Programming 2

Linked List with Sir Beets

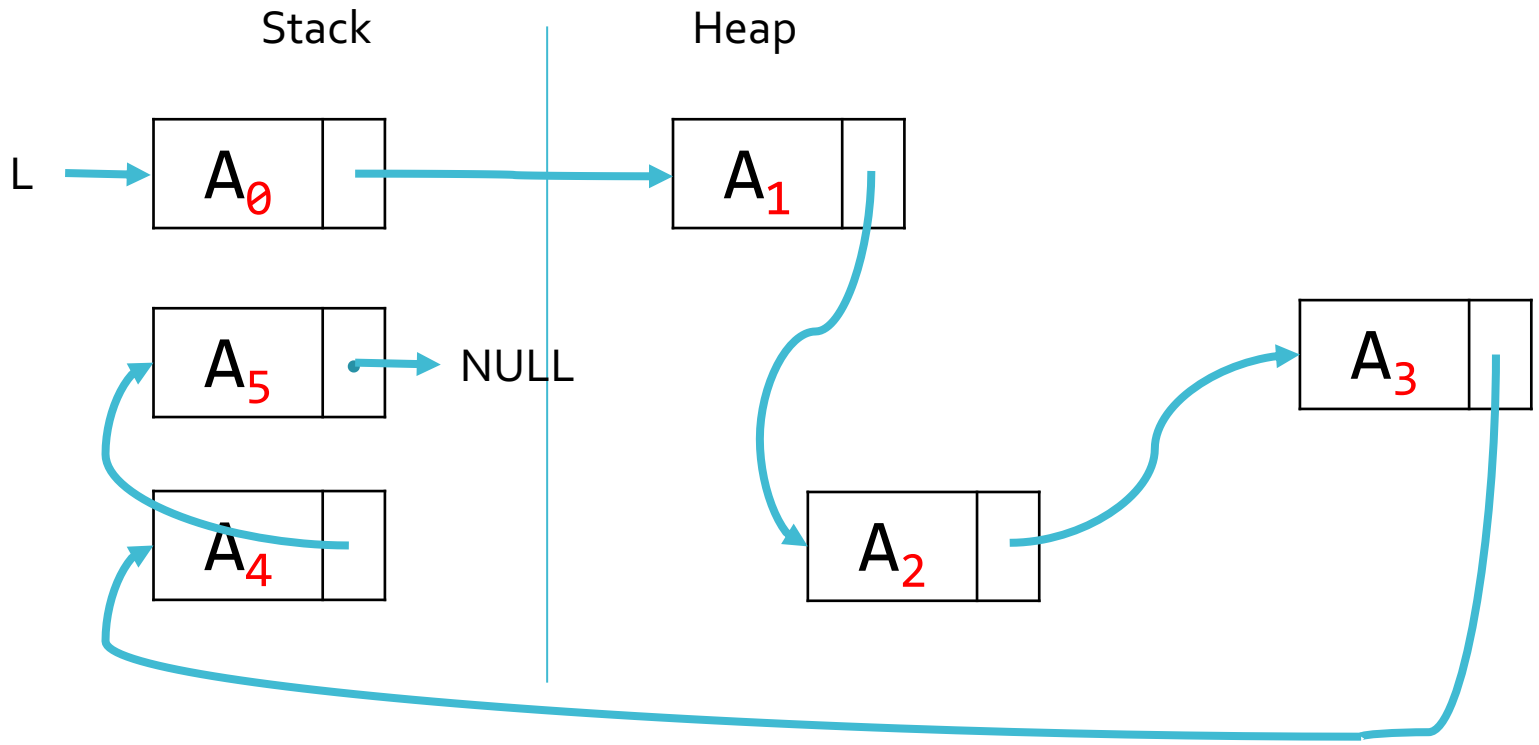
Topics

Linked List

- insert, delete, display, update (sorted and unsorted),
- malloc(), calloc(), realloc() and free()
- Array List vs. Linked List

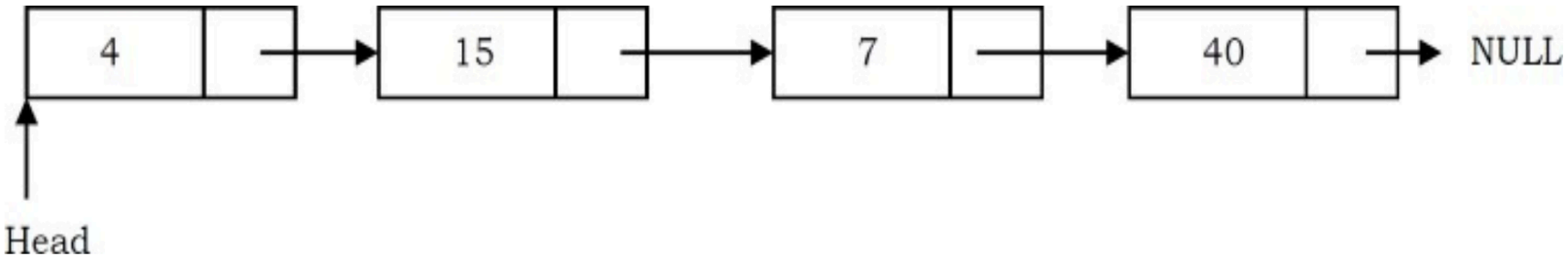
What is a Linked List?

The linked list consists of a **series of nodes (cell)**, which are not necessarily adjacent in memory. Each node contains the **element** and a **link** to a node containing its successor.



Properties of Linked List

- Successive elements are connected by pointers
- Last element points to NULL
- Can grow or shrink in size during execution of a program
- Does not necessarily waste memory space (but takes some extra memory for pointers)
- It allocates memory as list grows.



What problem was Linked Lists trying to solve?

- To avoid the linear cost of insertion and deletion
  - Ensure that the list is not stored contiguously
- Is Linked List an entirely a better approach and data structure for lists?

Structural Difference

Array List

```
#define SIZE 10
typedef struct{
        char data[SIZE];
        int count;
} List;
```

Linked List

```
typedef struct node {
        char data;
        struct node *link;
} celltype, * List;
```

ADT List Comparison

Array List

```
#define SIZE 10
typedef struct{
        char data[SIZE];
        int count;
} List;
```

FIXED SIZE

Linked List

```
typedef struct node {
        char data;
        struct node *link;
} celltype, * List;
```

DYNAMIC SIZE

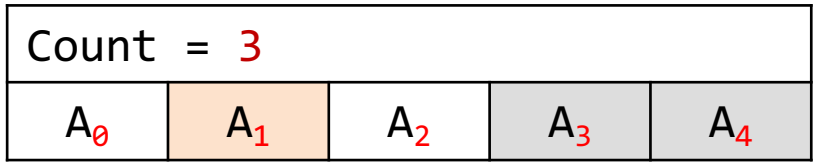
ADT List Comparison

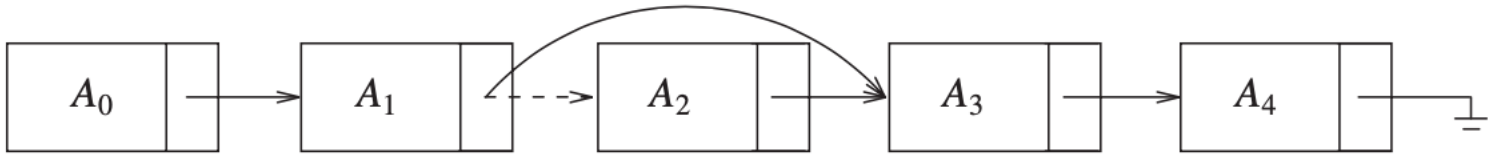
Array List

- Deleting a record creates an empty storage and needs special marking

Linked List

- Delete existing records while eliminating storage

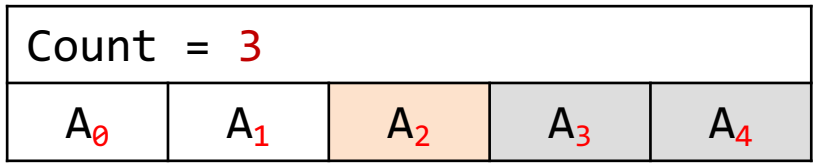




ADT List Comparison

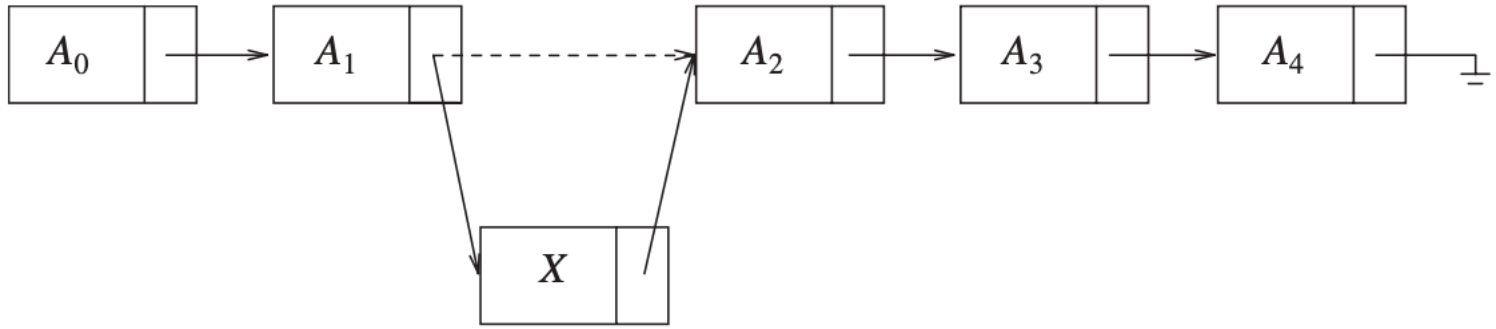
Array List

- Adding a record will have shifting issues
- Add/Delete is complicated, time-consuming, and inefficient



Linked List

- Add in proper order
- Provides a convenient method for maintaining a constantly changing list, without the need to continually reorder and restructure the entire list

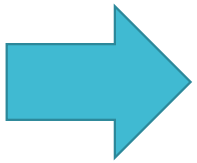


Activity
Data Structure

count 3

pos	A_i	**name**	**desc**	**qty**
1	A_0	Apple	Heals 10 HP	17
2	A_1	Clover	Nothing	3
3	A_2	O-Lance	AP +10	1

```
typedef struct node {
        char data;
        struct node *link;
} celltype, * List;
```



```
typedef struct node {
        struct {
          char name[10];
          char desc[15];
          int qty;
        } data;
        struct node *link;
} celltype, * List;
```

Fun Practice Codes

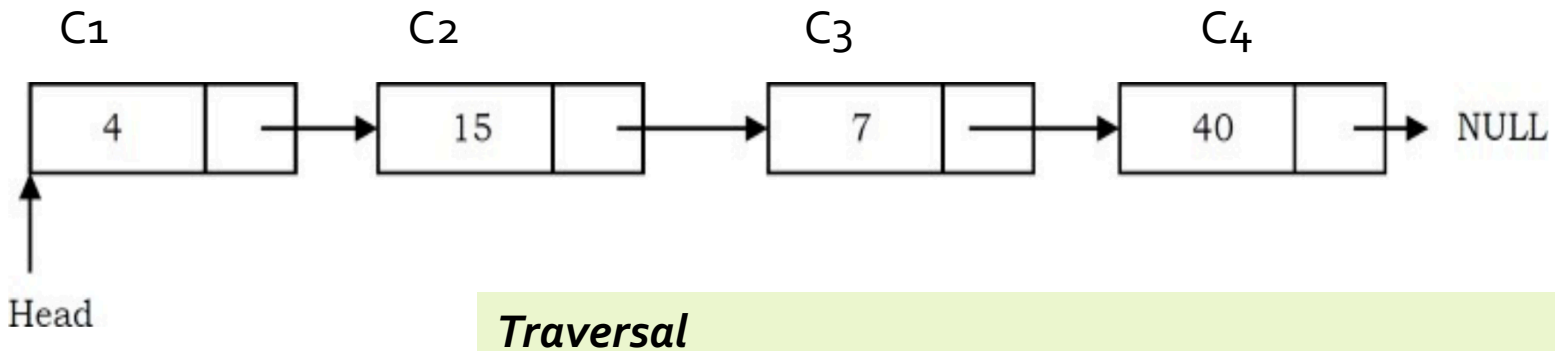
```
List L=NULL; /* empty list */
celltype c1 = {"Apple", "Heals 10HP", 17};
celltype c2 = {"Clover", "NothingP", 3};
celltype c3 = {"O-Lance", "AP +10", 1};

L = &c1;
c1.link = &c2;
c2.link = &c3;
c3.link = NULL;
```

Fun Practice Codes

```
List L=NULL; /* empty list */
celltype c1 = {"Apple", "Heals 10HP", 17};
celltype c3 = {"O-Lance", "AP +10", 1};

celltype *c2 = (List)malloc(1*sizeof(celltype));

L = &c1;
c1.link = c2;
c2->link = &c3;
c3.link = NULL;

/* assign values to members of the element at Position 2 */
```

Fun Practice Codes

```
void traverseList(List L){
  List trav;
  for( trav=L ; cond ; control) {
      traversal
  }
}
```



Traversal

printList(**List**) – prints all contents of a List
printOnly(**List**, **type**) – prints all contents of a List that are of a type
type – healing, weapon, other
count(**List**) – returns the number of nodes in a List

Insertion

Inserting in a Singly Linked List

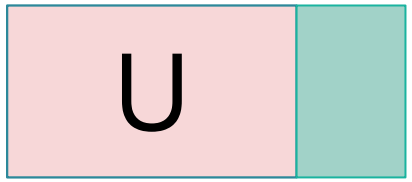
3 cases to consider:

1. Inserting a new node before the head (at the beginning)
2. Inserting a new node after the tail (at the end of the list)
3. Inserting a new node at the middle of the list (random location)

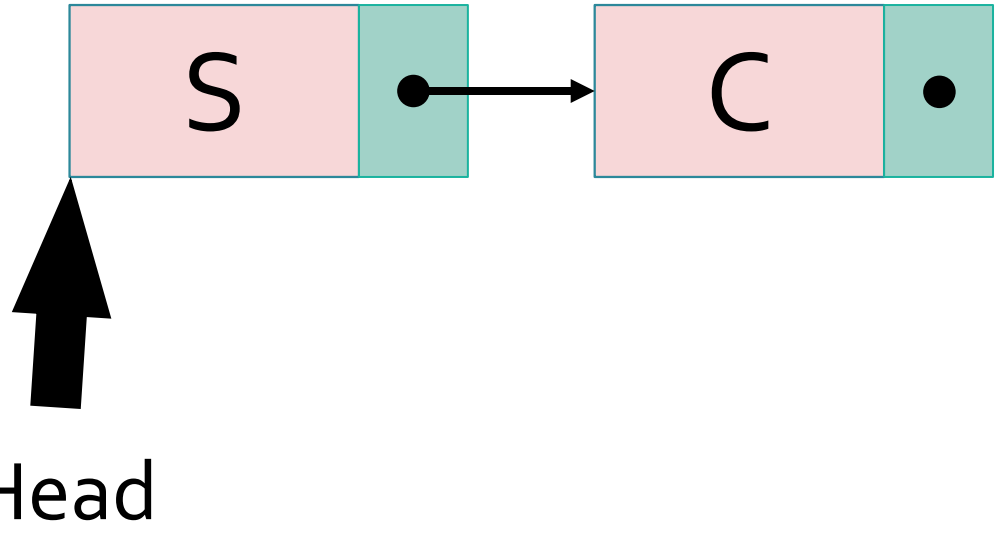
To insert an element in the linked list at some position *p*, assume that after inserting the element the position of this new node is *p*.

- Position 1 in array implementation starts at index zero [0]
- Position 1 in a linked list starts at the first element

At the beginning

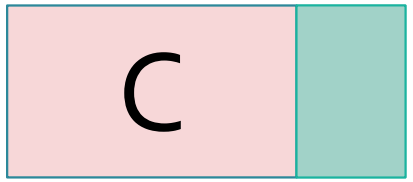


New Node

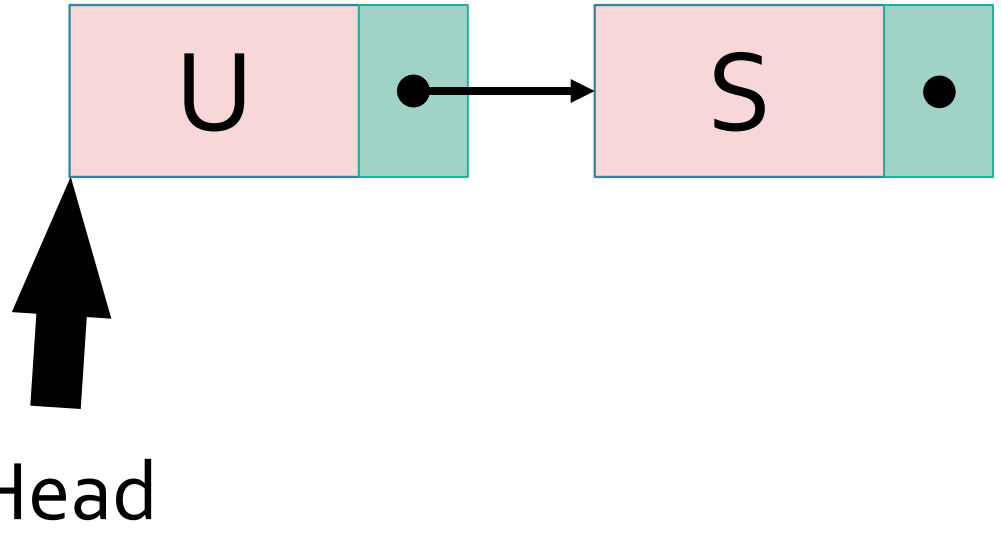


Head

At the end

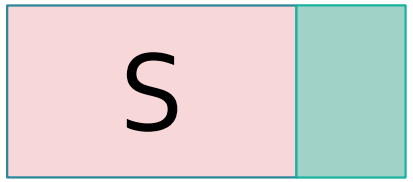


New Node

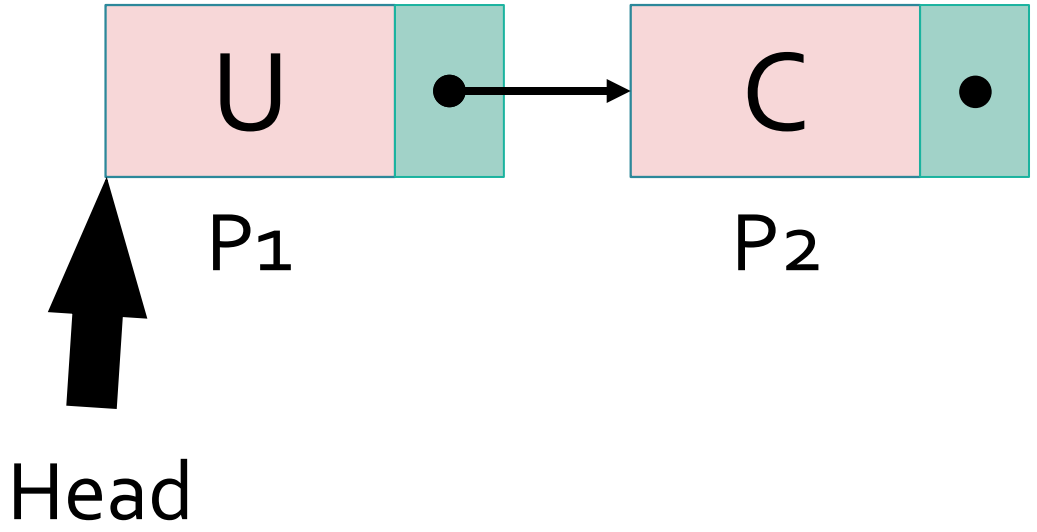


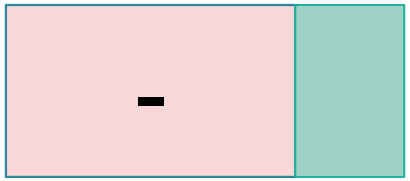
Head

Somewhere in the middle (Position Matters)

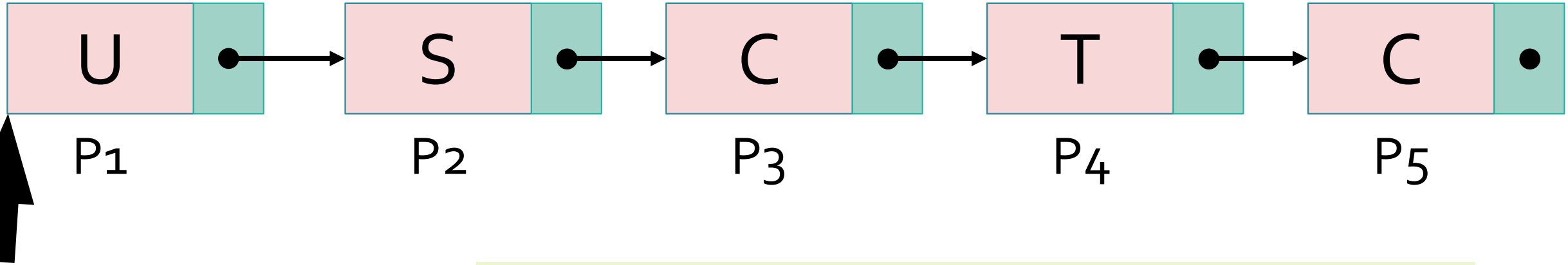


New Node





New Node



Insertion – adding an element in a list

insert(**List***, **x**) – add an element at the end of the List

insertStart(**List***, **x**) – add an element at the beginning of a List

insertPos(**List***, **x**, **pos**) – add an element at indicated position

Deletion

Deleting in a Singly Linked List

3 Cases like the insertion

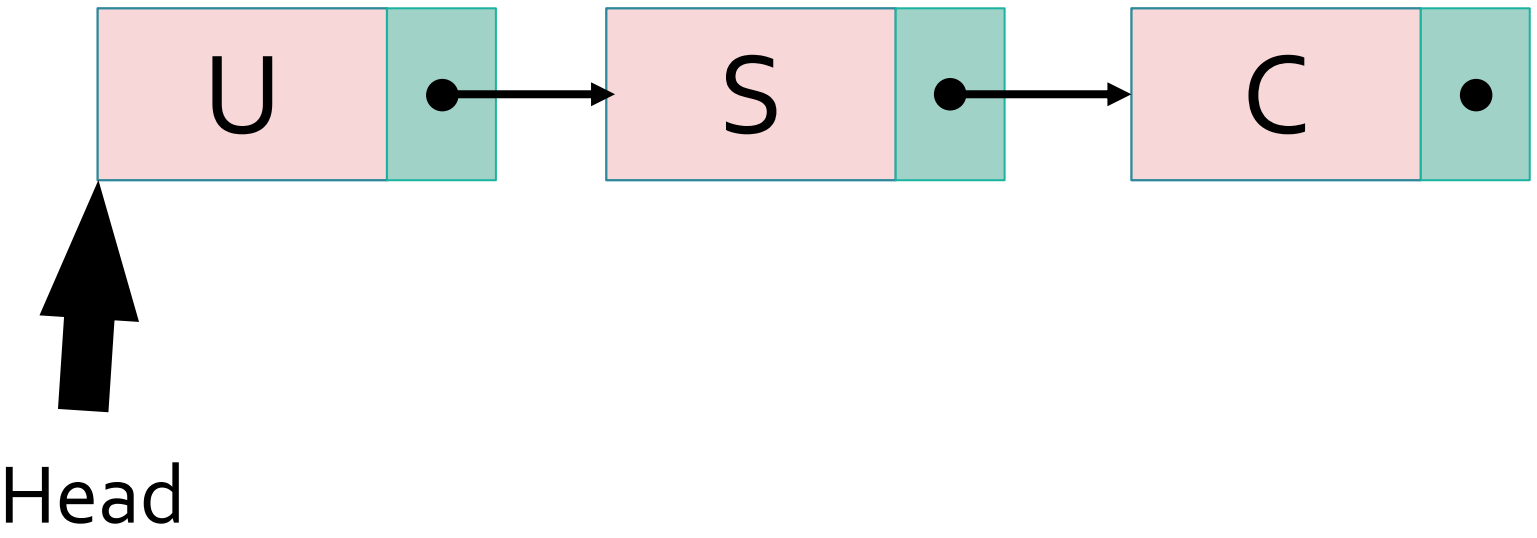
1. Deleting the first node
2. Deleting the last node
3. Deleting the intermediate (middle) node

void free (void *p)

- deallocates the space pointed to by **p**
- it does nothing if **p** is NULL
- **p** must be a pointer to space previously allocated by calloc(), malloc() or realloc()
- Releases a block of bytes previously reserved. The address of the first reserved location is passed as an argument to the function.

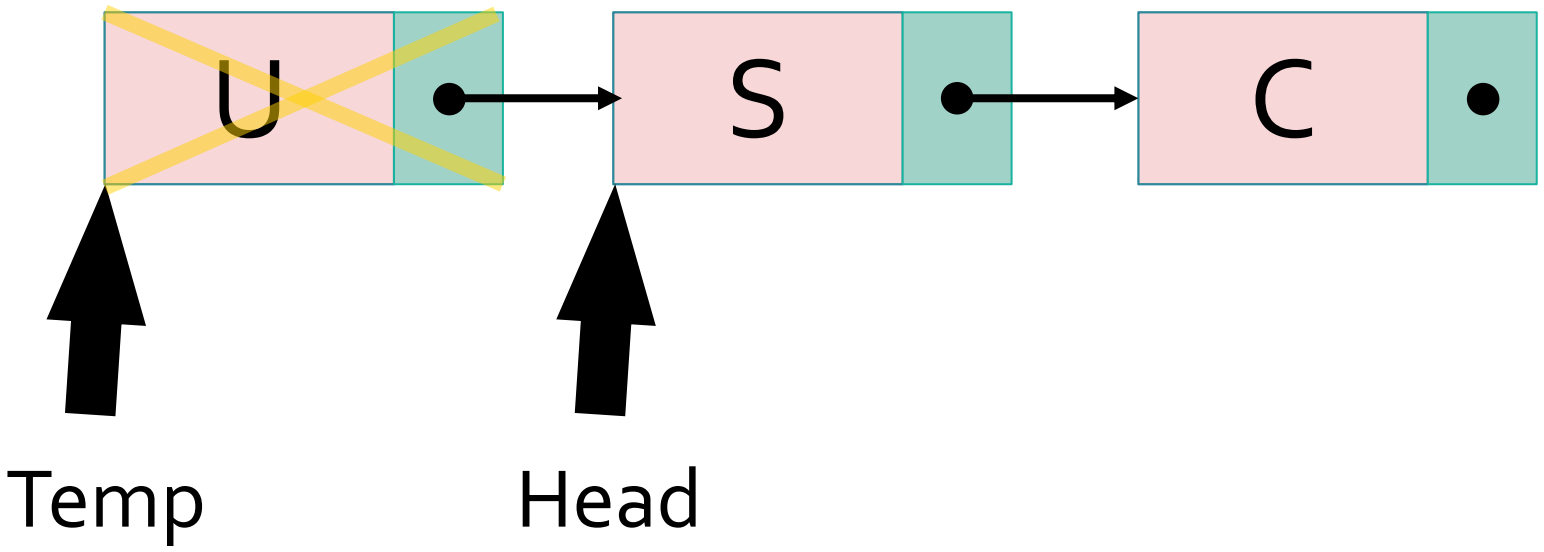
Deleting the First Node

Temp to Head



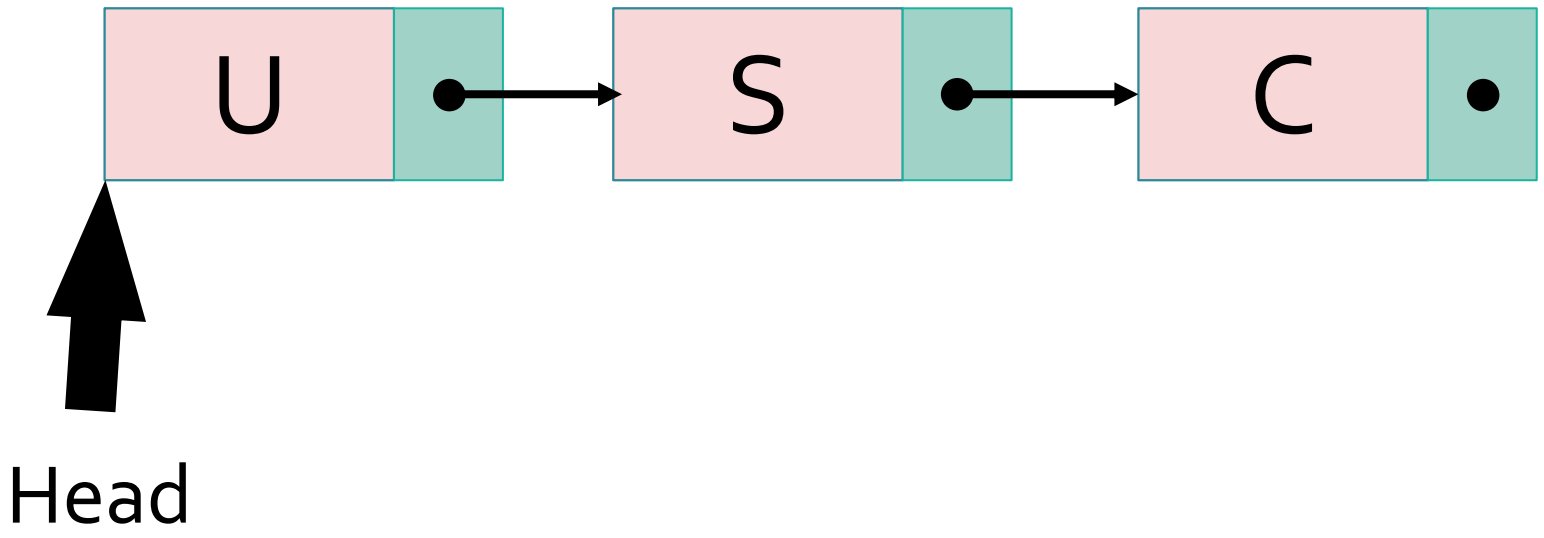
Deleting the First Node

Temp to Head



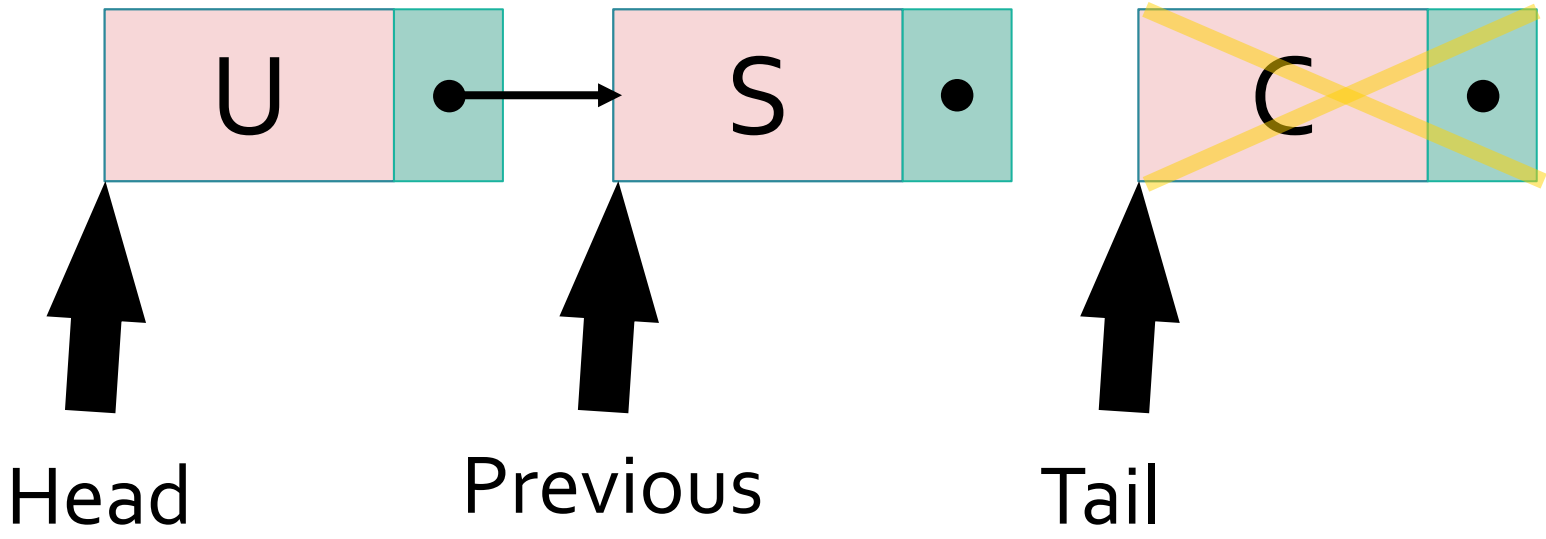
Deleting the Last Node

Traverse, and Make a Tail



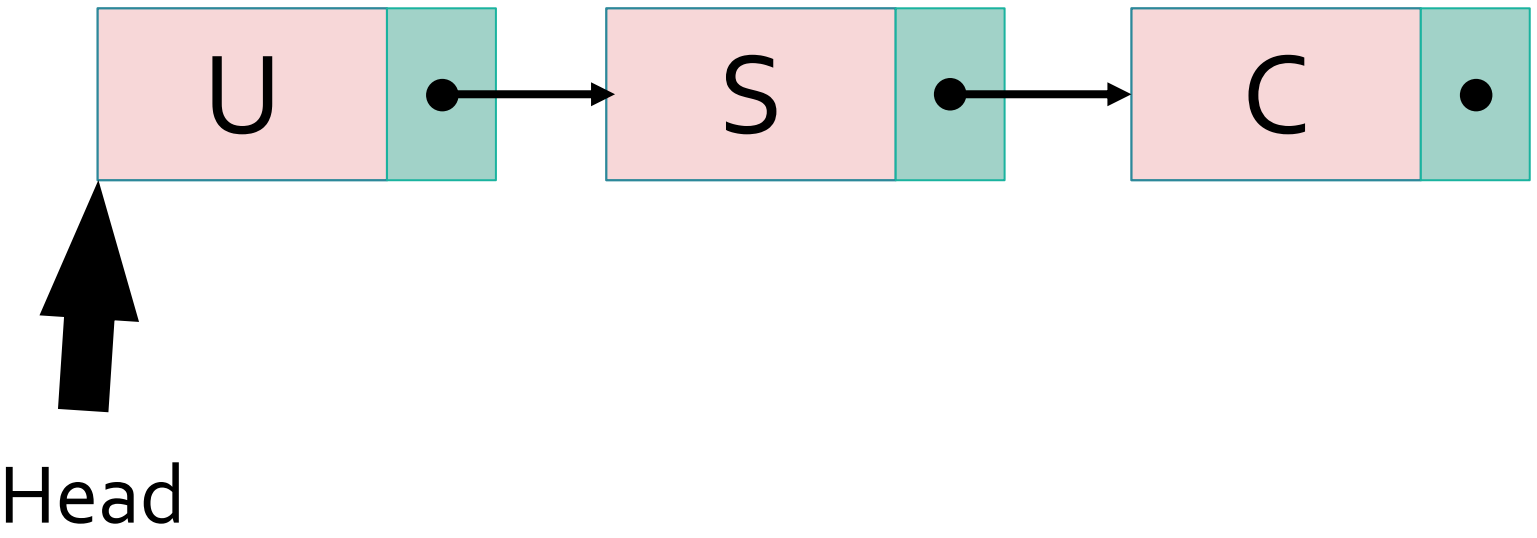
Deleting the Last Node

Traverse, and Make a Tail



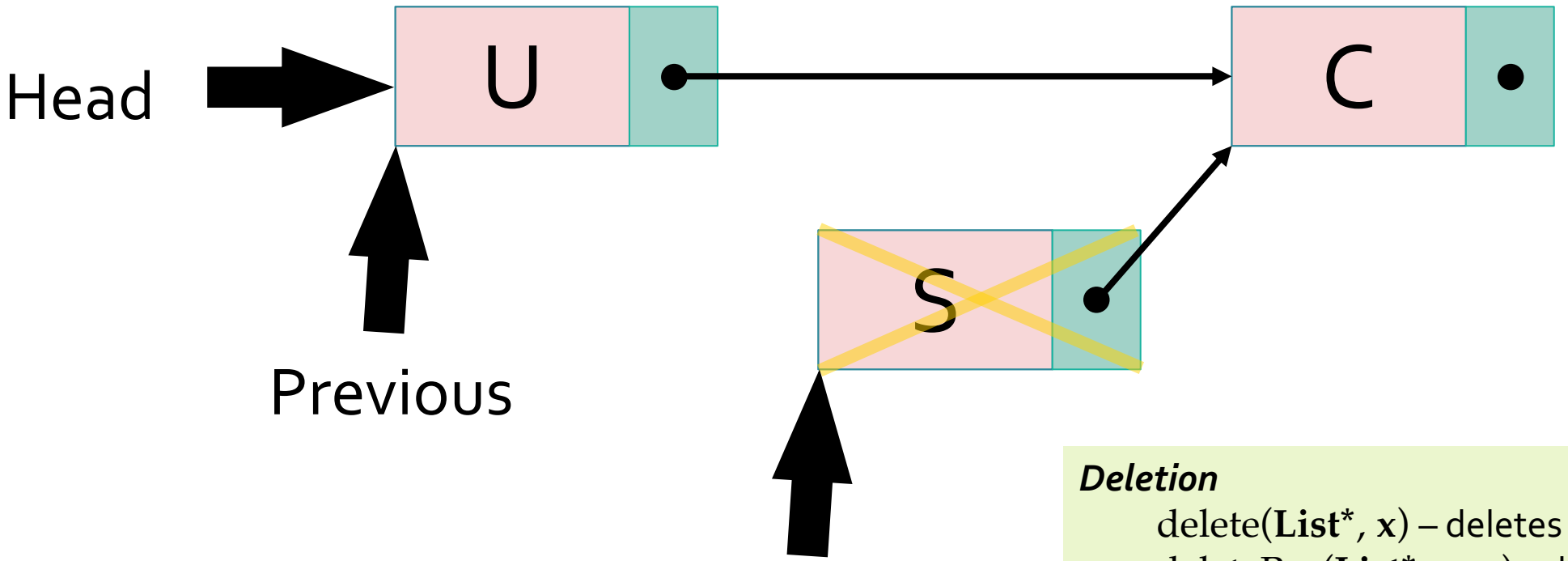
Deleting the Middle

Previous and ForDelete



Deleting the Middle

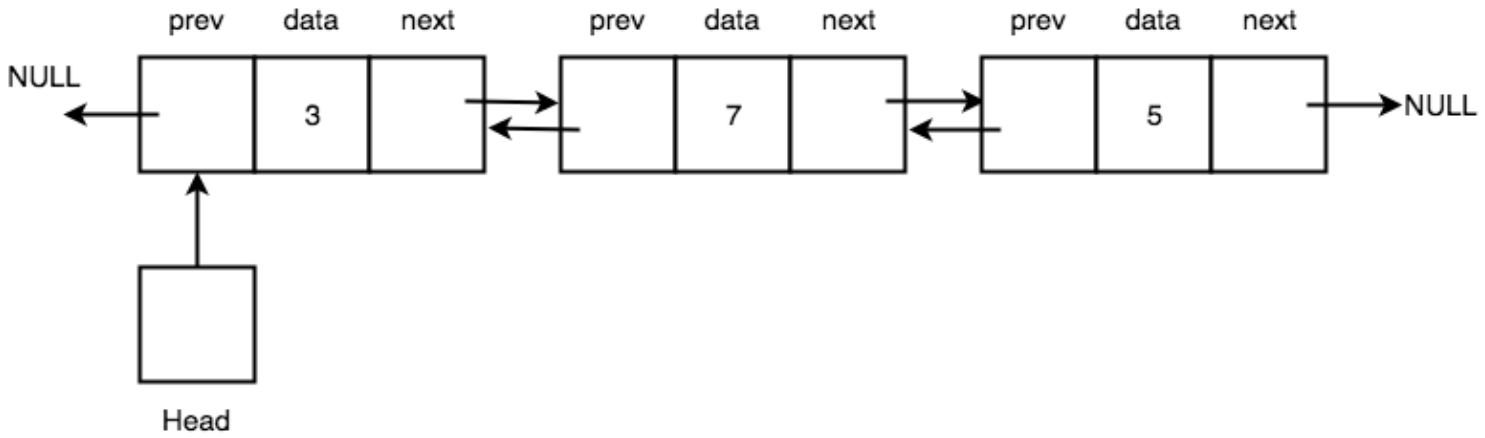
Previous and ForDelete

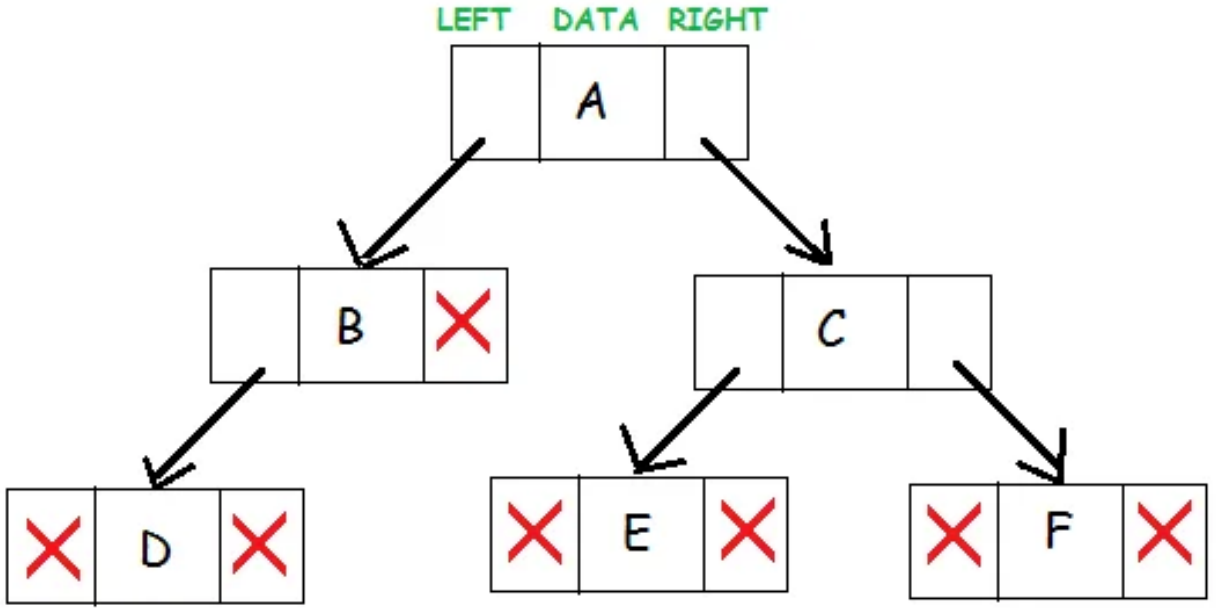


Deletion

delete(**List***, **x**) – deletes element x in the list
deletePos(**List***, **pos**) - deletes an element at the specified position
makeEmpty(**List***) – makes an empty list

Other Applications (DSA)





List Operations (Inventory)

Searching

- find(**List**, **x**) – returns the position of the first occurrence of an element **x** in the List

Sorting

- sort(**List***, **mode**) – sorts the List according to mode
  - mode – 1 is ascending, -1 is descending
  - item_code – a unique identifier for an item

Save your previous code (Array List)

Reconstruct your program into Linked Lists

Delete Entire Linked List

```
void deleteLinkedList(List *L){

        List temp, iterator;

        iterator = *L;
        while(iterator !=NULL) {
                temp = iterator->link;
                free(iterator);
                iterator = temp;
        }
        *L = NULL;

}
```

Remember to always check if the List is empty

Pre-Final Exam

***Schedule*:** July 19 6-8:30pm

***After July 13 Meeting*:**
Moved to July 20 6-8:30pm

- Conflicts with Eid al-Adha (Feast of Sacrifice)
- Waiting for official schedule and announcement from Malacañang Palace as stated in RA 9849 that it is a regular holiday.

Observations on Midterm Exam

- 20pts Test 1 Multiple Choice
- 20pts Test 2 Fill-in the Blank
- 10pts Test 3A
- 10pts Test 3B
- 20pts Test 3C

80pts Total, 48pts to Pass

Test 3B

Inner loop and condition

- Left to right is not considered (0 to count)
- Strictly bottom to top (right to left) traversal (count to 0)

Considered swapping of point elements

Excludes

- Incorrect access in swapping elements
- Incorrect elements swapped (member latitude or longitude)
- Incorrect type for temp (gCoor, float, pointList)