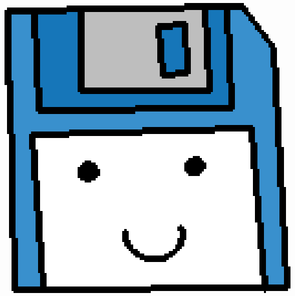
Welcome to CS 250

Written by Rachel J. Morris, last updated Aug 10, 2017

Welcome to CS 250!



Welcome to CS 250: Data Structures using C++!

This is a programming course that is standard to computer science curriculum. Let’s talk a bit about what we will be doing this semester!

What are Data Structures?

This course is about Data Structures – but what are data structures?

To be literal, they are structures built to store and work with data.

But maybe that’s not enough information!

What are Data Structures?

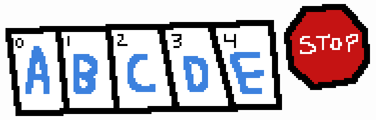
This course is about Data Structures – but what are data structures?

To be literal, they are structures built to store and work with data.

But maybe that’s not enough information!

What are Data Structures?

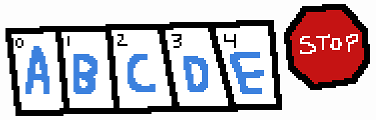
Learning about data structures includes learning to build structures that store their data in different manners, such as…



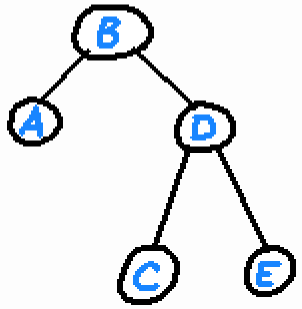
Storing data linearly, without a meaningful order.

What are Data Structures?

Learning about data structures includes learning to build structures that store their data in different manners, such as…



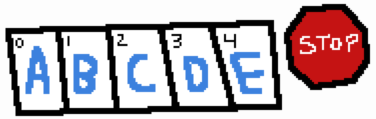
Storing data linearly, without a meaningful order.



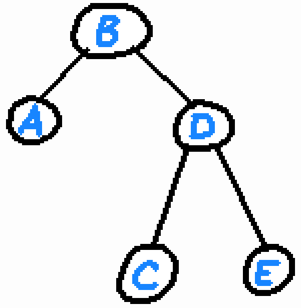
Data stored in a tree structure, which can make searching faster than a list.

What are Data Structures?

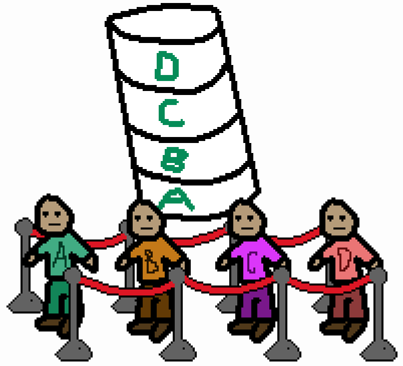
Learning about data structures includes learning to build structures that store their data in different manners, such as…



Storing data linearly, without a meaningful order.



Data stored in a tree structure, which can make searching faster than a list.



Storing data in a <u>queue</u> or a <u>stack</u> type structure, where you can only access the <u>front of the queue</u> or the <u>top of the stack</u>.

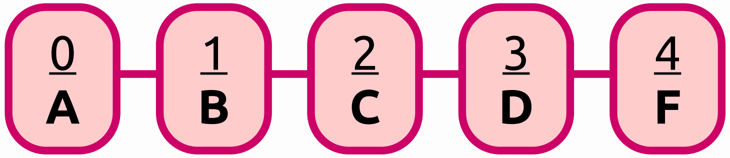
What are Data Structures?

As an example, let’s store data in a **list** and in a tree…
“A”, “B”, “C”, “D”, and “F”.

What are Data Structures?

As an example, let's store data in a **list** and in a tree…
"A", "B", "C", "D", and "F".

Linear List



How would we write an algorithm to find "D" in the list?

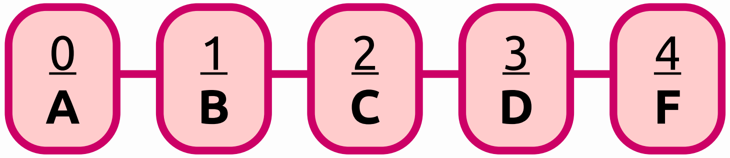
We could start at the beginning and keep checking each element, one-by-one, until you eventually find it.

This would take 4 checks to ensure "D" is found.

What are Data Structures?

As an example, let's store data in a **list** and in a tree…
"A", "B", "C", "D", and "F".

Linear List



But since this list is **sorted**, we could do it a little more intelligently.

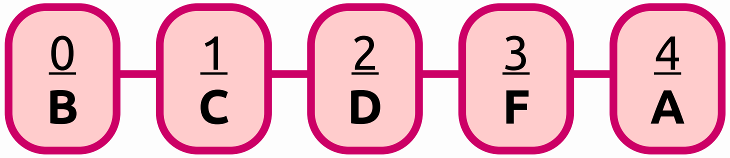
Maybe we know that "D" is closer to "F" than it is to "A", so perhaps with this one check, we decide to search end-to-beginning instead of beginning-to-end.

Then, going backwards, we just check "F" and "D" to ensure we've found it.

What are Data Structures?

As an example, let’s store data in a **list** and in a tree…
“A”, “B”, “C”, “D”, and “F”.

Linear List



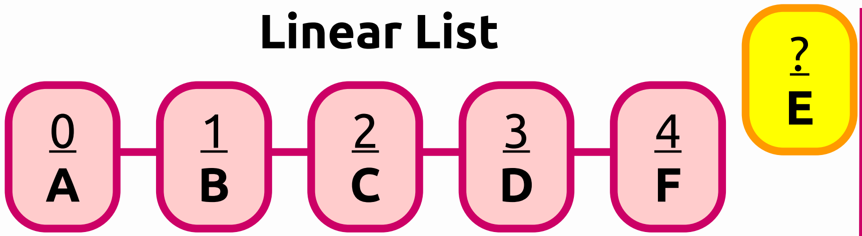
If the list **weren’t sorted**, there wouldn’t be a good way to intelligently search the list. If we just stuck to beginning-to-end, it might be fast (like searching for “B”), or it might be slow (like searching for “A”).

Worst case scenario, we’d have to look through the entire list to find our item.

What are Data Structures?

As an example, let's store data in a **list** and in a tree…
"A", "B", "C", "D", and "F".

Linear List



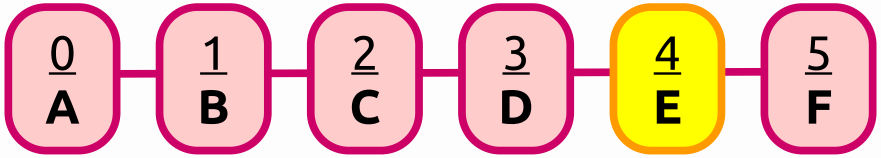
If we wanted to keep the list sorted, then extra work would pop up when we **insert** a new item.

If we insert something that belongs in the middle of the list, then we have to take the time to shift elements around to place our new item. **This can also be costly, in computing time.**

What are Data Structures?

As an example, let’s store data in a **list** and in a tree…
“A”, “B”, “C”, “D”, and “F”.

Linear List

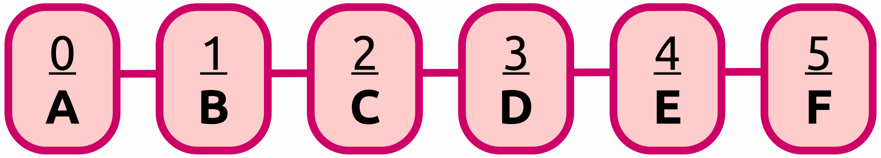


But, while we **add overhead** during our insertion process, keeping the data sorted ensures that our **search** can be done intelligently and speedy.

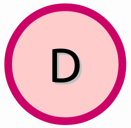
What are Data Structures?

As an example, let's store data in a **list** and in a tree…
"A", "B", "C", "D", and "F".

Linear List



Insert "D"

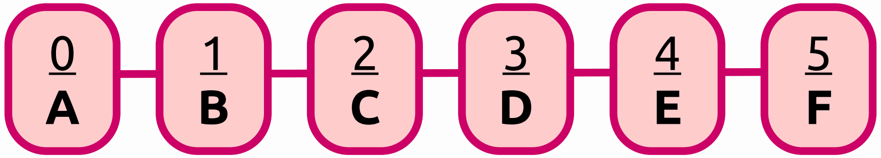


If we're using a binary search tree data structure, items can be inserted in any order, and the structure automatically stays sorted, as part of its insertion algorithm.

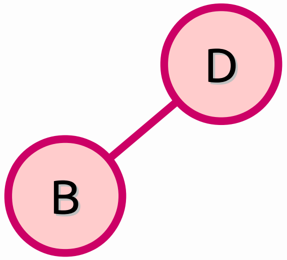
What are Data Structures?

As an example, let's store data in a **list** and in a tree…
"A", "B", "C", "D", and "F".

Linear List



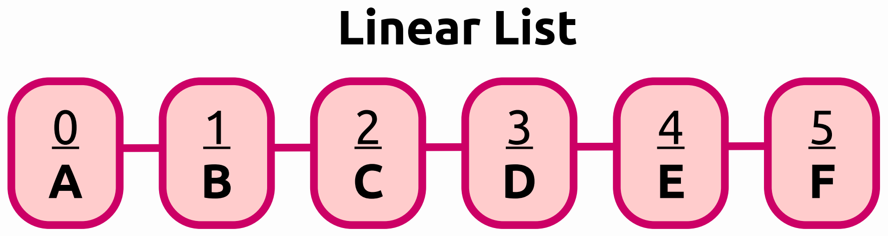
Insert "B"

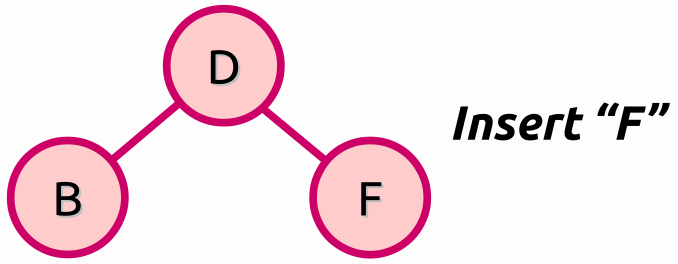


If we're using a binary search tree data structure, items can be inserted in any order, and the structure automatically stays sorted, as part of its insertion algorithm.

What are Data Structures?

As an example, let’s store data in a **list** and in a tree…
“A”, “B”, “C”, “D”, and “F”.



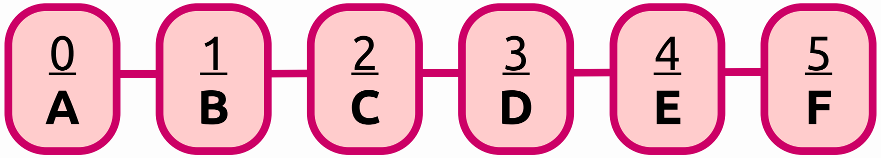


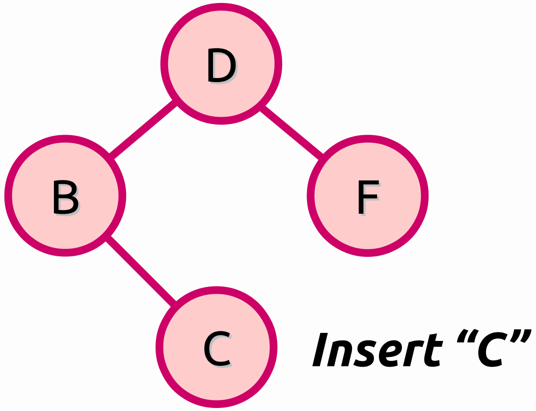
If we’re using a binary search tree data structure, items can be inserted in any order, and the structure automatically stays sorted, as part of its insertion algorithm.

What are Data Structures?

As an example, let’s store data in a **list** and in a tree…
“A”, “B”, “C”, “D”, and “F”.

Linear List



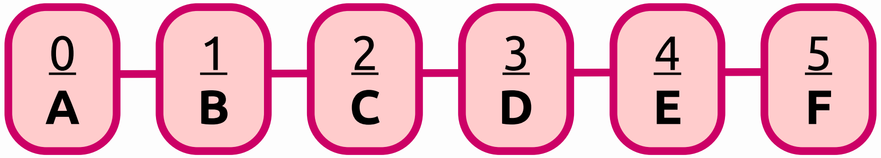


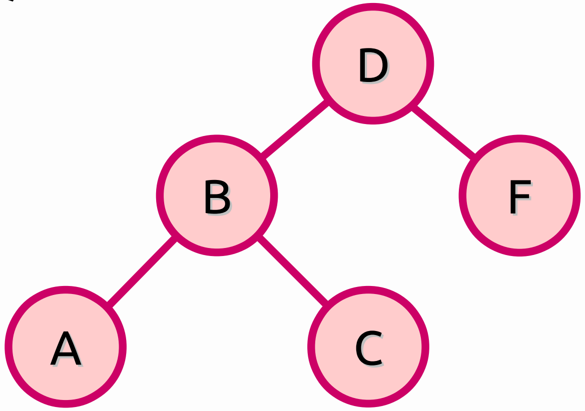
If we’re using a binary search tree data structure, items can be inserted in any order, and the structure automatically stays sorted, as part of its insertion algorithm.

What are Data Structures?

As an example, let's store data in a **list** and in a tree…
"A", "B", "C", "D", and "F".

Linear List



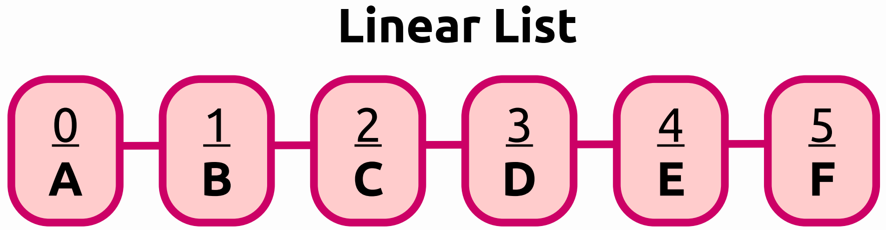


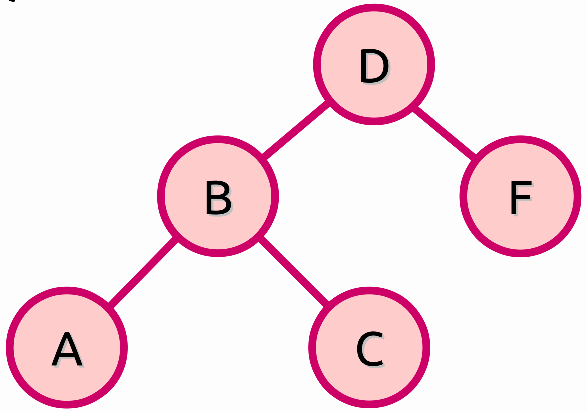
Insert "A"

If we're using a binary search tree data structure, items can be inserted in any order, and the structure automatically stays sorted, as part of its insertion algorithm.

What are Data Structures?

As an example, let's store data in a **list** and in a tree…
"A", "B", "C", "D", and "F".



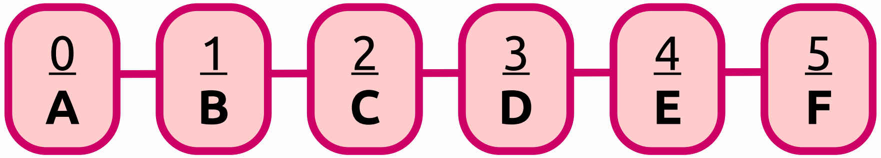


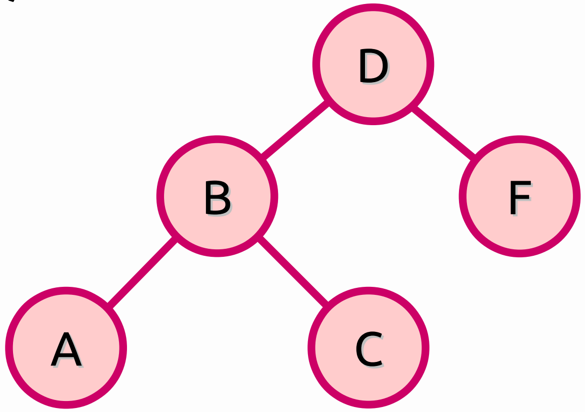
Inserting this way takes up more time than simply throwing data on the end of an unordered linear array, but it's about **trade-offs**. As part of insert, things get put in an ordered place.

What are Data Structures?

As an example, let's store data in a **list** and in a tree…
"A", "B", "C", "D", and "F".

Linear List



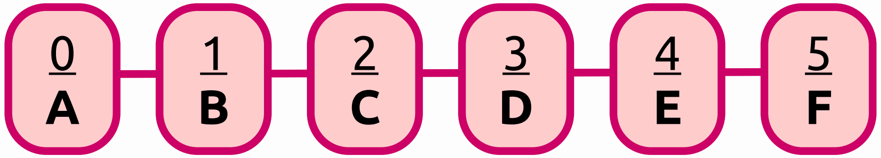


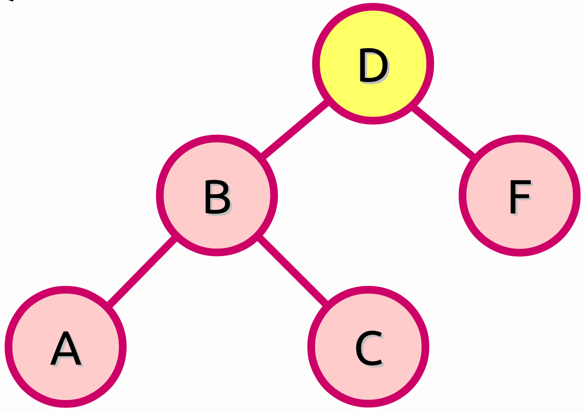
With Binary Search Trees, we have a series of **nodes**. New items with a smaller value, **go to the left**, and new items with a larger value **go to the right**.

What are Data Structures?

As an example, let's store data in a **list** and in a tree…
"A", "B", "C", "D", and "F".

Linear List





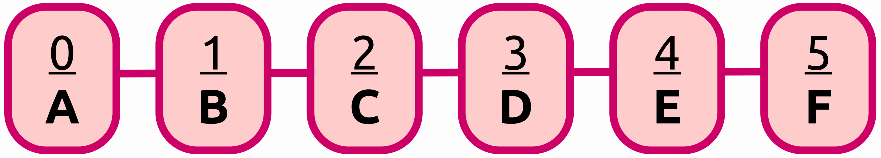
If we want to find "F",
we start at the **root node** (D), and ask:

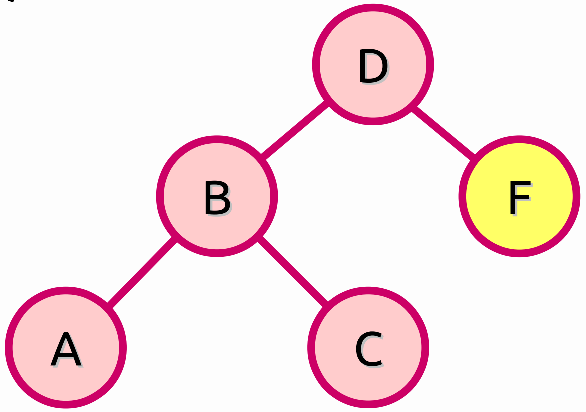
- Is "F" less than "D"? **NO**
- Is "F" greater than "D"? **YES**, go right!

What are Data Structures?

As an example, let’s store data in a **list** and in a tree…
“A”, “B”, “C”, “D”, and “F”.

Linear List



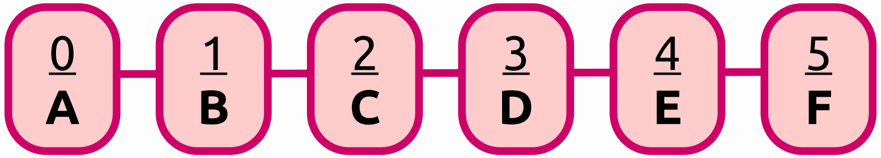


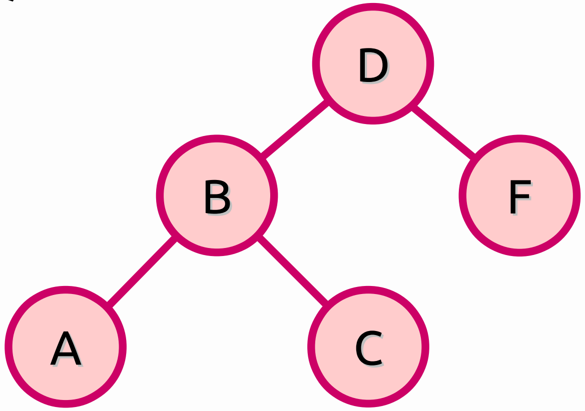
We move to the right and search – and we’ve found “F”. So we’ve only had to check “D” and “F” to get to our result.

What are Data Structures?

As an example, let’s store data in a **list** and in a tree…
“A”, “B”, “C”, “D”, and “F”.

Linear List





The search algorithm is similar to how the insertion algorithm works, so they actually have the same **average efficiency**.

What will we cover in this class?

In order to gain a good foundation in data structures, in this class we will cover the following topics…

Implementing data structures

- Linked Lists
- Queues
- Stacks
- Dictionaries
- Binary Search Trees
- Heaps
- Balanced Search Trees

What will we cover in this class?

But to be able to design our structures well, we also need to learn about…

<u>Implementing data structures</u>

- Linked Lists
- Queues
- Stacks
- Dictionaries
- Binary Search Trees
- Heaps
- Balanced Search Trees

<u>Design</u>

- Exception Handling
- Writing Tests

What will we cover in this class?

And a data structure isn't very useful if it is ***inefficient***, so we will also spend time focusing on…

Implementing data structures

- Linked Lists
- Queues
- Stacks
- Dictionaries
- Binary Search Trees
- Heaps
- Balanced Search Trees

Design

- Exception Handling
- Writing Tests

Efficiency

- Measuring algorithm efficiency
- Using O(n) notation
- Searching algorithms
- Sorting algorithms

What will we cover in this class?

Additionally, it is important to have a solid foundation in core C++ concepts, so during the class we will spend time reviewing topics like **pointers**, **memory management**, **object oriented programming, templates** and more.

If you're feeling weak on these topics, you should spend time reviewing these.

Tips for success…

Sometimes this class can be challenging. A few tips for doing well is…

- If you can't figure out how to implement some functionality, try to sketch out the problem on paper.
  For example: Step through inserting one item, two items, three items. How does it work?
- Getting acquainted with these topics requires practice, so make sure you have plenty of time to code.
- Make sure you have a solid grasp of core C++ topics. Make sure to review and practice if you have difficulty with some of them.
- Ask questions if you're feeling stuck!

Tips for success…

Sometimes this class can be challenging. A few tips for doing well is…

- Keep working at it… try different things! No code will happen if you don’t write code. Sometimes it just takes trying some small things.

- Data structures aren’t a secret – there are a lot of resources online that you can reference if you still have questions after the lectures / book.
  - **Check online! There are free e-books, lessons on YouTube, source code on GitHub, and plenty of reference pages online.**

- Make sure you understand *how* each data structure works. You don’t need to memorize the code, but if you know how it works, you should be able to implement it whenever you need it.

Data structures

I hope that you enjoy this course!

Ask the instructor questions at any time if you need help with a topic,

or keep in touch with some classmates so that you can study together.