CSE1030 – Introduction to Computer Science II

Lecture #17

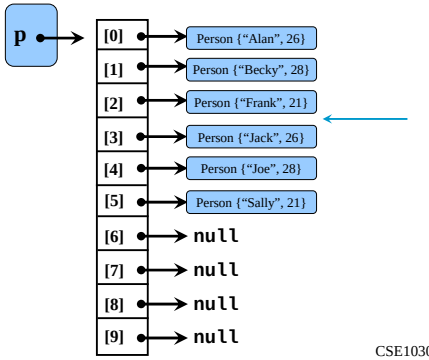
Introduction to Linked Lists

CSE1030 – Lecture #17

- Review
- Introduction to Linked Lists
- We're Done!

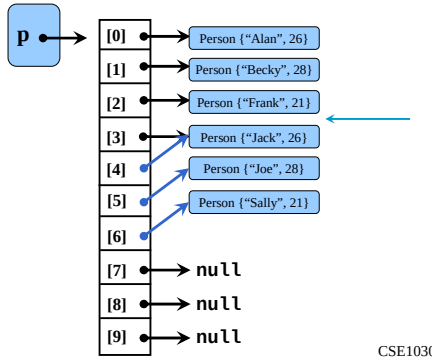
Remember the Array Insertion Problem?

- Want to add "Henry" but **Preserve** the **Order**



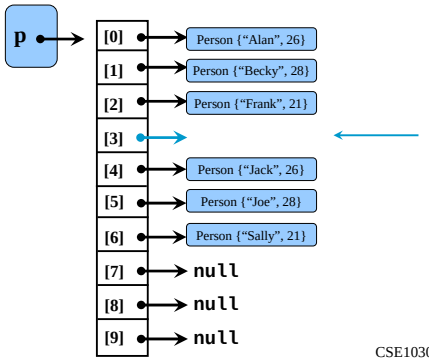
Need to Insert a Gap…

- (Which is Time Consuming)



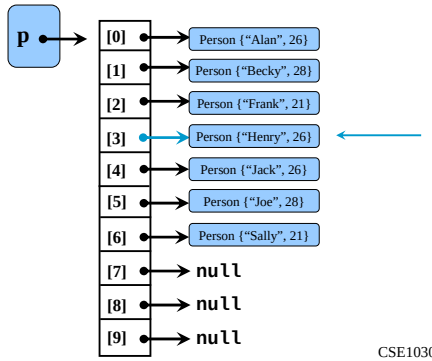
Need to Insert a Gap…

- (Which is Time Consuming)



Need to Insert a Gap…

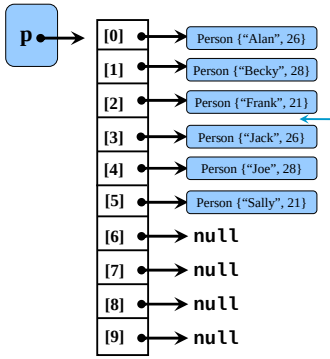
- (Which is Time Consuming)



CSE1030 – Lecture #17

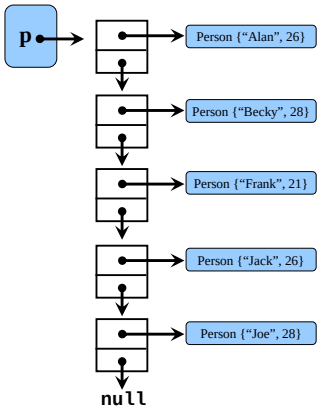
- Review
- Introduction to Linked Lists
- We’re Done!

Is there a way to do this Without Shifting?



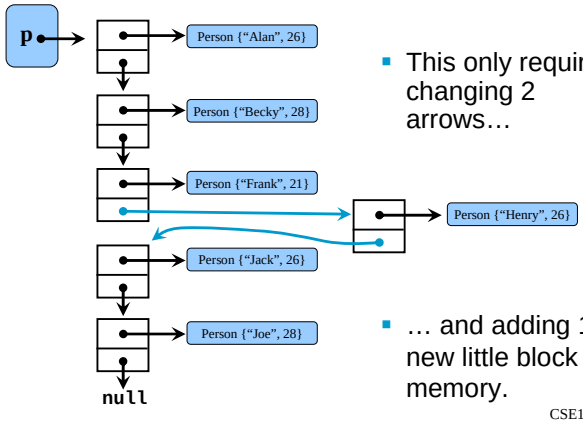
- The problem is that the array is a single contiguous block of memory
- Instead, if it was a series of little blocks, then we wouldn't have to shift anything…

How about we use lots of Little Blocks of Memory, instead of 1 Big one?



- Each Little Block holds an arrow (reference, pointer) to the data
- Each Little Block also has to provide a way to find the next little block

Now, Inserting Henry is Easy!

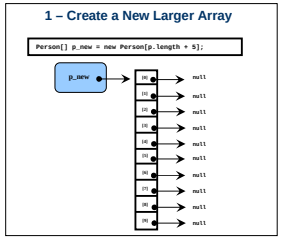


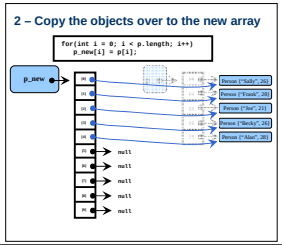
- This only required changing 2 arrows…
- … and adding 1 new little block of memory.

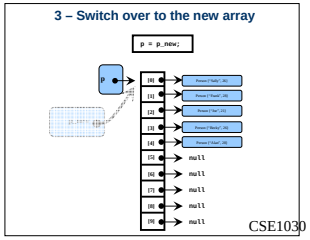
Implications

- We are still storing a collection of arrows (or "references", or "pointers") as we did when we used arrays
- But because the arrows are in their own individual little pieces of memory, nothing has to be shifted to insert new ones
- There are other benefits too…

More Implications

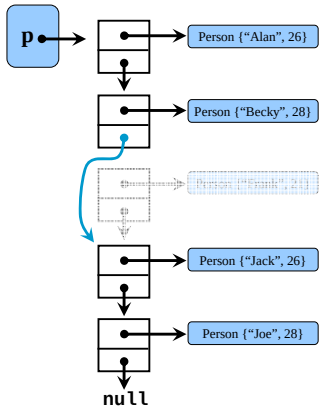






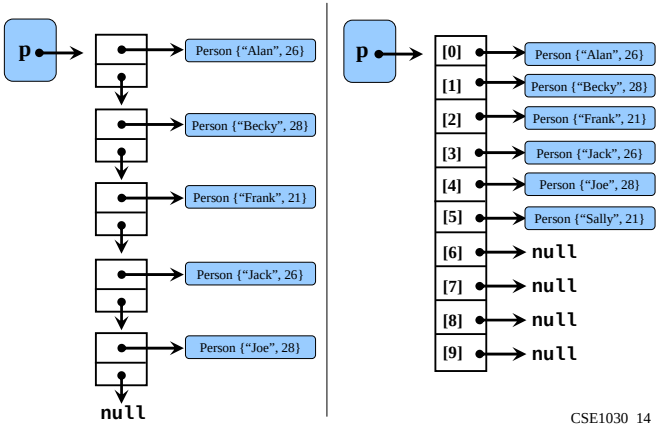
- Remember what a pain **Array Resizing** was?
- And **Inefficient** too
- There is **No Resizing** with the New Approach!

Deletion is also Easy



- Deleting "Frank" only requires us to update 1 pointer – Fast!

Negative Implications? Accessing Data

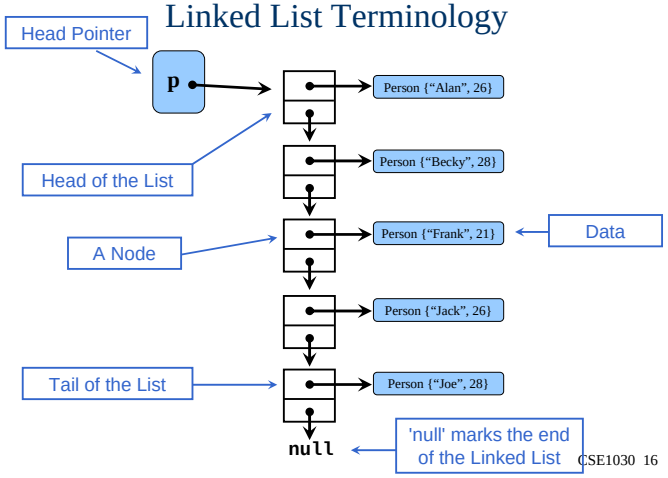


Arrays versus Linked Lists

- Good:
  - Access to any element is very fast: `p[i]`
  - Adding / Deleting from the **End** is **Fastest** (but can cause Resizing)
  - Efficient on Memory (only 1 arrow per Data item)
  - But empty slots waste memory
- Bad:
  - Insertion / Deletion anywhere but the end of the array
  - Resizing

- Good:
  - Insertion / Deletion is easy (just update some arrows)
  - Insertion or Deletion at the **Top** of the list is **Fastest**
  - There is no "Resizing Cost"
- Bad:
  - Accessing en element requires us to iterate along the List – Slower than array
  - Wastes more Memory (2 arrows per Data item)
  - Although it, doesn't have empty slots

Linked List Terminology



CSE1030 – Lecture #17

- Review
- Introduction to Linked Lists
- We're Done!

Next topic…

Linked Lists II