CSCA48 Winter 2016

Week 2 – Queues, Stacks and Number Conversion

Bo(Kenny) Zhao

University of Toronto Scarborough

January 13, 2016

UNIVERSITY OF TORONTO SCARBOROUGH

Who am I?

- Bo(Kenny) Zhao
- Undergraduate
  - Computer Science & Mathematics
- bo.zhao@mail.utoronto.ca
- or PM through Piazza
- Tutorial: TUT0013 Wednesday 10:00 – 11:00 MW160
- Practical: PRA004 Wednesday 16:00 – 17:00 BV469

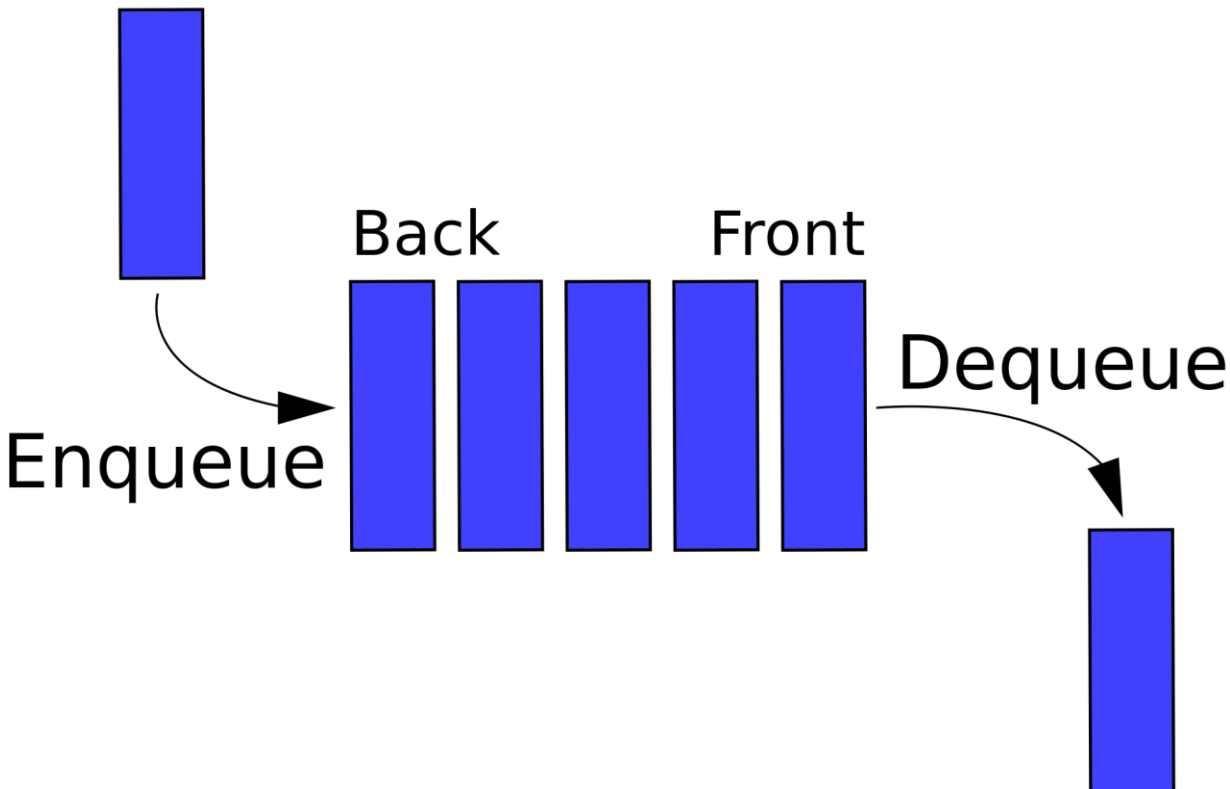
Office Hour

LEARNING OBJECTIVES

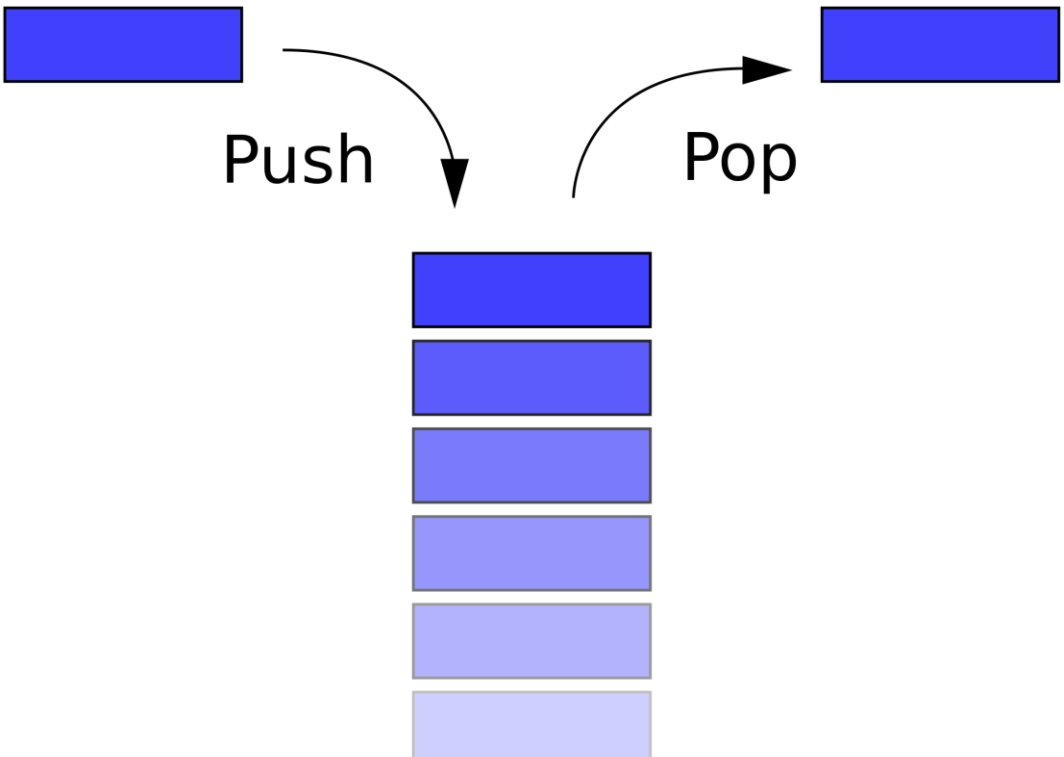
- At the end of the tutorial, you will be able to …

1. Create your own stacks using different implementations
    - Implementations of stacks
    - Efficiency of different implementations
2. Convert decimal numbers into binary
    - Algorithm
    - Implementation of the algorithm using stacks

QUEUE



STACK



STACK (first implementation)

```
1  class Stack:
2      '''A last-in, first-out (LIFO) stack of items'''
3
4      def __init__(self):
5          '''(Stack) -> NoneType
6          Create a new, empty stack.
7          '''
8
9
10     def push(self, new_obj):
11         '''(Stack, object) -> NoneType
12         Place new_obj on top of this stack.
13         '''
14
15
16     def pop(self):
17         '''(Stack) -> object
18         Remove and return the top item in this stack.
19         '''
20
21
22     def is_empty(self):
23         '''(Stack) -> bool
24         Return True iff this stack is empty
25         '''
26
27
```

Mangled code …

A. self._contents.append(new_obj)

B. self._contents = []

C. return self._contents.pop()

D. return self._contents == []

STACK (first implementation)

```
1  class Stack:
2      '''A last-in, first-out (LIFO) stack of items'''
3
4      def __init__(self):
5          '''(Stack) -> NoneType
6          Create a new, empty stack.
7          '''
8                B
9
10     def push(self, new_obj):
11         '''(Stack, object) -> NoneType
12         Place new_obj on top of this stack.
13         '''
14               A
15
16     def pop(self):
17         '''(Stack) -> object
18         Remove and return the top item in this stack.
19         '''
20               C
21
22     def is_empty(self):
23         '''(Stack) -> bool
24         Return True iff this stack is empty
25         '''
26               D
27
```

Mangled code …

A. self._contents.append(new_obj)

B. self._contents = []

C. return self._contents.pop()

D. return self._contents == []

STACK (second implementation)

```
1  class Stack:
2      '''A last-in, first-out (LIFO) stack of items'''
3
4      def __init__(self):
5          '''(Stack) -> NoneType
6          Create a new, empty stack.
7          '''
8
9
10     def push(self, new_obj):
11         '''(Stack, object) -> NoneType
12         Place new_obj on top of this stack.
13         '''
14
15
16     def pop(self):
17         '''(Stack) -> object
18         Remove and return the top item in this stack.
19         '''
20
21
22     def is_empty(self):
23         '''(Stack) -> bool
24         Return True iff this stack is empty
25         '''
26
27
```

Mangled code …

A. return self._contents.pop(0)

B. self._contents.insert(0, new_obj)

C. return self._contents == []

D. self._contents = []

STACK (second implementation)

```
1  class Stack:
2      '''A last-in, first-out (LIFO) stack of items'''
3
4      def __init__(self):
5          '''(Stack) -> NoneType
6          Create a new, empty stack.
7          '''
8                D
9
10     def push(self, new_obj):
11         '''(Stack, object) -> NoneType
12         Place new_obj on top of this stack.
13         '''
14               B
15
16     def pop(self):
17         '''(Stack) -> object
18         Remove and return the top item in this stack.
19         '''
20               A
21
22     def is_empty(self):
23         '''(Stack) -> bool
24         Return True iff this stack is empty
25         '''
26               C
27
```

Mangled code …

A. return self._contents.pop(0)

B. self._contents.insert(0, new_obj)

C. return self._contents == []

D. self._contents = []

STACK (third implementation)

```
1  class Stack:
2      '''A last-in, first-out (LIFO) stack of items'''
3
4      def __init__(self):
5          '''(Stack) -> NoneType
6          Create a new, empty stack.
7          '''
8
9
10     def push(self, new_obj):
11         '''(Stack, object) -> NoneType
12         Place new_obj on top of this stack.
13         '''
14
15
16     def pop(self):
17         '''(Stack) -> object
18         Remove and return the top item in this stack.
19         '''
20
21
22     def is_empty(self):
23         '''(Stack) -> bool
24         Return True iff this stack is empty
25         '''
26
27
```

Mangled code …

A. self._contents[self._height] = new_obj
self._height +=1

B. self._height -=1
return self._contents[self._height]

C. return self._height == 0

D. self._contents = {}
self._height = 0

STACK (third implementation)

```
1  class Stack:
2      '''A last-in, first-out (LIFO) stack of items'''
3
4      def __init__(self):
5          '''(Stack) -> NoneType
6          Create a new, empty stack.
7          '''
8                D
9
10     def push(self, new_obj):
11         '''(Stack, object) -> NoneType
12         Place new_obj on top of this stack.
13         '''
14               A
15
16     def pop(self):
17         '''(Stack) -> object
18         Remove and return the top item in this stack.
19         '''
20               B
21
22     def is_empty(self):
23         '''(Stack) -> bool
24         Return True iff this stack is empty
25         '''
26               C
27
```

Mangled code …

A. self._contents[self._height] = new_obj
self._height +=1

B. self._height -=1
return self._contents[self._height]

C. return self._height == 0

D. self._contents = {}
self._height = 0

STACKS -> BINARY CONVERSION

BINARY CONVERSION (algorithm)

Example: convert 37 into binary

decimal number	integer division by 2	remainder
37	18	1
18	9	0
9	4	1
4	2	0
2	1	0
1	0	1
0		

$$(37)_{10} = (100101)_{2}$$

BINARY CONVERSION (implementation)

```
1  from my_stacks import *
2
3  def convert_to_binary(decimal_number):
4      '''(int) -> str
5      Return a string representing of demical_number in binary.
6      REQ: decimal_number >= 0
7      '''
```

A. remainder_stack = Stack()

B. remainder_stack.push(remainder)

C. binary_string = ""

D. remainder = decimal_number % 2

E. decimal_number = decimal_number // 2

F. return binary_string

G. while not (remainder_stack.is_empty()):

H. binary_string += str(remainder_stack.pop())

I. while(decimal_number > 0):

BINARY CONVERSION (implementation & encapsulation)

- **To Wing**

SUMMARY

- Various implementations of a particular ADT
  - Find a new way to implement a stack
  - Hints: string, set of tuple, etc.
- Algorithm vs Code
  - Algorithm first, code second
- Encapsulation
  - Hide internal implementation