Unit 3: Linear List

Prepared By:
Tanvi Patel
Asst. Prof. (CSE)

Contents

- **Array:** Representation of arrays, Operations on Arrays, Applications of arrays

- **Stack:** Stack-Definitions & Concepts, Operations On Stacks, Applications of Stacks-Polish Expression, Reverse Polish Expression.

- **Queue:** Representation Of Queue, Operations On Queue, Circular Queue, Priority Queue, Array representation of Priority Queue, Double Ended Queue, Applications of Queue

- **Linked List:** Singly Linked List, Doubly Linked list,Circular linked list ,Linked implementation of Stack, Linked implementation of Queue, Applications of linked list

Array

- *Representation of arrays*
- *Operations on Arrays*
- *Applications of arrays*

Introduction to Array

◉ An array is a fundamental data structure available in most programming languages and it has a wide range of uses across different algorithms.

◉ Arrays consist of fixed-size data records that allow each element to be efficiently located based on its index.

◉ It is a container which can hold a fix number of items and these items should be of the same type. Most of the data structures make use of arrays to implement their algorithms.

◉ Performance-wise, it's very fast to look up an element contained in an array given the element's index. A proper array implementation guarantees a constant $O(1)$ access time for this case.
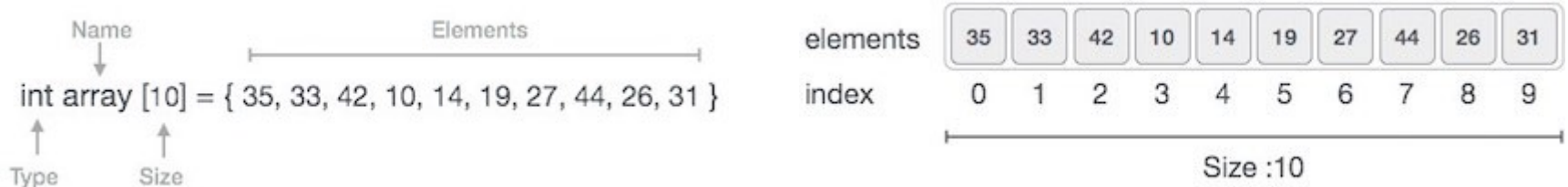
Introduction to Array (Cont.)

Following are the important terms to understand the concept of Array.

- **Element**– Each item stored in an array is called an element.
- **Index** – Each location of an element in an array has a numerical index, which is used to identify the element.

Representation of arrays

◉ Arrays can be declared in various ways in different languages. Below is an illustration.

Name
Elements
int array [10] = { 35, 33, 42, 10, 14, 19, 27, 44, 26, 31 }
Type   Size

elements   35 33 42 10 14 19 27 44 26 31
index   0   1   2   3   4   5   6   7   8   9
Size :10

◉ As per the above illustration, following are the important points to be considered.

◉ Index starts with 0.

◉ Array length is 10 which means it can store 10 elements.

◉ Each element can be accessed via its index. For example, we can fetch an element at index 6 as 9.

Operations on Arrays

Some of the basic operations supported by an array are as follows:

◉ **Traverse** - It prints all the elements one by one.

◉ **Insertion** - It adds an element at the given index.

◉ **Deletion** - It deletes an element at the given index.

◉ **Search** - It searches an element using the given index or by the value.

◉ **Update** - It updates an element at the given index.

Array is created in Python by importing array module to the python program. Then the array is declared as shown below.

```
from array import *

arrayName = array(typecode, [initializers])
```

Typecode are the codes that are used to define the type of value the array will hold.

Operations on Arrays (Cont.)

Type Code	C Type	Python Type	Minimum Size In Bytes
'b'	signed cahar	int	1
'B'	unsigned char	int	1
'u'	Py_UNICODE	unicode character	2
'h'	signed short	int	2
'H'	unsigned short	int	2
'i'	signed int	int	2
'I'	unsigned int	int	2
'l'	signed long	int	4
'L'	unsigned long	int	4
'q'	signed long long	int	8
'Q'	unsigned long long	int	8
'f'	float	float	4
'd'	double	float	8

Operations on Arrays (Cont.)

- Before looking at various array operations lets create and print an array using python.
- The below code creates an array named array1.

```
from array import *

array1 = array('i', [10,20,30,40,50])

for x in array1:
 print(x)
```

When we compile and execute the above program, it produces the following result –

Output

```
10
20
30
40
50
```

Accessing array elements

◉ We can access the array elements using the respective indices of those elements.

```python
import array as arr
a = arr.array('i', [2, 4, 6, 8])
print("First element:", a[0])
print("Second element:", a[1])
print("Second last element:", a[-1])
```

Output:

```
First element: 2
Second element: 4
Second last element: 8
```

Explanation: In the above example, we have imported an array, defined a variable named as "a" that holds the elements of an array and print the elements by accessing elements through indices of an array.

Insertion Operation

◉ Elements can be added to the Array by using built-in underline insert() function.

◉ Insert is used to insert one or more data elements into an array.

◉ Based on the requirement, a new element can be added at the beginning, end, or any given index of array.

◉ append() is also used to add the value mentioned in its arguments at the end of the array.

```
from array import *

array1 = array('i', [10,20,30,40,50])

array1.insert(1,60)

for x in array1:
 print(x)
```

Output

```
10
60
20
30
40
50
```

Operations on Arrays (Cont.)

Deletion Operation

◉ Deletion refers to removing an existing element from the array and re-organizing all elements of an array.

◉ Elements can be removed from the array by using built-in remove() function but an Error arises if element doesn't exist in the set.

◉ pop() function can also be used to remove and return an element from the array, but by default it removes only the last element of the array, to remove element from a specific position of the array, index of the element is passed as an argument to the pop() method.

Operations on Arrays (Cont.)

◉ The elements can be deleted from an array using Python's **del** statement. If we want to delete any value from the array, we can do that by using the indices of a particular element.

```python
import array as arr
number = arr.array('i', [1, 2, 3, 3, 4])
del number[2]                    # removing third element
print(number)                    # Output: array('i', [1, 2, 3, 4])
```

Output:

```
array('i', [10, 20, 40, 60])
```

Explanation: In the above example, we have imported an array and defined a variable named as "number" which stores the values of an array. Here, by using del statement, we are removing the third element [3] of the given array.

◉ Here, we remove a data element at the middle of the array using the python in-built remove() method.

```
from array import *

array1 = array('i', [10,20,30,40,50])

array1.remove(40)

for x in array1:
 print(x)
```

Output

```
10
20
30
50
```

Search Operation

◉   You can perform a search for an array element based on its value or its index.

◉   Here, we search a data element using the python in-built index() method.

```
from array import *

array1 = array('i', [10,20,30,40,50])

print (array1.index(40))
```

Output

```
3
```

Update Operation

◉ Update operation refers to updating an existing element from the array at a given index.

◉ Here, we simply reassign a new value to the desired index we want to update.

```python
from array import *

array1 = array('i', [10,20,30,40,50])

array1[2] = 80

for x in array1:
 print(x)
```

Output

```
10
20
80
40
50
```

Operations on Arrays (Cont.)

◉ Arrays are mutable, and their elements can be changed in a similar way like lists.

```python
import array as arr
numbers = arr.array('i', [1, 2, 3, 5, 7, 10])

# changing first element
numbers[0] = 0
print(numbers)    # Output: array('i', [0, 2, 3, 5, 7, 10])

# changing 3rd to 5th element
numbers[2:5] = arr.array('i', [4, 6, 8])
print(numbers)    # Output: array('i', [0, 2, 4, 6, 8, 10])
```

Output:

```
array('i', [0, 2, 3, 5, 7, 10])
array('i' ,[0, 2, 4, 6, 8, 10])
```

Explanation: In the above example, we have imported an array and defined a variable named as "numbers" which holds the value of an array. If we want to change or add the elements in an array, we can do it by defining the particular index of an array where you want to change or add the elements.

Application of Array

1) Array stores data elements of the same data type.

2) Maintains multiple variable names using a single name. Arrays help to maintain large data under a single variable name. This avoid the confusion of using multiple variables.

3) Arrays can be used for sorting data elements. Different sorting techniques like Bubble sort, Insertion sort, Selection sort etc use arrays to store and sort elements easily.

4) Lastly, arrays are also used to implement other data structures like Stacks, Queues, Heaps, Hash tables etc.

Application of Array

Example:

◉ You use arrays all the time in programming. Whenever you have to keep track of an ordered list of items, you will end up using an array. Be it a list of songs, a list of books, or list of anything for that matter. Even in the JSON data format, you will often use an array to hold a list of objects.

◉ Databases generally provide a different functionality, as in they let you keep data over time. You can retrieve that data and update it and view it from other computers using other web browsers. You will often retrieve information from a database and store it for local processing in an array. So arrays is an important concept in programming languages.

Summary

- Suppose, we need to see the average temperature observed in a month: Ideally, you create 30/31 variables; scan each variable value, take sum, and then average it out.

- Now consider if you have to average out the temperatures over 6 months or a year or more. It sounds crazy right, to scan 365 or more values, operate on them, and give the result? Also, in practical applications, the data involved is huge.

- **Arrays** are nothing but a collection of items of the same data type having a fixed size.

- It is a collection of similar data elements.

- These elements are stored in consecutive or contiguous blocks of memory.

- The memory locations where these items are stored are referenced to by *index or subscript.* Index is an ordinal number of the element beginning from the starting location of array.

- An array is referenced by a single name or variable.

Stack

- *Stack Definitions & Concepts*
- *Operations on Stacks*
- *Applications of Stacks–Polish Expression, Reverse Polish Expression*

Stack Definition and Concept

- Data structures are the key to organize storage in computers so that we can efficiently access and edit data.

- *Stacks* is one of the earliest data structures defined in computer science.

- In simple words, A stack is a linear data structure where data is arranged objects on over another. The data is stored in a similar order as plates are arranged one above another in the kitchen.

- The simple example of a stack is the **Undo** feature in the editor. The Undo fea͟͟͟͟͟͟͟͟ ͟͟vent that we have done.

- Think about the things you can do with such a pile of plates

- Put a new plate on top

- Remove the top plate

- If you want the plate at the bottom, you must first remove all the plates on top ͟͟͟͟s called **Last In First Out** - the last item that is the first item to go out.

Stack Definition and Concept (Cont.)

- **LIFO Principle of Stack**
- In prog
  called
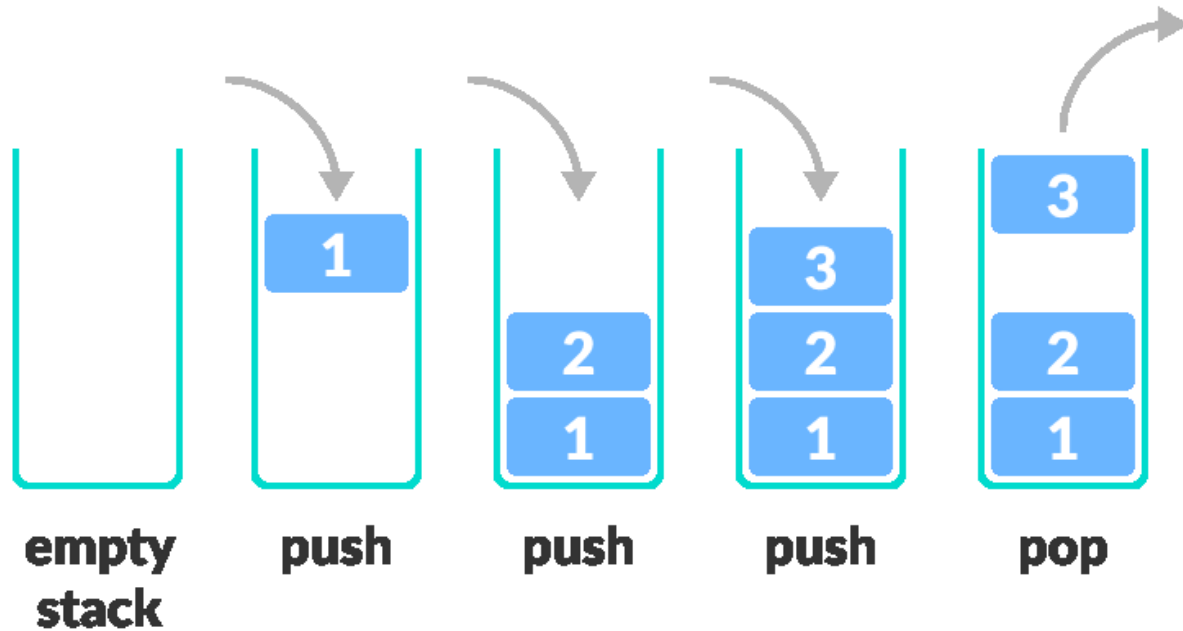
Why and When do we use Stack?

◉ Stacks are simple data structures that allow us to store and retrieve data sequentially. Talking about performance, a proper stack implementation is expected to take *O(1)* time for insert and delete operations.

◉ To understand Stack at the ground level, think about a pile of books. You add a book at the top of the stack, so the first one to be picked up will be the last one that was added to the stack.

◉ There are many real-world use cases for stacks, understanding them allows us to solve many data storage problems in an easy and effective way.

◉ Imagine you're a developer and you are working on a brand new word processor. You need to create an undo feature – allowing users to backtrack their actions until the beginning of the session. A stack is an ideal fit for this scenario. We can record every action of the user by pushing it to the stack. When the user wants to undo an action they can pop accordingly from the stack.

Operations on Stacks

A stack is an object (an abstract data type - ADT) that allows the following operations:

- **Push**: Add an element to the top of a stack
- **Pop**: Remove an element from the top of a stack
- **IsEmpty**: Check if the stack is empty
- **IsFull**: Check if the stack is full
- **Peek**: Get the value of the top element without removing it

Operations on Stacks (Cont.)

Push

◉ The operation to insert elements in a stack is called **push**. When we push the book on a stack, we put the book on the previous top element which means that the new book becomes the top element. This is what we mean when we use the push operation, we push elements onto a stack. We insert elements onto a stack and the last element to be pushed is the new top of the stack.

Pop

◉ There is another operation that we can perform on the stack, popping. Popping is when we take the top book of the stack and put it down. This implies that when we remove an element from the stack, the stack follows the First-In, Last Out property. This means that the top element, the last to be inserted, is removed when we perform the pop operation.

◉ Push and Pop are two fundamental routines that we'll need for this data structure.

Peek

◉ Another thing that we can do is view the top element of the stack so we can ask the data structure: "What's the top element?" and it can give that to us using the peek operation. Note that the peek operation does not remove the top element, it merely returns it.

◉ We can also check whether or not the stack is empty, and a few other things too, that will come along the way as we implement it.

Queue

- *Representation Of Queue*
- *Operations On Queue*
- *Circular Queue*
- *Priority Queue*
- *Array representation of Priority Queue*
- *Double Ended Queue*
- *Applications of Queue*

Introduction to Queue


Queue- Insertion and Deletion

Ticket Counter — A (front) B C D E (rear)

Insert

Ticket Counter — A (front) B C D E F (rear)
(insertion done from the rear)

Delete

Ticket Counter — B (front) C D E F (rear)
(deletion done from the front)
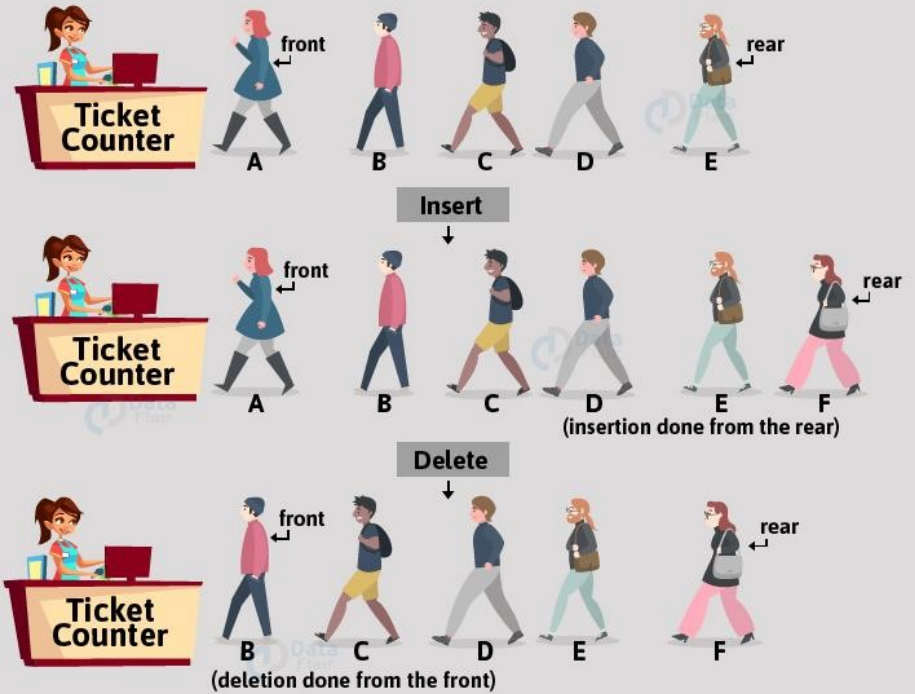
- A Queue is a abstract data structure which holds a collection of elements where they are added to the back of the queue and removed/deleted from the front of the queue.

- These ensures that the data are processed through the queue in the order in which they are received.

- In other words, a queue is a first in–first out (FIFO) structure.

- A queue is the same as a line.

Rear
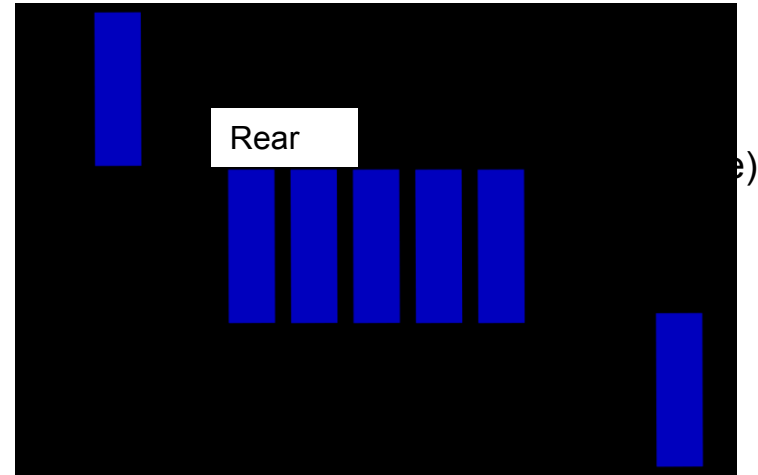
Where Queue are useful?

◉ Queues are useful in the scenario where there is only a single resource, but multiple objects that want to use or access this resource.

Without Queues

A
B
C
D

Resource

With Queues

D C B A → Resource

Waiting List
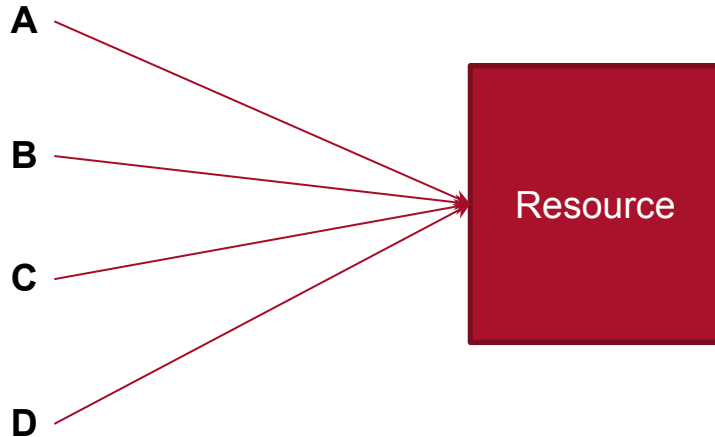
Application of Queue

◉ Queue, as the name suggests is used whenever we need to manage any group of objects in an order in which the first one coming in, also gets out first while the others wait for their turn, like in the following scenarios:

◉ Serving requests on a single shared resource, like a printer, CPU task scheduling etc.

◉ In real life scenario, Call Center phone systems uses Queues to hold people calling them in an order, until a service representative is free.

◉ All types of customer service(like railway reservation) centers are designed using the concept of queues.

Queue Operations

Mainly the following four basic operations are performed on queue::

- Enqueue
- Dequeue
- Front
- Rear

Queue Operations (Cont.)

Enqueue Operation: The queue insert operation is known as enqueue. After the data have been inserted into the queue, the new element becomes the rear.
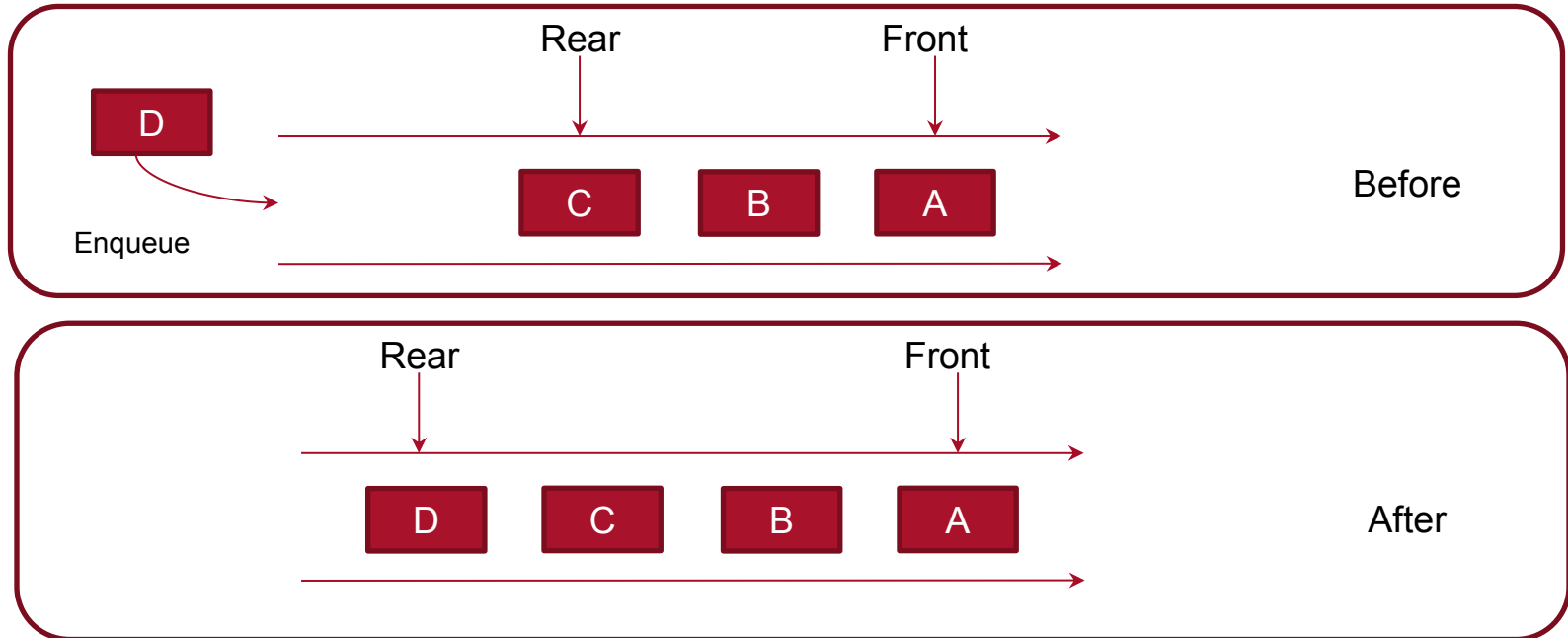
Dequeue Operation: The queue delete operation is known as dequeue. The data at the front of the queue are returned to the user and removed from the queue.

Rear

Front

Before

| D | C | B | A |

Rear

Front

Dequeue

After

| D | C | B |

A
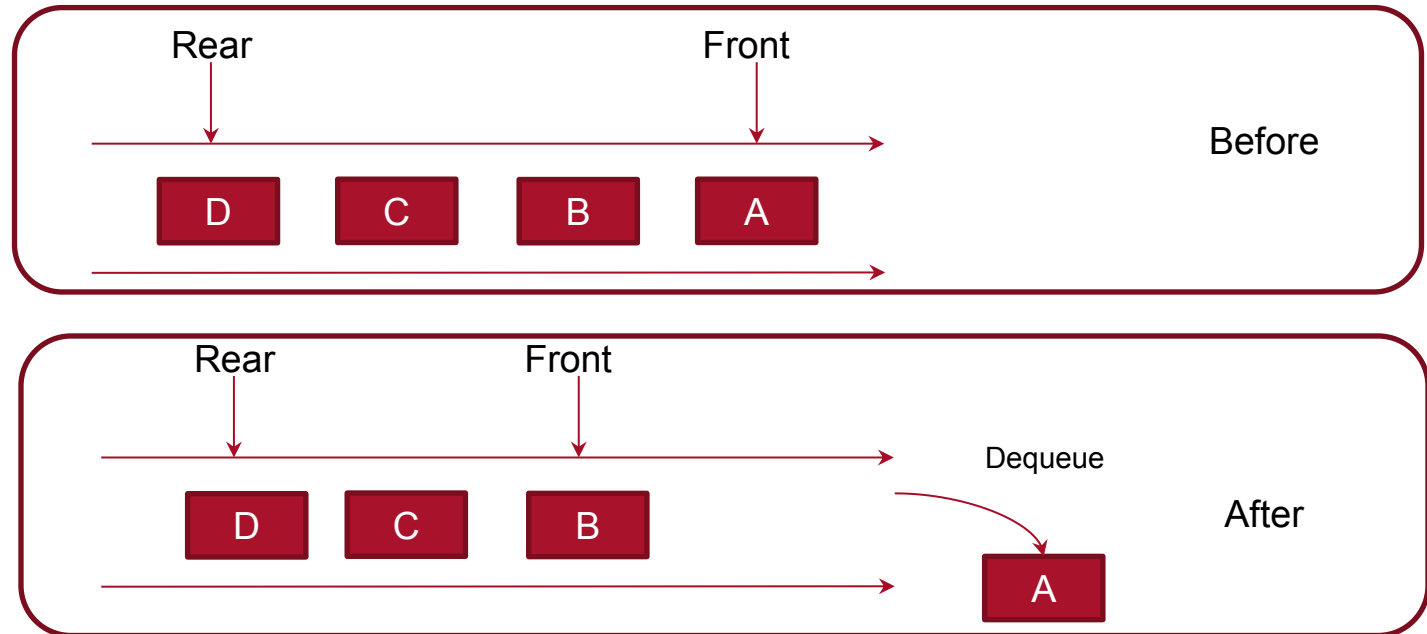
Queue Front: Data at the front of the queue can be retrieved with queue front.

◉ It returns the data at the front of the queue without changing the contents of the queue.
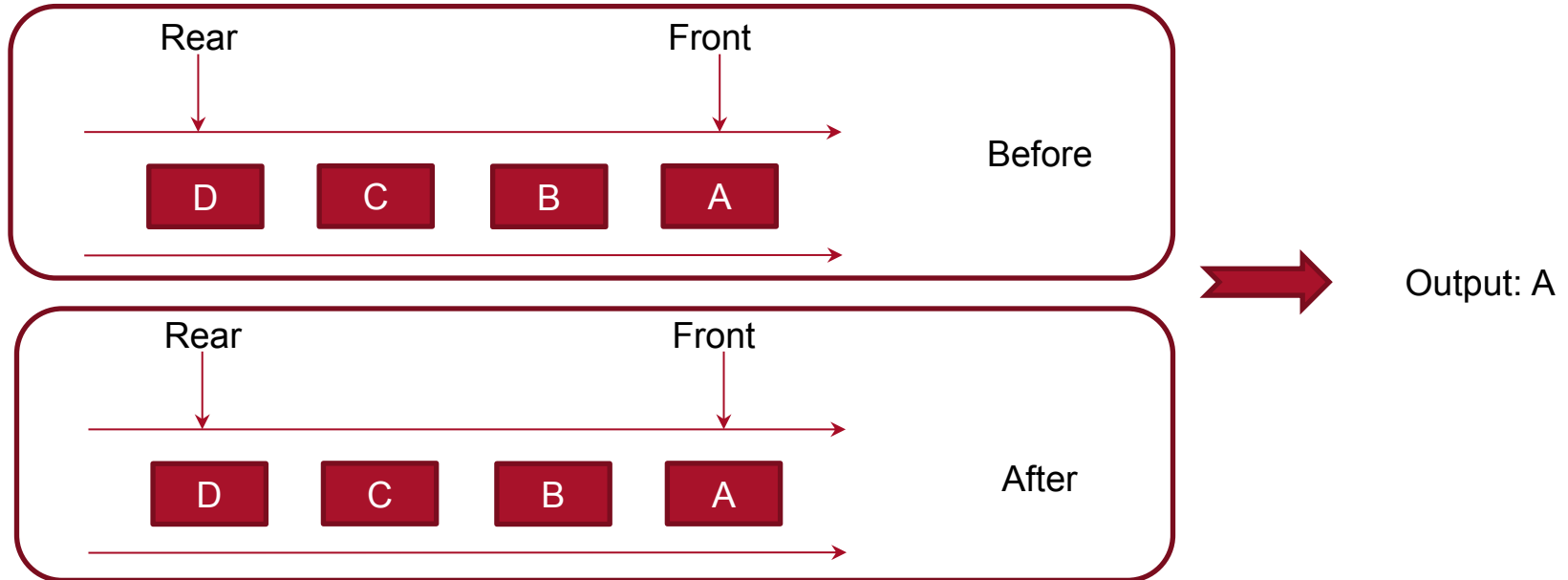
Queue Rear: Data at the rear of the queue can be retrieved with queue rear.

◉ It returns the data at the rearof the queue without changing the contents of the queue.
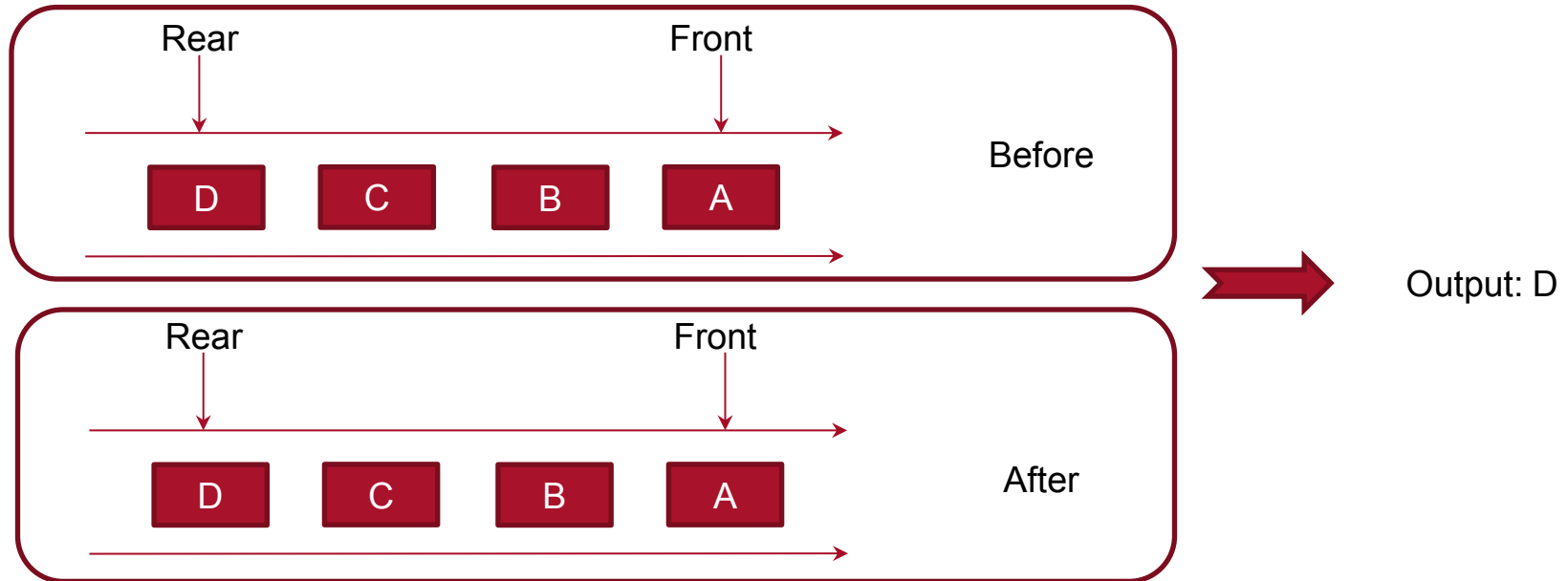
◉ Queue Example

1. Create a queue
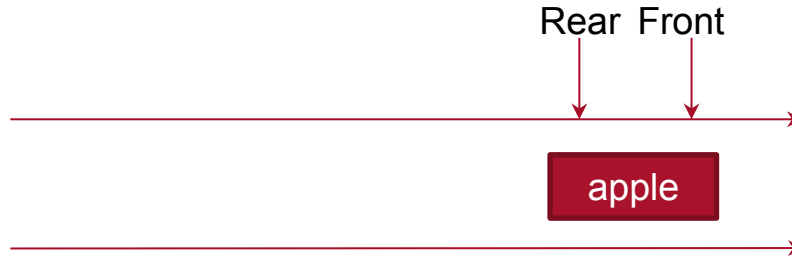
Rear Front

2. Enqueue"apple"

apple

Queue Operations (Cont.)

Rear   Front

3. Enqueue"mango"
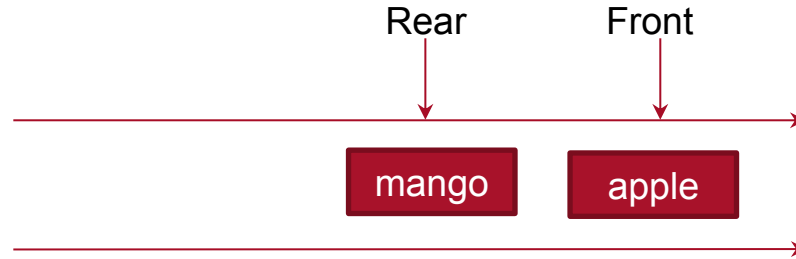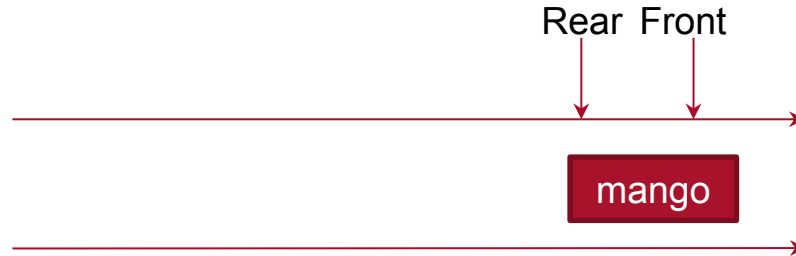
| mango | apple |

Rear Front

4. Dequeue

| mango |

Queue Operations (Cont.)

Rear   Front

5. Enqueue"cherry"

| cherry | mango |

Rear   Front

6. QueueFront

| cherry | mango |

Output: mango

Rear Front

7. Queue Rear

cherry mango

Output: cherry

Rear Front

8. Dequeue

cherry
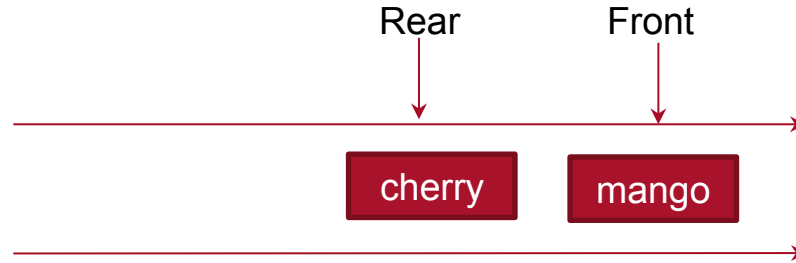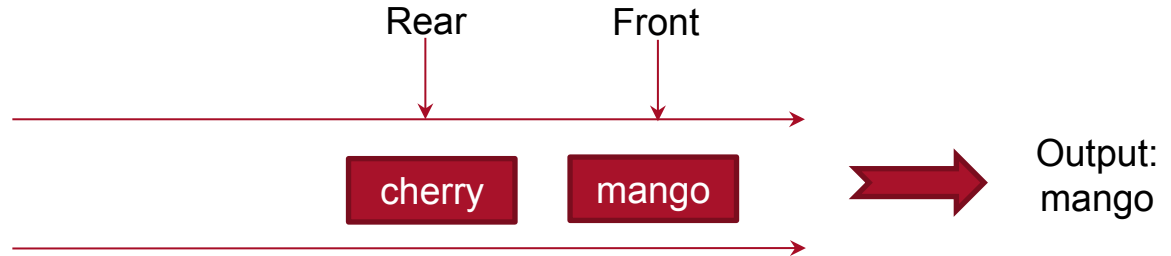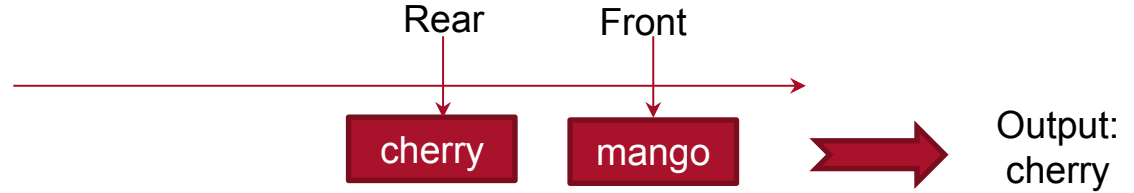
9. Dequeue
(Empty Queue)

Implementation of Queue

◉ We need two different structures to implement the queue: a queue head structure and a data node structure.

◉ After it is created, the queue will have one head node (having front, counter and rear) and zero or more data nodes (having data and link), depending on its current state.
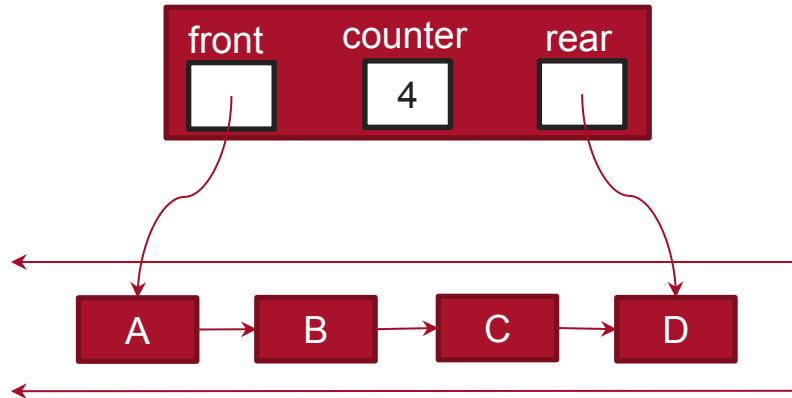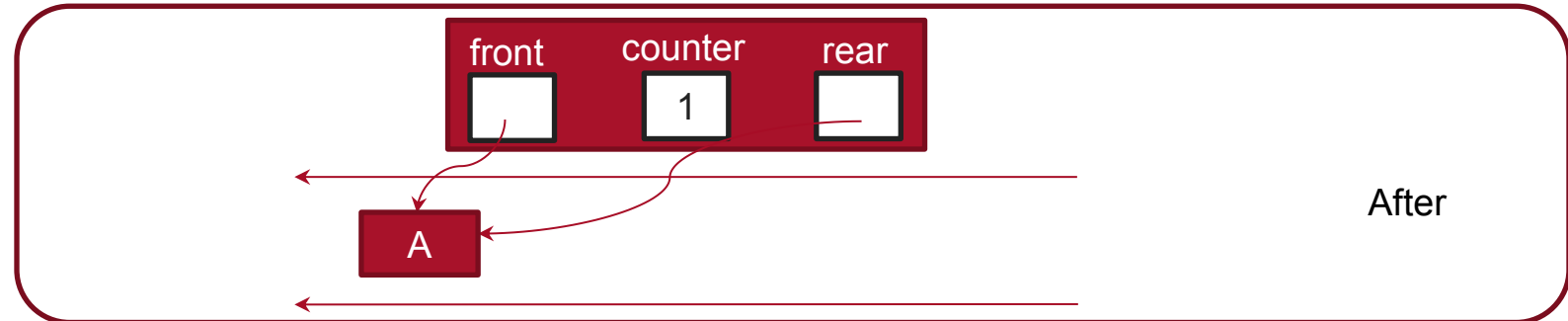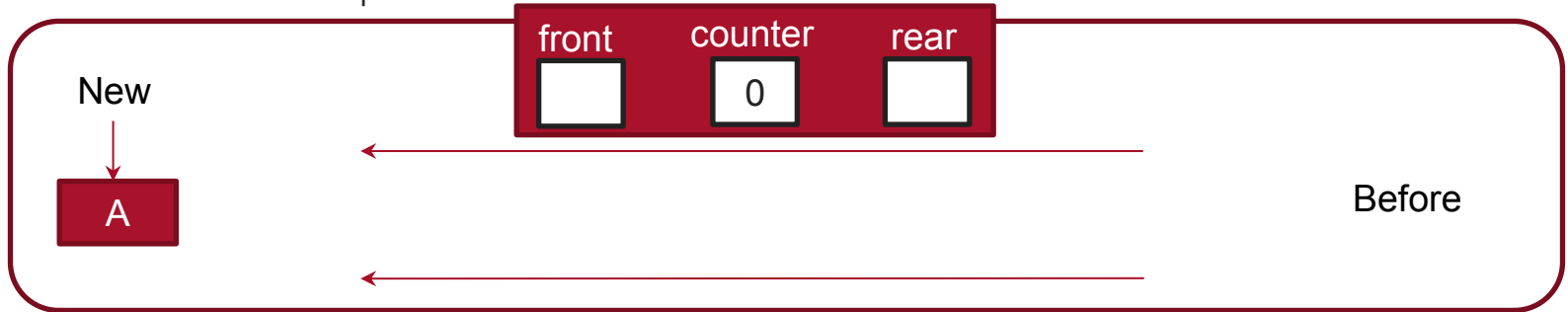
Create Queue

◉ Algorithm to Create Queue:

Step 1: Start

Step 2: Allocate queue head

Step 3: Set queue front to null

Step 4: Set queue rear to null

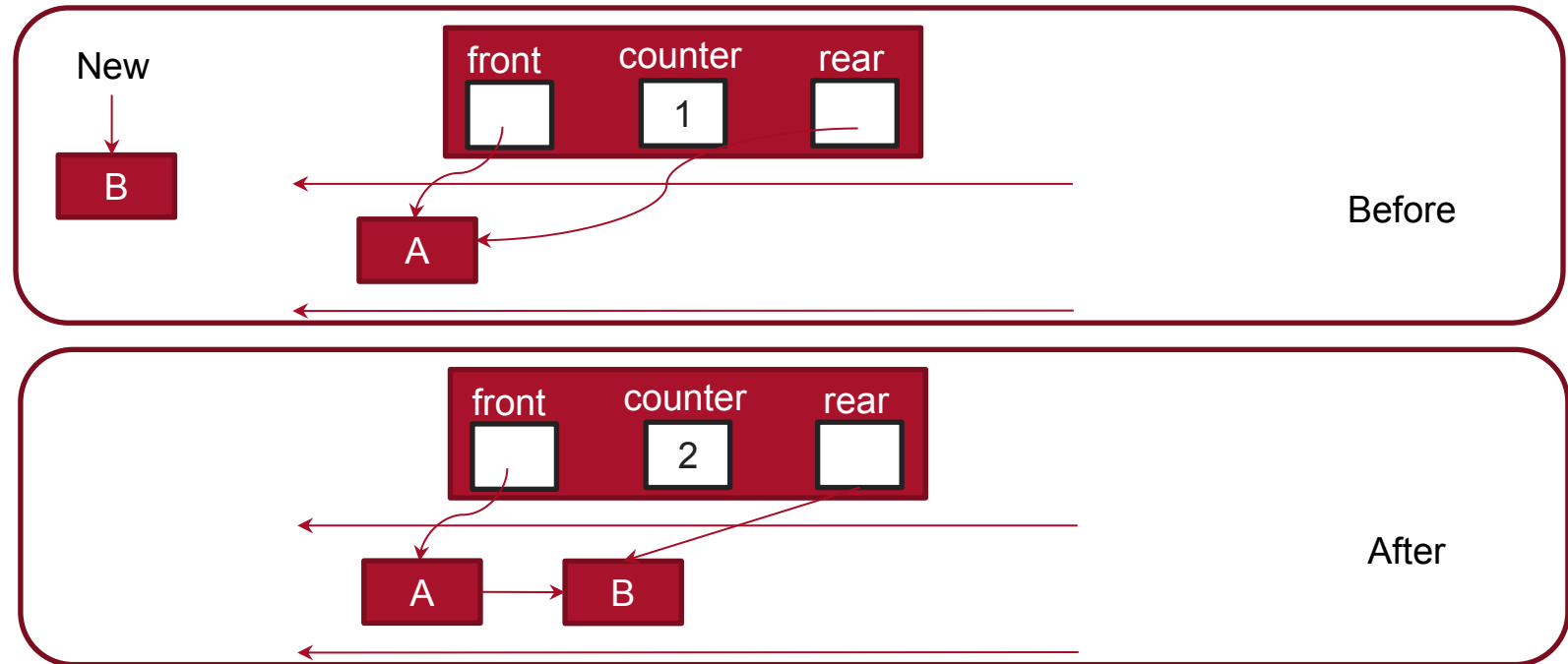Step 5: Set queue count to 0

Step 6: Return queue head

Step 7: End

Enqueue: There are 2 cases:

1. **Insert into an empty queue**: When we insert the data the queue's front and rear pointers must both be set to point to the new node.

2. **Insert into a queue with data**: We must point both the link field of the last node and the rear pointer to the new node.

Implementation of Queue (Cont.)

Enqueue: Algorithm to Enqueue:

Step 1: Start

Step 2: If queue is full return False else goto step 3

Step 3: Allocate new node and move the new data to new node data.

Step 4: Set new node next to null pointer.

Step 5: If queue is empty goto step 6 else goto step 7

Step 6: Set queue front to address of new node.

Step 7: Set next pointer of rear node to address of new node
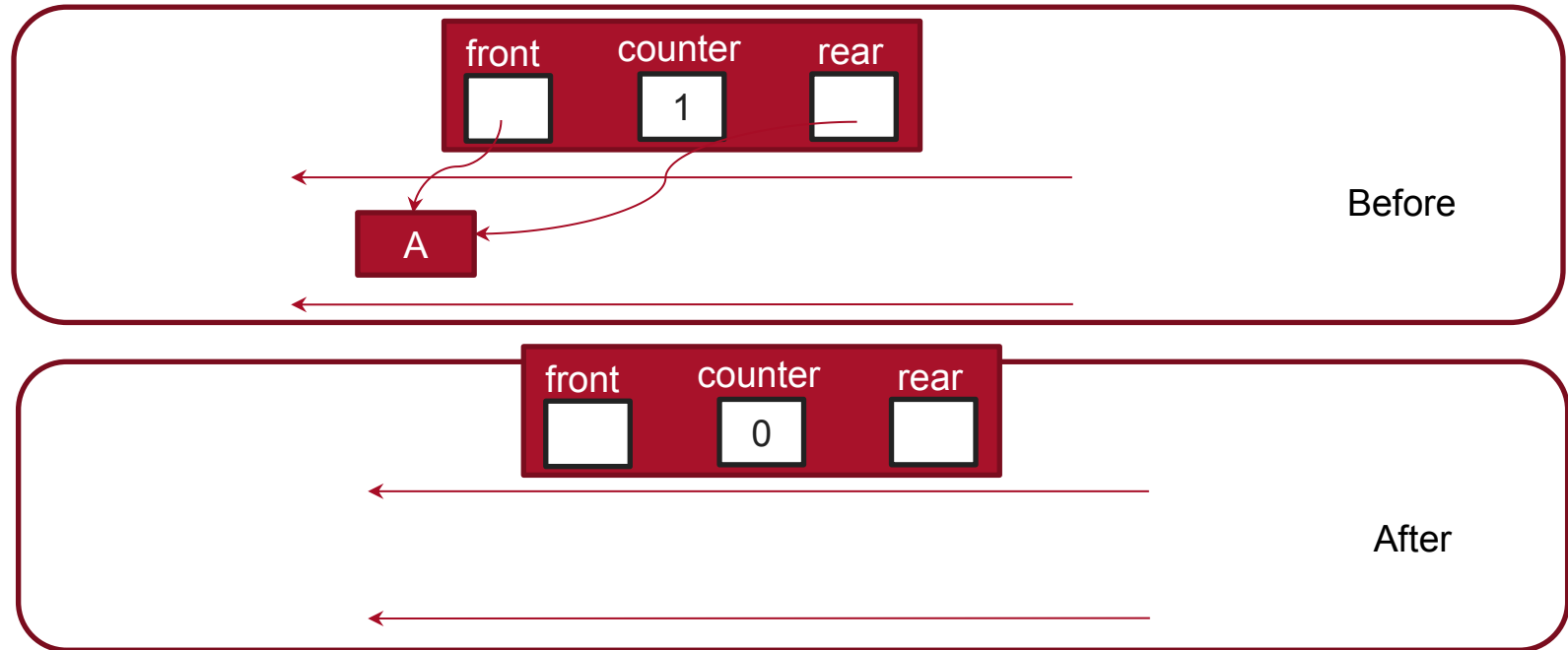
Step 8: Set queue rear to address of new node

Step 9: Increment the queue counter.

Step 10: End

Dequeue: There are 2 cases:
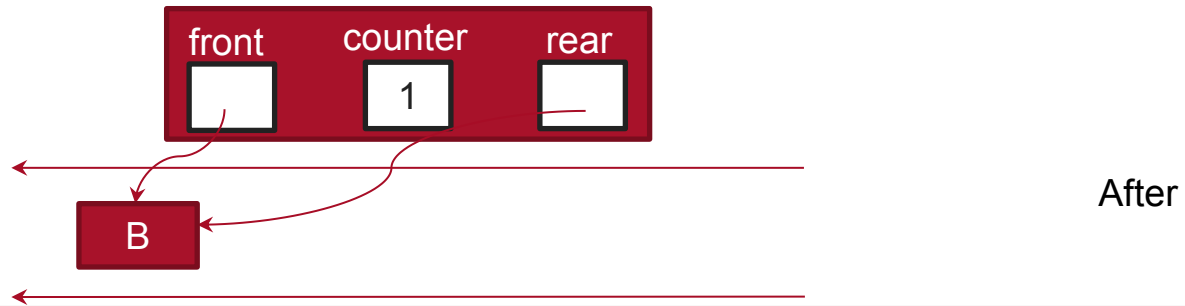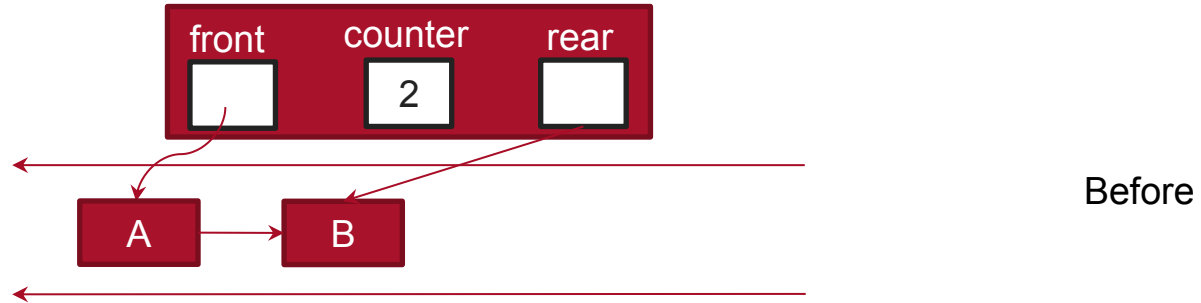
1. **Delete only item in queue**:

2. Delete item at front of queue:

Before

After

Implementation of Queue (Cont.)

Dequeue: Algorithm to dequeue:

Step 1: Start

Step 2: If queue is empty return False else goto step 3

Step 3: Move the front data to item

Step 4: If only one node present goto step 5 else goto step 6

Step 5: Set queue rear to null.

Step 6: Set queue front to queue front next.

Step 7: Decrement the queue counter.

Step 8: End

Implementation of Queue (Cont.)

Retrieving Queue Data: The only difference between the two retrieve queue operations— queue front and queue rear is which pointer is used, front or rear.

Queue Front Algorithm

Step 1: Start

Step 2: If queue is empty return False else goto Step 3.

Step 3: Return the data at front of the queue.

Step 4: Stop

Queue Rear Algorithm

Step 1: Start

Step 2: If queue is empty return False else goto Step 3.

Step 3: Return the data at rear of the queue.

Step 4: Stop

Implementation of Queue (Cont.)

Empty Queue: Empty queue returns true if the queue is empty and false if the queue contains data.

Algorithm

Step 1: Start

Step 2: If queue count=0 return True else return False

Step 3: Stop

Full Queue Algorithm

Step 1: Start

Step 2: : If memory not available return True else return False

Step 3: Stop

Implementation of Queue (Cont.)

Queue count returns the number of elements currently in the queue by simply returning the count found in the queue head node.

Algorithm

Step 1: Start

Step 2: Return queue count.

Step 3: Stop

Destroy Queue: It deletes all data in the queue.

Algorithm

Step 1: Start

Step 2: : If queue is not empty goto Step 3 else goto Step 5.

Step 3: Iterate over queue till it is not empty and delete the front node.

Step 4: Delete head structure.

Step 4: Stop

Queue Application

Below are some of the common places where queues are used:

◉ **Message Queues**: Used for communication between processes.

◉ **Process Scheduling**: All the processes running on your computer now, first wait in a queue called ready queue, waiting to be executed by your processor.

◉ **Data Buffers:** A buffer implements a queue and is typically used in communication when there is difference between the rate at which data is received and the rate at which it can be processed.

Exercise

◉ Categorizing the data of queue.

◉ Given N integers, the task is to insert those elements in the queue. Also, given M integers, task is to find the frequency of each number in the Queue.

◉ CPU Page Replacement Algorithm

Thank You