LINKED DATA STRUCTURES PART 1

S. Trowbridge 2020

Namespaces

Concept

a namespace groups classes, objects and functions under a similar name

using namespace X specifies the namespace X for the current scope

namespace for commands is resolved using the scope resolution operator
the namespace prefix is required if a namespace has not been specified

Example

```
namespace myvars {          // create a namespace
    int x = 5;              // create a variable in this namespace
}
namespace yourvars {        // create a namespace
    string x = “Hi”;        // create a variable in this namespace
}
cout << myvars::x;          // accesses the myvars namespace and prints 5
cout << yourvars::x;        // accesses the yourvars namespace and prints Hi
```

ex0a_namespaces.cpp

Standard Namespace

Concept

using namespace std at the top of a program sets std as the global namespace

proper C++ programs do not specify a global namespace for extensibility
in this situation the std:: prefix is required for all standard commands

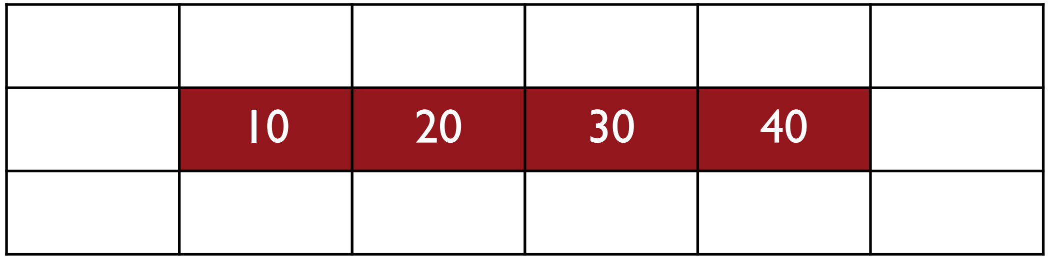
Example

```
std::string s = “Hi”;
std::cout << s << std::endl;
```

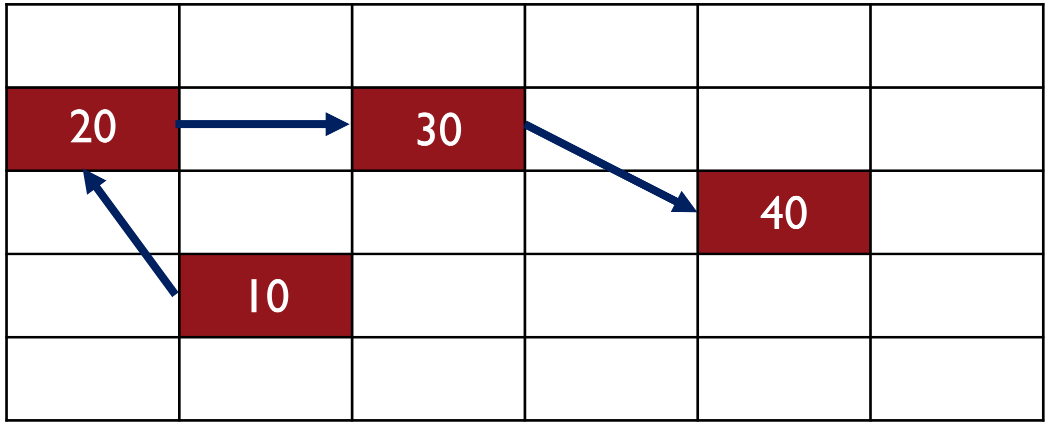
ex0b_namespaces.cpp

Linked Structures vs. Array

Array a contiguous block of memory where memory address indicate order

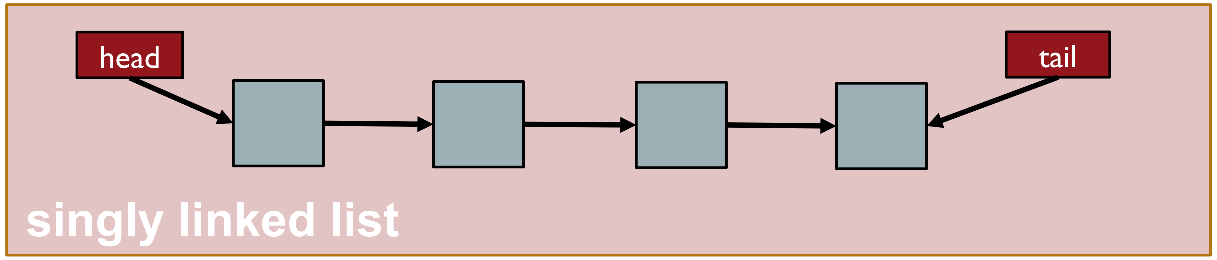


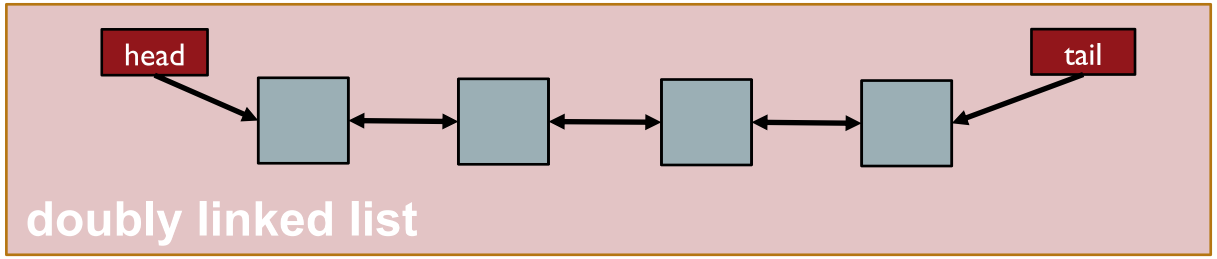
Linked List a non-contiguous block of memory linked by pointers indicating order

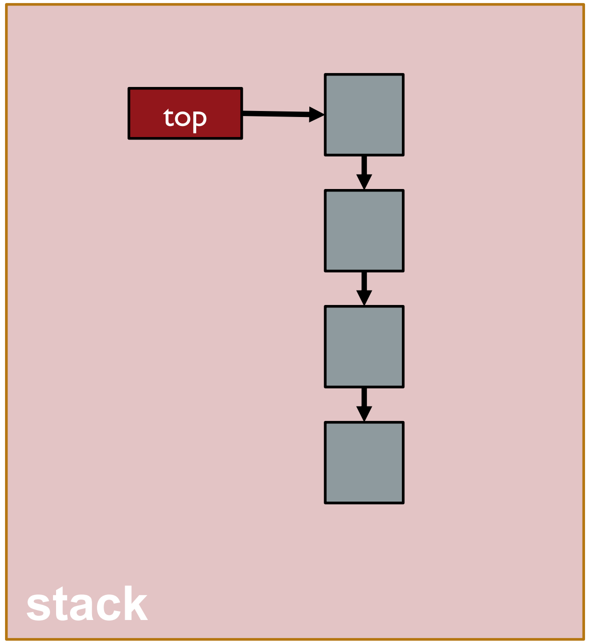


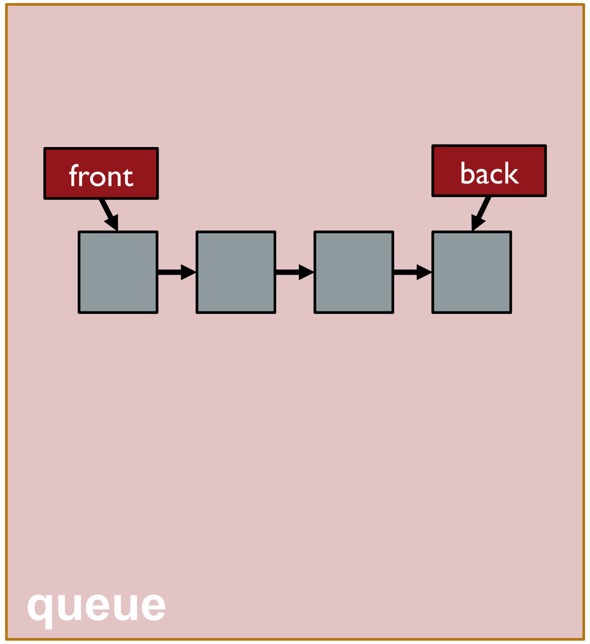
Linked Data Structures

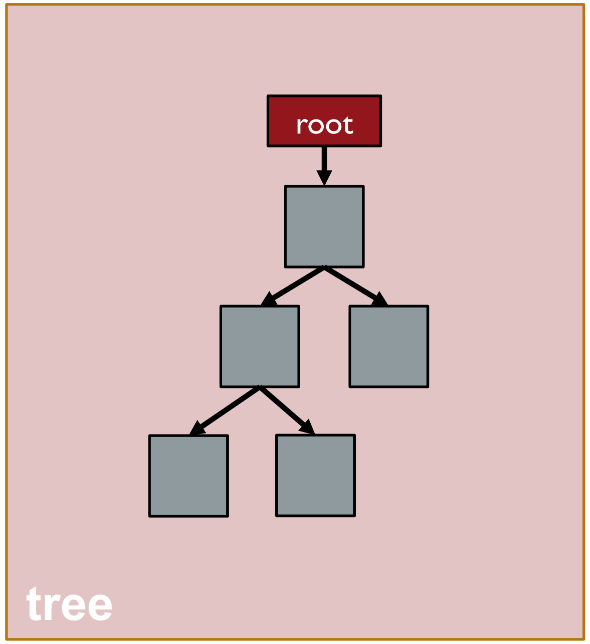
Linked structures including linked lists, stacks, queues, trees and graphs.

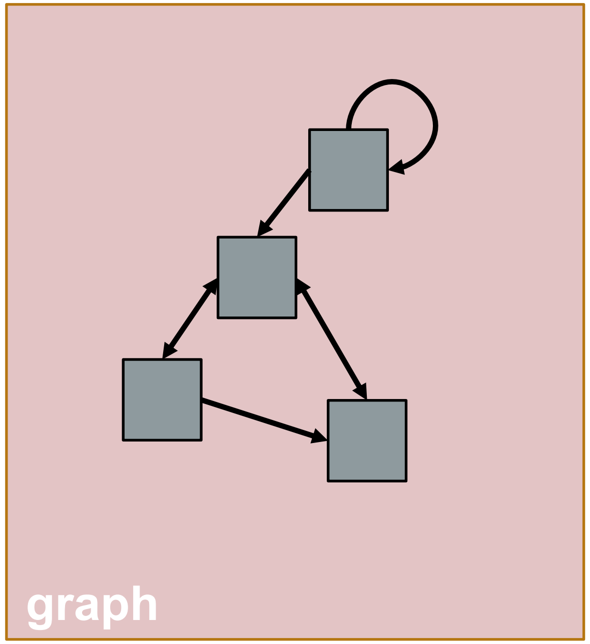












Linked Data Structures

In C++ linked structures are coded using pointers to link nodes of data.

Linked structures are implemented in a few ways including:

1. a node struct with associated functions (simplest)
2. a node class in composition with a container class
3. template node, container and iterator classes (most complex)

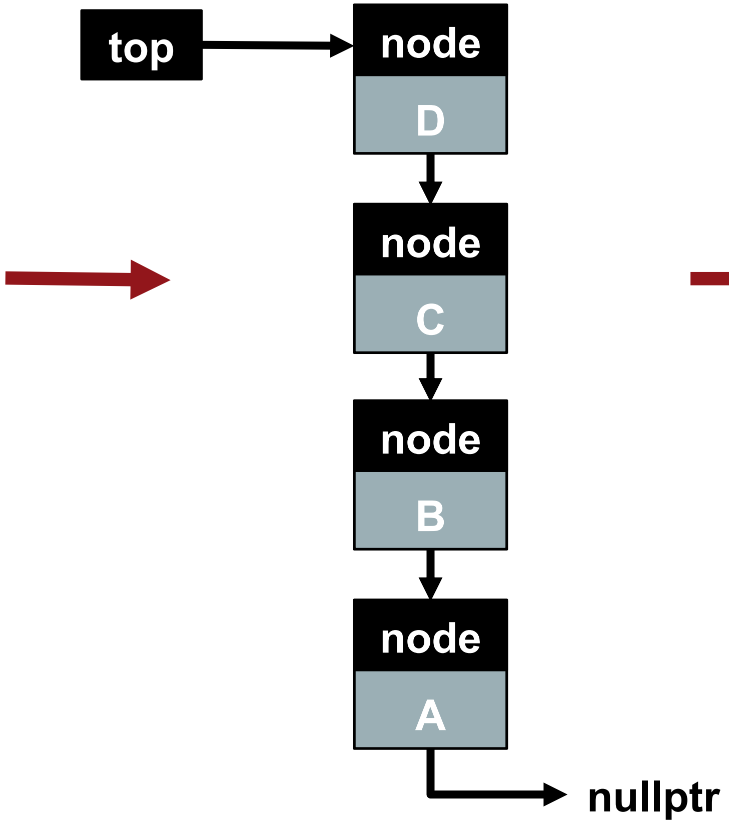
As an example of container design you can reference C++ Standard Containers:
https://www.cplusplus.com/reference/stl/

Stack

Concept a data structure stored in first-in-last-out (FILO) order

Top a pointer to the top node in the stack

Nodes enter:
A B C D



Nodes exit:
D C B A

Stack Node

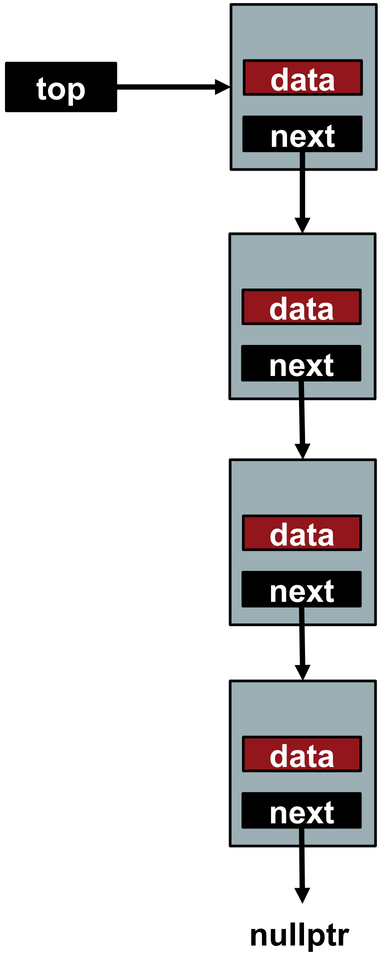
Concept stack nodes contain two variables, data and next

Data the information to be stored

Next a pointer to the next node in the stack

Note a new node initially points to nullptr

when a node is added to the stack, it points to the next node in the stack



Stack Node Class

Concept node classes are custom designed for the container they will be used with

Example a node class to store integers in a stack

```
class Node {
public:
    int data;                                            // integer data
    Node *next;                                          // pointer to next node

    Node(const int &d): data(d), next(nullptr) { }       // node constructor
                                                         // next points to nullptr
};
```

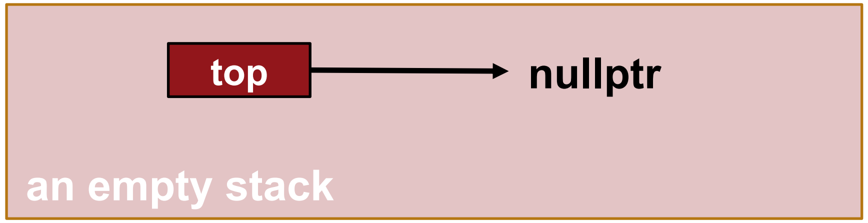
Note this is a simple implementation where the entire class is publicly accessible

Stack Class

Concept construct an empty stack which contains no node objects

Example

```
class Stack {
private:
    Node *topNode;                              // pointer to the top node in the stack
    int size;                                   // optional parameter to track # of nodes
public:
    Stack(): topNode(nullptr), size(0) { }      // top set to nullptr (empty list)
    additional functions
};
```



ex1_stack

Common Stack Functions

empty	return true if the stack is empty (stack contains no nodes)
push	add a node to the top of the stack
pop	remove the top node of the stack
top	return a reference to the data stored in the top node

Stack: Push

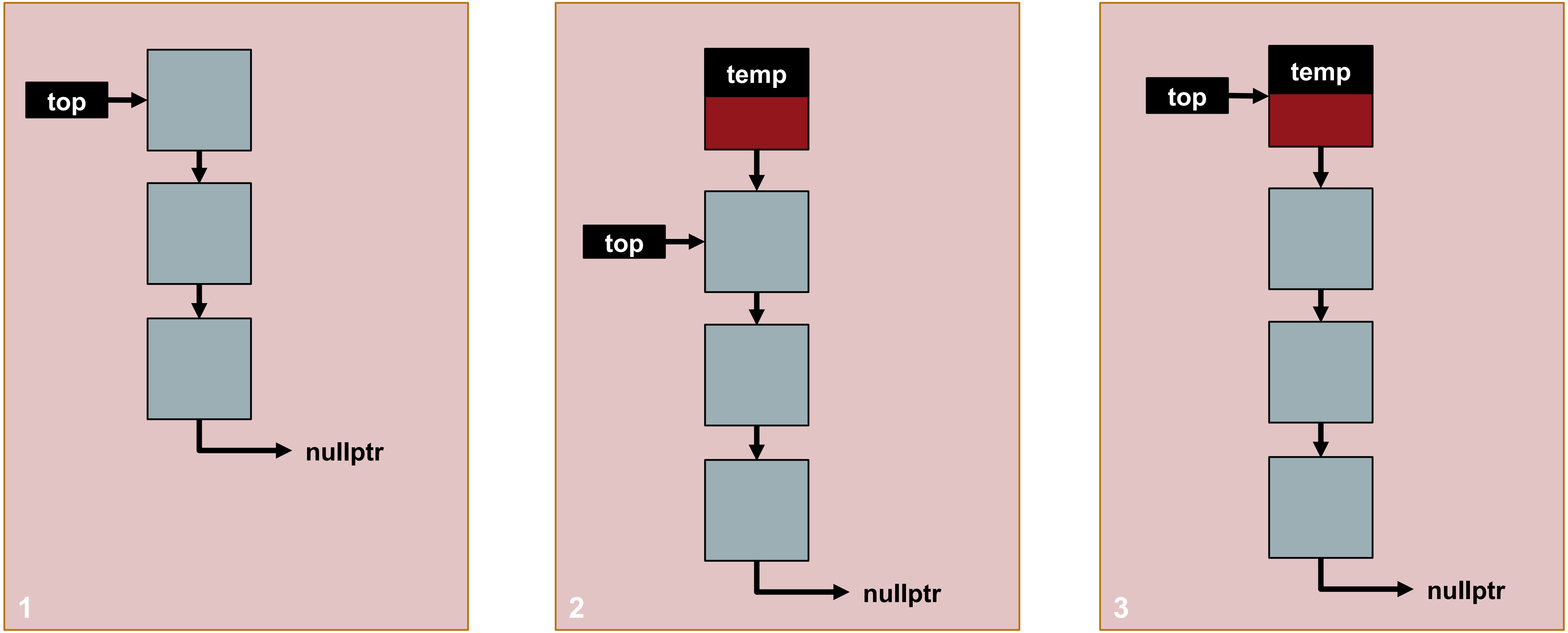
Concept add a node to the top of the stack

Example

```
void push(int n) {
        Node *temp = new Node(n);           // create a new node, temp
        temp->next = topNode;               // point temp->next to the current top node
        topNode = temp;                     // point topNode to temp
        ++size;                             // increment size
}
```

Stack: Push

Push: **add a node to the top of the stack**



Stack: Pop

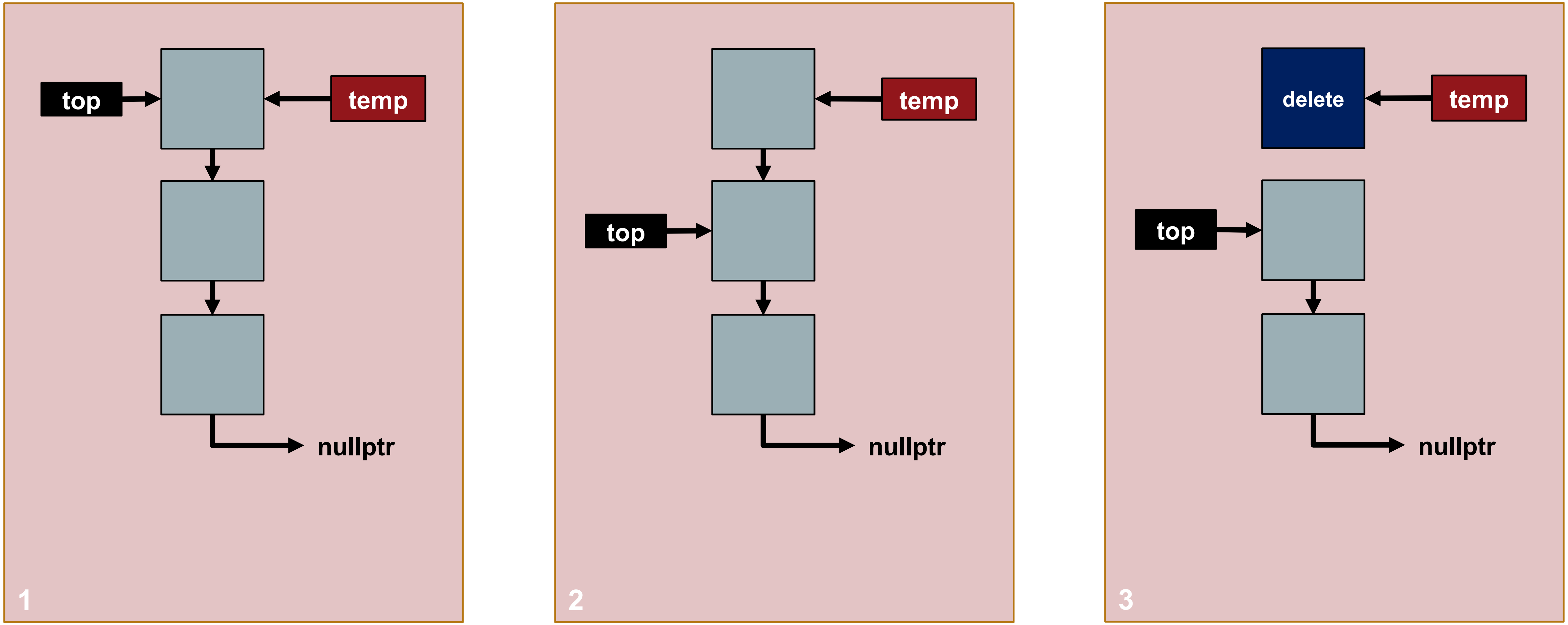
Concept remove the top node from the stack

Example

```
void pop(int n) {
    if(top == nullptr) { return; }          // stack empty, exit function
    Node *temp = topNode;                   // point temp to the current top node
    topNode = topNode->next;                // point topNode-> to the second node
    delete temp;                            // delete the original top node
    --size;                                 // decrement size
}
```

Stack: Pop

Push: remove the top node from the stack

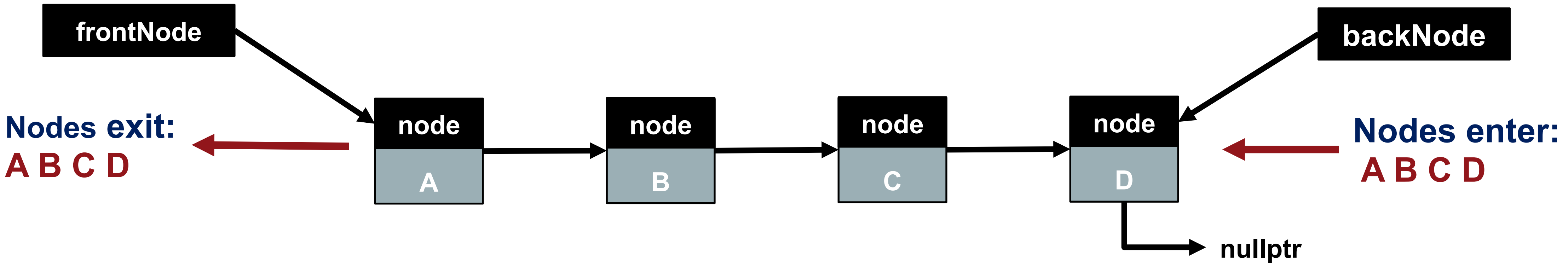


Queue

Concept a data structure stored in first-in-first-out (FIFO) order

frontNode a pointer to the front node in the stack

backNode a pointer to the back node in the stack

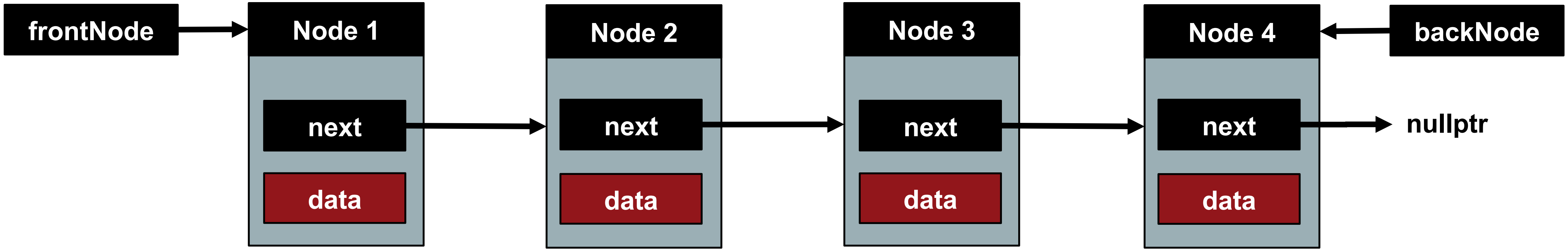


Queue Node

Concept queue nodes contain two variables, data and next

Data the information to be stored

Next a pointer to the next node in the queue



Queue Node Class

Concept node classes are custom designed for the container they will be used with

Example a node class to store integers in a queue

```
class Node {
public:
    int data;                                          // integer data
    Node *next;                                        // pointer to next node

    Node(const int &d): data(d), next(nullptr) { }     // node constructor
                                                       // next points to nullptr
};
```

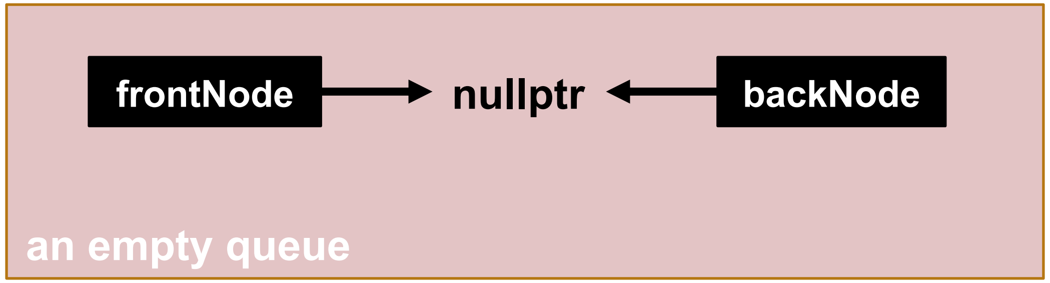
Note this is a simple implementation where the entire class is publicly accessible

Queue Class

Concept construct an empty queue

Example

```
class Queue {
private:
    Node *frontNode;                          // pointer to the first node in the stack
    Node *backNode;                           // pointer to the last node in the queue
    int size;                                 // optional parameter to track # of nodes
public:
    Queue(): frontNode(nullptr),              // top set to nullptr (empty list)
             backNode(nullptr), size(0) { }
    additional functions
};
```



an empty queue

Common Queue Functions

empty	return true if the list is empty (list contains no nodes)
front	return a reference to the first node’s data
back	return a reference to the last node’s data
push	add a node to the back of the queue
pop	remove a node from the front of the queue

ex2_queue

Queue: Push

Concept add a node to the **back** of the queue

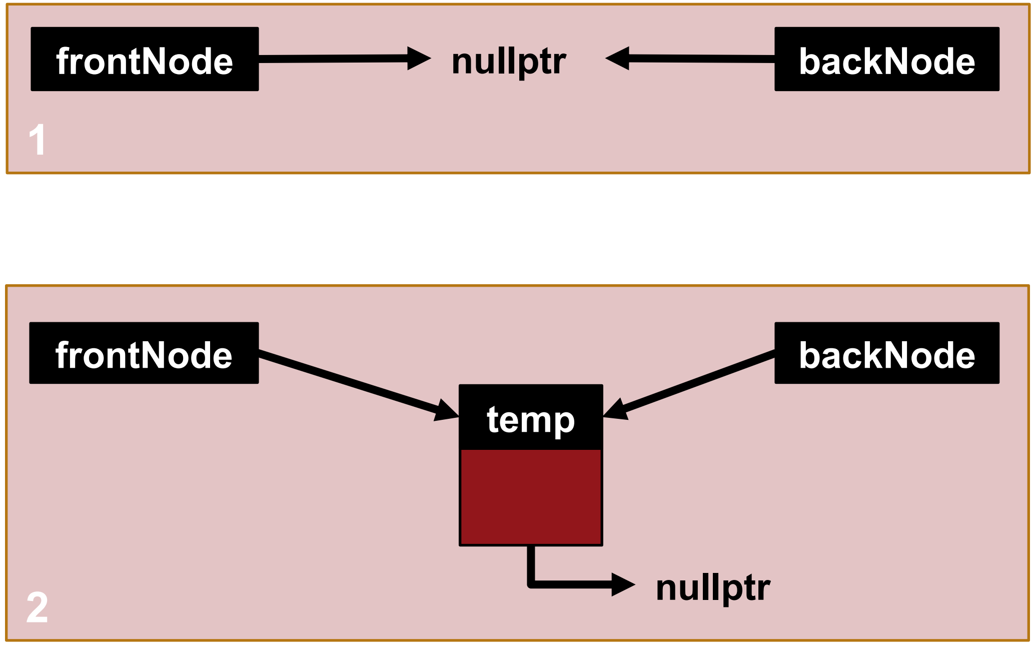
Example

```
void push(int n) {
    Node *temp = new Node(n);          // create a new node, temp
    if(frontNode == nullptr) {         // queue is empty:
        frontNode = temp;              //     set first node as temp
        backNode = temp;               //     set last node as temp
    } else {
        backNode->next = temp;         // point last node to temp
        backNode = temp;               // set temp as the last node
    }
    ++size;                            // increase size
}
```

ex2_queue

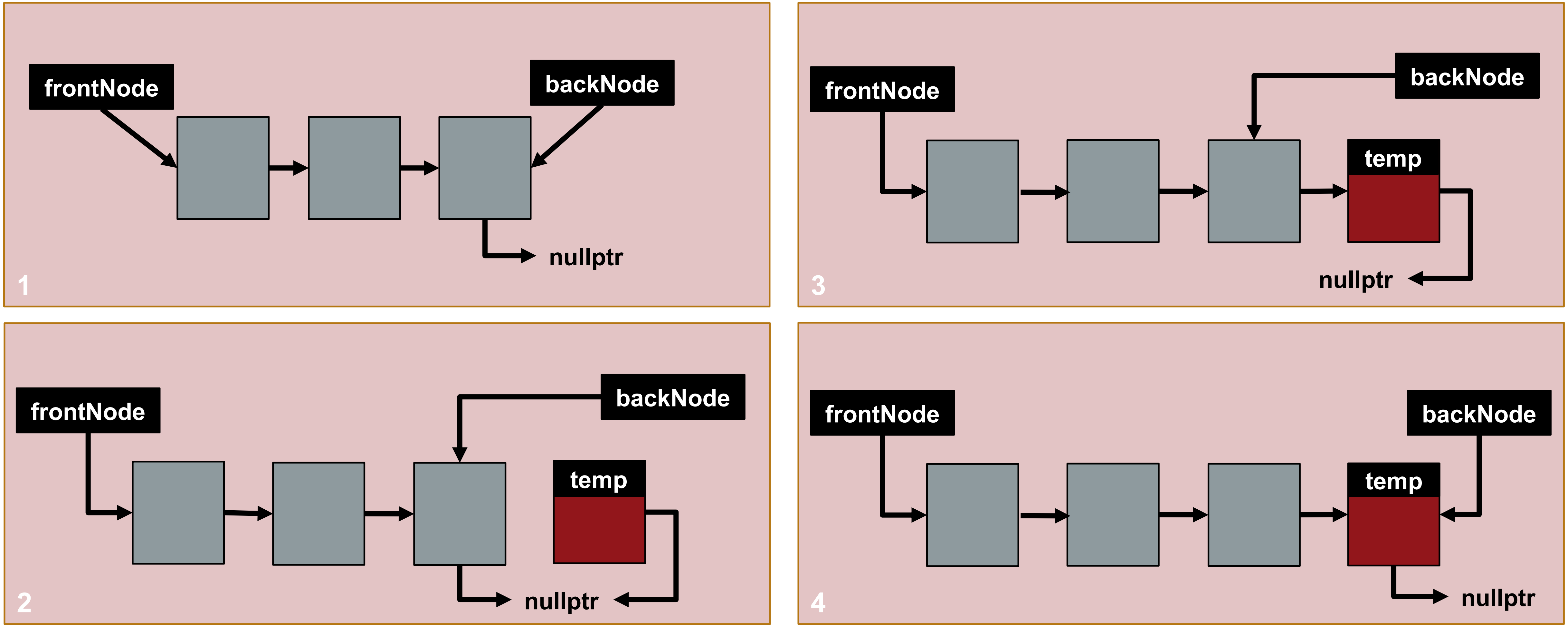
Queue: Push

Case 1: **add a node to the back of an empty queue**



Queue: Push

Case 2: add a node to the back of a non-empty queue



Queue: Pop

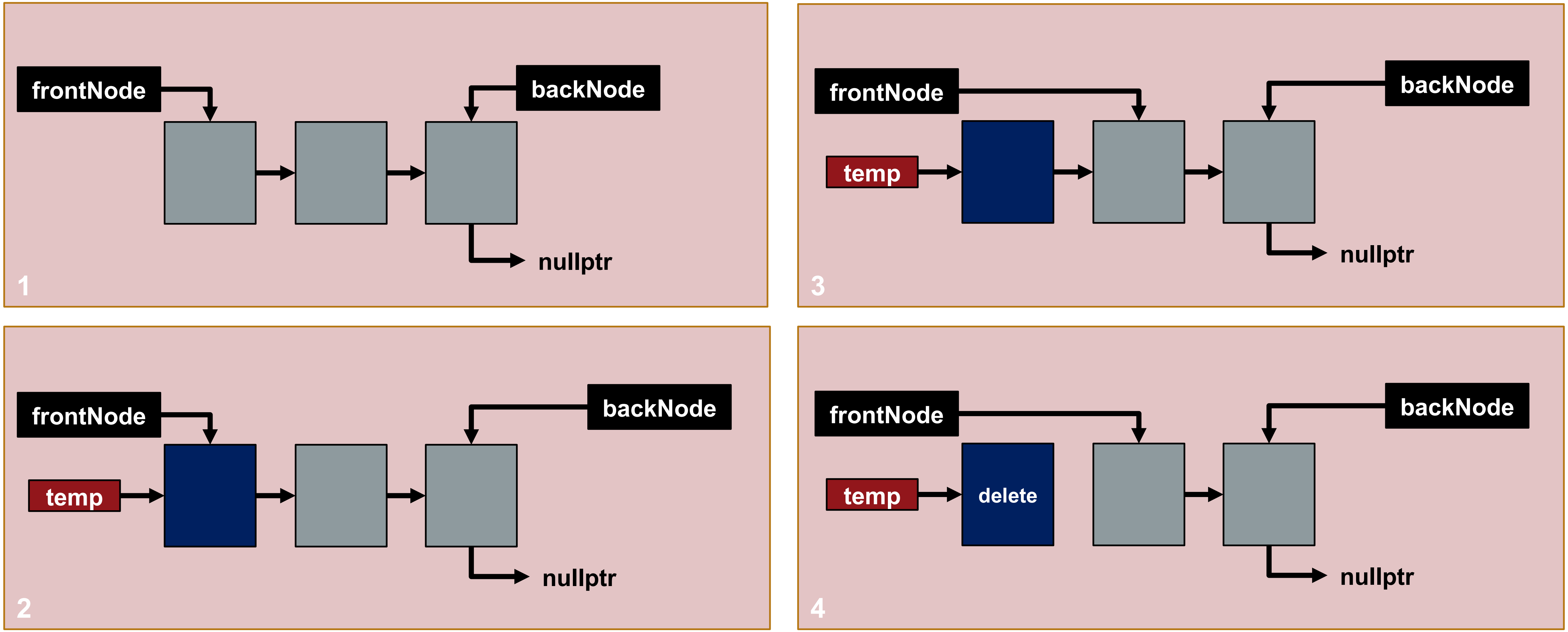
Concept remove the front node from the queue

Example

```
void pop(int n) {
    if(frontNode == nullptr) { return; }    // queue empty, exit function
    Node *temp = frontNode;                 // point temp to the first node
    frontNode = frontNode->next;            // point frontNode to the second node
    delete temp;                            // delete the original first node
    --size;                                 // decrement size
}
```

Queue: Pop

Push: remove the front node from the queue


frontNode
backNode
nullptr
1
frontNode
temp
backNode
nullptr
3
frontNode
temp
backNode
nullptr
2
frontNode
temp
delete
backNode
nullptr
4