Short-term Hands-on Supplementary Course on C Programming

SESSION 13: Structures and Files

KARTHIK D
NIVEDHITHA D

Mode: **Asynchronous**
Location: Online

Administrative Instructions

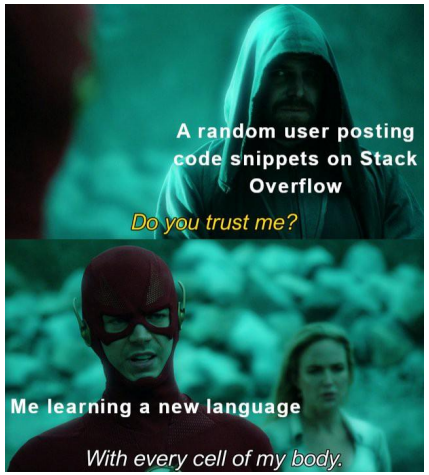
- Please fill out the feedback form - will be shared on our group.

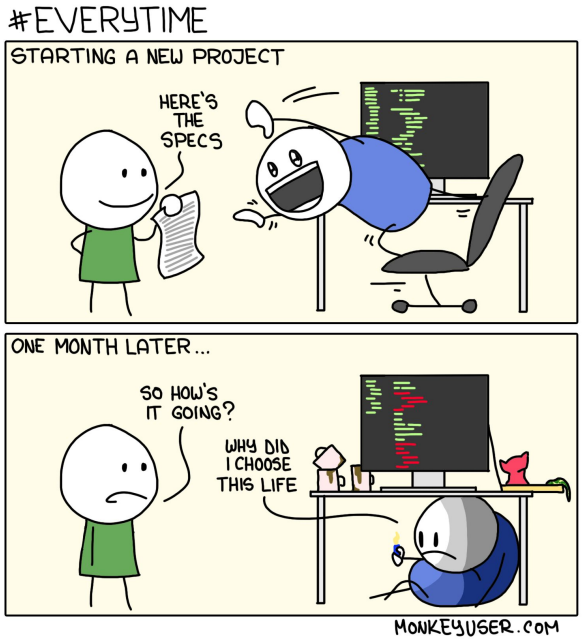
- Join us on Microsoft Teams,
  Team Code: **rzlaicv**

GITHUB REPOSITORY!** ⭐🌟

Nearing the End !!!

CAPSTONE PROJECT !!!!!!



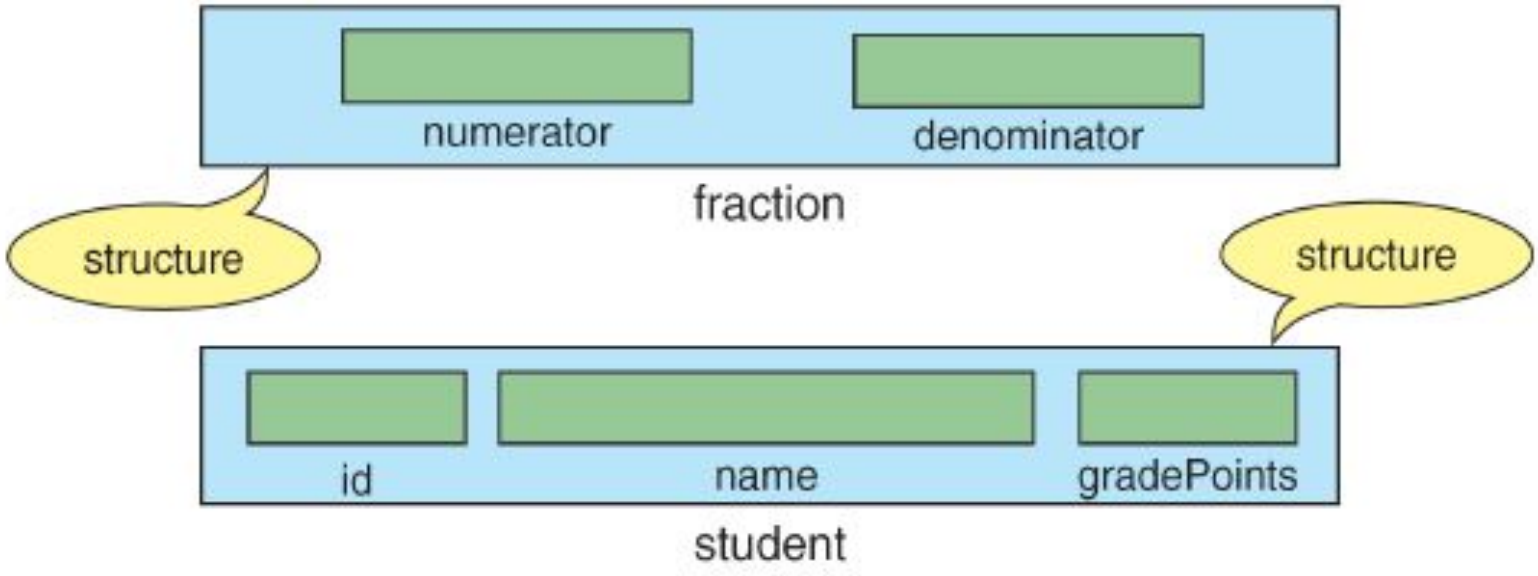


Agenda

1. Pointers to Structures
2. Dynamic Initialization of Structures
3. Self-referential Structures
4. Linked Lists using Structures
5. Structures and Files
6. Tutorial: Store a Linked List in a File
7. Next Session: The Capstone Project

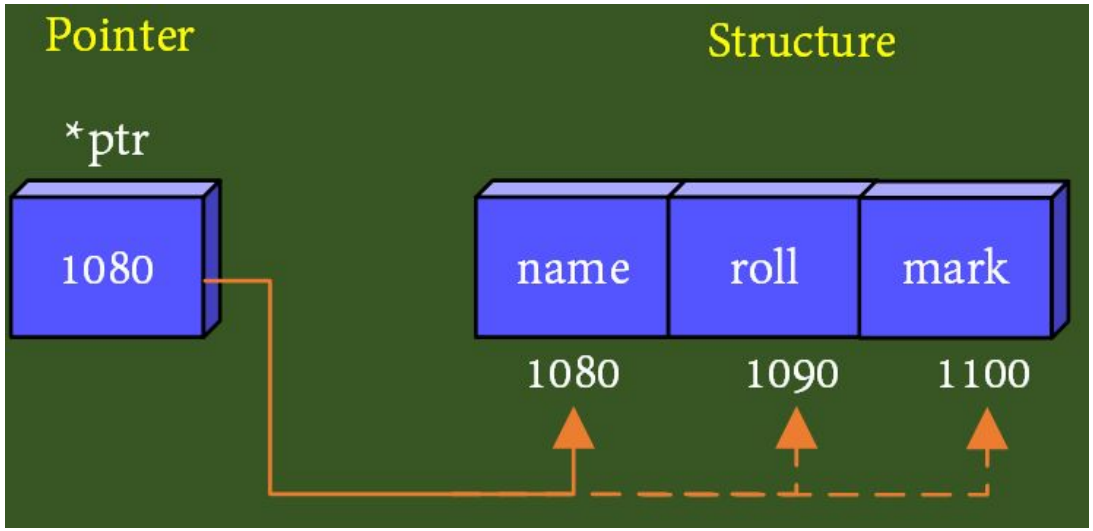
Recap: Structures in C

A **structure** in C is a user-defined data type. It is used to bind the two or more similar or different data types or data structures together into a single type. Structure is created using struct keyword and a structure variable is created using struct keyword and the structure tag name.

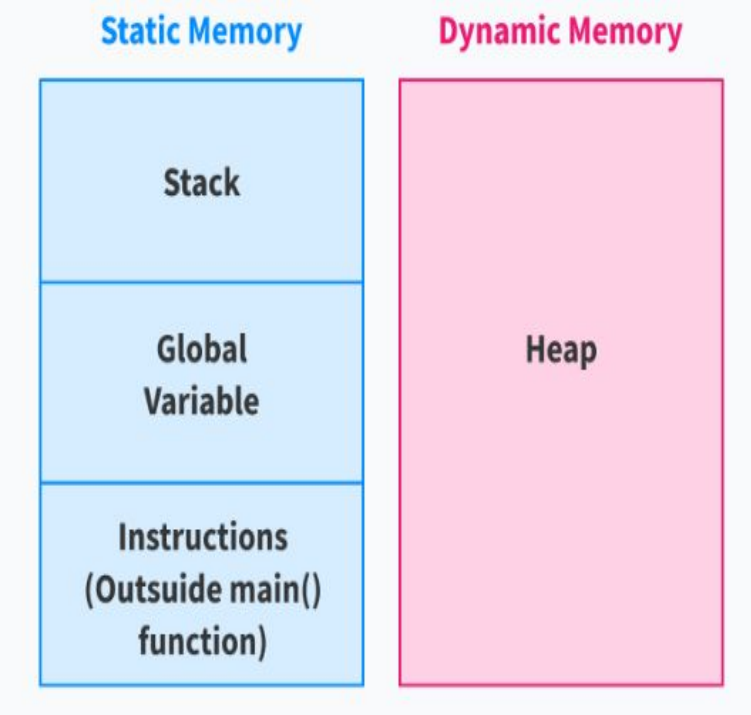


Pointers to Structures

- Declaration of structure pointer
- Initialization of struct pointer
- Accessing members through pointer
  - (*) asterisk and (.) operators
  - **(->) arrow operator** [recommended]



Dynamic Init of Structures

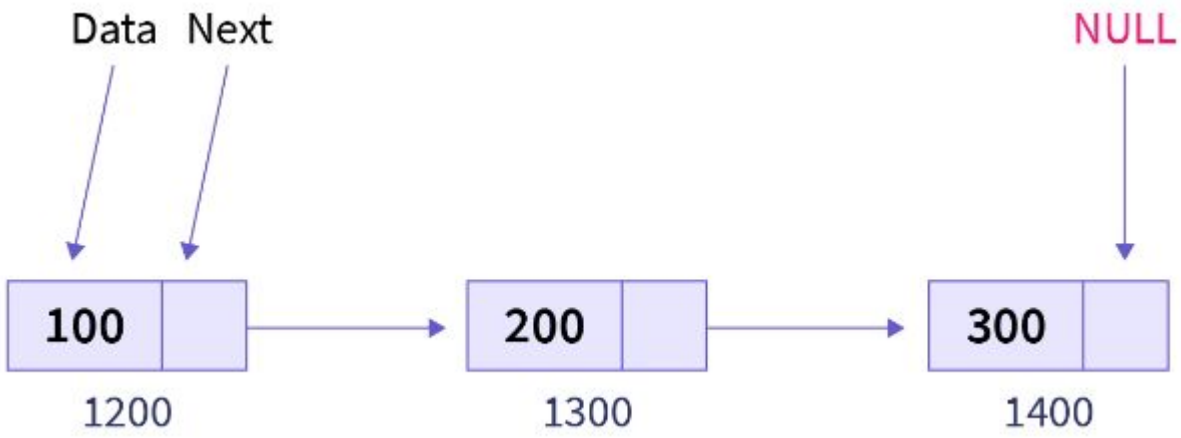


malloc() method	calloc() method
It is generally used to allocate a **single** memory block of the given size (in bytes) during the run-time of a program.	It is generally used to allocate contiguous (**multiple**) blocks of memory of given size (in bytes) during the run-time of a program.
It **doesn't initializes** the allocated memory block and contains some garbage value.	It **initializes** all the allocated memory blocks with 0 (zero) value.
malloc() takes a **single arugment** i.e. the size of memory block that is to be allocated.	calloc() takes **two arguments** i.e. the number of elements and the size of one element that are to be allocated.
syntax: **(cast-data-type *)malloc(size-in-bytes)**	syntax: **(cast-data-type *)calloc(num, size-in-bytes)**
Example : float *ptr = (float *)malloc(sizeof(float));	Example : float *ptr = (float *)calloc(10, sizeof(float));

Self-referential Structures

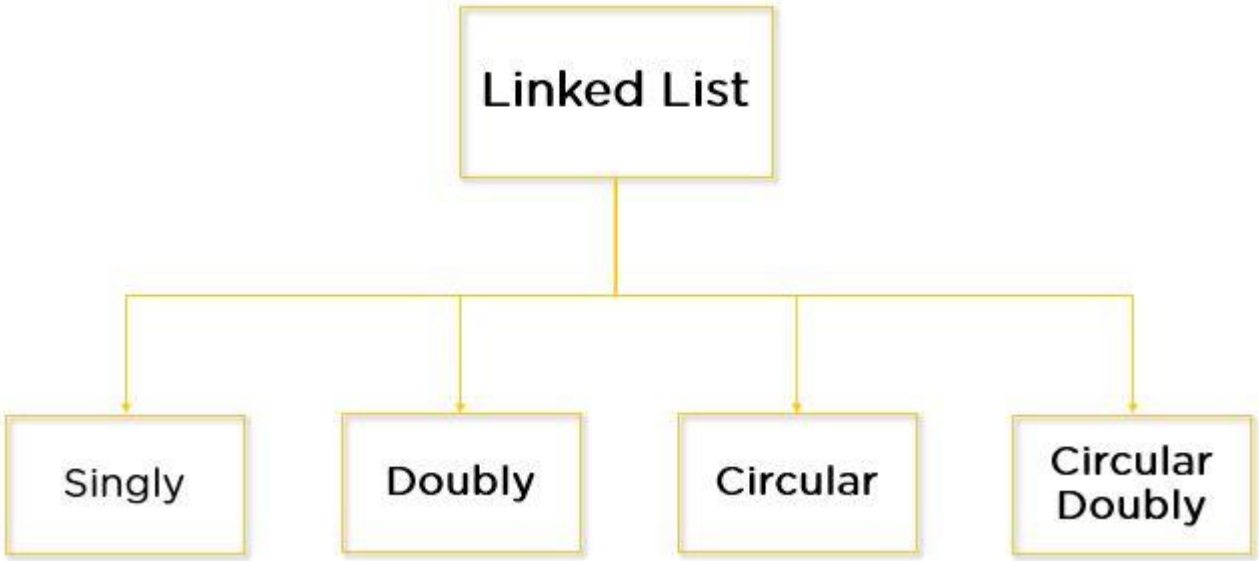
```c
struct Node
{
        int data;
        struct Node* next;
};
```

```c
struct Node* node1 = (struct Node*)malloc(sizeof(struct Node));
```



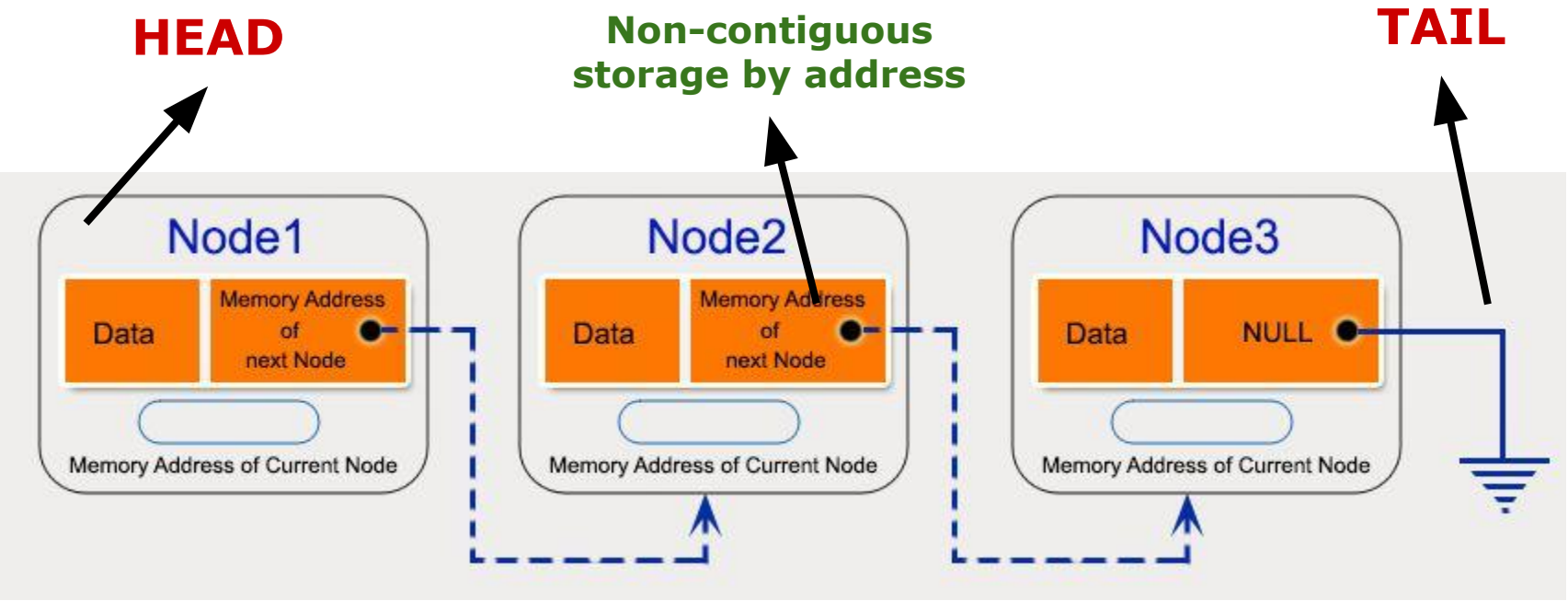
Linked Lists

- A **linear data structure**.
- Unlike arrays, elements are **NOT stored contiguously**.
- Elements are usually regarded as **nodes**.
- Nodes can store **complex**, **custom data formats** — defined by the underlying self-referential structure.
- Linked lists can be classified based on **how their nodes are linked** together.



Bonus: Singly Linked Lists

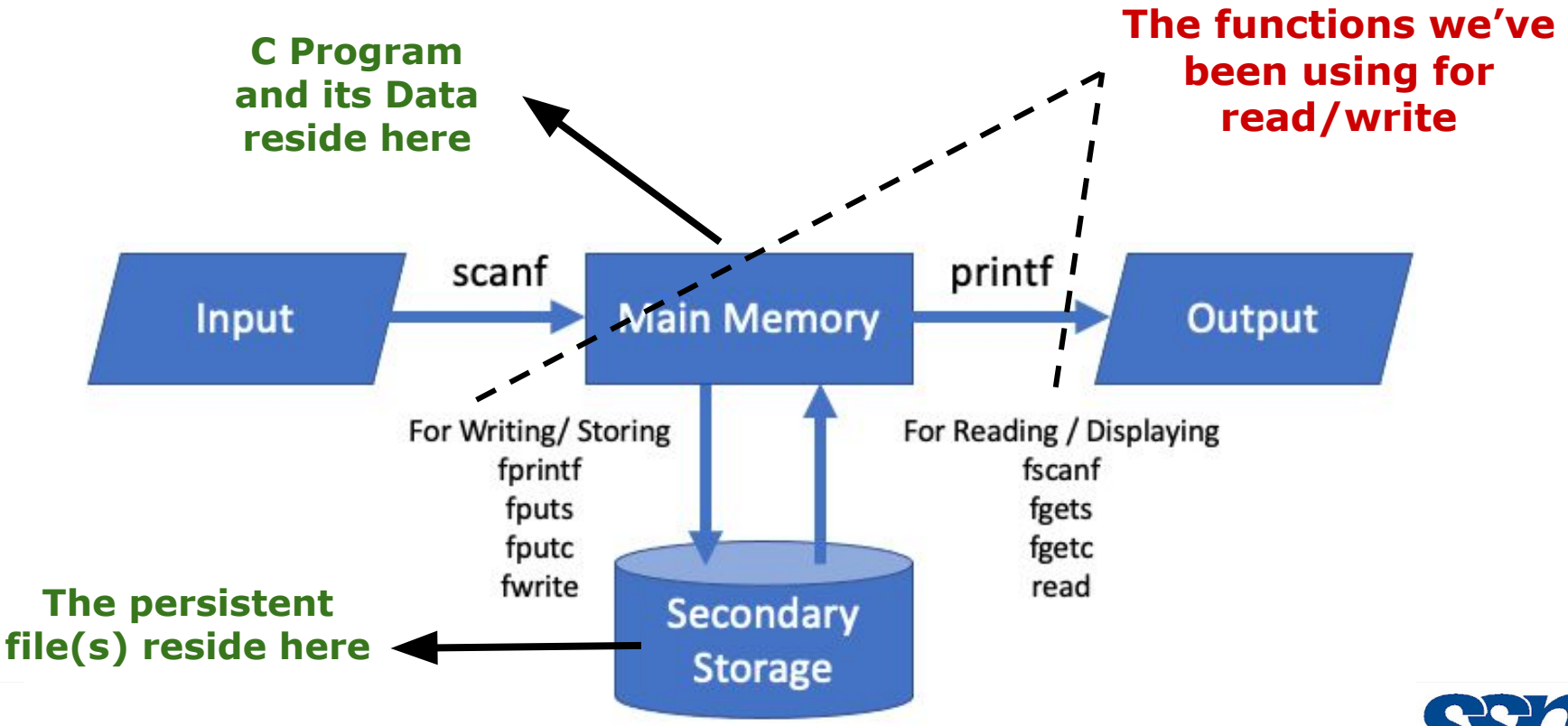
- **Unidirectional linked list** — each node only points to its next node.
- Can be **traversed in one direction only**, i.e., from head node to tail node.



- We'll now look at an *implementation of singly linked lists*, using self-referential structures

Recap: File Handling

- Files are used to **store data persistently** between executions of a program
- Basic steps in file manipulation — ***open a stream***, ***read/write***, ***close the stream***
- File access modes, position seeking, …



Structures and Files

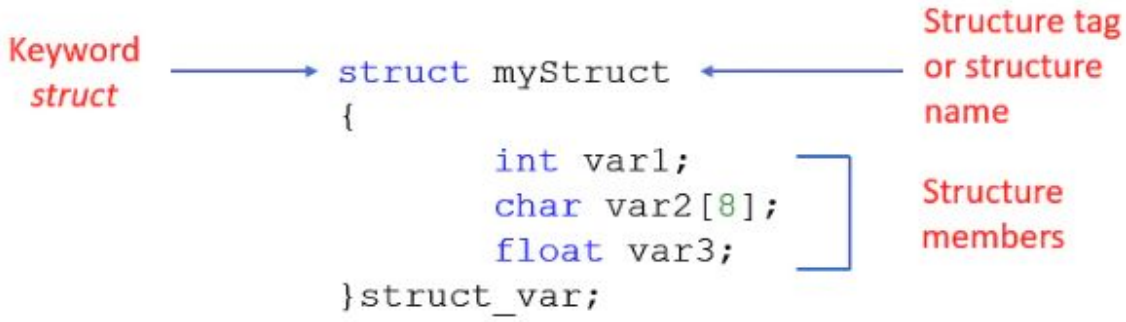
- Remember that there are 2 file data formats — **text and binary**
- Binary files are ideal to store structures in files

- A few implementation details:
  - Can't store pointer attributes directly — this requires additional measures, such as **data serialization**
  - Use the binary data read/write functions —

```
size_t fwrite( const void *buffer, size_t size, size_t count, FILE *stream );
size_t fread( const void *buffer, size_t size, size_t count, FILE *stream );
```

Structures and Files

- Remember how structures are represented in memory?



- The writing program, **encodes memory segment** into **bit-sequence**
- When written to a file, this **entire memory segment is stored as bits** in the binary file
- The reading program, **decodes bit-sequence** into structure's **memory segment** — the *structure definition must be known*!

TUTORIAL

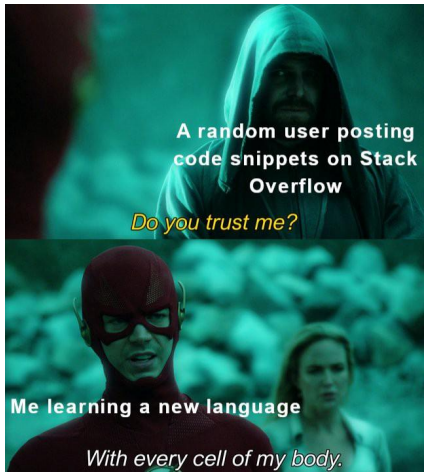
IMPLEMENTING SINGLY-LINKED LISTS

STORING and ACCESSING LINKED LISTS from FILES

- We **serialize** the list into nodes, when **storing** to the file
- We **reconstruct** the list, when **reading** from the file

- Of course, there are ***other serialization strategies*** that you could come up with!

Next Session



CAPSTONE PROJECT !!!!!!