

# Stacks and Queues

---

Dr. Robert Amelard  
(adapted from Dr. Igor Ivkovic)

[ramelard@uwaterloo.ca](mailto:ramelard@uwaterloo.ca)

# Objectives

---

- Introduction to Stacks
- Introduction to Queues

# Introduction to Stacks /1

- **Stacks and Queues:**

- Relatively simple data structures that act as the building blocks for more complex data structures

- **Stack ADT:**

- A list of data items where items are inserted and removed based on the **last-in first-out** (LIFO) principle
- That is, you can only insert and remove an item from one end of the stack, which is called the “top” of the stack”
- The item that you remove from the top of the stack will always be the last item that was inserted into the stack



# Introduction to Stacks /2

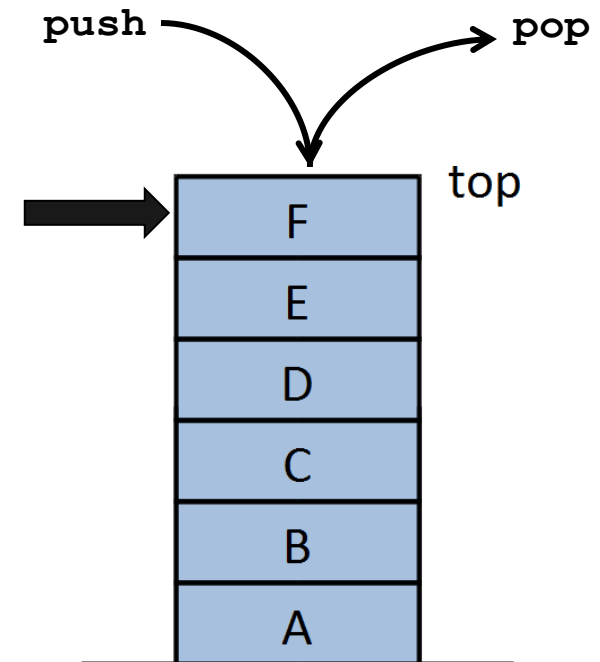
## ■ Examples of a stack:

- A stack of pancakes: each time you make a pancake, you put it onto the whole stack, and you eat the most recently cooked one
- Undo functionality
- Local (static) variable allocation
- Recursion!



## ■ A stack visualized:

- F is the last item added
- F will also be the first item removed



# Introduction to Stacks /3

---

- **Stack ADT Applied:**

- In a web browser, when you visit a new web page, the web address of the previous web page is pushed into a stack containing visited web addresses
- When you press the “back” button, the previous web address is popped from the stack, so you can navigate through the most-recently visited web page

# Introduction to Stacks /4

## ■ Stack ADT Operations:

- **Push:** inserts an item at the top of the stack
- **Pop:** removes an item from the top of the stack, and returns a reference to that item
- **Peek:** returns a reference to the top item without removing the item from the stack

```
class Stack { // public interface for Stack ADT
    ...
public:
    typedef int StackItem; // integer stack item

    bool push(StackItem item);
    StackItem pop();
    StackItem peek();
};
```

# Introduction to Stacks /5

- **Stack ADT can be implemented using two main representation strategies:**

- Linked representation (e.g., using **linked lists**)
- Sequential representation (e.g., using **arrays**)
  - Implemented based on the LIFO principle, and by following linked or sequential list operations discussed previously

- **Dynamic (Sequential) Stack:**

**Lab 2**

- A sequential list based on dynamically-resized arrays

- **Stack ADT Special Cases:**

- Pushing onto a full stack
- Popping an empty stack
- Peeking an empty stack
  - These special cases must be accounted for in the implementation of the stack ADT

# Stack: Linked Representation

---



# Stack Example: Numbers

---

```
Stack stack;  
stack.push(6);  
stack.push(18);  
stack.peek(); // returns 18  
stack.peek(); // returns 18  
stack.pop();  // returns 18  
stack.peek(); // returns 6
```

# Stack Example: Reverse Word

---

# Stack: Runtime

---

- Insert:  $O(1)$
- Delete:  $O(1)$
- Search:  $O(n)$ 
  - Do we use this in stack implementations?

# Introduction to Queues /1

## ■ Queue ADT:

- A list of data items where items are inserted and removed based on the **first-in first-out** (FIFO) principle
- That is, you can insert an item from one end of the queue, which we will refer to as “rear”, and remove an item from the other end, which we will refer to as “front”
- The item that you remove from the front of the queue will always be the first item that you have placed into it



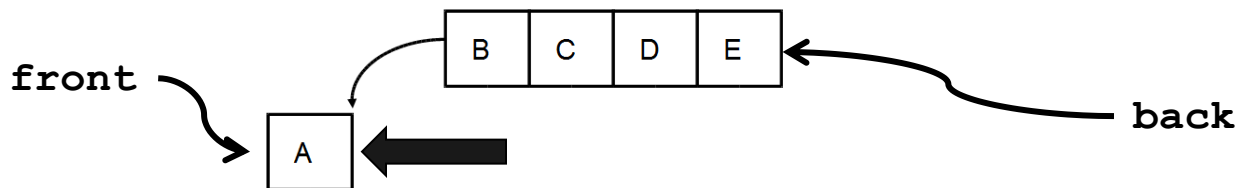
# Introduction to Queues /2

## ■ A queue in the physical world:

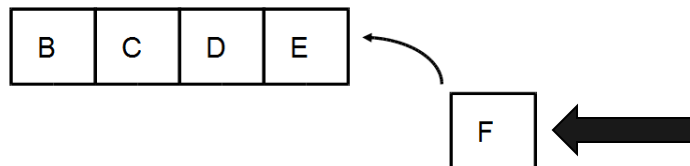
- Consider a line-up at the grocery store
- When a person enters the line-up, they are “enqueued” at the **rear** behind the last person who entered the queue
- The grocery store salesperson then “dequeues” a person from the **front** of the line to process their groceries

## ■ A queue visualized:

- A is first item removed since it is at the **front**



- F is the last item added, so it is placed at the **rear**



# Introduction to Queues /3

---

## ■ Queue ADT Applied:

- When your printer receives multiple print requests, a mechanism is needed to decide which documents to print first, and how to handle the rest of the documents that still need to be printed
- To that end, when a new print request is received, it is enqueued into a queue containing documents to print
- When the printer is ready to print a new document, it dequeues a document from the queue for printing

# Introduction to Queues /4

## ■ Queue ADT Operations:

- **Enqueue:** inserts an item at the rear of the queue
- **Dequeue:** removes an item from the front of the queue, and returns a reference to that item
- **Peek:** returns a reference to the front item without removing the item from the queue

```
class Queue { // public interface for Queue ADT
    ...
public:
    typedef int QueueItem; // integer queue item

    bool enqueue(QueueItem qItem);
    QueueItem dequeue();
    QueueItem peek();
};
```

# Introduction to Queues /5

- **Queue ADT can be implemented using two main representation strategies:**

- Linked representation (e.g., using linked lists)
- Sequential representation (e.g., using arrays)
  - Implemented based on the FIFO principle, and by following linked or sequential list operations discussed previously

**Lab 2**

- **Queue ADT Special Cases:**

- Enqueuing into a full queue
- Dequeuing an empty queue
- Peeking an empty queue
  - These special cases must be accounted for in the implementation of the queue ADT



# Queue: Linked Representation

---

# Queue Example: Numbers

---

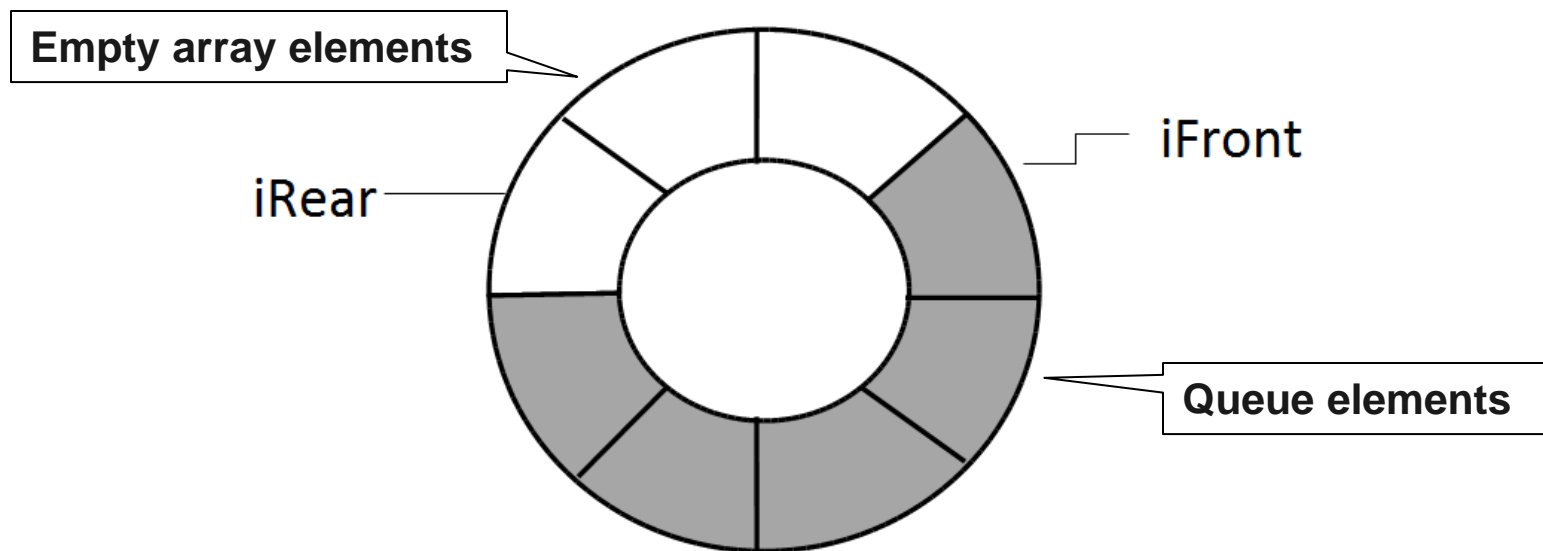
```
Queue queue;  
queue.enqueue(3);  
queue.enqueue(17);  
queue.dequeue(); // returns 3  
queue.enqueue(20);  
queue.dequeue(); // returns 17  
queue.dequeue(); // returns 20
```

# Queue Example: Morse Code

---

# Circular Queue

- A sequential list based on circular arrays, where you can always insert an item at the rear if the list is not full
- In the array, keep updating two indices: an index pointing to the front of the queue, and an index pointing to the location **immediately after** the rear
- The indices of the queue are computed using the modulo function (of the size of the array) to not exceed array size

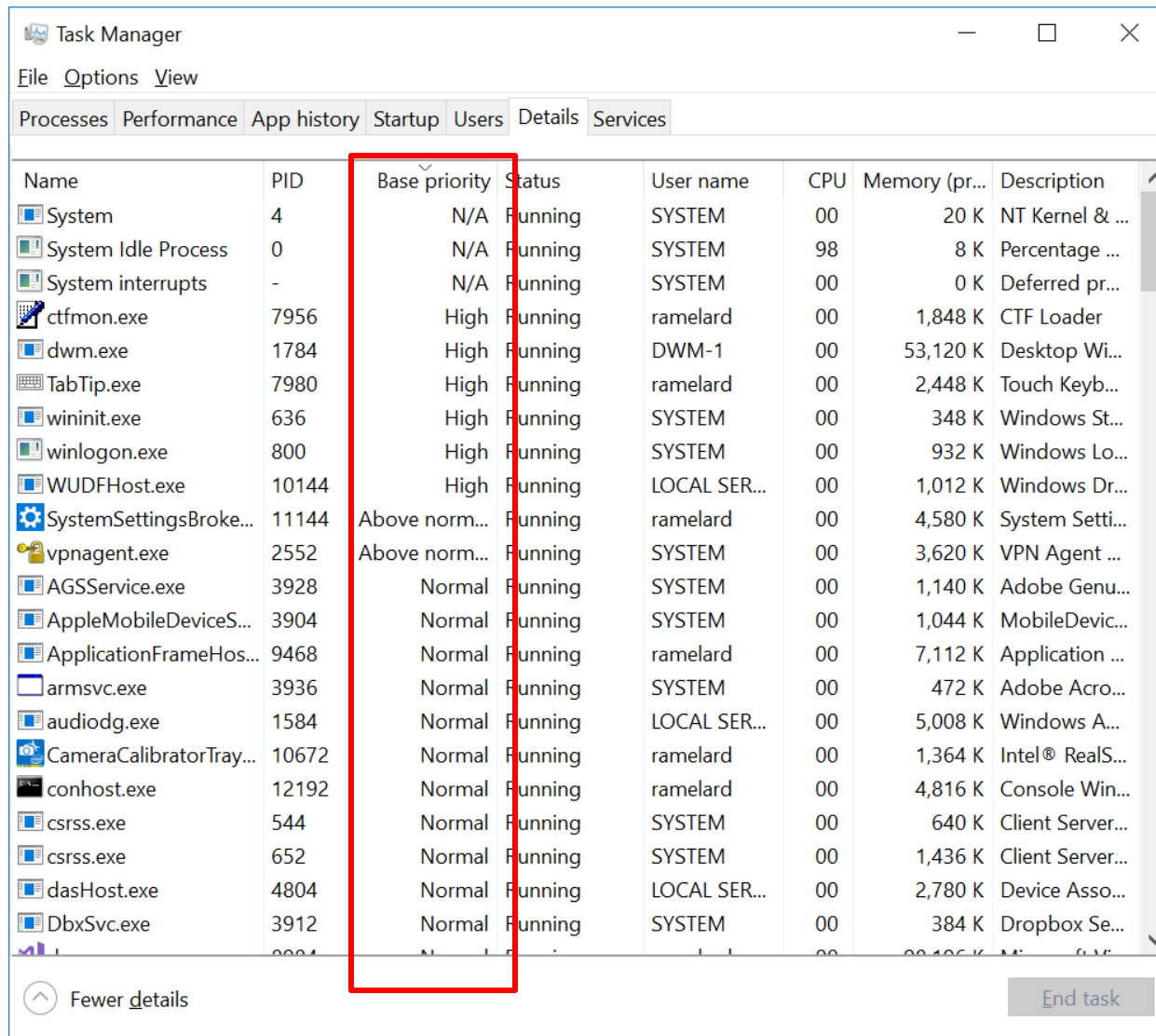


# Priority Queue

- Instead of FIFO, items are organized according to an inherent priority.
- **Example:** operating system process scheduling
- We will look at how to construct priority queues later in the course.

```
struct Item
{
    int value;
    int priority;
};
```

# Priority Queue



The screenshot shows the Windows Task Manager application with the 'Details' tab selected. A red rectangular box highlights the 'Base priority' column in the process list. The table below represents the data visible in the Task Manager window.

Name	PID	Base priority	Status	User name	CPU	Memory (pr...	Description
System	4	N/A	Running	SYSTEM	00	20 K	NT Kernel & ...
System Idle Process	0	N/A	Running	SYSTEM	98	8 K	Percentage ...
System interrupts	-	N/A	Running	SYSTEM	00	0 K	Deferred pr...
ctfmon.exe	7956	High	Running	ramelard	00	1,848 K	CTF Loader
dwm.exe	1784	High	Running	DWM-1	00	53,120 K	Desktop Wi...
TabTip.exe	7980	High	Running	ramelard	00	2,448 K	Touch Keyb...
wininit.exe	636	High	Running	SYSTEM	00	348 K	Windows St...
winlogon.exe	800	High	Running	SYSTEM	00	932 K	Windows Lo...
WUDFHost.exe	10144	High	Running	LOCAL SER...	00	1,012 K	Windows Dr...
SystemSettingsBroke...	11144	Above norm...	Running	ramelard	00	4,580 K	System Setti...
vpnagent.exe	2552	Above norm...	Running	SYSTEM	00	3,620 K	VPN Agent ...
AGSService.exe	3928	Normal	Running	SYSTEM	00	1,140 K	Adobe Genu...
AppleMobileDeviceS...	3904	Normal	Running	SYSTEM	00	1,044 K	MobileDevic...
ApplicationFrameHos...	9468	Normal	Running	ramelard	00	7,112 K	Application ...
armsvc.exe	3936	Normal	Running	SYSTEM	00	472 K	Adobe Acro...
audiodg.exe	1584	Normal	Running	LOCAL SER...	00	5,008 K	Windows A...
CameraCalibratorTray...	10672	Normal	Running	ramelard	00	1,364 K	Intel® RealS...
conhost.exe	12192	Normal	Running	ramelard	00	4,816 K	Console Win...
csrss.exe	544	Normal	Running	SYSTEM	00	640 K	Client Server...
csrss.exe	652	Normal	Running	SYSTEM	00	1,436 K	Client Server...
dasHost.exe	4804	Normal	Running	LOCAL SER...	00	2,780 K	Device Asso...
DbxSvc.exe	3912	Normal	Running	SYSTEM	00	384 K	Dropbox Se...

# Queue: Runtime

---

- Insert:  $O(1)$
- Delete:  $O(1)$
- Search:  $O(n)$ 
  - Do we use this in queue implementations?

## Challenge Question: “Candy Surprise” Dispensing Machine

- Consider a candy dispensing machine where various candies are loaded into the machine from the front in random order. There is only one “slot” for candies.

By default, the user cannot see the candies. However, they can pay 50 cents to see the first candy. They may choose, at any time, whether to purchase a candy from the machine for \$2.

- Design the CandySuprise class that implements this functionality. You may use/declaer the struct Candy as follows:

```
struct Candy
{
    string flavour;
    Candy* next;
};
```



## Challenge Question: “Candy Surprise” Dispensing Machine

---

- Suppose now that the machine is loaded from the back. How does this implementation change? Make the necessary changes to change this functionality, while keeping the class and method declarations the same.

# Lecture Notes Summary

---

- **What do you need to know?**
  - Stack ADT
  - Queue ADT
  - Circular queue
  - Priority Queue

# Food for Thought

---

- **Read:**

- Chapter 6 (Stacks and Queues) from the course handbook

- **Additional Readings:**

- Chapter 7 (Stacks) from “Data Structures and Other Objects Using C++” by Main and Savitch
- Chapter 8 (Queues) from “Data Structures and Other Objects Using C++” by Main and Savitch