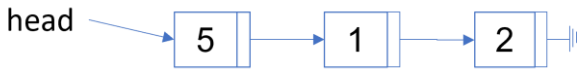
`head` is a global variable that points to the first element of the list. If `head` is `nullptr`, then the list is empty.

Question

Using **detailed C++-like pseudocode**, write a function called `copy_list()` that *returns* a pointer to the first node of a new singly-linked list that is a copy of the one head points to. The nodes in the copy are brand new nodes, and the list contains the same values, in the same order, as the original list. `head`, and the original list, are *not* modified in any way.

For example, suppose your program creates a list like this:

```
head = new Node{2, nullptr};
head = new Node{1, head};
head = new Node{5, head};
```



Then a copy is made after running this code:

```
Node* p = copy_list();
```

The original list is still there, unchanged.

Your `copy_list()` should work for any valid singly-linked list, not just this example! Make your answer efficient, and don't use arrays or vectors or other such tricks in your answer.

Depth of a Tree

The **depth of a node** `p` in a tree is defined to be the number of ancestors of `p`, excluding `p` itself. Using **detailed C++-like pseudocode**, write a function called `depth(T, p)` that **uses recursion** to calculate and return the depth of node `p` in tree `T`.

Assume the following:

- `T` is non-empty, and `p` points to a node in it
- `p->isRoot()` returns *true* if `p` points to the root of the tree, and *false* otherwise
- `p->parent` points to the parent of `p` (`nullptr` if `p` is the root)

O-notation: Linear functions

a) State the mathematical definition of "$f(n)$ is $O(g(n))$", as given in the textbook.

b) Using the mathematical O-notation definition given in the textbook, prove that $an + b$ is $O(n)$, where $a > 0$ and b are both constant integers.

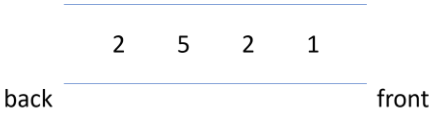
Summing a Queue

Suppose the `Queue` class contains `int`s, and has these methods:

- `Q.enqueue(x)` adds `x` to the back of `Q`
- `Q.dequeue()` removes the item at the front of `Q`; it's an error if `Q` is empty
- `Q.front()` returns a copy of the item at the front of `Q`; it's an error if `Q` is empty
- `Q.size()` returns the number of items in `Q`

Question
Write a function (**not** a method!) called `sum(Queue& Q)` that returns the sum of all the `int`s in `Q`. After calling `sum(Q)` the elements of `Q` should be the same, and in the same order, as before sum was called.

For example, suppose `Q` contains these elements:



Then `sum(Q)` returns 10, and `Q` looks exactly like it did before calling `sum`.

Important *Don't* use any extra data structures, e.g. no arrays, no vectors, no lists, no stacks, etc. You can use as many queues as you like in your answer (but try to use as few as possible). You cannot assume anything about how the queue is implement: only use methods defined above for it in your answer.

Important `sum` **can** modify `Q`, but when it's done `Q` should have the same elements in the same order as when it started.

Important `sum(Queue& Q)` is a function, not a method!

Algorithm Performance Estimation 1

If a **quadratic** algorithm takes 3 seconds to process 50 items, about how long would you expect it to take to process 100 items? Justify you answer.

Algorithm Performance Estimation 2

If an **exponential** algorithm takes 3 seconds to process 50 items, about how long would you expect it to take to process 100 items? Justify your answer.