NANYANG TECHNOLOGICAL UNIVERSITY

CE1107/CZ1107: DATA STRUCTURES AND ALGORITHMS

5A: **Stacks**

College of Engineering
School of Computer Engineering

TODAY

- Motivating application
- Stack data structure
- Stack implementation using linked lists
- Stack functions
  - push()
  - pop()
  - peek()
  - isEmptyStack()
- Working examples: Applications

LEARNING OBJECTIVES

After this lesson, you should be able to:

- Explain how a stack data structure operates
- Implement a stack using a linked list
- Choose a stack data structure when given an appropriate problem to solve

- You should also be able to
  - Implement a stack using an array (but we won't cover or test this)

MOTIVATING APPLICATION

- Scenario:
  - Container yard with 10x10 grid layout, and each square has a stack of containers
  - Overhead crane can move over any stack
  - Crane can hold/inspect one container at a time from top of stack and move it to any other stack
  - Grid square at (0,0) is empty
- Task: Simulate the operation of this container yard

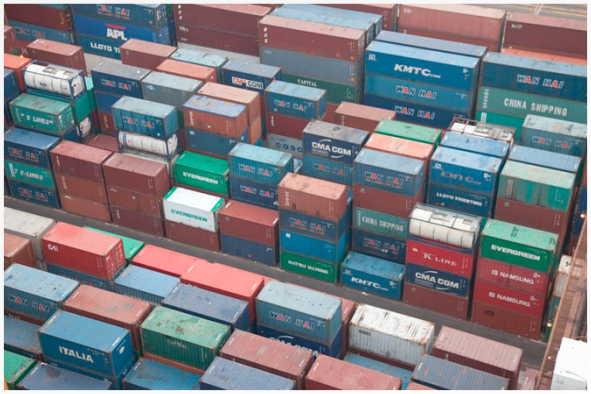


Image credits: http://kaidashton.blogspot.sg/2012/03/hong-kong-container-yards.html, http://es.made-in-china.com/co_goldhorse/product_RTG-Rubber-Tyre-Gantry-Crane-of-Port-for-Container-40FT-or-20FT-Payload-40-Ton_hhruniseg.html

TODAY

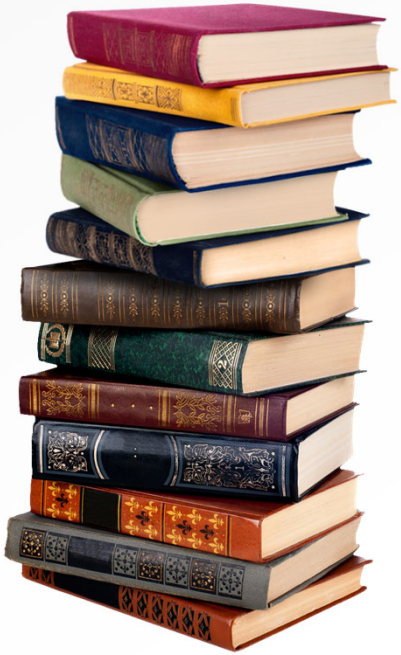
- Motivating application
- **Stack data structure**
- Stack implementation using linked lists
- Stack functions
  - push()
  - pop()
  - peek()
  - isEmptyStack()
- Working examples: Applications

PREVIOUSLY

- **Arrays**
  - Random access data structure
  - Access any element directly
    - array[index]
- **Linked lists**
  - Sequential access data structure
  - Have to go through a particular sequence when accessing elements
    - temp->next until you find the right node
- Today, consider one example of limited-access sequential data structures

STACK DATA STRUCTURE

- A **stack** is a data structure that operates like a physical stack of things
  - Stack of books, for example
  - Elements can only be added or removed at the top
- Key: Last-In, First-Out (LIFO) principle
  - Or, First-In, Last-Out (FILO)
- Often built on top of some other data structure
  - Arrays, Linked lists, etc.
  - We'll focus on a linked-list based implementation



STACK DATA STRUCTURE

- **Core operations**
  - Push: Add an item to the top of the stack
  - Pop: Remove an item from the top of the stack
- **Common helpful operations**
  - Peek: Inspect the item at the top of the stack without removing it
  - IsEmptyStack: Check if the stack has no more items remaining
- **Corresponding functions**
  - push()
  - pop()
  - peek()
  - isEmptyStack()
- We'll build a stack assuming that it only deals with integers
  - But as with linked lists, can deal with any contents depending on how you define the functions and the underlying implementation

TODAY

- Motivating application
- Stack data structure
- **Stack implementation using linked lists**
- Stack functions
  - push()
  - pop()
  - peek()
  - isEmptyStack()
- Working examples: Applications

STACK IMPLEMENTATION USING LINKED LISTS

- Recall that we defined a LinkedList structure
  - Encapsulates all required variables inside a single object
  - Conceptually neater to deal with
- Similarly, define a Stack structure.
  - We're going to build our stack on top of a linked list

```
typedef struct _stack{
     LinkedList ll;
} Stack;
```

STACK IMPLEMENTATION USING LINKED LISTS

- Stack structure

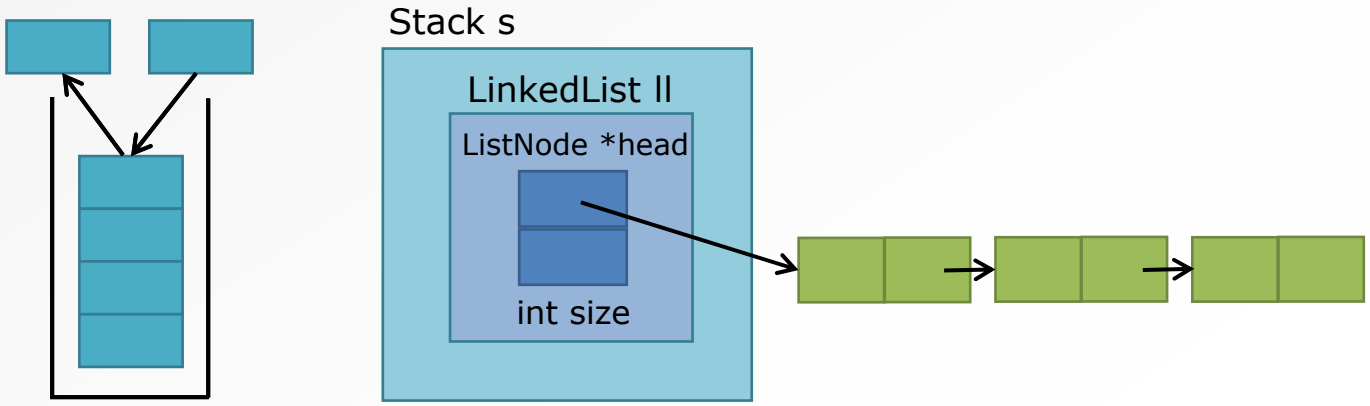
```
typedef struct _stack{
    LinkedList ll;
} Stack;
```

Notice this is a LinkedList, not a LinkedList *

actual linked list, not address of linked list

linked list can take care of size/nodes.

- Basically wrap up a linked list and use it for the actual data storage
- Just need to ensure we control where elements are added/removed
- Notice that the LinkedList already takes care of little things like keeping track of number of nodes, etc.

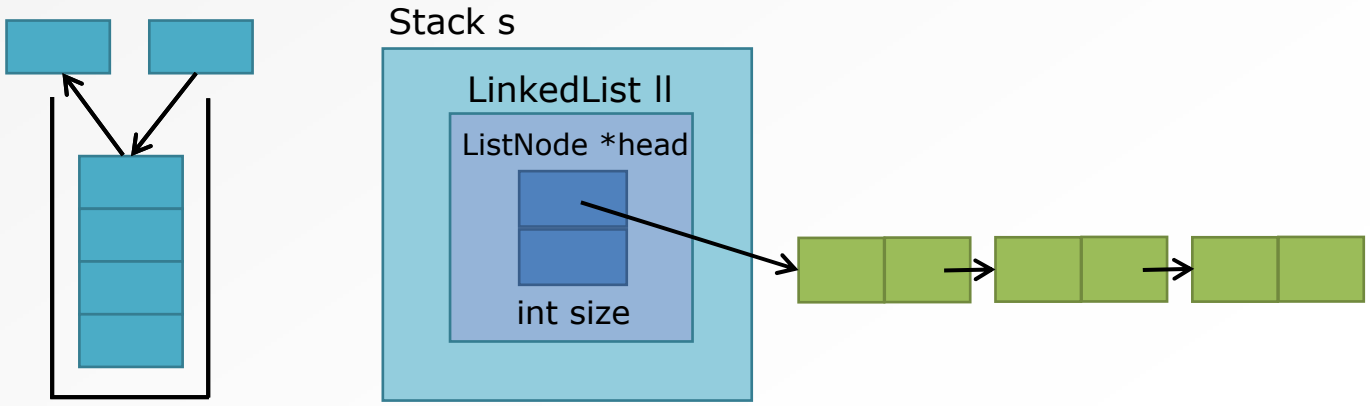


TODAY

- Motivating application
- Stack data structure
- Stack implementation using linked lists
- **Stack functions**
  - push()
  - pop()
  - peek()
  - isEmptyStack()
- Working examples: Applications

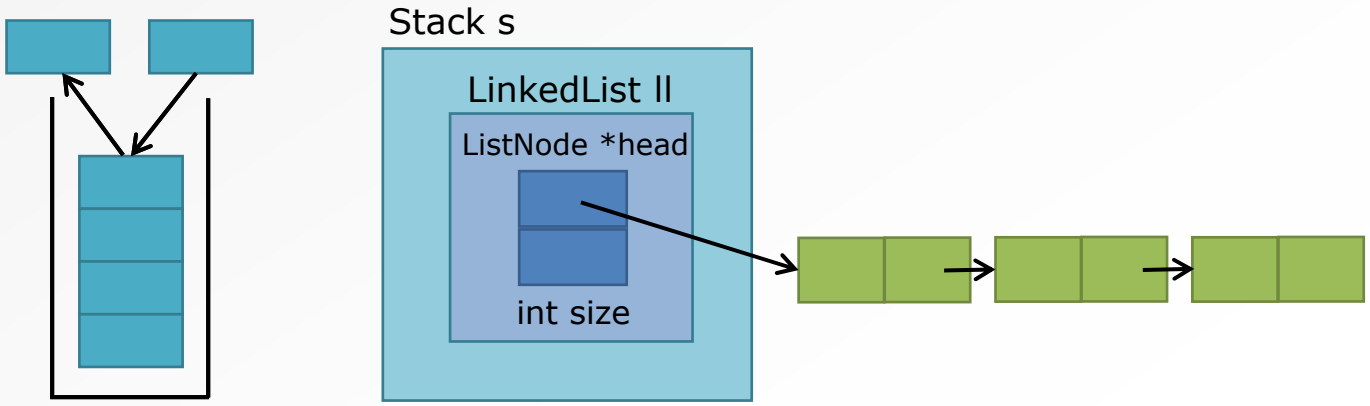
STACK FUNCTIONS: push()

- Push() function is the only way to add an element to the stack data structure
- Only allowed to push() onto the top of the stack
- Question:
  - Using a linked list as the underlying data storage, does the first linked list node represent the top of the bottom of the stack? Yes. (Index 0 position)



STACK FUNCTIONS: push()

- **Hands-on: Write the push() function**
  - Define the function prototype
  - Implement the function
- Answer is a few slides down, so don't look yet
- Requirements
  - Make use of the LinkedList functions we've already defined
  - Insert a new integer (what data type for the "item"?)
  - Insert at the top only (what index position?)



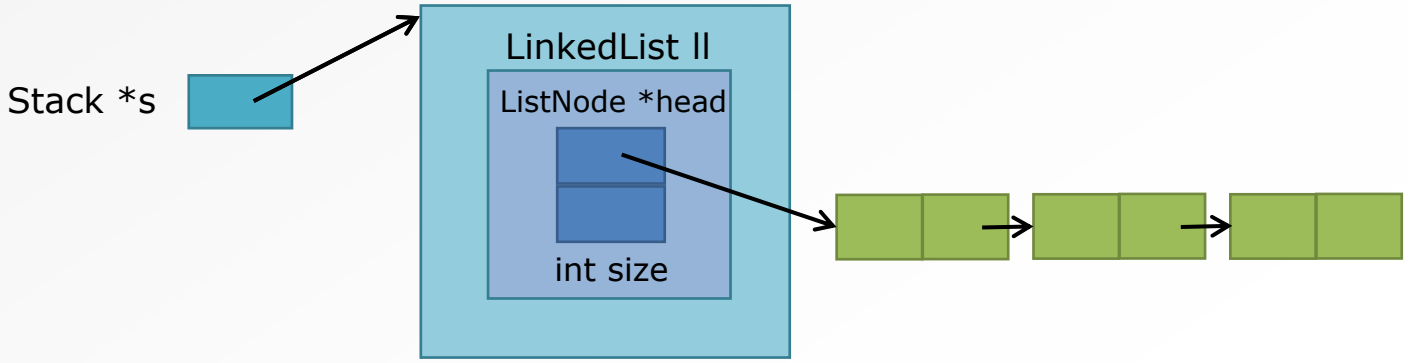
STACK FUNCTIONS: push()

- **Hands-on: Write the push() function**
  - Before looking at the code on the next slide, try writing the code for yourself

```
void push(Stack *s,                          ){



}
```



STACK FUNCTIONS: push()

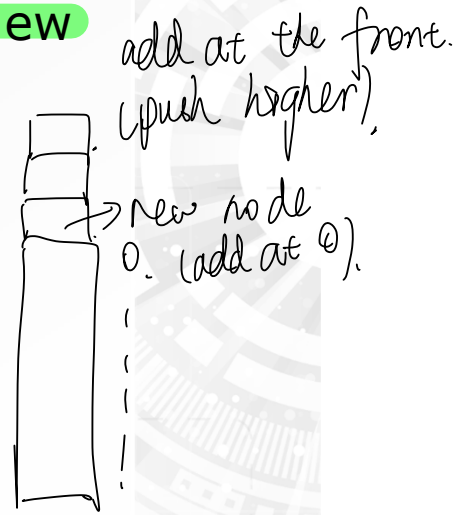
- First linked list node corresponds to the top of the stack
- Last linked list node corresponds to the bottom of the stack
- Pushing a new node onto the stack → adding a new node to the front of the linked list

⇒ add to start ⇒ stack

```
void push(Stack *s, int item){
    insertNode(&(s->ll), 0, item);
}
```

need to be an address

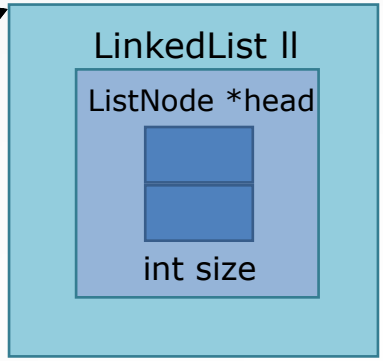
  - Notice that this is a very efficient operation
- We can also add the new nodes to the end of the linked list
  - Need to use a tail pointer to make the operation efficient



STACK FUNCTIONS: pop()

- Popping a value off the top of the stack is a two-step process
  - Get the value of the node at the front of the linked list
  - Removing that node from the linked list

```
int pop(Stack *s){
    int item;
    item = ((s->ll).head->item;
    removeNode(&(s->ll), 0);
    return item;
}
```



- Need a temporary int variable to hold the stored value because I can't get it after I remove the top node

STACK FUNCTIONS: peek()

- Peek at the value on the top of the stack
  - Get the value of the node at the front of the linked list
    - Without removing the node

```
int peek(Stack *s){
    return ((s->ll).head)->item;
}
```

STACK FUNCTIONS: isEmptyStack()

- Check to see if number of nodes == 0
- Make use of the built-in size variable in the LinkedList struct

```
int isEmptyStack(Stack *s){
    if ((s->ll).size == 0) return 1;
    return 0;
}
```

TODAY

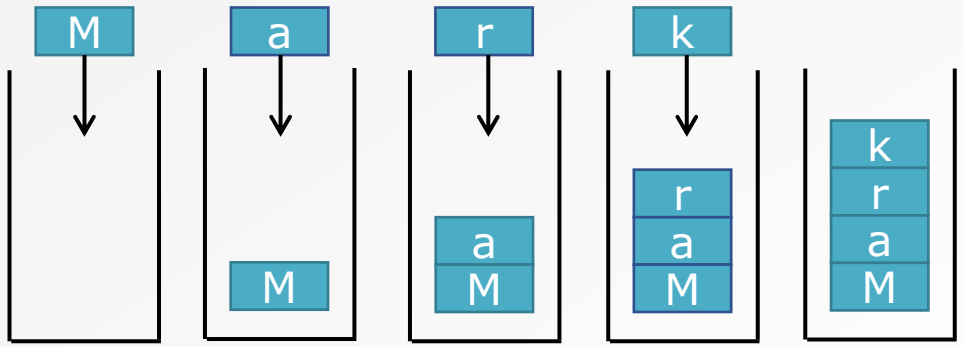
- Motivating application
- Stack data structure
- Stack implementation using linked lists
- Stack functions
  - push()
  - pop()
  - peek()
  - isEmptyStack()
- **Working examples: Applications**

SIMPLE APPLICATION #1: REVERSE STRING

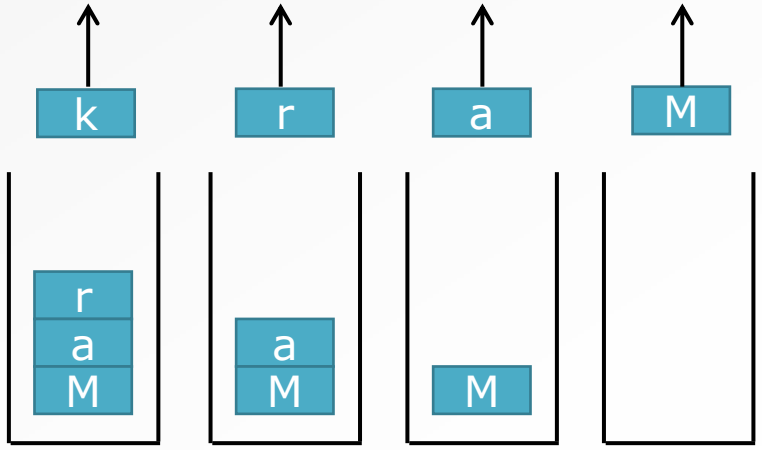
- Stacks are useful for reversing items
- Reverse a string: **Mark**
- Idea:
  - Push each letter on the stack
  - When there are no more letters in the original string, pop one by one from the stack
  - The letters will be popped in reverse order from their original position in the string

SIMPLE APPLICATION #1: REVERSE STRING

- Step 1
  - Push onto stack until no more letters



- Step 2
  - Pop from stack until stack is empty



SIMPLE APPLICATION #2: REVERSE LIST OF INTEGERS

- Similar to previous application, but with numbers

```
1   int main(){
2       int i = 0;
3       Stack s;
4       s.ll.head = NULL;
5
6       printf("Enter a number: ");
7       scanf("%d", &i);
8       while (i != -1){
9           push(&s, i);
10          printf("Enter a number: ");
11          scanf("%d", &i);
12      }
13      printf("Popping stack: ");
14      while (!isEmptyStack(&s))
15          printf("%d ", pop(&s));
16      return 0;
17  }
```

直接调用 pop 实现输出

BIGGER APPLICATION: MODELING A CONTAINER YARD

- Scenario:
  - Container yard with 10x10 grid layout, and each square has a stack of containers
  - Overhead crane can move over any stack
  - Crane can hold/inspect one container at a time from top of stack and move it to any other stack
  - Grid square at (0,0) is empty
- Task: Simulate the operation of this container yard

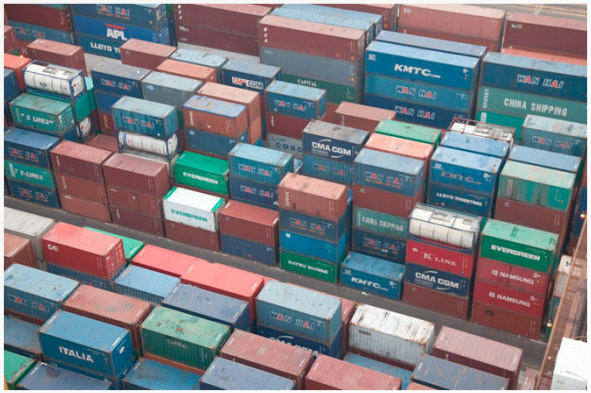


Image credits: http://kaidashton.blogspot.sg/2012/03/hong-kong-container-yards.html, http://es.made-in-china.com/co_goldhorse/product_RTG-Rubber-Tyre-Gantry-Crane-of-Port-for-Container-40FT-or-20FT-Payload-40-Ton_hhruniseg.html

BIGGER APPLICATION: MODELING A CONTAINER YARD

- Focus on one operation:
  - Find a container within a stack (each has an ID)
  - Given location of the stack (row,col)
  - Given ID of the container

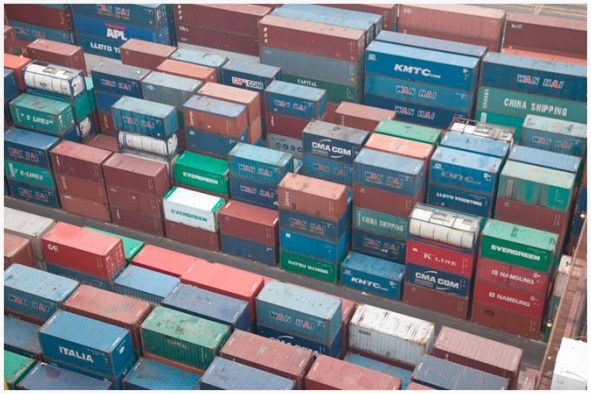
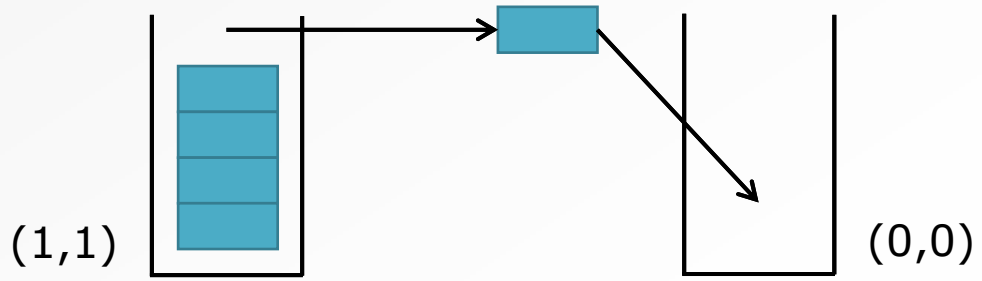


Image credits: http://kaidashton.blogspot.sg/2012/03/hong-kong-container-yards.html, http://es.made-in-china.com/co_goldhorse/product_RTG-Rubber-Tyre-Gantry-Crane-of-Port-for-Container-40FT-or-20FT-Payload-40-Ton_hhruniseg.html

BIGGER APPLICATION: MODELING A CONTAINER YARD

- To find a container with a certain ID (555) in a given stack (1,1):
  - Unload containers one at a time from the top of stack (1,1)
  - If ID matches, report found
  - Else, place container in temporary stack (0,0)
  - Unload next container (repeat)
- When container found or stack empty, need to replace unloaded containers
  - Unload one at a time from temporary stack (0,0)
  - Load one at a time on top of original stack (1,1)



MODELING A CONTAINER YARD (CODE PAGE 1)

```
int main(){

    int i, j, crane, row, col, targetid;
    Stack containeryard[10][10];

    for (i = 0; i < 10; i++)
        for (j = 0; j < 10; j++){
            containeryard[i][j].ll.head = NULL;
            containeryard[i][j].ll.size = 0;
        }

    row = col = 1;
    targetid = 555;

    // Initialize the target stack at (1,1)
    // Each container has an ID number - this will go in the stack
    for (i = 0; i < 10; i++)
        push(&(containeryard[row][col]), i*100+i*10+i);
```

MODELING A CONTAINER YARD (CODE PAGE 2)

```
    // Find a container within a stack
    // Row, col of stack and ID of container are given
    while (!isEmptyStack(&(containeryard[row][col]))){
        crane = pop(&(containeryard[row][col]));

        // Container found
        if (crane == targetid)
            break;

        // Still not found, so store this crane temporarily in (0,0)
        push(&(containeryard[0][0]), crane);
    }

    // Need to rebuild the original stack
    while (!isEmptyStack(&(containeryard[0][0])))
        push(&(containeryard[row][col]),
pop(&(containeryard[0][0])));

    if (crane == targetid)
        printf("Container found!\n");

    return 0;
}
```

TODAY

- Motivating application
- Stack data structure
- Stack implementation using linked lists
- Stack functions
  - push()
  - pop()
  - peek()
  - isEmptyStack()
- Working examples: Applications