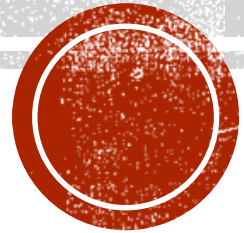
M3:
LINKED LISTS, STACKS, QUEUES

Summer 2019 – CSC 395 – ST: Algorithm & Data Structure Concepts



SOME RESOURCES

- https://legacy.gitbook.com/book/cathyatseneca/data-structures-and-algorithms/details

- See chapter 4 from http://people.cs.vt.edu/~shaffer/Book/C++3e20130328.pdf

SOME RESOURCES(2)

- suggested videos:
  - linked list: https://www.youtube.com/watch?v=lcNL_HLpcEs
  - linked list: https://www.youtube.com/watch?v=3svB0kM6f10

- visualization:
  - linked lists: https://visualgo.net/en/list

  - liang - stack: http://www.cs.armstrong.edu/liang/animation/web/Stack.html
  - liang - queue: http://www.cs.armstrong.edu/liang/animation/web/Queue.html
  - liang - array list: http://www.cs.armstrong.edu/liang/animation/web/ArrayList.html
  - liang - linked list: http://www.cs.armstrong.edu/liang/animation/web/LinkedList.html

LISTS

- A **list** is an **ordered sequence of values**.

- It may have properties such as being sorted/unsorted, having duplicate values or being unique.

- The most important part about list structures is that the data has an ordering (which is not the same as being sorted).
  - Ordering simply means that there is an idea that there is a "first" item, a "second" item and so on.

- Lists typically have a subset of the following operations:
  - initialize
  - add an item to the list (at the beginning, at the end, at a specified index)
  - remove an item from the list (at the beginning, at the end, at a specified index)
  - search
  - sort
  - iterate through all items, etc.

LISTS(2)

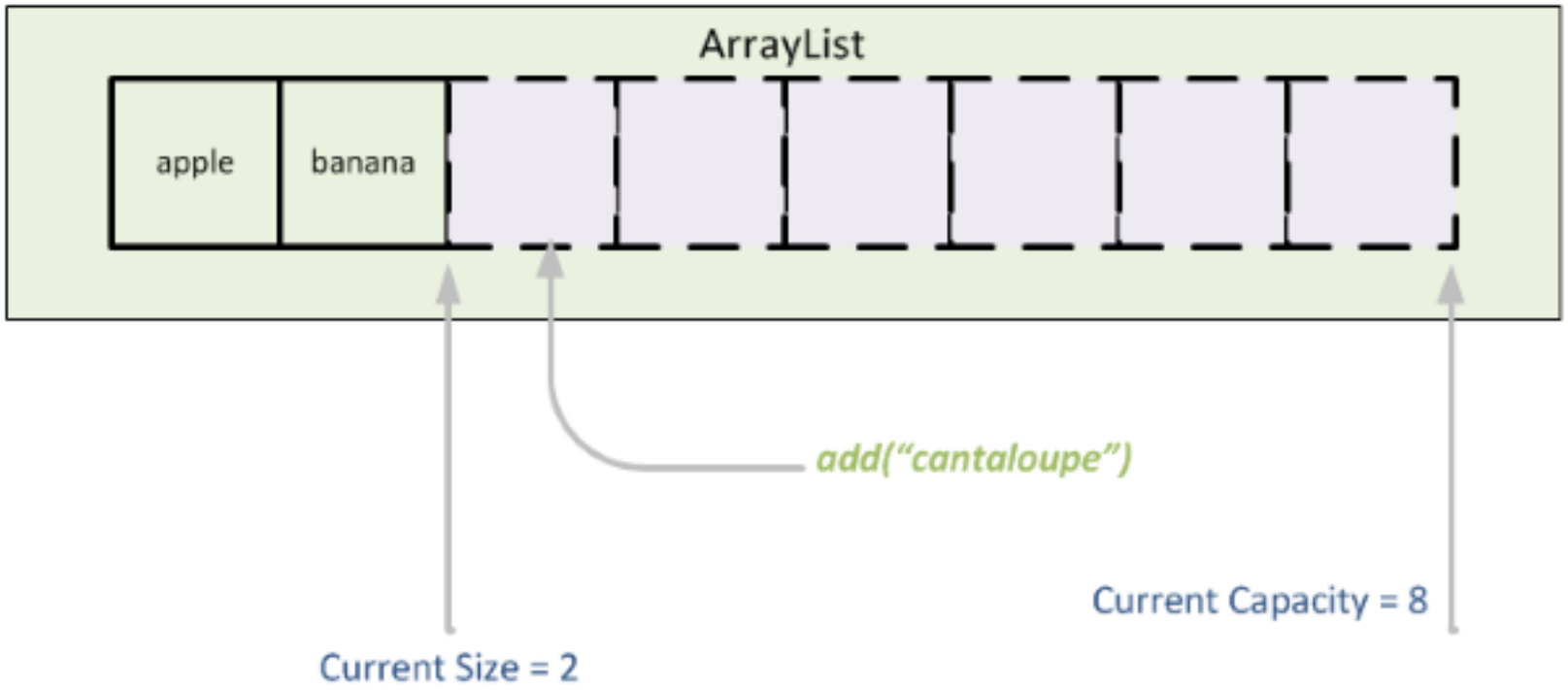
- Two general implementation methods are
  - use an array like data structure (based on arrays)
  - use a linked list data structure (based on linked nodes)
- We will look at each in turn.

- Note: here's a list of some important classes in the **System.Collections** namespace.
  We'll look (glance) at them at the end of this chapter. Unless otherwise specified do not use them!
  - ArrayList -- this implements the IList interface and represents a collection whole size can increase dynamically and can store objects of any type. Data inside an ArrayList instance can be accessed using the index.
  - Queue -- this represents a first–in first–out non–generic collection of objects
  - Stack -- this represents a last–in first–out non–generic collection of objects
  - ~~Hashtable -- this represents a collection that can store objects as key / value pairs~~
  - ~~BitArray -- this represents a collection that can manage a compact array of bit values where the values are represented as Booleans~~
  - ~~SortedList -- this represents a collection of objects stored as key / value pairs that are sorted by the keys. Objects in a sorted list are accessible both using their keys and indexes.~~

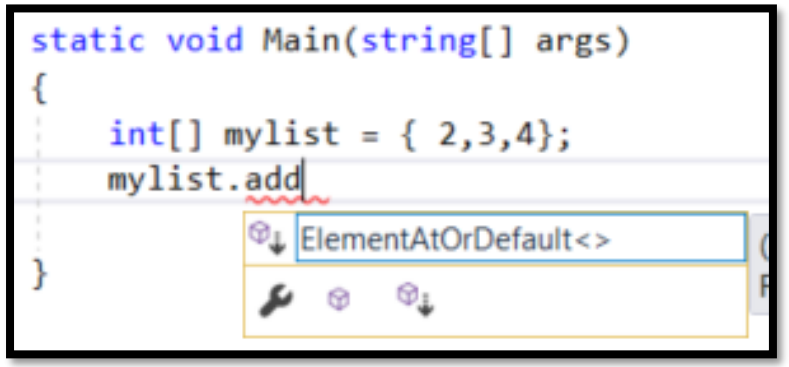
LISTS(3) – PLEASE REVIEW THIS LATER!

- If you used array:
  - items are stored in memory consecutively and
  - you can have direct access to any particular item through the use of its index in constant time.
  - When sorted, the list can be searched using binary search.
  - Making the list grow can be expensive and space is often wasted as large amount of space may be allocated but not used.
  - Insertion into anywhere other than the very end of the array is an expensive operation as it requires the shifting of all values from the point of insertion to the end of the array.
  - Removal of any value anywhere other than the very last item is also expensive as it also requires a shift in all items from the point of removal to the very end of the list.
- A linked list
  - is very easy to grow and shrink.
  - Data is not stored in consecutive memory locations so a large block of contiguous memory is not required even for storing large amounts of data.
  - Each piece of data requires the storage of an extra pointer. However, the amount of extra data is related to number of items in the list already.
  - A linked list cannot be searched using binary search as direct access to nodes are not available.
  - However, both insertion and removal of any node in the list (assuming that the position of the insertion/removal is known) is very efficient and runs in constant time.

ARRAY-BASED LISTS (ARRAY LISTS) WE'LL REVISIT THIS IN MODULE 6



ARRAY-BASED LISTS SE MODULE 6 ...



- Can you add a new value to an existing array?
- To implement a list using arrays, we allocate more space than is necessary.
- The array is used until it is full.
- Once an array is full, either all new insertions fail until an item is removed or the array must be reallocated.
  - The reallocation typically involves creating a larger array, copying over the old data, and making this the array.
  - As the process of growing an array requires the typically copying of the entire array, this is not something you want to do often
  - That is, you do not grow the array a few elements at a time but in large chunks instead.
  - The exact number of elements to grow by is implementation specific.

ARRAY-BASED LISTS – SEE MODULE 6

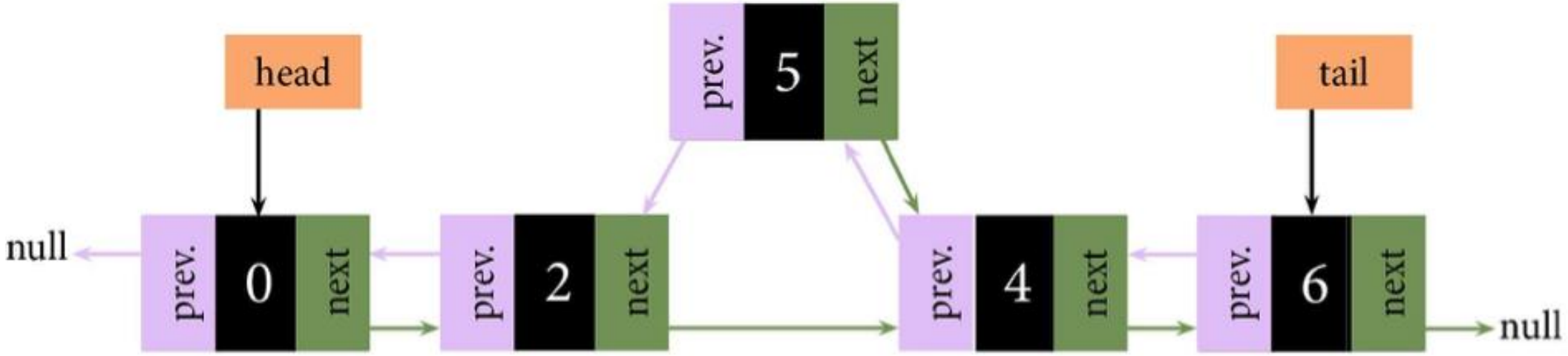
- Create a class MyArrayList that is an array-based list and implements:
  - isEmpty()
  - isFull()
  - size()
  - printAll()
  - addBack(value) ←add(value)
  - addFront(value)
  - insert(index)
  - deleteBack()
  - deleteFront()
  - delete(index)
  - contains(value)
  - clear()
  - sort()

what is the big Oh for each of these operations?

←think of these as user interface
methods define the behavior
they usually operate on data

Lists typically have a subset of the following operations:
- initialize
- add an item to the list
- remove an item from the list
- search
- sort
- iterate through all items, etc.

LINKED LISTS



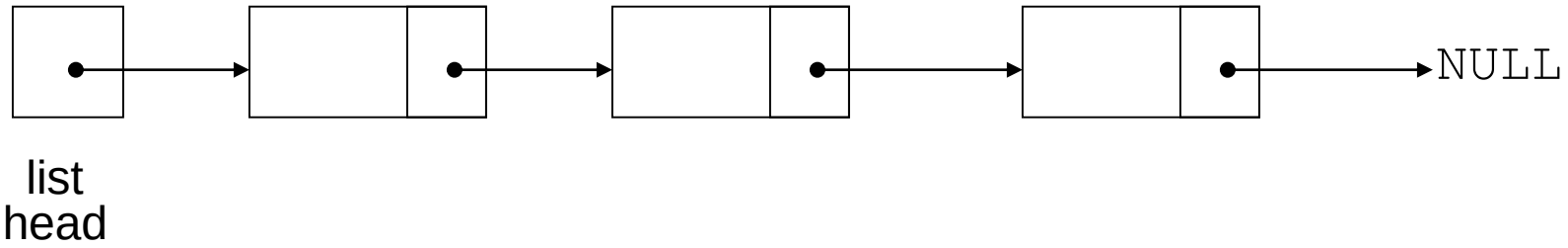
LINKED LIST ADT – DEFINITION

- A linked list is a collection of class objects called nodes.
  - Each node is linked to its successor node in the list using a reference to the successor node.
  - A node is made up of a field for storing data and the field for the node reference.
  - The reference to another node is called a link.
  - Moving through a linked list involves following the links from the beginning node to the ending node.
- A major difference between an array and a linked list
  - The elements in an array are referenced by position (the index).
    - Hence it takes O(1) to access each of these elements
  - The elements of a linked list are referenced by their relationship to the other elements of the linked list
    - Hence it takes up to O(n) to access an element

An Example Linked List.

LINKED LIST ADT – DEFINITION (2)

- Linked list: is a series of connected nodes, where each node is a data structure.
- It can grow or shrink in size as the program runs.
  This is possible because the nodes in a linked list are dynamically allocated.
- If new data need to be added to a linked list, the program simply allocates another node and inserts it into the series.
- If a particular piece of data needs to be removed from the linked list, the program deletes the node containing that data.



LINKED LIST OPERATIONS

Basic operations:

- append a node to the end of the list
- insert a node within the list (maintain order!)
- traverse the linked list
- delete a node
- delete/destroy the list

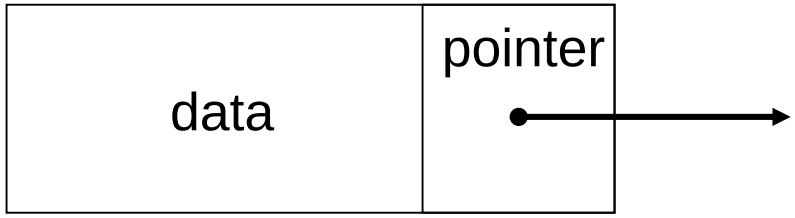
Other useful operations:

- isEmpty()
- sort()
- addFront()
- addBack()
- reverseList()

NODE ORGANIZATION

A node contains:

- **data**: one or more data fields – may be organized as structure, object, etc.
- **a pointer** that can point to another node



- Declaring a Node (use struct or class)

```
public class ListNode
  {
    int data;
    ListNode next;
  };
```

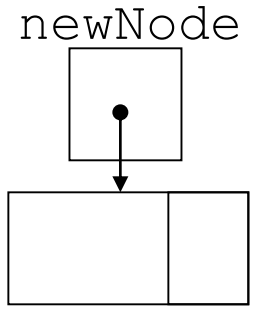
- No memory is allocated at this time

```
public class ListNode
{
    public int data;
    public ListNode next;

    //constructor
    public ListNode(int value)
    {
        data = value;
        next = null;
    }
}
```

CREATE A NEW NODE

- **Allocate memory for the new node:**

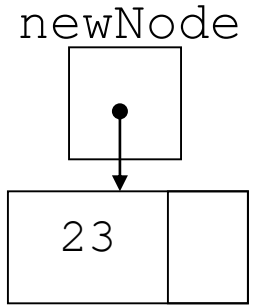
```
newNode = new ListNode();
```



- **Initialize the contents of the node:**

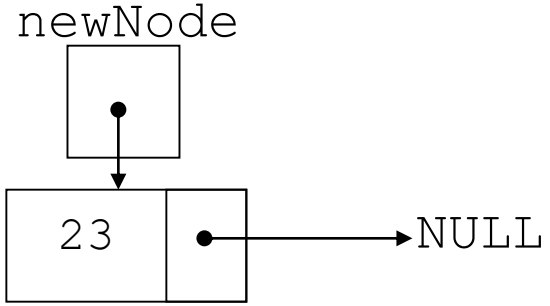
```
newNode.value = num;
```

[or call the node constructor]



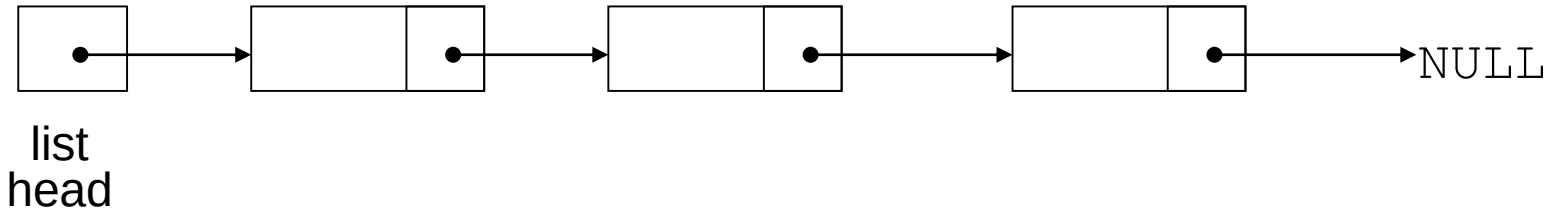
- **Set the pointer field to `null`:**

```
newNode.next = null;
```



LINKED LIST ORGANIZATION

- Linked list contains 0 or more nodes:

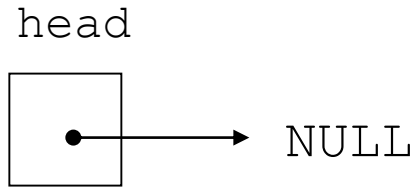


- Has a list head to point to first node
- Last node points to NULL

  - Define a pointer for the head of the list:

```
ListNode head = null;
```

  - Head pointer initialized to NULL to indicate an empty list



C# IMPLEMENTATION ... LIST ADT

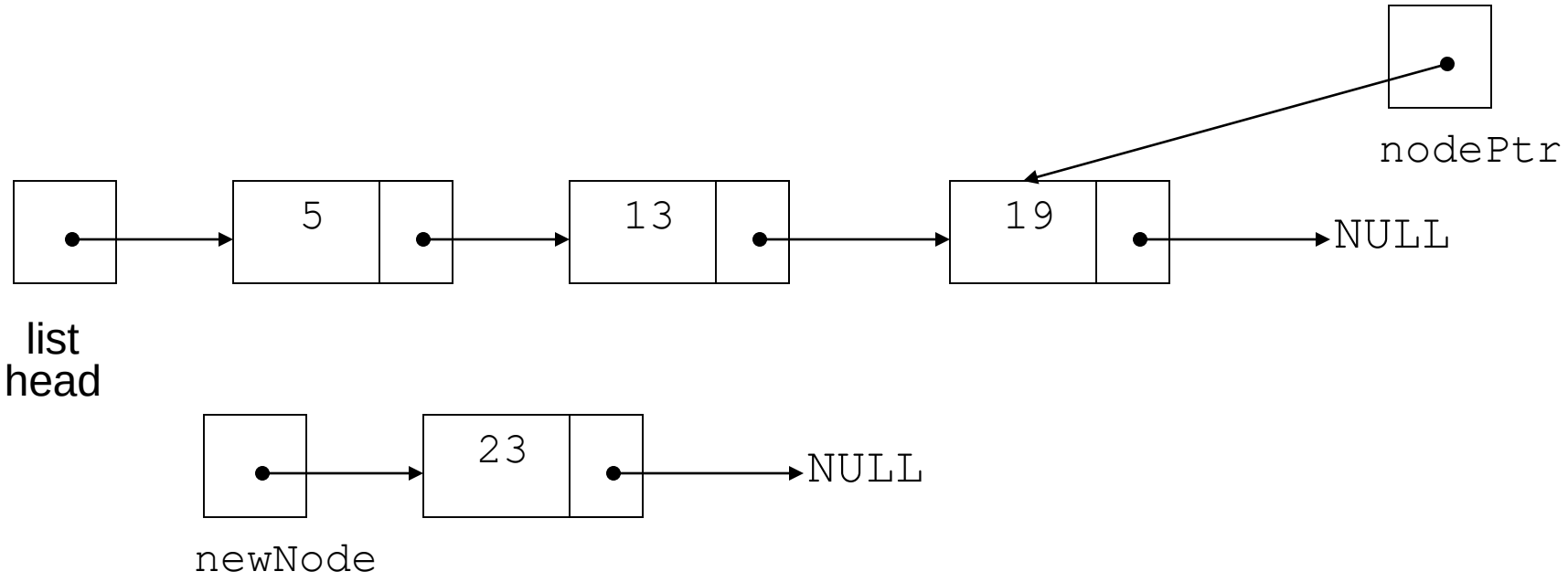
```
public class LinkedList{
  private:

        // Declare a structure for the list
        private class ListNode{ ... }


        private ListNode head;              // List head pointer

  public:
        // Constructor
        LinkedList() { head = null; }

        // Linked list operations
        void appendNode(double);
        void insertNode(double);
        void deleteNode(double);
        void displayList();
        void isEmpty();
        void addFront();
        void addBack();              ←same as appendNode...  Implement it!
        void reverseList();
        void deleteList();
        void sort();                 ←multiple ways to do it
  };
```

APPENDING A NODE

- Add a node to the end of the list

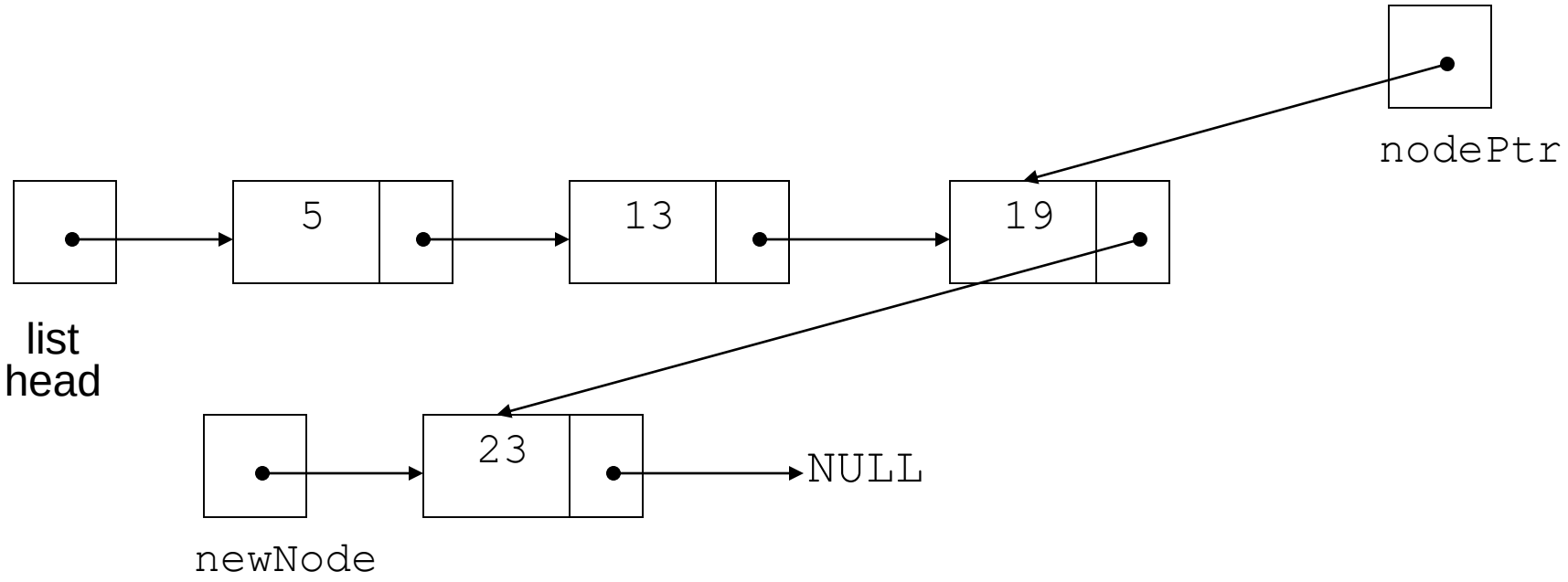
- Basic process:
  - Create the new node (as already described)
  - Add node to the end of the list:
    - If list is empty, set head pointer to this node
    - Else,
      - traverse the list to the end
      - set pointer of last node to point to new node

APPENDING A NODE



New node created, end of list located

APPENDING A NODE

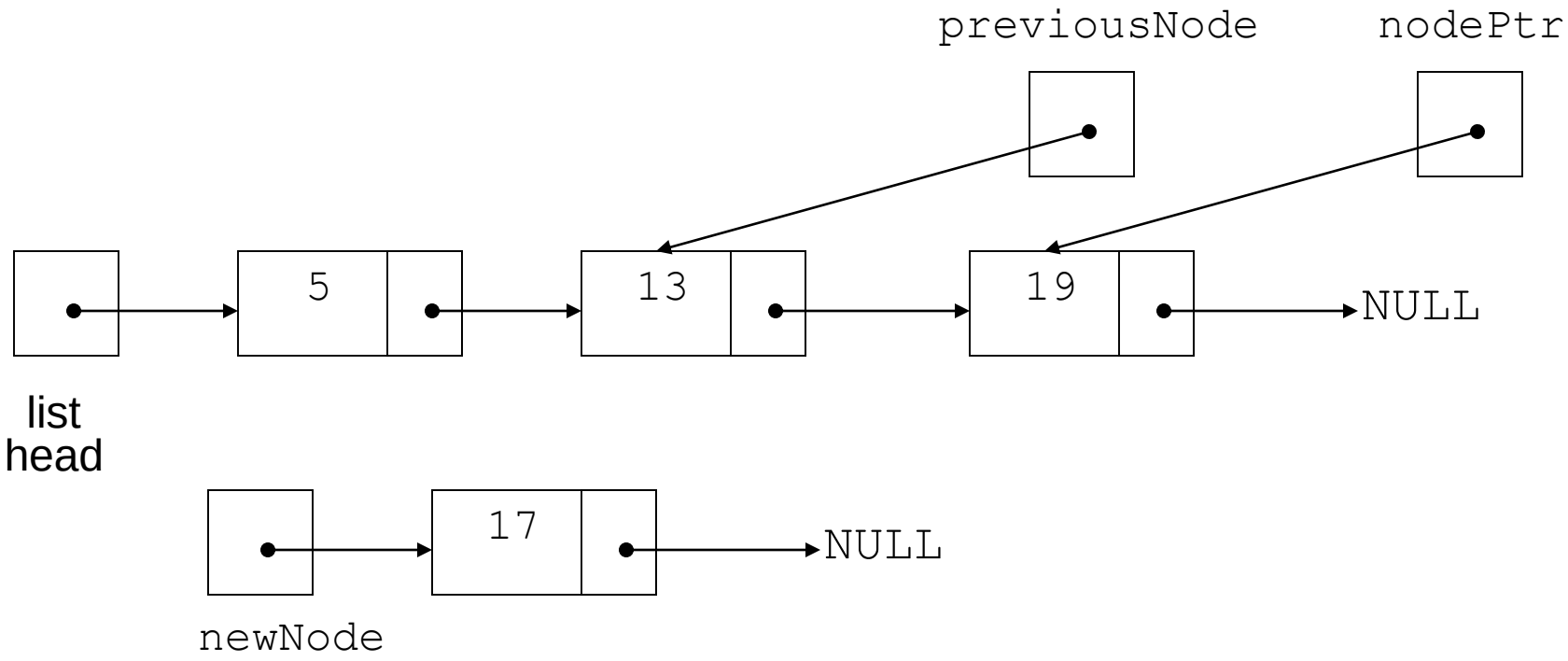


New node added to end of list

INSERTING A NODE INTO A LINKED LIST

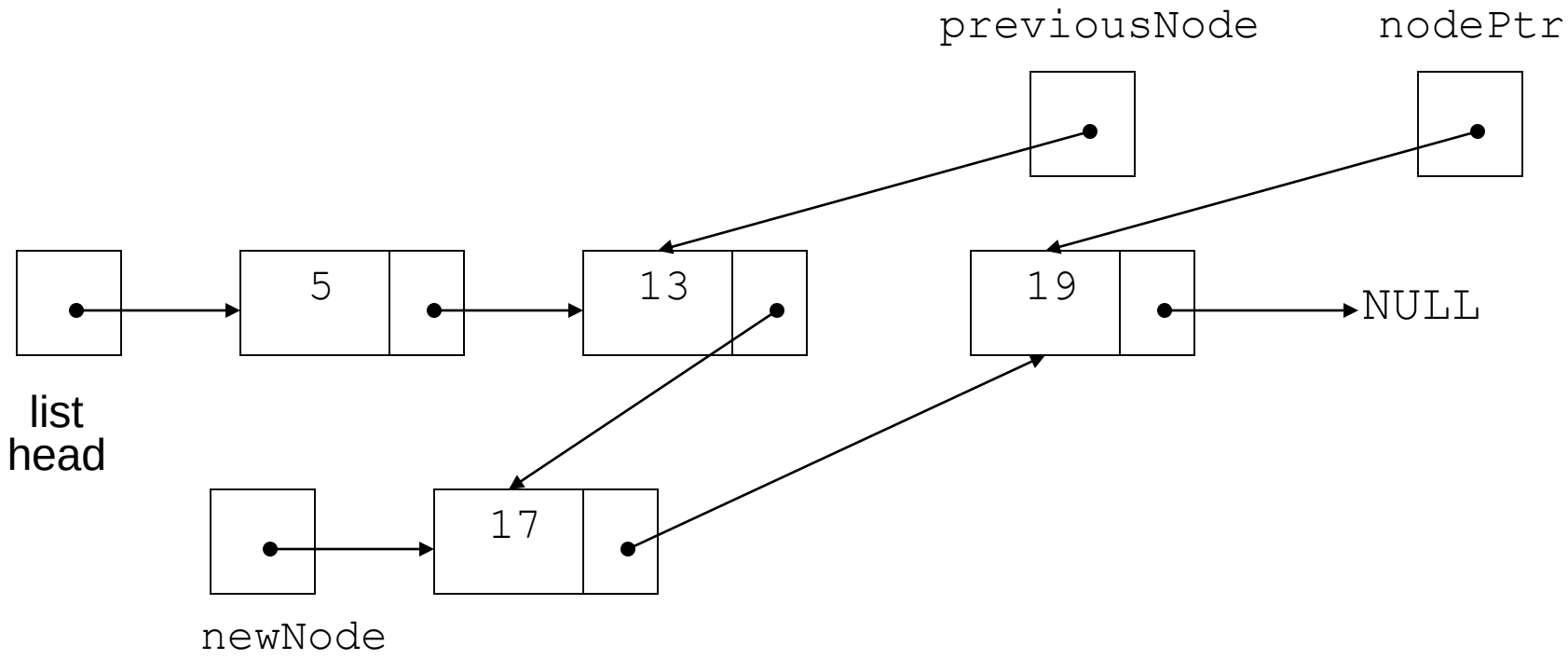
- Used to maintain a linked list in order
- Requires two pointers to traverse the list:
  - pointer to locate the node with data value greater than that of node to be inserted
  - pointer to 'trail behind' one node, to point to node before point of insertion
- New node is inserted between the nodes pointed at by these pointers

INSERTING A NODE INTO A LINKED LIST



New node created, correct position located

INSERTING A NODE INTO A LINKED LIST



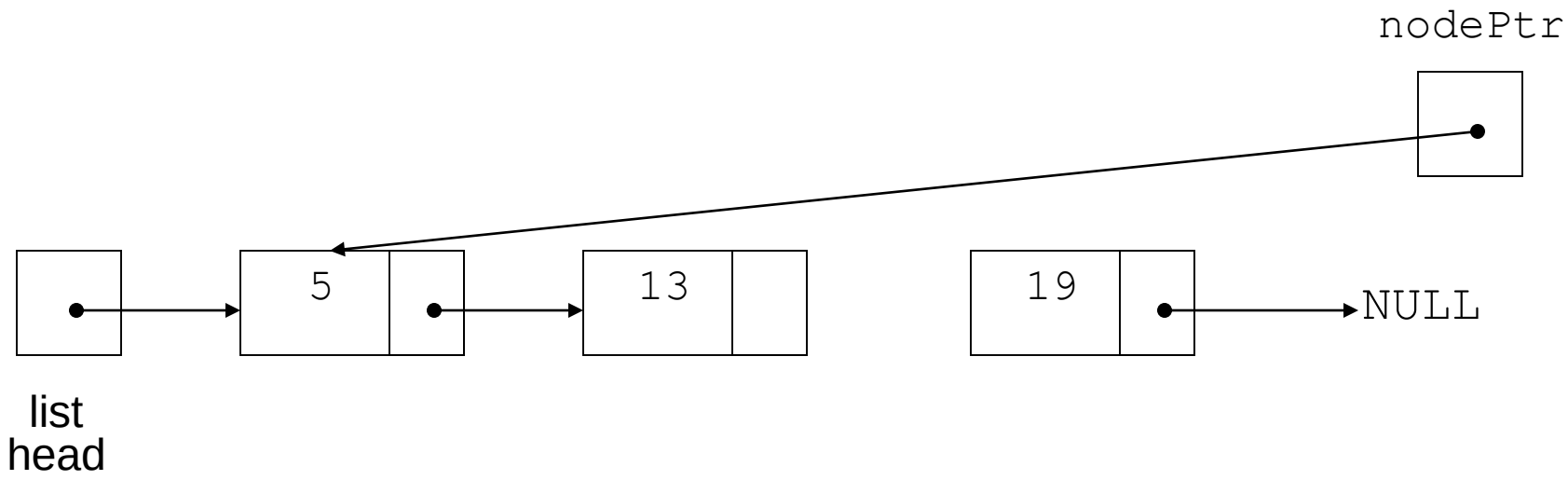
New node inserted in order in the linked list

TRAVERSING A LINKED LIST

- Visit each node in a linked list: display contents, validate data, etc.

- Basic process:
  - set a pointer to the contents of the head pointer
  - while pointer is not `NULL`
    - process data
    - go to the next node by setting the pointer to the pointer field of the current node in the list
  - end while

TRAVERSING A LINKED LIST

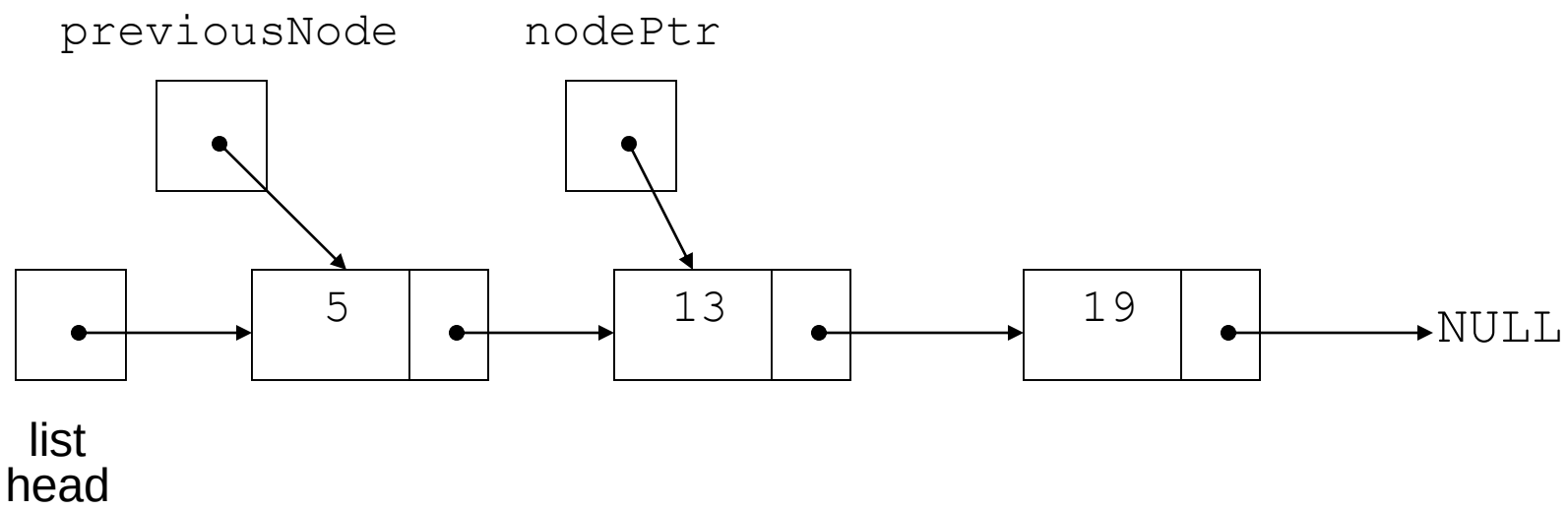


`nodePtr` points to the node containing `5`, then the node containing `13`, then the node containing `19`, then points to `NULL`, and the list traversal stops

DELETING A NODE

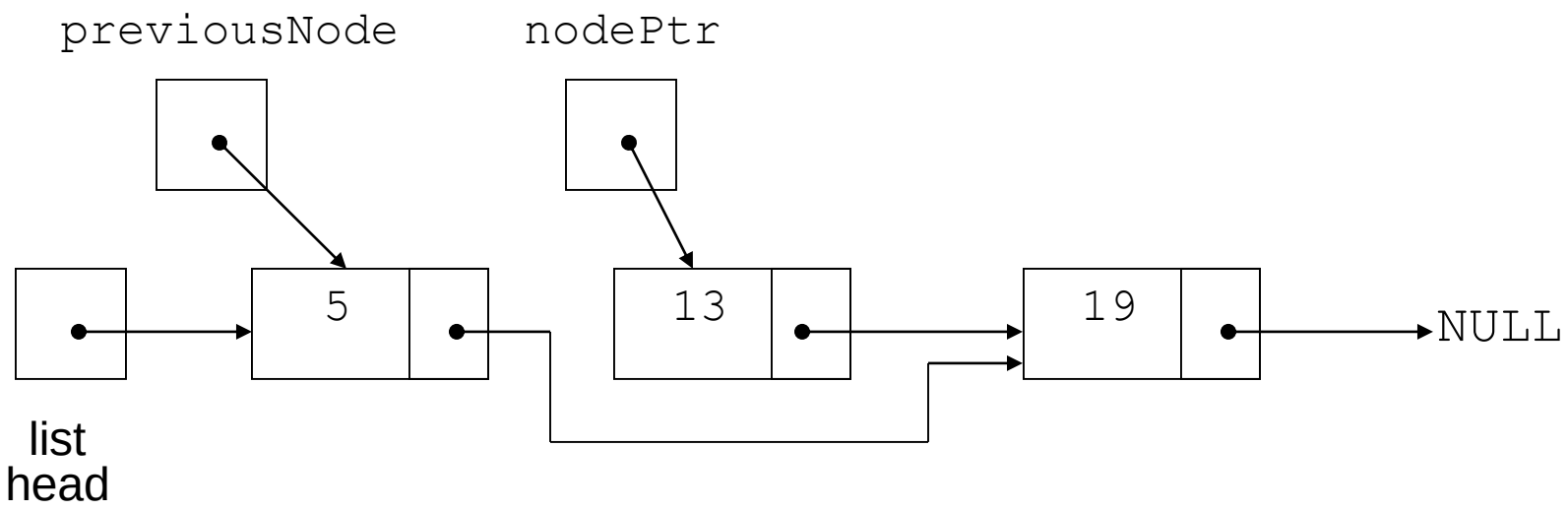
- Used to remove a node from a linked list
- If list uses dynamic memory, then delete node from memory
- Requires two pointers: one to locate the node to be deleted, one to point to the node before the node to be deleted

DELETING A NODE



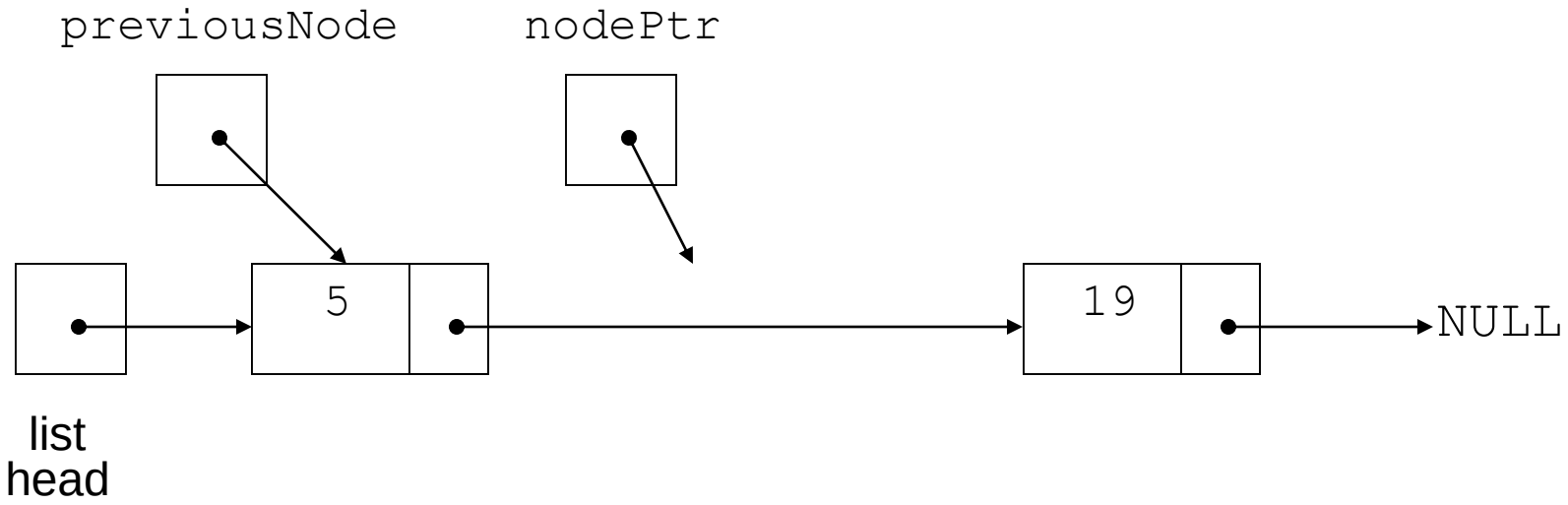
Locating the node containing `13`

DELETING A NODE



Adjusting pointer around the node to be deleted

DELETING A NODE



Linked list after deleting the node containing `13`

DOUBLY LINKED LISTS

- Let's talk about it.
  - Same interface (same methods) but the implementation may be different!

- Remember to use the recommended resources! You are expected to read more than what I cover in class so you have a better understanding of these topics
  - http://people.cs.vt.edu/~shaffer/Book/C++3e20130328.pdf

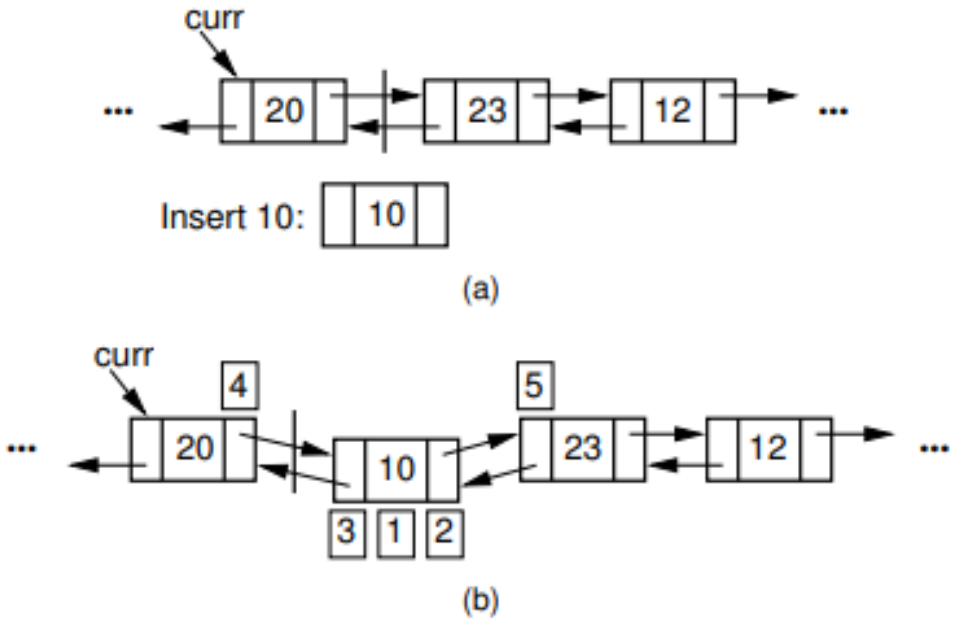


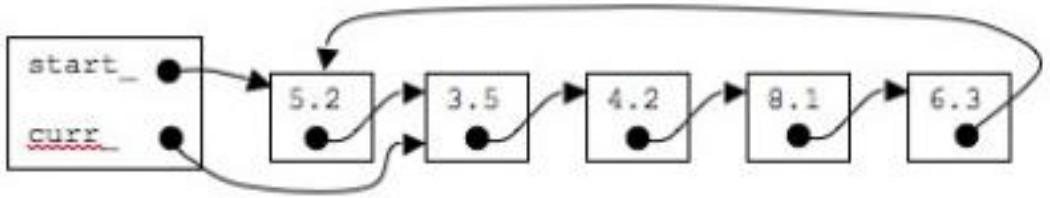
Figure 4.15 Insertion for doubly linked lists. The labels 1, 2, and 3 correspond to assignments done by the linked list node constructor. 4 marks the assignment to `curr->next`. 5 marks the assignment to the **`prev`** pointer of the node following the newly inserted node.

CIRCULAR LINKED LIST

- Let's talk about it.
  - Same interface (same methods) but the implementation may be different!

- Remember to use the recommended resources! You are expected to read more than what I cover in class so you have a better understanding of these topics

- There is also Circular Array List …



HOMEWORK FOR MODULE 3 – PART 1

```
// Linked list operations
void appendNode(double);
void insertNode(double);
void deleteNode(double);
void displayList();
void isEmpty();
void addFront();
void addBack();
void reverseList();
void deleteList();
void sort();
```

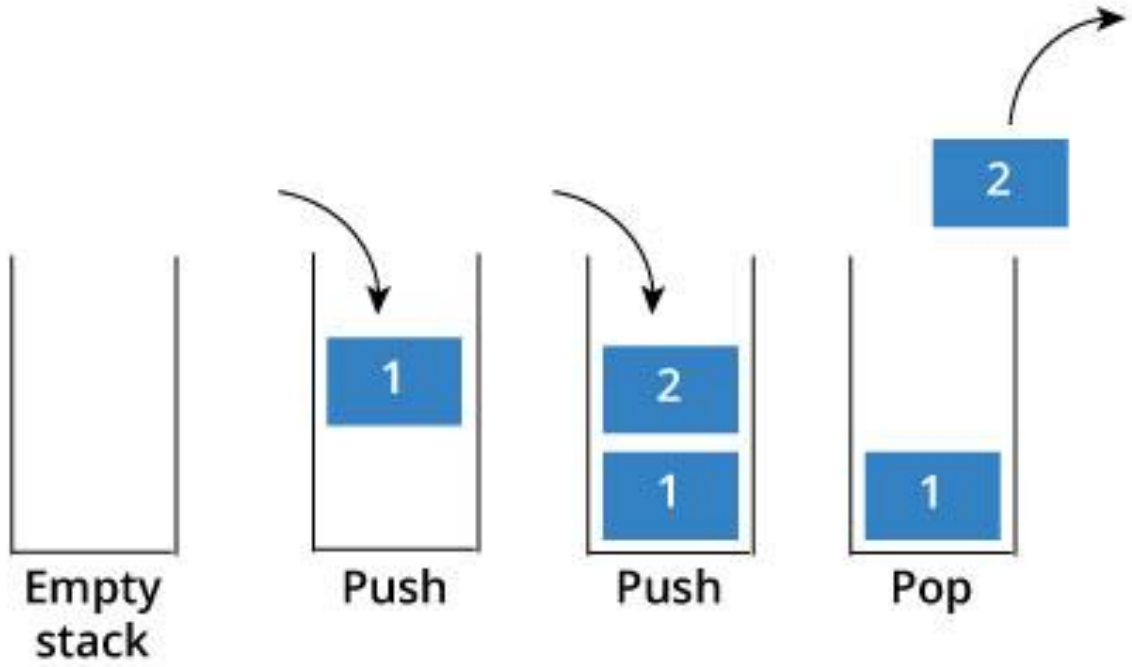
DUE: see moodle, 11:59 pm. **Make sure to include the big-Oh for each method!**

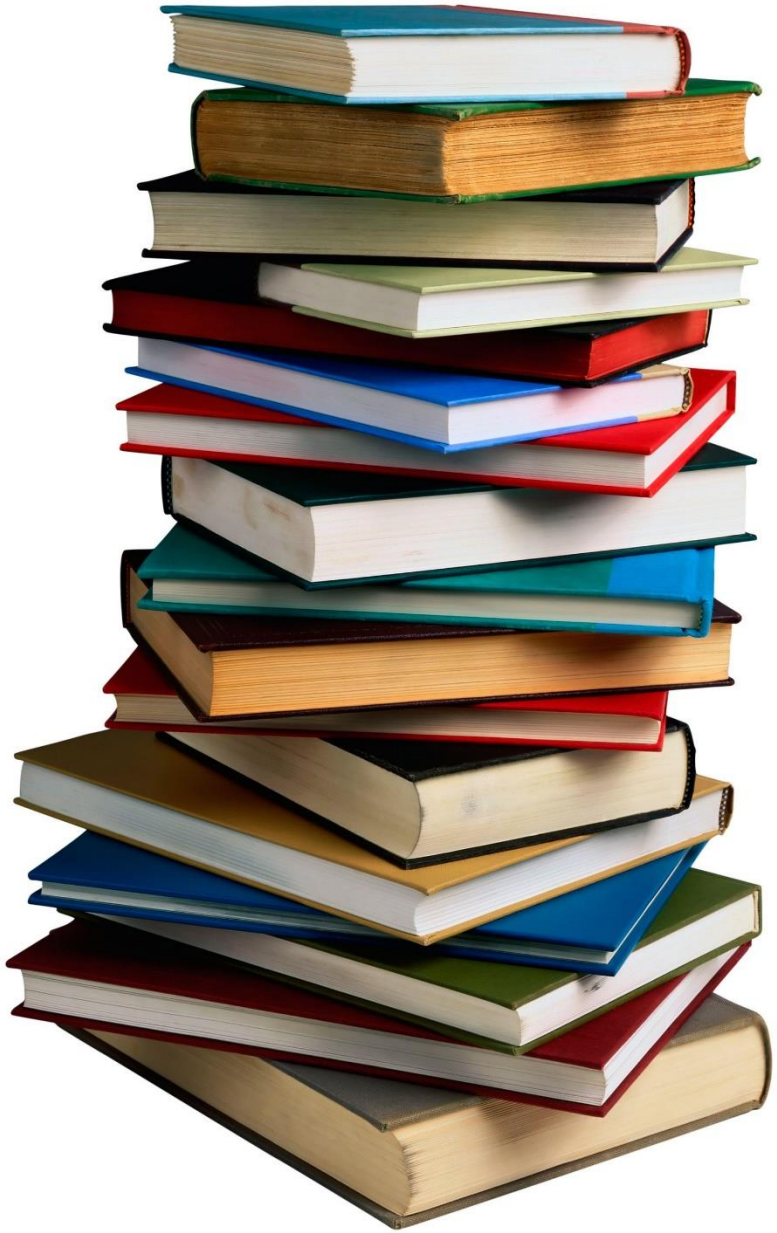
- see below for part 2, due see moodle!

Write **separate** C# programs for each problem below. Make sure to implement all methods shown here:

- **Problem 1 [30 points]:** What we saw in class was a singly linked list, because there was a link from one node to the next node. **Modify** the code in class (**make sure to use it!!!**) so it works with strings instead of integers.
- **Problem 2 [30 points]:** What we saw in class was a singly linked list, because there was a link from one node to the next node. **Modify** the code in class (**make sure to use it!!!**) so it works with integers (as we saw in class) but so it uses two links for each node, one for the next and one for the previous node. This is called doubly linked list. In addition to the methods in class also add a method void printReverse() that prints in reverse the list (it displays the last element, then the one next to last, …, and lastly it will display the first element)
- **Problem 3 [20 points]** To the code written for Problem 1 add a new method **void RemoveDuplicates()** that will remove all duplicates from the linked list. Do not use an extra array, use only O(1) extra memory for this operation. What is the running time?
- **Problem 4 [10 points]** To the code written for Problem 1 add a new method **bool IsPalindrome()** that will check whether the **singly** linked list is a palindrome. If you don't remember what palindrome is google the term or stop by my office. What is the running time? E.g. "church" → "monk" → "monk" → "church" is a palindrome.
- **Problem 5 [10 points]** To the code written for Problem 2 add a new method **bool IsPalindrome()** that will check whether the **doubly** linked list is a palindrome. If you don't remember what palindrome is google the term or stop by my office. What is the running time? E.g. 21 <-> 34 <-> 6 <-> 34 <->21 is a palindrome.

STACKS

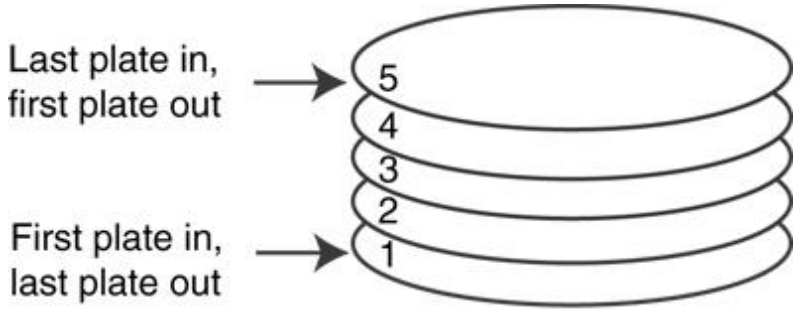




STACKS - DEFINITION

- Stack: a list of items that are accessible only from the end of the list, which is called the top of the stack.
- The standard model for a stack is the stack of trays at a cafeteria.
  - Trays are always removed from the top, and the when the dishwasher or busboy puts a tray back on the stack, it is placed on the top also.
- A stack is known as a Last-in, First-out (LIFO) data structure

- Stacks are used extensively in programming language implementations, from expression evaluation to handling function calls

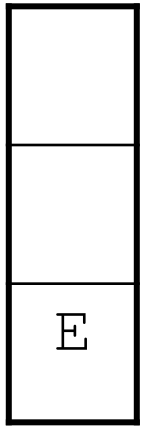


STACK ADT - OPERATIONS

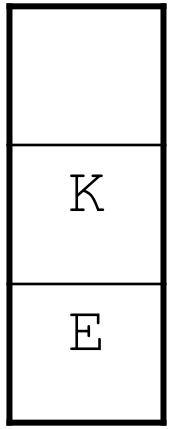
- The three primary operations of a stack
  - Push operation: Add an item to a stack with a push operation
  - Pop operation: Take an item off the stack with a Pop operation.
    - This operation returns the top item, but the operation also removes it from the stack
  - Peek operation (or top operation): Just view the top item without actually removing it.
- Other operations
  - Clear operation: remove all the items from a stack at one time.
  - Know how many items are in a stack at any time by calling the Count property.
  - Many implementations have a StackEmpty method that returns a true or false value depending on the state of the stack. We can also use the Count property for the same purposes.

STACK OPERATIONS - EXAMPLE

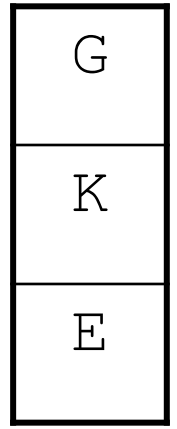
`push('E');`



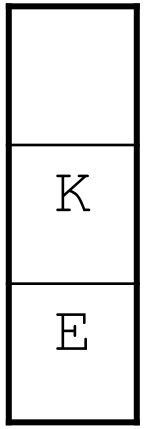
`push('K');`



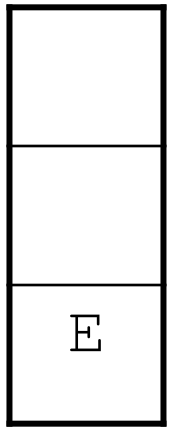
`push('G');`



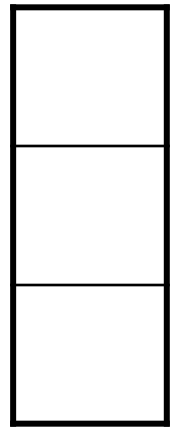
`pop();`
(remove `G`)



`pop();`
(remove `K`)



`pop();`
(remove `E`)



STACK - IMPLEMENTATION

It can be implemented in different ways

- **Static** ← fixed size (Storage requirements known prior to execution)
  - Using arrays

- **Dynamic** ←Grow and shrink as necessary
  - Using an ArrayList ←in **System.Collections**. We'll see them in the next module
  - Using linked lists

STACK - IMPLEMENTATION

Let's implement it using

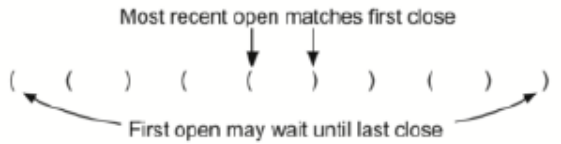
- linked lists
- arrays (using a fixed capacity)

STACKS – APPLICATIONS

- CCSCSE – 2012 competition, problem 7.
- Similarly one can check if html tags are well balanced

Problem 8

Balanced



We want to write a program that takes a string of opening and closing parentheses and checks to see whether it's *balanced*. We have exactly two types of parentheses for this problem: round brackets: () and square brackets: []. Assume that the string doesn't contain any other characters than these. This means no spaces, digits, letters, or other symbols.

Balanced parentheses require that there are an equal number of opening and closing parentheses. It requires that every opening parenthesis be closed in the reverse order opened. For example, `([])` is balanced, but `([)]` is not. It also requires that no closing parenthesis appears before an opening parenthesis.

Input
The file contains a positive integer n and a sequence of n strings of parentheses, one string per line. Each string is of length n, ($2 \leq n \leq 80$). You may assume each string contains only parentheses of type `()` and `[]`, and no other characters.

Output
Output each input string in the format below indicating whether or not it is balanced.

Sample Input

```
8
()
[]
([])
([()[]()])()
(([[()]]))
)(
((()(()))())
([)]
```

Output Corresponding to Sample Input

```
() is balanced
[] is balanced
([]) is balanced
([()[]()])() is balanced
(([[()]])) is not balanced
)( is not balanced
((()(()))()) is balanced
([)] is not balanced
```

STACKS – APPLICATIONS (2) - SKIP

- A palindrome is a string that is spelled the same forward and backward. For example
  - "dad", "madam", and "sees" are palindromes
  - "hello" is not a palindrome.
- One way to check strings to see if they're palindromes is to use a stack.
  - Read the string character by character,
  - Pushing each character onto a stack when it's read.
  - That is, storing the string backwards.
  - Popping each character off the stack
  - Comparing it to the corresponding letter starting at the beginning of the original string.
  - If at any point the two characters are not the same, the string is not a palindrome and we can stop the program.
  - Otherwise, the string is a palindrome

QUEUES


Apple Color LaserWriter 12/600
Printer Document View Help
Document Name Status Owner Pages Size
Untitled - Notepad Error - Print... User 1 840 bytes/932 ...
Microsoft Word - Document1 User 1 2.90 KB
Microsoft Word - Document1 User 1 2.24 KB
Document User 1 400 bytes
Microsoft Word - Document1 User 1 2.90 KB
5 document(s) in queue

QUEUE - DEFINITION

- A queue is a data structure where data enters at one end (at the rear of a list) and is removed from the other end (the front of the list).
- Queues are used to store items in the order in which they occur.
- Queues are an example of a first-in, first-out (FIFO) data structure.
- Queues are used to order processes submitted to an operating system or a print spooler, and simulation applications use queues to model customers waiting in a line.
- Queues are used to prioritize operating system processes and to simulate events in the real world, such as teller lines at banks and the operation of elevators in buildings

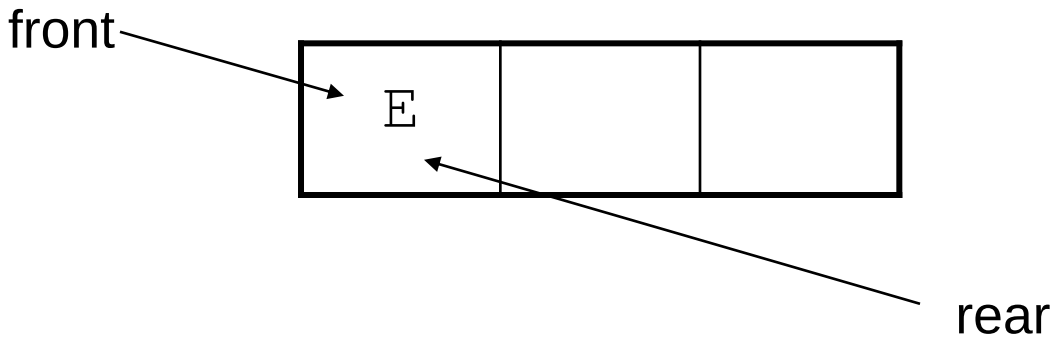
QUEUE ADT - OPERATIONS

- Primary operations
  - Enqueue operation: adding a new item at the end of the queue
  - Dequeue operation: removing an item from the front (or beginning) of the queue
  - Peek method: Viewing the beginning item
    - Simply returns the item without actually removing it from the queue.

QUEUE OPERATIONS - EXAMPLE

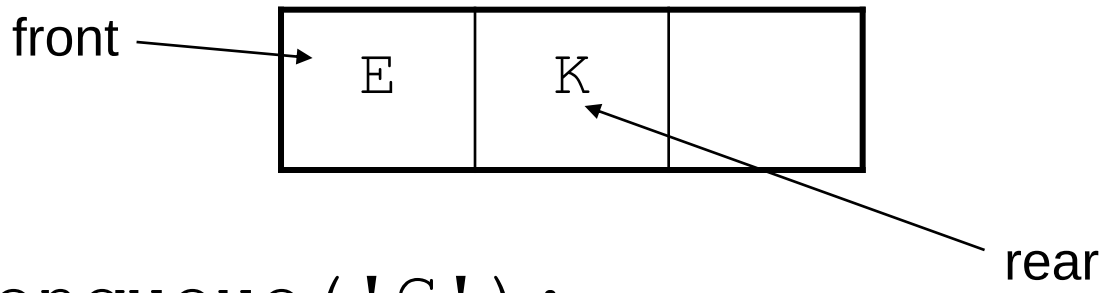
- **A currently empty queue that can hold `char` values:**

- `enqueue('E');`

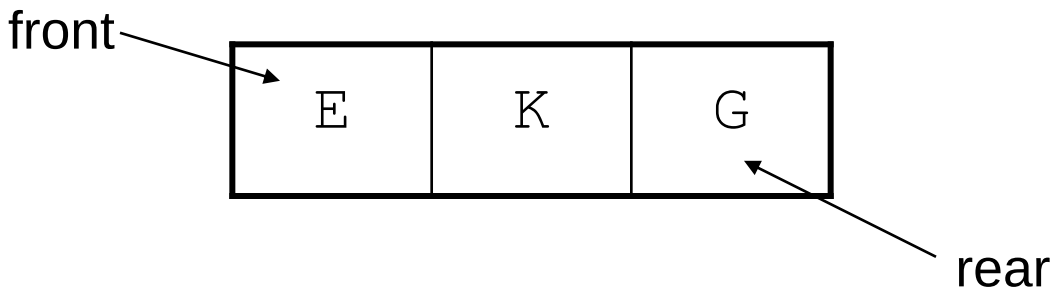


QUEUE OPERATIONS - EXAMPLE

- `enqueue('K');`

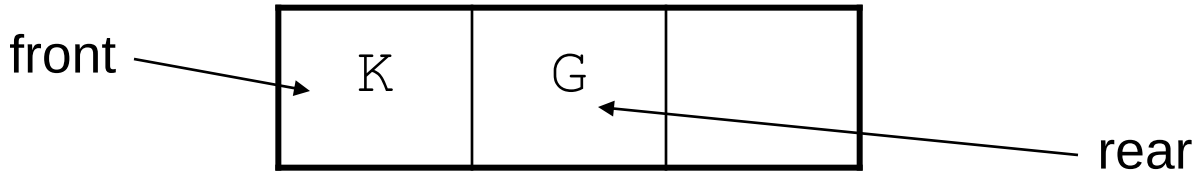


- `enqueue('G');`

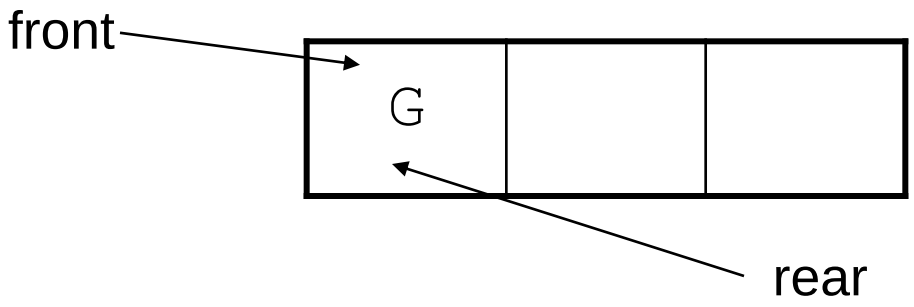


QUEUE OPERATIONS - EXAMPLE

- `dequeue(); // remove E`



- `dequeue(); // remove K`



QUEUE - IMPLEMENTATION

- It can be implemented in different ways
  - Static ← fixed size (Storage requirements known prior to execution)
    - Using arrays
  - Dynamic ←Grow and shrink as necessary
    - Using an ArrayList ←in **System.Collections**. We'll see them in the next module
    - Using linked lists

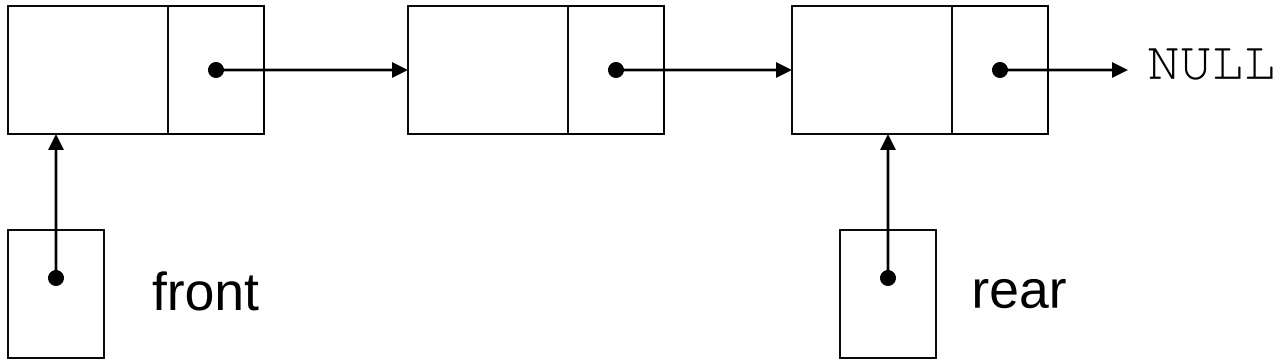
AN ARRAYLIST-BASED QUEUE IN C#

```
private ArrayList pqueue;

public CQueue()
{
    pqueue = new ArrayList();
}
public void EnQueue(object item)
{
    pqueue.Add(item);
}
public void DeQueue()
{
    pqueue.RemoveAt(0);
}
```

```
public object Peek()
{
    return pqueue[0];
}
public void ClearQueue()
{
    pqueue.Clear();
}

public int Count()
{
    return pqueue.Count;
}
```

DYNAMIC QUEUES

- Like a stack, a queue can be implemented using a linked list
- Allows dynamic sizing, avoids issue of shifting elements or wrapping indices



QUEUES – APPLICATIONS

- What does the following code fragment do to the queue **queue**?

```
Stack stack = new Stack();
while (!queue.isEmpty())
   stack.push(queue.dequeue());
while (!stack.isEmpty())
   queue.enqueue(stack.pop());
```

QUEUES – APPLICATIONS(2)

- What does the following code fragment do?

```
IntQueue q = new IntQueue();
q.enqueue(0);
q.enqueue(1);
for (int i = 0; i < 10; i++) {
    int a = q.dequeue();
    int b = q.dequeue();
    q.enqueue(b);
    q.enqueue(a + b);
    System.out.println(a);
}
```

QUEUES – APPLICATIONS(3)

- **Stack with one queue.** Show how to implement a stack using one queue.
  - *Hint:* to delete an item, get all of the elements on the queue one at a time, and put them at the end, except for the last one which you should delete and return.

TO DO - IF TIME … PRACTICE

- On the test I plan to include **insert** or **delete**. I also plan to ask you to write a method you have not seen in the class
  - Know lists (linked lists and array lists), stacks, queues
  - I'll post the midterm grades … they are just an estimation of your current grade (assuming you do the same on all tests & homeworks…)

- Singly or doubly linked list: should we store int or string values?
  - Reverse a list using a stack
  - Count the number of elements in a linked list

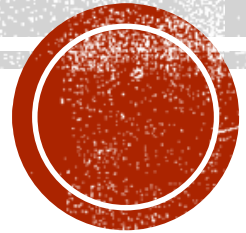
- **Doubly** linked list ←some of you made mistakes on big-oh on the hw - make sure to ask questions
  - void printReverse() void Reverse() ←also talk about running time for singly linked list using this link
  - bool IsPalindrome()
  - void RemoveDuplicates()
  - printReverseRecursive()

- **Stack with one queue.** Show how to implement a stack using one queue.
  - *Hint:* to delete an item, get all of the elements on the queue one at a time, and put them at the end, except for the last one which you should delete and return.

- If time
  - Merge two sorted linked list into one sorted one and what is the time and space complexity? [Glassdoor.com: Microsoft interview question]
  - if you have a linked list ordered like : n1-n2-n3-n4-n5-n6-n7-NULL.
    how to sort it to be at the order : n2-n1-n4-n3-n6-n5-n7-NULL [Glassdoor.com: Microsoft]
  - How to find the 3rd element from end, in a singly linked, in a single pass?

USING C# LIBRARY COLLECTIONS

Summer 2019 – CSC 395 – ST: Algorithm & Data Structure Concepts



RESOURCES

- https://docs.microsoft.com/en-us/dotnet/api/system.collections.queue?view=netframework-4.7.2
  - System.Collections
- https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic?view=netframework-4.7.2
  - System.Collections.Generic
    - Not on the test, but useful to know

MAIN TOPICS – SOME DATA STRUCTURES

System.Collections

- ArrayList
- ~~Hashtable~~
- ~~SortedList~~
- Stack
- Queue

System.Collections.Generic

- ~~Dictionary<TKey, TValue>~~
- ~~HashSet<T>~~
- ~~SortedDictionary<TKey, TValue>~~
- ~~SortedList<TKey, TValue>~~
- ~~SortedSet<T>~~
- ~~List<T>~~
- ~~Queue<T>~~
- ~~Stack<T>~~
- LinkedList<T>

ARRAYLIST

Namespace: System.Collections

Implements the IList interface using an array whose size is dynamically increased as required.

Some methods and properties:

- Capacity, Count
- Sort(), Add(),Clear(), Clone(), IndexOf(),…

```
/*
This code produces output similar to the following:

myAL
    Count:    3
    Capacity: 4
    Values:   Hello   World   !

*/
```

```
using System;
using System.Collections;
public class SamplesArrayList  {

   public static void Main()  {

      // Creates and initializes a new ArrayList.
      ArrayList myAL = new ArrayList();
      myAL.Add("Hello");
      myAL.Add("World");
      myAL.Add("!");

      // Displays the properties and values of the ArrayList.
      Console.WriteLine( "myAL" );
      Console.WriteLine( "    Count:    {0}", myAL.Count );
      Console.WriteLine( "    Capacity: {0}", myAL.Capacity );
      Console.Write( "    Values:" );
      PrintValues( myAL );
   }

   public static void PrintValues( IEnumerable myList )  {
      foreach ( Object obj in myList )
         Console.Write( "   {0}", obj );
      Console.WriteLine();
   }

}
```

STACK

Namespace: System.Collections

Represents a simple last-in-first-out (LIFO) non-generic collection of objects.

Some methods and properties:

- Count
- Clear(), Clone(), Peek(), Pop(), Push(),…

```
/*
This code produces the following output.

myStack
    Count:    3
    Values:    !    World    Hello
*/
```

```
using System;
using System.Collections;
public class SamplesStack  {

   public static void Main()  {

      // Creates and initializes a new Stack.
      Stack myStack = new Stack();
      myStack.Push("Hello");
      myStack.Push("World");
      myStack.Push("!");

      // Displays the properties and values of the Stack.
      Console.WriteLine( "myStack" );
      Console.WriteLine( "\tCount:    {0}", myStack.Count );
      Console.Write( "\tValues:" );
      PrintValues( myStack );
   }

   public static void PrintValues( IEnumerable myCollection )  {
      foreach ( Object obj in myCollection )
         Console.Write( "    {0}", obj );
      Console.WriteLine();
   }

}
```

QUEUE

Namespace: System.Collections

Represents a first-in, first-out collection of objects.

Some methods and properties:

- Count
- Clear(), Clone(), Peek(), Enqueue(), Dequeue(),…

```
/*
This code produces the following output.

myQ
    Count:    3
    Values:    Hello    World    !
*/
```

```
using System;
using System.Collections;
public class SamplesQueue  {

   public static void Main()  {

      // Creates and initializes a new Queue.
      Queue myQ = new Queue();
      myQ.Enqueue("Hello");
      myQ.Enqueue("World");
      myQ.Enqueue("!");

      // Displays the properties and values of the Queue.
      Console.WriteLine( "myQ" );
      Console.WriteLine( "\tCount:    {0}", myQ.Count );
      Console.Write( "\tValues:" );
      PrintValues( myQ );
   }


   public static void PrintValues( IEnumerable myCollection )  {
      foreach ( Object obj in myCollection )
         Console.Write( "    {0}", obj );
      Console.WriteLine();
   }
}
```

GENERICS

LINKED LIST – GENERICS

Namespace: System.Collections.Generic

Represents a doubly linked list.

Properties:

- Count, First, Last

Methods:

- AddFirst, AddLast, Clear, Contains, Find, Remove, RemoveFirst, RemoveLast

```
// Create the link list.
string[] words =
    { "the", "fox", "jumps", "over", "the", "dog" };
LinkedList<string> sentence = new LinkedList<string>(words);
Display(sentence, "The linked list values:");
Console.WriteLine("sentence.Contains(\"jumps\") = {0}",
    sentence.Contains("jumps"));

// Add the word 'today' to the beginning of the linked list.
sentence.AddFirst("today");
Display(sentence, "Test 1: Add 'today' to beginning of the list:");

// Move the first node to be the last node.
LinkedListNode<string> mark1 = sentence.First;
sentence.RemoveFirst();
sentence.AddLast(mark1);
Display(sentence, "Test 2: Move first node to be last node:");

// Change the last node be 'yesterday'.
sentence.RemoveLast();
sentence.AddLast("yesterday");
Display(sentence, "Test 3: Change the last node to 'yesterday':");

// Move the last node to be the first node.
mark1 = sentence.Last;
sentence.RemoveLast();
sentence.AddFirst(mark1);
Display(sentence, "Test 4: Move last node to be first node:");
```

QUEUE – GENERICS

Namespace: System.Collections.Generic

Represents a variable size last-in-first-out (LIFO) collection of instances of the same specified type.

Properties:

- Count

Methods:

- Clear, Peek, Pop, Push, ToString,ToArray

```
Stack<string> numbers = new Stack<string>();
numbers.Push("one");
numbers.Push("two");
numbers.Push("three");
numbers.Push("four");
numbers.Push("five");

// A stack can be enumerated without disturbing its contents.
foreach( string number in numbers )
{
    Console.WriteLine(number);
}

Console.WriteLine("\nPopping '{0}'", numbers.Pop());
Console.WriteLine("Peek at next item to destack: {0}",
    numbers.Peek());
Console.WriteLine("Popping '{0}'", numbers.Pop());

// Create a copy of the stack, using the ToArray method and the
// constructor that accepts an IEnumerable<T>.
Stack<string> stack2 = new Stack<string>(numbers.ToArray());

Console.WriteLine("\nContents of the first copy:");
foreach( string number in stack2 )
{
    Console.WriteLine(number);
}

// Create an array twice the size of the stack and copy the
// elements of the stack, starting at the middle of the
// array.
string[] array2 = new string[numbers.Count * 2];
numbers.CopyTo(array2, numbers.Count);

// Create a second stack, using the constructor that accepts an
// IEnumerable(Of T).
Stack<string> stack3 = new Stack<string>(array2);
```

QUEUE – GENERICS

Namespace: System.Collections.Generic

Represents a first-in, first-out collection of objects

Properties:

- Count

Methods:

- Clear, Dequeue, Enqueue, Peek, ToString, ToArray

```
Queue<string> numbers = new Queue<string>();
numbers.Enqueue("one");
numbers.Enqueue("two");
numbers.Enqueue("three");
numbers.Enqueue("four");
numbers.Enqueue("five");

// A queue can be enumerated without disturbing its contents.
foreach( string number in numbers )
{
    Console.WriteLine(number);
}

Console.WriteLine("\nDequeuing '{0}'", numbers.Dequeue());
Console.WriteLine("Peek at next item to dequeue: {0}",
    numbers.Peek());
Console.WriteLine("Dequeuing '{0}'", numbers.Dequeue());

// Create a copy of the queue, using the ToArray method and the
// constructor that accepts an IEnumerable<T>.
Queue<string> queueCopy = new Queue<string>(numbers.ToArray());

Console.WriteLine("\nContents of the first copy:");
foreach( string number in queueCopy )
{
    Console.WriteLine(number);
}

// Create an array twice the size of the queue and copy the
// elements of the queue, starting at the middle of the
// array.
string[] array2 = new string[numbers.Count * 2];
numbers.CopyTo(array2, numbers.Count);
```

OTHER SUGGESTED READING

- https://www.infoworld.com/article/3023041/application-development/my-two-cents-on-collections-in-c.html

C# LIBRARY

- ArrayList -- this implements the IList interface and represents a collection whole size can increase dynamically and can store objects of any type. Data inside an ArrayList instance can be accessed using the index.
- Hashtable -- this represents a collection that can store objects as key / value pairs
- BitArray -- this represents a collection that can manage a compact array of bit values where the values are represented as Booleans
- Queue -- this represents a first-in first-out non-generic collection of objects
- Stack -- this represents a last-in last-out non-generic collection of objects
- SortedList -- this represents a collection of objects stored as key / value pairs that are sorted by the keys. Objects in a sorted list are accessible both using their keys and indexes.
- https://www.infoworld.com/article/3023041/application-development/my-two-cents-on-collections-in-c.html

REAL WORLD APPLICATIONS

- Stacks are used for the undo buttons in various applications. The recent most changes are pushed into the stack.
  - Even the back button on the browser works with the help of the stack where all the recently visited web pages are pushed into the stack.
  - Also for functions calling other functions … discussion on execution stack.
  - An important application of stacks is in parsing. For example, a compiler must parse arithmetic expressions written using infix notation.

- Queues are used in case of printers or uploading images. Where the first one to be entered is the first to be processed.
  - Queue is useful in CPU scheduling, Disk Scheduling.
  - Handling of interrupts in real-time systems. The interrupts are handled in the same order as they arrive, First come first served.
  - When data is transferred asynchronously between two processes. Queue is used for synchronization. Examples : IO Buffers, pipes, file IO, etc.

INTERVIEW QUESTIONS (JUST FOR PRACTICE!)

- What is a linked list?
- What is LIFO?
- What is a queue?
- What is a stack? What operations can be performed on stacks?
- What is FIFO?
- What is the primary advantage of a linked list?
- What is the difference between a PUSH and a POP?
- Differentiate STACK from ARRAY.
- What are doubly linked lists?
- How do you search for a target key in a linked list?
- What are the various operations that can be performed on different Data Structures?
- What is Stack and where it can be used?
- What is a Queue, how it is different from stack and how is it implemented?

INTERVIEW PROBLEMS (JUST FOR PRACTICE!)

- Check for balanced parentheses in an expression (hint: use a stack)
- Reverse a String using Stack
- Polish notation … + X Y Operators are written before their operands… / * A + B C D
- How to implement a stack using queue?
- How to implement a queue using two stacks? [Glassdoor.com: Microsoft interview questions]
- Print Nth node from the end of a Linked List
- middle of a given linked list (hint: multiple ways, one clever way is to use two pointers)
- Linkedlist Reversal
- Find node in Linkedlist
- Merge two sorted Linkedlist

INTERVIEW PROBLEMS (2) (JUST FOR PRACTICE!)

- Find nth to last element in Singly Linkedlist
- Delete nth element from headnode
- Reverse a Stack
- How to remove duplicates from a sorted linked list?
- How to find the 3rd element from end, in a singly linked, in a single pass?
- Ways to print a singly-linked list in reverse order. Advantages/disadvantages.
- Add every other element in a linked list
- Swap two adjacent node of a linked list. [Glassdoor.com: Microsoft interview question]
- Reverse a linked list. [Glassdoor.com: Microsoft interview question]
- Delete a node from a linkedlist [Glassdoor.com: Microsoft interview question]

INTERVIEW PROBLEMS (3) (JUST FOR PRACTICE!)

- Merge two sorted linked list into one sorted one and what is the time and space complexity? [Glassdoor.com: Microsoft interview question]
- if you have a linked list ordered like : n1-n2-n3-n4-n5-n6-n7-NULL.
  how to sort it to be at the order : n2-n1-n4-n3-n6-n5-n7-NULL [Glassdoor.com: Microsoft]

PROGRAMMING COMPETITION

- CCSC SE 2011:
  15 homework extra points available

Problem 7

Josephus

The Josephus problem is the following game. There are n people, numbered 1 to n, sitting in a circle. Starting at person 1, a hot potato is passed. After m passes, the person holding the hot potato is eliminated, the circle closes ranks, and the game continues with the person who was sitting after the eliminated person picking up the hot potato. The last remaining person wins.

The Josephus problem arose in first century A.D., in a cave on a mountain in Israel, where Jewish zealots were besieged by Roman soldiers. The zealots voted to form a suicide pact rather than surrender to the Romans. Josephus suggested the game, and rigged the game so he would get the last lot. That is how we know about this game; in effect Josephus cheated!

Write a program that takes as input the number of people in the circle, n, and the number of passes, m. Display the sequence of people removed. The final person printed is deemed the winner. Note that person 1 is always the designated first person, and starts with the hot potato. You may assume the number of passes remains constant for each test case, and once person number k is eliminated, then the next round begins with person $(k+1)$ holding the hot potato.

Input

The first line of input is an integer T, $(2 \leq T \leq 20)$, representing the number of test cases. This is followed by T lines, each containing a test case with two integers separated by a single space. The first integer represents n, the number of people where $(2 \leq n \leq 40)$, and the second represents m, the number of passes where $(0 \leq m \leq 50)$.

Output

Your program should print the sequence of n people removed. Each person should be separated by a single space.

Sample Input

```
5
5 0
5 2
7 3
10 10
6 8
```

Output Corresponding to Sample Input

```
1 2 3 4 5
3 1 5 2 4
4 1 6 5 7 3 2
1 3 6 10 8 9 5 2 4 7
3 1 2 6 4 5
```

HOMEWORK FOR MODULE 3 – PART 1

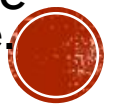
```
// Linked list operations
void appendNode(double);
void insertNode(double);
void deleteNode(double);
void displayList();
void isEmpty();
void addFront();
void addBack();
void reverseList();
void deleteList();
void sort();
```

DUE: see moodle, 11:59 pm. **Make sure to include the big-Oh for each method!**

- see below for part 2, due see moodle!

Write **separate** C# programs for each problem below. Make sure to implement all methods shown here:

- **Problem 1 [30 points]:** What we saw in class was a singly linked list, because there was a link from one node to the next node. **Modify** the code in class (**make sure to use it!!!**) so it works with strings instead of integers.
- **Problem 2 [30 points]:** What we saw in class was a singly linked list, because there was a link from one node to the next node. **Modify** the code in class (**make sure to use it!!!**) so it works with integers (as we saw in class) but so it uses two links for each node, one for the next and one for the previous node. This is called doubly linked list. In addition to the methods in class also add a method void printReverse() that prints in reverse the list (it displays the last element, then the one next to last, …, and lastly it will display the first element)
- **Problem 3 [20 points]** To the code written for Problem 1 add a new method **void RemoveDuplicates()** that will remove all duplicates from the linked list. Do not use an extra array, use only O(1) extra memory for this operation. What is the running time?
- **Problem 4 [10 points]** To the code written for Problem 1 add a new method **bool IsPalindrome()** that will check whether the **singly** linked list is a palindrome. If you don't remember what palindrome is google the term or stop by my office. What is the running time? E.g. "church" → "monk" → "monk" → "church" is a palindrome.
- **Problem 5 [10 points]** To the code written for Problem 2 add a new method **bool IsPalindrome()** that will check whether the **doubly** linked list is a palindrome. If you don't remember what palindrome is google the term or stop by my office. What is the running time? E.g. 21 <-> 34 <-> 6 <-> 34 <->21 is a palindrome.

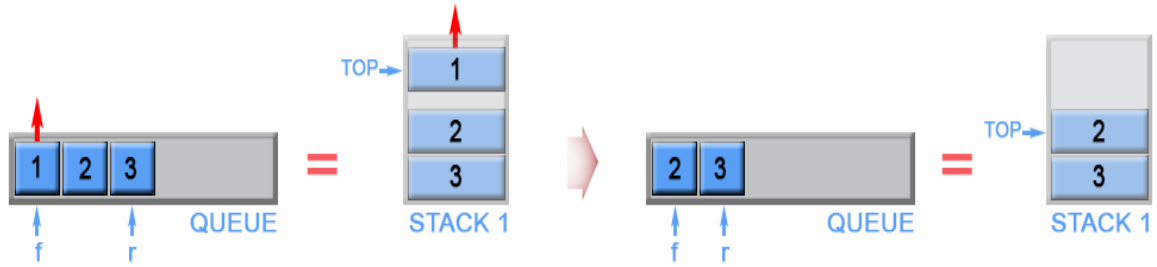


HOMEWORK FOR MODULE 3 – PART 2

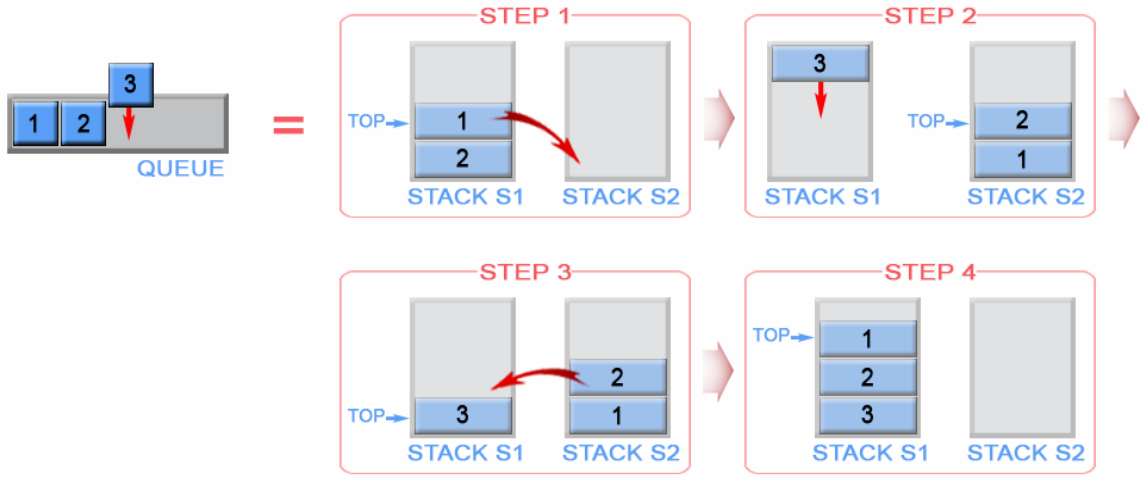
DUE: see moodle, 11:59 pm

Write separate C# programs for each problem below

- **Problem 1 [25 points]** Let Q be a non-empty queue, and let S be an empty stack. Write a C# program that reverses the order of the elements in Q, using S.
  - For example, if initially the order of the objects in Q is 1,2,3,4,5,6, then after reversing the objects, the order of the objects in Q is 6,5,4,3,2,1.
- **Problem 2 [25 points]** Write a program that opens a text file ("input.txt") and reads its contents. Then using a stack it reverses the lines of the file and saves them into another file ("output.txt"). Hint: use **System.IO.File.WriteAllLines** and **System.IO.File.ReadAllLines**,
- **Problem 3 [50 points]** Implement a Queue class using two stacks (use the side images as hints) What is the running time for enqueue() and dequeue()?



Popping element "1" from the queue



Pushing element "3" to the queue