香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Introduction to Computer Engineering: Programming Applications

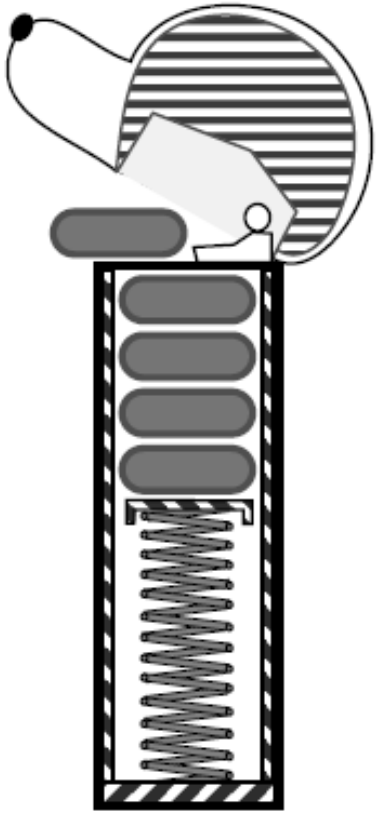
Lecture 9
Data Structure and Algorithms Part II
Stack and Queue

Prof. Junhua Zhao
School of Science and Engineering

Stack

- A stack is a collection of objects that are inserted and removed according to the last-in, first-out (LIFO) principle

- A user may insert objects into a stack at any time, but may only access or remove the most recently inserted object that remains (at the so-called “top” of the stack)



Example: Web Browser

- Internet Web browsers store the addresses of recently visited sites in a stack. Each time a user visits a new site, that site's address is "pushed" onto the stack of addresses. The browser then allows the user to "pop" back to previously visited sites using the "back" button.

Elon Musk liked

DogeDesigner @cb_doge · 16h

In 1995, Elon Musk applied for a job at Netscape, sent his resume, but he was too shy to talk to anyone. So he decided to start his own company (Zip2).

Imagine if @elonmusk had got his dream job at Netscape. 👀

Elon Musk and 2 others

401 539 6,103 1M

Example: Text editor

- Text editors usually provide an “undo” mechanism that cancels recent editing operations and reverts to former states of a document. This undo operation can be accomplished by keeping text changes in a stack.

The stack class

- Generally, a stack may contain the following methods:

S.push(e):	Add element e to the top of stack S.
S.pop( ):	Remove and return the top element from the stack S; an error occurs if the stack is empty.
S.top( ):	Return a reference to the top element of stack S, without removing it; an error occurs if the stack is empty.
S.is_empty( ):	Return True if stack S does not contain any elements.
len(S):	Return the number of elements in stack S; in Python, we implement this with the special method __len__.

The Code of Stack Class

```
class ListStack:

    def __init__(self):
        self.__data = list()

    def __len__(self):
        return len(self.__data)

    def is_empty(self):
        return len(self.__data) == 0

    def push(self,e):
        self.__data.append(e)

    def top(self):
        if self.is_empty():
            print('The stack is empty.')
        else:
            return self.__data[self.__len__()-1]

    def pop(self):
        if self.is_empty():
            print('The stack is empty.')
        else:
            return self.__data.pop()
```

The code to use stack class

```
def main():
    s = ListStack()
    print('The stack is empty? ',s.is_empty())
    s.push(100)
    s.push(200)
    s.push(300)
    print(s.top())
    print(s.pop())
    print(s.top())
```

Practice: Reverse a list using stack

- Write a program to reverse the order of a list of numbers using the stack class

Solution:

```
from stack import ListStack

def reverse_data(oldList):
    s = ListStack()
    newList = list()

    for i in oldList:
        s.push(i)

    while (not s.is_empty()):
        mid = s.pop()
        newList.append(mid)

    return newList

def main():
    oldList = [1,2,3,4,5]
    newList = reverse_data(oldList)
    print(newList)
```

Practice: Brackets match checking

- In correct arithmetic expressions, the opening brackets must match the corresponding closing brackets. Write a program to check whether all the opening brackets have matched closing brackets.

Solution:

```
from stack import ListStack

def is_matched(expr):
    lefty = '([{'
    righty = ')]}'

    s = ListStack()

    for c in expr:

        if c in lefty:
            s.push(c)
        elif c in righty:
            if s.is_empty():
                return False
            if righty.index(c)!=lefty.index(s.pop()):
                return False
    return s.is_empty()

def main():
    expr = '1+2*(3+4)-[5-6]'
    print(is_matched(expr))
    expr = '(((()}]'
    print(is_matched(expr))
```

Practice: Matching Tags in HTML Language

- HTML is the standard format for hyperlinked documents on the Internet

- In an HTML document, portions of text are delimited by HTML tags. A simple opening HTML tag has the form “<name>” and the corresponding closing tag has the form “</name>”

HTML Tags

- Commonly used HTML tags that are used in this example include

  - body: document body
  - h1: section header
  - center: center justify
  - p: paragraph
  - ol: numbered (ordered) list
  - li: list item

An example of HTML document

```
<body>
<center>
<h1> The Little Boat </h1>
</center>
<p> The storm tossed the little
boat like a cheap sneaker in an
old washing machine.  The three
drunken fishermen were used to
such treatment, of course, but
not the tree salesman, who even as
a stowaway now felt that he
had overpaid for the voyage. </p>
<ol>
<li> Will the salesman die? </li>
<li> What color is the boat? </li>
<li> And what about Naomi? </li>
</ol>
</body>
```

(a)

The Little Boat

The storm tossed the little boat like a cheap sneaker in an old washing machine. The three drunken fishermen were used to such treatment, of course, but not the tree salesman, who even as a stowaway now felt that he had overpaid for the voyage.

1. Will the salesman die?
2. What color is the boat?
3. And what about Naomi?

(b)

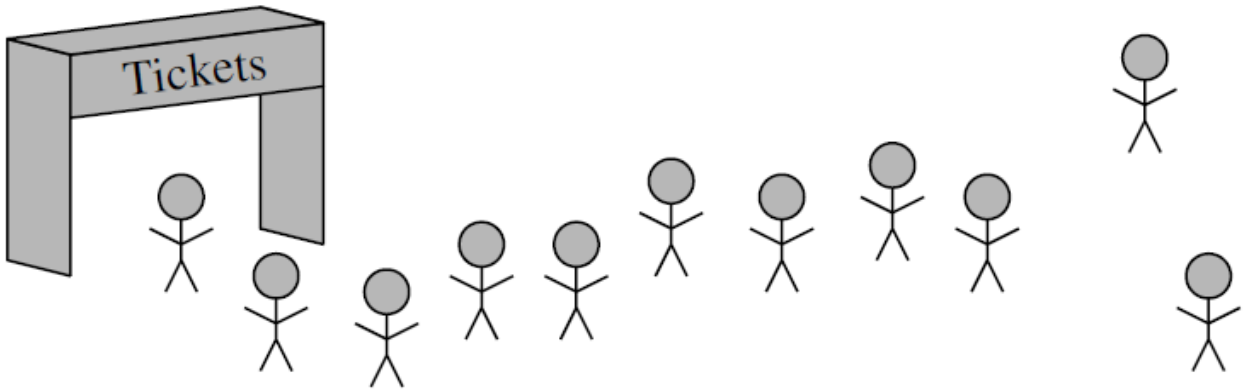
Solution:

```
from stack import ListStack

def is_matched_html(raw):
    s = ListStack()
    j = raw.find('<')

    while j!=-1:
        k = raw.find('>', j+1)
        if k==-1:
            return False
        tag = raw[j+1:k]

        if not tag.startswith('/'):
            s.push(tag)
        else:
            if s.is_empty():
                return False
            if tag[1:]!=s.pop():
                return False
        j = raw.find('<', k+1)

    return s.is_empty()

def main():
    fhand = open('sampleHTML.txt','r')
    raw = fhand.read()
    print(raw)
    print(is_matched_html(raw))
```

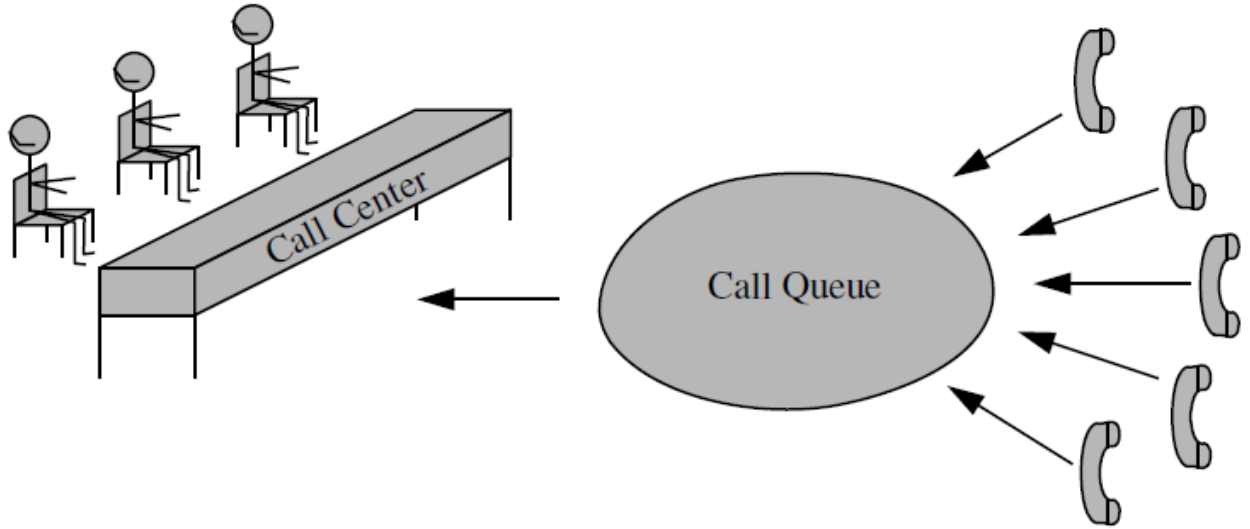
Queue

- Queue is another fundamental data structure
- A queue is a collection of objects that are inserted and removed according to the first-in, first-out (FIFO) principle
- Elements can be inserted at any time, but only the element that has been in the queue the longest can be next removed

Applications of Queue



(a)



(b)

The queue class

- The queue class may contain the following methods:

Q.enqueue(e):	Add element e to the back of queue Q.
Q.dequeue( ):	Remove and return the first element from queue Q; an error occurs if the queue is empty.
Q.first( ):	Return a reference to the element at the front of queue Q, without removing it; an error occurs if the queue is empty.
Q.is_empty( ):	Return True if queue Q does not contain any elements.
len(Q):	Return the number of elements in queue Q; in Python, we implement this with the special method __len__.

The code of queue class

```
class ListQueue:
    default_capacity = 5

    def __init__(self):
        self.__data = [None]*ListQueue.default_capacity
        self.__size = 0
        self.__front = 0
        self.__end = 0

    def __len__(self):
        return self.__size

    def is_empty(self):
        return self.__size ==0

    def first(self):
        if self.is_empty():
            print('Queue is empty.')
        else:
            return self.__data[self.__front]
```

```
def dequeue(self):

    if self.is_empty():
        print('Queue is empty.')
        return None

    answer = self.__data[self.__front]
    self.__data[self.__front] = None
    self.__front = (self.__front+1) \
                   % ListQueue.default_capacity
    self.__size -=1
    return answer

def enqueue(self,e):
    if self.__size == ListQueue.default_capacity:
        print('The queue is full.')
        return None

    self.__data[self.__end] = e
    self.__end = (self.__end+1) \
                 % ListQueue.default_capacity
    self.__size += 1

def outputQ(self):
    print(self.__data)
```

Practice: Chemistry Reaction Simulation

- Write a program simulates a chemical reactor where molecules are sequentially added to a reaction chamber and processed in a First-In, First-Out (FIFO) manner. This models how reactants enter a reactor and undergo reactions one by one, reflecting the queue's behavior.

Chemistry Reaction Simulation

```
# Chemistry function to simulate a reaction
def simulate_reaction(molecule):
    """Simulate a chemical reaction based on molecule type."""
    reaction_products = {
        "H2": "H2O (with O2)",           # Hydrogen reacts with oxygen to form water
        "O2": "O3 (with electricity)",   # Oxygen can form ozone under specific conditions
        "N2": "NH3 (with H2)",           # Nitrogen and hydrogen form ammonia
        "CO2": "C6H12O6 (with H2O)"      # Photosynthesis-like reaction (simplified)
    }
    return reaction_products.get(molecule, "No reaction")

# Example of using the queue for chemical reactions
def chemistry_queue_example():
    # Create a queue to simulate molecules entering the reaction chamber
    molecule_queue = Queue()

    # List of molecules entering the chamber in sequence
    molecules = ["H2", "O2", "N2", "CO2", "H2", "O2"]

    # Enqueue each molecule into the queue
    for molecule in molecules:
        molecule_queue.enqueue(molecule)
        print(f"Molecule {molecule} added to the queue.")

    print("\nProcessing molecules in the reaction chamber (FIFO order):")

    # Process each molecule in FIFO order
    while not molecule_queue.is_empty():
        molecule = molecule_queue.dequeue()
        reaction_result = simulate_reaction(molecule)
        print(f"Molecule {molecule} reacted to form: {reaction_result}")

# Run the chemistry queue example
chemistry_queue_example()
```

Bubble sort

- Bubble sort is a simple sorting algorithm

- Its general procedure is:

1) Iterate over a list of numbers, compare every element i with the following element i+1, and swap them if i is larger
2) Iterate over the list again and repeat the procedure in step 1, but ignore the last element in the list
3) Continuously iterate over the list, but each time ignore one more element at the tail of the list, until there is only one element left

Practice: Bubble sort over a standard list

```
def bubble(bubbleList):
    listLength = len(bubbleList)
    while listLength > 0:
        for i in range(listLength - 1):
            if bubbleList[i] > bubbleList[i+1]:
                buf = bubbleList[i]
                bubbleList[i] = bubbleList[i+1]
                bubbleList[i+1] = buf
        listLength -= 1
    return bubbleList


def main():
    bubbleList = [3, 4, 1, 2, 5, 8, 0, 100, 17]
    print(bubble(bubbleList))
```

Quick sort

- Quick sort is a widely used algorithm, which is more efficient than bubble sort

- The main procedure of quick sort algorithm is:

1) Pick an element, called a pivot, from the array
2) Partitioning: reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation
3) Recursively apply the above steps to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values

Practice: Quick sort over a standard list

```
def quickSort(L, low, high):
    i = low
    j = high
    if i>=j:
        return L
    pivot = L[i]

    while i<j:
        while i<j and L[j]>=pivot:
            j = j - 1
        L[i] = L[j]

        while i<j and L[i]<=pivot:
            i = i + 1
        L[j] = L[i]
    L[i] = pivot

    quickSort(L, low, i-1)
    quickSort(L, j+1, high)
    return L

L = [6, 5, 3, 10, 12, 2, 4]
print(quickSort(L, 0, 6))
```