SCC.121: Fundamentals of Computer Science
Lecture 1: Introduction to Data Structures and Abstract Data Types

Amit Chopra

amit.chopra@lancaster.ac.uk

What this is about

In this course, we are interested in three things:

- Abstract Data Type (ADT): The facilities and functionality.
  What it does.
- The data structure employed by the ADT.
  How it does what it does.
- The implementation of the data structure.
  How it is stored and manipulated.

Abstract Data Type

Data Structure

Implementation

Why its important

- The facilities and functionality of an Abstract Data Type (ADT) - *What it does.*
    - As users of an ADT, it is important to understand what the functionality is and how we can access that functionality.

- The data structure employed by the ADT - *How it does what it does.*
    - As programmers of an ADT, we need to understand what options are available to implement an ADT, evaluate them and pick the best one.

- The implementation of the data structure - *How it is stored and manipulated.*
    - As Computer Scientists, we should understand the basics of the data structure.

An ADT example: The Queue

- The queue ADT is defined by the following structure and operations:
- A queue is an ordered collection of items which are added at one end, called the “tail,” and removed from the other end, called the “head”.
- Queues maintain a FIFO ordering property.
  - First In First Out (FIFO)

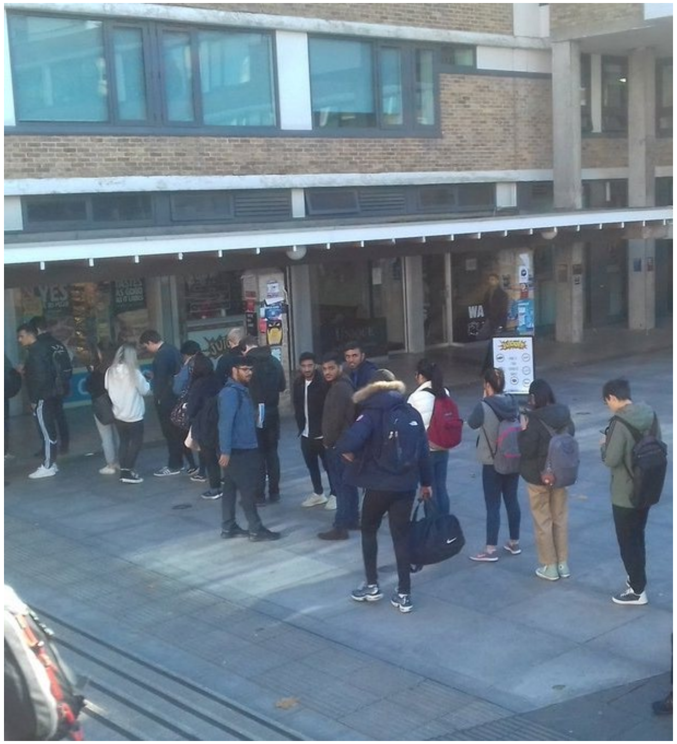


Image: https://twitter.com/MattHanley/status/1186977278003159040

Queue ADT: operations

Queue()

- Creates a new queue that is empty.
- Parameters: none
- Returns: empty queue

enqueue(item)

- Adds a new item to the tail of the queue.
- Parameters: the item to add.
- Returns: nothing

dequeue()

- If not empty, removes the front item from the queue.
- Parameters: none
- Returns: The item removed from the queue.

isEmpty()

- Tests to see whether the queue is empty.
- Parameters: none
- Returns: (boolean) true if empty, false otherwise

size()

- Counts the number of items in the queue.
- Parameters: none.
- Returns: the number of items in the queue.

The Queue Abstract Data Type

- This description should be adequate enough to give you an understanding of what a queue is capable of.
- Perhaps example code could be provided showing the queue being used in practice.
- It could be elaborated with conceptual diagrams and examples of the queue before and after an operation is carried out.
- But, those conceptual diagrams and examples could also be abstract and bear little relation to how the queue is actually implemented.

Queue behaviour: enqueue

size() = 4

enqueue ()

size() = 5

Queue behaviour: dequeue

size() = 5

dequeue()

size() = 4

Why is it called “abstract”?

- The definition of an ADT only mentions what operations can be performed but not how these operations will be implemented.

- The ADT does not specify how the data will be organized in memory, or the algorithms to be used for implementing the operations.

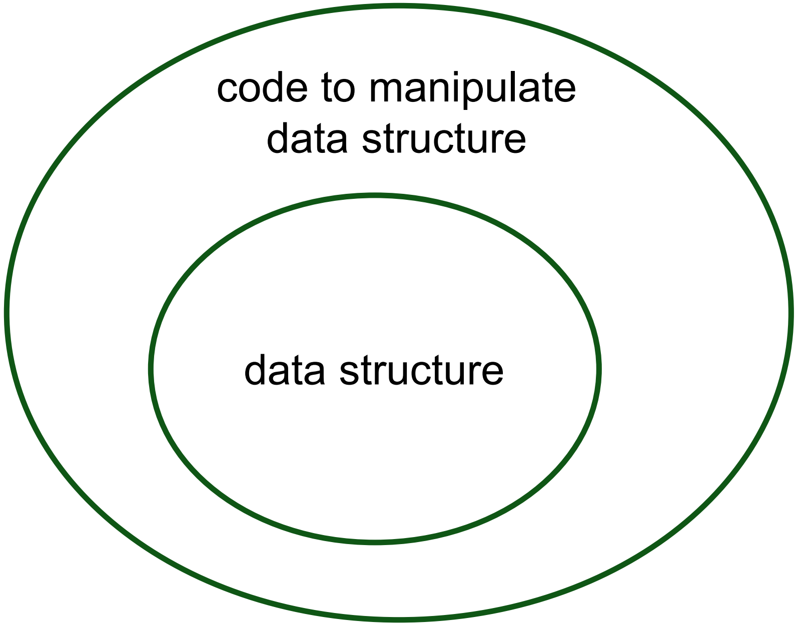
- Abstract because it is an **implementation independent** view.

- The process of providing only the necessary information and hiding the internal details is known as **abstraction**.

What’s the difference between data structures and an abstract data type?

- A **data structure** is the physical representation of the structure of the data being stored in memory.

- An **abstract data type** is both the data structure and the procedures/functions which manipulate that data structure.



Level of support for ADTs

- The level of support for ADTs differs among programming languages.

- Most languages will allow us to introduce data structures as user-defined types.
  - this is a minimal level of support.

- Some languages will allow us to **encapsulate** the data structure and offer operations as part of the definition of the ADT.

Encapsulation

- The only way the user can interact with a variable of that ADT is through an ***interface***, a set of procedures that operate on the data structure.
- The user cannot directly manipulate the physical stored data structure.
- Encapsulation is a highly desirable property for Software Engineering.

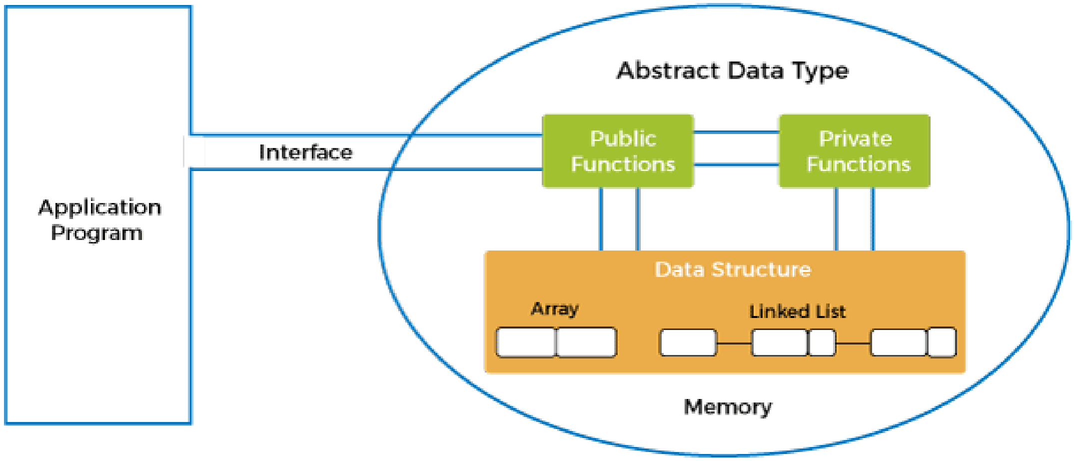


Image taken from: https://www.javatpoint.com/abstract-data-type-in-data-structure

“Strong” encapsulation

- If the encapsulation is “strong” then details of how the data structure is actually stored can be completely hidden from the user.

- The user will have a conceptual (or “abstract”) understanding of what the ADT does, not how it does it.

Encapsulation in ‘C’

- ‘C’ does not offer encapsulation.
- So the best we can do is:
  - declare a new data structure to model our ADT
  - provide a set of procedures/functions that operate on the data structure.

- For example, a string in ‘C’ is represented as a character array.
- A set of operations on strings is provided in a standard library.

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13
value	‘H’	‘e’	‘l’	‘l’	‘o’	‘,’	‘ ‘	‘W’	‘o’	‘r’	‘l’	‘d’	‘!’	‘\0’

Encapsulation in Java

- Java does offer encapsulation.
- So we can define a new ADT as a class.

- A class contains both:
  - the data structure and
  - the set of procedures/functions (called methods in Java) that operate on the data structure.

- The data structure can be declared as private and so are “hidden” from the class user.
  - If the class contains methods which are not part of the ADT interface, these can also be private.

- Everything is a class in Java!

Possible Queue Implementation

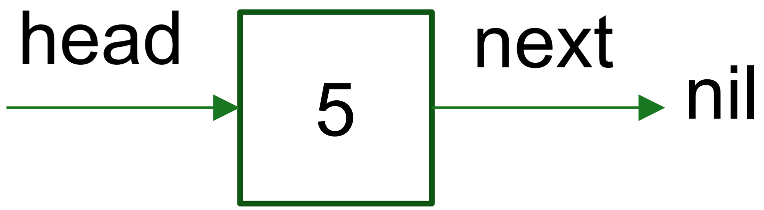
Data structure

```
Element {
    data;
    nextPtr ;
}
```

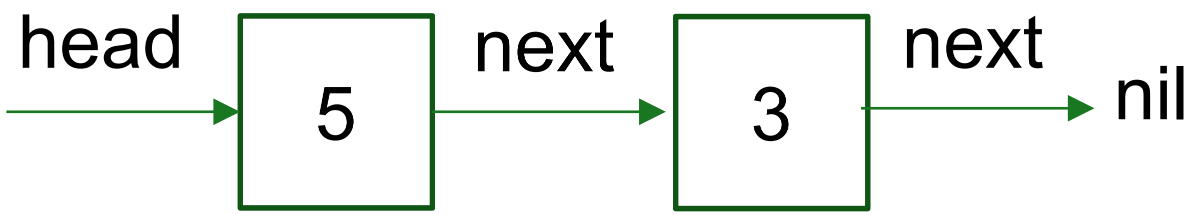
- head is pointer to the head of the queue
- Empty queue implies head = nil (does not point to anything)
- Initially head = nil

Queue Evolution

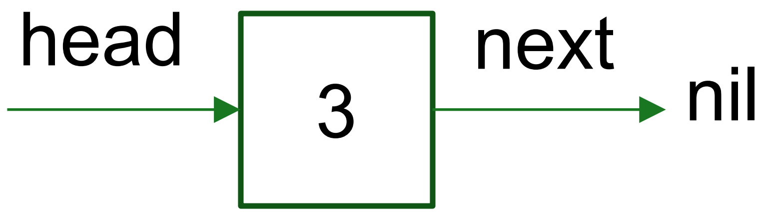
- Enqueue(5)



- Enqueue(3)

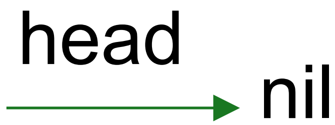


- Dequeue() returns 5



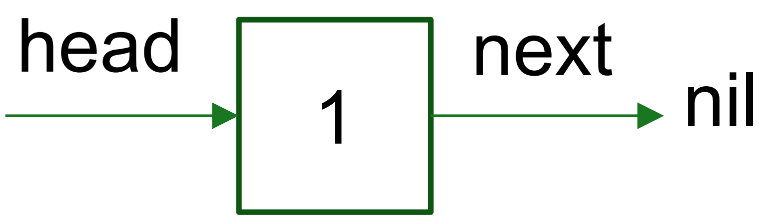
Queue Evolution (continued)

- Dequeue() returns 3



- Dequeue returns nothing; “Empty queue”

- Enqueue(1)



Enqueue(data)

```
el  = create new Element;
el.data = data;
el.next = nil
if (head is nil)   // queue is empty
      head = el
else //get to end of queue and it there
      tmp = head
      while(tmp.next is not nil)
            tmp = tmp.next
      tmp.next =el
```

Arrays

- Declare an array in C:

size

contains 26 integer elements

```
int counts [26];
```

data type

variable name

indicates we are declaring an array

An array example

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
indices	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
counts	0	0	0	1	1	0	0	1	0	0	0	3	0	0	2	0	0	1	0	0	0	0	1	0	0	0

- Here is the array after we have counted up the characters in “hello world”
- To increment the count for ‘l’, we can write
  - `counts[11] = counts[11] + 1;`
- To print the value of the count for ‘h’ we can write
  - `printf(“%d\n”,counts[7]);`

Accessing a single element

- To access a single element of an array, we use an index value.

```
counts[11] = counts[11] + 1;

printf("%d",counts[7]);
```

- Once we have selected a single element using an index, we can treat that element as we would treat any simple variable of that type.

Array indexes

- In C (and Java), the type of value used for an index is an int.

- Anywhere we would expect an index value, we could have:
  - a literal int value,
  - an int variable,
  - an expression that evaluated to an int,
  - a method call that returns an int.

```
counts[x] = 7;
counts[x+2] = 144;
z = z + counts[nextElement(4)];
```

Upper and Lower Bounds

counts	0	0	0	1	0	.	0	0	1	0	0	0	elements
	0	1	2	3	4	.	20	21	22	23	24	25	indices

- **Upper bound (upb)**: index of the last element
- **Lower bound (lwb)**: index of the first element
  - upb: 25; lwb: 0 for `counts'

- In C (and Java), the lwb is **always** zero.

- The possible (valid) values for an index lie in the range *lwb .. upb*.
- In this example, the index range would be *0 .. 25*.

Array and for loop

- To initialize the counts array, we have a loop as follows:

```
for (int i = 0; i < 26; i++ )
    counts[i] = 0;
```

Bounded Length Queue Using Arrays

- Assume queue can have at most MAX_SIZE elements
- Can use array of length MAX_SIZE
- If enqueue(), when queue size is MAX_SIZE, it fails: “Queue Full”

Summary

- Abstract data types (ADTs)
  - Emphasize the user’s point of view
  - Queue
- Data structures help implement ADTs
  - Different data structures yield different implementations of an ADT
  - Queue via pointers between elements
  - Queue via arrays