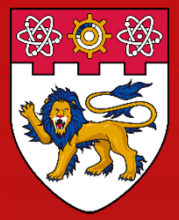


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CE1107/CZ1107: DATA STRUCTURES AND ALGORITHMS

Linked Lists

College of Engineering
School of Computer Science and Engineering

TODAY

- Data structures as nodes + links
- Storing lists in arrays
- Linked lists
- Implementing a node
- Implementing a linked list
- Common mistakes

LEARNING OBJECTIVES

After this lesson, you should be able to:

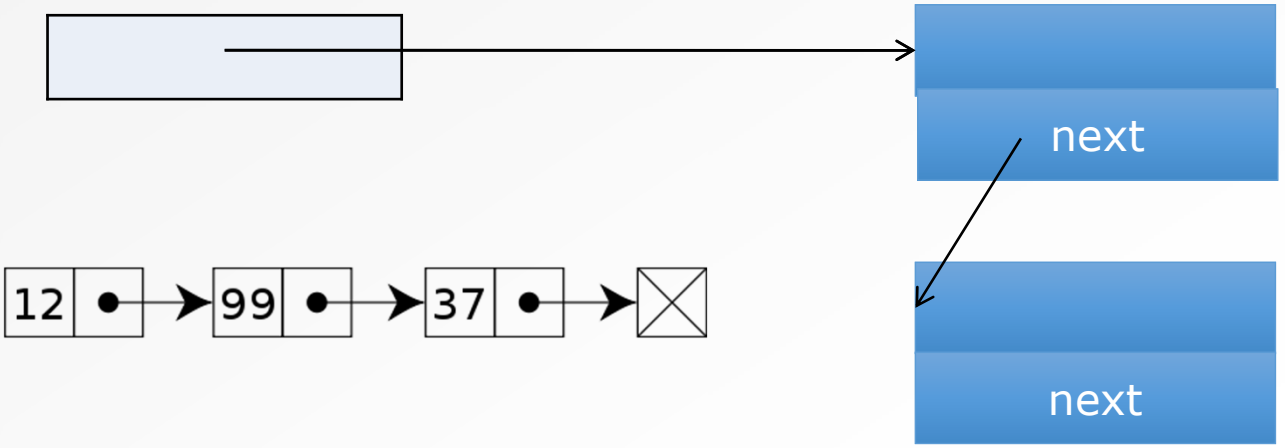
- Create a linked list with dynamic nodes using malloc()
- Design your own node structure

TODAY

- **Data structures as nodes + links**
- Storing lists in arrays
- Linked lists
- Implementing a node
- Implementing a linked list
- Common mistakes

MALLOC() BASICS: STRUCT TO STRUCT

- Recall what we did with malloc()
  - Dynamically allocated structs
  - First struct points to the second struct, second points to the third…
  - If the first struct is deleted, the second struct is "lost"
- This is the core idea behind a linked list data structure



NODES + LINKS

- Each of the structs we created is a distinct node
- Chunk containing two components:
  - Data field(s)
  - Links to other nodes
- Data structure = nodes + links
- Different arrangements of links between nodes
- How is this useful?

LIST STORAGE

- Suppose we are trying to store a list of items
  - List of names
  - List of numbers
  - etc.
- Sequential data
  - Each item has a place in the sequence
  - Each item comes after another item
- You already know one way to store this list
  - Arrays

TODAY

- Data structures as nodes + links
- **Storing lists in arrays**
- Linked lists
- Implementing a node
- Implementing a linked list
- Common mistakes

STORING A LIST OF NUMBERS IN AN ARRAY

Static Array Version

- Allocate some fixed size array
- Wasted space

```
1   int numOfNumbers;
2   Int numArray[1000];
3
4   scanf("%d", &numOfNumbers);
5
6   for (i=0; i<numOfNumbers; i++){
7           scanf("%d", &numArray[i]);
8
9   }
```

STORING A LIST OF NUMBERS IN AN ARRAY

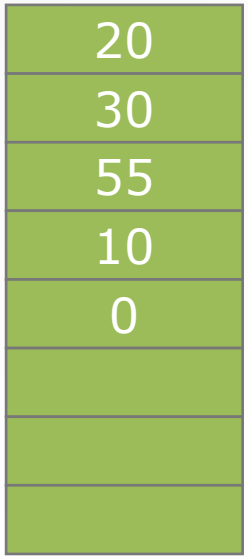
Malloc()ed Array Version

- Allocate exactly the right sized array
- Looks like a good solution
  - No wasted space
  - But what happens when we want to change the list?

```
1   #include <stdlib.h>
2   int main(){
3           int n;
4           int *int_arr;
5           printf("How many integers do you have?");
6           scanf("%d", &n);
7           int_arr = malloc(n * sizeof(int));
8           if (int_arr == NULL) printf("Uh oh.\n");
9
10          // Loop over array and store integers entered
11  }
12
```

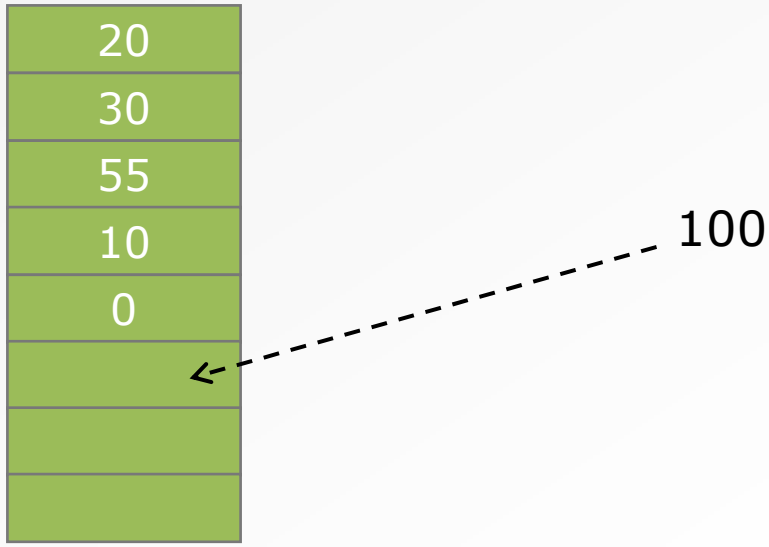
MODIFYING LISTS STORED IN ARRAYS

- Suppose I have an existing list of numbers in an int[]
- Now I want to do the following
  - Add a number
    - At the front
    - At the back
    - In the middle
  - Remove a number
    - From the front
    - From the back
    - From the middle
  - Move a number to a different position
- Is it doable? Easy to do?



ADD AN ITEM TO THE BACK OF AN ARRAY

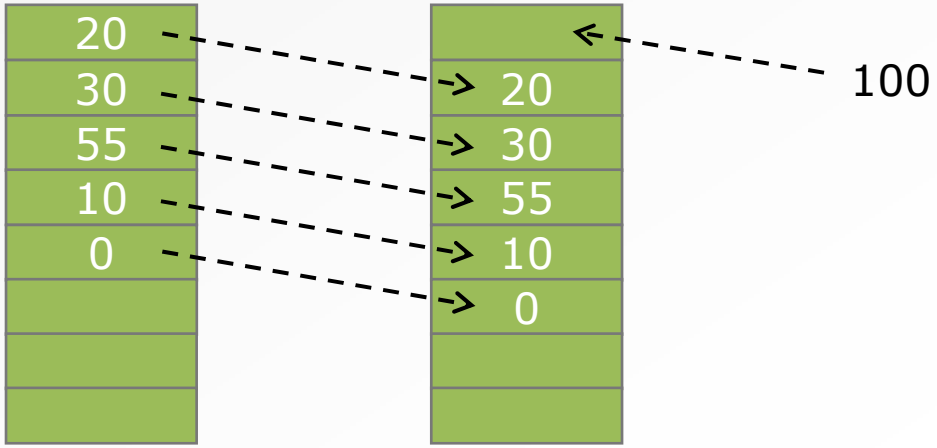
1. Assuming array has at least one unused element at the end, insert new item into next empty array element



ADD AN ITEM TO THE START OF AN ARRAY

1. Create an empty array element at the front by shifting all existing elements down by one space, assuming array has at least one unused element at the end
2. Insert new item into empty array element

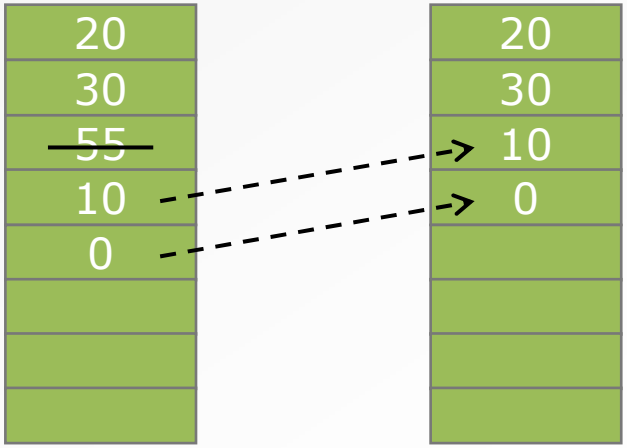
- What happens when you have an array of 1000000 elements?



REMOVE AN ITEM FROM THE MIDDLE OF AN ARRAY

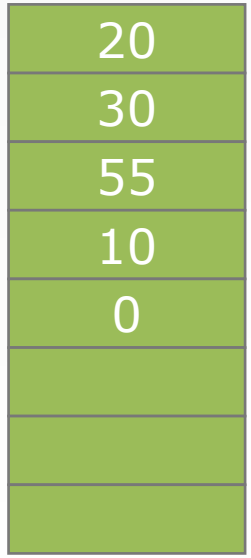
1. Remove item from array
2. ”Remove” empty space by shifting elements up by one space to form a single contiguous block

- What happens when you have an array of 1000000 elements?



STORING LISTS OF ITEMS IN ARRAYS

- Items have to be stored in contiguous block
- No gaps in between items
- Easy to:
  - Add at the back
  - Remove from the back
- Not so easy to:
  - Add at the front/middle
  - Remove from the front/middle
  - Add items when all array elements have been used to store a value



STORING LISTS OF ITEMS IN ARRAYS

- Each item's position in the sequence comes from the array element where it is stored:
  - If item #2 is stored in array[1], item #3 must be stored in array[2]
  - If item #2 is stored in array[11], item #3 must be stored in array[12]
- As a result, modifying lists of items can be tricky
- Need to think of a different way to store lists

LIST DATA STRUCTURE

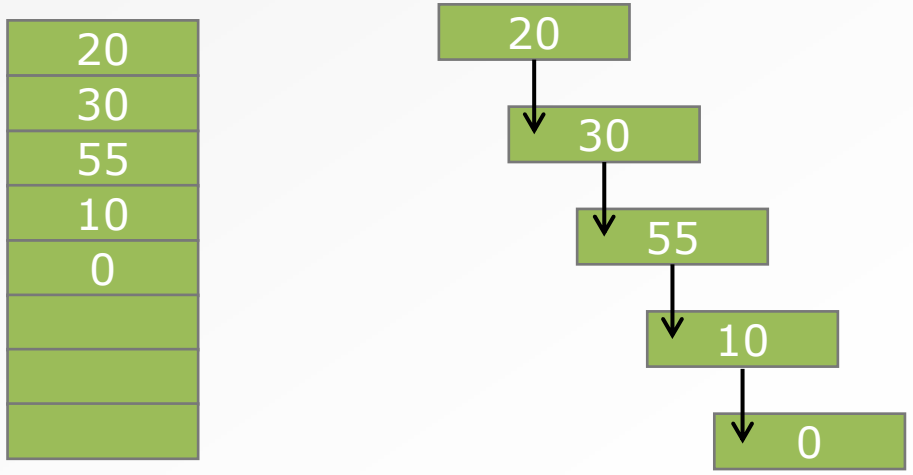
- What we want
  - Easy to add a new item anywhere
  - Easy to remove an item anywhere
  - Easy to move an item around in the list
- Arrays can’t support these requirements
- Back to the idea of nodes + links
  - Each item is stored in separate node
  - Connect nodes together with links

TODAY

- Data structures as nodes + links
- Storing lists in arrays
- **Linked lists**
- Implementing a node
- Implementing a linked list
- Common mistakes

LINKED LIST DATA STRUCTURE

- Each node stores one item
- Each node points to the next node
- Create each node dynamically (using malloc())
- Position in the sequence depends on arrangement of links

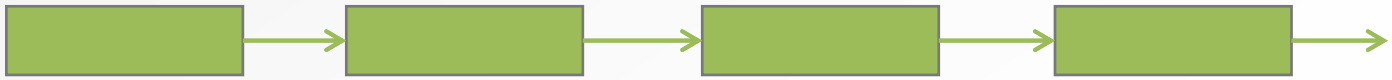


LINKED LISTS

Photo extracted from Wikimedia Commons by Buchoamerica under CC BY-SA 3.0.

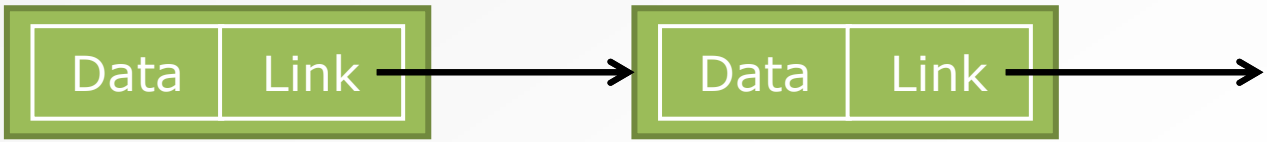
BASIC LINKED LIST

- Different types of data can be stored in a node
- Singly-linked list
  - Each node is connected to at most one other node
  - Each node keeps track of the next node
- Let's declare the node structure first



BASIC LINKED LIST NODES

- Each node is a ListNode structure
- Basic nodes have two components:
  - Data stored in that node
  - Link to the next node in the sequence



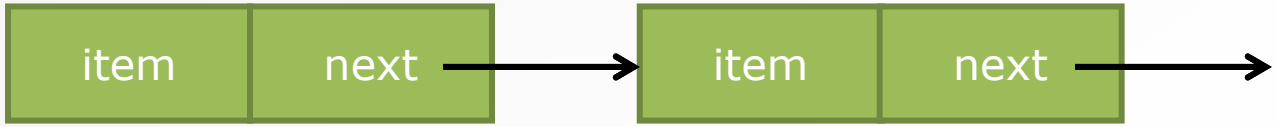
TODAY

- Data structures as nodes + links
- Storing lists in arrays
- Linked lists
- **Implementing a node**
- Implementing a linked list
- Common mistakes

BASIC LINKED LIST NODES

- Basic node structure
- For now, assume that a node stores an integer

```
typedef struct _listnode{
    int item;
    struct _listnode * next;
} ListNode;
```



BASIC LINKED LIST NODES

- Let's statically create a node
  - Declared at compile time

```
ListNode static_node;
static_node.item = 50;
static_node.next = null;
```

BASIC LINKED LIST NODES

- Now, let's dynamically create a new node
  - Use malloc to allocate memory while your program is running

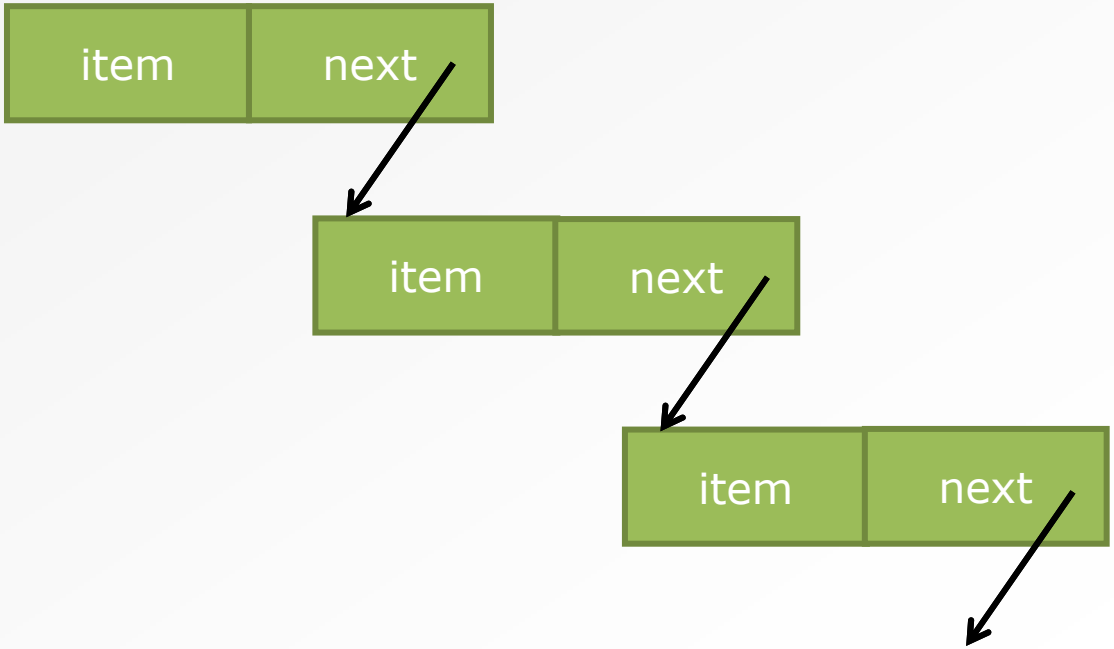
```
ListNode*dy_node= malloc(sizeof(ListNode));
dy_node->item = 50;
dy_node->next = NULL;
```

TODAY

- Data structures as nodes + links
- Storing lists in arrays
- Linked lists
- Implementing a node
- **Implementing a linked list**
- Common mistakes

LINKED LIST OF NODES

- We have created the ListNode structure to represent a node of data
- A linked list will have some/many nodes



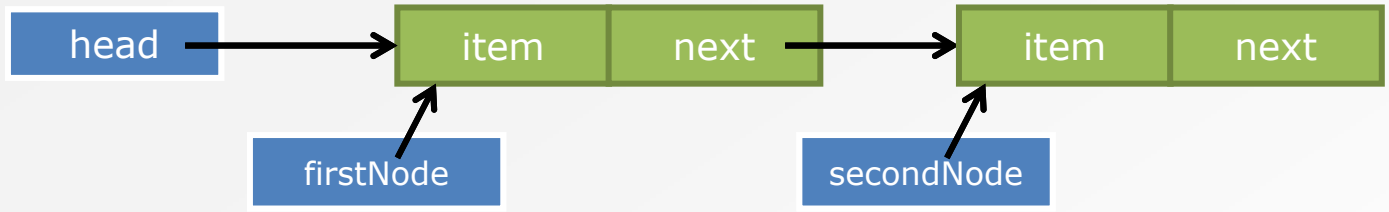
LINKED LIST OF NODES

- Each node tracks the next node that comes after it
  - Last node tracked by the second-last node
  - #4 node tracked by #3 node
  - Whole sequence of nodes accessible by starting from the first node in the sequence
  - But who tracks the first node?

LINKED LIST OF NODES

- Without the address of the first node, everything else is inaccessible
- Add a pointer variable ***head*** to save the address of the first ListNode struct
- What is the data type for head?

SINGLY-LINKED LIST OF INTEGERS (TWO NODES)

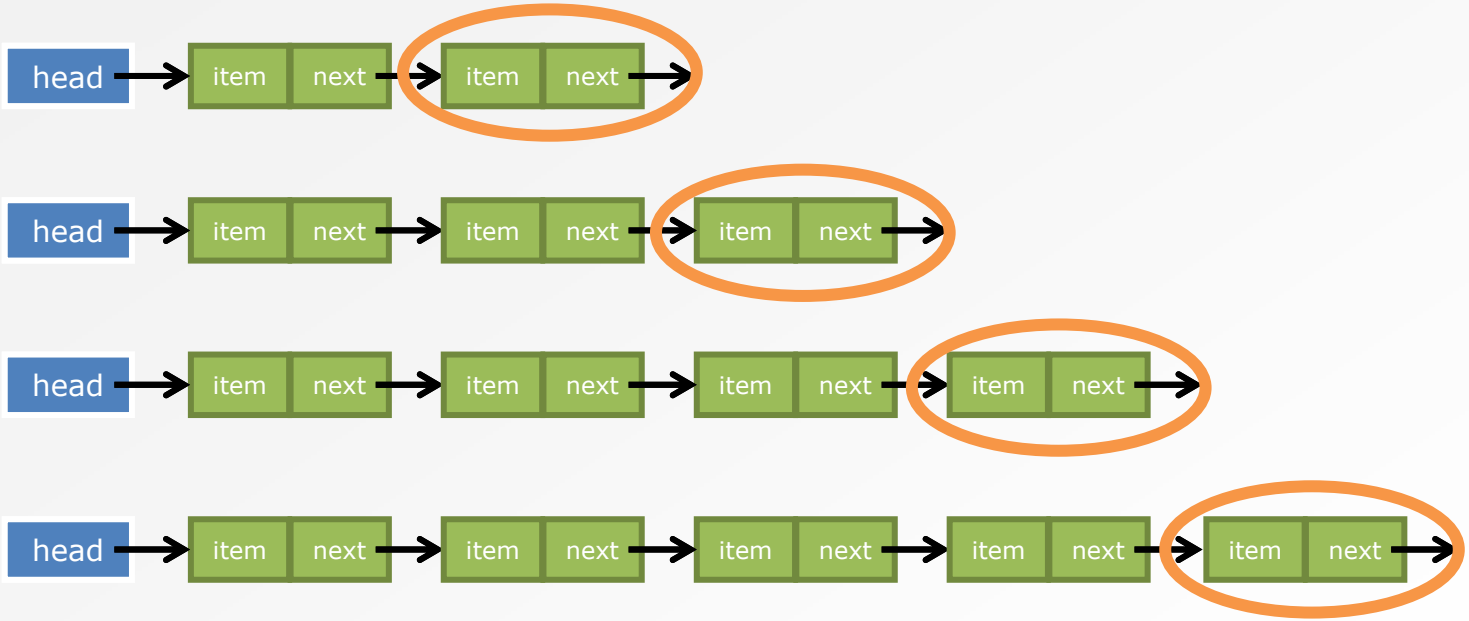



```
1    typedef struct node{
2            int item;
3            struct node *next;
4    } ListNode;
5
6    int main(){
7            ListNode *head, *firstNode, *secondNode;
8
9            firstNode = malloc(sizeof(ListNode));
10           secondNode = malloc(sizeof(ListNode));
11
12           head = firstNode;
13           firstNode->next = secondNode;
14           secondNode->next = NULL;
15   }
```

BACK TO LAB QUESTION: STORE A LIST OF NUMBERS

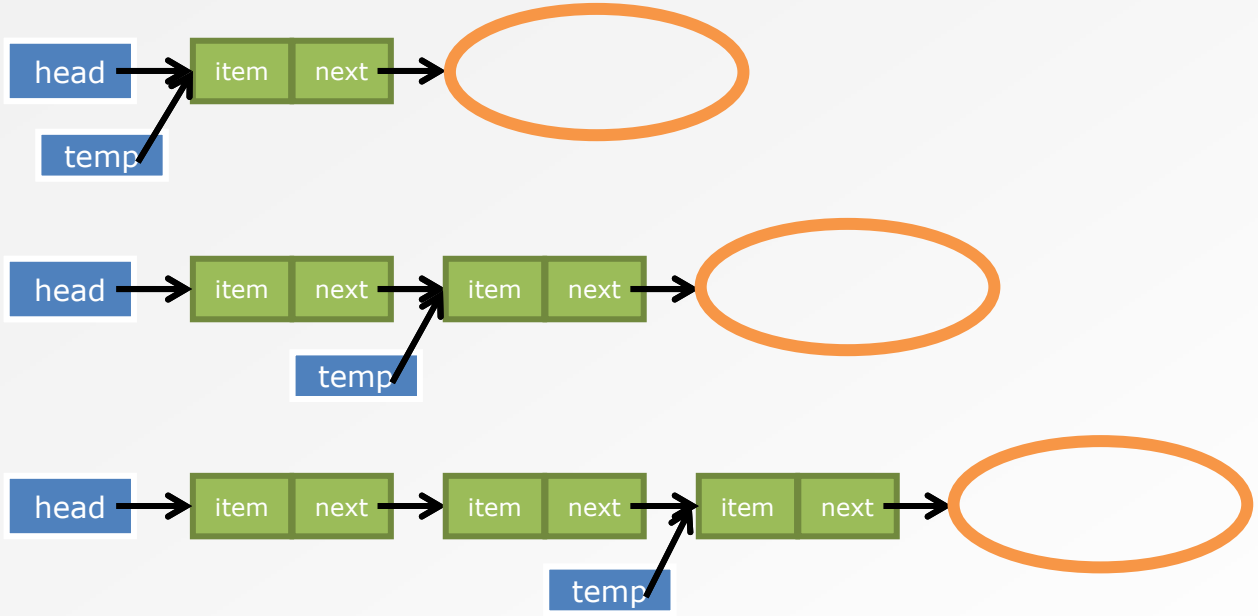
- Previously, we used malloc() to create int array to store all numbers after numOfNumbers was known
- This time, use malloc() to create a new ListNode for each number
  - Get input until input == –1
  - For each input number, create a new node to store the value
  - Arrange all the ListNodes as a linked list

BACK TO LAB QUESTION: STORE A LIST OF NUMBERS



- Address of each new ListNode is saved in next pointer of previous node
- Need a way to keep track of the last ListNode at any time
  - Use another pointer variable

BACK TO LAB QUESTION: STORE A LIST OF NUMBERS



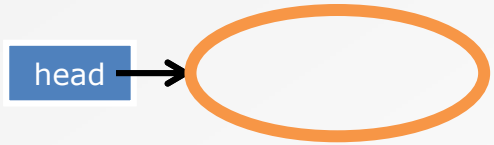
- *temp* pointer stores address of the last ListNode at any time
- Create a new ListNode

```
temp->next = malloc(sizeof(ListNode));
```

BACK TO LAB QUESTION: STORE A LIST OF NUMBERS

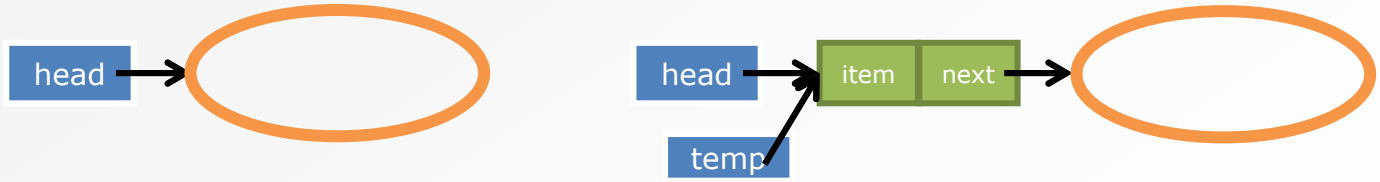
- Watch out for special case
  - First node in the linked list
  - *head* == NULL
  - Need to update the *head* pointer

```
head = malloc(sizeof(ListNode));
```



BACK TO LAB QUESTION: STORE A LIST OF NUMBERS

- After the first ListNode has been created
  - *head* pointer points to first ListNode
  - Can now use *temp* pointer to keep track of last node
  - In this case, *temp* also points to the first ListNode



SINGLY-LINKED LIST OF INTEGERS

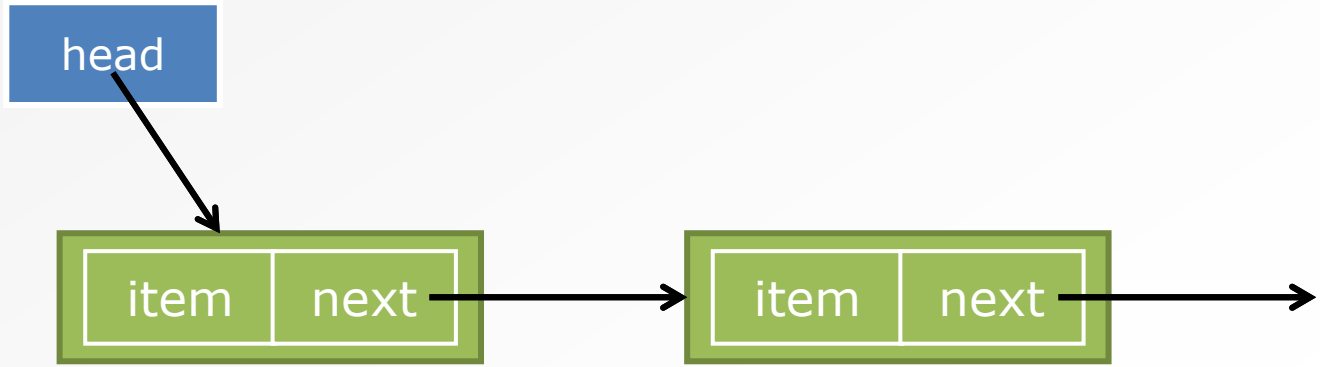
```
1    typedef struct node{
2        int item;  struct node *next;
3    } ListNode;
4
5    int main(){
6        ListNode *head = NULL, *temp;
7        int i = 0;
8
9        scanf("%d", &i);
10       while (i != -1){
11           if (head == NULL){
12               head = malloc(sizeof(ListNode));
13               temp = head;
14           }
15           else{
16               temp->next = malloc(sizeof(ListNode));
17               temp = temp->next;
18           }
19           temp->item = i;
20           scanf("%d", &i);
21       }
22       temp->next = null;
23   }
```

TODAY

- Data structures as nodes + links
- Storing lists in arrays
- Linked lists
- Implementing a node
- Implementing a linked list
- **Common mistakes**

COMMON MISTAKES

- **Very important!**
  - *head* is a node pointer
  - Points to the first node
  - *head* is not the "first node"
  - *head* is not the "head node"



TODAY

- Data structures as nodes + links
- Storing lists in arrays
- Linked lists
- Implementing a node
- Implementing a linked list
- Common mistakes

NEXT LECTURE

- Write functions for commonly used operations
  - Add a node to a linked list
  - Remove a node from a linked list
  - Etc.
- Use a linked list and the functions above in an application