HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

DATA STRUCTURES AND ALGORITHMS BASIC LAB

DATA STRUCTURES AND ALGORITHMS BASIC LAB

ONE LOVE. ONE FUTURE.

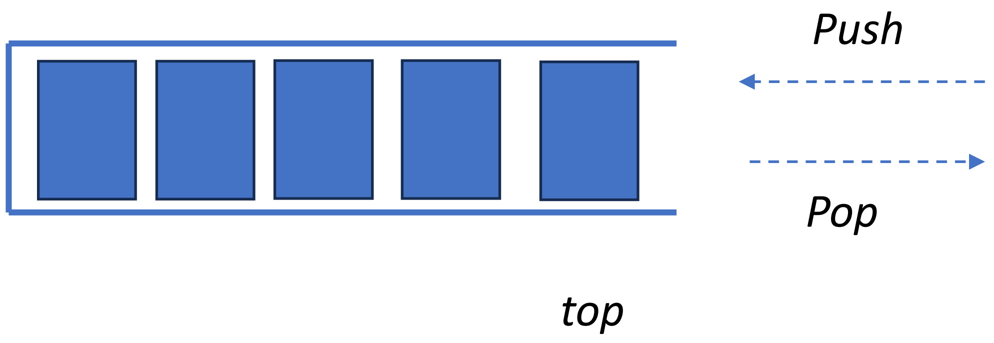
CONTENTS

- EXERCISE 1: SIMULATION STACK (P.03.08.01)
- EXERCISE 2: SIMULATION QUEUE (P.03.08.02)
- EXERCISE 3: CHECK PARENTHESIS (P.03.08.03)
- EXERCISE 4: WATER JUGS (P.03.08.04)

STACK AND QUEUE

A stack:

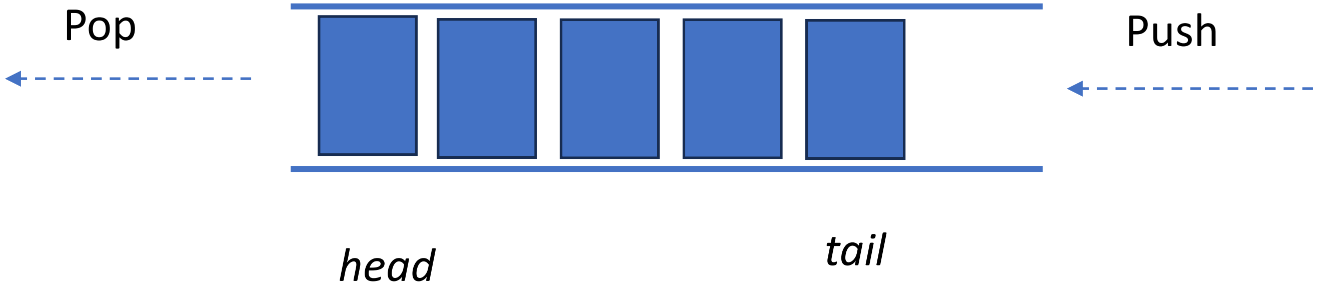
- A linear list of object
- Push and Remove are operated at top (head) of the list (First-In-Last-Out)
- Commonly used operations:
  - Push(x,S): Insert an element x into stack S
  - Pop(S): Remove an element from S
  - Top(S): Access the element on the top of S
  - Empty(S): Return true if S is empty



STACK AND QUEUE

A Queue:

- Is a linear list of objects. A queue has two pointers: *head* and *tail*
- Inserting a new element is operated at *tail* and removing is at *head* (First-In-First-Out)
- Commonly used operations:
  - Enqueue(x,Q) (Push): Insert a new element x into Q
  - Dequeue(Q) (Pop): Remove an element from Q
  - Empty(Q): Return true if Q is empty



EXERCISE 1: SIMULATION STACK

- Perform a sequence of operations over a stack, each element is an integer:
  - PUSH v: push a value v into the stack
  - POP: remove an element out of the stack and print this element to stdout (print NULL if the stack is empty)
- Input
  - Each line contains a command (operration) of type
    - PUSH v
    - POP
- Output
  - Write the results of POP operations (each result is written in a line)

EXERCISE 1: SIMULATION STACK

Example:

stdin	stdout
PUSH 1	3
PUSH 2	2
PUSH 3	5
POP	
POP	
PUSH 4	
PUSH 5	
POP	
#	

EXERCISE 1: SIMULATION STACK - PSEUDOCODE

- Using a singly linked list (pointed by *top*) to implement a stack:
  - Pop: remove an element on the *top*.
  - Push: add a new element to the *top*.

```
struct Node{
    int value;
    struct Node* next;
}
```

```
makeNode(x){
   p = new Node();
   p -> value = x;
   return p;
}
```

```
Pop(){
if top==NULL return NULL;
x = top;
top = top-> next;
return x;
}
```

```
Push(x){
p = makeNode(x);
p->next= top;
top = p;
}
```

EXERCISE 2: SIMULATION QUEUE (P.03.08.02)

- Perform a sequence of operations over a queue, each element is an integer:
  - PUSH v: push a value v into the queue
  - POP: remove an element out of the queue and print this element to stdout (print NULL if the queue is empty)
- Input
  - Each line contains a command (operration) of type
    - PUSH v
    - POP
- Output
  - Write the results of POP operations (each result is written in a line)

EXERCISE 2: SIMULATION QUEUE

Example 2.1:

stdin	stdout
PUSH 1	1
PUSH 2	2
PUSH 3	3
POP	
POP	
PUSH 4	
PUSH 5	
POP	
#	

EXERCISE 2: SIMULATION QUEUE

Example 2.2:

stdin	stdout
PUSH 1	1
POP	NULL
POP	4
PUSH 4	
POP	
#	

EXERCISE 2: SIMULATION QUEUE - PSEUDOCODE

- Using a singly link list pointed by *head* và *tail* to implement a queue:
  - Pop: remove the element at *head*
  - Push: add a new element to *tail*

```
struct Node{
    int value;
    struct Node* next;
}
```

```
makeNode(x){
   p = new Node();
   p -> value = x;
   return p;
}
```

```
Pop(){

if head = tail = NULL return ‘’;

v=head->value

head = head->next

return v

}
```

```
Push(x){

p = makeNode(x)

if head=tail=NULL then
head=tail=p; return;

tail->next = p

tail = p

return

}
```

EXERCISE 3: CHECK PARENTHESIS

- Given a string containing only characters (,), [,] {, }. Write a program that checks whether the string is correct in expression.

Example:

  - ([]{()}()[]): correct
  - ([]{()]()[]): incorrect

- Input
  - One line contains the string (the length of the string is less than or equal to 10^6
- Output
  - Write 1 if the sequence is correct, and write 0, otherwise

stdin	stdout
(()[][]{}){}{}[][]({[]()})	1

EXERCISE 3: CHECK PARENTHESIS - PSEUDOCODE

- Using a stack, from left to right, if meet an open parenthesis then push to the stack, otherwise (a close parenthesis):
    - If the top of the stack is the matched open parenthesis then pop from stack
    - Otherwise: return false

```
match(a,b){
if (a==‘(‘ and b ==’)’) or (a==‘{’ and
b==‘}’)
or (a==’[’ and b == ‘]’) return true;
return false;
}
```

```
Check(s){
stack h;
for i = 1…len(s){
   if s[i] in (’(’, ‘[‘, ‘{‘) then push(s[i]);
   else if (s[i] match top(h)) then pop(h);
   else return false;
}
if h is empty return true;
return false;
}
```

EXERCISE 4: WATER JUGS (P.03.08.04)

- There are two jugs, a-litres jug and b-litres jug (a, b are positive integers). There is a pump with unlimited water. Given a positive integer c, how to get exactly c litres.
- Input
  - Line 1: contains positive integers a, b, c (1 <= a, b, c <= 900)
- Output
  - Write the number of steps or write -1 (if no solution found)

Example:

stdin	stdout
6 8 4	4

EXERCISE 4: WATER JUGS - PSEUDOCODE

Idea: Enumerating the amount of water in the jugs (a pair of two integer x,y) using a queue with the shortest steps.

- Mark (x,y) if visited
- Pop each (x,y) at the head of the queue then push to the tail the new states not visited yet but can be reached from (x,y) by 1 step. Increase num_steps by 1.
- If reach to the target state then return num_steps , otherwise return -1 when finish.

```
mark(t){
   m[t] = 1;
}
```

```
is_mark(t){
   return m[t] == 1;
}
```

```
target(t){
   return c in t;
}
```

```
update_num_steps(t,r){
   num[t] = r + 1;
}
```

```
next_steps(t){
    r = list_of_next_1_step_from(t);
    return r;
}
```

```
Check(a,b,c){
(x,y) = (0,0); mark((0,0));
queue q; q.push((x,y))
while not empty(q){
t = pop(q);
for ti in next_steps(t){
   if target(ti) then return num[t]+1];
   if not is_mark(ti) then {push(ti); mark(ti);
update_num_steps(ti, num[t]);}
}
}
return -1;
}
```

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !