Laboratorio de Programación

2016 -2

Juan Camilo Rada[1]

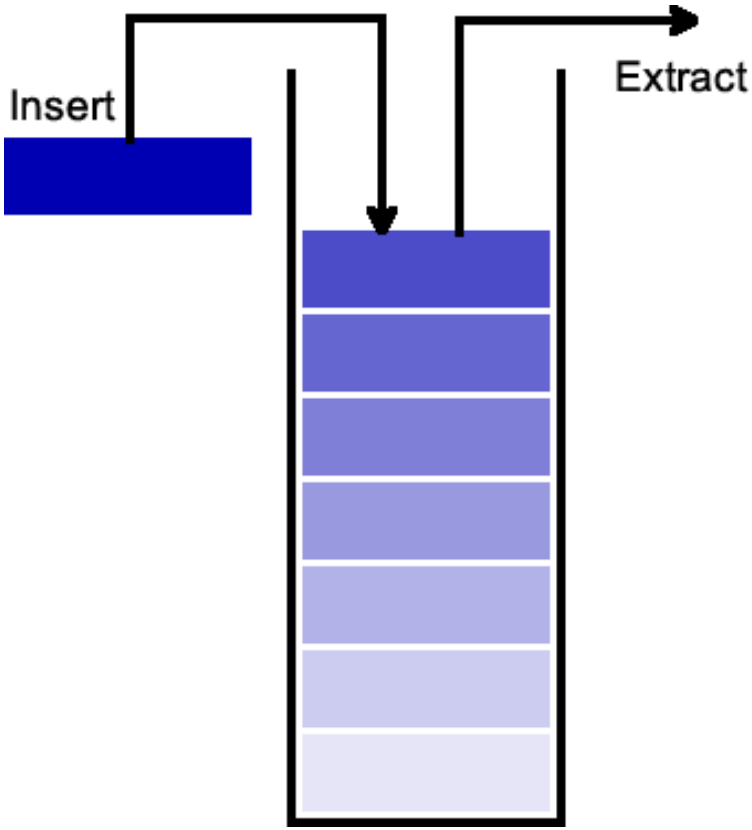
[1]Departamento de Electronica y ciencias de la computación

Septiembre de 2016

Outline

Outline

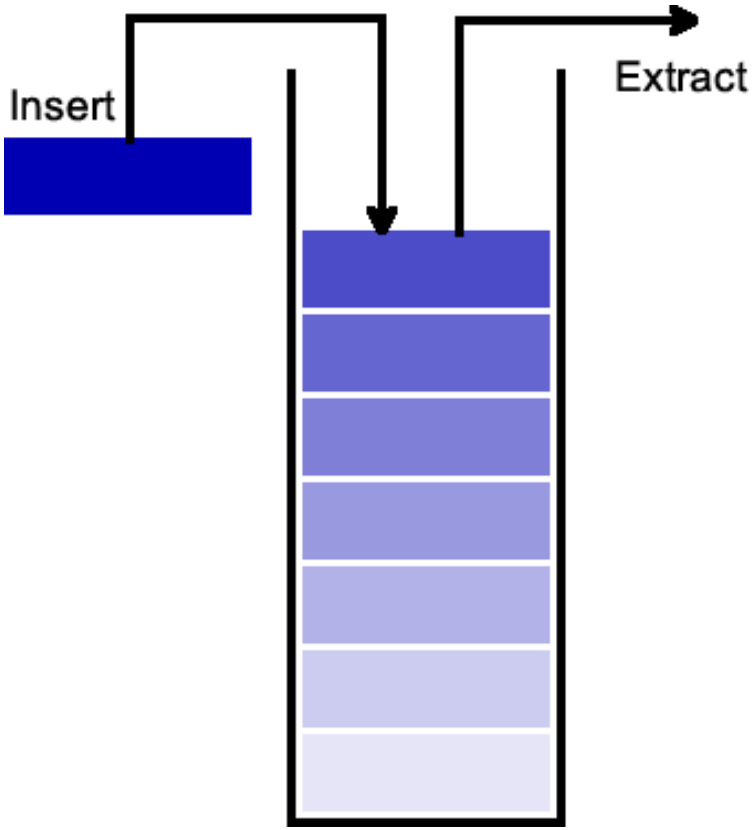
Stack



Definition

Is a type of `LIFO` list.

Stack

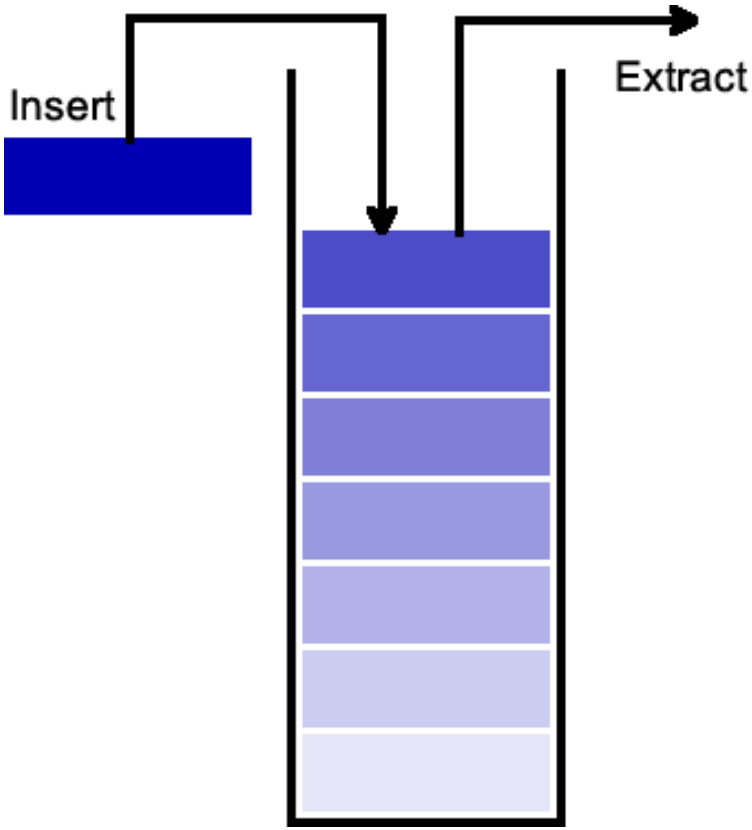


Definition

Is a type of `LIFO` list.

- *LIFO*: Last In First Out

Stack

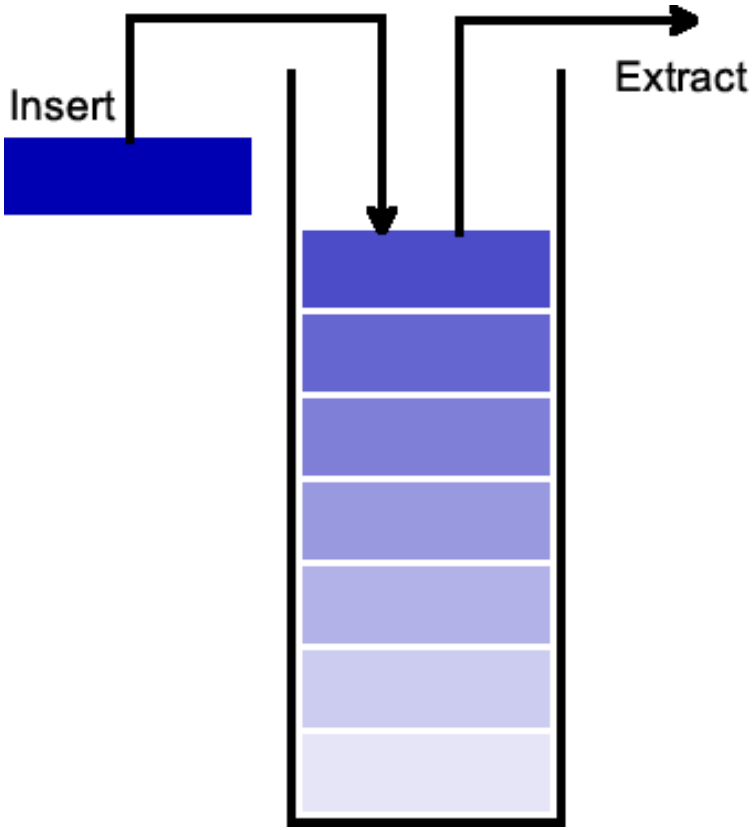


Definition

Is a type of `LIFO` list.

- *LIFO*: Last In First Out
- The *Last* element to In is the *First* to out

Stack

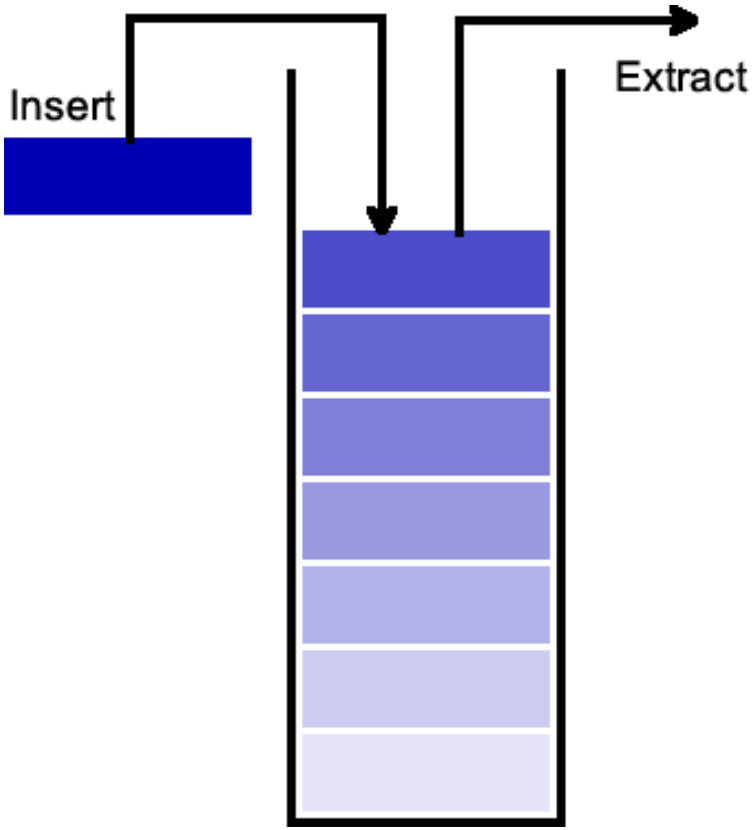


Definition

Is a type of `LIFO` list.

- *LIFO*: Last In First Out
- The *Last* element to In is the *First* to out
- Elements are "stacked" on top over each other

Stack

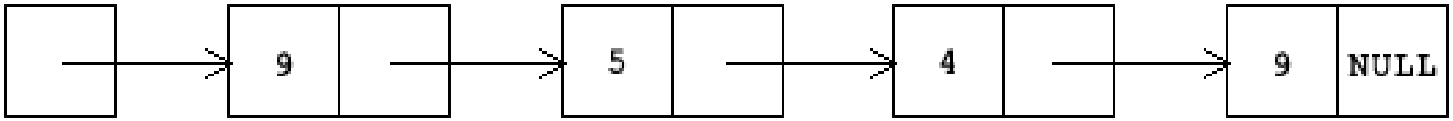


Definition

Is a type of `LIFO` list.

- *LIFO*: Last In First Out
- The *Last* element to In is the *First* to out
- Elements are "stacked" on top over each other
- We have to remove element on top to retrieve a value

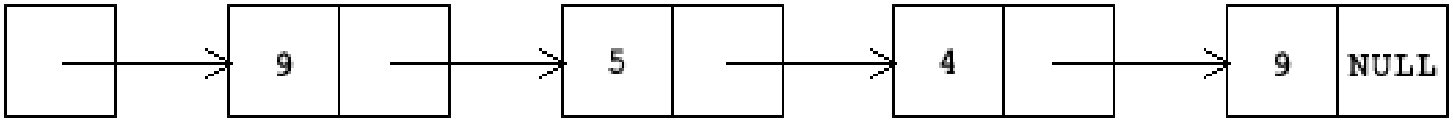
Stack



Definition

- We implement a dynamic stack using a `Linked List`

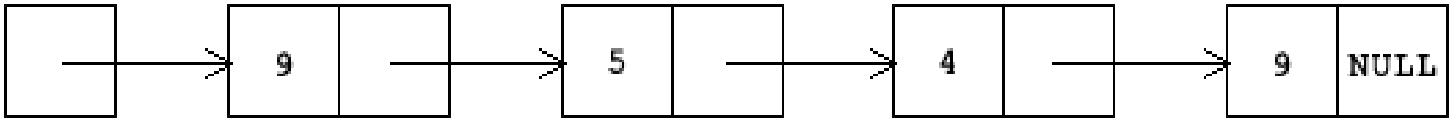
Stack



Definition

- We implement a dynamic stack using a `Linked List`
- We don't need to control the stack size (Except for memory limitations)

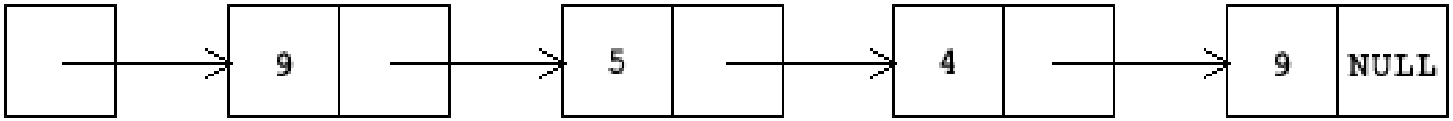
Stack



Definition

- We implement a dynamic stack using a `Linked List`
- We don't need to control the stack size (Except for memory limitations)
- New elements will be added at the top of the list

Stack



Definition

- We implement a dynamic stack using a `Linked List`
- We don't need to control the stack size (Except for memory limitations)
- New elements will be added at the top of the list
- We will have 3 basic operations ...

Outline

Stack

Operations

- `push`
  Input: A pointer to the stack, and the element to insert
  Do: Inserts the element at the top of the stack

Stack

Operations

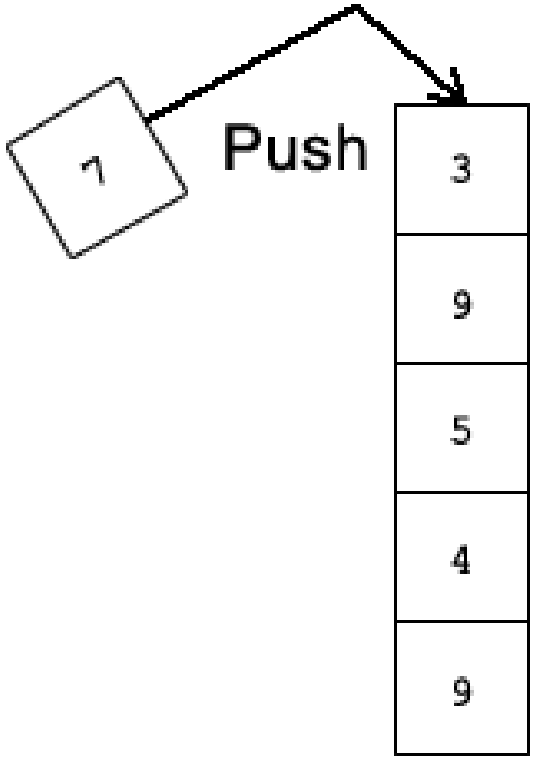
- `push`
  Input: A pointer to the stack, and the element to insert
  Do: Inserts the element at the top of the stack
- `pop`
  Input: A pointer to the stack
  Do: Return the value at the top and remove the element from the stack

Stack

Operations

- `push`
  Input: A pointer to the stack, and the element to insert
  Do: Inserts the element at the top of the stack
- `pop`
  Input: A pointer to the stack
  Do: Return the value at the top and remove the element from the stack
- `peek`
  Input: A pointer to the stack
  Do: Return the value at the top of the stack `0`

Stack - Push



Definition

- Inserts a new element at the top of the stack

Stack - Push

Algorithm

1. Let `ref`, the reference of the pointer at the head node of the Stack, and `value` the value to insert

Stack - Push

Algorithm

1. Let `ref`, the reference of the pointer at the head node of the Stack, and `value` the value to insert
2. Create a dynamic node which contains the input `value`

Stack - Push

Algorithm

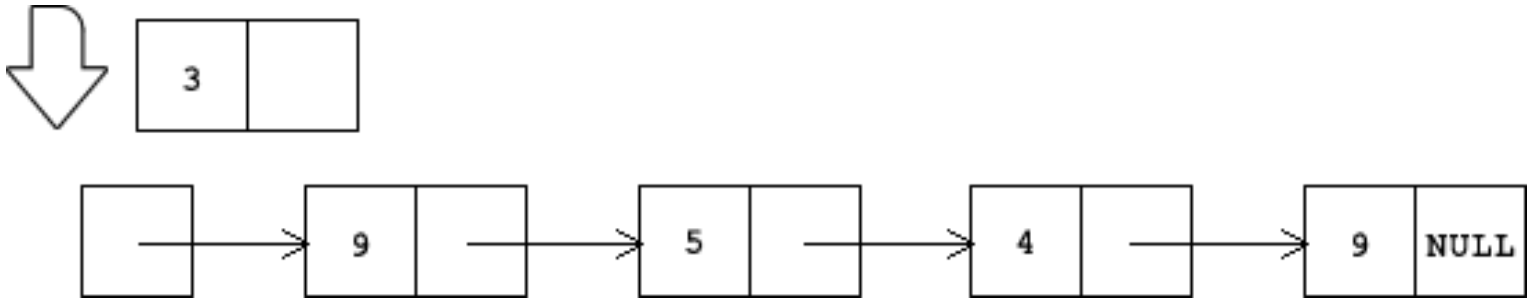
1. Let `ref`, the reference of the pointer at the head node of the Stack, and `value` the value to insert
2. Create a dynamic node which contains the input `value`
3. Point the `next` pointer of the new node, to the one referenced by `ref`

Stack - Push

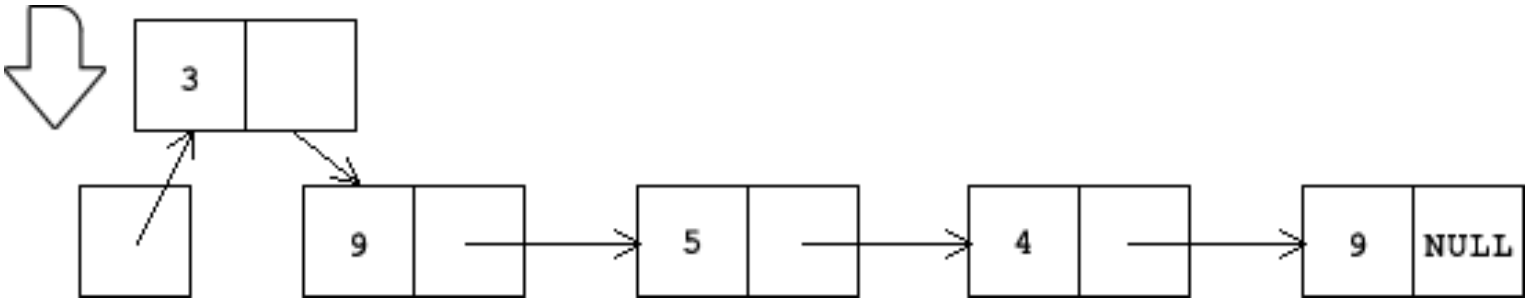
Algorithm

1. Let `ref`, the reference of the pointer at the head node of the Stack, and `value` the value to insert
2. Create a dynamic node which contains the input `value`
3. Point the `next` pointer of the new node, to the one referenced by `ref`
4. Now `ref` should contains the reference to the new node

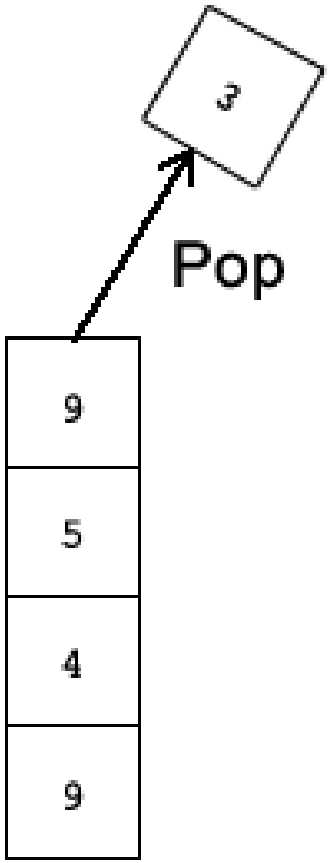
Stack - Push



Stack - Push



Stack - Pop



Definition

Remove and return the top node of the stack

Stack - Pop

Algorithm

1. Let `ref`, the reference of the pointer at the head node of the Stack

Stack - Pop

Algorithm

1. Let `ref`, the reference of the pointer at the head node of the Stack
2. Check if `ref` points to null. If so, the stack is empty and there is nothing to pop. Else ...

Stack - Pop

Algorithm

1. Let `ref`, the reference of the pointer at the head node of the Stack
2. Check if `ref` points to null. If so, the stack is empty and there is nothing to pop. Else ...
3. Create a `temp` pointer to a `Node` and make it points to the pointer referenced by `ref`

Stack - Pop

Algorithm

1. Let `ref`, the reference of the pointer at the head node of the Stack
2. Check if `ref` points to null. If so, the stack is empty and there is nothing to pop. Else ...
3. Create a `temp` pointer to a `Node` and make it points to the pointer referenced by `ref`
4. Get and store the value at the top of the stack

Stack - Pop

Algorithm

1. Let `ref`, the reference of the pointer at the head node of the Stack
2. Check if `ref` points to null. If so, the stack is empty and there is nothing to pop. Else ...
3. Create a `temp` pointer to a `Node` and make it points to the pointer referenced by `ref`
4. Get and store the value at the top of the stack
5. Now `ref` has to point to the next `node` of the stack

Stack - Pop

Algorithm

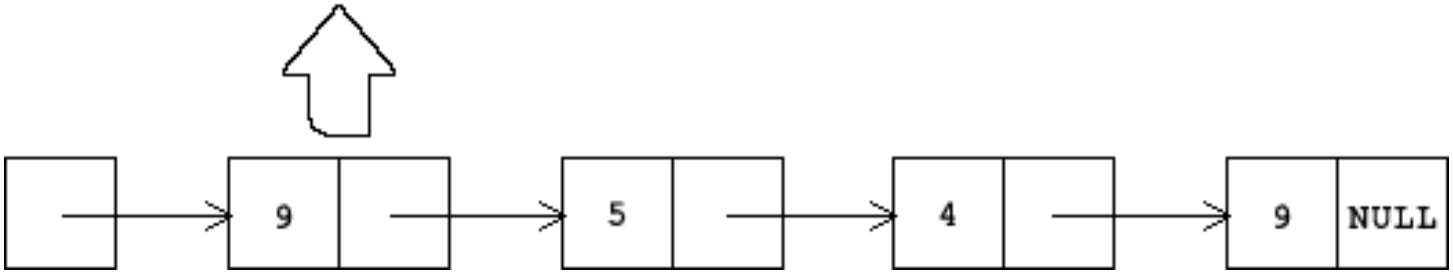
1. Let `ref`, the reference of the pointer at the head node of the Stack
2. Check if `ref` points to null. If so, the stack is empty and there is nothing to pop. Else ...
3. Create a `temp` pointer to a `Node` and make it points to the pointer referenced by `ref`
4. Get and store the value at the top of the stack
5. Now `ref` has to point to the next `node` of the stack
6. Free the memory of the `temp` node

Stack - Pop

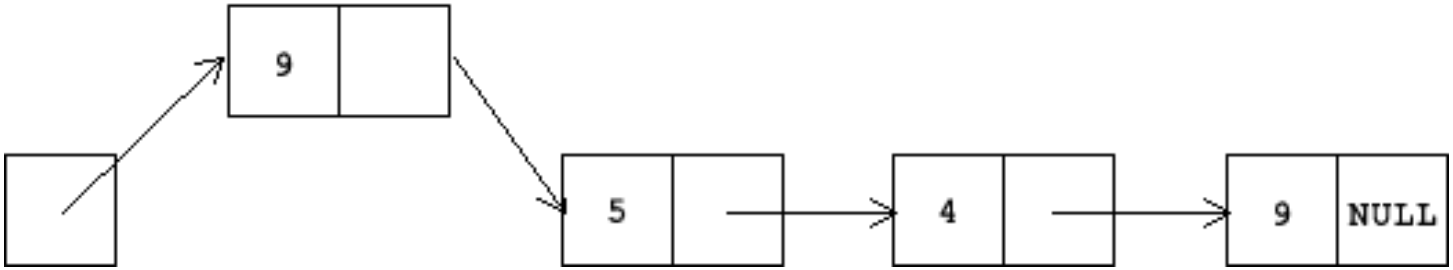
Algorithm

1. Let `ref`, the reference of the pointer at the head node of the Stack
2. Check if `ref` points to null. If so, the stack is empty and there is nothing to pop. Else ...
3. Create a `temp` pointer to a `Node` and make it points to the pointer referenced by `ref`
4. Get and store the value at the top of the stack
5. Now `ref` has to point to the next `node` of the stack
6. Free the memory of the `temp` node
7. Return the value stored at the step 4

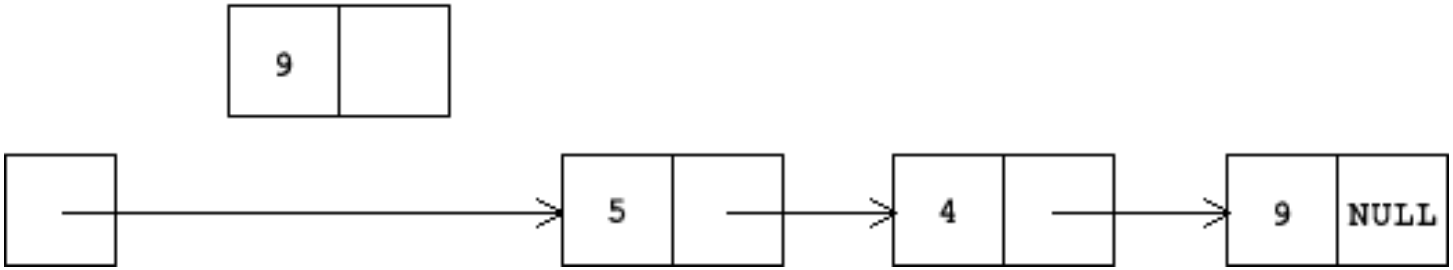
Stack - Pop



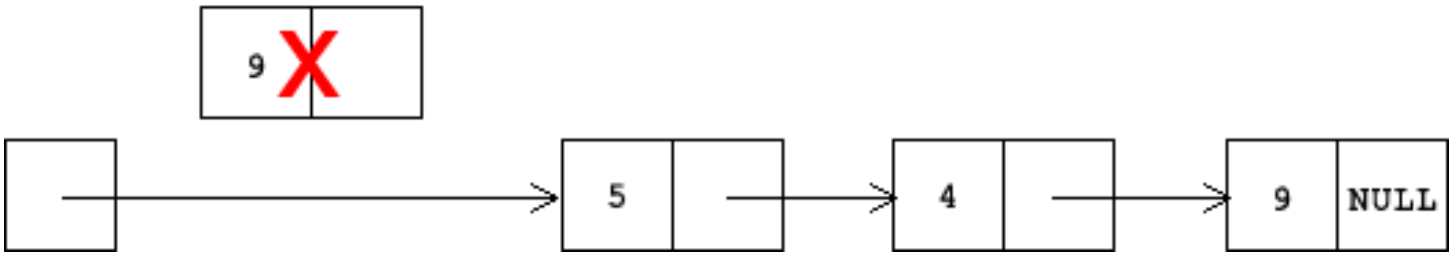
Stack - Pop



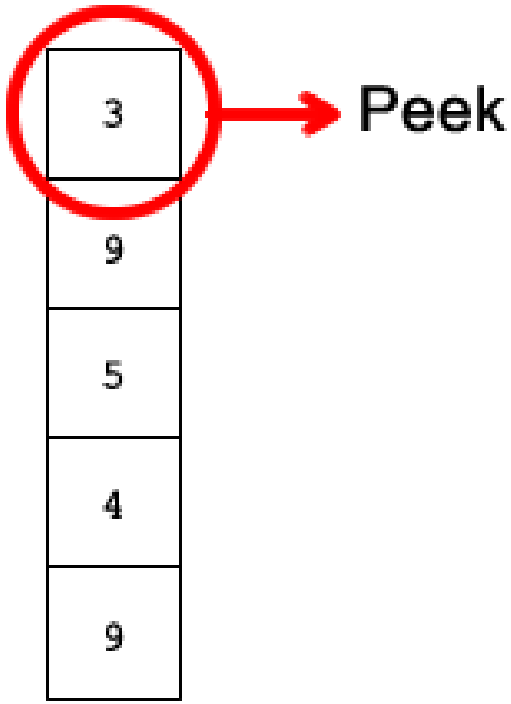
Stack - Pop



Stack - Pop



Stack - Peek

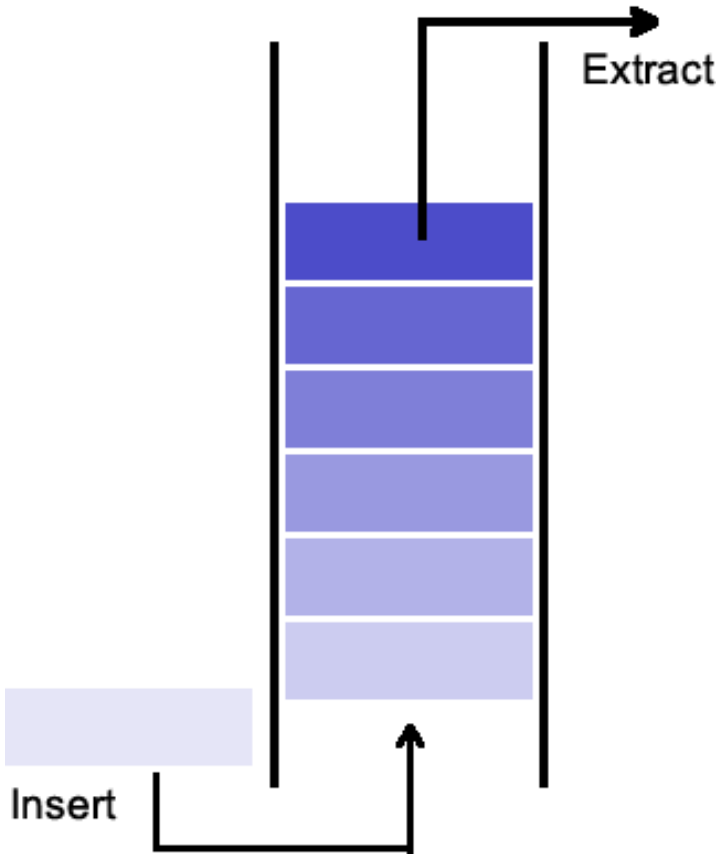


Definition

Returns the value at the top of the stack

Outline

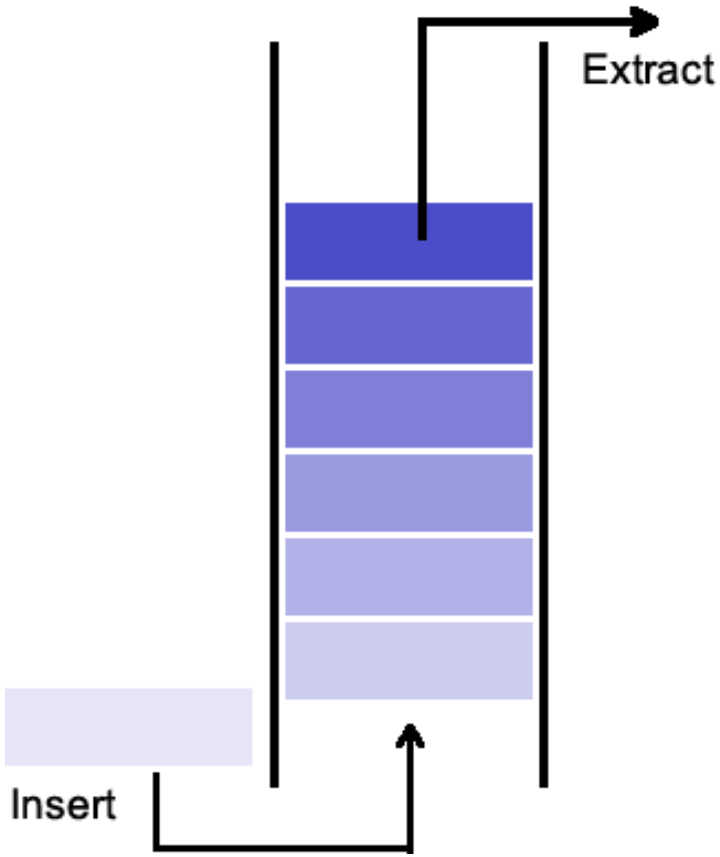
Queue



Definition

Is a type of `FIFO` list.

Queue

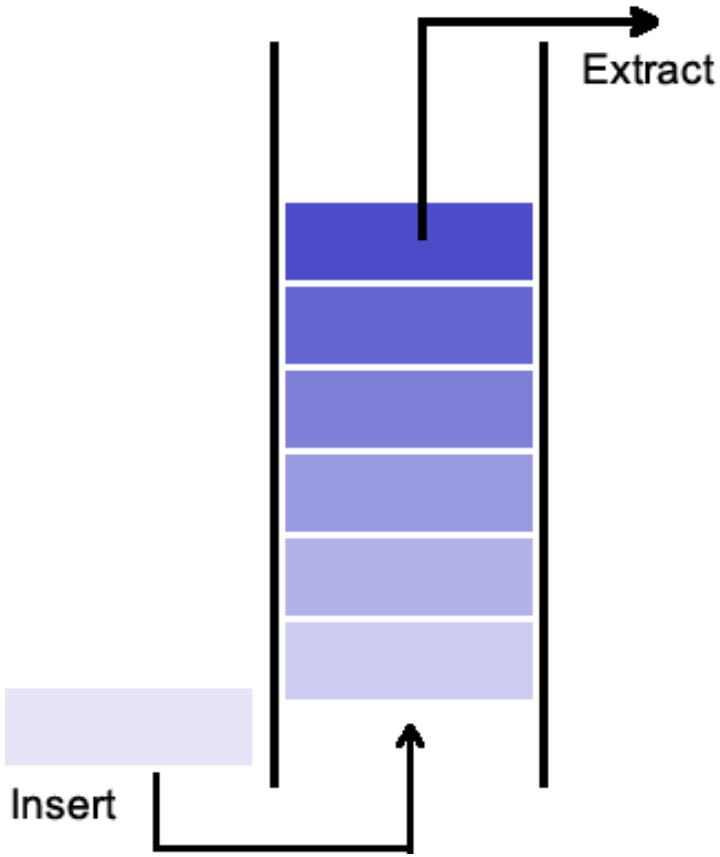


Definition

Is a type of `FIFO` list.

- *LIFO*: First In First Out

Queue

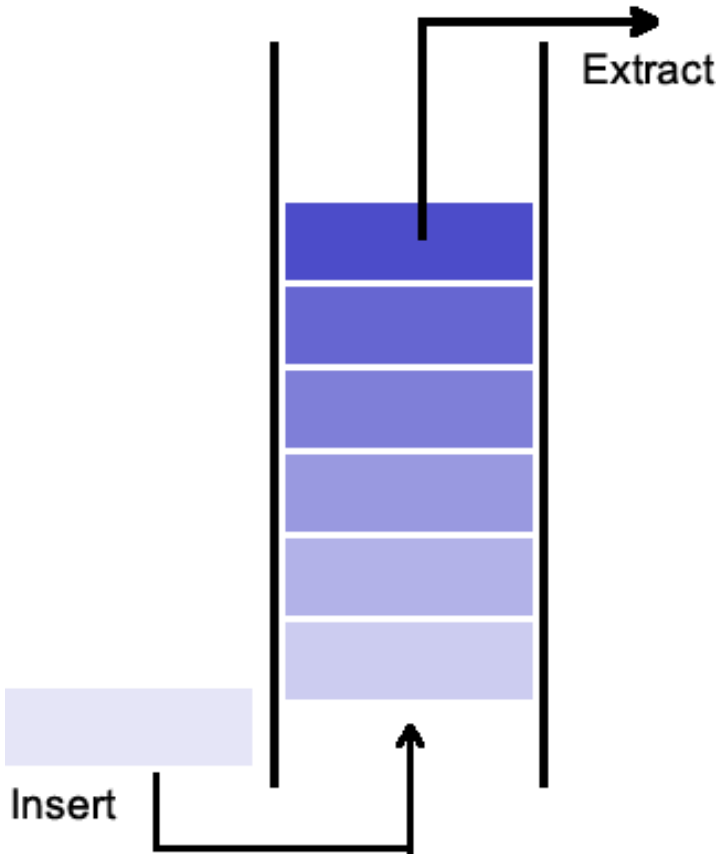


Definition

Is a type of `FIFO` list.

- *LIFO*: First In First Out
- The *First* element to In is the *First* to out

Queue

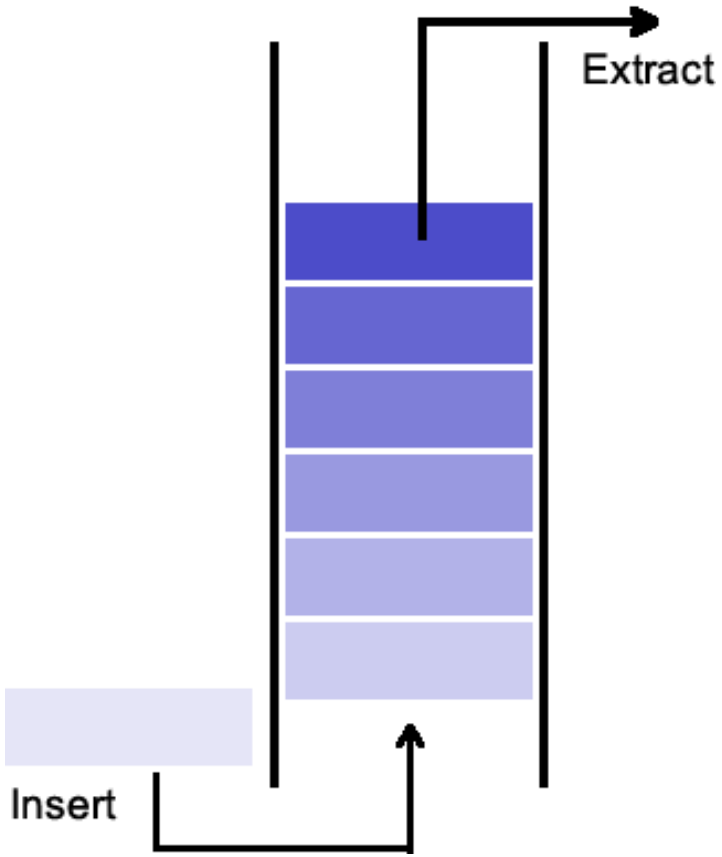


Definition

Is a type of `FIFO` list.

- *LIFO*: First In First Out
- The *First* element to In is the *First* to out
- Elements are "en-queued" at the end of the list

Queue

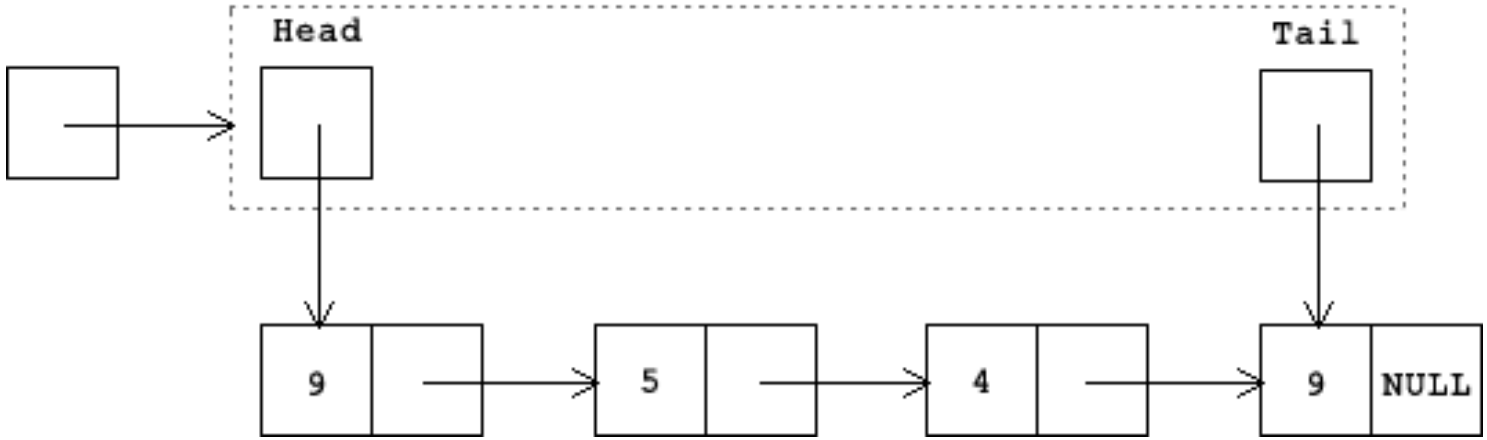


Definition

Is a type of `FIFO` list.

- *LIFO*: First In First Out
- The *First* element to In is the *First* to out
- Elements are "en-queued" at the end of the list
- We have to remove element on top to retrieve a value

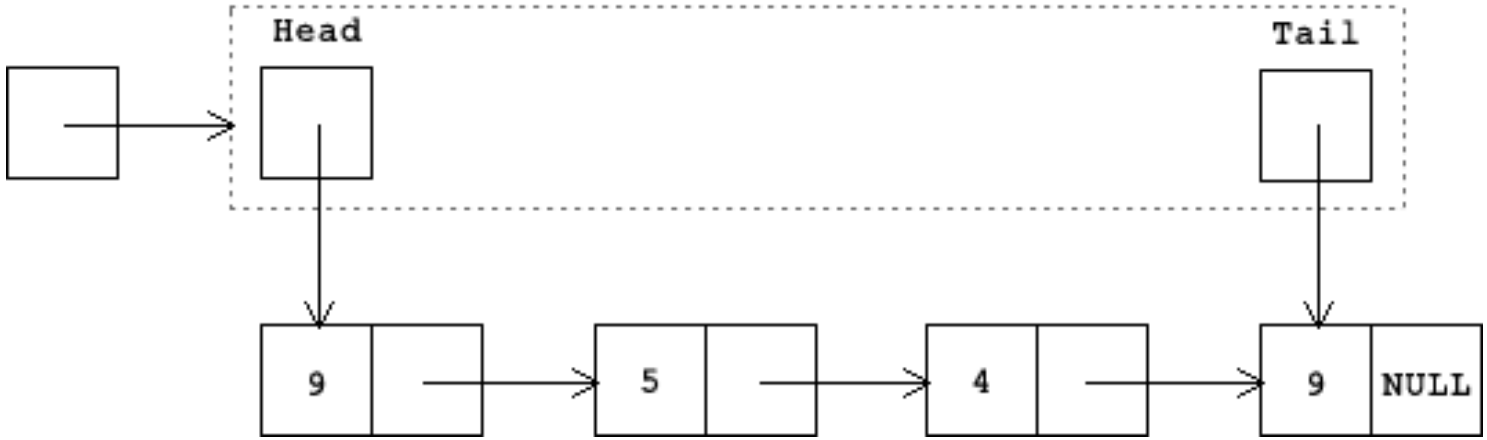
Queue



Definition

- We implement a dynamic queue using a `Linked List`

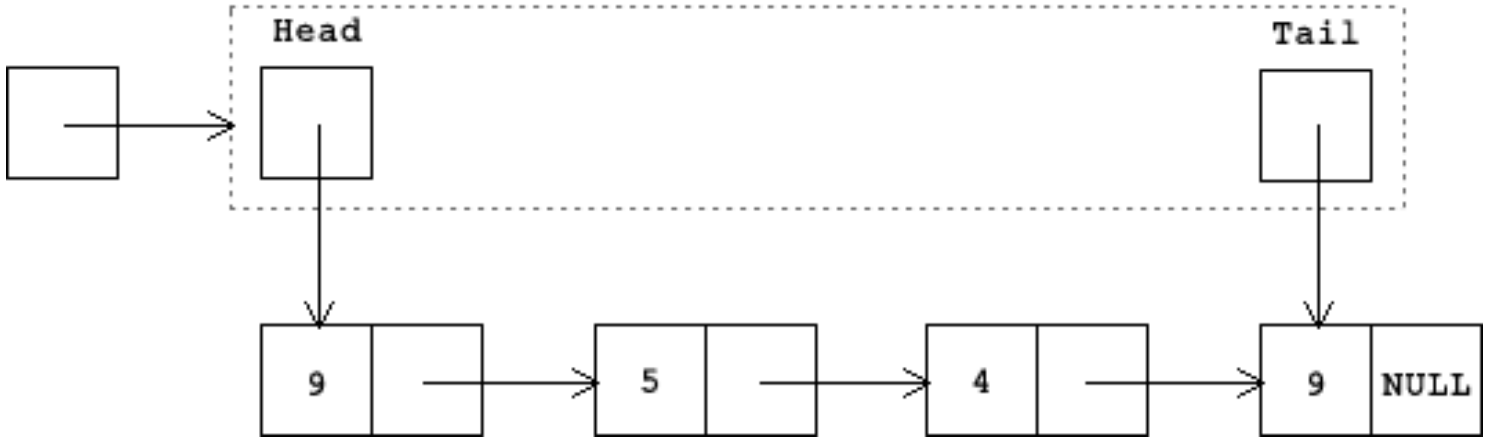
Queue



Definition

- We implement a dynamic queue using a `Linked List`
- We don't need to control the queue size (Except for memory limitations)

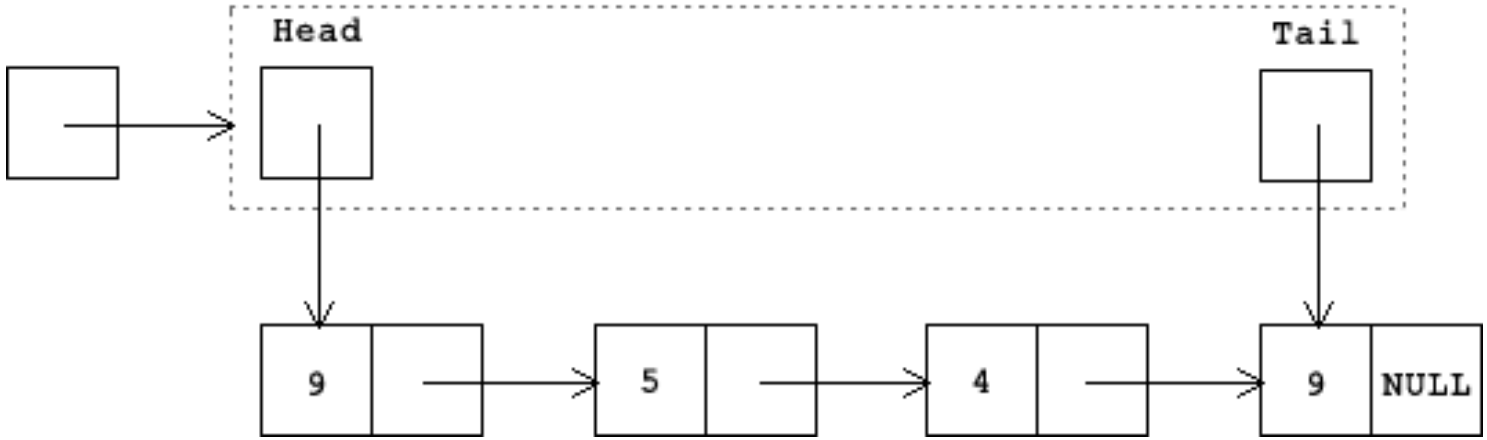
Queue



Definition

- We implement a dynamic queue using a `Linked List`
- We don't need to control the queue size (Except for memory limitations)
- New elements will be added at the end of the list

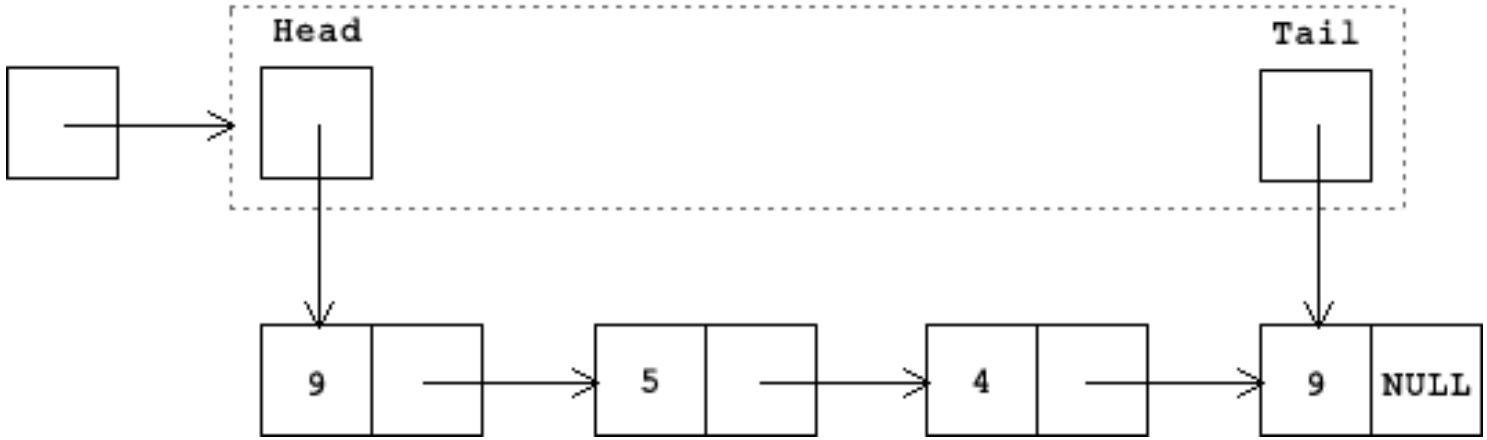
Queue



Parts

- All the data will be stored into a `Linked List`

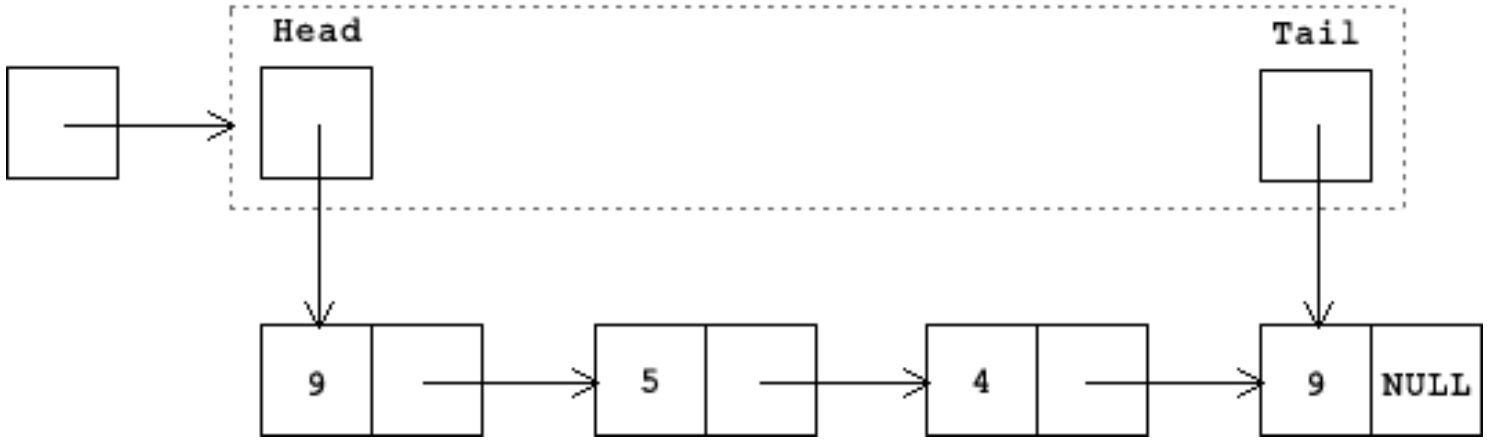
Queue



Parts

- All the data will be stored into a `Linked List`
- We need a new structure `Queue` where we will preserve two pointers:

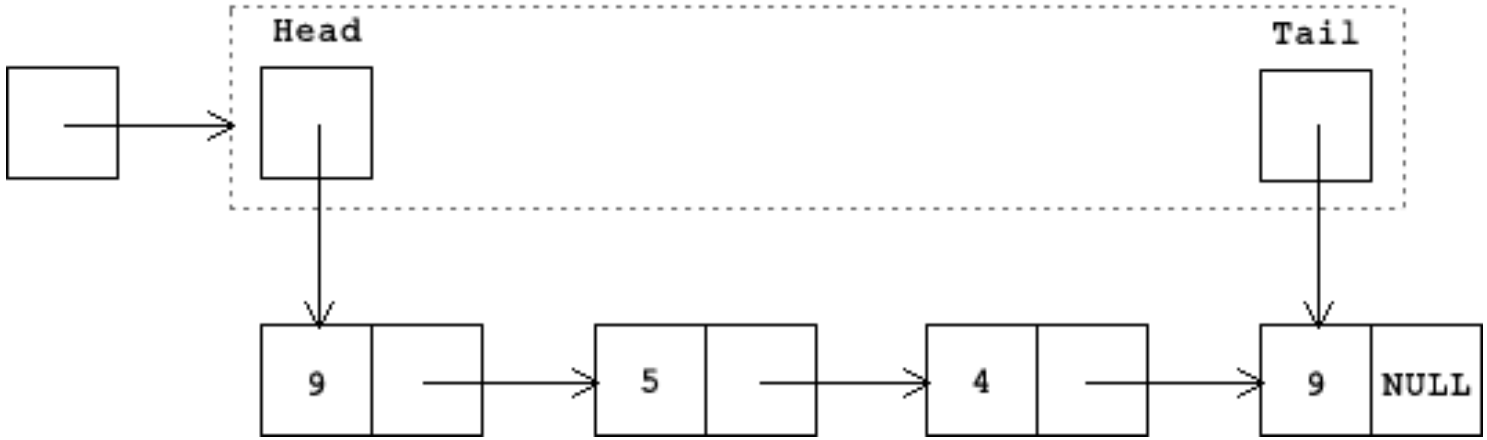
Queue



Parts

- All the data will be stored into a `Linked List`
- We need a new structure `Queue` where we will preserve two pointers:
  - `head`: A pointer to the head of the linked-list

Queue



Parts

- All the data will be stored into a `Linked List`
- We need a new structure `Queue` where we will preserve two pointers:
  - `head`: A pointer to the head of the linked-list
  - `tail`: A pointer to the end of the linked-list
- We will have 2 basic operations ...

Outline

Queue

Operations

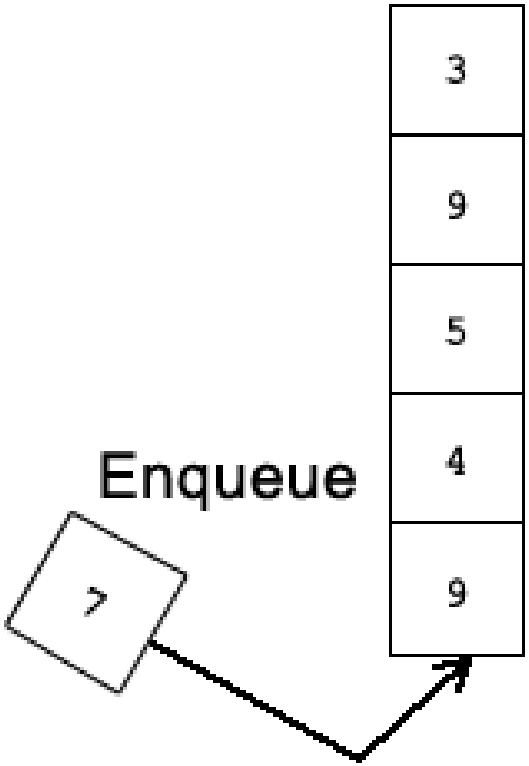
- `enqueue`
  Input: A pointer to the queue, and the element to insert
  Do: Inserts the element at the end of the queue

Queue

Operations

- enqueue
  Input: A pointer to the queue, and the element to insert
  Do: Inserts the element at the end of the queue
- dequeue
  Input: A pointer to the stack
  Do: Return the value at the top and remove the element from the queue

Stack - enqueue



Definition

- Inserts a new element at the end of the queue

Stack - enqueue

Algorithm

1. Let queue, the reference of the pointer to the queue, and value the value to insert

Stack - enqueue

Algorithm

1. Let `queue`, the reference of the pointer to the queue, and `value` the value to insert
2. Create a dynamic node which contains the input `value`

Stack - enqueue

Algorithm

1. Let `queue`, the reference of the pointer to the queue, and `value` the value to insert
2. Create a dynamic node which contains the input `value`
3. Check if the `Head` pointer of the `queue` is `NULL`

Stack - enqueue

Algorithm

1. Let `queue`, the reference of the pointer to the queue, and `value` the value to insert
2. Create a dynamic node which contains the input `value`
3. Check if the `Head` pointer of the `queue` is `NULL`
4. If so, the queue is empty. Let's point the `head` and the `tail` node of the `queue`, to the node created in the previous step

Stack - enqueue

Algorithm

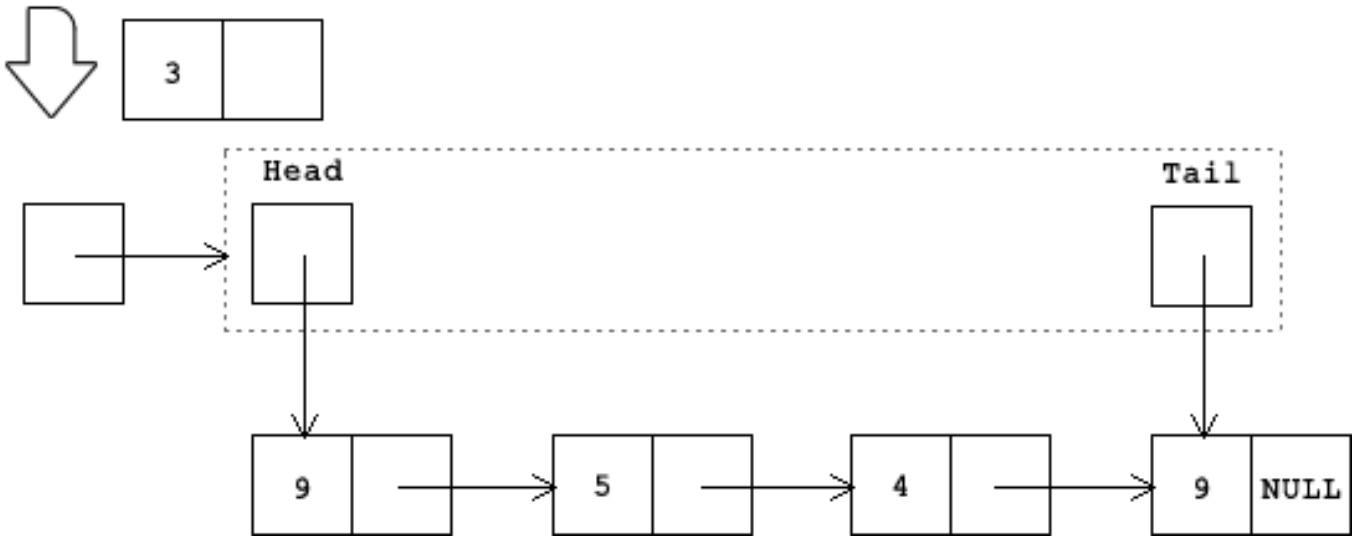
1. Let `queue`, the reference of the pointer to the queue, and `value` the value to insert
2. Create a dynamic node which contains the input `value`
3. Check if the `Head` pointer of the `queue` is `NULL`
4. If so, the queue is empty. Let's point the `head` and the `tail` node of the `queue`, to the node created in the previous step
5. Else, the `next` pointer of the node at the `tail` should point to the new node

Stack - enqueue

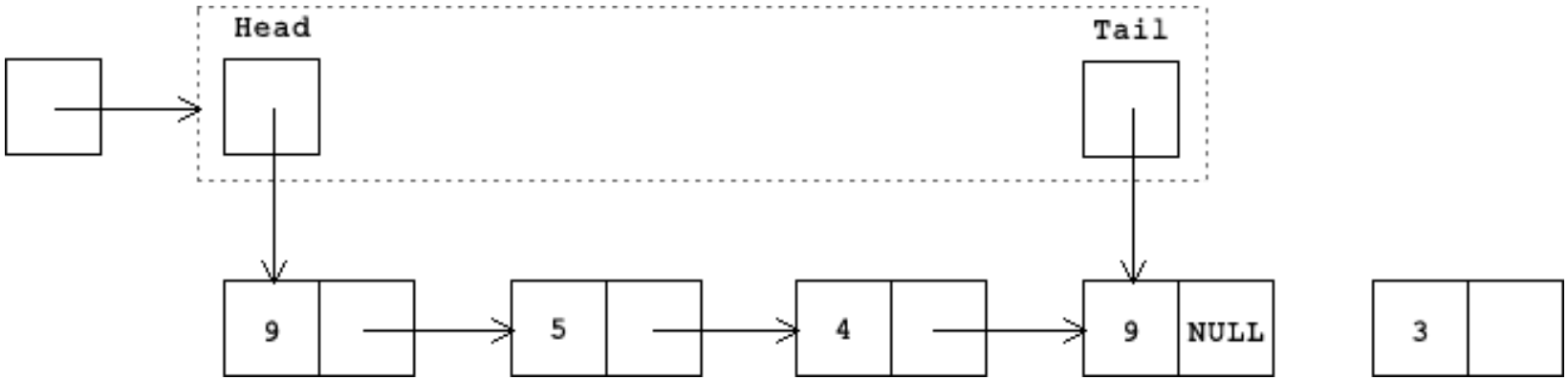
Algorithm

1. Let `queue`, the reference of the pointer to the queue, and `value` the value to insert
2. Create a dynamic node which contains the input `value`
3. Check if the `Head` pointer of the `queue` is `NULL`
4. If so, the queue is empty. Let's point the `head` and the `tail` node of the `queue`, to the node created in the previous step
5. Else, the `next` pointer of the node at the `tail` should point to the new node
6. Finally, the `tail` should point to the new node

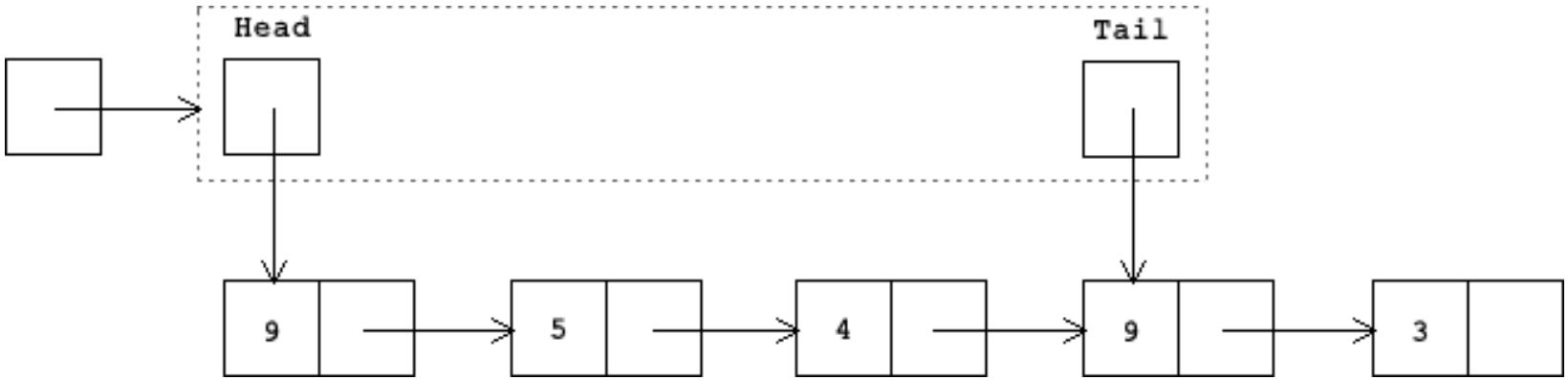
Stack - Enqueue



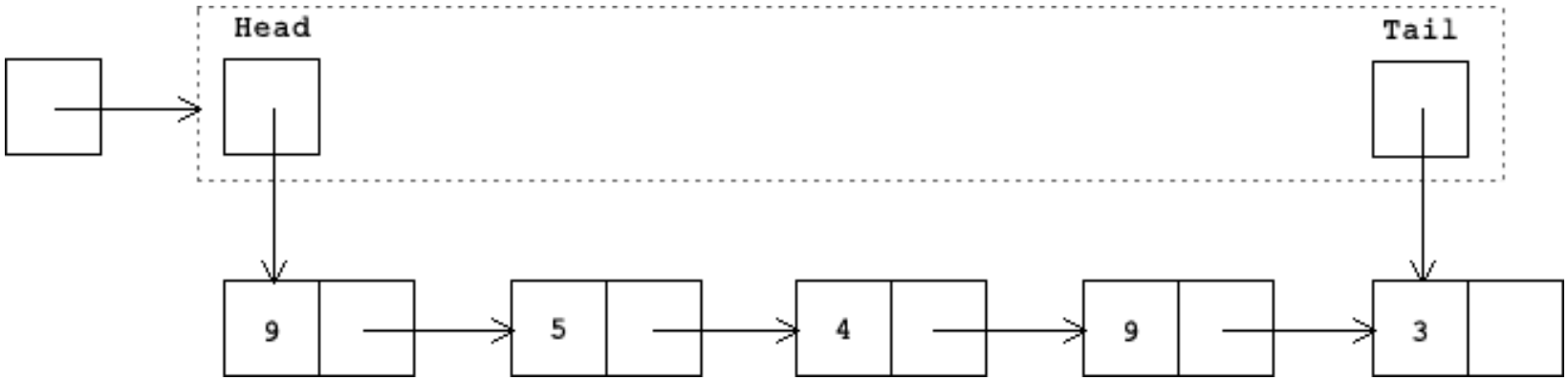
Stack - Enqueue



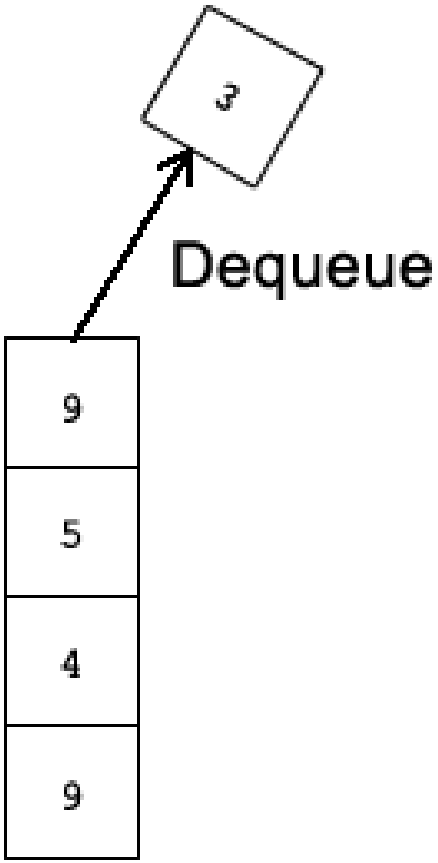
Stack - Enqueue



Stack - Enqueue



Stack - dequeue



Definition

Remove and return the top node of the queue

Stack - dequeue

Algorithm

1. Let queue, the reference of the pointer to the queue

Stack - dequeue

Algorithm

1. Let queue, the reference of the pointer to the queue
2. Check if the `head` pointer of the queue points to `NULL`. If so, the queue is empty and there is nothing to dequeue. Else ...

Stack - dequeue

Algorithm

1. Let `queue`, the reference of the pointer to the queue
2. Check if the `head` pointer of the `queue` points to `NULL`. If so, the queue is empty and there is nothing to dequeue. Else ...
3. Create a `temp` pointer to the `head` pointer of the `queue`

Stack - dequeue

Algorithm

1. Let `queue`, the reference of the pointer to the queue
2. Check if the `head` pointer of the `queue` points to `NULL`. If so, the queue is empty and there is nothing to dequeue. Else ...
3. Create a `temp` pointer to the `head` pointer of the `queue`
4. Get the value of the node at the `head` of the `queue`

Stack - dequeue

Algorithm

1. Let `queue`, the reference of the pointer to the queue
2. Check if the `head` pointer of the `queue` points to `NULL`. If so, the queue is empty and there is nothing to dequeue. Else ...
3. Create a `temp` pointer to the `head` pointer of the `queue`
4. Get the value of the node at the `head` of the `queue`
5. Move the `head` pointer to the `next` node of the `temp` node

Stack - dequeue

Algorithm

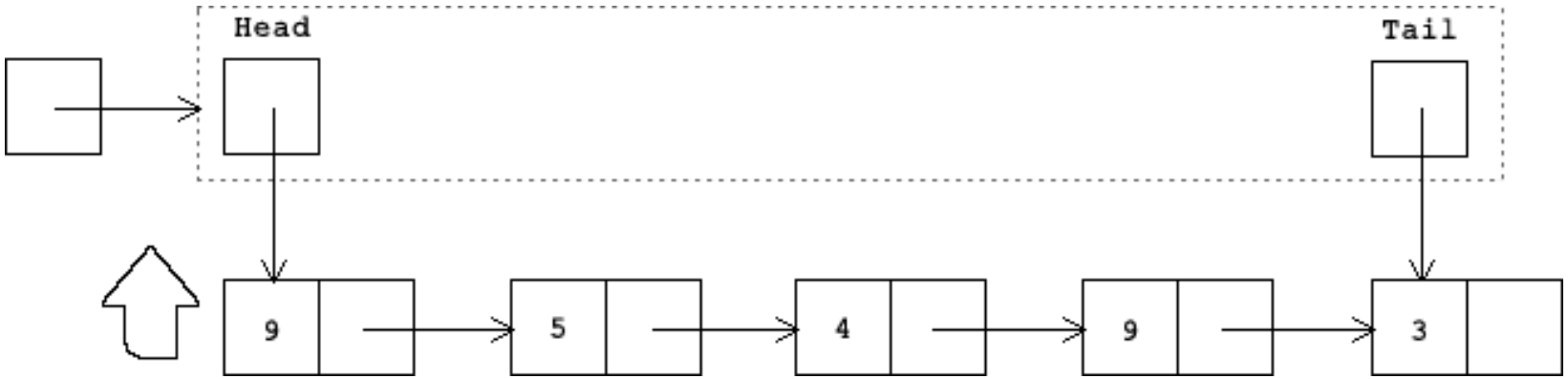
1. Let `queue`, the reference of the pointer to the queue
2. Check if the `head` pointer of the `queue` points to `NULL`. If so, the queue is empty and there is nothing to dequeue. Else ...
3. Create a `temp` pointer to the `head` pointer of the `queue`
4. Get the value of the node at the `head` of the `queue`
5. Move the `head` pointer to the `next` node of the `temp` node
6. Free the memory of the `temp` node

Stack - dequeue

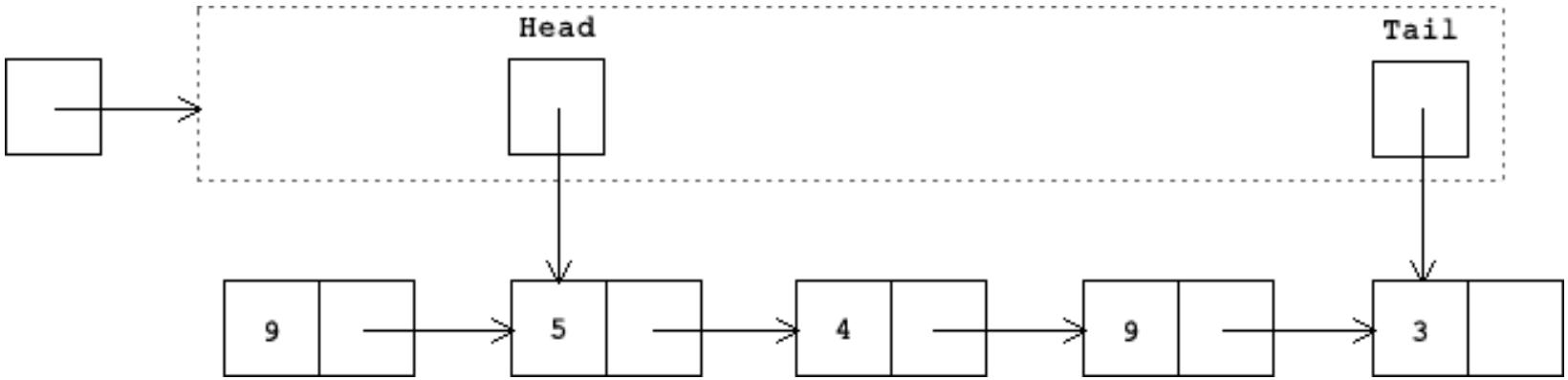
Algorithm

1. Let `queue`, the reference of the pointer to the queue
2. Check if the `head` pointer of the `queue` points to `NULL`. If so, the queue is empty and there is nothing to dequeue. Else ...
3. Create a `temp` pointer to the `head` pointer of the `queue`
4. Get the value of the node at the `head` of the `queue`
5. Move the `head` pointer to the `next` node of the `temp` node
6. Free the memory of the `temp` node
7. Return the value obtained at the step 4

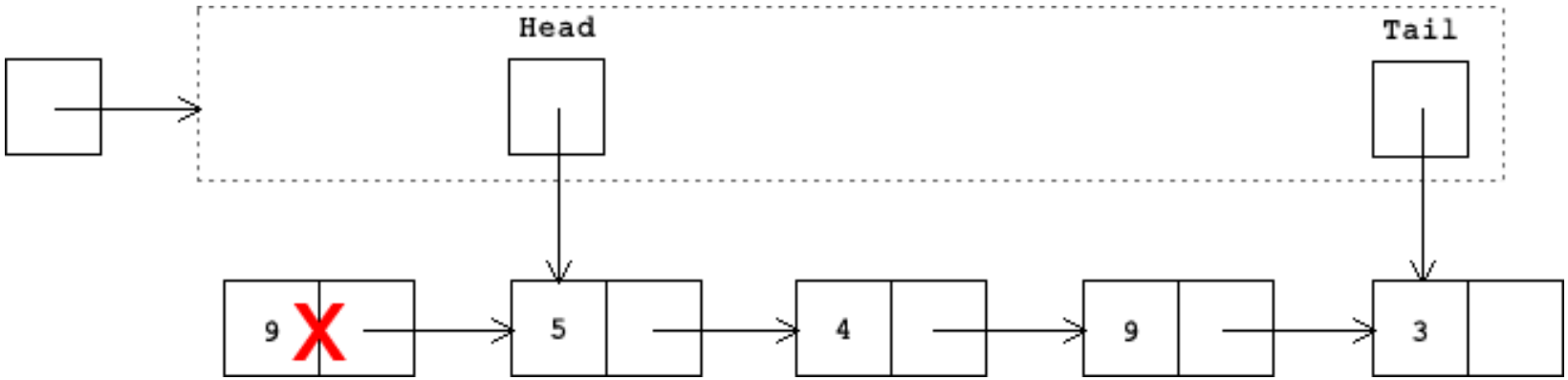
Stack - Dequeue



Stack - Dequeue



Stack - Dequeue



Stack - Dequeue