Linked Lists

•••

Recitation 4/13

Allocation Types

Local/Automatic Variables: Inside of blocks, functions, etc. Stored in the call stack (end of IWRAM on the GBA)

Global/Static Variables: Declared with static keyword or outside functions and blocks. Stored at the beginning of IWRAM on the GBA

Dynamic: Managed using functions like `malloc` and `free`

Dynamic Memory

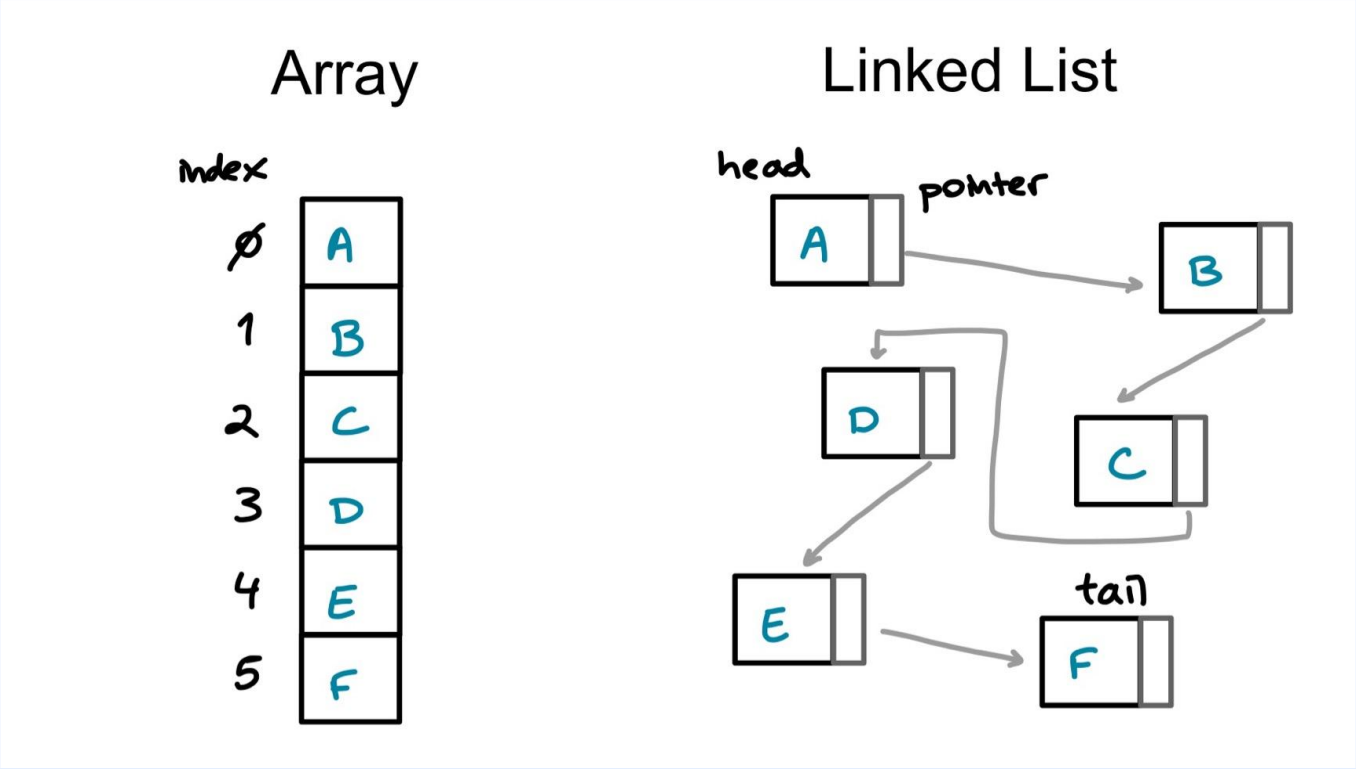
Static variables are persistent, but must be **pre-allocated** before the program starts

Automatic variables are allocated whenever, but are **temporary** (in the call stack)

Dynamic memory is **persistent** and can be **allocated at runtime**

- **We’re going to use this for linked lists!**

Arrays vs Linked Lists

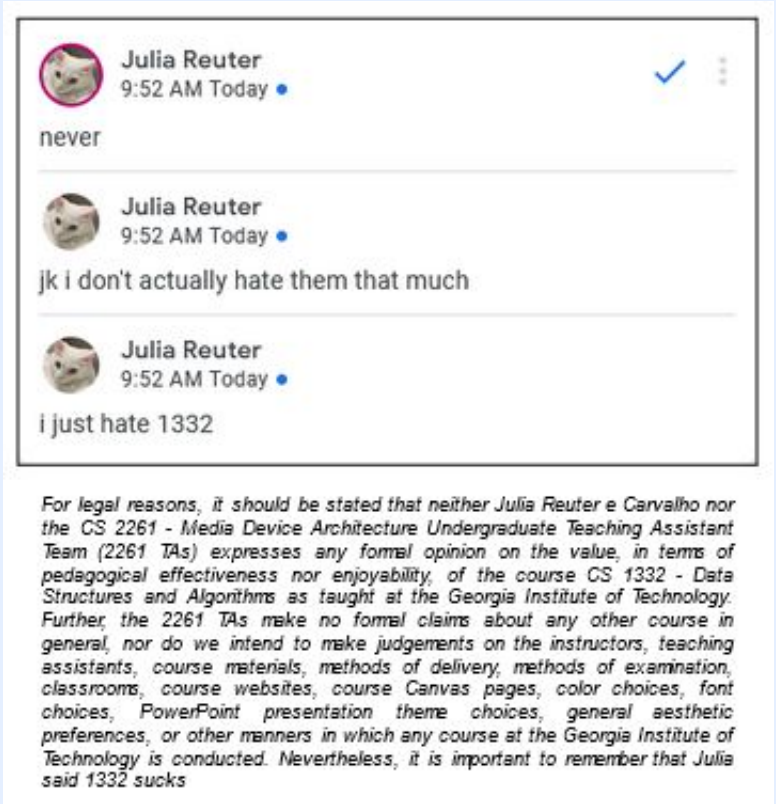


Arrays vs Linked Lists

Arrays	Linked Lists
All elements stored contiguously in memory, must be resized	Elements stored at different parts of memory, no need for resizing
O(1) access: indexing (arr[x])	O(n) access except at head/tail: traversal
Slow deletions and insertions except at end	No need for resizing, deletions are faster
Native to C	Must be implemented by the programmer (unless you use a library!) [which we don’t allow :)]

When Linked List?

- Helpful if you don’t know the amount of elements you’ll need since you won’t need to resize it later
- Especially good if you know most or all of your operations occur at the head or tail
- Really good for backing lot of other data structures (e.g. Stack, Queue)



When Array?

- When not Linked List

Flavors of Linked List

Singly-Linked: Each node points to next. Boring, simple. Vanilla.

Doubly-Linked: Each node points to next AND previous. Cookies and cream. This is what you will implement in the lab!

Circular: Last node's next node is the head. Kinda neat. Pistachio (according to Julia.)

Remember how to insert a node at the front in O(1) time?

Very Important Table

Type of Linked List	Equivalent Ice Cream Flavor
Singly-Linked	Vanilla
Doubly-Linked	Cookies and Cream
Circular	Pistachio (According to Julia)

Coding Time

Whatever you do don't look in **`SUPER SECRET NO LOOKING ALLOWED.txt`**!!!!

(Feel free to use it if you get behind but try to follow along with me if you can)

Coding Time

(Activity 1.0) Complete `createList`

```
1  createList:
2      list = malloc(size of list struct)
3      if list is NULL:
4          print a warning
5      else:
6          list.head = NULL
7          list.tail = NULL
8      return list
```

Coding Time

(Activity 1.1) Complete `createNode`

```
1  createNode(data):
2      node = malloc(size of node struct)
3      if list is NULL:
4          print a warning
5      else:
6          node.data = data
7          node.next = NULL
8          node.prev = NULL
9      return node
10
```

Coding Time

(Activity 1.2) Complete pushBack

```
1  pushBack(dllist, data):
2      node = createNode(data)
3      if dllist.head is NULL: // list isn't empty
4          dllist.tail.next = node
5          node.prev = dllist->tail
6      else: // list is empty
7          dllist.head = node
8      return node
9
```

Coding Time

(Activity 2.0) Set `currentNode` to the head of the list and then build and run

(Activity 2.1) Move the pointer forward/backward when keys are pressed

The Lab

You'll need to complete a few additional functions for the lab: the rest of these slides are pseudocode to help you do this!

deleteList

```
1 deleteList(dllist):
2     while list head is not null:
3         pop front node of list
4
5     free dllist
6
```

updateNodePositions

This one is specific to our implementation of Snake (it's what actually makes the snake move)

```
1  updateNodePositions(dllist):
2      curr = tail of dllist
3      while curr is not null and curr.prev is not null:
4          curr.position = curr.prev.position
5          curr = curr.prev
6
```