CSIT881
Programming and Data Structures

Array and Linked List

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Dr. Joseph Tonien

Objectives

- Linear data structure
- Array
- Linked List

Linear data structure

A linear data structure is a collection of items whose order is sequential. It means that each item (except the last item) has a unique next item, and that each item (except the first item) has a unique previous item.

This sequential order is a **logical order** and it may not be reflected in a real physical order in computer memory.

Linear data structure

An Array is an abstract data type:

- a fixed-length sequence of items of the **same data type** that are stored contiguously in computer memory;
- items can be randomly accessed via its index;

7	3	8	5

Linear data structure

7	3	8	5

An array contains a fixed number of items stored sequentially in a computer memory.

If we want to increase the length of an array to store more items, we may need to *allocate new memory* for the whole new array and *copy* each item from the old array to the new array. We also need to free the memory of the old array.

- In low level programming language, memory management is a tricky business;
- In high level programming language (such as Java and Python), memory management is handled automatically behind the scene.

NumPy array

```
import numpy

help(numpy.ndarray)
```

```
Help on class ndarray in module numpy:


class ndarray(builtins.object)
 |  ndarray(shape, dtype=float, buffer=None,
 |          offset=0, strides=None, order=None)
 |
 |  An array object represents a multidimensional,
 |  homogeneous array of fixed-size items.
```

NumPy array

Creating a one-dimensional array:

```
import numpy

fiboArr = numpy.array([ 0, 1, 1, 2, 3, 5, 8, 13])

for i in range(0, len(fiboArr)):
    print(fiboArr[i])
```

NumPy array

Creating a one-dimensional array:

```
import numpy

arr1 = numpy.zeros(shape=5, dtype=numpy.int64)
print(arr1)

arr2 = numpy.ones(shape=7, dtype=numpy.int64)
print(arr2)
```

```
[0 0 0 0 0]
```

```
[1 1 1 1 1 1 1]
```

some other types

```
numpy.bool, numpy.str, numpy.byte,
numpy.int32, numpy.int64, numpy.float32,
numpy.float64, numpy.complex, numpy.object,
...
```

NumPy array

Creating a 2-dimensional array:

```
import numpy

magicSquare = numpy.array([[ 8, 1, 6], [3, 5, 7], [4, 9, 2]])

print(magicSquare)
```

```
[[8 1 6]
 [3 5 7]
 [4 9 2]]
```

NumPy array

Creating a 2-dimensional array:

```
import numpy

arr3 = numpy.zeros(shape=( 2, 3), dtype=bool)
print(arr3)
```

```
[[False False False]
 [False False False]]
```

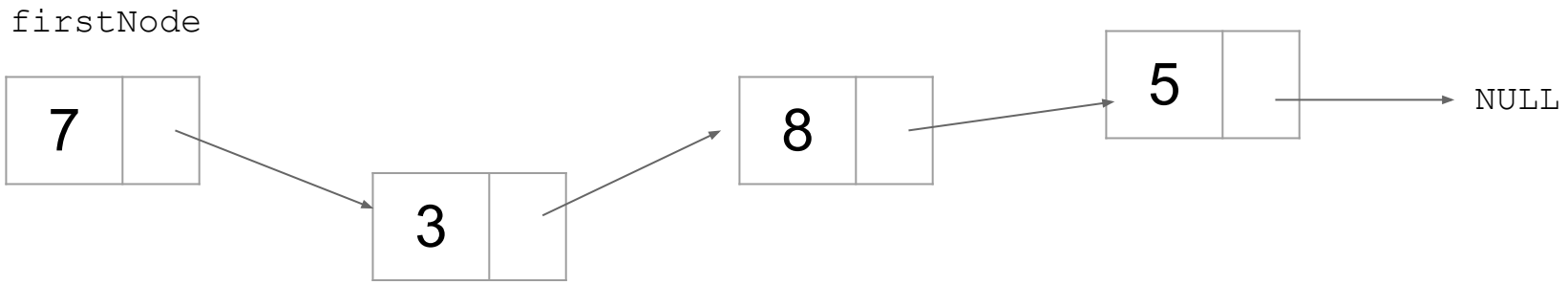
```
arr4 = numpy.ones(shape=( 3, 2), dtype=numpy.int32)
print(arr4)
```

```
[[1 1]
 [1 1]
 [1 1]]
```

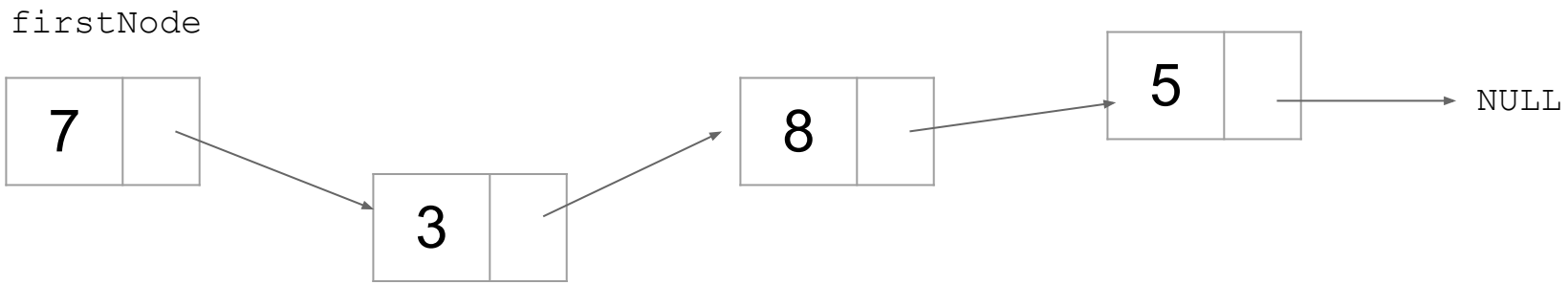
(ADT) Linked List

A linked list is an abstract data type:

- containing a sequence of data items whose **physical order** may not be sequential in computer memory;
- the data items may not be homogeneous, i.e. may have different sizes;
- its **logical order** is sequential by employing a collection of nodes where each node contains the data and **a reference to the next node**;



(ADT) Linked List



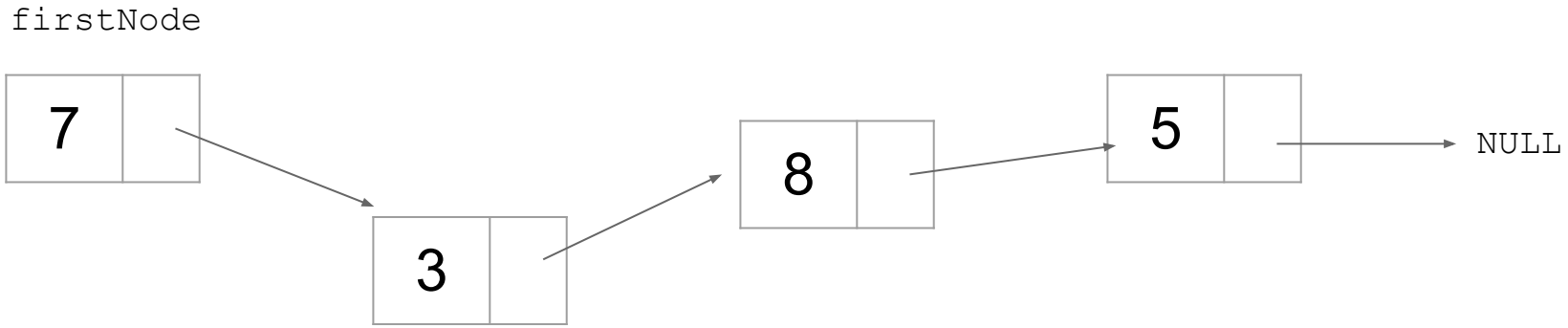
- each node contains the data and **a reference to the next node**;

- it is easy to add a new node in any position in the list and delete an old node, making the list grows dynamically;

(ADT) Linked List

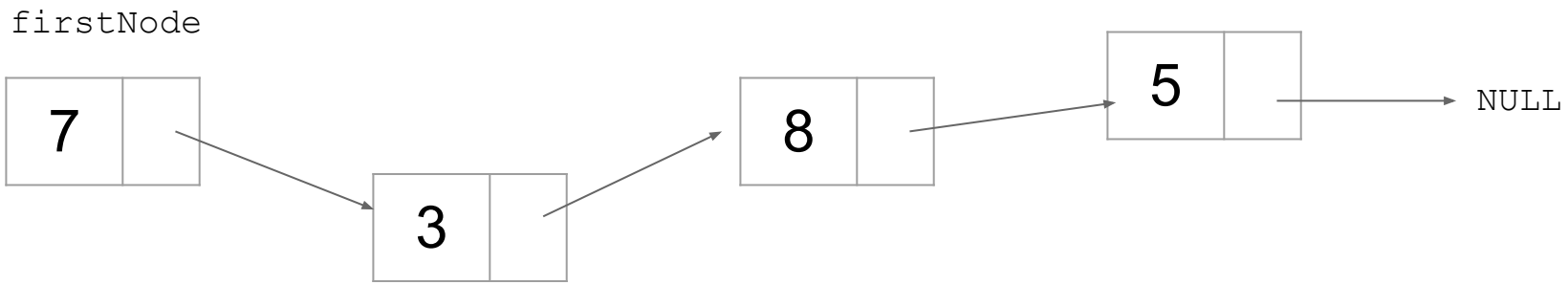
Comparison between Array and Linked List

- fixed length, homogeneous size, stored sequentially in memory;
- static structure, hard to extend memory to store more items.



- variable-length list, non-homogeneous size, items stored independently in memory;
- dynamic structure, easy to add and delete items.

(ADT) Linked List



Some common operations on linked list:

- Get first node
- Get last node
- Get the ith node
- Add an item to the start of the list
- Add an item to the end of the list
- Remove first node
- Search for an item...

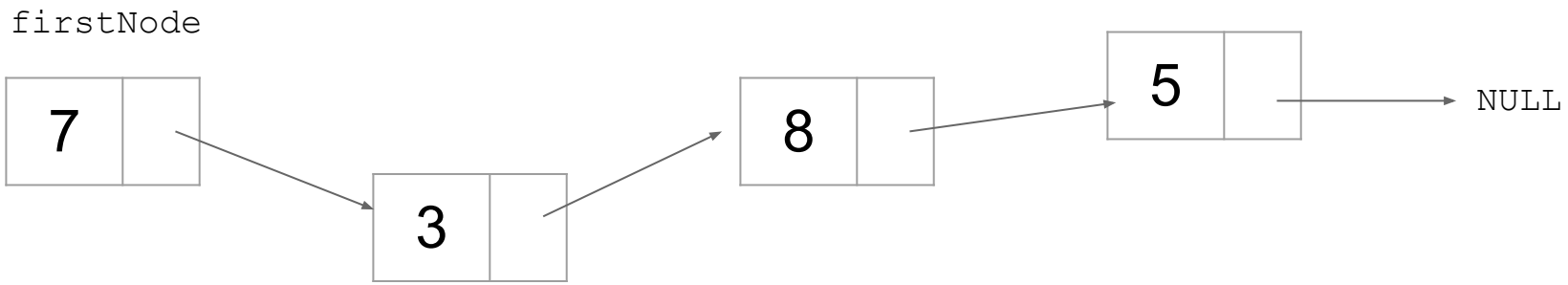
(ADT) Linked List

Some disadvantage:

- only the first node is readily accessible, access to other nodes need a linear travelling from the first node;
- easy to operate on the start of the list, but hard to operate at the end of the list because only reference to the first node is available;
- linear searching through a list.

Some programming language may add additional functionalities to their linked list implementation to have some of the above lacking behaviours.

Implementation of (ADT) Linked List

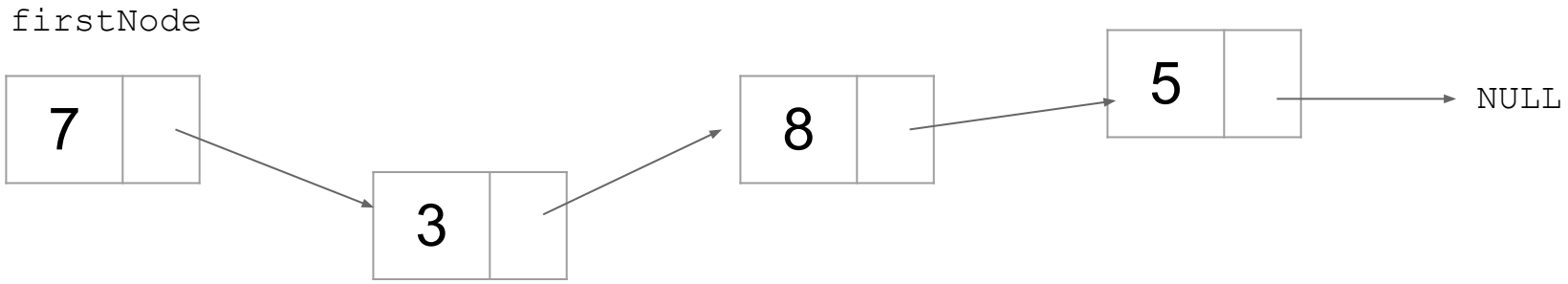


pseudocode

```
record Node
{
    datum           // the data being stored in the node
    Node next       // a reference to the next node
}
```

```
record LinkedList
{
    Node firstNode // first node of the list (null for empty list)
}
```

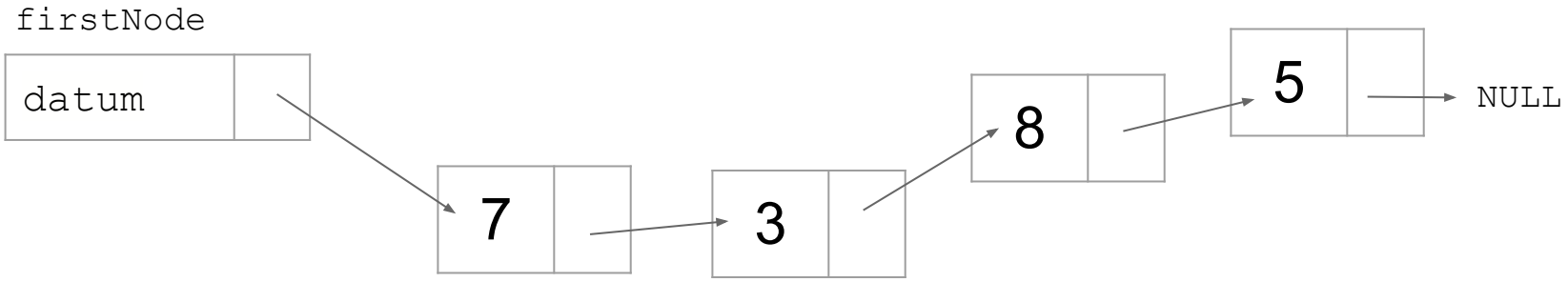
Implementation of (ADT) Linked List



pseudocode

```
Function constructor()
// Construct an empty linked list
{
    firstNode = NULL
}

Function getFirstNode()
// Returns the first node of the linked list.
// If the linked list is empty then returns NULL
{
    RETURN firstNode
}
```

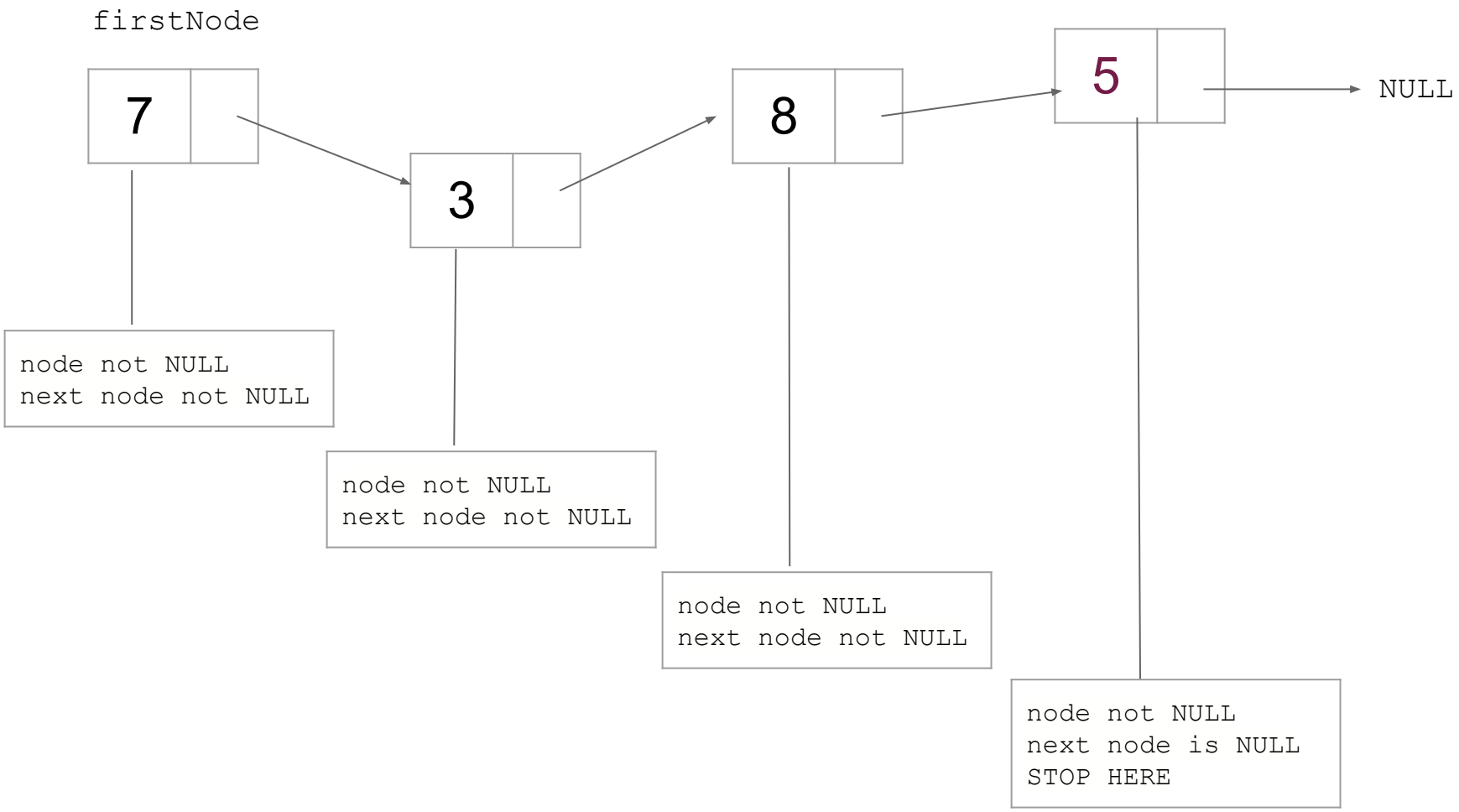
Implementation of (ADT) Linked List



pseudocode

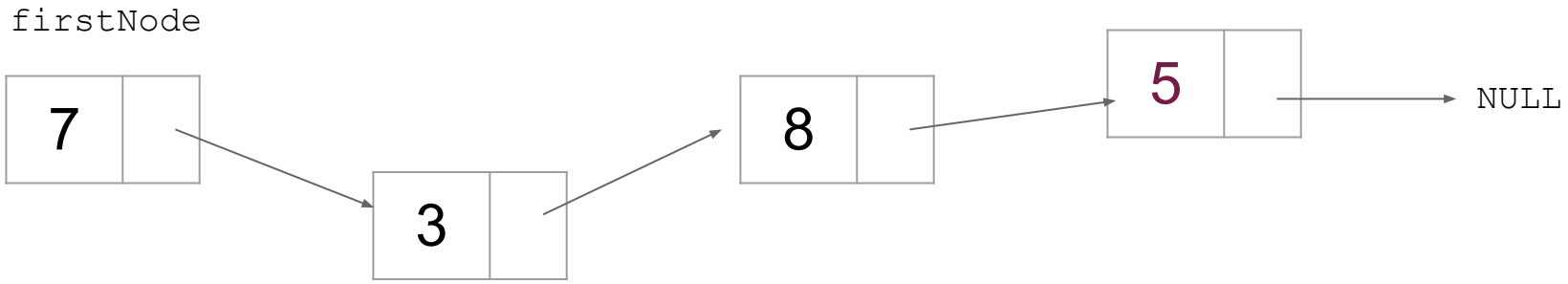
```
Function insertFirst(datum)
// Inserts a node holding the datum at the beginning of this list
{
    // Create a new node
    Node newNode = create Node
    newNode.datum = datum
    newNode.next = firstNode

    // Set the new node to be the first node of the list
    firstNode = newNode
}
```

Implementation of (ADT) Linked List




```
Function getLastNode()
// Returns the last node of the linked list.
// If the list is empty then returns NULL
```

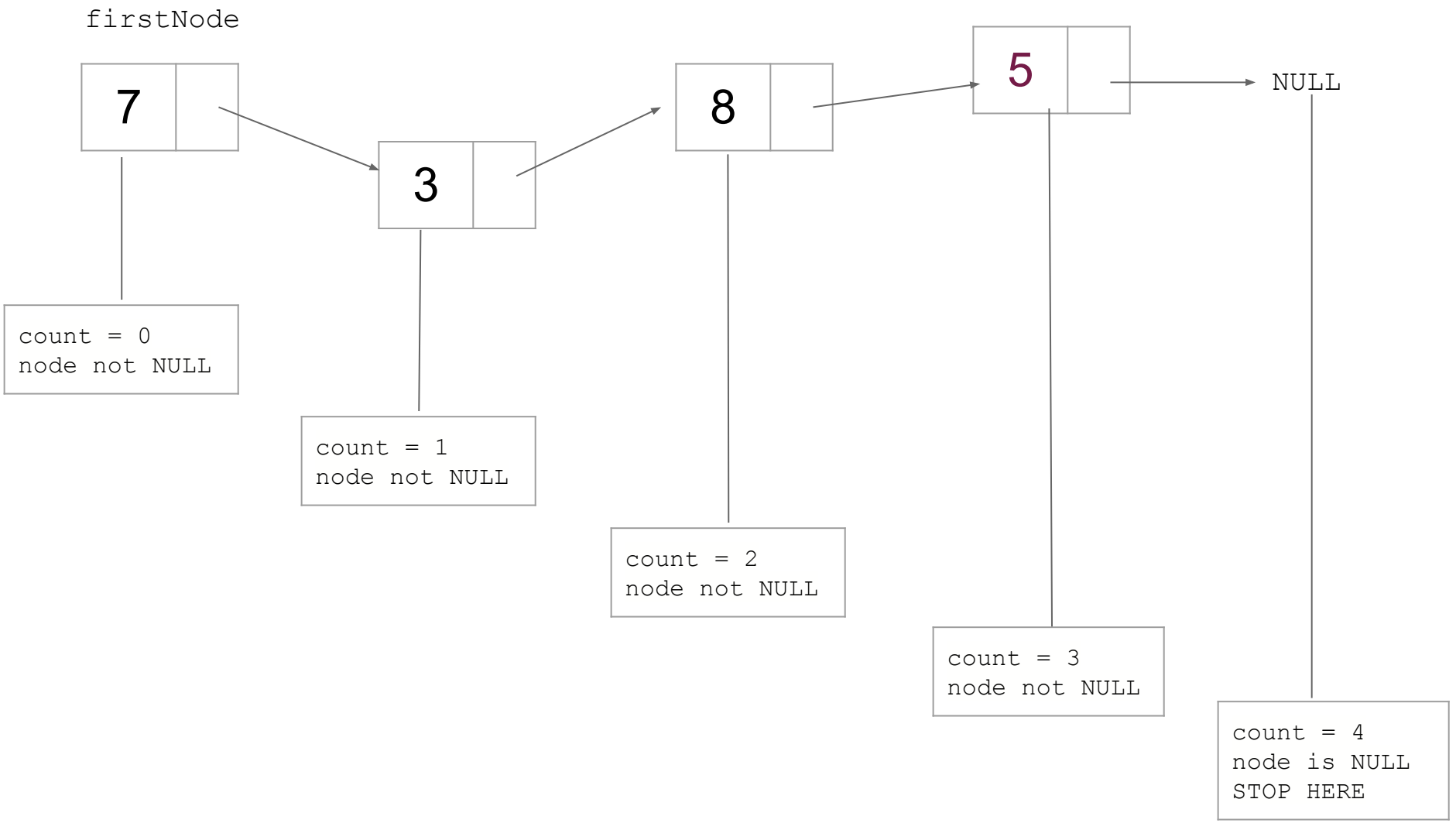
Implementation of (ADT) Linked List



pseudocode

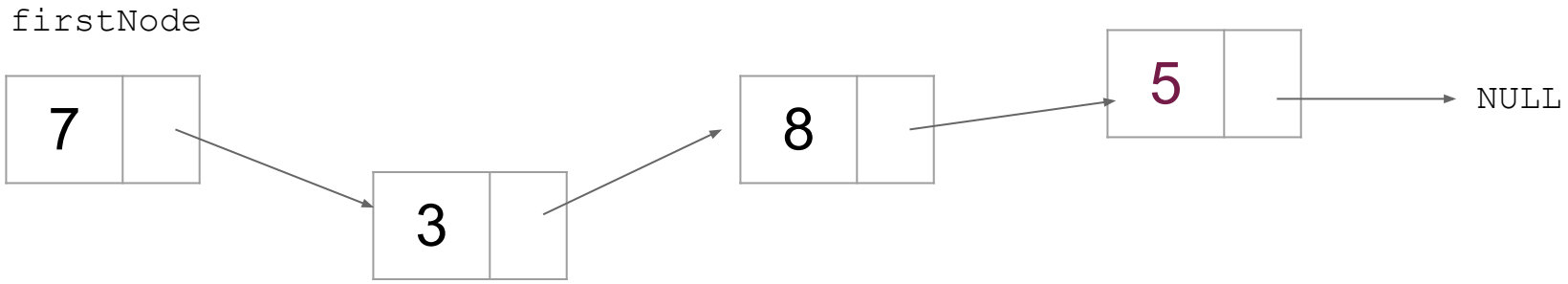
```
Function getLastNode()
// Returns the last node of the linked list.
// If the list is empty then returns NULL
{
    // start at the first node
    node = firstNode

    // travel down the list
    WHILE node != NULL and node.next != NULL
        node = node.next
    END WHILE

    RETURN node
}
```

Implementation of (ADT) Linked List




```
Function getLength()
// Returns the length of the linked list
```

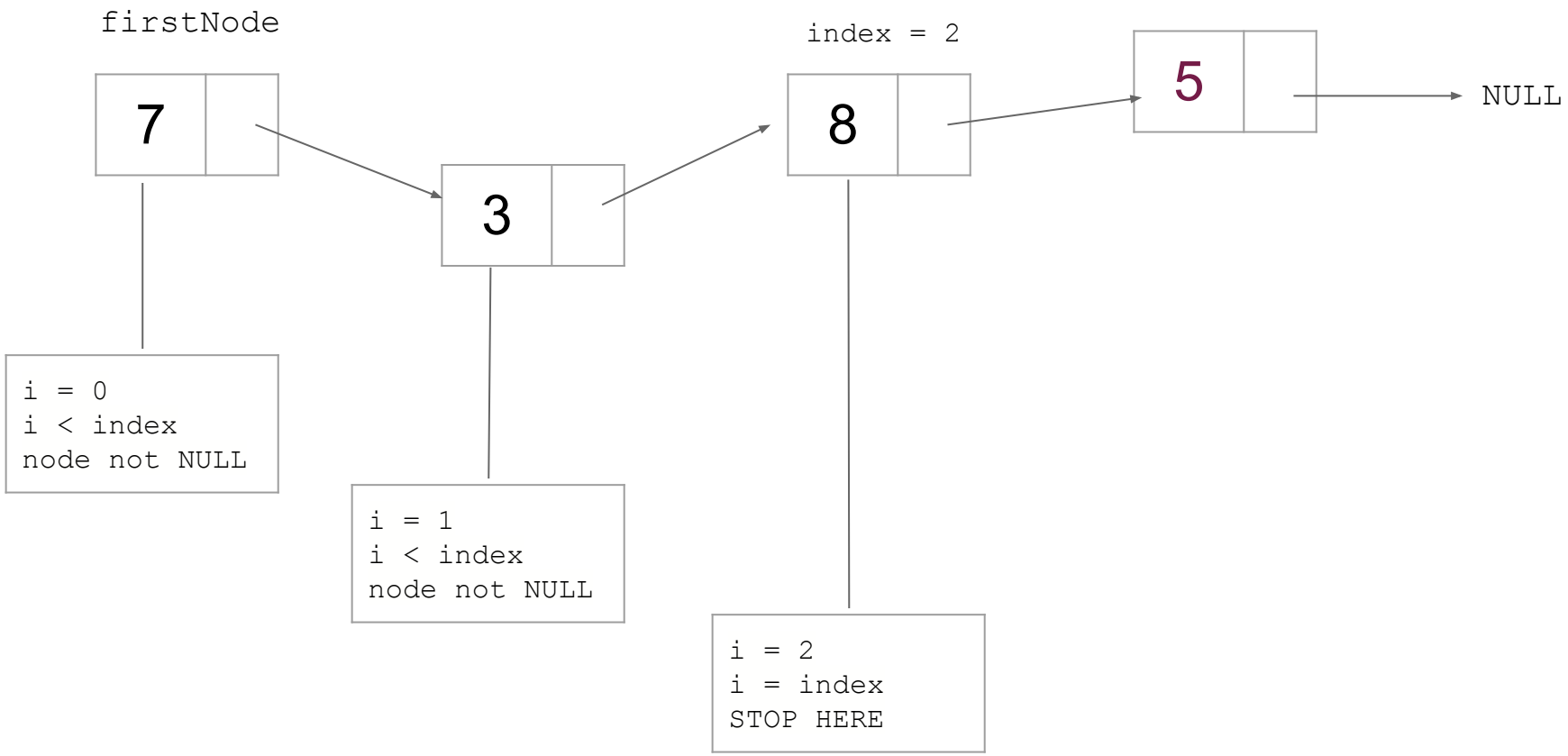
Implementation of (ADT) Linked List



pseudocode

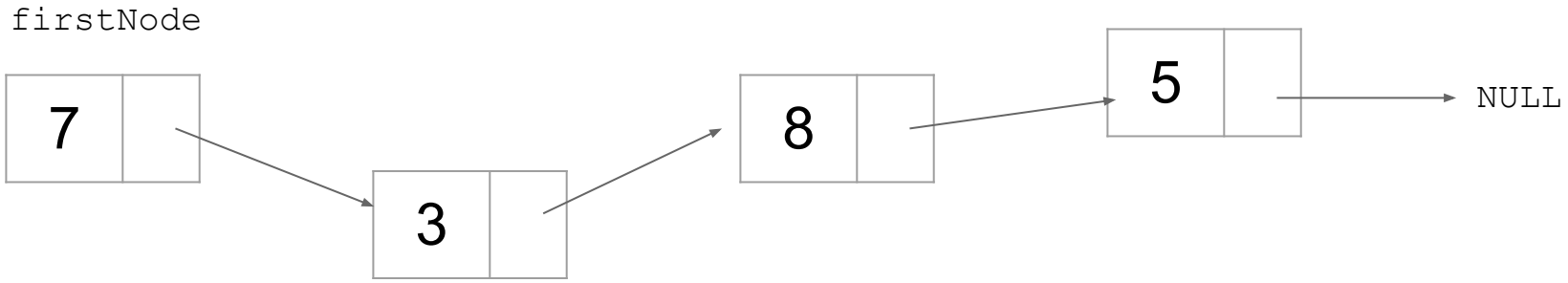
```
Function getLength()
// Returns the length of the linked list
{
    // counting number of nodes
    count = 0

    // start at the first node
    node = firstNode

    // travel down the list and increase the count
    WHILE node != NULL
        node = node.next
        count = count + 1
    END WHILE

    RETURN count
}
```

Implementation of (ADT) Linked List




```
Function getNode(index)
// Returns the node at the specified index in this list.
// Returns NULL if there is no node at that position.
// Index 0 refers to the first node.
```

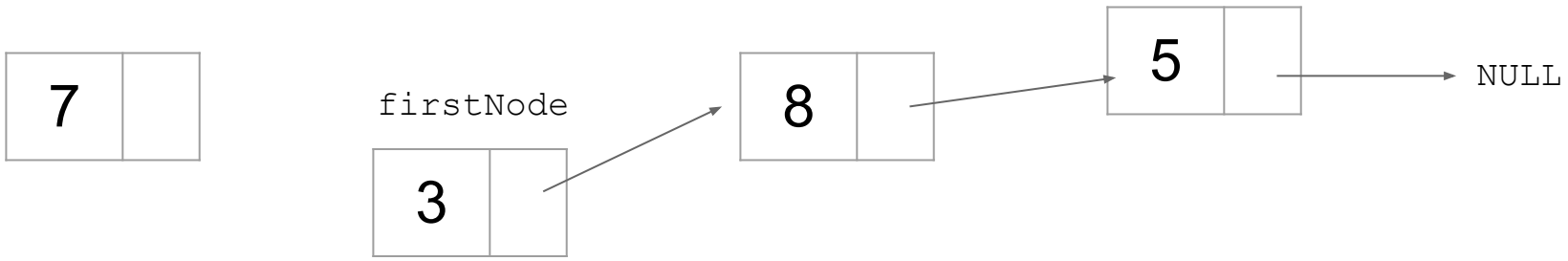
Implementation of (ADT) Linked List



pseudocode

```
Function getNode(index)
// Returns the node at the specified index in this list.
// Returns NULL if there is no node at that position.
// Index 0 refers to the first node.
{
    // start at the first node
    node = firstNode
    i = 0

    // travel down the list until the index
    WHILE i < index and node != NULL
        node = node.next
        i = i + 1
    END WHILE

    RETURN node
}
```

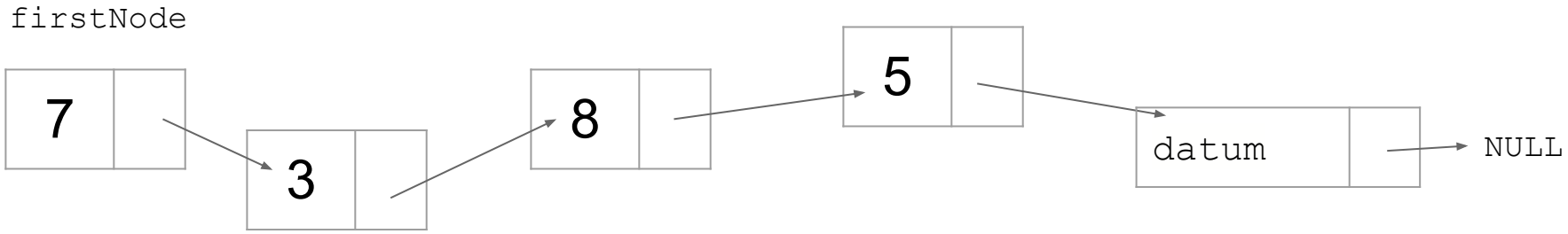
Implementation of (ADT) Linked List



pseudocode

```
Function removeFirst()
// Removes the first node and returns its datum.
// Returns NULL if the list is empty.
{
    // check empty list
    IF firstNode is NULL
        RETURN NULL

    // get the first datum
    firstDatum = firstNode.datum

    // reset the first node of the list
    firstNode = firstNode.next

    RETURN firstDatum
}
```

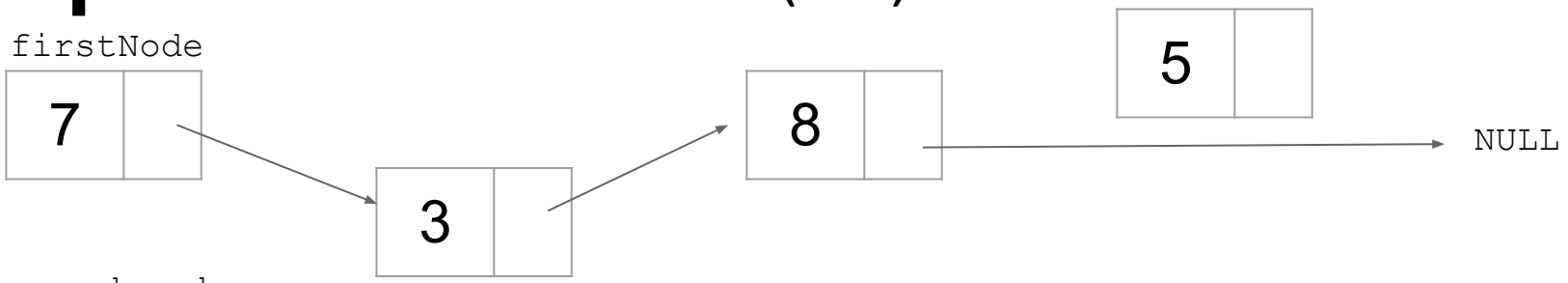
Implementation of (ADT) Linked List



pseudocode

```
Function insertLast(datum)
// Inserts a node holding the datum at the end of the list
{
    lastNode = getLastNode()

    IF lastNode is NULL
        // list is currently empty
        insertFirst(datum)
    ELSE
        // create a new node
        Node newNode = create Node
        newNode.datum = datum
        newNode.next = NULL
        // link the current last node to this new node
        lastNode.next = newNode
}
```

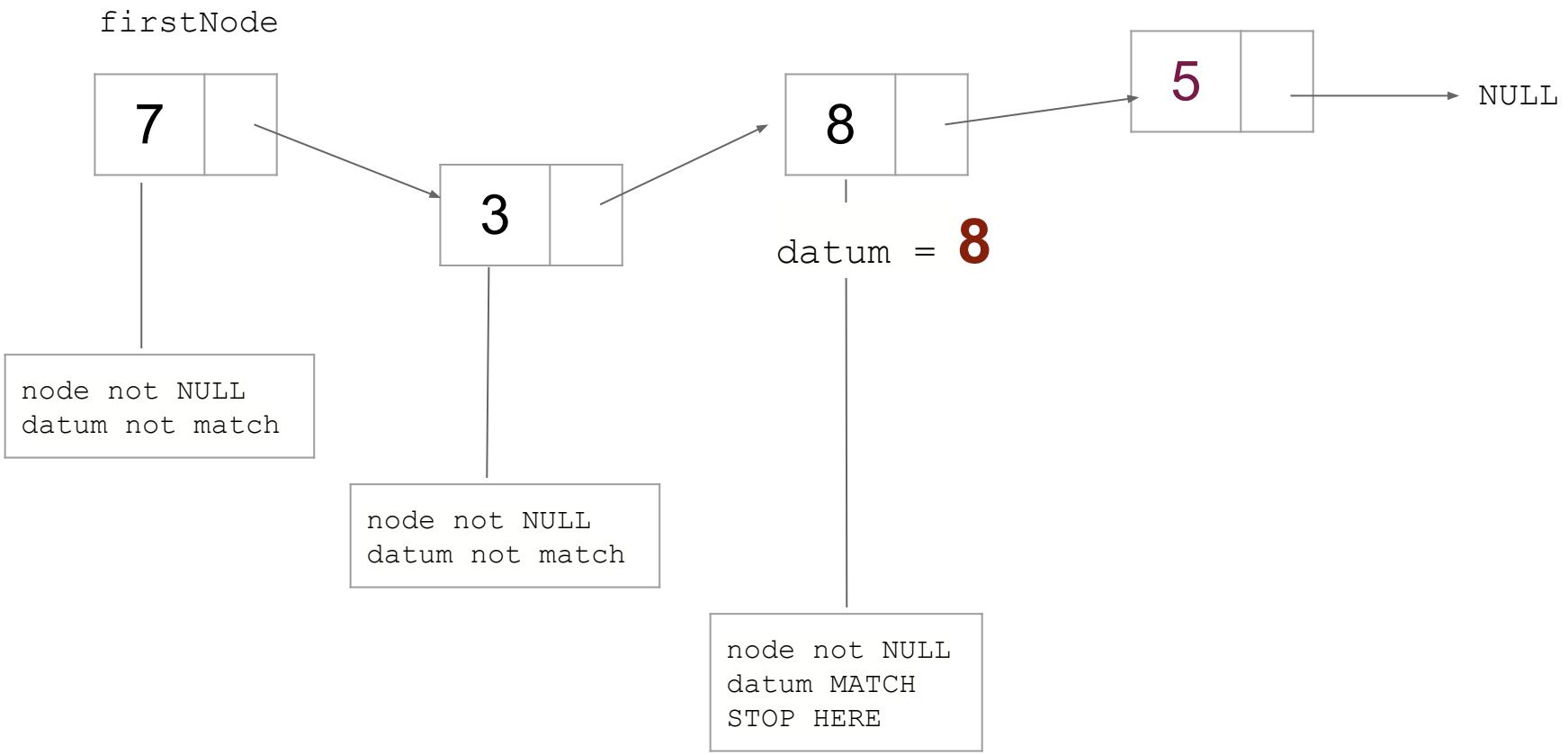
Implementation of (ADT) Linked List



pseudocode

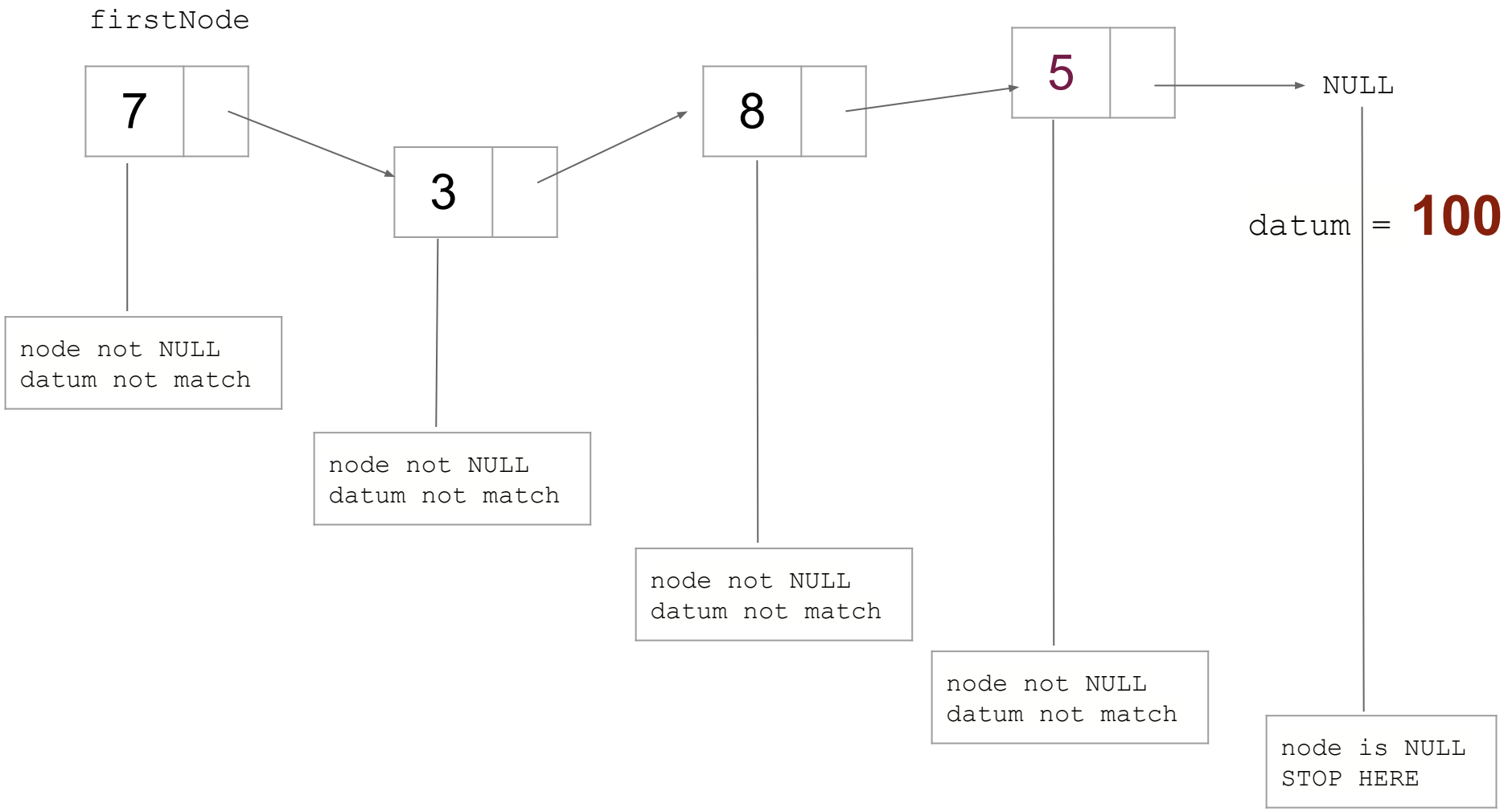
```
Function removeLast()
// Removes the last node of this list and returns its datum.
// Returns NULL if the list is empty.
{
    IF firstNode is NULL  // empty list case
        RETURN NULL
    END IF

    // find the last node and its previous node
    prevNode = NULL
    lastNode = firstNode
    WHILE lastNode != NULL and lastNode.next != NULL
        prevNode = lastNode
        lastNode = lastNode.next
    END WHILE

    IF prevNode is NULL
        // this list currently has one item
        firstNode = NULL
    ELSE
        prevNode.next = NULL
    END IF

    RETURN lastNode.datum
}
```

Implementation of (ADT) Linked List



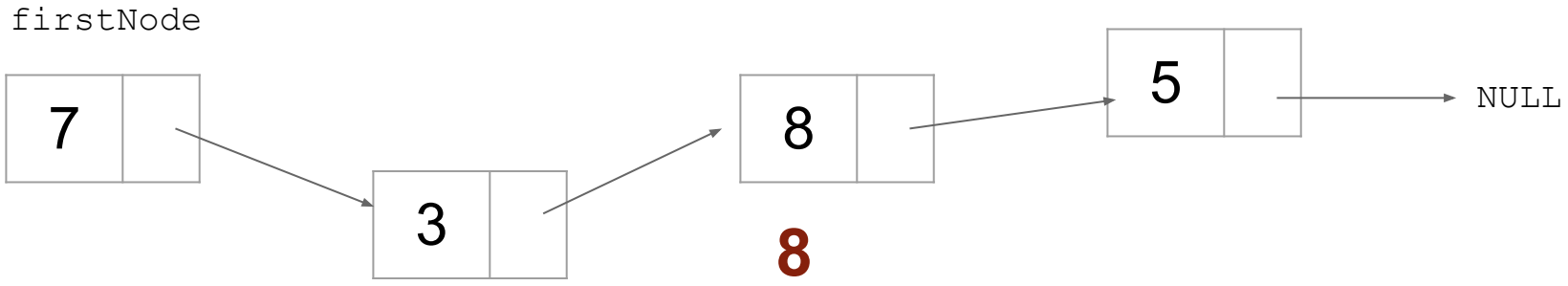

```
Function search(datum)
// Returns the node that has the datum matched
// Returns NULL if there is no match.
```

Implementation of (ADT) Linked List




```
Function search(datum)
// Returns the node that has the datum matched
// Returns NULL if there is no match.
```

Implementation of (ADT) Linked List



pseudocode

```
Function search(datum)
// Returns the node that has the datum matched
// Returns NULL if there is no match.
{
    // start at the first node
    node = firstNode

    // travel down the list until datum found
    WHILE node != NULL
        IF node.datum matches datum
            RETURN node
        END IF
        node = node.next
    END WHILE

    RETURN NULL
}
```

Implementation of (ADT) Linked List

Python implementation

```
class Node:
    """
    Representing a node consisting of
    - datum: the datum stored at the node
    - next: reference to the next node
    """

    def __init__(self, datum, next):
    #{
        self.datum = datum
        self.next = next
    #}
```

Implementation of (ADT) Linked List

Python implementation

```
class MyLinkedList:
    """
    Implementation of a linked list
    that connect firstNode -> node -> node -> ... -> NULL
    """

    def __init__(self):
    #{
        """
        Constructs an empty linked list
        """
        self.firstNode = None
    #}
```

Implementation of (ADT) Linked List

Python implementation

```
class MyLinkedList:

    def insertFirst(self, datum):
    #{
        """
        Inserts a node holding the datum at the beginning of this list.
        """

        # create a new node
        newNode = Node(datum=datum, next= self.firstNode)

        # set the new node to be the first node of the list
        self.firstNode = newNode
    #}
```

Implementation of (ADT) Linked List

Python implementation

```
class MyLinkedList:


    def getLastNode(self):
    #{
        """
        Returns the last node of this list.
        If the list is empty then returns None
        """

        # start at the first node
        node = self.firstNode


        # travel down the list
        while (node != None) and (node.next != None):
            node = node.next

        return node
    #}
```

Implementation of (ADT) Linked List

Python implementation

```
class MyLinkedList:


    def getLength(self):
    #{
        """
        Returns the length of the list.
        If the list is empty then returns 0.
        """

        # counting number of nodes
        count = 0

        # start at the first node
        node = self.firstNode

        # travel down the list and increase count
        while node != None:
        #{
            node = node.next
            count = count + 1
        #}

        return count
    #}
```

Implementation of (ADT) Linked List

Python implementation

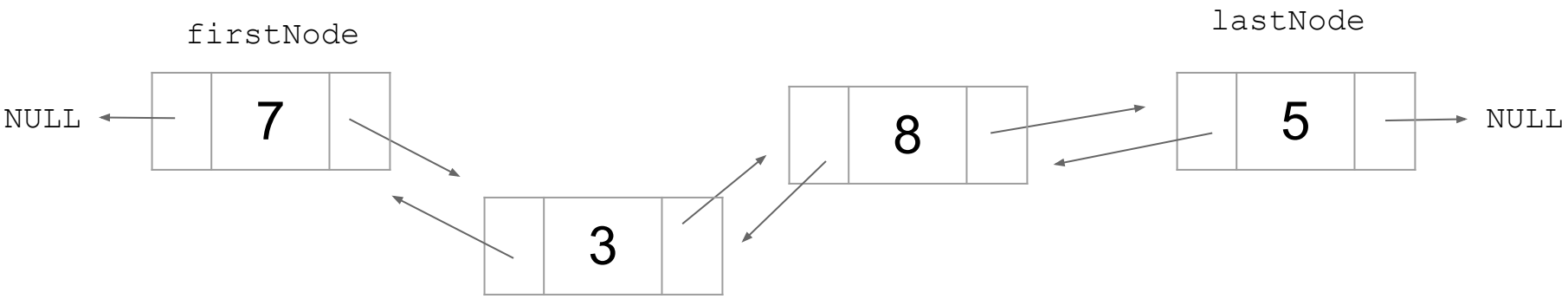
```
class MyLinkedList:


    def getNode(self, index):
    #{
        """
        Returns the node at the specified index in this list.
        Returns None if there is no node at that position.
        Index 0 refers to the first node.
        """

        # start at the first node
        node = self.firstNode
        i = 0

        # travel down the list until the index
        while (i < index) and (node != None):
        #{
            node = node.next
            i = i + 1
        #}

        return node
    #}
```

Implementation of (ADT) Linked List

Python implementation

```
class MyLinkedList:


    def removeFirst(self):
    #{
        """
        Removes the first node of this list and return its datum.
        Returns None if the list is empty.
        """

        # if the list is empty
        if (self.firstNode == None):
            return None

        # get the first datum
        firstDatum = self.firstNode.datum

        # reset the first node of this list
        self.firstNode = self.firstNode.next

        return firstDatum
    #}
```

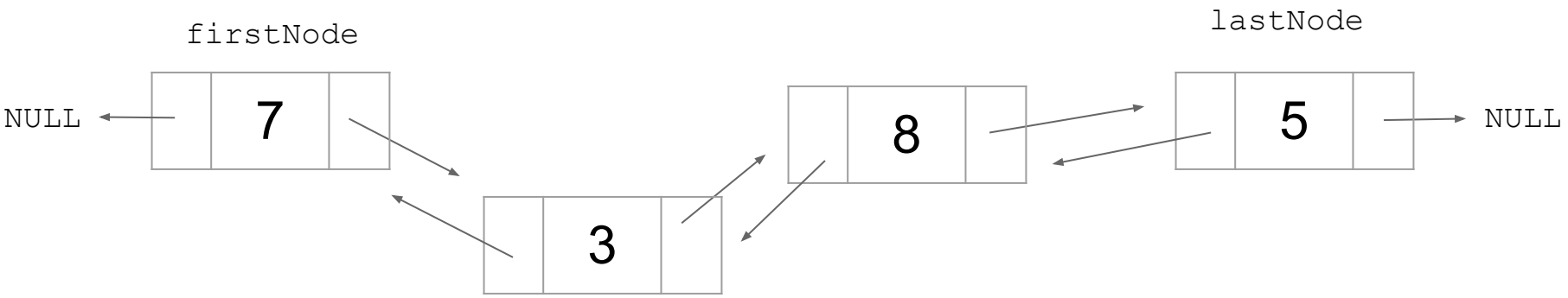
(ADT) Doubly Linked List



- each node contains the data and two node references:
  - one reference to the **next node**;
  - one reference to the **previous node**;

- a doubly linked list has two references: one to its **first node** and one to its **last node**.

(ADT) Doubly Linked List



pseudocode

```
record Node
{
    datum           // the data being stored in the node
    Node prev       // a reference to the previous node
    Node next       // a reference to the next node
}
```

```
record DoublyLinkedList
{
    Node firstNode // first node of the list (null for empty list)
    Node lastNode  // last node of the list (null for empty list)
}
```

References

- Python 3 documentation
  https://docs.python.org/3/

- NumPy Reference
  https://numpy.org/doc/stable/reference/