Stacks and Queues

Instructor: Dr. Sunho Lim (Ph.D., Assistant Professor)

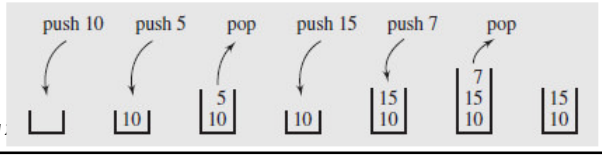
Lecture 05

sunho.lim@ttu.edu

Adapted partially from Data Structures and Algorithms in C++, Adam Drozdek, 4th Edition, Cengage Learning; and Algorithms and Data Structures, Douglas Wilhelm Harder, Mmath

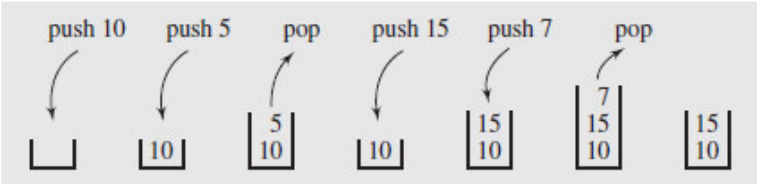
Stacks

- A ***stack*** ??
  - a restricted access **linear data structure**
  - only be accessed at one of its ends for adding and removing data elements
  - e.g., a stack of trays in a cafeteria; trays are removed from the top and placed back on the top
  - ***last-in first-out (LIFO)*** structure
- Restrictions
  - only remove items that are available
  - can't add more items if there is no room



Stacks (cont.)

- Stack operations:
  - ***clear( )***: clears the stack
  - ***isEmpty( )***: determines if the stack is empty
  - ***push(el)***: pushes the data item ***el*** onto the top of the stack
  - ***pop( ):*** removes the top element from the stack
  - ***topEl( ):*** returns the value of the top element of the stack without removing it
- E.g., a series of pushes and pops



Stacks (cont.)

- Particularly useful in situations
  - data have to be stored and processed in **reverse order**
- Numerous applications:
  - evaluating expressions and parsing syntax
  - balancing **delimiters** in program code, e.g., [, {, (
  - converting numbers between bases
  - processing financial data
  - backtracking algorithms

Stacks (cont.)

- Balancing delimiters in program code, e.g., [, {, ((cont.)
  - e.g., `while (m < (n[8] + o))`
  - open delimiters, e.g., '(', '[', '{',
  - close delimiters, e.g., ')', ']', '}'
  - first opening parenthesis must be matched with the last closing parenthesis
- A delimiter matching algorithm:
  - a pseudo code
  - e.g., s = t[5] + u / (v * (w+y));

```
while not end of file {
  if ch is `(`, `[`, or `{`
    push (ch);
  else if ch is `)`, `]`, or `}`
    if ch and popped off delimiter
do not match
      failure;
  read next character ch;
}

if stack is empty
  success;
else
  failure;
```

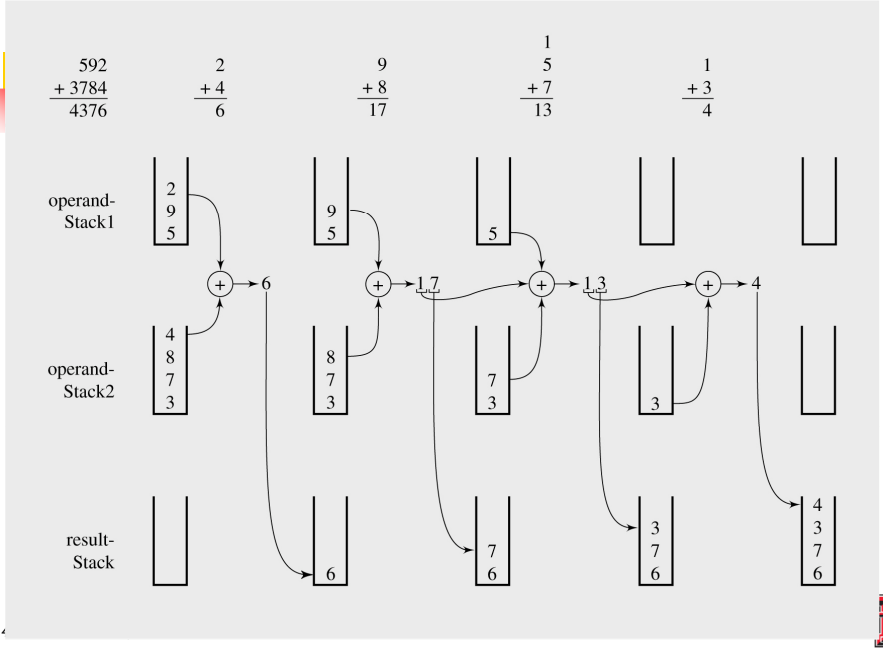

Stacks (co

- s = t[5] + u / (v * (w + y));
  - checking delimiters
  - open delimiters, e.g., '(', '[', '{',
  - close delimiters, e.g., ')', ']', '}'

Stack	Nonblank Character Read	Input Left
empty		s = t[5] + u / (v * (w + y));
empty	s	= t[5] + u / (v * (w + y));
empty	=	t[5] + u / (v * (w + y));
empty	t	[5] + u / (v * (w + y));
[	[	5] + u / (v * (w + y));
[	5	] + u / (v * (w + y));
empty	]	+ u / (v * (w + y));
empty	+	u / (v * (w + y));
empty	u	/ (v * (w + y));
empty	/	(v * (w + y));
(	(	v * (w + y));
(	v	* (w + y));
(	*	(w + y));
(		
(	(	w + y));
(		
(	w	+y));
(		
(	+	y));
(		
(	y	));
(	)	);
empty	)	;
empty	;	

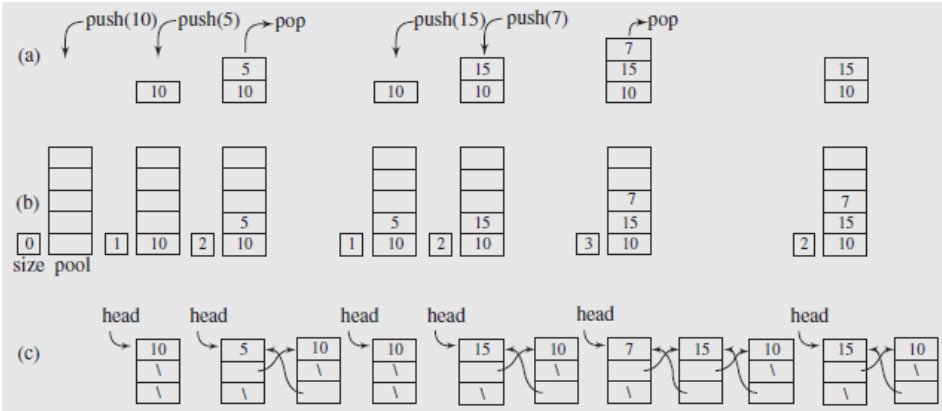
Stacks (cont.)

- Adding numbers 592 + 3784



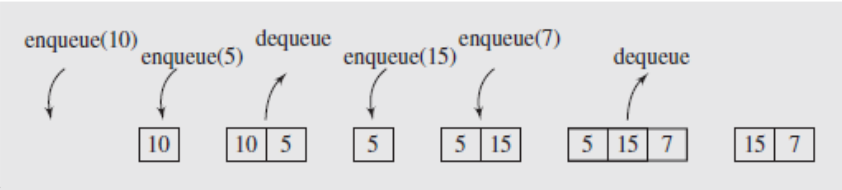
Stacks (cont.)

- Implementing stack using,
  - vector (or array) – **stack pointer**
  - double linked list with single "head" pointer
    - inserting at the beginning of list
    - deleting at the beginning of list



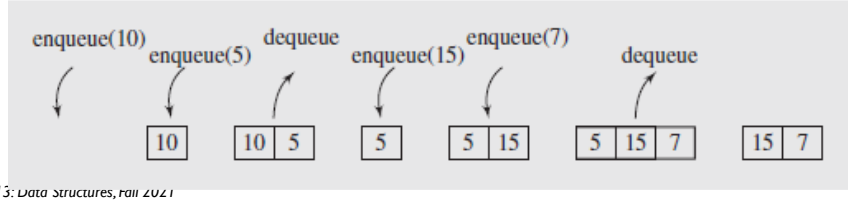
Queues

- A ***queue*** ??
  - a restricted access linear data structure
  - Use both ends with additions restricted to one end (the ***rear***) and deletions to the other (the ***front***)
  - ***first-in first-out (FIFO)*** structures



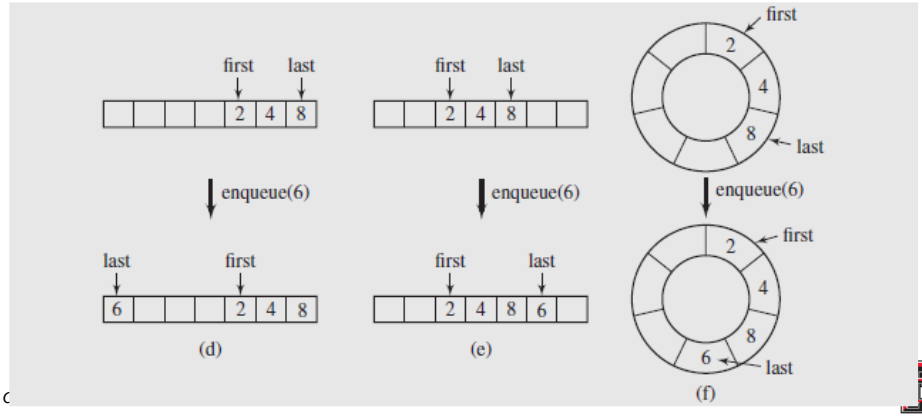
Queues (cont.)

- Queue operations:
  - ***clear( )***: clears the queue
  - ***isEmpty( )***: determines if the queue is empty
  - ***enqueue(el)***: adds the data item ***el*** to the end of the queue
  - ***dequeue( )***: removes the element from the front of the queue
  - ***firstEl( )***: returns the value of the first element of the queue without removing it
- E.g., a series of enqueues and dequeues



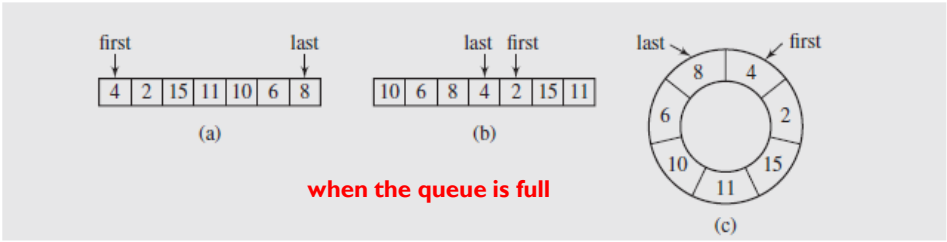
Queues (cont.)

- Implementing using an array
  - treats the array as though it were **circular**



Queues (cont.)

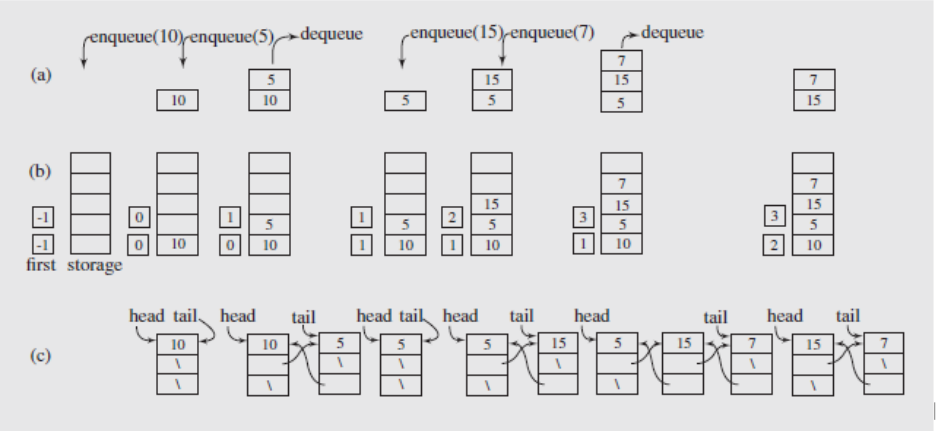
- Implementing using an array (cont.)



when the queue is full

Queues (cont.)

- Implementing queue using,
  - array (e.g., vector) – enqueue and dequeue **pointers**
  - double linked list with "head" and "tail" pointers
    - inserting at the end of list
    - deleting at the beginning of list



Queues (cont.)

- Used in a wide variety of applications,
  - especially in studies of service simulations
  - e.g., a very advanced body of mathematical theory, called ***queuing theory***

Priority Queues

- In some circumstances,
  - priorities associated the elements of the queue → affect the order of processing
- A ***priority queue*** ??
  - elements are removed based on priority and position
  - difficulty in implementing such a structure
    - trying to accommodate the priorities while still maintaining efficient enqueuing and dequeuing