EECS 402 Discussion 12!

Complexity, Simulations, Inheritance

Notes on p4

P4 Notes

Head and tail are just pointers, not nodes

- P4 due soon!
- More on what head and tail actually are

New topic: Complexity

Problem: We need a way to measure the speed of a program

- Options:
  - Seconds?
  - Instructions?

Solution: Time complexity

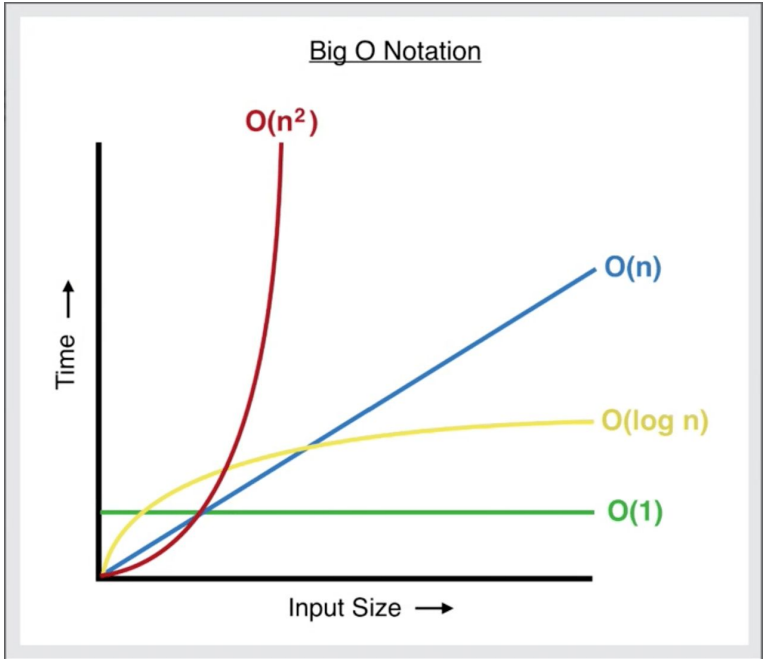
- We separate programs into broad “speed” categories based on **how their runtime changes as input size grows**
- What does this mean?
  - If I want to sort a list of 200 elements, it will take longer than a list of 20 elements
  - Adding 5 and 10 takes the same amount of time as adding 10 and 20
  - These algorithms fall into **different complexity classes**

Format

- **Big-O notation**: A way to write time complexity with a function that grows **faster than or as fast as** the algorithm runtime

Common Big-O Complexities

- For loop from 0 to N O(n) linear
- Nest for loop from 0 to N in both $O(n^2)$ exponential
- Accessing an element in an array O(1) constant



Notes on Time Complexity

- Time complexity vs. actual run time
  - O(1) can run slower than O(n) in *some* cases, but O(n) is considered slower because of how it scales
- Example:
  - A for loop from 0 to 10000000000 = O(1)
  - A for loop from 0 to N = O(n) but is faster for values where n < 10000000000
    - As soon as n > 10000000000, it is slower
- Think about how your program will scale

Battle Between Time and Memory

- Often times, decreasing time causes memory to increase and vice versa
- Tradeoffs with every decision made
- Make decisions based off of what you want to optimize

Using an array, which functions are fast? Which are slow?

- appendFront(int val)
- appendBack(int val)
- insert(int index, int val)
- popFront()
- popBack()
- remove(int index)
- valueAt(int index)
- print()
- clear()
- size()

Using an array (with extra space to add values), which functions are fast? Which are slow?

- appendFront(int val) Slow
- appendBack(int val) Fast (usually)
- insert(int index, int val) Slow
- popFront() Slow
- popBack() Fast
- remove(int index) Slow
- valueAt(int index) Fast
- print() Fast (relatively)
- clear() Fast
- size() Needs extra state

Using a doubly linked list (with head and tail), which functions are fast? Which are slow?

- appendFront(int val)
- appendBack(int val)
- insert(int index, int val)
- popFront()
- popBack()
- remove(int index)
- valueAt(int index)
- print()
- clear()
- size()

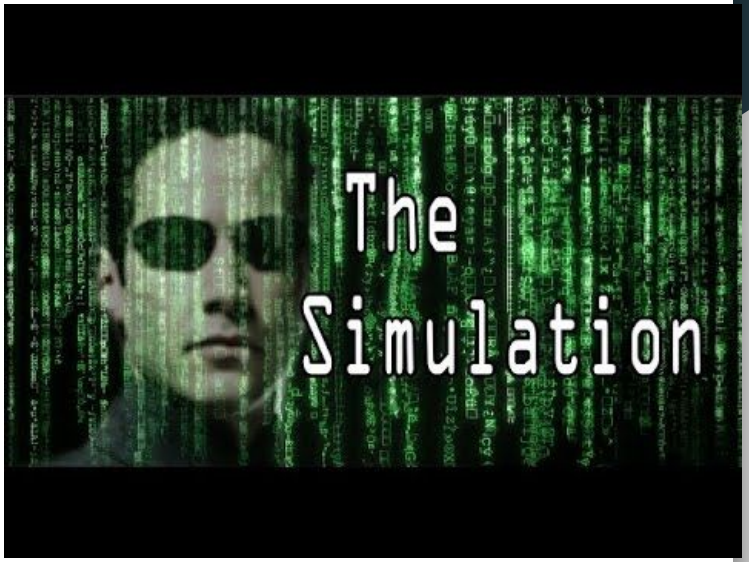
Using a doubly linked list (with head and tail), which functions are fast? Which are slow?

- appendFront(int val) Fast
- appendBack(int val) Fast
- insert(int index, int val) Slow
- popFront() Fast
- popBack() Fast
- remove(int index) Slow
- valueAt(int index) Slow
- print() Slower than an array
- clear() Slow
- size() Fast (store the size)

Simulations

Overview

- We are using an “event-driven” simulation
- This means your simulation will keep a list of “upcoming events” and continually execute the next element in the list
- For P5, each event will schedule another event (until the end)

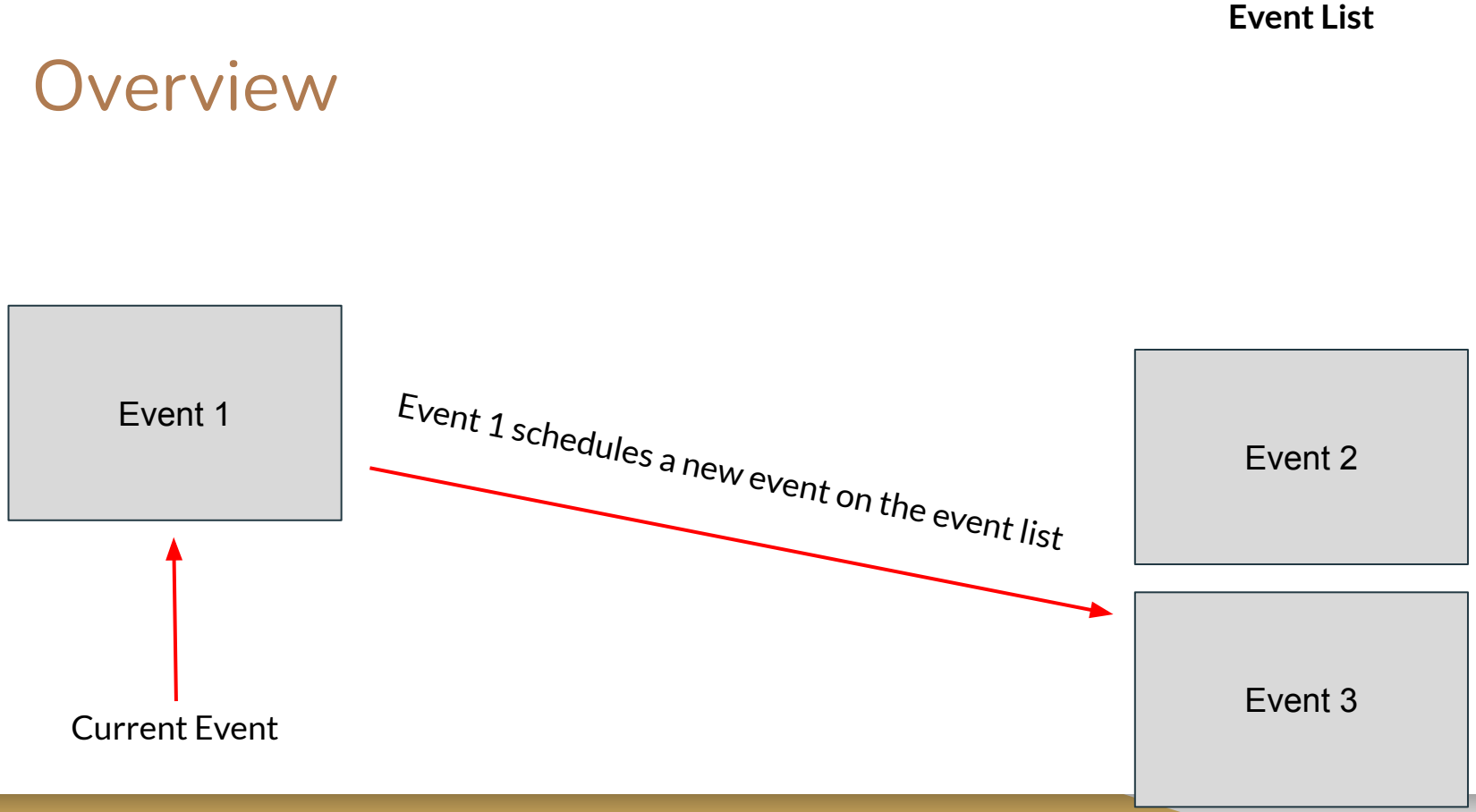


Overview

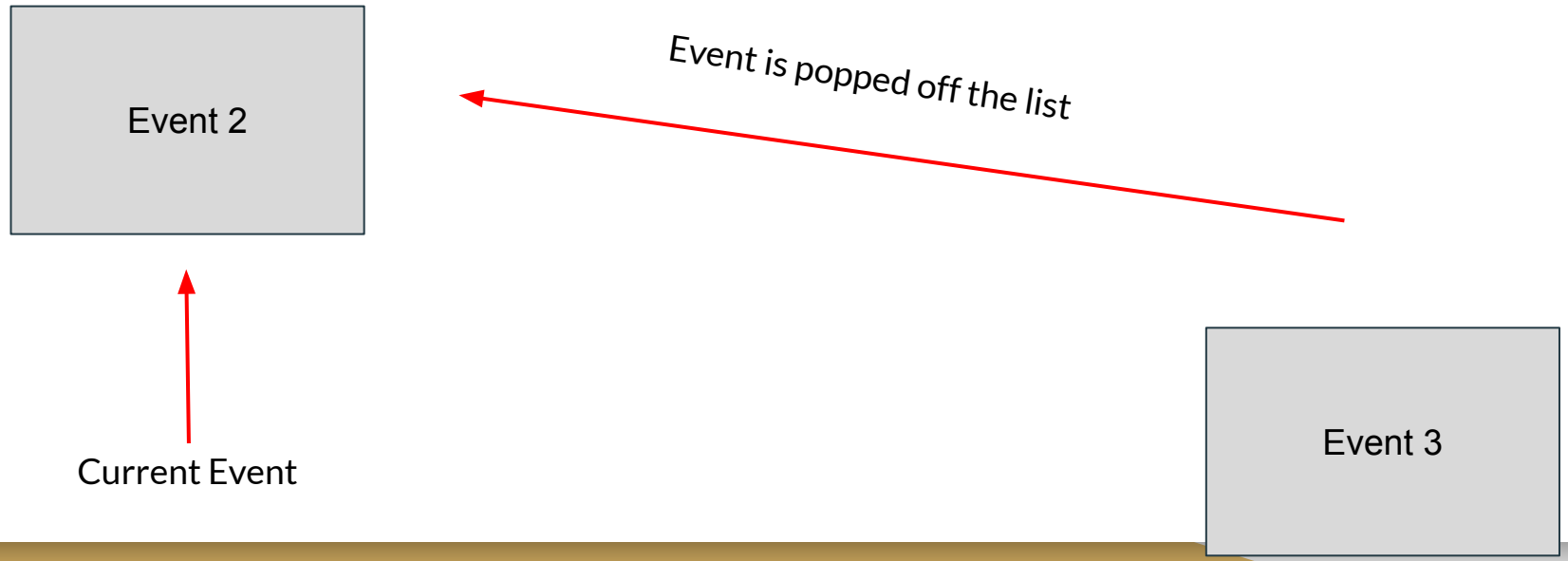
Overview


Event List
Event is popped off the list
Event 1
Event 2
Current Event

Overview


Event List
Event 1
Event 1 schedules a new event on the event list
Event 2
Event 3
Current Event

Overview

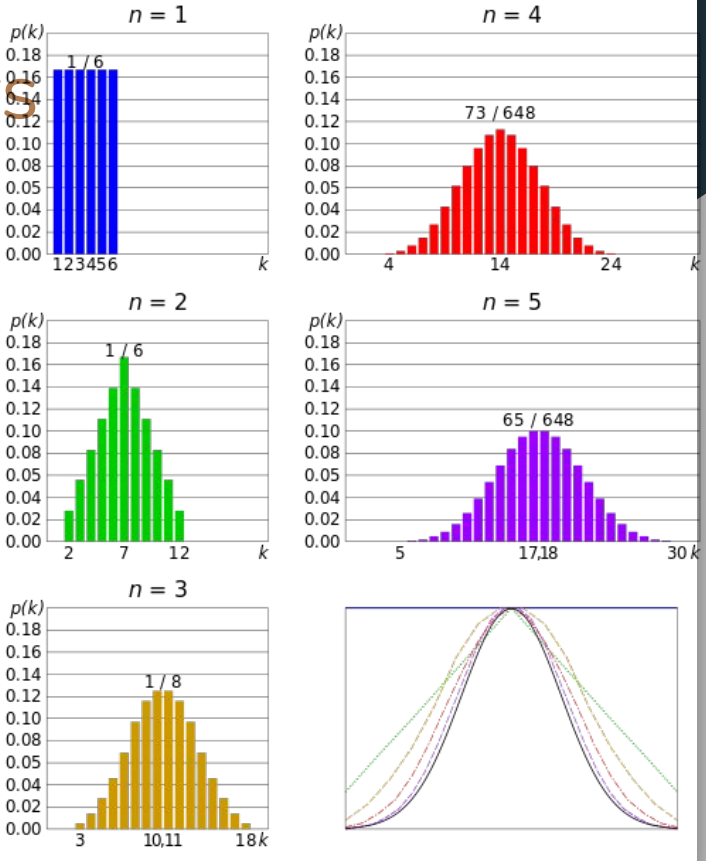

Event List
Event 2
Event is popped off the list
Current Event
Event 3

Random Numbers

- Library called <cstdlib>
- Important functions:
  - srand(int seed)
    - Returns a random integer based on the seed
  - rand()
    - Returns a random integer based on the seed (either set in srand or the default seed)

Advanced Random Numbers

- What if I want to generate random numbers according to some distribution?
- We could get a rough approximation by adding n different random numbers
- Example: Summing dice

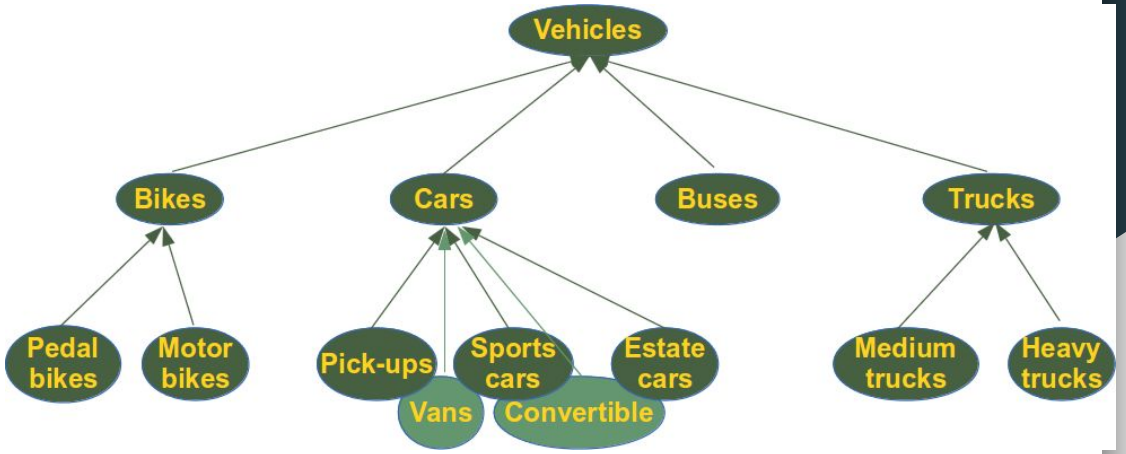


Inheritance

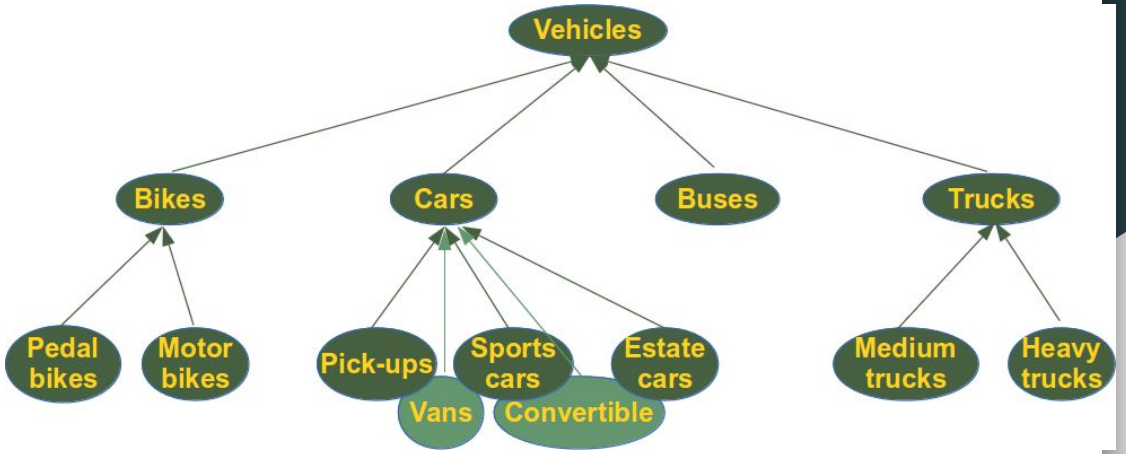
Principles of Object Oriented Programming

- Encapsulation
  - We want data and the functions that edit that data to be in the same place
  - Classes!
- **Inheritance**
  - **We want a way to relate sets of data and functionality to each other**
  - **Child / parent classes!**

Example


Vehicles
Bikes
Cars
Buses
Trucks
Pedal bikes
Motor bikes
Pick-ups
Sports cars
Estate cars
Vans
Convertible
Medium trucks
Heavy trucks

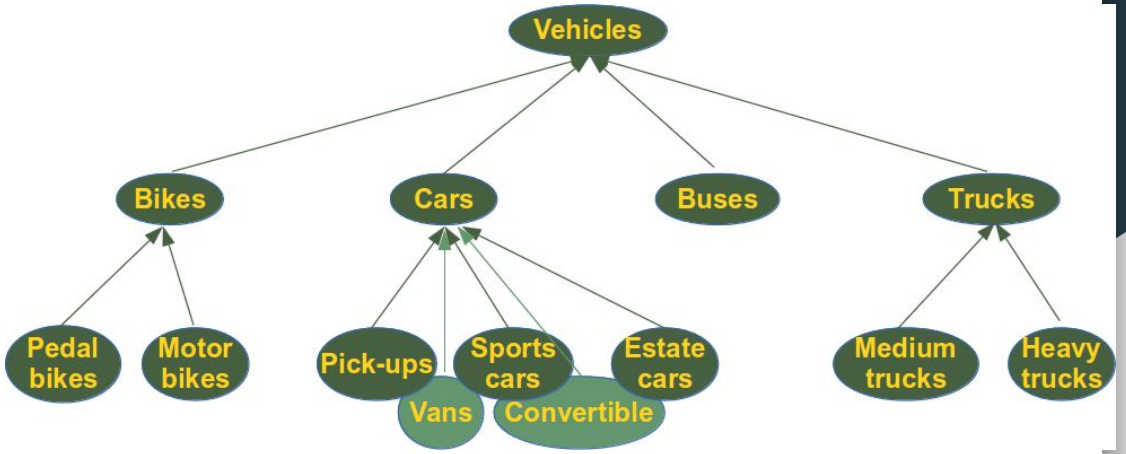
Example



Vehicle

- color
- numWheels

Example



Vehicle

- color
- numWheels

Bikes

- tireDiameter
- horn
- brake()

Example

Vehicle

- color
- numWheels

Bikes

- tireDiameter
- horn
- brake()

Motor bikes

- mpg
- engineCC
- accelerate()

Implementing in C++

1. In class header: class <ChildClass>: public <ParentClass>
    a. Nearly every time you use inheritance it will be public

Simple example

- Rectangle class inherits generic shape properties and functions
- Both getArea and setWidth/height are called in the same way despite being in different classes

```
// Base class
class Shape {
   public:
      void setWidth(int w) {
         width = w;
      }
      void setHeight(int h) {
         height = h;
      }

   private:
      int width;
      int height;
};

// Derived class
class Rectangle: public Shape {
   public:
      int getArea() {
         return (width * height);
      }
};
```

Simple example

```
int main(void) {
   Rectangle Rect;

   Rect.setWidth(5);
   Rect.setHeight(7);

   // Print the area of the object.
   cout << "Total area: " << Rect.getArea() << endl;

   return 0;
}
```

```
// Base class
class Shape {
   public:
      void setWidth(int w) {
         width = w;
      }
      void setHeight(int h) {
         height = h;
      }

   private:
      int width;
      int height;
};

// Derived class
class Rectangle: public Shape {
   public:
      int getArea() {
         return (width * height);
      }
};
```