List - Doubly Linked List, Stack and Queue

Data Structures and Algorithms

Dept. Computer Science
Faculty of Computer Science and Engineering
Ho Chi Minh University of Technology, VNU-HCM

Overview

1 Other linked lists

Doubly Linked List

Circularly Linked List

2 Stacks

Basic operations of Stacks

Implement a stack in C/C++

Applications of Stack

3 Queues

Basic operations of Queues

Implement a queue in C/C++

Applications of Queue

Other linked lists

Doubly Linked List (DLL)

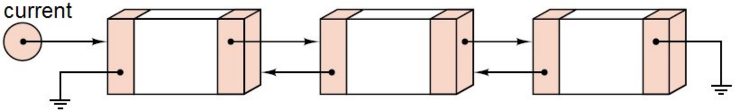


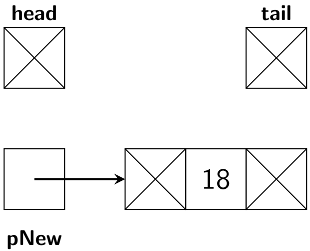
Figure: Doubly Linked List allows going forward and backward.

Structure

```
node                              list
    data <dataType>                   head <pointer>
    next <pointer>                    tail <pointer>
    prev <pointer>                    count <integer>
end node                          end list
```

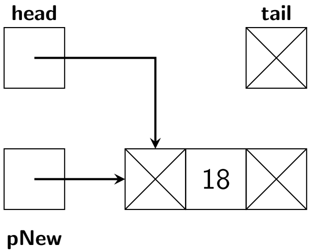
DLL: Insert into an empty list

DLL: Insert into an empty list



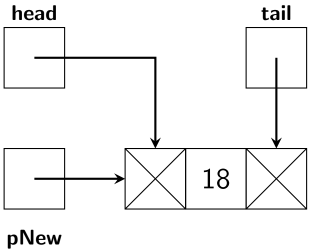
- Allocate pNew to the node with inserting element.

DLL: Insert into an empty list



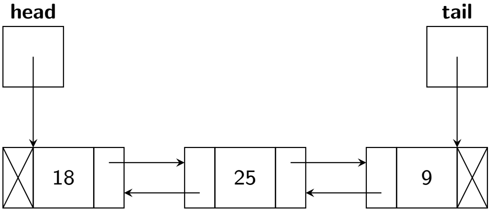
- Allocate `pNew` to the node with inserting element.
- `head = pNew`

DLL: Insert into an empty list

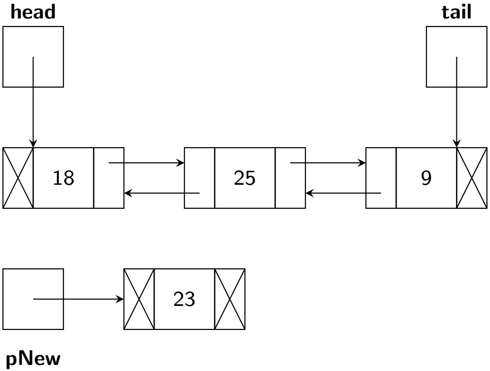


- Allocate `pNew` to the node with inserting element.
- `head = pNew`
- `tail = pNew`

DLL: Prepend to a non-empty list

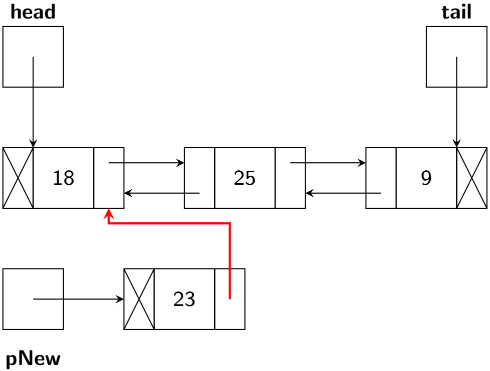


DLL: Prepend to a non-empty list



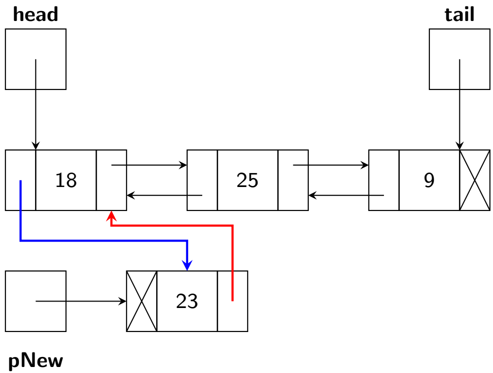
- Allocate `pNew` to the node with inserting element.

DLL: Prepend to a non-empty list



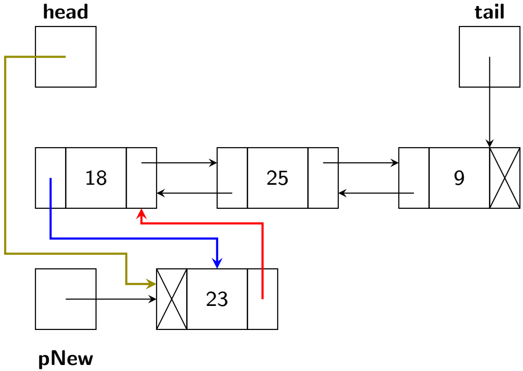
- Allocate `pNew` to the node with inserting element.
- `pNew->next = head`

DLL: Prepend to a non-empty list



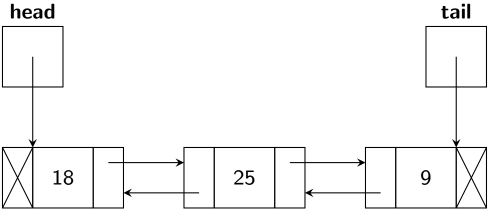
- Allocate pNew to the node with inserting element.
- pNew->next = head
- head->prev = pNew

DLL: Prepend to a non-empty list

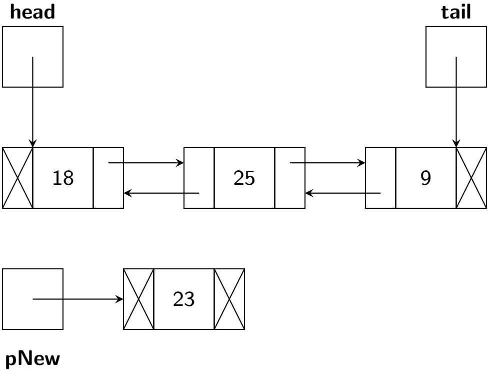


- Allocate pNew to the node with inserting element.
- pNew->next = head
- head->prev = pNew
- head = pNew

DLL: Append to a non-empty list

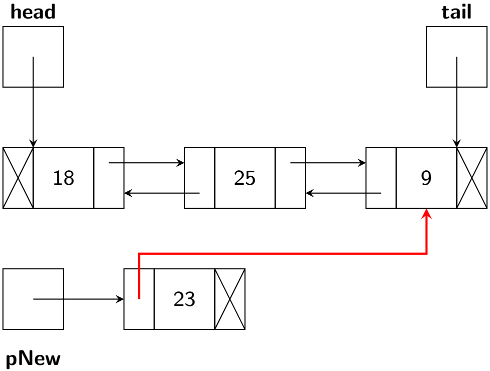


DLL: Append to a non-empty list



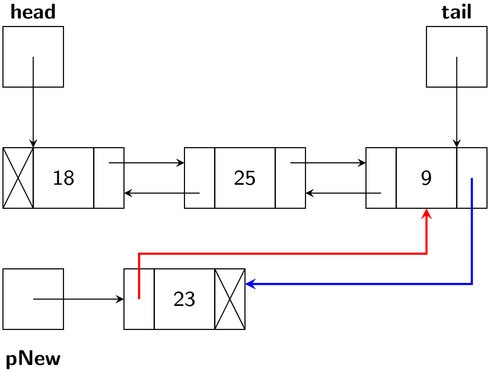
- Allocate `pNew` to the node with inserting element.

DLL: Append to a non-empty list



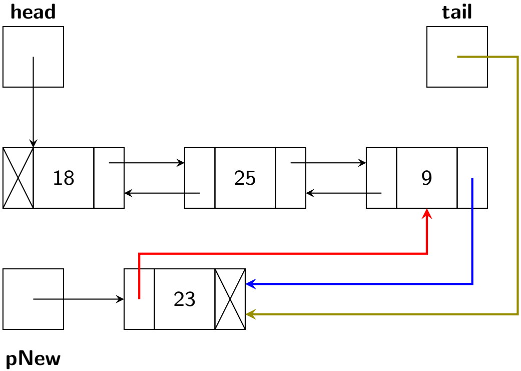
- Allocate `pNew` to the node with inserting element.
- `pNew->prev = tail`

DLL: Append to a non-empty list



- Allocate pNew to the node with inserting element.
- pNew->prev = tail
- tail->next = pNew

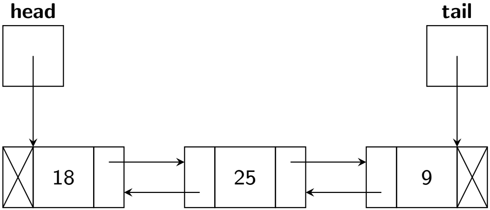
DLL: Append to a non-empty list



- Allocate `pNew` to the node with inserting element.
- `pNew->prev = tail`
- `tail->next = pNew`
- `tail = pNew`

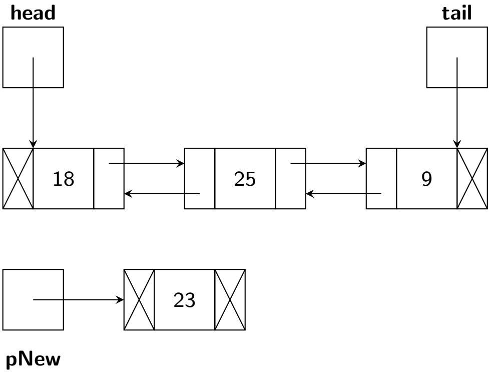
DLL: Insert into a non-empty list at specific index

Insert at index 2.



DLL: Insert into a non-empty list at specific index

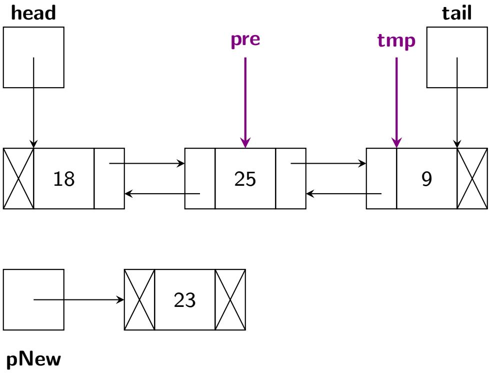
Insert at index 2.



- Allocate `pNew` to the node with inserting element.

DLL: Insert into a non-empty list at specific index

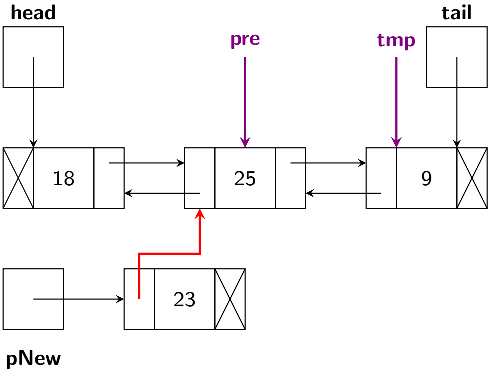
Insert at index 2.



- Allocate `pNew` to the node with inserting element.
- Find `pre, tmp` points the node at `index - 1` and `index`.

DLL: Insert into a non-empty list at specific index

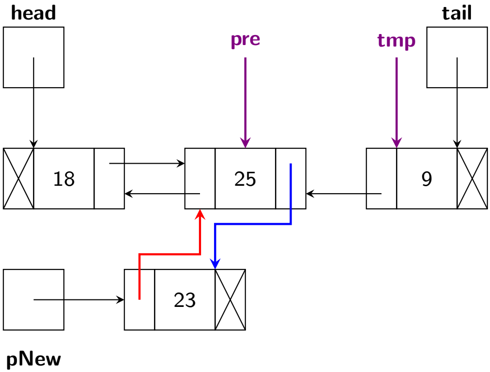
Insert at index 2.



- Allocate `pNew` to the node with inserting element.
- Find `pre, tmp` points the node at `index - 1` and `index`.
- `pNew->prev = pre`

DLL: Insert into a non-empty list at specific index

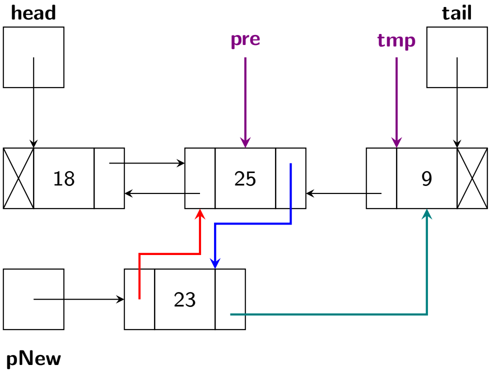
Insert at index 2.



- Allocate `pNew` to the node with inserting element.
- Find `pre`, `tmp` points the node at `index - 1` and `index`.
- `pNew->prev = pre`
- `pre->next = pNew`

DLL: Insert into a non-empty list at specific index

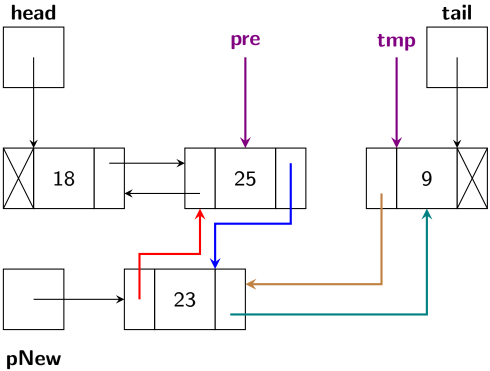
Insert at index 2.



- Allocate pNew to the node with inserting element.
- Find pre, tmp points the node at index - 1 and index.
- pNew->prev = pre
- pre->next = pNew
- pNew->next = tmp

DLL: Insert into a non-empty list at specific index

Insert at index 2.

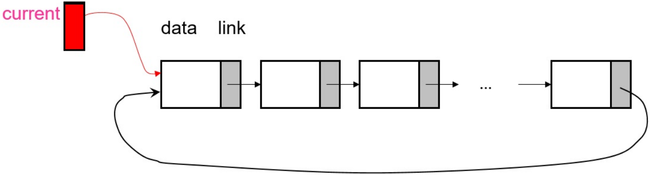


- Allocate `pNew` to the node with inserting element.
- Find `pre, tmp` points the node at `index - 1` and `index`.
- `pNew->prev = pre`
- `pre->next = pNew`
- `pNew->next = tmp`
- `tmp->prev = pNew`

DLL: Other operations

Your exercise: Use the conceptual idea to implement the doubly linked list.

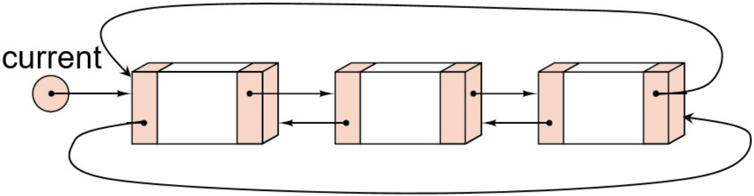
Circularly Linked List



Structure

```
node                    list
    data <dataType>       current <pointer>
    link <pointer>      end list
end node
```

Double circularly Linked List



Structure

```
node
    data <dataType>
    next <pointer>
    previous <pointer>
end node
```

```
list
    current <pointer>
end list
```

STACK

Linear List Concepts

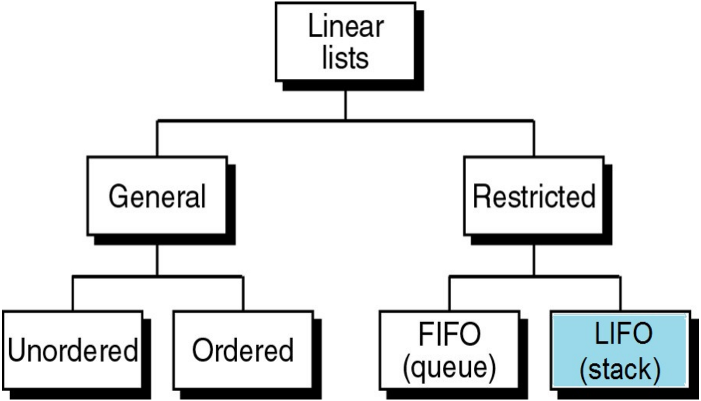
General list:

- No restrictions on which operation can be used on the list.
- No restrictions on where data can be inserted/deleted.

Restricted list:

- Only some operations can be used on the list.
- Data can be inserted/deleted only at the ends of the list.

Linear list concepts



Stack

Definition

A stack of elements of type T is a finite, ordered sequence of elements of T, in which all insertions and deletions are restricted to one end, called the top.

Stack is a Last In - First Out (LIFO) data structure.
LIFO: The last item put on the stack is the first item that can be taken off.

Basic operations of Stacks

Basic operations of Stacks

Basic operations:

- `Construct` a stack, leaving it empty.
- `Push` an element: put a new element on to the top of the stack.
- `Pop` an element: remove the top element from the top of the stack.
- `Top` an element: retrieve the top element.

Basic operations of Stacks

Extended operations:

- Determine whether the stack is `empty` or not.
- Determine whether the stack is `full` or not.
- Find the `size` of the stack.
- `Clear` the stack to make it empty.

Basic operations of Stacks: Push

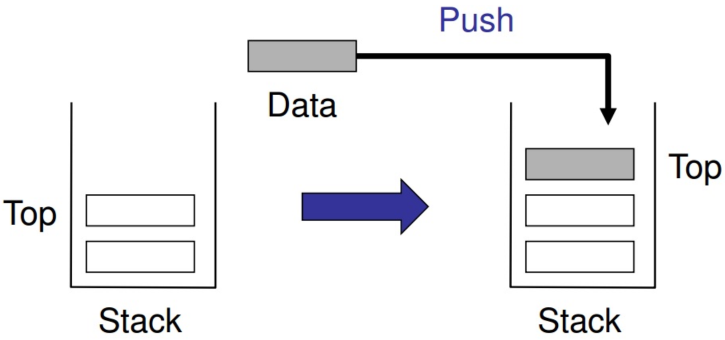


Figure: Successful Push operation

Basic operations of Stacks: Push

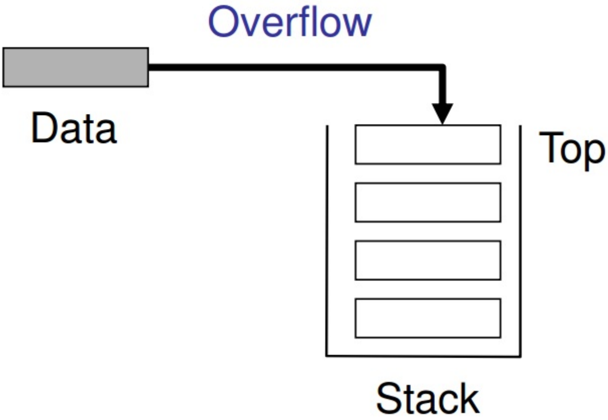


Figure: Unsuccessful Push operation. Stack remains unchanged.

Basic operations of Stacks: Pop

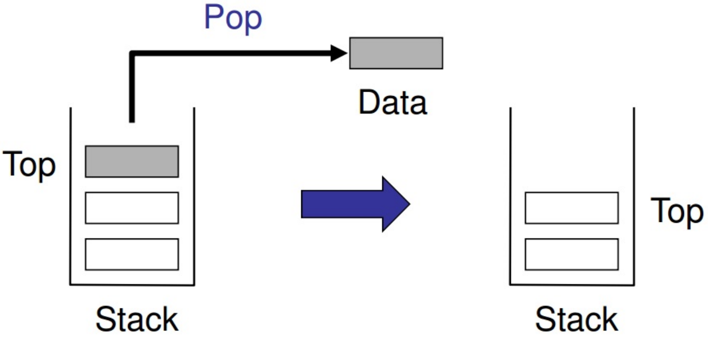


Figure: Successful Pop operation

Basic operations of Stacks: Pop

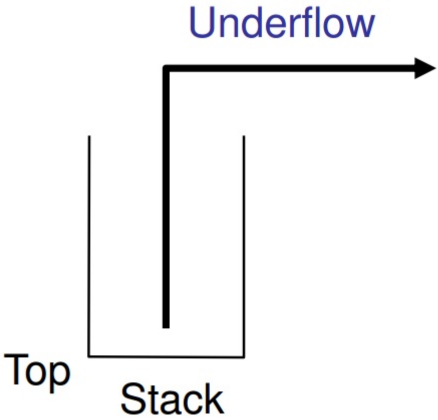


Figure: Unsuccessful Pop operation. Stack remains unchanged.

Basic operations of Stacks: Top

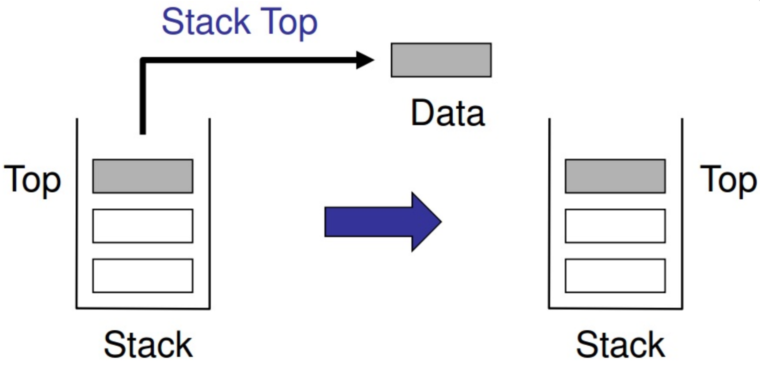


Figure: Successful Top operation. Stack remains unchanged.

Basic operations of Stacks: Top

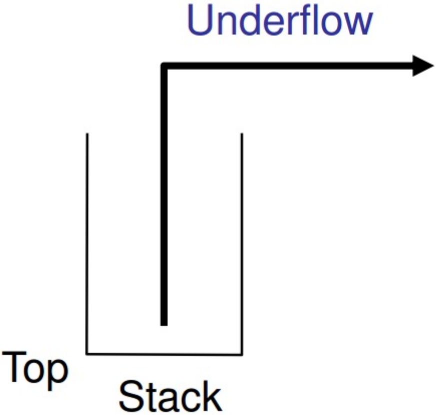


Figure: Unsuccessful Top operation. Stack remains unchanged.

Implement a stack in C/C++

Implement a stack in C/C++

Using List ADT to implement a stack in C/C++:

- Use array list, singly linked list or double linked list to implement a stack.
- Evaluate the difference in complexity for each implementation.

Implement a stack in C/C++

```cpp
class IntStack {
private:
    IntSLinkedList* list;

public:
    IntStack() {
        this->list = new IntSLinkedList();
    }

    ~IntStack() {
        this->list->clear();
        delete list;
    }

    void push(int element) {
        // this->list->add(0, element);
        this->list->add(element);
    }

    int pop() {
        // return this->list->removeAt(0);
        int size = this->list->size();
        return this->list->removeAt(size - 1);
    }
};
```

Implement a stack in C/C++

```
class IntStack {
private:
    IntSLinkedList* list;

public:


    int peek() {
        // return this->list->get(0);
        int size = this->list->size();
        return this->list->get(size - 1);
    }
};
```

Implement a stack in C/C++

```cpp
class IntStack {
private:
    IntSLinkedList* list;

public:
    bool empty() {
        return this->list->empty();
    }

    bool contains(int element) {
        return this->list->contains(element);
    }
};
```

Applications of Stack

Applications of Stack

- Reversing data items
  - Reverse a list
  - Convert Decimal to Binary
- Parsing
  - Brackets Parse
- Postponement of processing data items
  - Infix to Postfix Transformation
  - Evaluate a Postfix Expression
- Backtracking
  - Goal Seeking Problem
  - Knight's Tour
  - Exiting a Maze
  - Eight Queens Problem

QUEUE

Queue

Definition

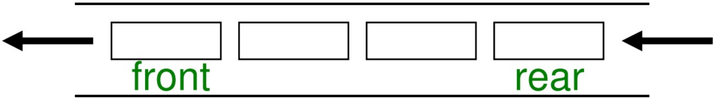
A queue of elements of type T is a finite sequence of elements of T, in which data can only be inserted at one end called the rear, and deleted from the other end called the front.

Queue is a First In - First Out (FIFO) data structure. FIFO: The first item stored in the queue is the first item that can be taken out.

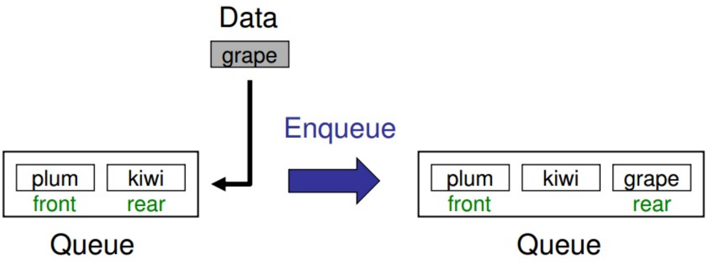
Basic operations of Queues

Basic operations:

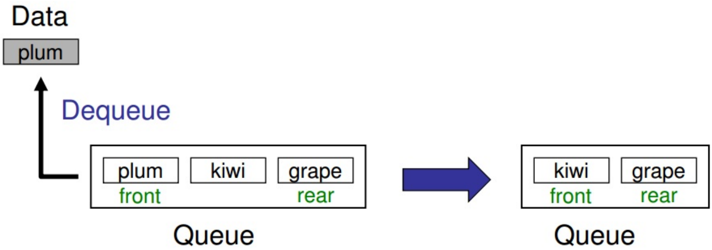
- `Construct` a queue, leaving it empty.
- `Enqueue`: put a new element in to the rear of the queue.
- `Dequeue`: remove the first element from the front of the queue.
- `Queue Front`: retrieve the front element.
- `Queue Rear`: retrieve the rear element.



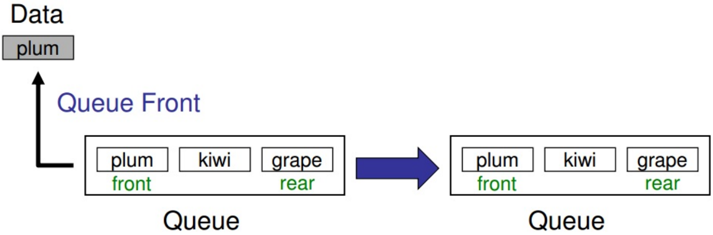
Basic operations of Queues: Enqueue



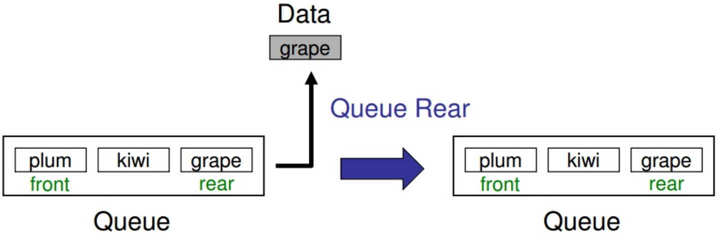
Basic operations of Queues: Dequeue



Basic operations of Queues: Queue Front



Basic operations of Queues: Queue Rear



Implement a queue in C/C++

Implement a queue in C/C++

Using List ADT to implement a queue in C/C++:

- Use array list, singly linked list or double linked list to implement a queue.
- Evaluate the difference in complexity for each implementation.

Implement a queue in C/C++

```
class IntQueue {
private:
    IntSLinkedList* list;

public:
    IntQueue() {
        this->list = new IntSLinkedList();
    }

    ~IntQueue() {
        this->list->clear();
        delete list;
    }
};
```

Implement a queue in C/C++

```cpp
class IntQueue {
private:
    IntSLinkedList* list;

public:
    void enqueue(int element) {
        // this->list->add(0, element);
        this->list->add(element);
    }

    int dequeue() {
        // int size = this->list->size();
        // return this->list->removeAt(size - 1);
        return this->list->removeAt(0);
    }
};
```

Implement a queue in C/C++

```
class IntStack {
private:
    IntSLinkedList* list;

public:
    int front() {
        // int size = this->list->size();
        // return this->list->get(size - 1);
        return this->list->get(0);
    }

    int rear() {
        // return this->list->get(0);
        int size = this->list->size();
        return this->list->get(size - 1);
    }

    bool empty() {
        return this->list->empty();
    }

    bool contains(int element) {
        return this->list->contains(element);
    }
};
```

Applications of Queue

Applications of Queue

- Polynomial Arithmetic
- Categorizing Data
- Evaluate a Prefix Expression
- Radix Sort
- Queue Simulation

THANK YOU.