CS 261
Data Structures

Lecture 5

Function Pointers

Dynamic Array

6/28/22, Tuesday

Oregon State University

Lecture Topics:

- Function Pointers
- Dynamic Array
- Begin Linked List

C Basics – Function pointers

- To use this function pointer
  - the calling function will need access to a function for *comparing* elements, i.e., integers
  - This function will have to match the prototype of the function pointer argument to our sort()
  - E.g.,

```
int compare_ints(void* a, void* b) {
  int* ai = a, *bi = b;  /* Cast void* back to int*. */
  if (*ai < *bi)
    return 0;
  else
    return 1;
}
```

  - Function call will be:

```
sort((void**)array_of_ints, number_of_ints, compare_ints);
```

C Basics – Function pointers

```
void sort(void** arr, int n, int (*cmp)(void* a, void* b));
```

- Within sort():
  - Whenever we need to compare two values from the array being sorted, we can just call cmp()

```
if (cmp(arr[i], arr[j]) == 0) {
  /* Put arr[i] before arr[j] in the sorted array. */
}
else {
  /* Put arr[i] after arr[j] in the sorted array. */
}
```

- Demo….

FYI: GDB Setup

In your home directory, type:

`python /nfs/farm/classes/eecs/spring2021/cs161-001/public_html/gdb/set_up.py`

- And answer **‘y’** to the question (as follows):

```
flip1 ~ 169% python /nfs/farm/classes/eecs/spring2021/cs161-001/public_html/gdb/set_up.py
--2021-05-16 21:12:24--  http://classes.engr.oregonstate.edu/eecs/spring2021/cs161-001/gdb/gdbinit
Resolving classes.engr.oregonstate.edu (classes.engr.oregonstate.edu)... 128.193.40.12
Connecting to classes.engr.oregonstate.edu (classes.engr.oregonstate.edu)|128.193.40.12|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 279 [text/plain]
Saving to: ‘/nfs/stak/users/songyip/.gdb/gdbinit’

100%[=================================================================================================>] 279         --.-K/s   in 0s

2021-05-16 21:12:24 (28.8 MB/s) - ‘/nfs/stak/users/songyip/.gdb/gdbinit’ saved [279/279]

--2021-05-16 21:12:24--  http://classes.engr.oregonstate.edu/eecs/spring2021/cs161-001/gdb/gdb_dashboard.py
Resolving classes.engr.oregonstate.edu (classes.engr.oregonstate.edu)... 128.193.40.12
Connecting to classes.engr.oregonstate.edu (classes.engr.oregonstate.edu)|128.193.40.12|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 64591 (63K) [text/plain]
Saving to: ‘/nfs/stak/users/songyip/.gdb/gdb_dashboard.py’

100%[=================================================================================================>] 64,591      --.-K/s   in 0s

2021-05-16 21:12:24 (138 MB/s) - ‘/nfs/stak/users/songyip/.gdb/gdb_dashboard.py’ saved [64591/64591]

Do you want to install peda to ~/.gdbinit (y/n) ?
y
```

FYI: GDB Setup (cont.)

Once setup successfully, you will have a .gdb folder and a .gdbinit file under your home directory, and you can verify it with:

- `ls .gdb`
- `cat .gdbinit`

```
flip1 ~ 170% ls .gdb
gdb_dashboard.py  gdbinit
flip1 ~ 171% cat .gdbinit
set auto-load safe-path /
source ~/.gdb/gdb_dashboard.py
set history save
set verbose off
set print pretty on
set print array off
set print array-indexes on
set python print-stack full
python Dashboard.start()
dashboard -layout registers assembly source stack memory expressions
```

FYI: Using GDB

- Compile with debugging symbols (-g flag), e.g.:

```
gcc -std=c99 filename.c -g -o exe_name
```

- Run it with GDB:

```
gdb ./exe_name
```

FYI: Common GDB Commands

1. `break` – set up break points, e.g.: `b *main`   `break 10`
2. `run` – begin execution (until a break point)
3. `print` – see the values of data, e.g. `print i`   `print &ptr`   `print &main`
4. `next` and `step` – step line by line through the program
5. `continue` – continue until a break point OR the end of the program
6. `backtrace` – prints a backtrace of all stack frame (locate seg fault!!!)
7. x/**100**wx [address or register] – read memory
    - Examine
    - **100** values
    - sized as word (w, 4 bytes)
        - b – byte
        - g – 8 bytes
    - **In hexadecimal (x)**
        - d - decimal

Lecture Topics:

- Function Pointers
- Dynamic Array
- Begin Linked List

Abstract Data Type (ADT)

- Abstract Data Type (ADT) – a mathematical model for data types
- Specifies:
  - the type of data stored
  - the operations supported on them
  - the types of parameters of the operations.

- Why “abstract”?
  - an implementation-independent view of the data type

Dynamic Arrays

- Elements in an array are stored in a contiguous block of memory
- Allow random access (direct access)
  - i.e., time to access the 1st element = time to access the last element
  - By using array subscript ([]):

```
int* array = malloc(1000 * sizeof(int));
array[0] = 0;
array[999] = 0;
```

Dynamic Arrays (cont.)

- Basic operations:
  - get – Gets the value of the element stored at a given index in the array
  - set – Sets/updates the value of the element stored at a given index in the array
  - insert – Inserts a new value into the array at a given index.
    - Sometimes, dynamic array implementations limit insertion to a specific location in the array, e.g. only at the end.
  - remove – Removes an element at a given index from the array
    - Sometimes, dynamic array implementations avoid moving elements up a spot by only allowing the last element to be removed

Dynamic Arrays (cont.)

- Drawbacks:
  - Fixed size, must be specified when the array is created
    - For static array:

```
int array[50];
```

    - For dynamic array:

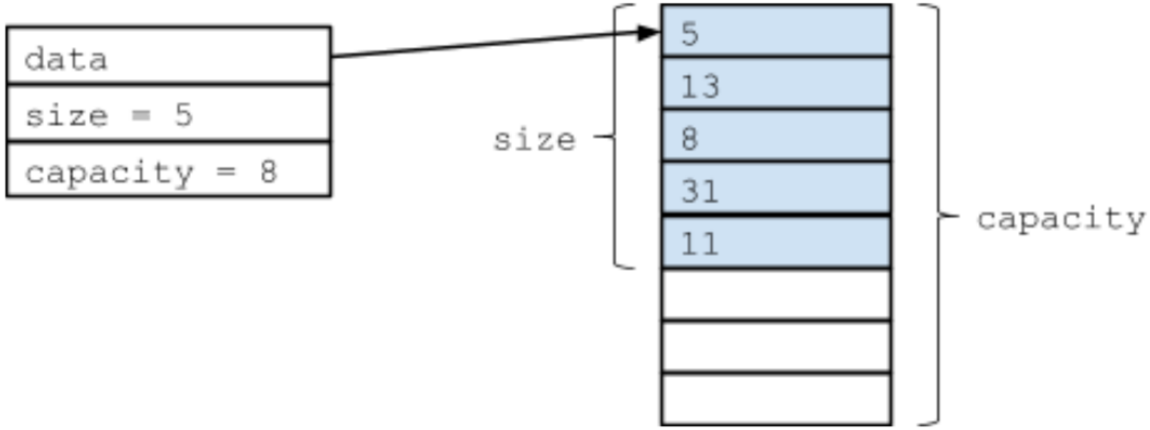
```
int *array = malloc (50 * sizeof(int));
```

→Need to allocate more memory if we need to store more data

- How?

- Dynamic array DS doesn't have a fixed capacity
  - Has a variable size and can grow as needed

Dynamic Arrays (cont.)



- Need to keep track of three things:
  - data – underlying data storage array
  - size – number of elements currently stored in the array
  - capacity – number of elements data has space for before it must be resized
- How it works?
  - An array of known capacity is maintained by the dynamic array DS.
  - As elements are inserted, they are simply stored in data
  - If an element is inserted into the dynamic array, and there isn't capacity for it in the underlying data storage array (data), the capacity of the underlying data storage array is doubled. Then the new element is inserted into this larger data storage array.

Dynamic Arrays

5	

5	8

5	8	1	

5	8	1	4

5	8	1	4	9			

5	8	1	4	9	0		

5	8	1	4	9	0	6	

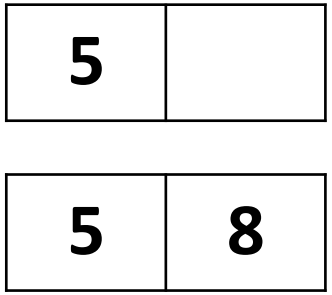
5	8	1	4	9	0	6	7

5	8	4	9	0	6	7	

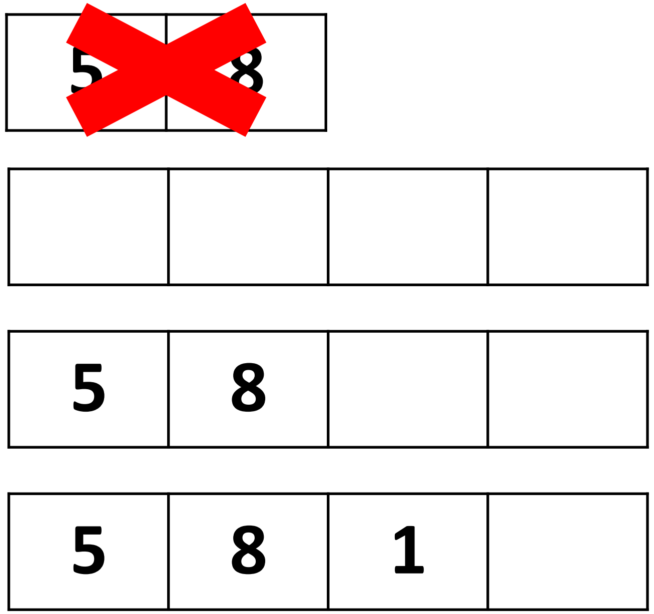
5	8	4	9	6	7		

Inserting an element into dynarray

- Case 1: if size < capacity
  - At least one free spot in data
  - Insert the new element

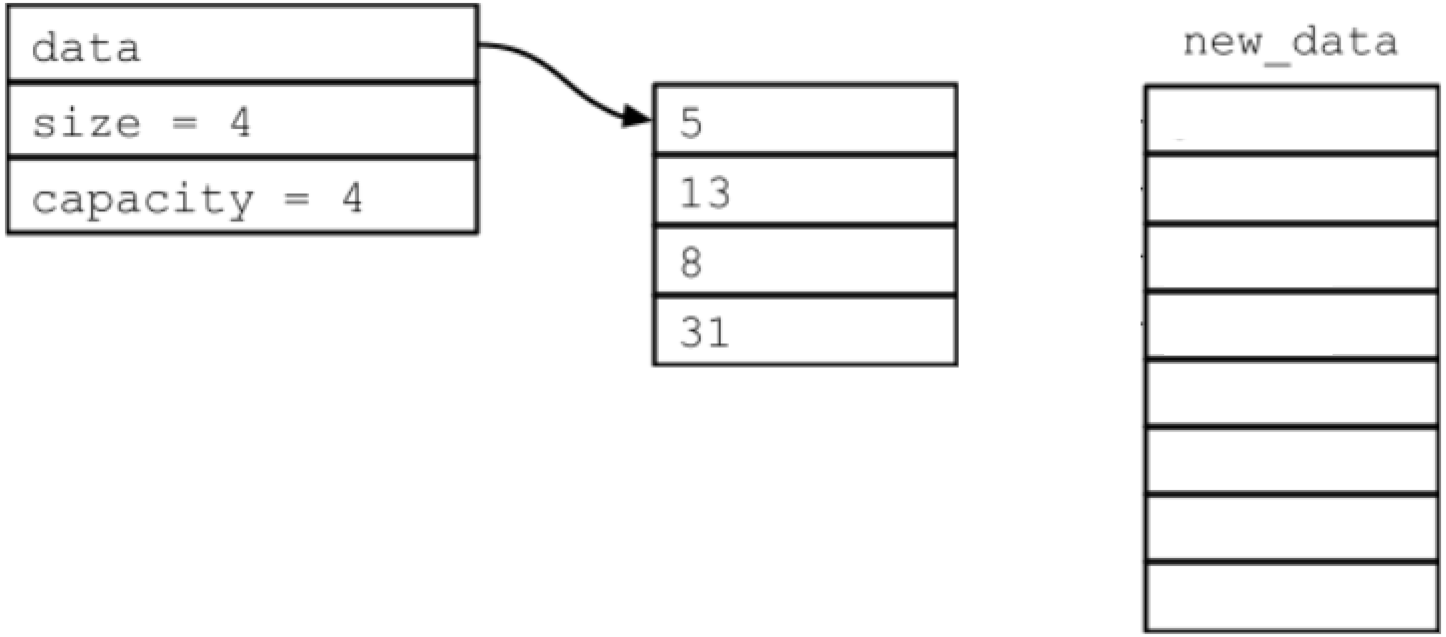


- Case 2: if size == capacity
  - No free spot in data
  - Step 1: allocate a new array that has twice the capacity
  - Step 2: copy all elements from data to new array
  - Step 3: delete the old data array
  - Step 4: Insert the new element



Another Example

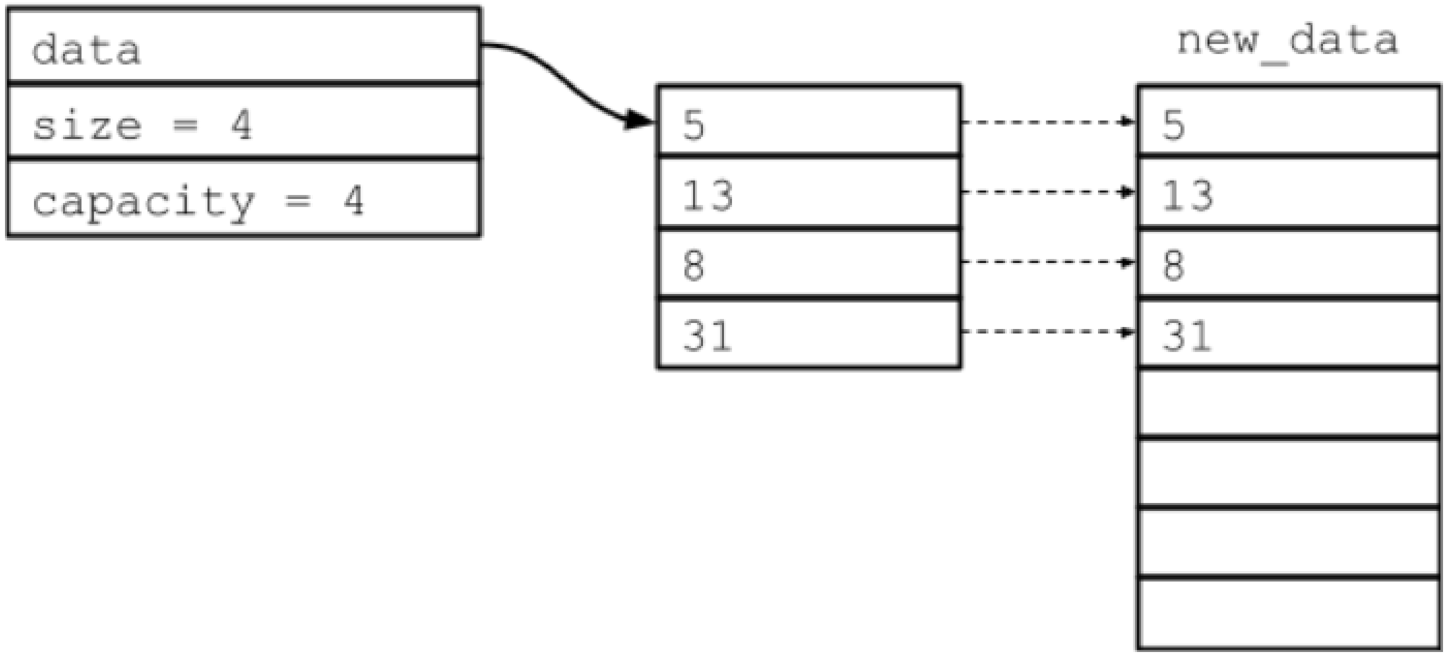
- Insert 16 to the following dynamic array:



- Step 1: allocate a new array that has twice the capacity

Another Example

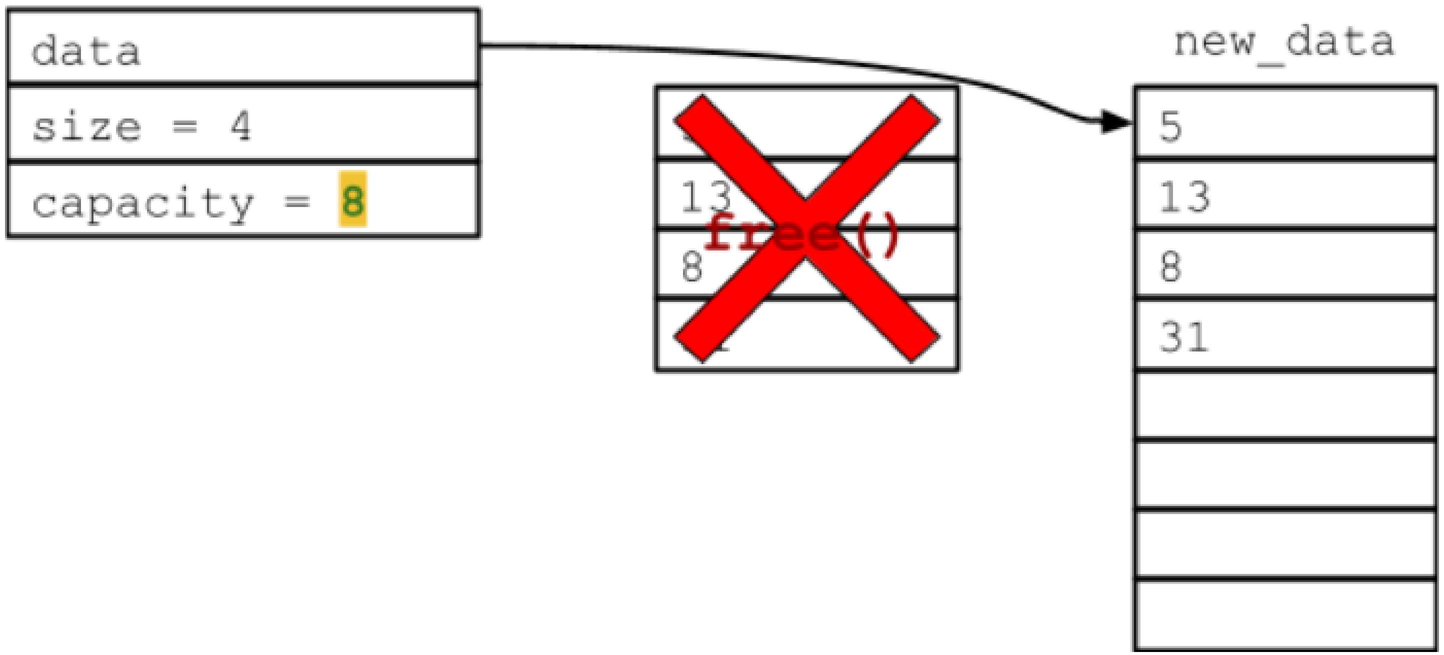
- Insert 16 to the following dynamic array:



- Step 2: copy all elements from data to new array

Another Example

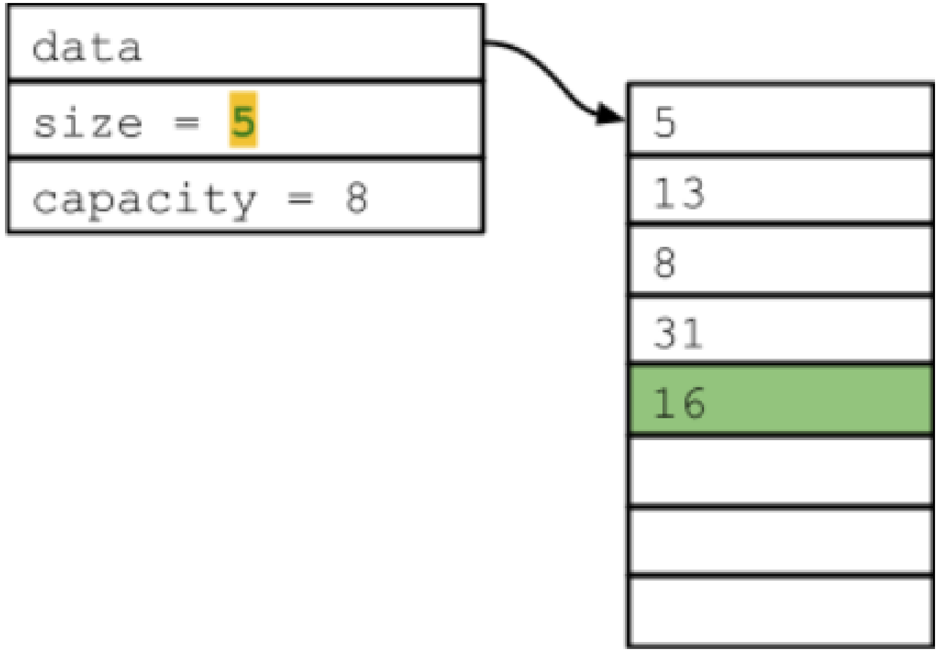
- Insert 16 to the following dynamic array:



- Step 3: delete the old data array and update data

Another Example

- Insert 16 to the following dynamic array:



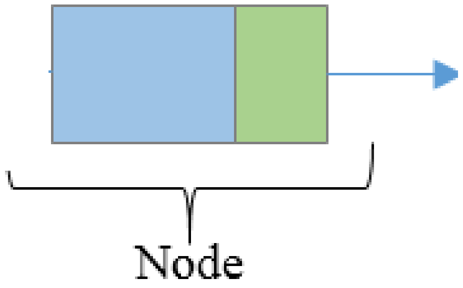
- Step 4: Insert the new element

Lecture Topics:

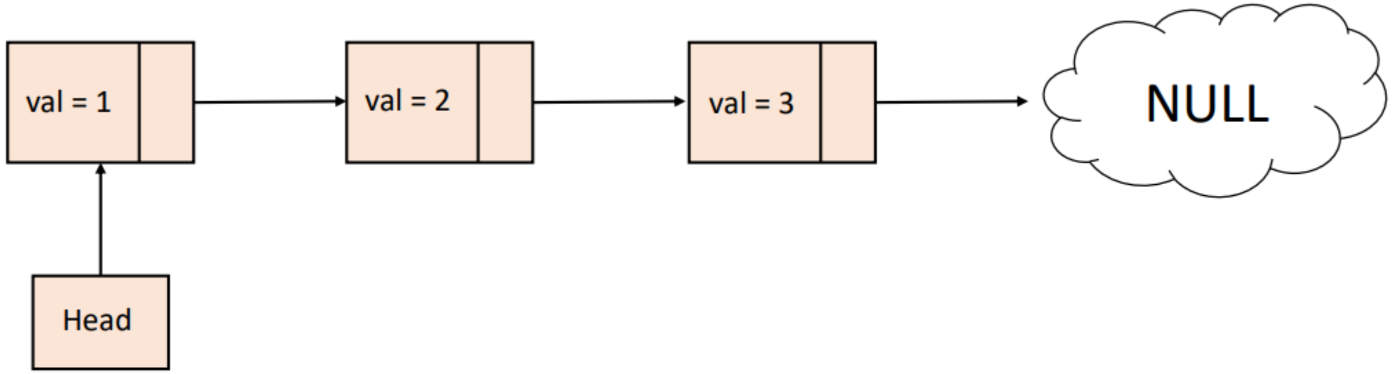
- Function Pointers
- Dynamic Array
- Begin Linked List

Linked List

```
struct node {
        void* val;
        struct node* next;
};
```



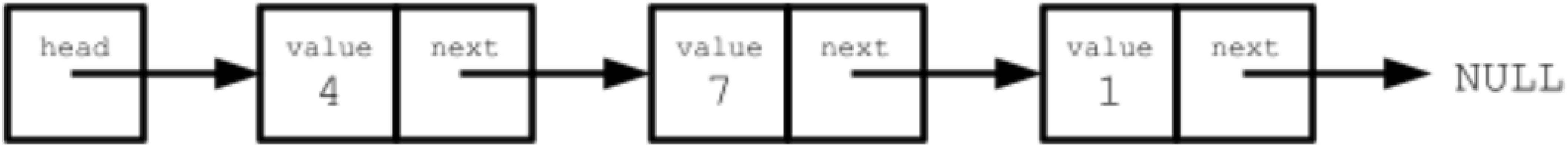
- Linear Data Structure
- Elements in a linked list are stored in nodes and chained together
  - Not in contiguous memory
  - Thus, no random access
- A linked list in which each node points only to the next link in the list is known as a singly-linked list.
  - E.g.:



Linked List

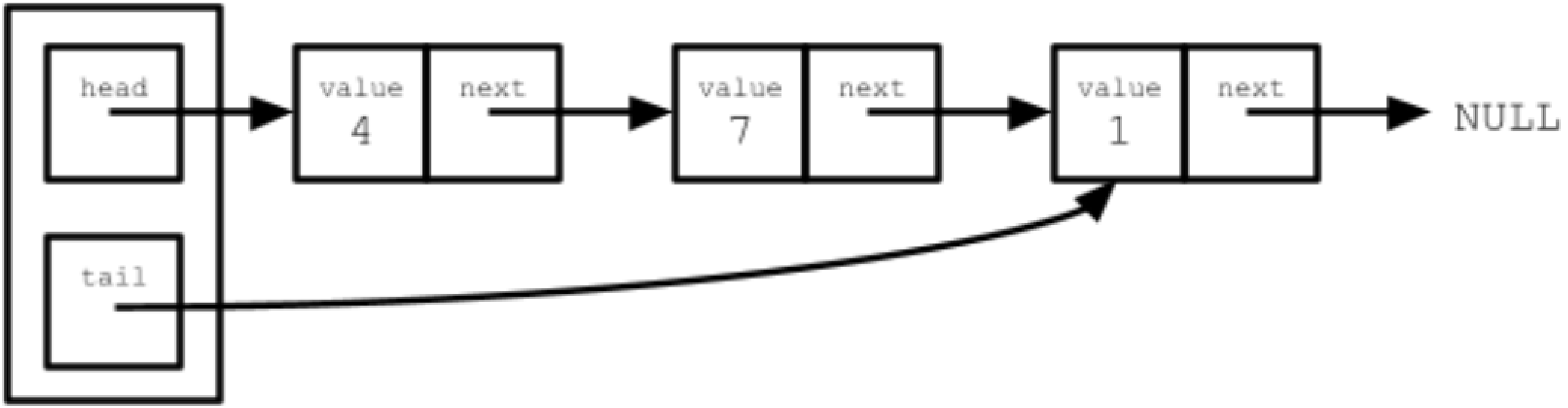
- Always contains as many nodes as it has stored values
  - Add an element → allocate a node, add it to the list
  - Remove an element → free the node from the list

- Many forms of linked list:
  - Keeps track only of the first element in the list, known as head



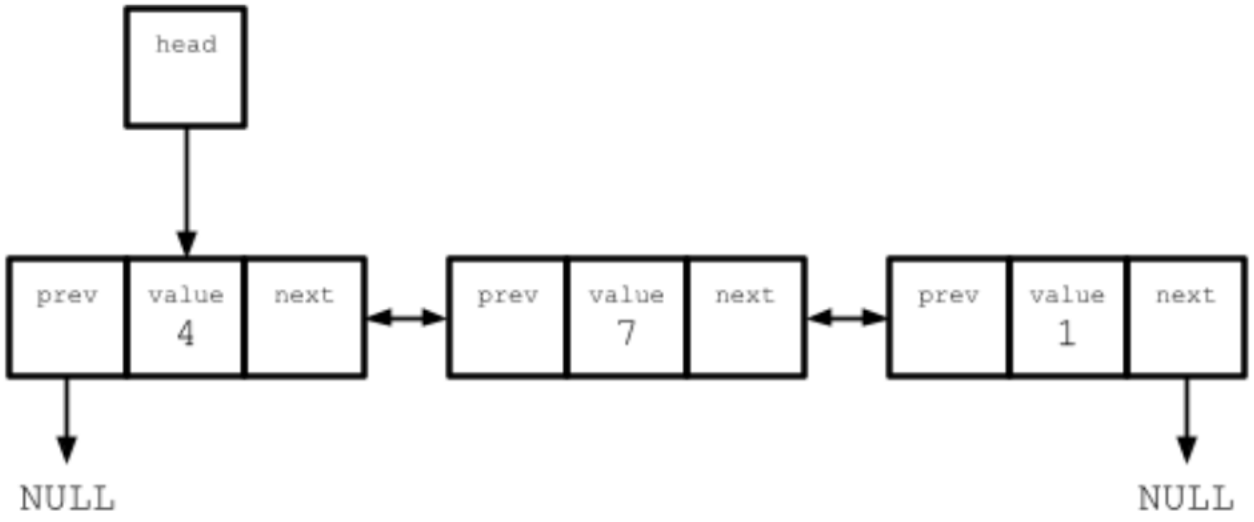
Linked List

- Many forms of linked list:
  - Keeps track only of the first element in the list, known as head
  - Keeps track of both the head of the list and the tail, or last element



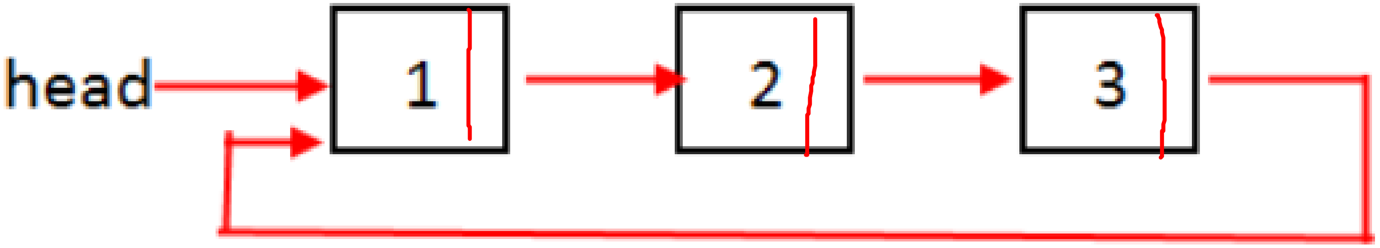
Linked List

- Many forms of linked list:
  - Keeps track only of the first element in the list, known as head
  - Keeps track of both the head of the list and the tail, or last element
  - Each node keeps track of both the *next* link and the *previous* link in the list, known as a doubly-linked list



Linked List

- Many forms of linked list:
  - Keeps track only of the first element in the list, known as head
  - Keeps track of both the head of the list and the tail, or last element
  - Each node keeps track of both the *next* link and the *previous* link in the list, known as a doubly-linked list
  - Last node points to the first node, known as circular-linked list



Linked List

- Many forms of linked list:
  - With sentinels, which are special nodes to designate the front/end of the list
    - E.g.: a doubly-linked list using both front and back sentinels