

Iterators

In C++, iterators provide an interface for client code access to data in a way that abstracts away the internals of the data structure.

An instance of an iterator is a current location in a pass through the data structure:

Type	Cur. Location	Current Data	Next
Linked List			
Array			
Hypercube			

The iterator minimally implements three member functions:

- operator***, Returns the current data
- operator++**, Advance to the next data
- operator!=**, Determines if the iterator is at a different location

Implementing an Iterator

A class that implements an iterator must have two pieces:

1. [Implementing Class]: Must implement:

-

-

2. [Implementing Class' Iterator]:

A separate class (usually an internal class) that extends `std::iterator` and implements an iterator. This requires:

-

-

-

Locations of `::begin` and `::end` iterators:

Type	<code>::begin()</code>	<code>::end()</code>
Linked List		
Array		

Using an Iterator

```

stlList.cpp
1  #include <vector>
2  #include <string>
3  #include <iostream>
4
5  struct Animal {
6      std::string name, food;
7      bool big;
8      Animal(std::string name = "blob", std::string food = "you",
9      bool big = true) :
10         name(name), food(food), big(big) { /* nothing */ }
11 };
12 int main() {
13     Animal g("giraffe", "leaves", true),
14         p("penguin", "fish", false), b("bear");
15     std::vector<Animal> zoo;
16
17     zoo.push_back(g);
18     zoo.push_back(p); // std::vector's insertAtEnd
19     zoo.push_back(b);
20
21     for ( std::vector<Animal>::iterator it = zoo.begin();
22           it != zoo.end(); it++ ) {
23         std::cout << (*it).name << " " << (*it).food << std::endl;
24     }
25     return 0;
26 }

```

Q: What does the above code do?

For-Each loop with Iterators

```

stlList-forEach.cpp
20 for ( const Animal & animal : zoo ) {
21     std::cout << animal.name << " " << animal.food << std::endl;
22 }

```

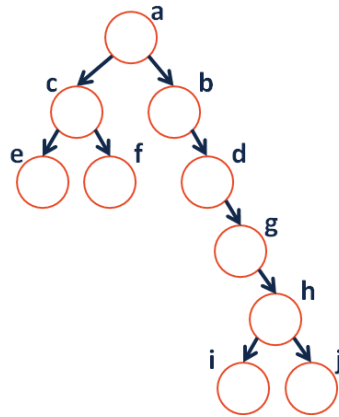
Trees!

“The most important non-linear data structure in computer science.”

- David Knuth, *The Art of Programming*, Vol. 1

We will primarily talk about **binary trees**:

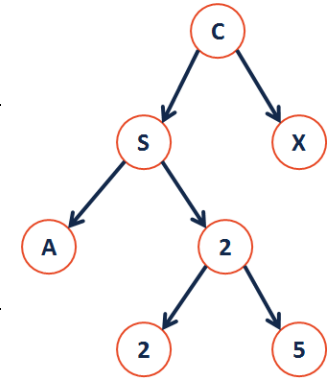
- What’s the longest **English word** you can make using the **vertex** labels in the tree (repeats allowed)?
- Find an **edge** that is not on the longest **path** in the tree. Give that edge a reasonable name.
- One of the vertices is called the **root** of the tree. Which one?
- How many parents does each vertex have?
- Which vertex has the fewest **children**?
- Which vertex has the most **ancestors**?
- Which vertex has the most **descendants**?
- List all the vertices in b’s left **subtree**.
- List all the **leaves** in the tree.



Definition: Binary Tree

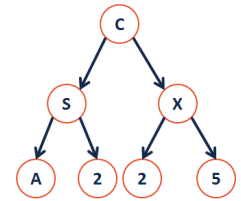
A *binary tree* **T** is either:

Tree Property: Tree Height

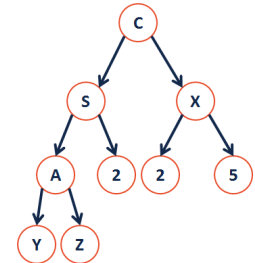


Tree Property: Full

Tree Property: Perfect



Tree Property: Complete



CS 225 – Things To Be Doing:

1. mp_lists extra credit deadline is Monday!
2. lab_quacks in lab this week
3. Exam 2 next Friday
4. Daily POTDs practice for Exam 2