University of Guelph Prof. Pascal Matsakis

CIS2520

2. Stacks and Queues

Reading suggestion: Chapter 7 (except Section 7.7) of the textbook

STACK ADT

interface
sequential implementation
linked implementation
applications

STACK ADT: The Driveway Example 2.3

build a driveway	()
park your crappy yellow car	(yellow)
park my beautiful blue jaguar	(yellow,blue)
park the grey car	(yellow,blue,grey)
move the last car	(yellow,blue)
move the last car	(yellow)
park the grey car	(yellow,grey)

Reading suggestion: Chapter 7 (except Section 7.7) of the textbook

STACK ADT: Main Operations 2.4

build a driveway	**Create**: ∅ ➔ Stack[T]
park a car	**Push**: TxStack[T] ➔ Stack[T]
move the last car	**Pop**: Stack[T] ➔ Stack[T]
determine whether full	**Full**: Stack[T] ➔ Boolean
determine whether empty	**Empty**: Stack[T] ➔ Boolean
find the number of cars	**Size**: Stack[T] ➔ N
find the last car	**Top**: Stack[T] ➔ T

Reading suggestion: Chapter 7 (except Section 7.7) of the textbook

STACK ADT: Stacks vs. Lists 2.5

Create: ∅ ➔ Stack[T]	**Create**: ∅ ➔ List[T]
Push: TxStack[T] ➔ Stack[T]	**Insert**: TxNxList[T] ➔ List[T]
Pop: Stack[T] ➔ Stack[T]	**Remove**: NxList[T] ➔ List[T]
Full: Stack[T] ➔ Boolean	**Full**: List[T] ➔ Boolean
Empty: Stack[T] ➔ Boolean	**Empty**: List[T] ➔ Boolean
Size: Stack[T] ➔ N	**Size**: List[T] ➔ N
Top: Stack[T] ➔ T	**Peek**: NxList[T] ➔ T

The Stack ADT is a restriction of the List ADT.
A stack is a **LIFO** (Last In, First Out) structure.

STACK ADT: Preconditions and Postconditions 2.6

Create: ∅ ➔ Stack[T]	**constructor**
Push: TxStack[T] ➔ Stack[T]	**mutators**
Pop: Stack[T] ➔ Stack[T]	
Full: Stack[T] ➔ Boolean	**accessors**
Empty: Stack[T] ➔ Boolean	
Size: Stack[T] ➔ N	
Top: Stack[T] ➔ T	

{¬Full(S)} Push(I,S)	**preconditions**
{¬Empty(S)} Top(S)	
Push(I,S) {Top(S)=I}	**postconditions**
Pop(S) {Size(S)=Size(**old** S)−1}	
Empty(S)=(Size(S)=0)	**invariant**

STACK ADT: Axioms 2.7

Create: ∅ ➔ Stack[T]

constructor

Push: TxStack[T] ➔ Stack[T]
Pop: Stack[T] ➔ Stack[T]

mutators

Full: Stack[T] ➔ Boolean
Empty: Stack[T] ➔ Boolean
Size: Stack[T] ➔ N
Top: Stack[T] ➔ T

accessors

Empty(Create())
¬ Empty(Push(I,S))
¬ Full(Pop(S))
Top(Push(I,S))=I
Pop(Push(I,S))=S

axioms

Reading suggestion: Chapter 7 (except Section 7.7) of the textbook

Stack ADT

INTERFACE

sequential implementation

linked implementation

applications

INTERFACE: Example

```
#include "StackType.h"     /* imports the concrete data structure */
                          /* definitions of Item and Stack */

extern void Initialize (Stack *S);
extern void Push (Item I, Stack *S);
extern void Pop (Stack *S);
extern int Full (Stack *S);
extern int Empty (Stack *S);
extern int Size (Stack *S);
extern void Top (Stack *S, Item *I);
extern void Destroy (Stack *S);
```

StackInterface.h

```
#include "StackInterface.h"
......
```

myProgram.c

Reading suggestion: Chapter 7 (except Section 7.7) of the textbook

Stack ADT

interface

SEQUENTIAL IMPLEMENTATION

linked implementation

applications

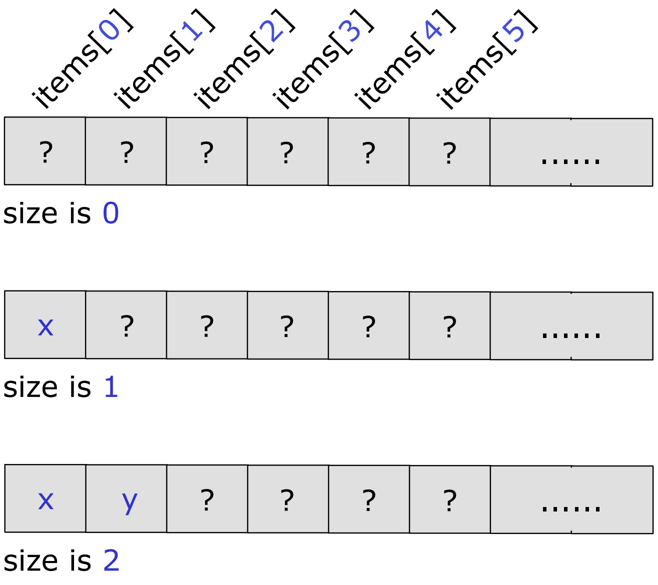
SEQUENTIAL IMPLEMENTATION: Example (1/3) 2.11

```
#include "ArbitraryInterface.h"        /* or could simply be, e.g., */
typedef Arbitrary Item;               /* typedef int Item; */

#define MAXSTACKSIZE 100
typedef struct {
          Item items[MAXSTACKSIZE];
          int size;
} Stack;
```

StackType.h

SEQUENTIAL IMPLEMENTATION: Example (2/3) 2.12



size is 0

size is 1

size is 2

SEQUENTIAL IMPLEMENTATION: Example (3/3) 2.13

```
#include "StackInterface.h"
...........

void Initialize (Stack *S) {
          ......
}

void Push (Item I, Stack *S) {
          ......
}

......
```

StackImplementation.c

Reading suggestion: Chapter 7 (except Section 7.7) of the textbook

Stack ADT

interface

sequential implementation

LINKED IMPLEMENTATION

applications

LINKED IMPLEMENTATION: Example (1/3)

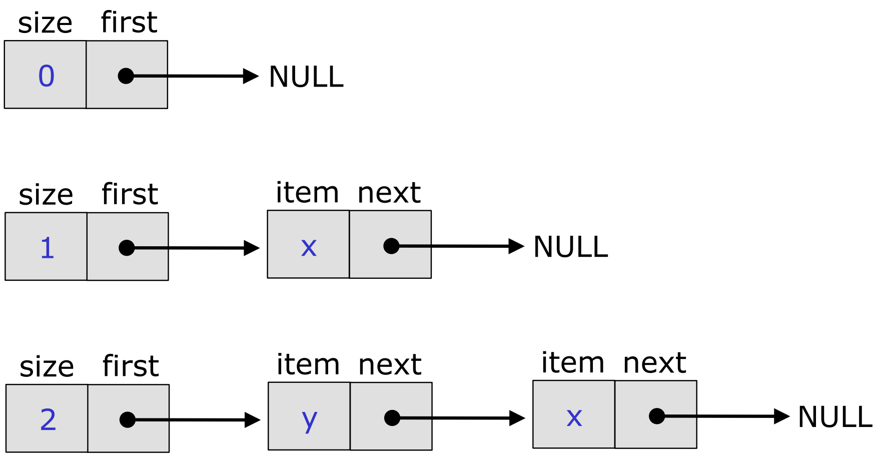
```
#include "ArbitraryInterface.h”
typedef Arbitrary Item;

typedef struct StackNodeTag {
                Item item;
                struct StackNodeTag *next;
} StackNode;

typedef struct {
                int size;
                StackNode *first;
} Stack;
```

StackType.h

LINKED IMPLEMENTATION: Example (2/3)



LINKED IMPLEMENTATION: Example (3/3) 2.17

```
#include "StackInterface.h"
............

void Initialize (Stack *S) {
          ......
}

void Push (Item I, Stack *S) {
          ......
}

......
```

StackImplementation.c

Reading suggestion: Chapter 7 (except Section 7.7) of the textbook

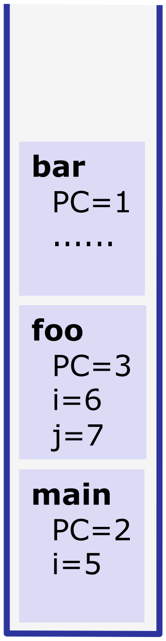
Stack ADT
interface
sequential implementation
linked implementation
APPLICATIONS

APPLICATIONS: Execution Stack

```
main () {
        int i=5;
        foo(i+1);
        ......
}

foo (int i) {
        int j;
        j=i+1;
        bar(j);
        ......
}

bar (int k) {
        ......
        ......
}
```



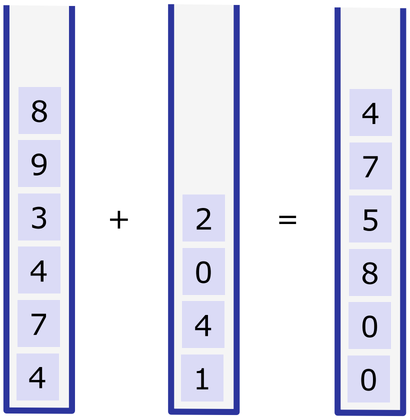
bar
PC=1
......

foo
PC=3
i=6
j=7

main
PC=2
i=5

APPLICATIONS: Addition (with large operands)

474398
+
1402
=
475800



APPLICATIONS: Other Examples

Direct applications:

- ✧ Page-visited history in a web browser
- ✧ Undo sequence in a text editor

Indirect applications:

- ✧ Auxiliary data structure for algorithms
- ✧ Component of other data structures

Reading suggestion: Chapter 7 (except Section 7.7) of the textbook

QUEUE ADT

interface
sequential implementation
linked implementation
applications

QUEUE ADT: The Tollbooth Example 2.23

install a tollbooth	()
here comes my beautiful blue jaguar	(blue)
here comes the grey car	(blue,grey)
here comes your crappy yellow car	(blue,grey,yellow)
next please!	(grey,yellow)

QUEUE ADT: Main Operations 2.24

install a tollbooth	**Create**: ∅ ➔ Queue[T]
here comes a car	**Enqueue**: TxQueue[T] ➔ Queue[T]
next please!	**Dequeue**: Queue[T] ➔ Queue[T]
determine whether full	**Full**: Queue[T] ➔ Boolean
determine whether empty	**Empty**: Queue[T] ➔ Boolean
find the number of cars	**Size**: Queue[T] ➔ N
find the first car	**Head**: Queue[T] ➔ T
find the last car	**Tail**: Queue[T] ➔ T

QUEUE ADT: Queues vs. Lists 2.25

Create: ∅ ➔ Queue[T]
Enqueue: TxQueue[T] ➔ Queue[T]
Dequeue: Queue[T] ➔ Queue[T]
Full: Queue[T] ➔ Boolean
Empty: Queue[T] ➔ Boolean
Size: Queue[T] ➔ N
Head: Queue[T] ➔ T
Tail: Queue[T] ➔ T

Create: ∅ ➔ List[T]
Insert: TxNxList[T] ➔ List[T]
Remove: NxList[T] ➔ List[T]
Full: List[T] ➔ Boolean
Empty: List[T] ➔ Boolean
Size: List[T] ➔ N
Peek: NxList[T] ➔ T

The Queue ADT is a restriction of the List ADT.
A queue is a **FIFO** (First In, First Out) structure.

Reading suggestion: Chapter 7 (except Section 7.7) of the textbook

QUEUE ADT: Preconditions and Postconditions 2.26

Create: ∅ ➔ Queue[T]

constructor

Enqueue: TxQueue[T] ➔ Queue[T]
Dequeue: Queue[T] ➔ Queue[T]

mutators

Full: Queue[T] ➔ Boolean
Empty: Queue[T] ➔ Boolean
Size: Queue[T] ➔ N
Head: Queue[T] ➔ T
Tail: Queue[T] ➔ T

accessors

{¬Full(Q)} Enqueue(I,Q)
{¬Empty(Q)} Head(Q)

preconditions

Enqueue(I,Q) {¬Empty(Q)}
Dequeue(Q) {Size(Q)=Size(**old** Q)−1}

postconditions

Empty(Q)=(Size(Q)=0)

invariant

Reading suggestion: Chapter 7 (except Section 7.7) of the textbook

QUEUE ADT: Axioms

Create: ∅ ➔ Queue[T]
Enqueue: TxQueue[T] ➔ Queue[T]
Dequeue: Queue[T] ➔ Queue[T]
Full: Queue[T] ➔ Boolean
Empty: Queue[T] ➔ Boolean
Size: Queue[T] ➔ N
Head: Queue[T] ➔ T
Tail: Queue[T] ➔ T

constructor

mutators

accessors

Empty(Create())
¬ Empty(Enqueue(I,Q))
¬ Full(Dequeue(Q))
Tail(Enqueue(I,Q))=I
Dequeue(Enqueue(I,Q))=Q

axioms

Reading suggestion: Chapter 7 (except Section 7.7) of the textbook

Queue ADT

INTERFACE

sequential implementation
linked implementation
applications

INTERFACE: Example

```
#include "QueueType.h"    /* imports the concrete data structure */
                         /* definitions of Item and Queue */

extern void Initialize (Queue *Q);
extern void Enqueue (Item I, Queue *Q);
extern void Dequeue (Queue *Q);
extern int Full (Queue *Q);
extern int Empty (Queue *Q);
extern int Size (Queue *Q);
extern void Head (Queue *Q, Item *I);
extern void Tail (Queue *Q, Item *I);
extern void Destroy (Queue *Q);
```

QueueInterface.h

```
#include "QueueInterface.h"
......
```

myProgram.c

Reading suggestion: Chapter 7 (except Section 7.7) of the textbook

Queue ADT

interface

SEQUENTIAL IMPLEMENTATION

linked implementation

applications

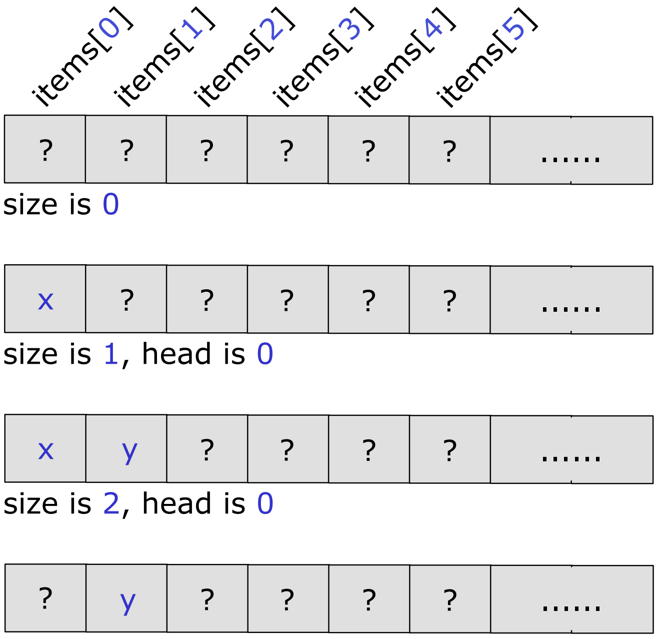
SEQUENTIAL IMPLEMENTATION: Example (1/3)

```
#include "ArbitraryInterface.h"
typedef Arbitrary Item;

#define MAXQUEUESIZE 100
typedef struct {
        Item items[MAXQUEUESIZE];
        int size;
        int head;
} Queue;
```

QueueType.h

SEQUENTIAL IMPLEMENTATION: Example (2/3)



items[0]	items[1]	items[2]	items[3]	items[4]	items[5]	
?	?	?	?	?	?	......

size is 0

x	?	?	?	?	?	......

size is 1, head is 0

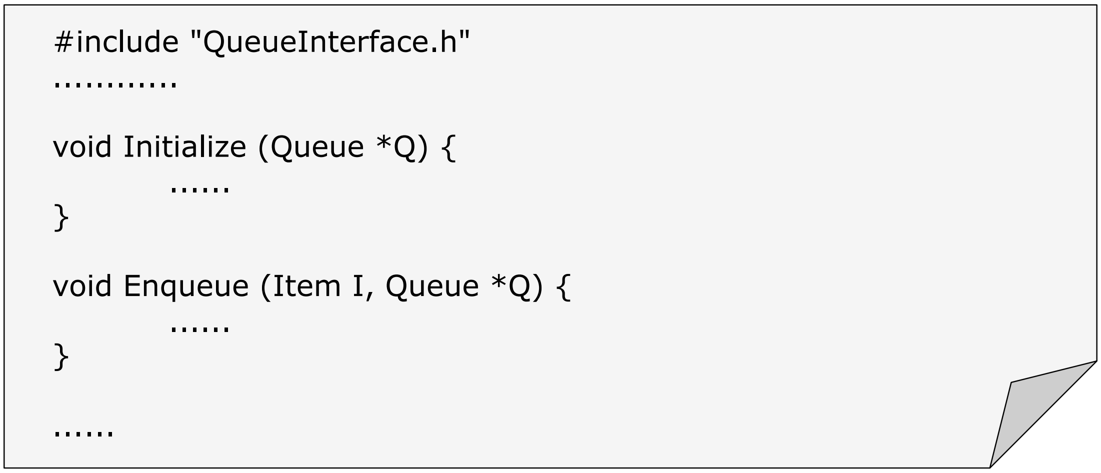
x	y	?	?	?	?	......

size is 2, head is 0

?	y	?	?	?	?	......

size is 1, head is 1

SEQUENTIAL IMPLEMENTATION: Example (3/3) 2.33



```
#include "QueueInterface.h"
...........

void Initialize (Queue *Q) {
          ......
}

void Enqueue (Item I, Queue *Q) {
          ......
}

......
```

QueueImplementation.c

Reading suggestion: Chapter 7 (except Section 7.7) of the textbook

Queue ADT
interface
sequential implementation
LINKED IMPLEMENTATION
applications

LINKED IMPLEMENTATION: Example (1/3)

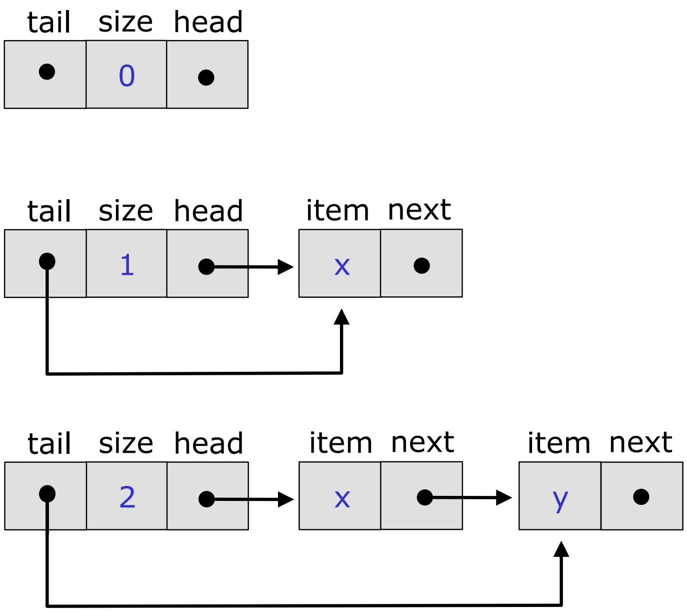
```
#include "ArbitraryInterface.h”
typedef Arbitrary Item;

typedef struct QueueNodeTag {
                Item item;
                struct QueueNodeTag *next;
} QueueNode;

typedef struct {
                int size;
                QueueNode *head;
                QueueNode *tail;
} Queue;
```

QueueType.h

LINKED IMPLEMENTATION: Example (2/3)



LINKED IMPLEMENTATION: Example (3/3)

```
#include "QueueInterface.h"
…………

void Initialize (Queue *Q) {
          ……
}

void Enqueue (Item I, Queue *Q) {
          ……
}

……
```

QueueImplementation.c

Reading suggestion: Chapter 7 (except Section 7.7) of the textbook

Queue ADT
interface
sequential implementation
linked implementation

APPLICATIONS

APPLICATIONS: Examples

Direct applications:

- Waiting lists
- Access to shared resources (e.g., printer)

Indirect applications:

- Auxiliary data structure for algorithms
- Component of other data structures

Reading suggestion: Chapter 7 (except Section 7.7) of the textbook

ADT
interface
sequential implementation
linked implementation
applications

THE END