Stacks

Joseph Chuang-Chieh Lin (林莊傑)

Department of Computer Science & Engineering,
National Taiwan Ocean University

Fall 2025

Outline

1. Definition
2. Implementation

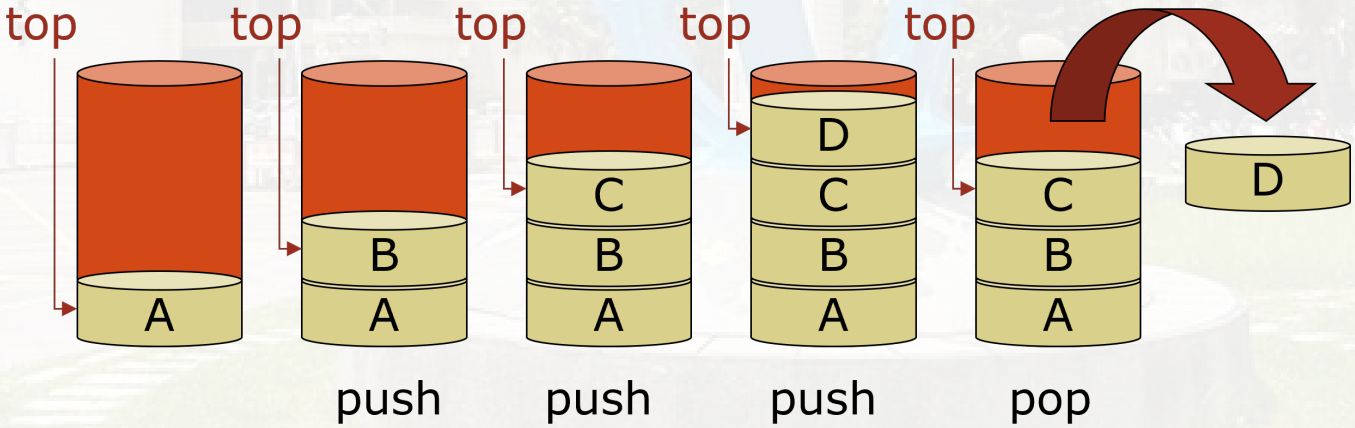
Outline

1 Definition

2 Implementation

Definition

- A stack is an ordered list in which insertions and deletions are made at the end "top".
  - insertions: push/add
  - deletions: pop/remove
- Last-In-First-Out (LIFO).



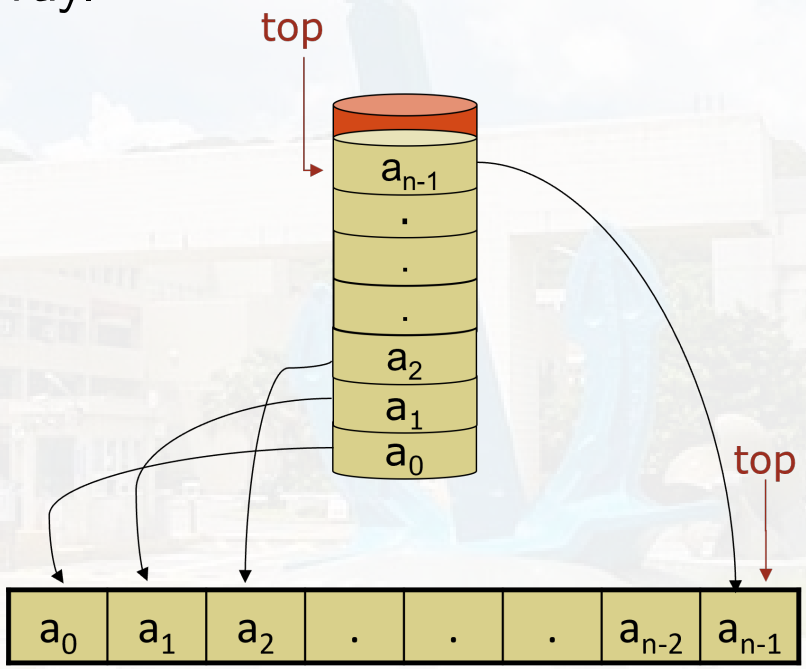
Outline

1 Definition

2 Implementation


Stack Implementation: Array

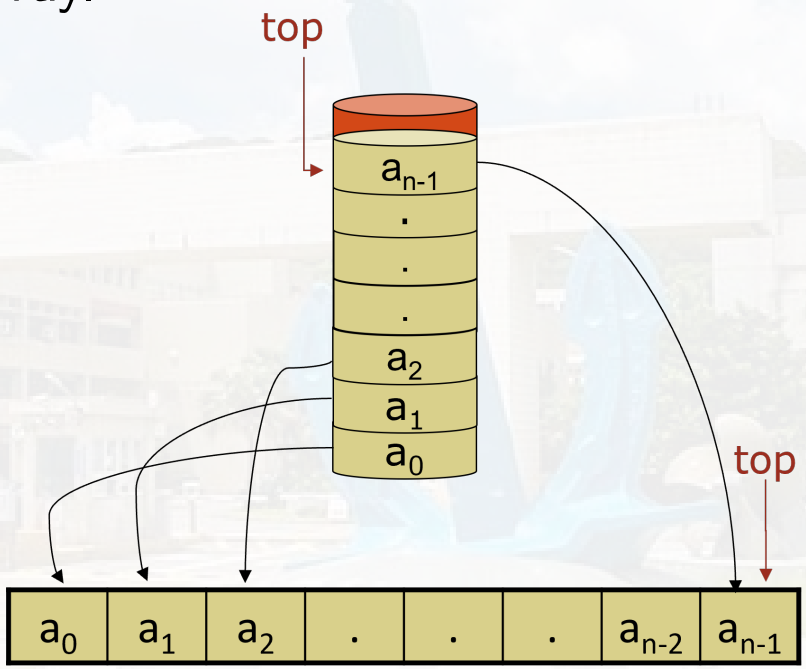
- The easiest way to implement the stack ADT is using one-dimensional array.



- An example in C++

Stack Implementation: Array

- The easiest way to implement the stack ADT is using one-dimensional array.



- An example in C++ (another way: using a linked list; will be introduced in the future).

Functions for Stacks

- Create a stack.
  - Create an empty stack with maximum size `MAX_STACK_SIZE`.

```c
#define MAX_STACK_SIZE 101

typedef struct {
    int key; // can be of other types...
    /* other fields? */
} element;

element stack[MAX_STACK_SIZE];
int top = -1; // initially no element
```

Functions for Stacks (2/2)

- IsEmpty
  - Return `true` if the stack is empty and `false` otherwise.

Functions for Stacks (2/2)

- IsEmpty
  - Return `true` if the stack is empty and `false` otherwise.
    `top < 0`
- IsFull
  - Return `true` if the stack is full and `false` otherwise.

Functions for Stacks (2/2)

- IsEmpty
  - Return `true` if the stack is empty and `false` otherwise.
    `top < 0`
- IsFull
  - Return `true` if the stack is full and `false` otherwise.
    `top >= MAX_STACK_SIZE-1;`
- Push (or Add)
  - Insert the element into the top of the stack.

Functions for Stacks (2/2)

- IsEmpty
  - Return `true` if the stack is empty and `false` otherwise.
    `top < 0`
- IsFull
  - Return `true` if the stack is full and `false` otherwise.
    `top >= MAX_STACK_SIZE-1;`
- Push (or Add)
  - Insert the element into the top of the stack.
    `stack[++top] = element;`
- Pop (or Delete)
  - Remove and return the item on the top of the stack.

Functions for Stacks (2/2)

- IsEmpty
  - Return `true` if the stack is empty and `false` otherwise.
    `top < 0`
- IsFull
  - Return `true` if the stack is full and `false` otherwise.
    `top >= MAX_STACK_SIZE-1;`
- Push (or Add)
  - Insert the element into the top of the stack.
    `stack[++top] = element;`
- Pop (or Delete)
  - Remove and return the item on the top of the stack.
    `return stack[top--];`

isFull & isEmpty

```
bool isEmpty() {
// note: bool type requires stdbool.h
    if (top == -1) return true;
    else return false;
}

bool isFull() {
    if (top >= MAX_STACK_SIZE-1) return true;
    else return false;
}
```

push & stackFull

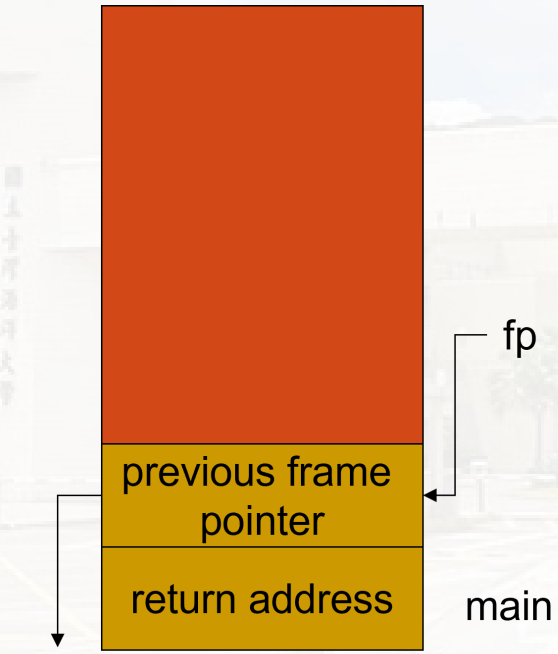
```
void push(element item) {
// add an item to the global stack
    if (top >= MAX_STACK_SIZE - 1) {
        stackFull();
    }
    stack[++top] = item;
}

void stackFull() {
    fprintf(stderr, "Stack is FULL!!");
    exit(EXIT_FAILURE);
}
```

pop & stackEmpty

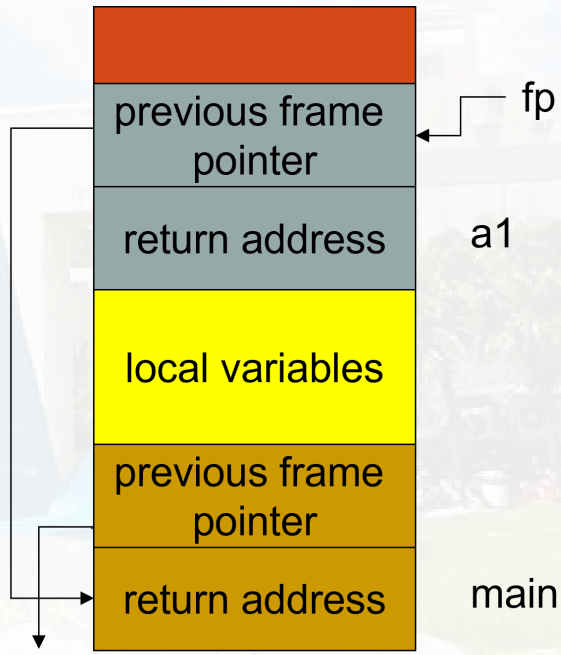
```
element pop() { // no argument is required!
// return the top element from the stack
    if (top == - 1) {
        return stackEmpty(); // return an error key
    }
    return stack[top--];
}

element stackEmpty() {
    element errKey; // depending the struct of element
    errKey.key = -99;
    fprintf(stderr, "Stack is EMPTY!!");
    return errKey;
}
```

Supplementary: System Stack

- Stack frame of a function call



system stack **before** a_1 is invoked



system stack **after** a_1 is invoked

Easy Example: Summation from 1 to n

```
int sum(int n) {
    element result;
    result.key = 0;
    element tmp;
    tmp.key = n;
    push(tmp); // bootstrapping here!

    while (!isEmpty()) {
        tmp = pop(stack);
        result.key += tmp.key;
        tmp.key = tmp.key-1;
        if (tmp.key > 0) push(tmp);
    }
    return result.key;
}
```

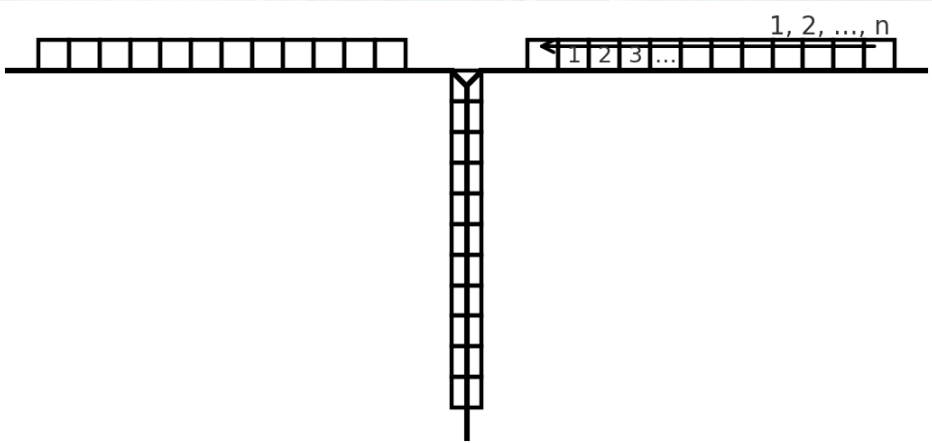
Left Home Exercise: Fibonacci

- Try to use a stack to turn the following recursive function into a non-recursive one.

```
int Fibo(int n) {
    if (n == 0) return 0;
    else if (n == 1) return 1;
    else return Fibo(n-1) + Fibo(n-2);
}
```

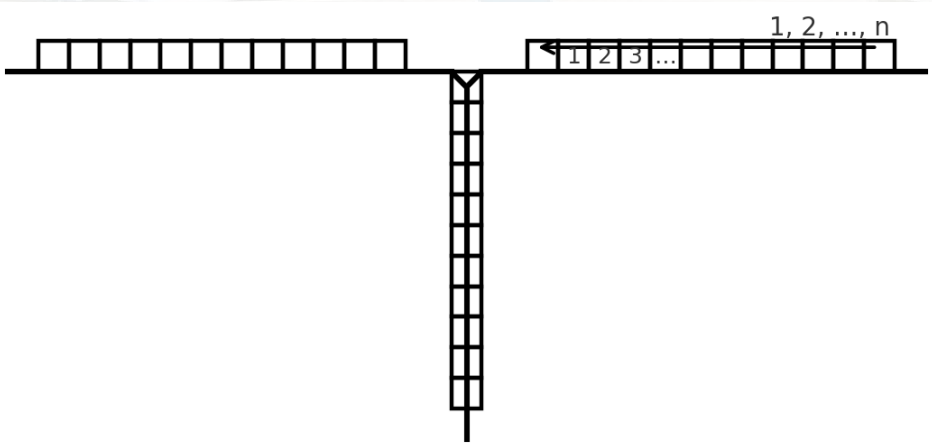

Exercise: Railroad Switching Network (1/2)

- Consider the railroad switching network given in the below figure. Railroad cars numbered $1, 2, 3, \ldots, n$ are initially in the top right track segment.
- Railroad cars can be moved into the vertical track segment one at a time from either of the horizontal segments and then moved from the vertical segment to any one of the horizontal segments.
- The vertical segment operates as a stack as new cars enter at the top and cars depart the vertical segment from the top.



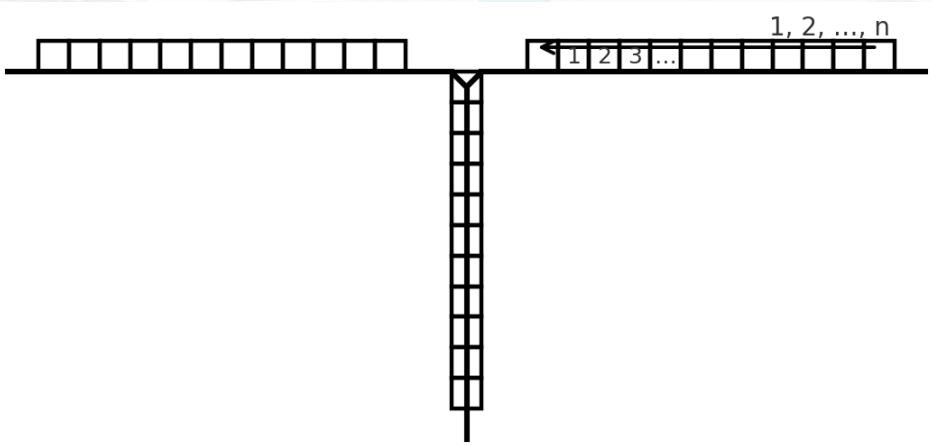
Exercise: Railroad Switching Network (2/2)

- For $n = 3$ and $n = 4$, what are the possible permutations of the cars that can be obtained? Are any permutations not possible?



Exercise: Railroad Switching Network (2/2)

- For $n = 3$ and $n = 4$, what are the possible permutations of the cars that can be obtained? Are any permutations not possible?



- What if the stack has the capacity limit (say, two)?

Discussions