COSC 2306
Data Programming

Stacks

ADT vs Data Structure

- ADT (Abstract data type)
  - A conceptual model that defines the operations and behaviors for a data structure
  - Separate "what" operations a data structure can perform from "how" those operations are implemented
  - Programming language-independent
- Data structures:
  - Programming implementations of ADTs
- Examples:
  - Stacks
  - Queues
  - Linked Lists

ADT and Data Structure

- Why ADT/data structures are needed?
  - Readability: easier to understand codes
  - Clearness: guidance for algorithm designs
  - Efficiency: easy to implement and extend
  - Safety: control the access pattern of data

Stacks

- What is a stack?
- How to implement a stack?
- How to use the stack?

Stacks

- ADT
  - Linear data structure
  - Elements added, removed from one end only
  - Last In First Out (LIFO)

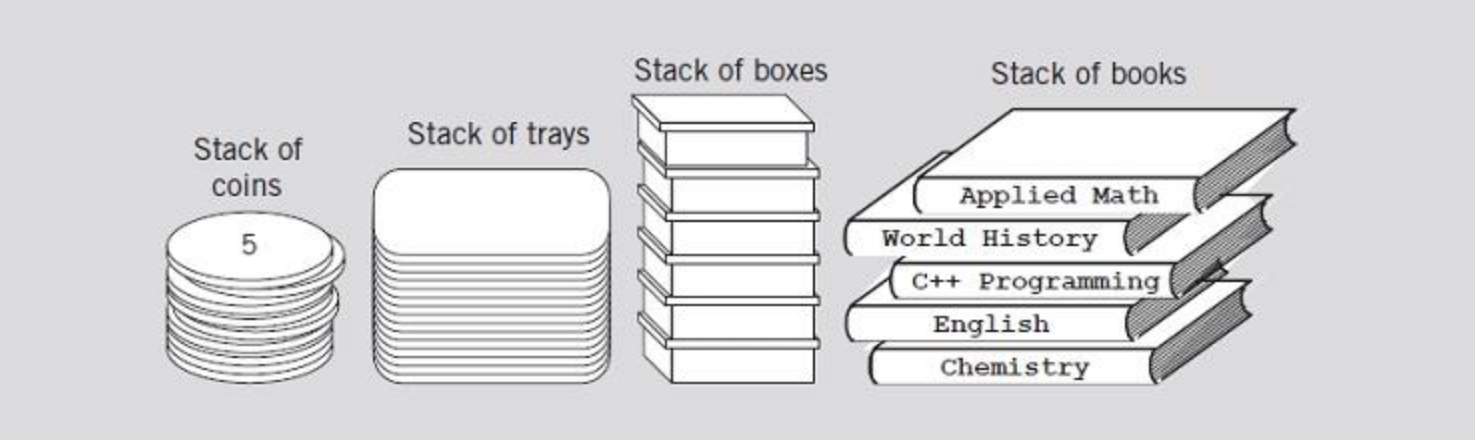
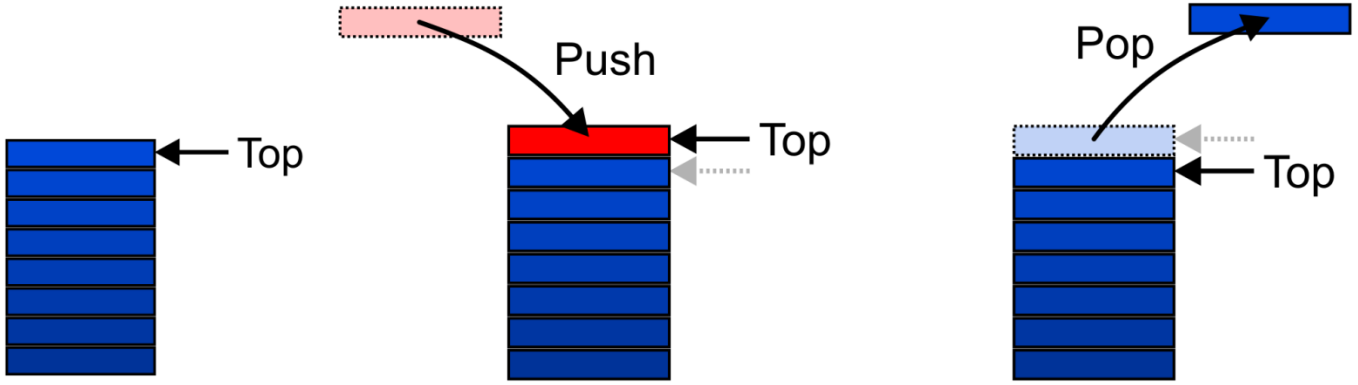


FIGURE 7-1 Various examples of stacks

Operations on stacks



Figures from Waterloo Engineering

Operations on stacks

- `push`
  - Add element onto the stack (if room available)
- `peek`
  - Retrieve top element of the stack
- `pop`
  - Remove top element from the stack
- `isEmpty`
  - Returns `true` if the stack is empty
- `size`
  - Returns the number of elements in the stack

Operations on stacks

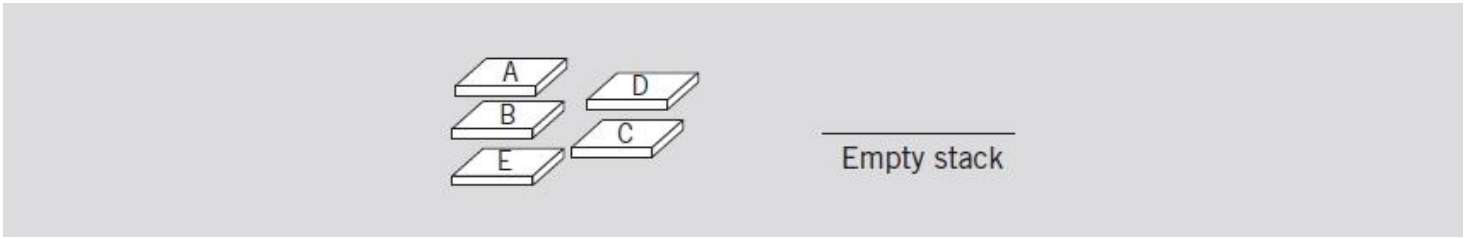


FIGURE 7-2 Empty stack

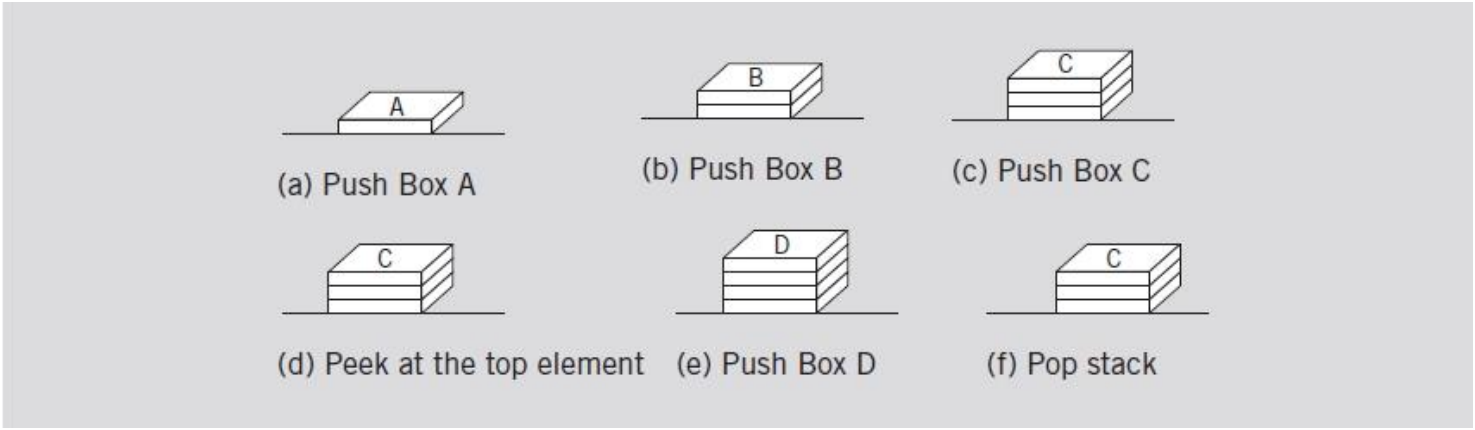


FIGURE 7-3 Stack operations

Simple list implementation

```
my_stack = []
my_stack.append(1)
my_stack.append(2)
last = my_stack.pop()
print(last)
print(my_stack)


>2
>[1]
```

Implementing a stack in Python

- Using a list storing stack elements
  - Index 0 (the first element): bottom
  - Index -1 (the last element): top

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

Implementing a stack in Python

- Using a list storing stack elements
  - Index 0 (the first element): bottom
  - Index -1 (the last element): top

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

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

Implementing a stack in Python

```
class Stack:
    ……
    def push(self, item):
        self.items.append(item)
        #Add an element to the end of the list
    def pop(self):
        return self.items.pop()
         #Get and remove the element at the end of the list
```

Implementing a stack in Python

```
class Stack:
    ……
    def peek(self):
        return self.items[-1]

    def size(self):
        return len(self.items)
```

Implementing a stack in Python

```
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()

    def peek(self):
        return self.items[-1]

    def size(self):
        return len(self.items)
```

Implementing a stack in Python

- Time cost for stack operations:
  - Push: O(1)
  - Pop: O(1)
  - Peek: O(1)
  - Get the stack bottom pop(n): O(n)

Example

- How is the stack after the operations?

```
st = Stack()
st.push(1)
st.push(2)
st.push(3)
```

[1, 2, 3]

Example

- How is the stack after the operations?

```
st = Stack()
st.push("alpha")
st.push("beta")
st.peek()
st.pop()
st.push("gamma")
st.peek()
```

['alpha', 'gamma']

Example

- How is the stack after the operations?

```
st = Stack()
st.push(4)
st.peek()
st.pop()
st.pop()
st.push(3)
st.push(2)
```

```
Traceback (most recent call last):
  File "main.py", line 27, in <module>
    st.pop()
  File "main.py", line 12, in pop
    return self.items.pop()
IndexError: pop from empty list
```

Implementing a stack in Python

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

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

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

    def pop(self):
        if self.isEmpty():
            return None
        return self.items.pop()

    def peek(self):
        if self.isEmpty():
            return None
        return self.items[-1]

    def size(self):
        return len(self.items)
```

Example

- What is the output?

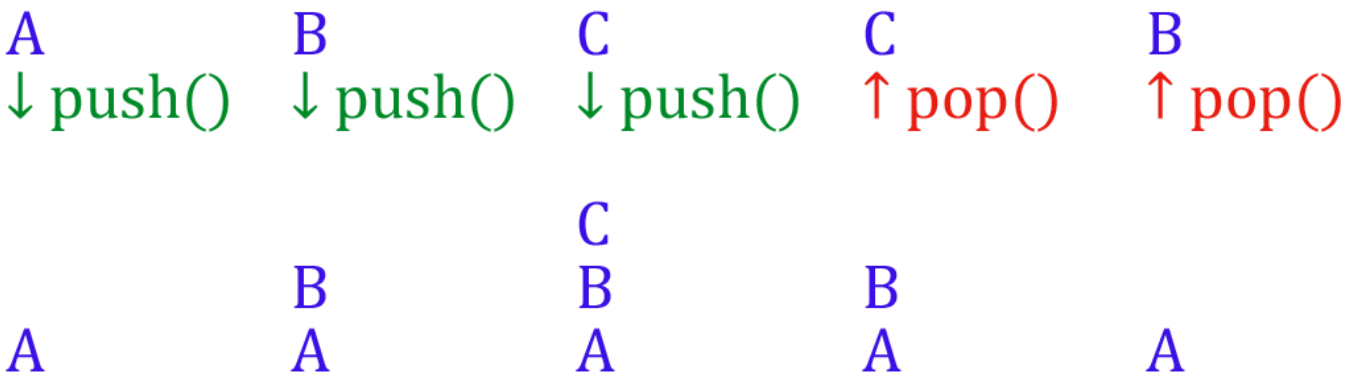
```
st = Stack()
n = 10
for i in range(n):
    st.push(i)

while not st.isEmpty():
    print(st.pop())
```

```
9
8
7
6
5
4
3
2
1
0
```

Practice: reverse string

- Write a program that reverses the input string provided by a user



push() and pop() operations on a stack

Practice : reverse string

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

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

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

    def pop(self):
        if self.isEmpty():
            return None
        return self.items.pop()

    def peek(self):
        if self.isEmpty():
            return None
        return self.items[-1]

    def size(self):
        return len(self.items)
```