Basic Data Structures

Stacks, Queues and Lists

Ruoyu Wang

Maths and Stats Programming Club
University College London

r.wang.13@ucl.ac.uk

November 26, 2014

Overview

1. Introduction
2. Primitive Data Types
3. Composite Data Types
4. Abstract Data Types
   - Stacks
   - Queues
   - Pointers
   - Lists

When the world is built...

'Let there be data.'
'Okay, now we have the data. Let us store the data... wait, where is my list?'
'What? There's no list in this programming language?! I have to create one!'

From the origin

In Python, when we are going to store some values (e.g. randomly generate some numbers), we usually use the codes below:

Python: Generate Random Numbers

```
list_0=[]
for i in range(10):
    list_0.append(random.randint(1,100))
```

From the origin

In Python, when we are going to store some values (e.g. randomly generate some numbers), we usually use the codes below:

Python: Generate Random Numbers

```
list_0=[]
for i in range(10):
    list_0.append(random.randint(1,100))
```

Here we used an abstract data structure, namely a list ($list_0$). How are the ten numbers stored?

From the origin

From the origin

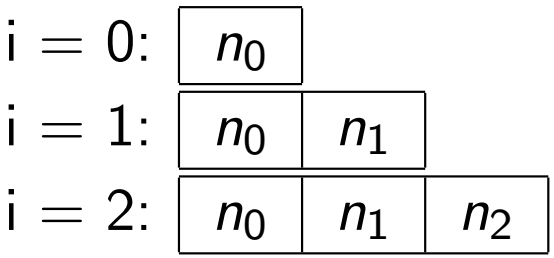
i $= 0$: $\boxed{n_0}$

From the origin

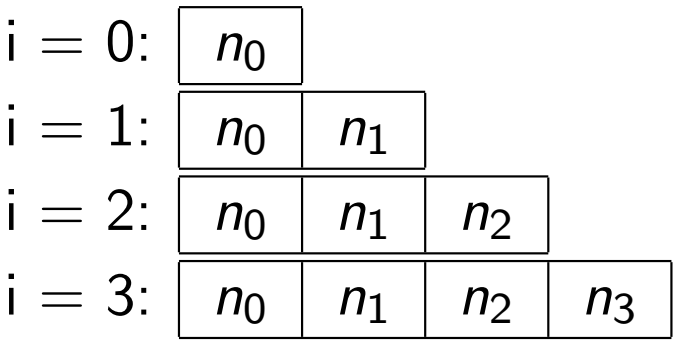
i = 0: n_0

i = 1: n_0 | n_1

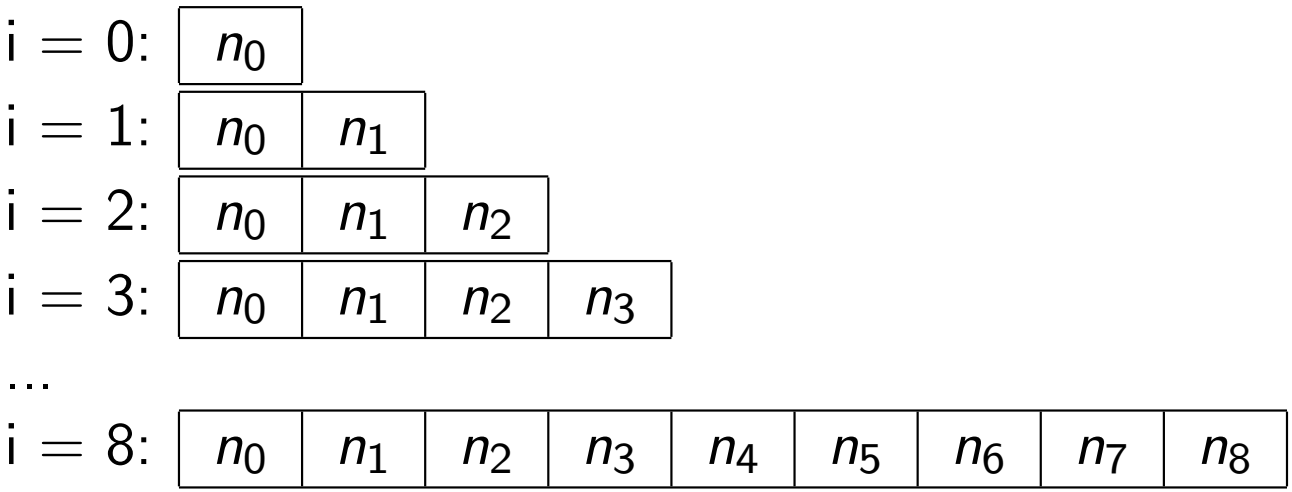
From the origin



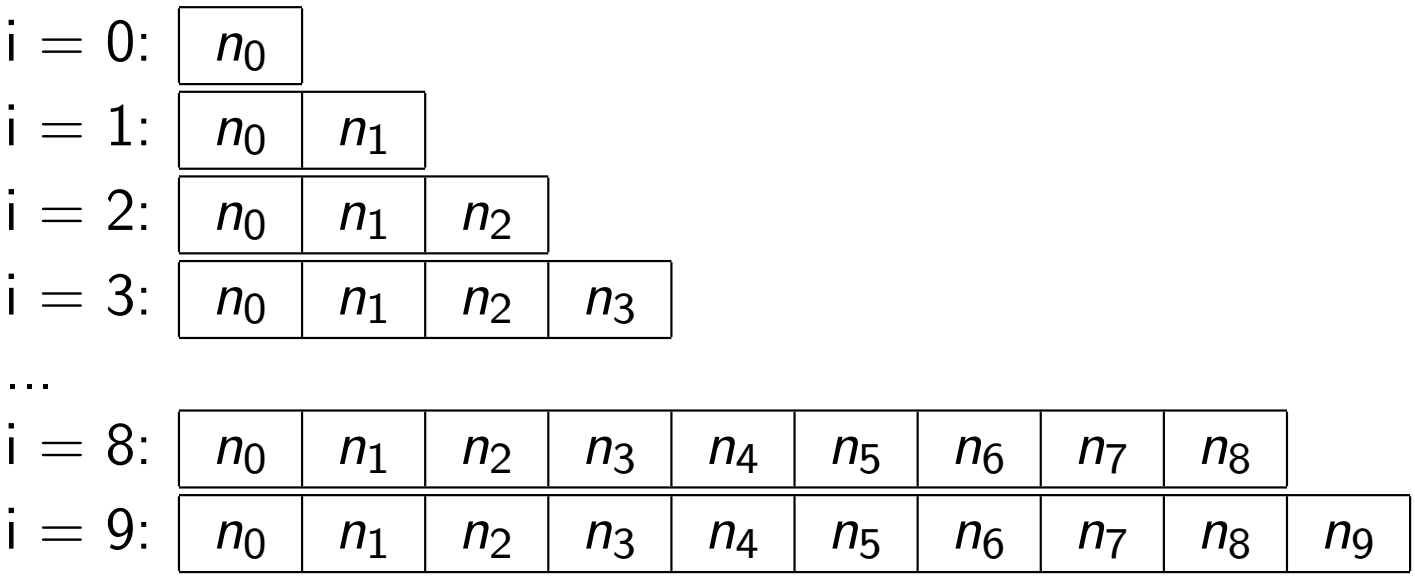
From the origin



From the origin



From the origin



From the origin

i = 0: | n_0 |

i = 1: | n_0 | n_1 |

i = 2: | n_0 | n_1 | n_2 |

i = 3: | n_0 | n_1 | n_2 | n_3 |

...

i = 8: | n_0 | n_1 | n_2 | n_3 | n_4 | n_5 | n_6 | n_7 | n_8 |

i = 9: | n_0 | n_1 | n_2 | n_3 | n_4 | n_5 | n_6 | n_7 | n_8 | n_9 |

So the way that we place data into a list is appending a cell at the end each time.

Different Data Types

We are going to talk about three main classes of data types:

- Primitive Data Types (PDT)
- Composite Data Types (CDT)
- Abstract Data Types (ADT)

The lower is the more advanced.

Primitive Data Types

Primitive Data Types (PDT)

Primitive Data Types usually indicate the most basic types that a programming language could provide. They are something like a building brick in Mine-craft.

On the next page, you are going to guess what names of those types are. But don't worry, you know them!

List of Primitive Data Types

Type	Example

List of Primitive Data Types

Type	Example
	2, 3, 4, 5, 6, 7

List of Primitive Data Types

Type	Example
Integer	2, 3, 4, 5, 6, 7

List of Primitive Data Types

Type	Example
Integer	2, 3, 4, 5, 6, 7
	True, False

List of Primitive Data Types

Type	Example
Integer	2, 3, 4, 5, 6, 7
Boolean	True, False

List of Primitive Data Types

Type	Example
Integer	2, 3, 4, 5, 6, 7
Boolean	True, False
	"Hello, world!"

List of Primitive Data Types

Type	Example
Integer	2, 3, 4, 5, 6, 7
Boolean	True, False
String	"Hello, world!"

List of Primitive Data Types

Type	Example
Integer	2, 3, 4, 5, 6, 7
Boolean	True, False
String	"Hello, world!"
	'c', 'h', 'a', 'r'

List of Primitive Data Types

Type	Example
Integer	2, 3, 4, 5, 6, 7
Boolean	True, False
String	"Hello, world!"
Character	'c', 'h', 'a', 'r'

List of Primitive Data Types

Type	Example
Integer	2, 3, 4, 5, 6, 7
Boolean	True, False
String	"Hello, world!"
Character	'c', 'h', 'a', 'r'
	3.14, 2.71

List of Primitive Data Types

Type	Example
Integer	2, 3, 4, 5, 6, 7
Boolean	True, False
String	"Hello, world!"
Character	'c', 'h', 'a', 'r'
Float	3.14, 2.71

Composite Data Types

Composite Data Types (CDT)

Composite Data Types indicate data types which are a composition (set) of primitive types.

Composite Data Types

Composite Data Types (CDT)

Composite Data Types indicate data types which are a composition (set) of primitive types.

For example, an array is a composite data types, which is known as matrix in Mathematics. The difference between an array and other primitive types is that an array can get more than one value. (For an integer variable, it can only contain one value, namely the number. But for an array, it can have several cells of values, just as a matrix can have $m \times n$ numbers.)

n_{11}	n_{12}	n_{13}
n_{21}	n_{22}	n_{23}
n_{31}	n_{32}	n_{33}

0.1	0.2	0.3
0.4	0.5	0.6
0.7	0.8	0.9

'A'	'B'	'C'
'D'	'E'	'F'
'G'	'H'	'I'

The array above contains 9 values, which are 9 integers (or floats, or anything else).

Composite Data Types

Another example of CDT is a tuple, which is of format:

Python: A tuple

```
tup1=(9, 26, 281)
```

Composite Data Types

Another example of CDT is a tuple, which is of format:

Python: A tuple

```
tup1=(9, 26, 281)
```

A tuple is like an array (matrix) of $1 \times n$, or a pair of numbers. Since it contains more than one value, it is a CDT.

Composite Data Types

Another example of CDT is a tuple, which is of format:

Python: A tuple

```
tup1=(9, 26, 281)
```

A tuple is like an array (matrix) of $1 \times n$, or a pair of numbers. Since it contains more than one value, it is a CDT.
Caution: A float type is not a CDT. Many people split a float into two parts, an integer part and a decimal part, which they regard as two values. This is incorrect simply because in most languages, a float type is simply a series of binary numbers, not two parts.

Composite Data Types

Another example of CDT is a tuple, which is of format:

Python: A tuple

```
tup1=(9, 26, 281)
```

A tuple is like an array (matrix) of $1 \times n$, or a pair of numbers. Since it contains more than one value, it is a CDT.
Caution: A float type is not a CDT. Many people split a float into two parts, an integer part and a decimal part, which they regard as two values. This is incorrect simply because in most languages, a float type is simply a series of binary numbers, not two parts.
In contrast to float type, a complex type ($a + bi$) is a CDT. (Why?)

Classes

All class types are CDTs.
A class defines an object with two things: Attributes (data) and methods (functions). Since it can be constructed to comprise two or more attributes, it is a CDT.
(You may wish to revisit Niko's slides upon OOP)

Abstract Data Types

Abstract Data Types (ADT)

An abstract data type (ADT) is a mathematical model for a certain class of data structures that have similar behaviours.

Abstract Data Types

Abstract Data Types (ADT)

An abstract data type (ADT) is a mathematical model for a certain class of data structures that have similar behaviours.

Using a metaphor, an ADT is like an algebraic structure (e.g. groups, rings) in Mathematics.

Abstract Data Types

Abstract Data Types (ADT)

An abstract data type (ADT) is a mathematical model for a certain class of data structures that have similar behaviours.

Using a metaphor, an ADT is like an algebraic structure (e.g. groups, rings) in Mathematics.
Don't be frightened by 'Mathematics'! You are only required to learn specific structures at this stage.
We are going to talk about three important ADTs: Stacks, queues and linked lists.

Stacks

Stack

Stacks are dynamic sets in which the element removed from the set by the DELETE operation is specified. They obey Last-In-First-Out (LIFO) rule.

Stacks

Stack

Stacks are dynamic sets in which the element removed from the set by the DELETE operation is specified. They obey Last-In-First-Out (LIFO) rule.

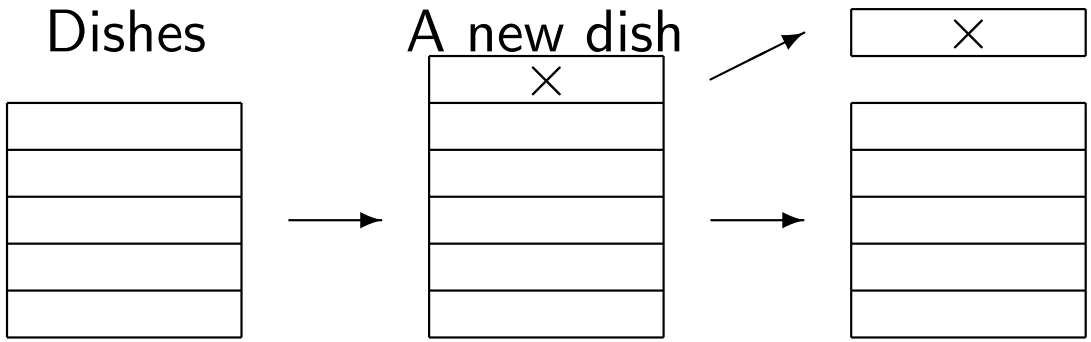
Imagine you have stacked dishes in your cupboard and you are going to take a dish for dinner. You can't get the one at the bottom of the dish stack, otherwise you will break all dishes it underpins. The only dish you can take is the one at the top, which is the last one you have stacked. Now obeying LIFO rule, the dish stack can be viewed as a stack.

Stacks

Stack

Stacks are dynamic sets in which the element removed from the set by the DELETE operation is specified. They obey Last-In-First-Out (LIFO) rule.

Imagine you have stacked dishes in your cupboard and you are going to take a dish for dinner. You can't get the one at the bottom of the dish stack, otherwise you will break all dishes it underpins. The only dish you can take is the one at the top, which is the last one you have stacked. Now obeying LIFO rule, the dish stack can be viewed as a stack.



Only the last dish put in could be the first to take out.

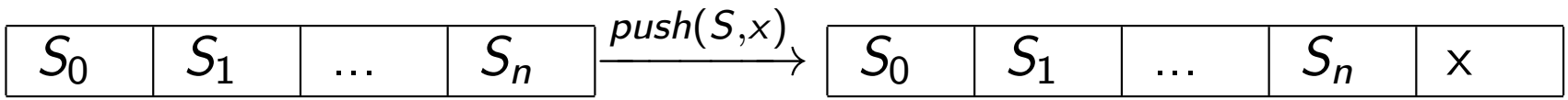
Stack Operations

We define a stack S using an array $S[0, 1, ..., n]$. Then we are going to define some basic operations on S.

Stack Operations

We define a stack S using an array $S[0, 1, ..., n]$. Then we are going to define some basic operations on S.

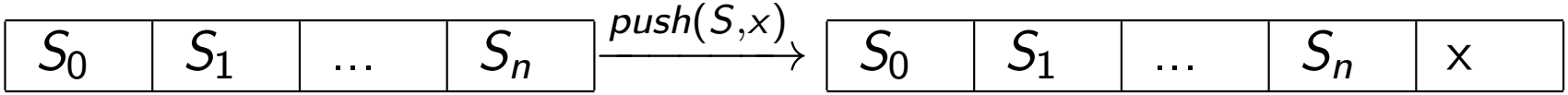
- $push(S, x)$: INSERT x into S. (At the end of S)



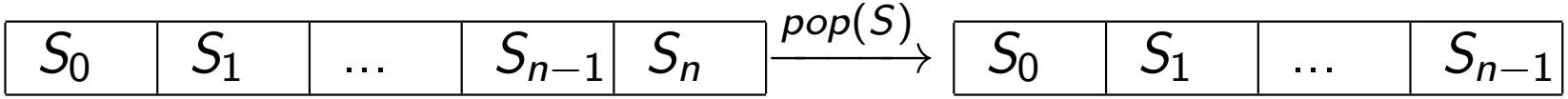
Stack Operations

We define a stack S using an array $S[0, 1, ..., n]$. Then we are going to define some basic operations on S.

- $push(S, x)$: INSERT x into S. (At the end of S)



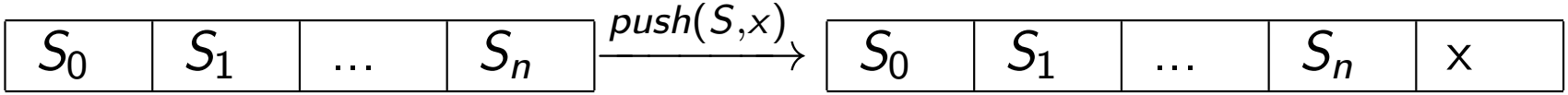
- $pop(S)$: DELETE an element from S. (LIFO)



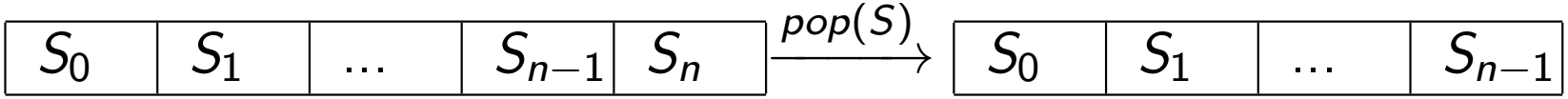
Stack Operations

We define a stack S using an array $S[0, 1, ..., n]$. Then we are going to define some basic operations on S.

- $push(S, x)$: INSERT x into S. (At the end of S)



- $pop(S)$: DELETE an element from S. (LIFO)



- $isEmpty(S)$: Return *True* if the stack is empty, otherwise return *False*.

Stacks in Python

In Python, we can construct a stack class in Python using the following code:

Python: Stack class

```
class Stack:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        return self.items.pop()
```

Queues

Queue

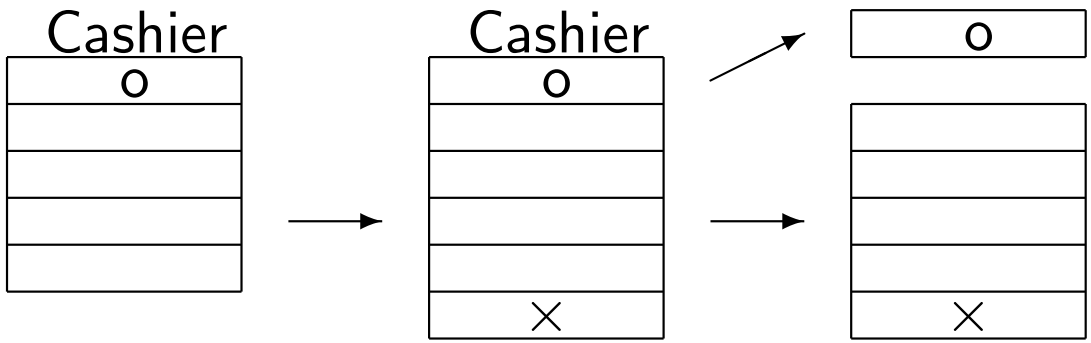
Queues are dynamic sets in which the element removed from the set by the DELETE operation is specified. They obey First-In-First-Out (FIFO) rule.

Queues

Queue

Queues are dynamic sets in which the element removed from the set by the DELETE operation is specified. They obey First-In-First-Out (FIFO) rule.

Imagine you have a queue of consumers waiting to pay a cashier. First come first served, the consumer has come first can leave earlier than other consumers. Now obeying FIFO rule, the waiting queue can be viewed as a queue.



Even if a new consumer comes, the first consumer still leaves the first.

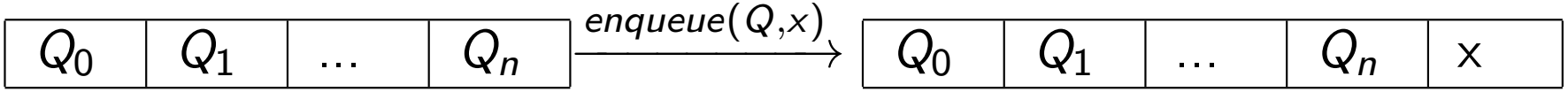
Queue Operations

We define a queue Q using an array $Q[0, 1, ..., n]$. Then we are going to define some basic operations on Q.

Queue Operations

We define a queue Q using an array $Q[0, 1, ..., n]$. Then we are going to define some basic operations on Q.

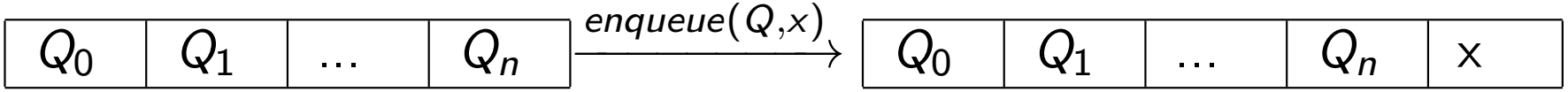
- *enqueue*(Q, x): INSERT x into Q. (At the end of Q)



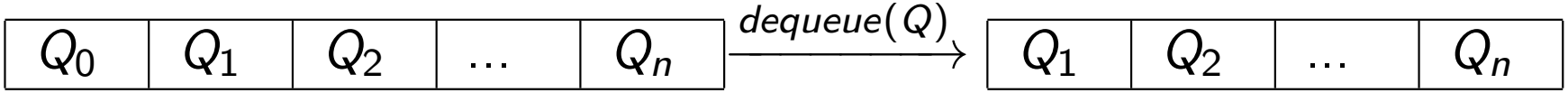
Queue Operations

We define a queue Q using an array $Q[0, 1, ..., n]$. Then we are going to define some basic operations on Q.

- *enqueue*(Q, x): INSERT x into Q. (At the end of Q)



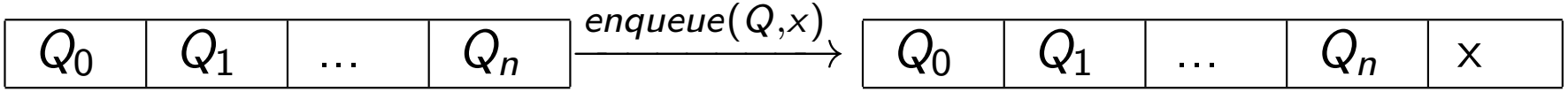
- *dequeue*(Q): DELETE an element from Q. (FIFO)



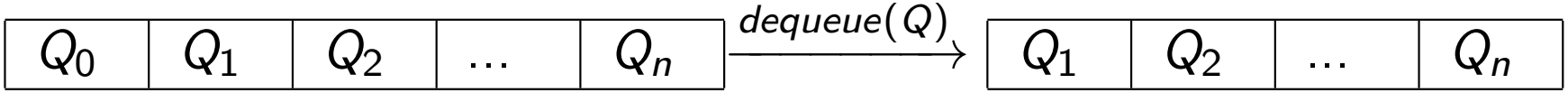
Queue Operations

We define a queue Q using an array $Q[0, 1, ..., n]$. Then we are going to define some basic operations on Q.

- *enqueue*(Q, x): INSERT x into Q. (At the end of Q)



- *dequeue*(Q): DELETE an element from Q. (FIFO)

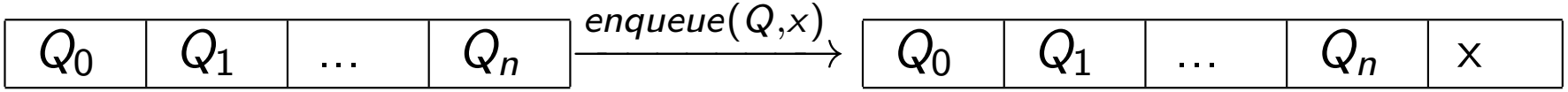


- *isEmpty*(Q): Return *True* if the queue is empty, otherwise return *False*.

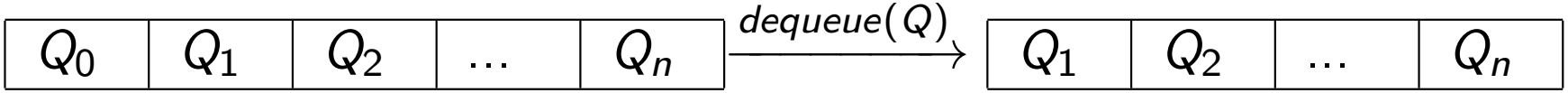
Queue Operations

We define a queue Q using an array $Q[0, 1, ..., n]$. Then we are going to define some basic operations on Q.

- *enqueue*(Q, x): INSERT x into Q. (At the end of Q)



- *dequeue*(Q): DELETE an element from Q. (FIFO)



- *isEmpty*(Q): Return *True* if the queue is empty, otherwise return *False*.

How to construct a queue in Python is left as an exercise. (Pretty similar to the stack class.)

Pointers

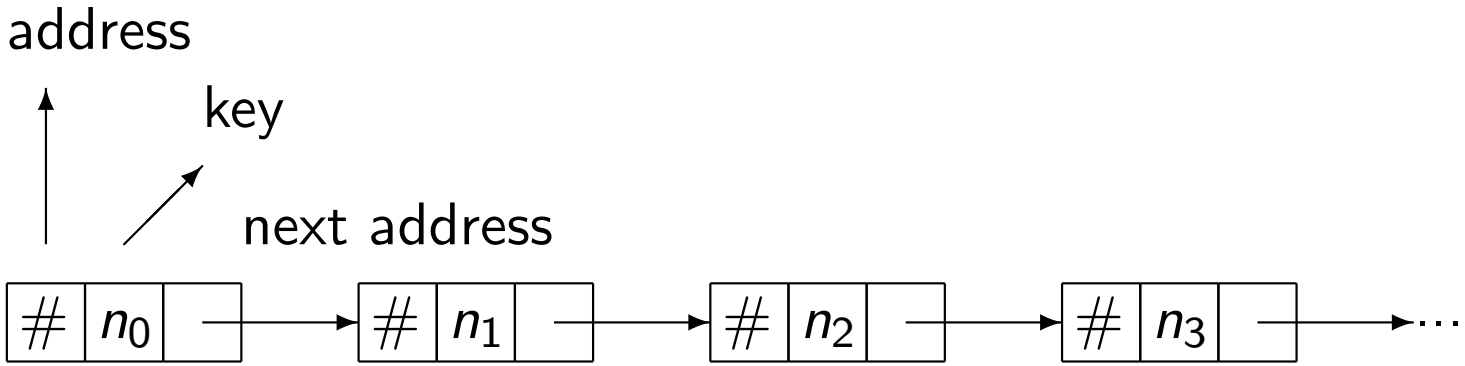
Pointer

A pointer is a programming language object, whose value refers directly to (or "points to") another value stored elsewhere in the computer memory using its address.

Pointers

Pointer

A pointer is a programming language object, whose value refers directly to (or "points to") another value stored elsewhere in the computer memory using its address.



A set of pointers is like a hint-hunt game, where in every step you can find the hint to the next treasure, and next treasure at hinted location (address) contains gold (key) and the next hint (next address) to another treasure. In this chain every treasure could be found.

A Review of Lists

You may have used lists in Python for a lot of times, usually as a way to store cells of data, as using lists as arrays. Before talking about lists formally, a question is placed: what is the difference between list type and array type?

A Review of Lists

You may have used lists in Python for a lot of times, usually as a way to store cells of data, as using lists as arrays. Before talking about lists formally, a question is placed: what is the difference between list type and array type?

- In some languages, the number of elements in an array must be declared prior to using it. (And unable to expand.) In lists you can expand as long as memory permits!

A Review of Lists

You may have used lists in Python for a lot of times, usually as a way to store cells of data, as using lists as arrays. Before talking about lists formally, a question is placed: what is the difference between list type and array type?

- In some languages, the number of elements in an array must be declared prior to using it. (And unable to expand.) In lists you can expand as long as memory permits!
- The idea of arrays and that of lists are different. In an array all data are stored in cells which are continuous neighbours, but in a list all data are only linked and are not necessarily next to each other (Recall the pointers!).

Lists

Linked Lists

A linked list is an abstract data type that represents a sequence of values.
Lists are like sequences in Mathematics.
In the context of this talk we define linked lists to be the linked chains of pointers.

Lists

Linked Lists

A linked list is an abstract data type that represents a sequence of values. Lists are like sequences in Mathematics.
In the context of this talk we define linked lists to be the linked chains of pointers.

Singly Linked Lists

Singly linked lists contain nodes which have a **key** field as well as a **next** field. i.e. it is one-way route and you cannot go back.

Lists

Linked Lists

A linked list is an abstract data type that represents a sequence of values. Lists are like sequences in Mathematics.
In the context of this talk we define linked lists to be the linked chains of pointers.

Singly Linked Lists

Singly linked lists contain nodes which have a **key** field as well as a **next** field. i.e. it is one-way route and you cannot go back.

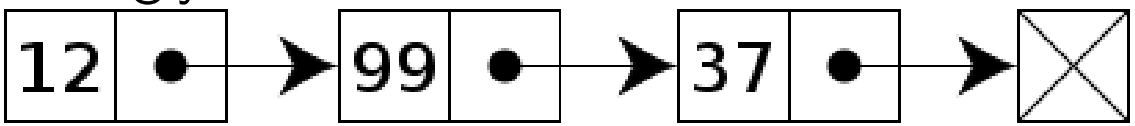
Doubly Linked Lists

In a doubly linked list, each node contains, besides the next-node link, a second link field pointing to the previous node in the sequence, called **prev** (Previous).

Graphic Illustration

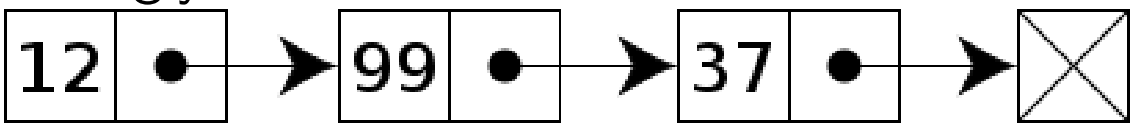
Graphic Illustration

A singly linked list:

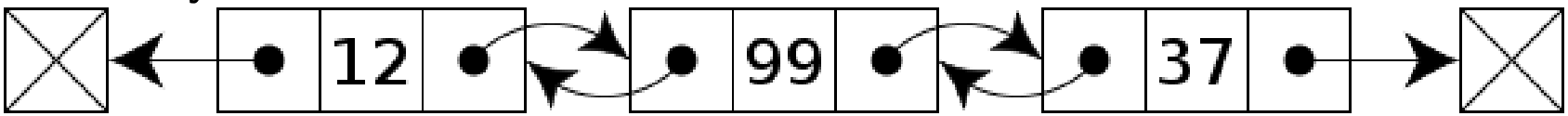


Graphic Illustration

A singly linked list:

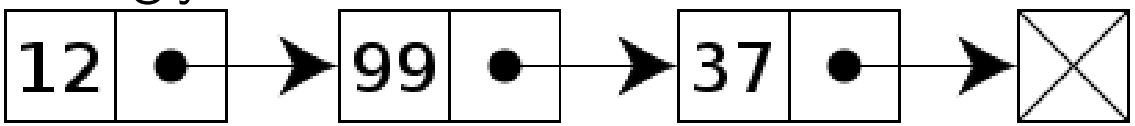


A doubly linked list:

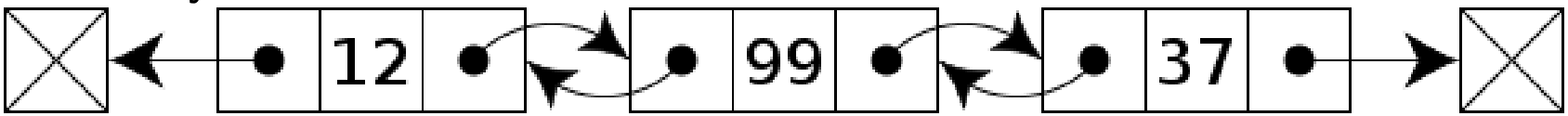


Graphic Illustration

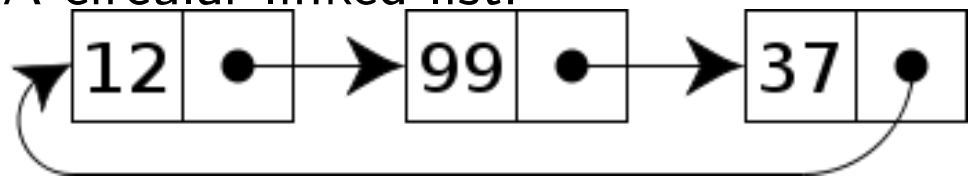
A singly linked list:



A doubly linked list:



A circular linked list:



The End