NANYANG TECHNOLOGICAL UNIVERSITY

CE1007/CZ1007 DATA STRUCTURES

Lecture 02: **Linked List**

Dr. Owen Noel Newton Fernando

College of Engineering

School of Computer Science and Engineering

OUTLINE

- Linear Data Structure
- Data structures as nodes + links
- Storing Lists in Arrays
- Storing Lists in Links: Linked List
- Implementing a node
- Implementing Linked List
- Common Mistakes

YOU SHOULD BE ABLE TO...

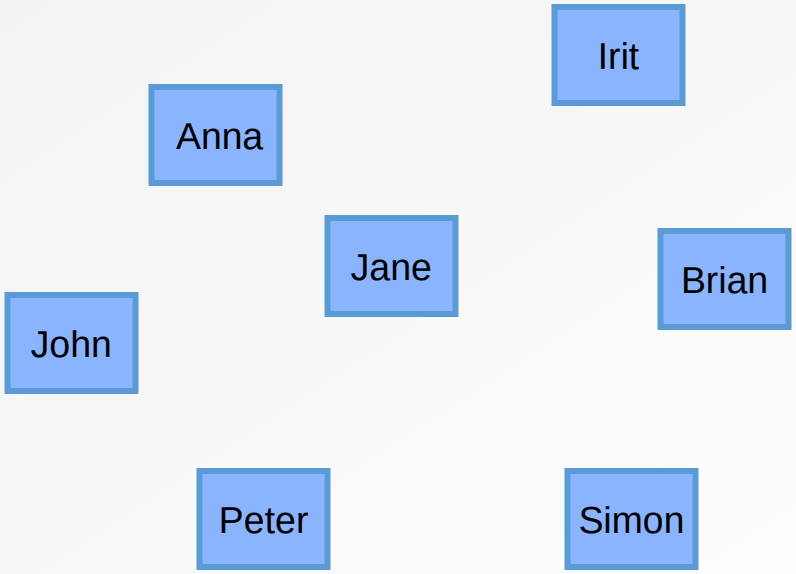
- Create a linked list with dynamic nodes using malloc()
- Design your own Node structure

OUTLINE

- **Linear Data Structure**
- Data structures as nodes + links
- Storing Lists in Arrays
- Storing Lists in Links: Linked List
- Implementing a node
- Implementing Linked List
- Common Mistakes

LINEAR DATA STRUCTURE

- Suppose you have a set of names



- How do you manage them?

LINEAR DATA STRUCTURE

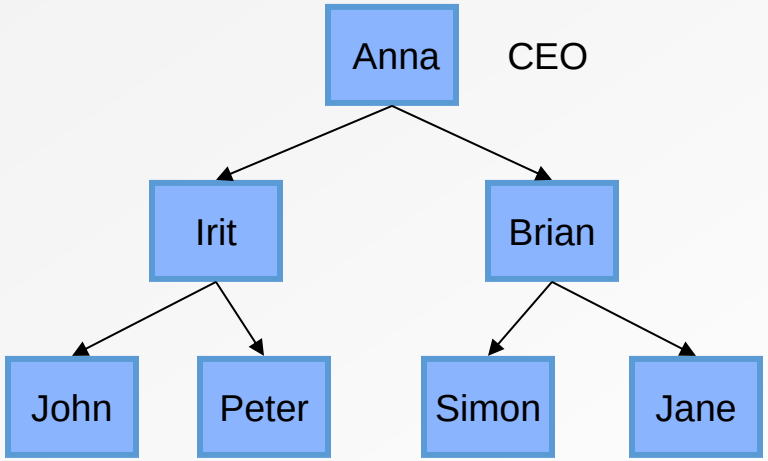
- Suppose you have a set of names

List

- If they are in the waiting list

NON-LINEAR DATA STRUCTURE

- Suppose you have a set of names

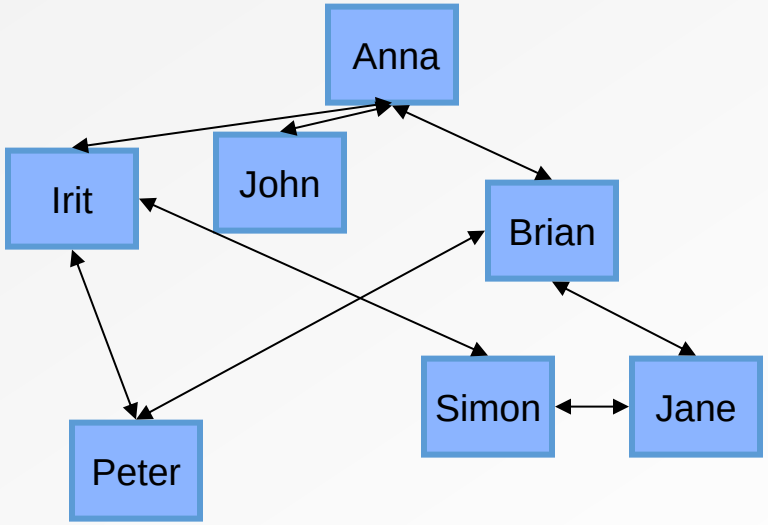


Tree

- Company organization

NON-LINEAR DATA STRUCTURE

- Suppose you have a set of names



Graph

- Friendship network

THE SIMPLEST DATA STRUCTURE

- List

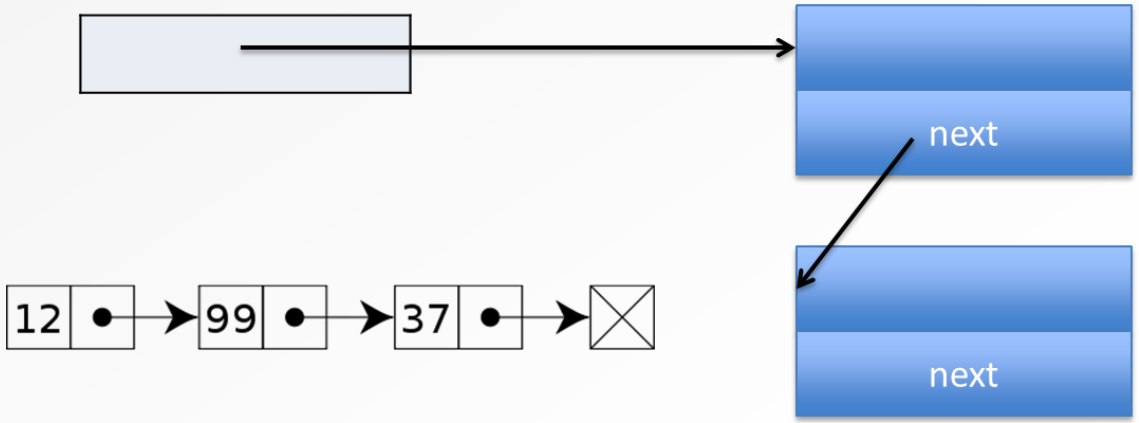
- Sequential data
  - **Order** among items (No.1, No.2., No.3, …)
    Each item has a place in the sequence
    Each item comes after another item
- Store a list of items
  - List of names, list of numbers ,etc.
  - Two ways to store a list: Array, Linked list

OUTLINE

- Linear Data Structure
- **Data structures as nodes + links**
- Storing Lists in Arrays
- Storing Lists in Links: Linked List
- Implementing a node
- Implementing Linked List
- Common Mistakes

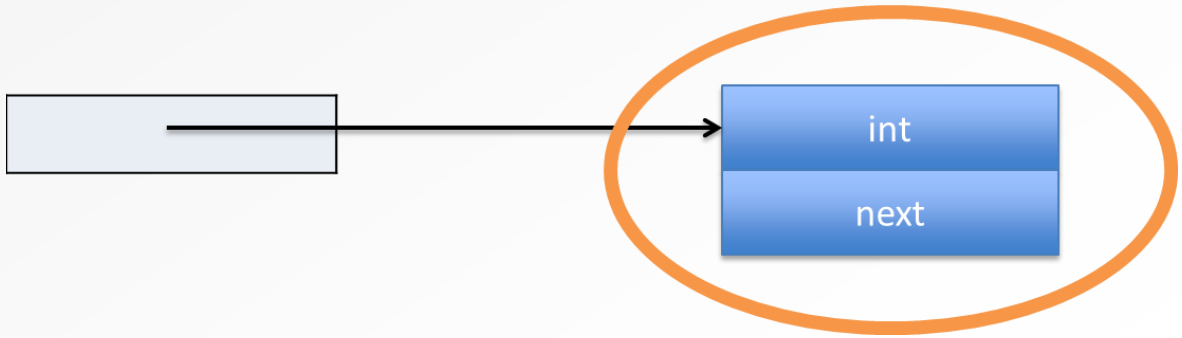
MALLOC() BASICS: STRUCT TO STRUCT

- 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

OUTLINE

- Linear Data Structure
- Data structures as nodes + links
- **Storing Lists in Arrays**
- Storing Lists in Links: Linked List
- Implementing a node
- Implementing Linked List
- Common Mistakes

STORING A LIST IN AN ARRAY

(Static array version, talked in last lecture)

- Allocate some fixed size array

```
1.  void main (){
2.      int n;
3.      int numArray[100];
4.      scanf("%d", &n);  //user input how many integers
5.      for (int i=0; i<n; i++){
6.              scanf("%d", &numArray[i]);}
7.  }
```

What if n<100 or n>100??

STORING A LIST IN AN ARRAY

(Dynamic array version, talked in last lecture)

- Allocate exactly the right sized array

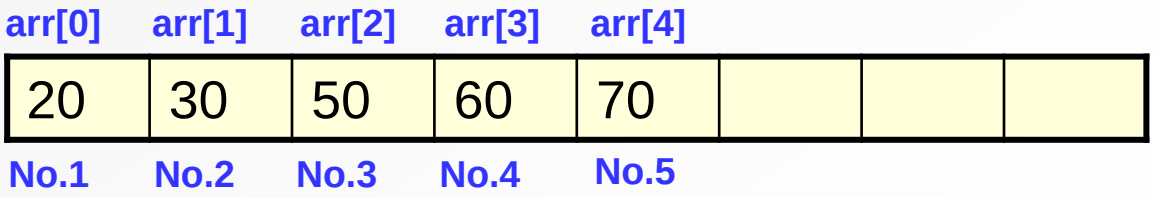
```
1.  #include <stdlib.h>
2.  void main (){
3.      int n;
4.      int *numArray;
5.      scanf("%d", &n);
6.      numArray = malloc(sizeof(int)*n);
7.      for (int i=0; i<n; i++){
8.              scanf("%d", numArray+i); }
9.  }
```

Looks like a good solution
But what if we want to change the list?

LISTS STORED IN ARRAYS

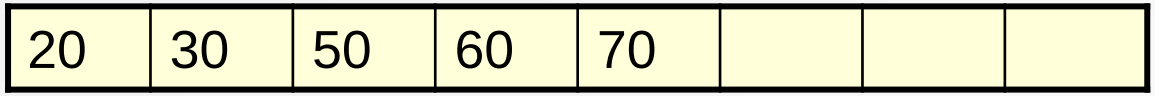
- Items have to be stored in **contiguous** block
- No gaps in between items
- **Easy to**
  - **Random access to items in the sequence.**

    e.g., the ith item can be accessed by arr[i-1]



CHANGE LISTS STORED IN ARRAYS

- The existing list:

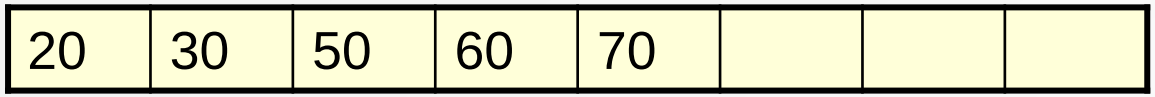


  - Add a number
    - At the front
    - At the back
    - In the middle
  - Remove a number
    - From the front
    - From the back
    - From the middle
  - Move a number to a different position

- Is it easy to do?

CHANGE LISTS STORED IN ARRAYS

- The existing list:



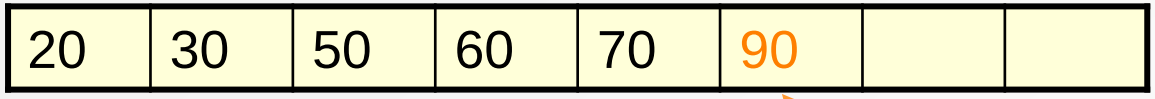
  - **Add a number**
    - At the front
    - At the back
    - In the middle
  - Remove a number
    - From the front
    - From the back
    - From the middle
  - Move a number to a different position

The array should have at least 1 unused element for adding a number

- Is it easy to do?

CHANGE LISTS STORED IN ARRAYS

- The existing list:



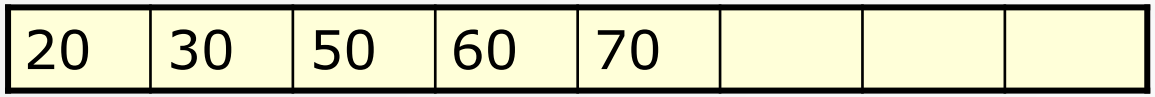
  - Add a number
    - At the front
    - **At the back**
    - In the middle
  - Remove a number
    - From the front
    - From the back
    - From the middle
  - Move a number to a different position

Insert the new item into the next empty array element

- Is it easy to do?

CHANGE LISTS STORED IN ARRAYS

- The existing list:



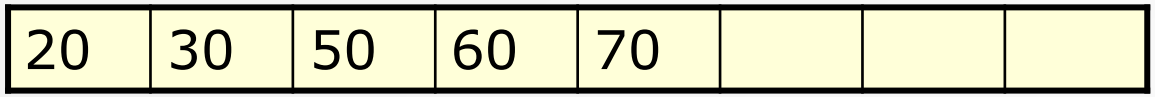
  - Add a number
    - **At the front**
    - At the back
    - In the middle
  - Remove a number
    - From the front
    - From the back
    - From the middle
  - Move a number to a different position

1. **Shift all elements to the right by 1**
2. **Insert the new item 10 into the first array element**

- Is it easy to do?

CHANGE LISTS STORED IN ARRAYS

- The existing list:



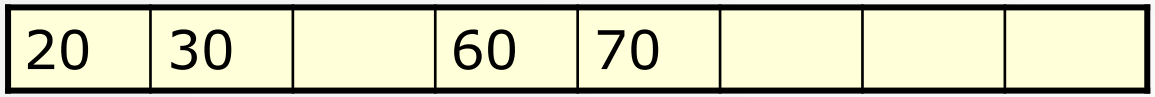
  - Add a number
    - At the front
    - At the back
    - In the middle
  - Remove a number
    - From the front
    - From the back
    - **From the middle**
  - Move a number to a different position

1. **Remove item from array**
2. **Shift all elements to the left by 1**

- Is it easy to do?

CHANGE LISTS STORED IN ARRAYS

- The existing list:



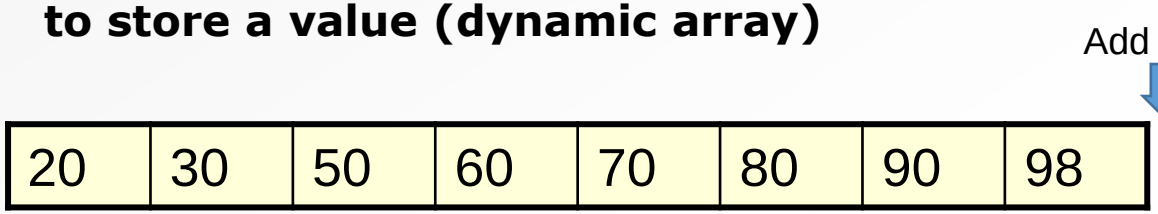
  - Add a number
    - At the front
    - At the back
    - In the middle
  - Remove a number
    - From the front
    - From the back
    - **From the middle**
  - Move a number to a different position

1. **Remove item 50 from array**
2. **Shift all elements to the left by 1**

- Is it easy to do?

REMARKS ON "LIST STORED IN ARRAY"

- Items have to be stored in **contiguous** block
- No gaps in between items
- **Easy to**
  - **Random access to items in the sequence**
  - Add at / remove from the back
- Not so easy to
  - Add at / remove from the front/middle
  - **Add items when all array elements have been used to store a value (dynamic array)**



LIST DATA STRUCTURE

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

OUTLINE

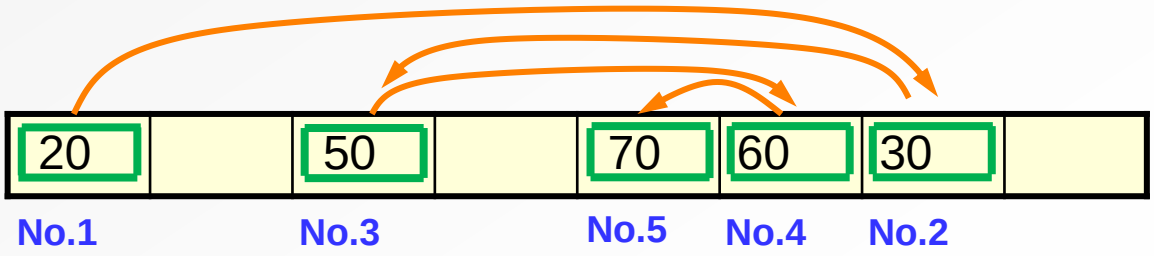
- Linear Data Structure
- Data structures as nodes + links
- Storing Lists in Arrays
- **Storing Lists in Links: Linked List**
- Implementing a node
- Implementing Linked List
- Common Mistakes

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

Link: pointer to the next item

Linked list: nodes with links

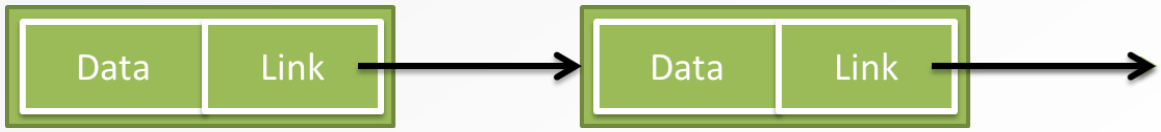


BASIC LINKED LIST

- 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 2 components
  - Data stored in that node
  - Link to the next node in the sequence



OUTLINE

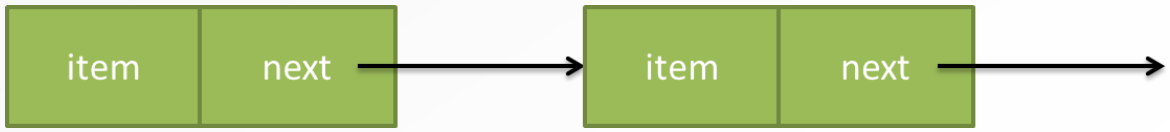
- Linear Data Structure
- Data structures as nodes + links
- Storing Lists in Arrays
- Storing Lists in Links: Linked List
- **Implementing a node**
- Implementing Linked List
- Common Mistakes

BASIC LINKED LIST NODE

- Each node has a ListNode structure
- Basic nodes have two components
  - Data stored in that node: integer, char, …
  - Link: pointer pointing to the next node in the sequence

MINIMUM SETTINGS

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



BASIC LINKED LIST NODES

- Lets statically create a node
  - Declared at compile time

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

BASIC LINKED LIST NODES

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

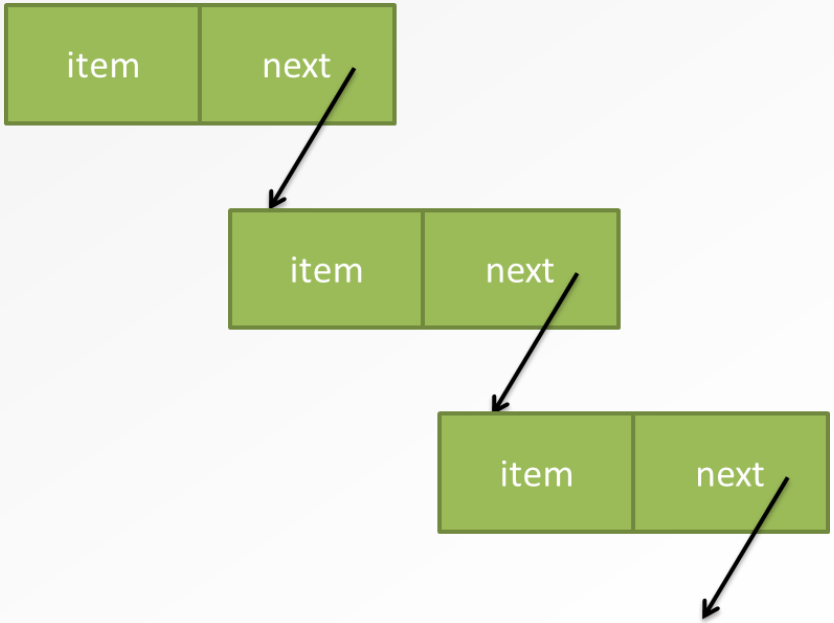
```
ListNode *dy_node = malloc(sizeof(ListNode));
dy_node->item = 50;
dy_node->next = NULL;
```

OUTLINE

- Linear Data Structure
- Data structures as nodes + links
- Storing Lists in Arrays
- Storing Lists in Links: Linked List
- Implementing a node
- **Implementing 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



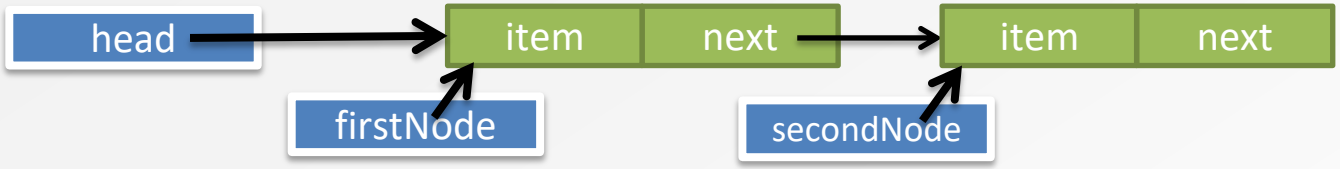
LINKED LIST OF NODES

- 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

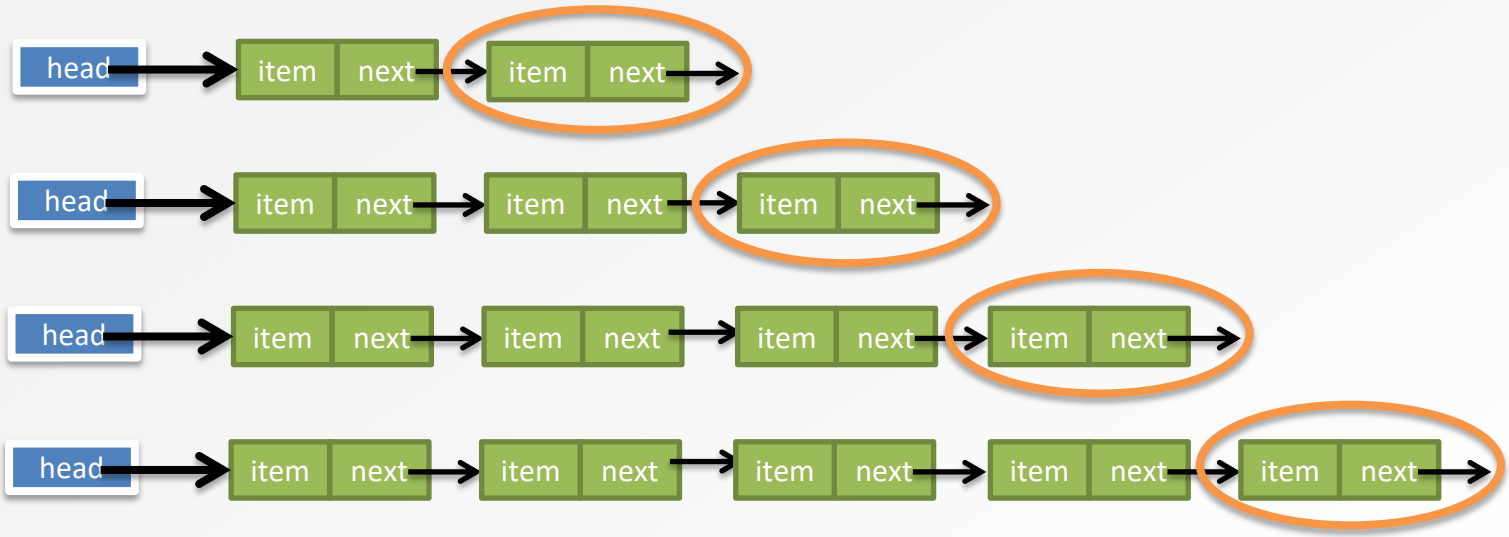



```
1     typedef struct node{
2         int item;
3         struct node *next;
4     } ListNode;
5
6     int main(){
7         ListNode *head, *firstNode, *secondNode;
8
9         firstNode = malloc(sizeof(ListNode));
10        secondNode = malloc(sizeof(ListNode));
11
12        head = firstNode;
13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

STORE A LIST OF NUMBERS

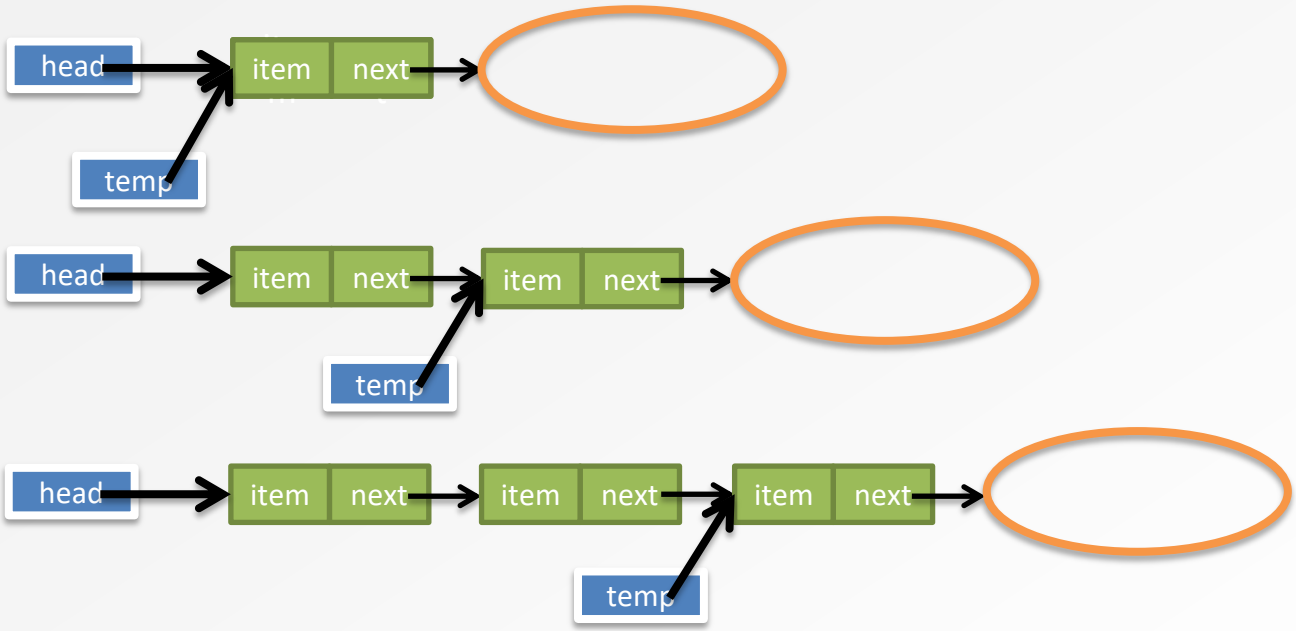
- Previously, used malloc() to create int array to store all numbers after numOfNumbers was known
- This time, use malloc() to create a new ListNode for each number
  - Get input until input == -1
  - For each input number, create a new Node to store the value
  - Arrange all the ListNodes as a linked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

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

STORE A LIST OF NUMBERS



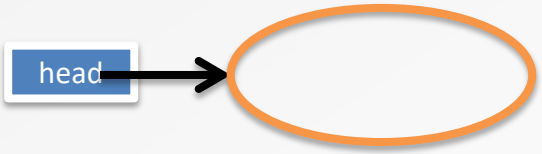
- *temp* pointer stores address of the last ListNode at any time
- Create a new ListNode

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

STORE A LIST OF NUMBERS

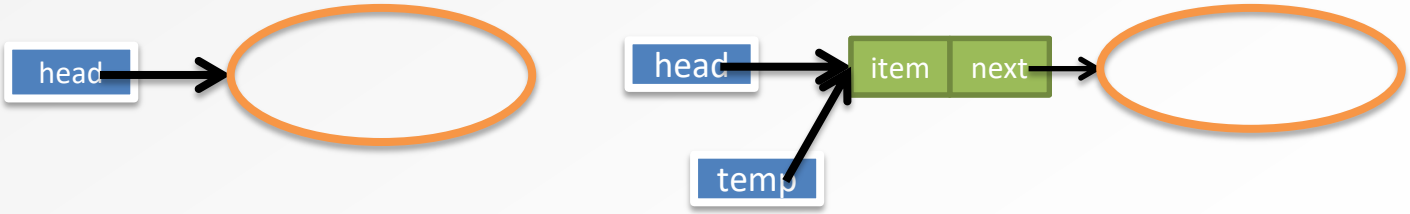
- Watch out for special case
  - First node in the linked list
  - *head* == NULL
  - Need to update the *head* pointer

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



STORE A LIST OF NUMBERS

- After 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

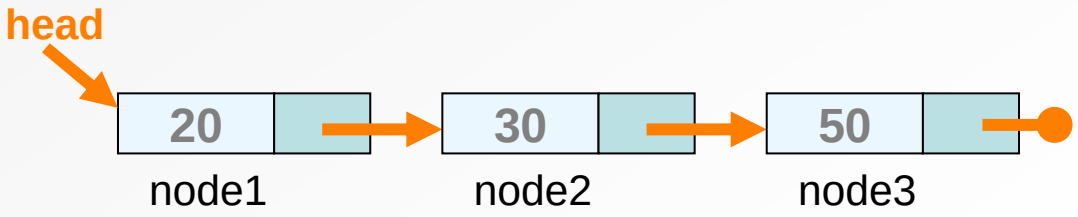
```
1       typedef struct node{
2            int item; struct node *next;
3       } ListNode;
4
5       int main(){
6            ListNode *head = NULL, *temp;
7            int i = 0;
8
9            scanf("%d", &i);
10           while (i != -1){
11                if (head == NULL){
12                     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      }
```

OUTLINE

- Linear Data Structure
- Data structures as nodes + links
- Storing Lists in Arrays
- Storing Lists in Links: Linked List
- Implementing a node
- Implementing 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"

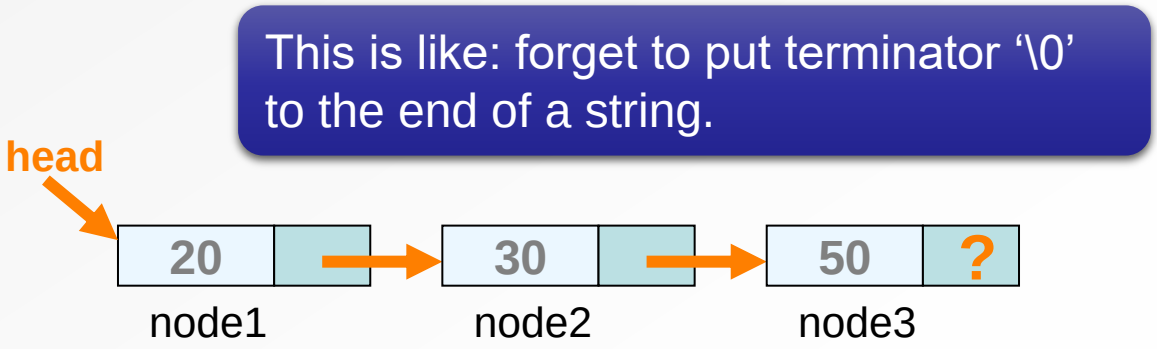


COMMON MISTAKES

- **Forget** to deal with the last node differently.
  - when we build a linked list, or insert a node to the end of the list,

    forget to set the pointer ***next*** of the **last node** to be **NULL**

  This will cause mistake when tracking the linked list.



NEXT TIME...

- 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

STRUGGLING (IN ORDER) TO LEARN

- http://anniemurphypaul.com/2014/02/when-and-how-to-let-learners-struggle/
- Allowing learners to struggle will actually help them learn better, according to research on "productive failure" conducted by Manu Kapur, a researcher at the Learning Sciences Lab at the National Institute of Education of Singapore.
- …model adopted by many teachers when introducing others to new knowledge — providing lots of structure and guidance early on, until the students or workers show that they can do it on their…not the best way to promote learning.
- …better to let neophytes wrestle with the material on their own for a while, refraining from giving them any assistance at the start.
- The struggles of the second group have what Kapur calls a "hidden efficacy": they lead people to understand the deep structure of problems, not simply their correct solutions. When these students encounter a new problem of the same type on a test, they're able to transfer the knowledge they've gathered more effectively than those who were the passive recipients of someone else's expertise.