

COMP2521 24T2

Abstract Data Types

Sim Mautner

`cs2521@cse.unsw.edu.au`

abstraction
abstract data types
stacks and queues
sets

Slides adapted from those by Kevin Luxa 2521 24T1

Abstraction

ADTs

Stacks

Queues

Sets

Abstraction

is the process of
hiding or generalising
the **details** of an object or system
to **focus on its high-level meaning** or behaviour

Abstraction

ADTs

Stacks

Queues

Sets

Assembly languages abstract away machine code

```
0000000000000000 <fn>:
  0: 55                push rbp
  1: 48 89 e5          mov rbp, rsp
  4: 89 7d ec          mov DWORD PTR [rbp-0x14], edi
  7: c7 45 fc 01 00 00 00 mov DWORD PTR [rbp-0x04], 0x1
  e: c7 45 f8 01 00 00 00 mov DWORD PTR [rbp-0x08], 0x1
15: eb 0e            jmp 25 <fn+0x25>
17: 8b 45 fc          mov eax, DWORD PTR [rbp-0x04]
1a: 0f af 45 f8       imul eax, DWORD PTR [rbp-0x08]
1e: 89 45 fc          mov DWORD PTR [rbp-0x04], eax
21: 83 45 f8 01       add DWORD PTR [rbp-0x08], 0x1
25: 8b 45 f8          mov eax, DWORD PTR [rbp-0x08]
28: 3b 45 ec          cmp eax, DWORD PTR [rbp-0x14]
2b: 7e ea            jle 17 <fn+0x17>
2d: 8b 45 fc          mov eax, DWORD PTR [rbp-0x04]
30: 5d                pop rbp
31: c3                ret
```

Abstraction

ADTs

Stacks

Queues

Sets

Modern programming languages abstract away assembly code

```
push rbp
mov rbp, rsp
mov DWORD PTR [rbp-0x14], edi
mov DWORD PTR [rbp-0x04], 0x1
mov DWORD PTR [rbp-0x08], 0x1
jmp 25 <fn+0x25>
mov eax, DWORD PTR [rbp-0x04]
imul eax, DWORD PTR [rbp-0x08]
mov DWORD PTR [rbp-0x04], eax
add DWORD PTR [rbp-0x08], 0x1
mov eax, DWORD PTR [rbp-0x08]
cmp eax, DWORD PTR [rbp-0x14]
jle 17 <fn+0x17>
mov eax, DWORD PTR [rbp-0x04]
pop rbp
ret
```

```
int fn(int n) {
    int res = 1;
    for (int i = 1; i <= n; i++) {
        res *= i;
    }
    return res;
}
```

Abstraction

ADTs

Stacks

Queues

Sets

A function abstracts away the details or steps of a computation

Abstraction

ADTs

Stacks

Queues

Sets

We drive a car by using a steering wheel and pedals

We operate a television through a remote control

We deposit and withdraw money to/from our bank account via an ATM

Abstraction

ADTs

Stacks

Queues

Sets

To use a system,
it should be enough to
understand **what** its components do
without knowing **how**...

Abstraction

ADTs

Example - List
Interface
Benefits of ADTs
ADTs in C
Other examples

Stacks

Queues

Sets

A data type is...

- a collection or grouping of values
 - could be atomic, e.g., `int`, `double`
 - could be composite/structured, e.g., arrays, structs
- a collection of operations on those values

Examples:

- `int`
 - operations: addition, multiplication, comparison
- array of `ints`
 - operations: index lookup, index assignment

Abstraction

ADTs

- Example - List
- Interface
- Benefits of ADTs
- ADTs in C
- Other examples

Stacks

Queues

Sets

An abstract data type...

is a description of a data type
from the point of view of a **user**,
in terms of the **operations** on the data type and
the **behaviour** of these operations.

Importantly, an ADT does not specify
how the data type or operations should be implemented.

Abstraction

ADTs

Example - List

Interface

Benefits of ADTs

ADTs in C

Other examples

Stacks

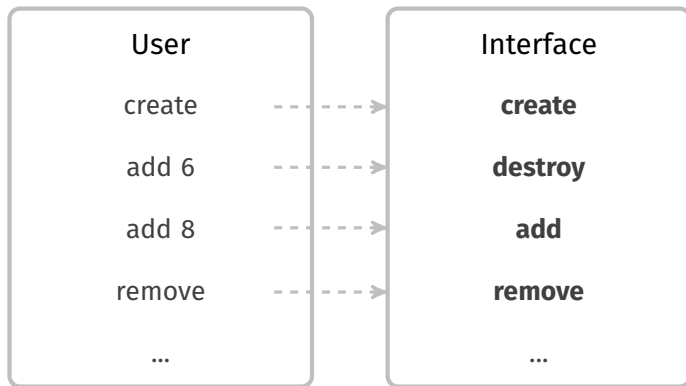
Queues

Sets

- What makes a List, a List?
- What things do we need a List to do?
 - create
 - add to start
 - add to end
 - remove from start
 - remove from end
 - destroy

The set of operations provided by an ADT is called the **interface**.

Users of an ADT only see and interact with the interface.



Abstraction

ADTs

Example - List

Interface

Benefits of ADTs

ADTs in C

Other examples

Stacks

Queues

Sets

- An ADT interface must:
1. clearly describe the behaviour of each operation
 2. describe the conditions under which each operation can be used

Example:

remove from end
removes the item at
the end of the list
assumes that the list is not empty

Abstraction

ADTs

Example - List

Interface

Benefits of ADTs

ADTs in C

Other examples

Stacks

Queues

Sets

- What makes a List, a List?
- What things do we need a List to do?
- What details do we need to know before we can use the List?
 - <https://cgi.cse.unsw.edu.au/~cs2521/24T2/lectures/code/wk3-adts/list-adt/List.h>
- What would it look like to use the List?
 - https://cgi.cse.unsw.edu.au/~cs2521/24T2/lectures/code/wk3-adts/list-adt/test_list.c
- What do we *not* need to know?
 - Anything else

Abstraction

ADTs

Example - List

Interface

Benefits of ADTs

ADTs in C

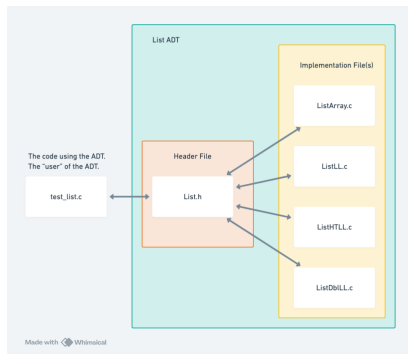
Other examples

Stacks

Queues

Sets

What could be going on behind the scenes?



<https://cgi.cse.unsw.edu.au/~cs2521/24T2/lectures/code/wk3-adts/list-adt/summary.pdf>

Abstraction

ADTs

Example - List

Interface

Benefits of ADTs

ADTs in C

Other examples

Stacks

Queues

Sets

What are the advantages of ADTs?

Abstraction

ADTs

Example - List
Interface

Benefits of ADTs

ADTs in C
Other examples

Stacks

Queues

Sets

What are the advantages of ADTs?

- Design by Contract: Programmers don't need to know *how* the ADT will be implemented, in order to start using it. The `.h` file documents the agreement between both parties.

Abstraction

ADTs

Example - List
Interface

Benefits of ADTs

ADTs in C
Other examples

Stacks

Queues

Sets

What are the advantages of ADTs?

- **Design by Contract:** Programmers don't need to know *how* the ADT will be implemented, in order to start using it. The `.h` file documents the agreement between both parties.
- **Modular:** The implementation of the ADT can be changed without updates being required to the broader application.

Abstraction

ADTs

Example - List
Interface

Benefits of ADTs

ADTs in C
Other examples

Stacks

Queues

Sets

What are the advantages of ADTs?

- **Design by Contract:** Programmers don't need to know *how* the ADT will be implemented, in order to start using it. The `.h` file documents the agreement between both parties.
- **Modular:** The implementation of the ADT can be changed without updates being required to the broader application.
- **Security:** Prevents operations which shouldn't be possible, from being executed on the data.

Abstraction

ADTs

Example - List

Interface

Benefits of ADTs

ADTs in C

Other examples

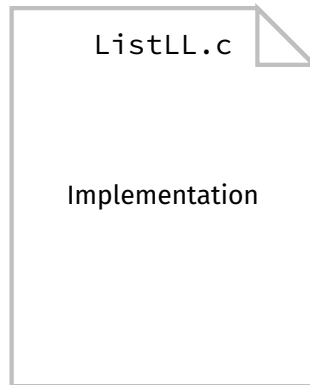
Stacks

Queues

Sets

In C, abstract data types are implemented using two files:

- a `.h` file that contains the **interface**
- a `.c` file that contains the **implementation**



Abstraction

ADTs

Example - List

Interface

Benefits of ADTs

ADTs in C

Other examples

Stacks

Queues

Sets

The interface includes:

- forward declaration of the struct for the concrete representation
 - via `typedef struct t *T`
 - **the struct is not defined in the interface**
- function prototypes for all operations
- clear description of operations
 - via comments
- a contract between the ADT and clients
 - documentation describes how an operation can be used
 - and what the expected result is *as long as the operation is used correctly*

Abstraction

ADTs

- Example - List
- Interface
- Benefits of ADTs

ADTs in C

- Other examples

Stacks

Queues

Sets

List.h

```
typedef struct list List;

/** Creates a new empty list */
List * ListNew(void);

/** Frees memory allocated to the list */
void ListDestroy(List * l);

/** Adds an item to the end of the list */
void ListInsertEnd(List * l, int item);

/** Removes the item end of the list
    Assumes that the list is not empty */
int ListRemoveEnd(List * l);
```

Abstraction

ADTs

Example - List

Interface

Benefits of ADTs

ADTs in C

Other examples

Stacks

Queues

Sets

The implementation includes:

- concrete definition of the data structures
 - definition of `struct t`
- function implementations for all operations

Abstraction

ADTs

- Example - List
- Interface
- Benefits of ADTs

ADTs in C

- Other examples

Stacks

Queues

Sets

Stack.c

```
struct list {  
    ...  
};  
  
List * ListNew(void) {  
    ...  
}  
  
void ListDestroy(List * l) {  
    ...  
}  
  
void ListInsertEnd(List * l, int item) {  
    ...  
}  
  
int ListRemoveEnd(List * l) {  
    ...  
}
```

Abstraction

ADTs

Example - List
Interface
Benefits of ADTs
ADTs in C
Other examples

Stacks

Queues

Sets

A user of an ADT #includes the interface and uses the interface functions to interact with the ADT.

```
user.c

#include "List.h"

int main(void) {
    List * l = ListNew();
    ListInsertStart(l, 6);
    ListInsertStart(l, 8);
    int item = ListRemoveEnd(l);
    ...
}
```


Abstraction

ADTs

Example - List

Interface

Benefits of ADTs

ADTs in C

Other examples

Stacks

Queues

Sets

Naming conventions:

- ADTs are defined in files whose names start with an uppercase letter
 - For example, for a List ADT:
 - The interface is defined in `List.h`
 - The implementation is defined in `List.c`
- ADT interface function names are in PascalCase and begin with the name of the ADT

Abstraction

ADTs

Example - List
Interface
Benefits of ADTs
ADTs in C
Other examples

Stacks

Queues

Sets

- Stack
- Queue
- Set
- Multiset
- Map
- Graph
- Priority Queue

Abstraction

ADTs

Example - List

Interface

Benefits of ADTs

ADTs in C

Other examples

Stacks

Queues

Sets

Example of an ADT: Stack

A stack is a linear collection of items
with two main operations:

push

adds an item to the top of the stack

pop

removes the item at the top of the stack

Abstraction

ADTs

- Example - List
- Interface
- Benefits of ADTs
- ADTs in C
- Other examples

Stacks

Queues

Sets

User

Stack

Operations

push

adds an item to the top of the stack

pop

removes the item at the top of the stack



Abstraction

ADTs

- Example - List
- Interface
- Benefits of ADTs
- ADTs in C
- Other examples

Stacks

Queues

Sets

User

push 8

push 3

push 7

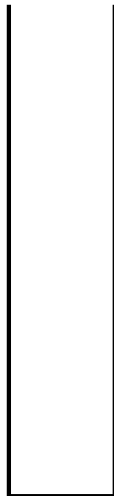
pop

pop

push 1



Stack



Operations

push

adds an item to the top of the stack

pop

removes the item at the top of the stack

User

push 8

push 3

push 7

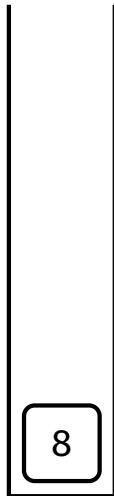
pop

pop

push 1



Stack



Operations

push

adds an item to the top of the stack

pop

removes the item at the top of the stack

Abstraction

ADTs

- Example - List
- Interface
- Benefits of ADTs
- ADTs in C
- Other examples

Stacks

Queues

Sets

User

push 8

push 3

push 7

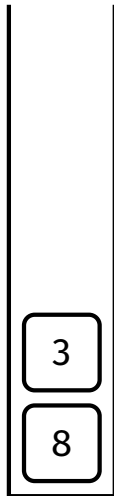
pop

pop

push 1



Stack



Operations

push

adds an item to the top of the stack

pop

removes the item at the top of the stack

Abstraction

ADTs

- Example - List
- Interface
- Benefits of ADTs
- ADTs in C
- Other examples

Stacks

Queues

Sets

User

push 8

push 3

push 7

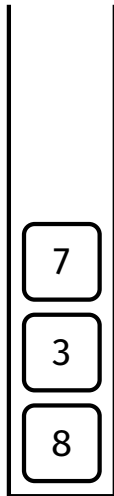
pop

pop

push 1



Stack



Operations

push

adds an item to the top of the stack

pop

removes the item at the top of the stack

Abstraction

ADTs

- Example - List
- Interface
- Benefits of ADTs
- ADTs in C
- Other examples

Stacks

Queues

Sets

User

push 8

push 3

push 7

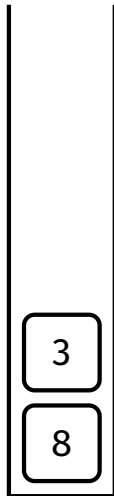
pop ⇒ 7

pop

push 1



Stack



Operations

push

adds an item to the top of the stack

pop

removes the item at the top of the stack

Abstraction

ADTs

- Example - List
- Interface
- Benefits of ADTs
- ADTs in C
- Other examples

Stacks

Queues

Sets

User

push 8

push 3

push 7

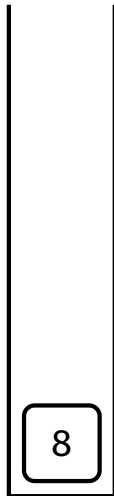
pop \Rightarrow 7

pop \Rightarrow 3

push 1



Stack



Operations

push

adds an item to the top of the stack

pop

removes the item at the top of the stack

Abstraction

ADTs

- Example - List
- Interface
- Benefits of ADTs
- ADTs in C
- Other examples

Stacks

Queues

Sets

User

push 8

push 3

push 7

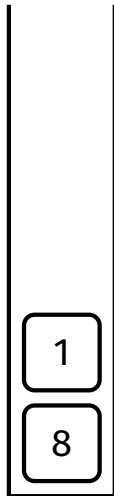
pop \Rightarrow 7

pop \Rightarrow 3

push 1



Stack



Operations

push

adds an item to the top of the stack

pop

removes the item at the top of the stack

Abstraction

ADTs

Example - List

Interface

Benefits of ADTs

ADTs in C

Other examples

Stacks

Queues

Sets

- 1 Decide what operations you want to provide
 - Operations to **create, query, manipulate**
 - What are their inputs and outputs?
 - What are the conditions under which they can be used (if any)?
- 2 Provide the function signatures and documentation for these operations in a `.h` file
- 3 The “developer” builds a concrete implementation for the ADT in a `.c` file
- 4 The “user” `#includes` the interface in their program and uses the provided functions

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

Queues

Sets

A **stack** is a collection of items,
such that the **last** item to enter
is the **first** item to leave:

Last In, First Out (LIFO)

(Think stacks of books, plates, etc.)

Abstraction

ADTs

Stacks

Example Usage
Interface
Implementation

Queues

Sets

A **stack** is a collection of items, such that the **last** item to enter is the **first** item to leave:

Last In, First Out (LIFO)

(Think stacks of books, plates, etc.)

- web browser history
- text editor undo/redo
- balanced bracket checking
- HTML tag matching
- RPN calculators
(...and programming languages!)
- function calls

Abstraction

ADTs

Stacks

Example Usage
Interface
Implementation

Queues

Sets

A stack supports the following operations:

push

add a new item to the top of the stack

pop

remove the topmost item from the stack

size

return the number of items on the stack

peek

get the topmost item on the stack without removing it

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

Queues

Sets

A Stack ADT can be used to check for balanced brackets.

Example of balanced brackets:

([{ }])

Examples of unbalanced brackets!

())) ((

([{ })]

([]) ([

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

Queues

Sets

Sample input: ([{ }])

char	stack	check
		-
((-

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

Queues

Sets

Sample input: ([{ }])

char	stack	check
		-
((-
[([-

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

Queues

Sets

Sample input: ([{ }])

char	stack	check
		-
((-
[([-
{	([{	-

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

Queues

Sets

Sample input: ([{ }])

char	stack	check
		-
((-
[([-
{	([{	-
}	([{ }	{ = }

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

Queues

Sets

Sample input: ([{ }])

char	stack	check
		-
((-
[([-
{	([{	-
}	([{ }	{ = }
]	([{ }	[=]

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

Queues

Sets

Sample input: ([{ }])

char	stack	check
		-
((-
[([-
{	([{	-
}	([{ }	{ = }
]	([}	[=]
)	((=)

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

Queues

Sets

Sample input: ([{ }])

char	stack	check
		-
((-
[([-
{	([{	-
}	([{	{ = }
]	([[=]
)	((=)
EOF		is empty

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

Queues

Sets

Sample input: ([{ }])

char	stack	check
((-
		-

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

Queues

Sets

Sample input: ([{ })]

char	stack	check
		-
((-
[([-

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

Queues

Sets

Sample input: ([{ }])

char	stack	check
		-
((-
[([-
{	([{	-

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

Queues

Sets

Sample input: ([{ })]

char	stack	check
		-
((-
[([-
{	([{	-
}	([{ }	{ = }

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

Queues

Sets

Sample input: ([{ })]

char	stack	check
		-
((-
[([-
{	([{	-
}	([{ }	{ = }
)	([{ }	[≠)

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

Queues

Sets

Sample input: ([{ })]

char	stack	check
		-
((-
[([-
{	([{	-
}	([{ = }
)	([≠) fail!

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

Queues

Sets

```
typedef struct stack *Stack;

/** Creates a new, empty Stack */
Stack StackNew(void);

/** Frees memory allocated for a Stack */
void StackFree(Stack s);

/** Adds an item to the top of a Stack */
void StackPush(Stack s, Item it);

/** Removes an item from the top of a Stack
    Assumes that the Stack is not empty */
Item StackPop(Stack s);

/** Gets the number of items in a Stack */
int StackSize(Stack s);

/** Gets the item at the top of a Stack
    Assumes that the Stack is not empty */
Item StackPeek(Stack s);
```

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

Array

Linked list

Queues

Sets

How to implement a stack?

array

linked list

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

Array

Linked list

Queues

Sets

Dynamically allocate an array with an initial capacity

Fill the array sequentially — $s[0]$, $s[1]$, ...

Maintain a counter of the number of items on the stack

Abstraction

ADTs

Stacks

Example Usage

Interface

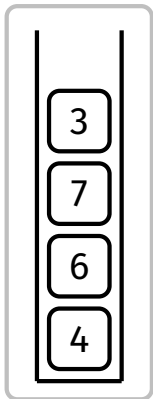
Implementation

Array

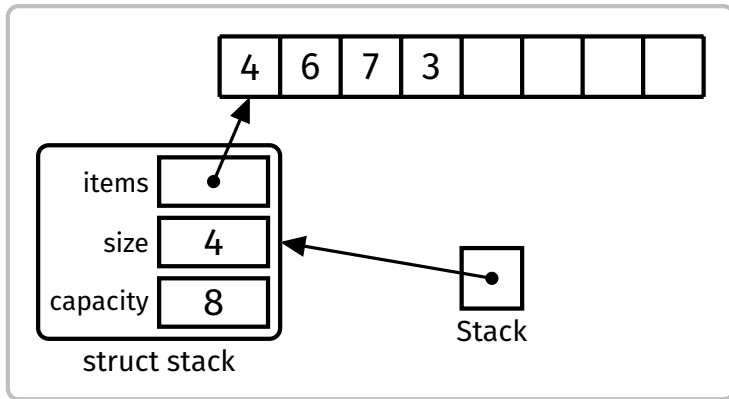
Linked list

Queues

Sets



User's view



Concrete representation

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

Array

Linked list

Queues

Sets

Example

Perform the following operations:

PUSH(9), PUSH(2), PUSH(6), POP, POP, PUSH(8)

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

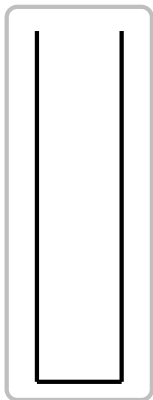
Array

Linked list

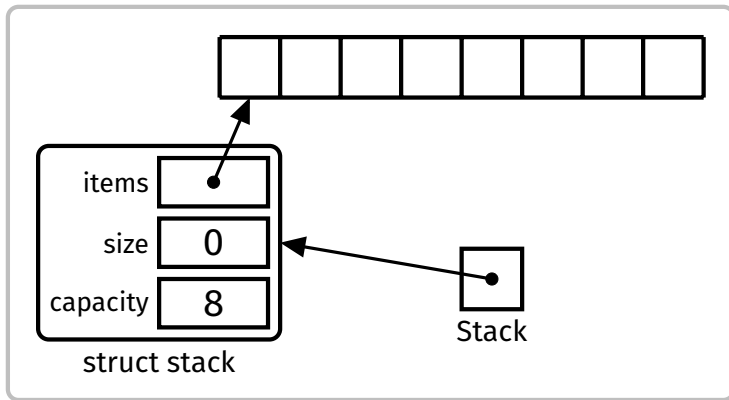
Queues

Sets

PUSH(9) PUSH(2) PUSH(6) POP POP PUSH(8)



User's view



Concrete representation

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

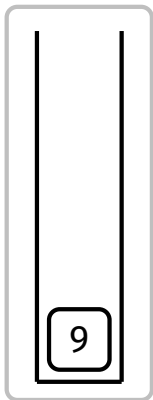
Array

Linked list

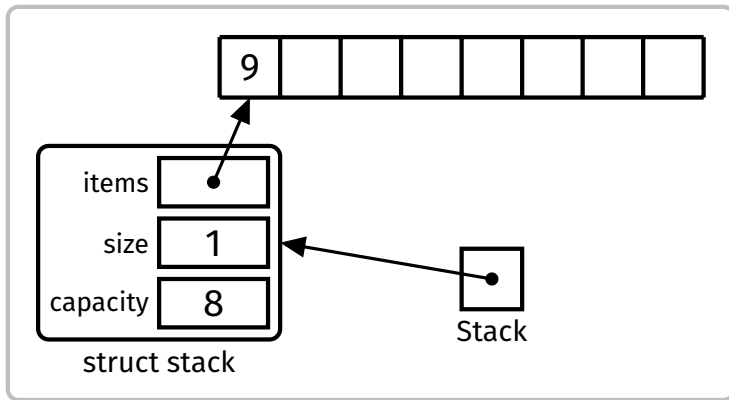
Queues

Sets

PUSH(9) PUSH(2) PUSH(6) POP POP PUSH(8)



User's view



Concrete representation

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

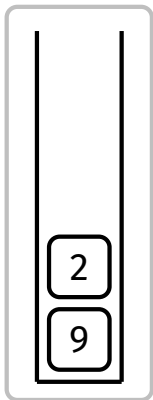
Array

Linked list

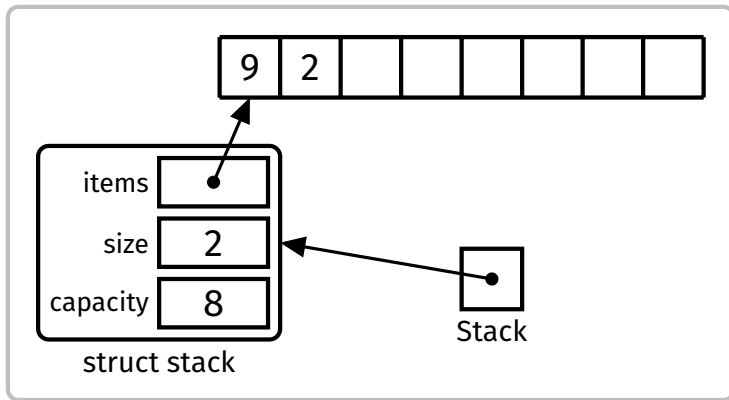
Queues

Sets

PUSH(9) **PUSH(2)** PUSH(6) POP POP PUSH(8)



User's view



Concrete representation

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

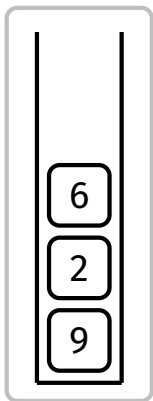
Array

Linked list

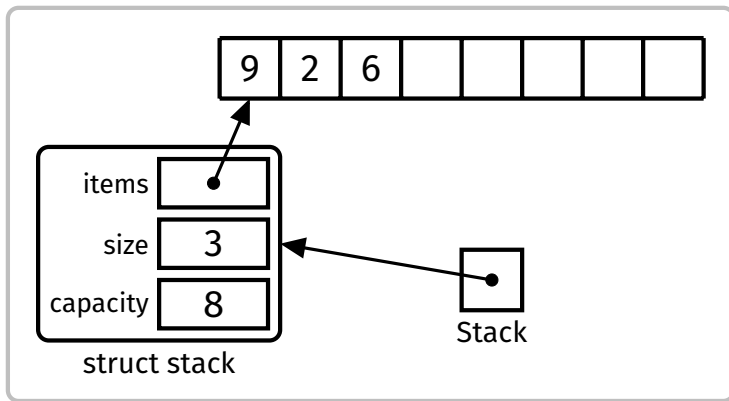
Queues

Sets

PUSH(9) PUSH(2) **PUSH(6)** POP POP PUSH(8)



User's view



Concrete representation

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

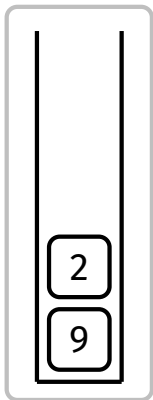
Array

Linked list

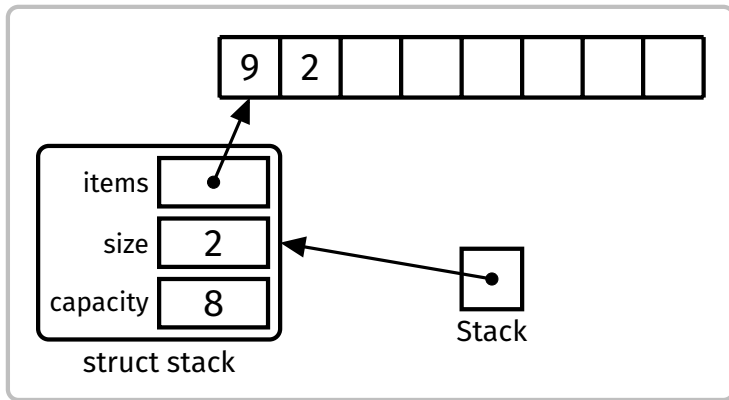
Queues

Sets

PUSH(9) PUSH(2) PUSH(6) POP ⇒ 6 POP PUSH(8)



User's view



Concrete representation

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

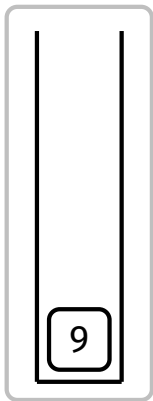
Array

Linked list

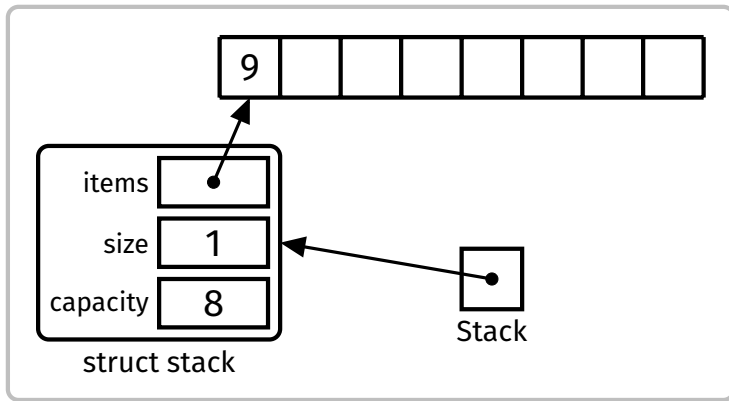
Queues

Sets

PUSH(9) PUSH(2) PUSH(6) POP \Rightarrow 6 **POP \Rightarrow 2** PUSH(8)



User's view



Concrete representation

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

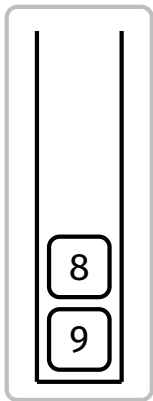
Array

Linked list

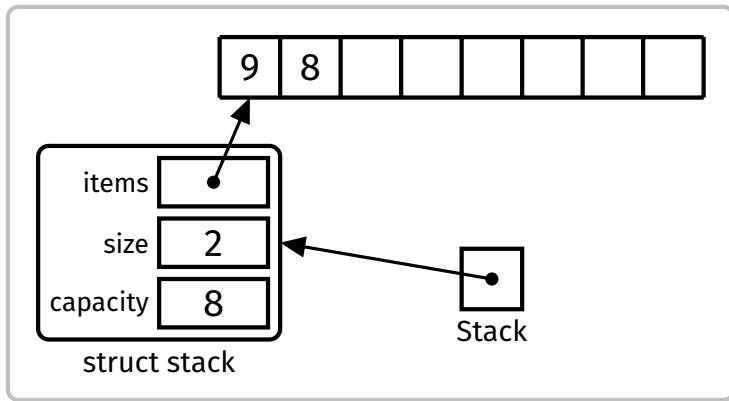
Queues

Sets

PUSH(9) PUSH(2) PUSH(6) POP \Rightarrow 6 POP \Rightarrow 2 **PUSH(8)**



User's view



Concrete representation

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

Array

Linked list

Queues

Sets

Cost of push:

- Inserting item at index `size` is $O(1)$
- What if array is full?
 - If we double the size of the array with `realloc(3)` each time it is full, push will still be $O(1)$ on average

Cost of pop:

- Accessing item at index `(size - 1)` is $O(1)$

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

Array

Linked list

Queues

Sets

Store items in a linked list

To push an item, insert it at the beginning of the list

To pop an item, remove it from the beginning of the list

Abstraction

ADTs

Stacks

Example Usage

Interface

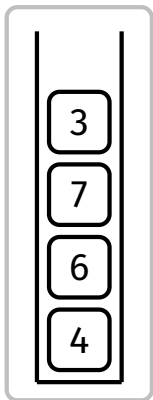
Implementation

Array

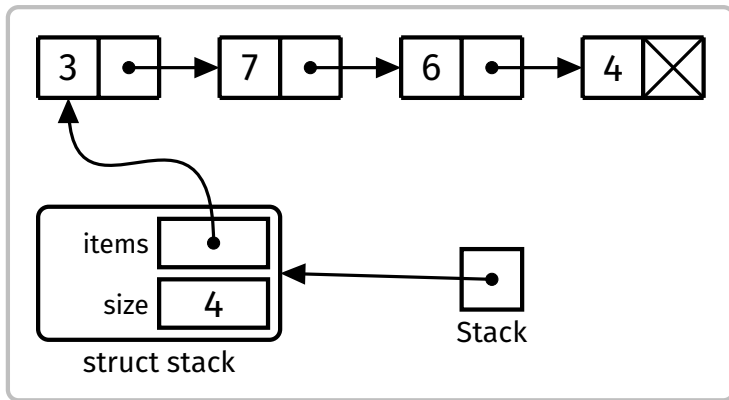
Linked list

Queues

Sets



User's view



Concrete representation

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

Array

Linked list

Queues

Sets

Example

Perform the following operations:

PUSH(9), PUSH(2), PUSH(6), POP, POP, PUSH(8)

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

Array

Linked list

Queues

Sets

PUSH(9)

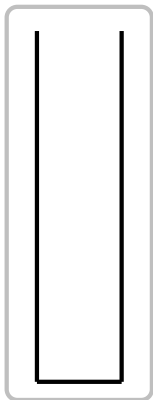
PUSH(2)

PUSH(6)

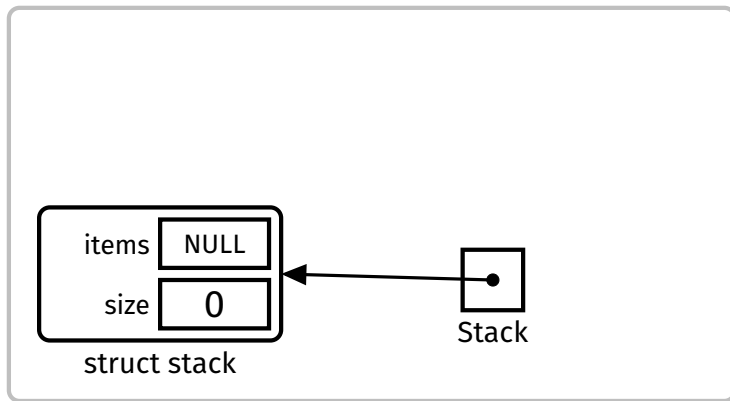
POP

POP

PUSH(8)



User's view



Concrete representation

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

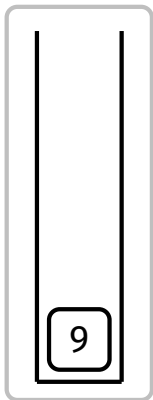
Array

Linked list

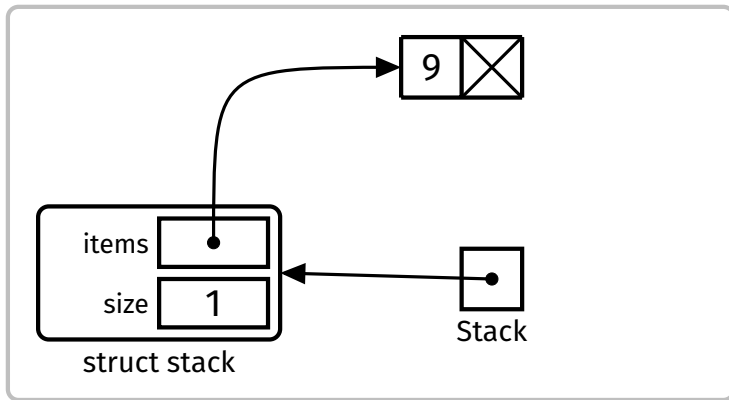
Queues

Sets

PUSH(9) PUSH(2) PUSH(6) POP POP PUSH(8)



User's view



Concrete representation

Abstraction

ADTs

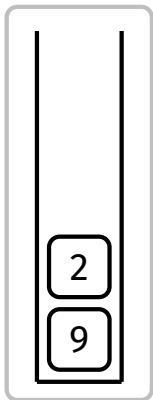
Stacks

Example Usage
Interface
Implementation
Array
Linked list

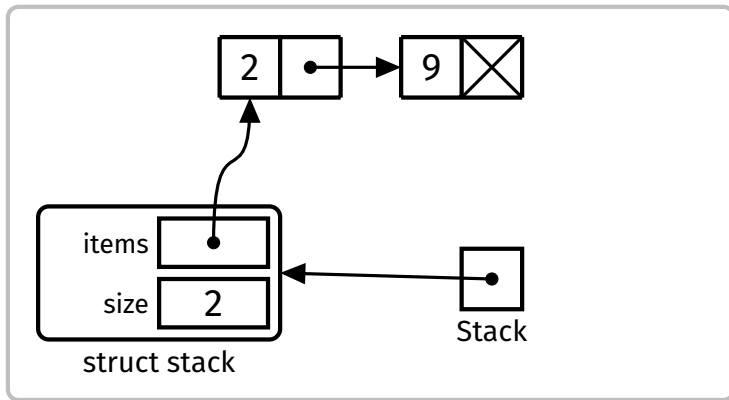
Queues

Sets

PUSH(9) **PUSH(2)** PUSH(6) POP POP PUSH(8)



User's view



Concrete representation

Abstraction

ADTs

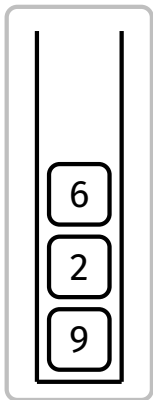
Stacks

Example Usage
Interface
Implementation
Array
Linked list

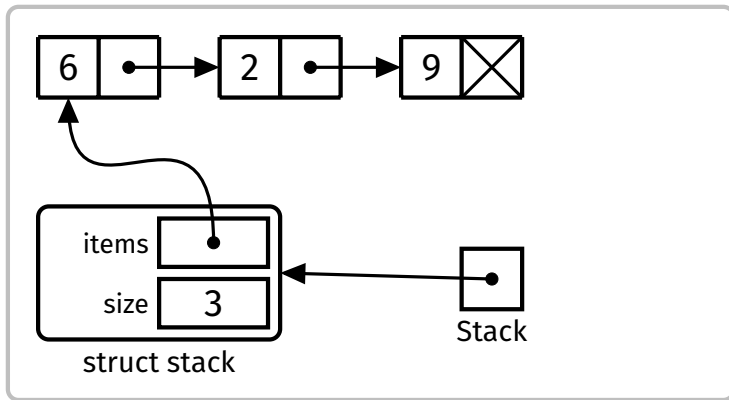
Queues

Sets

PUSH(9) PUSH(2) **PUSH(6)** POP POP PUSH(8)



User's view



Concrete representation

Abstraction

ADTs

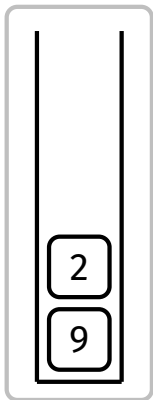
Stacks

- Example Usage
- Interface
- Implementation
- Array
- Linked list

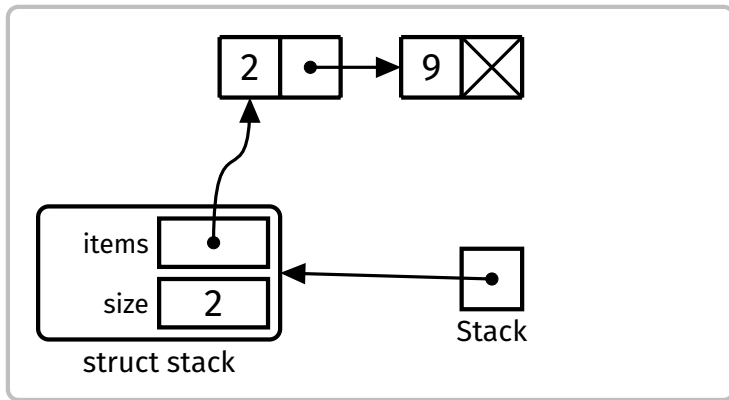
Queues

Sets

PUSH(9) PUSH(2) PUSH(6) **POP ⇒ 6** POP PUSH(8)



User's view



Concrete representation

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

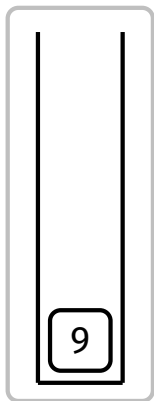
Array

Linked list

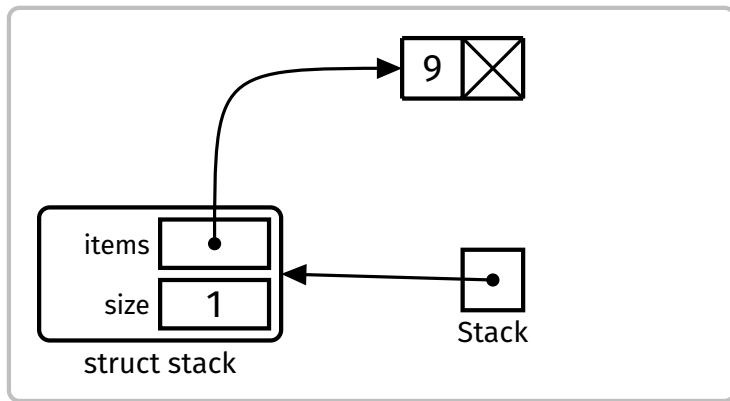
Queues

Sets

PUSH(9) PUSH(2) PUSH(6) POP \Rightarrow 6 **POP \Rightarrow 2** PUSH(8)



User's view



Concrete representation

Abstraction

ADTs

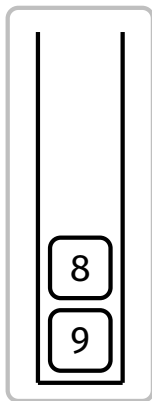
Stacks

Example Usage
Interface
Implementation
Array
Linked list

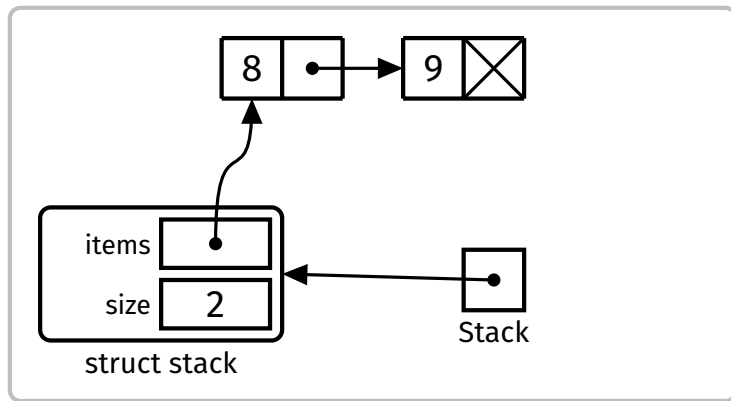
Queues

Sets

PUSH(9) PUSH(2) PUSH(6) POP ⇒ 6 POP ⇒ 2 **PUSH(8)**



User's view



Concrete representation

Abstraction

ADTs

Stacks

Example Usage

Interface

Implementation

Array

Linked list

Queues

Sets

Cost of push:

- Inserting at the beginning of a linked list is $O(1)$

Cost of pop:

- Removing from the beginning of a linked list is $O(1)$

Abstraction

ADTs

Stacks

Queues

Interface

Implementation

Sets

A **queue** is a collection of items, such that the **first** item to enter is the **first** item to leave:

First In, First Out (FIFO)

(Think queues of people, etc.)

Abstraction

ADTs

Stacks

Queues

Interface

Implementation

Sets

A **queue** is a collection of items, such that the **first** item to enter is the **first** item to leave:

First In, First Out (FIFO)

(Think queues of people, etc.)

- waiting lists
- call centres
- access to shared resources (e.g., printers)
- processes in a computer

Abstraction

ADTs

Stacks

Queues

Interface

Implementation

Sets

A queue supports the following operations:

enqueue

add a new item to the end of the queue

dequeue

remove the item at the front of the queue

size

return the number of items in the queue

peek

get the frontmost item of the queue, without removing it

Abstraction

ADTs

Stacks

Queues

Interface

Implementation

Sets

```
typedef struct queue *Queue;

/** Create a new, empty Queue */
Queue QueueNew(void);

/** Free memory allocated to a Queue */
void QueueFree(Queue q);

/** Add an item to the end of a Queue */
void QueueEnqueue(Queue q, Item it);

/** Remove an item from the front of a Queue
    Assumes that the Queue is not empty */
Item QueueDequeue(Queue q);

/** Get the number of items in a Queue */
int QueueSize(Queue q);

/** Get the item at the front of a Queue
    Assumes that the Queue is not empty */
Item QueuePeek(Queue q);
```

Abstraction

ADTs

Stacks

Queues

Interface

Implementation

Linked list

Array

Sets

How to implement a queue?

array

linked list (easier)

Abstraction

ADTs

Stacks

Queues

Interface

Implementation

Linked list

Array

Sets

To enqueue an item, insert it at the end of the list

To dequeue an item, remove it from the beginning of the list

Abstraction

ADTs

Stacks

Queues

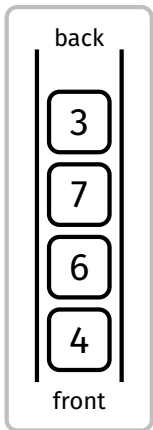
Interface
Implementation

Linked list

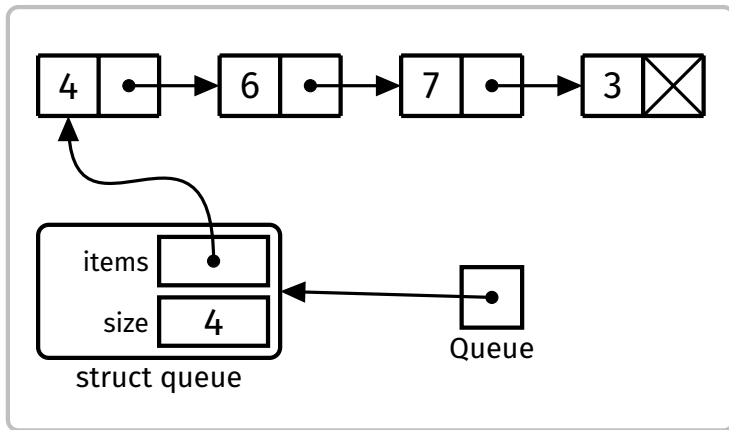
Array

Sets

What's the problem with this design?



User's view



Concrete representation

Abstraction

ADTs

Stacks

Queues

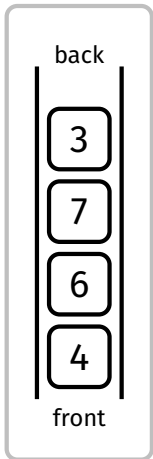
Interface
Implementation

Linked list

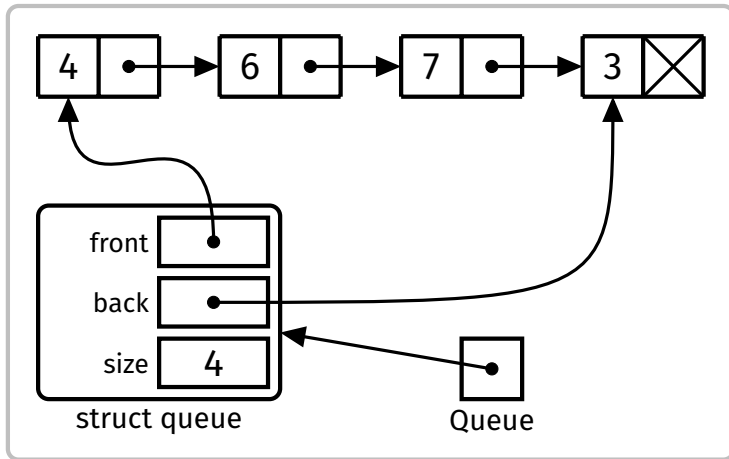
Array

Sets

Improved design



User's view



Concrete representation

Abstraction

ADTs

Stacks

Queues

Interface

Implementation

Linked list

Array

Sets

Example

Perform the following operations:

ENQ(9), ENQ(2), ENQ(6), DEQ, DEQ, ENQ(8)

Abstraction

ADTs

Stacks

Queues

Interface
Implementation
Linked list

Array

Sets

ENQ(9)

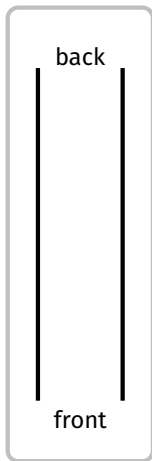
ENQ(2)

ENQ(6)

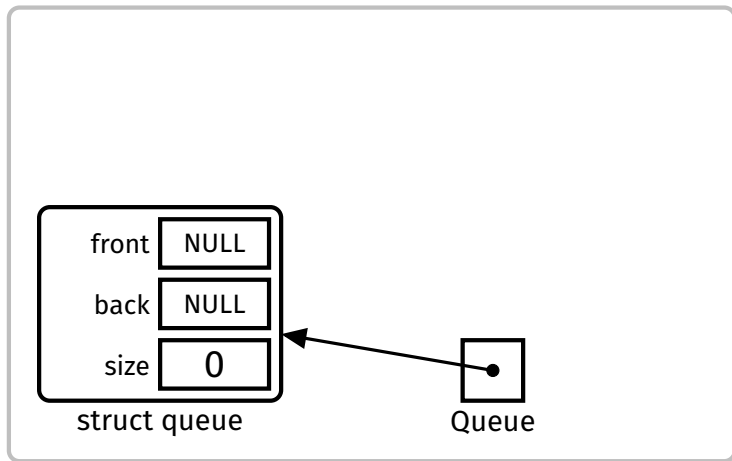
DEQ

DEQ

ENQ(8)



User's view



Concrete representation

Abstraction

ADTs

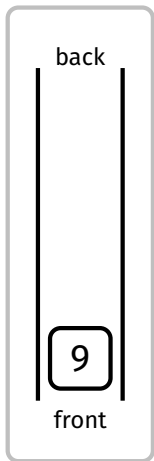
Stacks

Queues

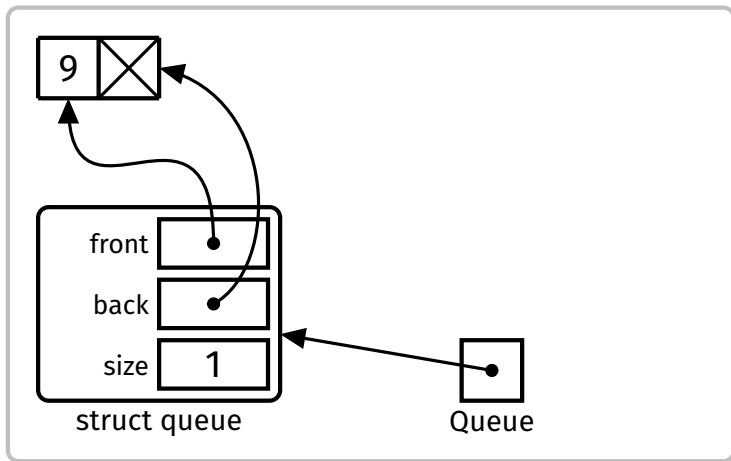
Interface
Implementation
Linked list
Array

Sets

ENQ(9) ENQ(2) ENQ(6) DEQ DEQ ENQ(8)



User's view



Concrete representation

Abstraction

ADTs

Stacks

Queues

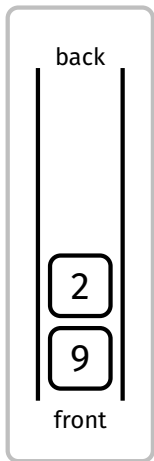
Interface
Implementation

Linked list

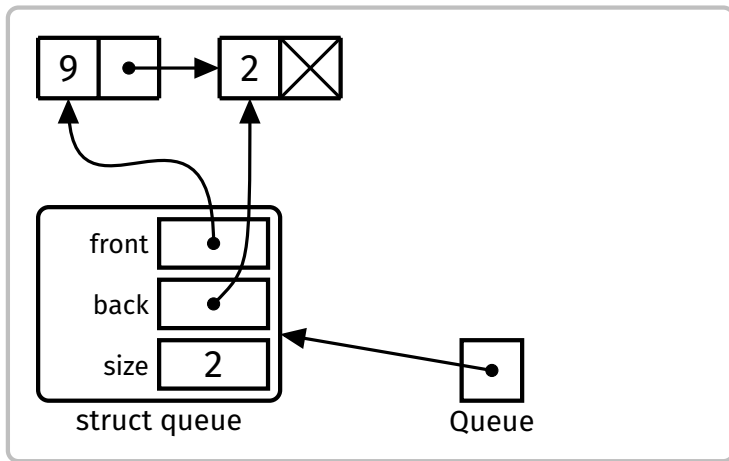
Array

Sets

ENQ(9) ENQ(2) ENQ(6) DEQ DEQ ENQ(8)



User's view



Concrete representation

Abstraction

ADTs

Stacks

Queues

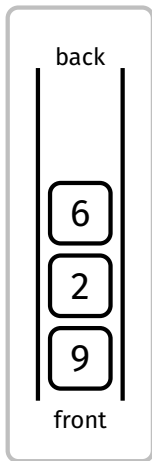
Interface
Implementation

Linked list

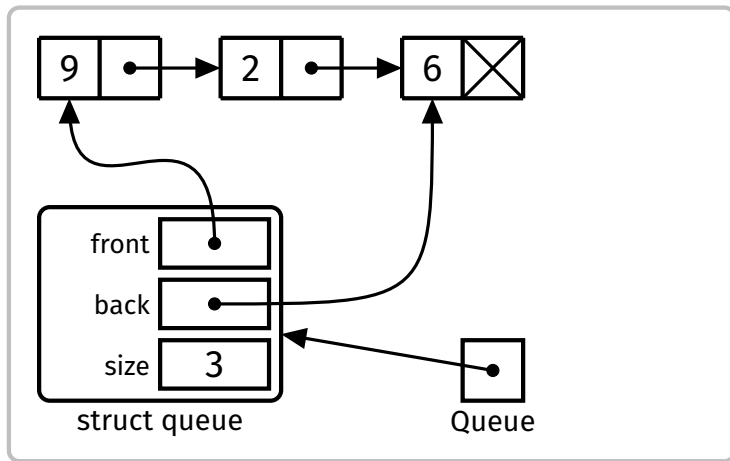
Array

Sets

ENQ(9) ENQ(2) ENQ(6) DEQ DEQ ENQ(8)



User's view



Concrete representation

Abstraction

ADTs

Stacks

Queues

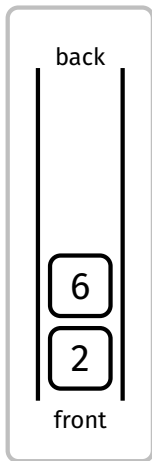
Interface
Implementation

Linked list

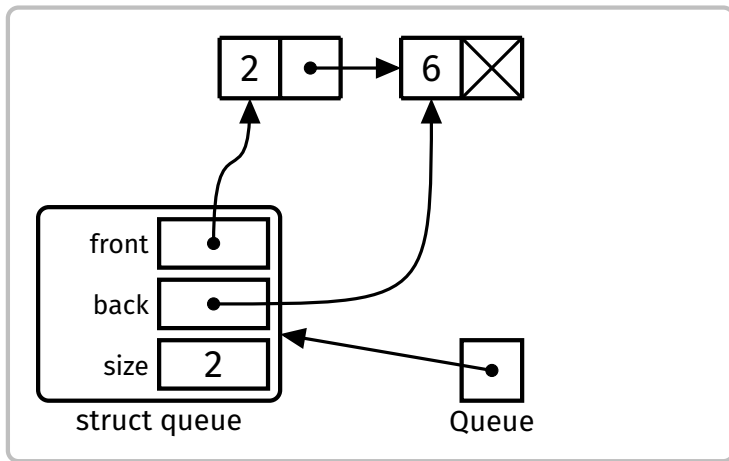
Array

Sets

ENQ(9) ENQ(2) ENQ(6) **DEQ ⇒ 9** DEQ ENQ(8)



User's view



Concrete representation

Abstraction

ADTs

Stacks

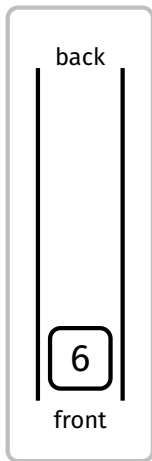
Queues

Interface
Implementation
Linked list

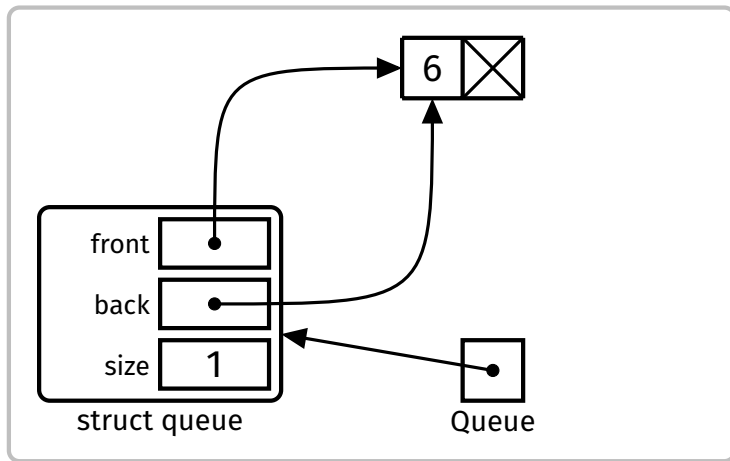
Array

Sets

ENQ(9) ENQ(2) ENQ(6) DEQ \Rightarrow 9 **DEQ \Rightarrow 2** ENQ(8)



User's view



Concrete representation

Abstraction

ADTs

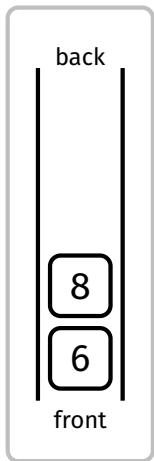
Stacks

Queues

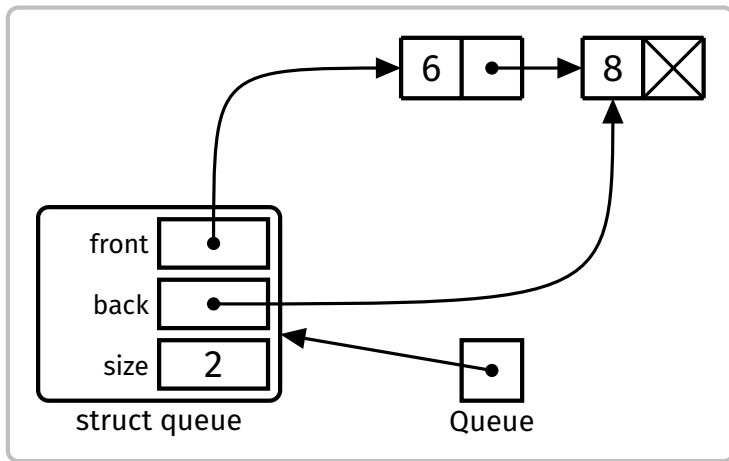
Interface
Implementation
Linked list
Array

Sets

ENQ(9) ENQ(2) ENQ(6) DEQ ⇒ 9 DEQ ⇒ 2 **ENQ(8)**



User's view



Concrete representation

Abstraction

ADTs

Stacks

Queues

Interface

Implementation

Linked list

Array

Sets

Cost of enqueue:

- Inserting at the end of the linked list is $O(1)$

Cost of dequeue:

- Removing from the beginning of the linked list is $O(1)$

Abstraction

ADTs

Stacks

Queues

Interface

Implementation

Linked list

Array

Sets

Dynamically allocate an array with an initial capacity

Maintain an index to the front of the queue

Maintain a counter of the number of items in the queue

Abstraction

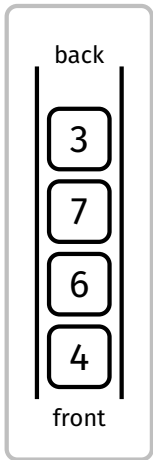
ADTs

Stacks

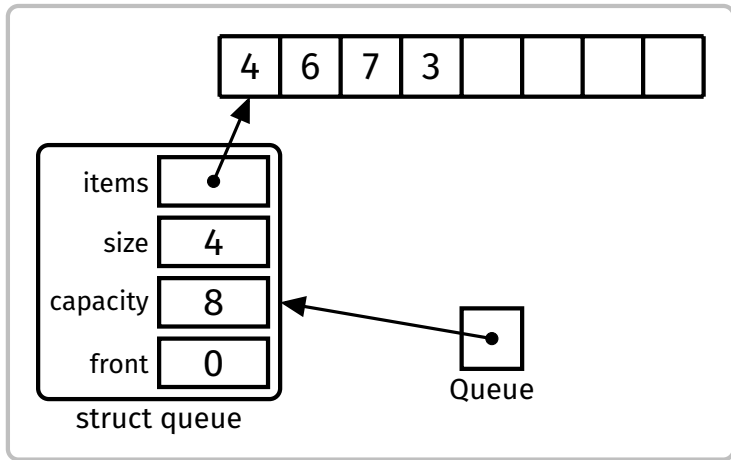
Queues

Interface
Implementation
Linked list
Array

Sets



User's view



Concrete representation

Abstraction

ADTs

Stacks

Queues

Interface

Implementation

Linked list

Array

Sets

Example

Perform the following operations:

ENQ(9), ENQ(2), ENQ(6), DEQ, DEQ, ENQ(8)

Abstraction

ADTs

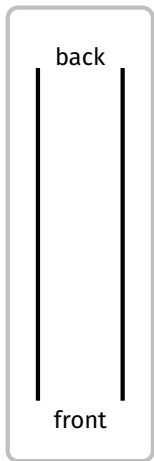
Stacks

Queues

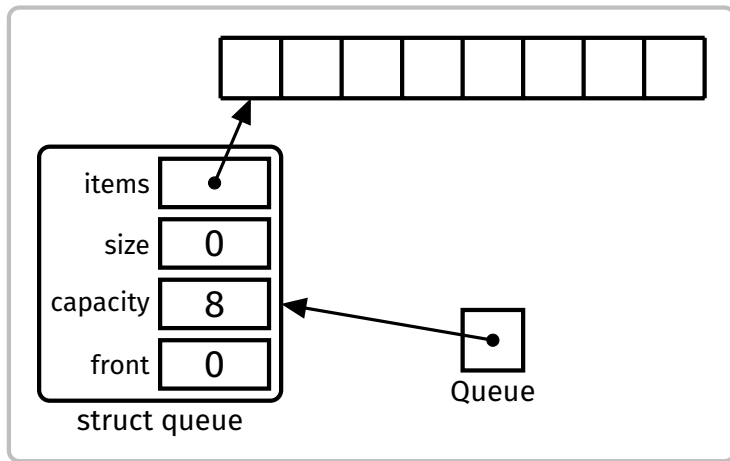
Interface
Implementation
Linked list
Array

Sets

ENQ(9) ENQ(2) ENQ(6) DEQ DEQ ENQ(8)



User's view



Concrete representation

Abstraction

ADTs

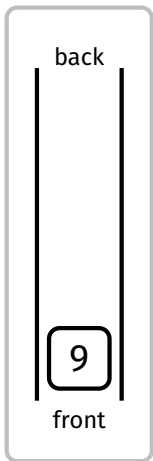
Stacks

Queues

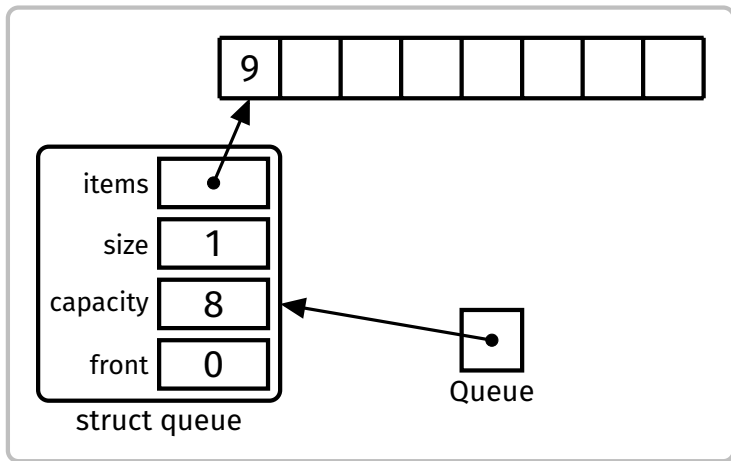
Interface
Implementation
Linked list
Array

Sets

ENQ(9) ENQ(2) ENQ(6) DEQ DEQ ENQ(8)



User's view



Concrete representation

Abstraction

ADTs

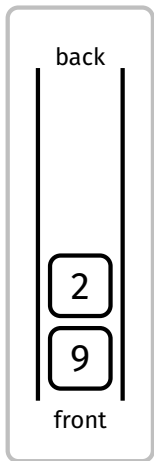
Stacks

Queues

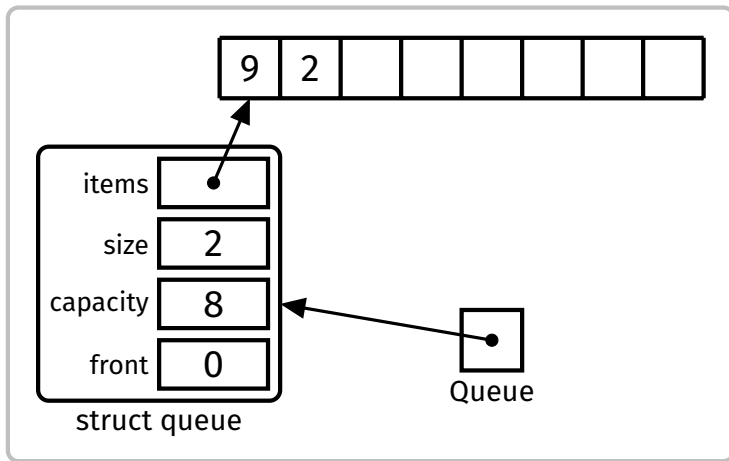
Interface
Implementation
Linked list
Array

Sets

ENQ(9) ENQ(2) ENQ(6) DEQ DEQ ENQ(8)



User's view



Concrete representation

Abstraction

ADTs

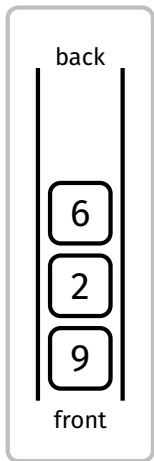
Stacks

Queues

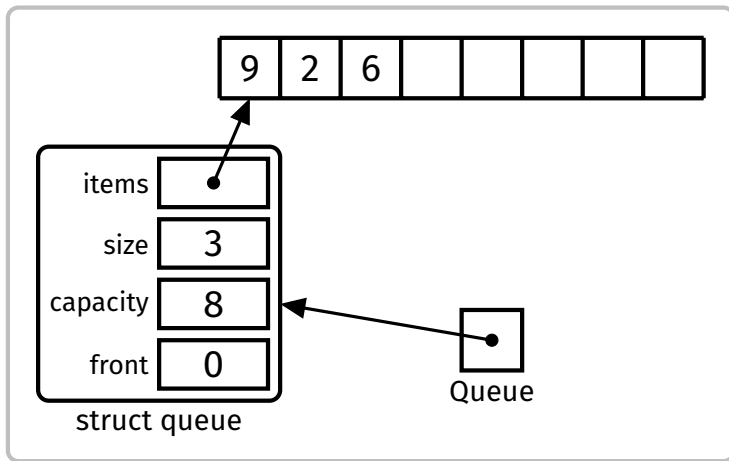
Interface
Implementation
Linked list
Array

Sets

ENQ(9) ENQ(2) ENQ(6) DEQ DEQ ENQ(8)



User's view



Concrete representation

Abstraction

ADTs

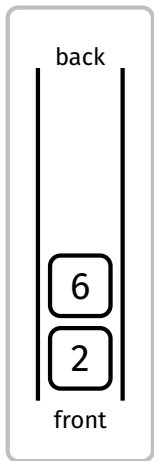
Stacks

Queues

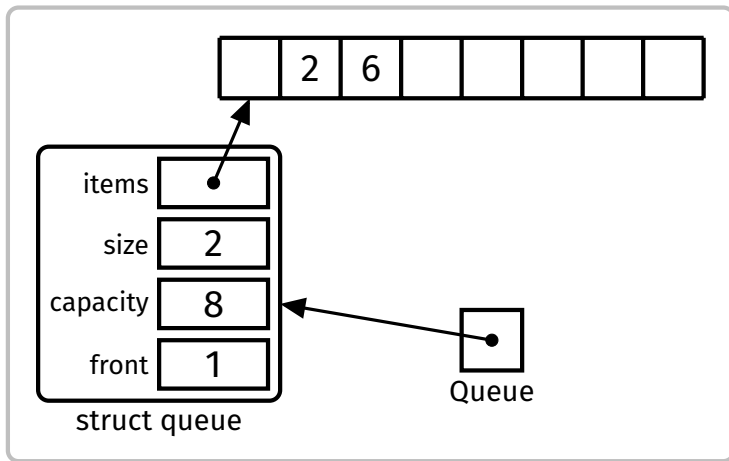
Interface
Implementation
Linked list
Array

Sets

ENQ(9) ENQ(2) ENQ(6) DEQ ⇒ 9 DEQ ENQ(8)



User's view



Concrete representation

Abstraction

ADTs

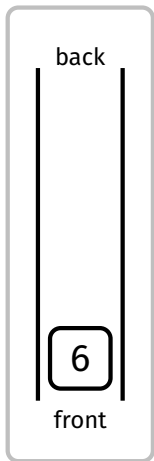
Stacks

Queues

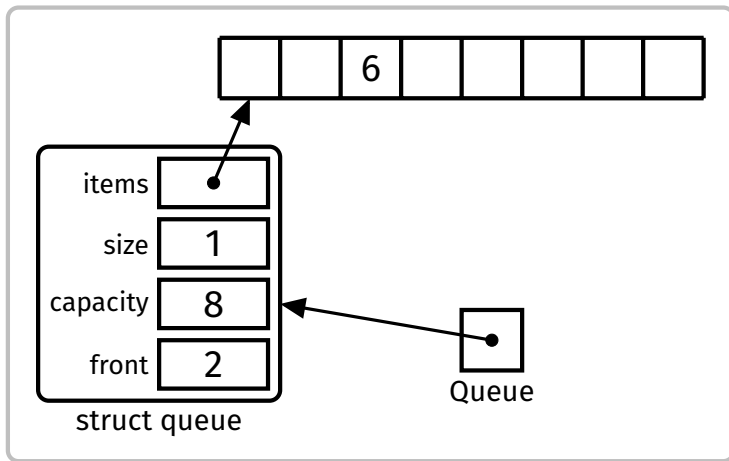
Interface
Implementation
Linked list
Array

Sets

ENQ(9) ENQ(2) ENQ(6) DEQ \Rightarrow 9 **DEQ \Rightarrow 2** ENQ(8)



User's view



Concrete representation

Abstraction

ADTs

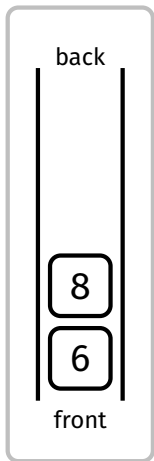
Stacks

Queues

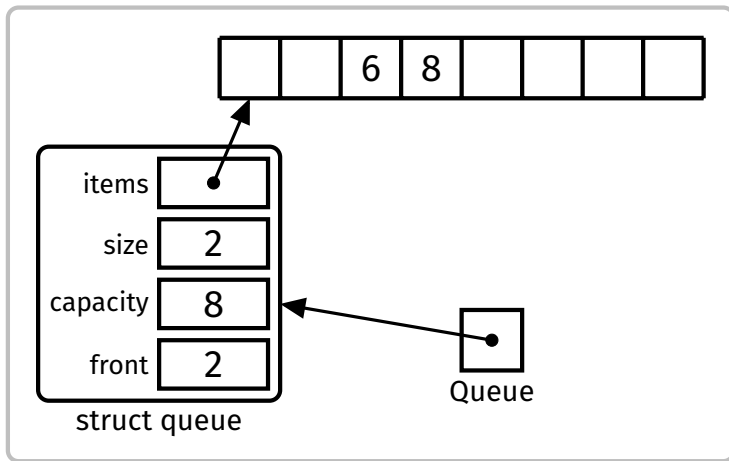
Interface
Implementation
Linked list
Array

Sets

ENQ(9) ENQ(2) ENQ(6) DEQ ⇒ 9 DEQ ⇒ 2 ENQ(8)



User's view



Concrete representation

Abstraction

ADTs

Stacks

Queues

Interface

Implementation

Linked list

Array

Sets

Cost of enqueue:

- Dequeue involves calculating insertion index and inserting item at that index $\Rightarrow O(1)$

Cost of dequeue:

- Dequeue involves accessing item at index front $\Rightarrow O(1)$

Abstraction

ADTs

Stacks

Queues

Sets

Interface

Example Usage

Implementation

Summary

A set is an unordered collection of distinct elements.
In this lecture we are concerned with sets of integers.

Abstraction

ADTs

Stacks

Queues

Sets

Interface

Example Usage

Implementation

Summary

Basic set operations:

- Create an empty set
- Insert an item into the set
- Delete an item from the set
- Check if an item is in the set
- Get the size of the set
- Display the set

Abstraction

ADTs

Stacks

Queues

Sets

Interface

Example Usage

Implementation

Summary

```
#include <stdbool.h>

typedef struct set *Set;

/** Creates a new empty set */
Set SetNew(void);

/** Free memory used by set */
void SetFree(Set set);

/** Inserts an item into the set */
void SetInsert(Set set, int item);

/** Deletes an item from the set */
void SetDelete(Set set, int item);

/** Checks if an item is in the set */
bool SetContains(Set set, int item);

/** Returns the size of the set */
int SetSize(Set set);

/** Displays the set */
void SetShow(Set set);
```

Abstraction

ADTs

Stacks

Queues

Sets

Interface

Example Usage

Implementation

Summary

Counting and displaying distinct numbers:

```
#include <stdio.h>
```

```
#include "Set.h"
```

```
int main(void) {  
    Set s = SetNew();  
  
    int val;  
    while (scanf("%d", &val) == 1) {  
        SetInsert(s, val);  
    }  
  
    printf("Number of distinct values: %d\n", SetSize(s));  
    printf("Values: ");  
    SetShow(s);  
  
    SetFree(s);  
}
```

Abstraction

ADTs

Stacks

Queues

Sets

Interface

Example Usage

Implementation

Unordered array

Ordered array

Linked list

Summary

Different ways to implement a set:

- Unordered array
- Ordered array
- Ordered linked list

Abstraction

ADTs

Stacks

Queues

Sets

Interface

Example Usage

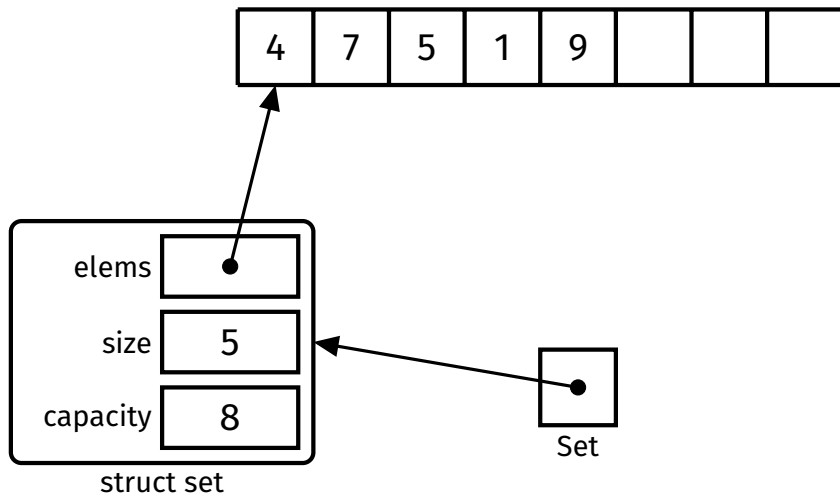
Implementation

Unordered array

Ordered array

Linked list

Summary



Abstraction

ADTs

Stacks

Queues

Sets

Interface

Example Usage

Implementation

Unordered array

Ordered array

Linked list

Summary

How do we check if an element exists?

- Perform linear scan of array $\Rightarrow O(n)$

```
bool SetContains(Set s, int elem) {  
    for (int i = 0; i < s->size; i++) {  
        if (s->elems[i] == elem) {  
            return true;  
        }  
    }  
  
    return false;  
}
```


How do we insert an element?

- If the element doesn't exist, insert it after the last element

```
void SetInsert(Set s, int elem) {  
    if (SetContains(s, elem)) {  
        return;  
    }  
  
    if (s->size == s->capacity) {  
        // error message  
    }  
  
    s->elems[s->size] = elem;  
    s->size++;  
}
```

Time complexity: $O(n)$

- SetContains is $O(n)$ and inserting after the last element is $O(1)$

Abstraction

ADTs

Stacks

Queues

Sets

Interface

Example Usage

Implementation

Unordered array

Ordered array

Linked list

Summary

How do we delete an element?

- If the element exists, overwrite it with the last element

```
void SetDelete(Set s, int elem) {
    for (int i = 0; i < s->size; i++) {
        if (s->elems[i] == elem) {
            s->elems[i] = s->elems[s->size - 1];
            s->size--;
            return;
        }
    }
}
```

Time complexity: $O(n)$

- Finding the element is $O(n)$, overwriting it with the last element is $O(1)$

Abstraction

ADTs

Stacks

Queues

Sets

Interface

Example Usage

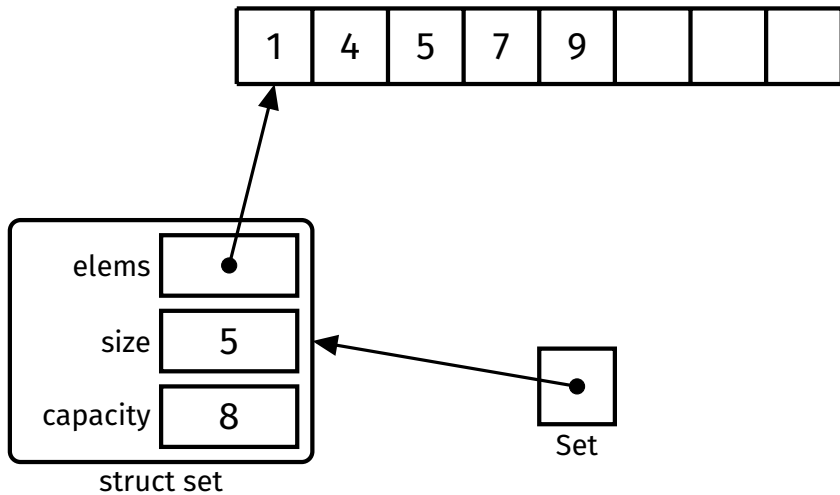
Implementation

Unordered array

Ordered array

Linked list

Summary



Abstraction

ADTs

Stacks

Queues

Sets

Interface

Example Usage

Implementation

Unordered array

Ordered array

Linked list

Summary

How do we check if an element exists?

- Perform binary search $\Rightarrow O(\log n)$

```
bool SetContains(Set s, int elem) {
    int lo = 0;
    int hi = s->size - 1;

    while (lo <= hi) {
        int mid = (lo + hi) / 2;
        if (elem < s->elems[mid]) {
            hi = mid - 1;
        } else if (elem > s->elems[mid]) {
            lo = mid + 1;
        } else {
            return true;
        }
    }

    return false;
}
```

Abstraction

ADTs

Stacks

Queues

Sets

Interface

Example Usage

Implementation

Unordered array

Ordered array

Linked list

Summary

How do we insert an element?

- Use binary search to find the index of the smallest element which is *greater than or equal to* the given element
- If this element *is* the given element, then it already exists, so no need to do anything
- Otherwise, insert the element at that index and shift everything greater than it up

Abstraction

ADTs

Stacks

Queues

Sets

Interface

Example Usage

Implementation

Unordered array

Ordered array

Linked list

Summary

Time complexity of insertion?

- Binary search lets us find the insertion point in $O(\log n)$ time
- ...but we still have to potentially shift up to n elements, which is $O(n)$

Abstraction

ADTs

Stacks

Queues

Sets

Interface

Example Usage

Implementation

Unordered array

Ordered array

Linked list

Summary

How do we delete an element?

- Use binary search to find the element
- If the element exists, shift everything greater than it down

Time complexity?

- Binary search lets us find the element in $O(\log n)$ time
- ...but we still have to potentially shift up to n elements, which is $O(n)$

Abstraction

ADTs

Stacks

Queues

Sets

Interface

Example Usage

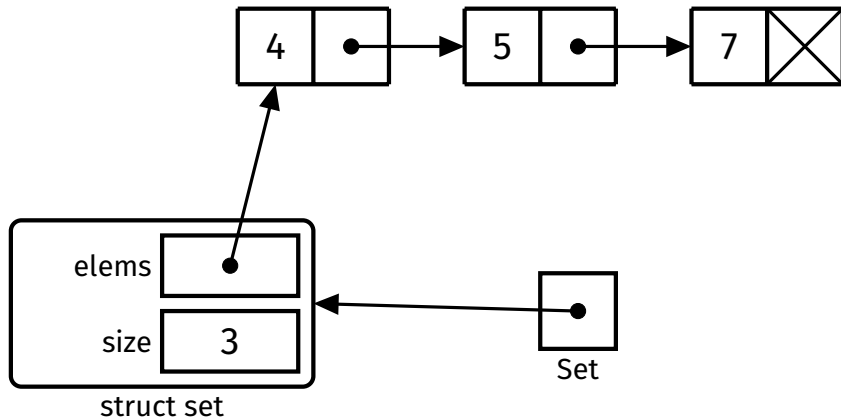
Implementation

Unordered array

Ordered array

Linked list

Summary



Abstraction

ADTs

Stacks

Queues

Sets

Interface

Example Usage

Implementation

Unordered array

Ordered array

Linked list

Summary

How do we check if an element exists?

- Traverse the list $\Rightarrow O(n)$

```
bool SetContains(Set s, int elem) {  
    for (struct node *curr = s->elems; curr != NULL; curr = curr->next) {  
        if (curr->elem == elem) {  
            return true;  
        }  
    }  
  
    return false;  
}
```

Abstraction

ADTs

Stacks

Queues

Sets

Interface

Example Usage

Implementation

Unordered array

Ordered array

Linked list

Summary

We always have to traverse the list from the start. Therefore...

- Insertion and deletion are also $O(n)$

However, this analysis hides a crucial advantage of linked lists:

- Finding the insertion/deletion point is $O(n)$
- But inserting/deleting a node is $O(1)$, as no shifting is required

Abstraction

ADTs

Stacks

Queues

Sets

Interface

Example Usage

Implementation

Summary

Data Structure	Contains	Insert	Delete
Unordered array	$O(n)$	$O(n)$	$O(n)$
Ordered array	$O(\log n)$	$O(n)$	$O(n)$
Ordered linked list	$O(n)$	$O(n)$	$O(n)$

Abstraction

ADTs

Stacks

Queues

Sets

Interface

Example Usage

Implementation

Summary

<https://forms.office.com/r/riGKCze1cQ>

