Data Structure and Algorithms

Lecture 3

Stacks

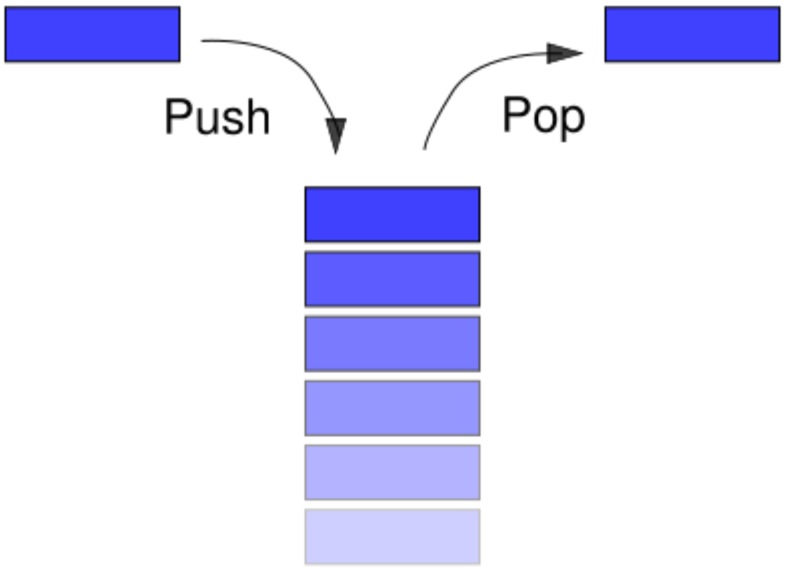
Stacks

- A stack is a list in which insertion and deletion take place at the same end
  - This end is called top
  - The other end is called bottom

- Stacks are known as LIFO (Last In, First Out) lists.
  - The last element inserted will be the first to be retrieved

- E.g. a stack of Plates, books, boxes etc.

Insertion and deletion on stack



Stack-Related Terms

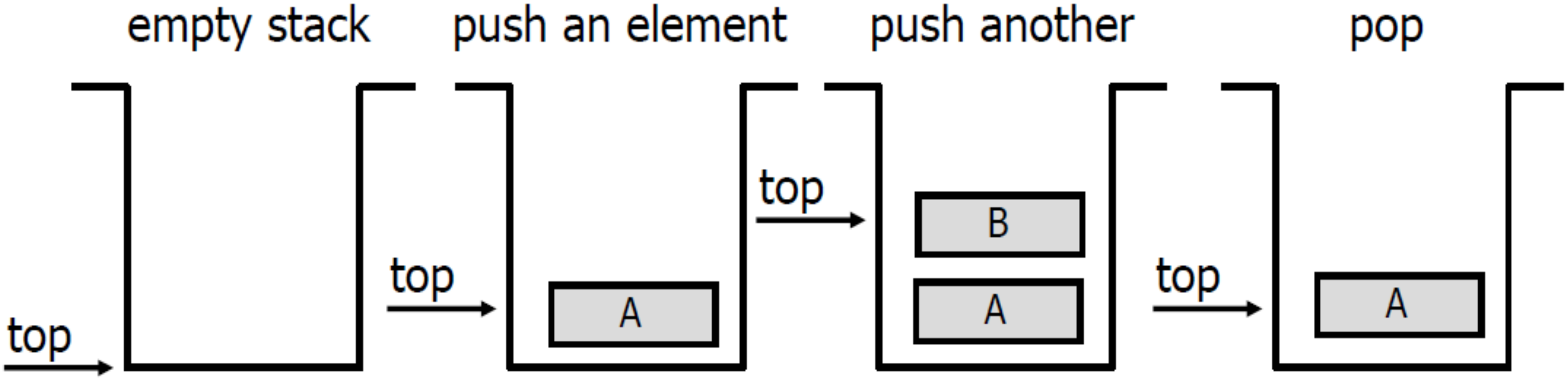
- Top
  - A pointer that points the top element in the stack.
- Stack Underflow (Empty Stack)
  - When there is no element in the stack, the status of stack is known as stack underflow.
- Stack Overflow (Full Stack)
  - When the stack contains equal number of elements as per its capacity and no more elements can be added, the status of stack is known as stack overflow

Operation On Stack

- Creating a stack
- Checking stack---- either empty or full
- Insert (PUSH) an element in the stack
- Delete (POP) an element from the stack
- Access the top element
- Display the elements of stack

Push and Pop

- Primary operations: Push and Pop
- Push
  - Add an element to the top of the stack.
- Pop
  - Remove the element at the top of the stack.



Stack applications

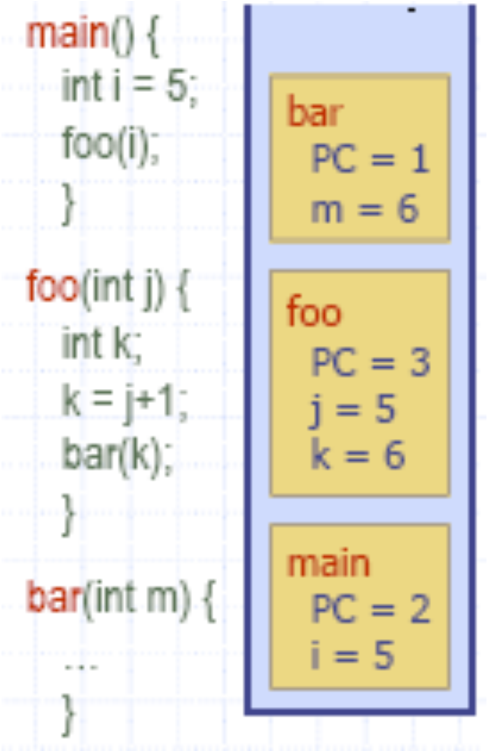
- “Back” button of Web Browser
  - History of visited web pages is pushed onto the stack and popped when “back” button is clicked
- “Undo” functionality of a text editor
- Reversing the order of elements in an array
- Saving local variables when one function calls another, and this one calls another, and so on.

C++ Run-time Stack

- The C++ run-time system keeps track of the chain of active functions with a stack

- When a function is called, the run-time system pushes on the stack a frame containing
  - Local variables and return value
  - Program counter, keeping track of the statement being executed

- When a function returns, its frame is popped from the stack and control is passed to the method on top of the stack



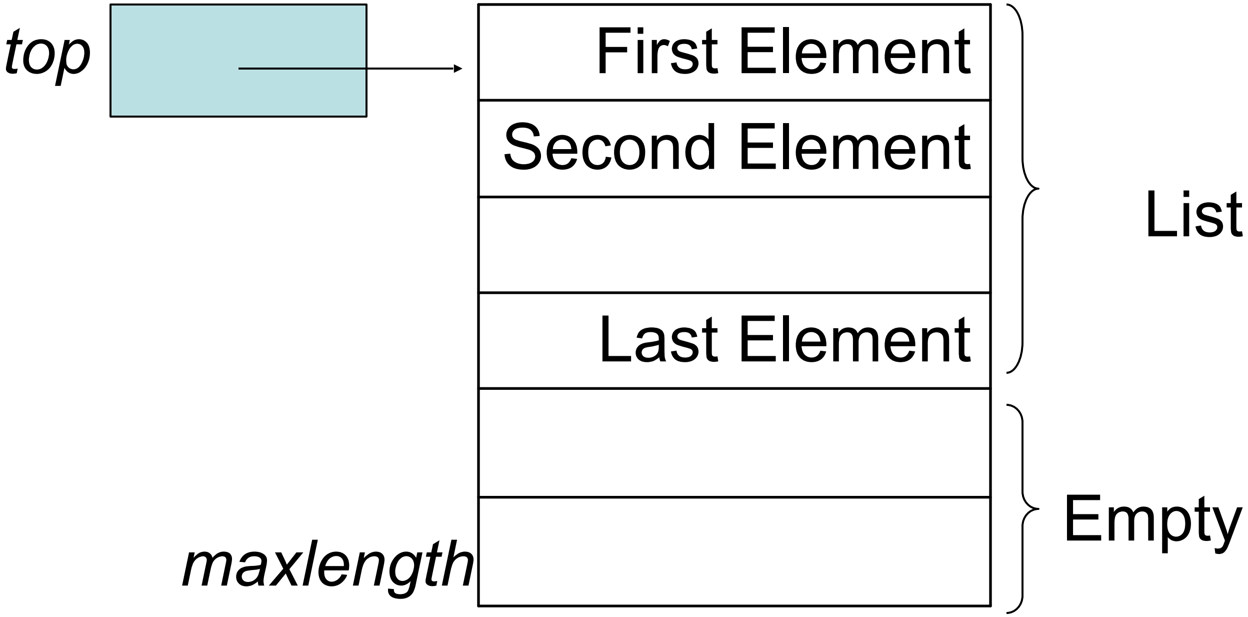
```
main() {
  int i = 5;
  foo(i);
  }

foo(int j) {
  int k;
  k = j+1;
  bar(k);
  }

bar(int m) {
  ...
  }
```

Stack Implementation

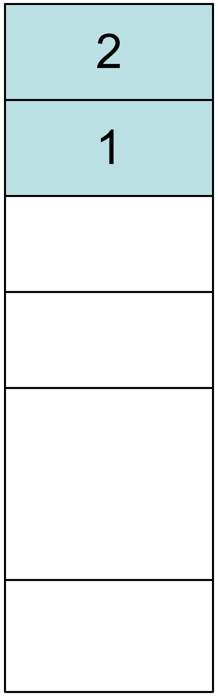
- Implementation can be done in two ways
  - Static implementation
  - Dynamic Implementation
- Static Implementation
  - Stacks have **fixed size**, and are implemented as **arrays**
  - It is also inefficient for utilization of memory
- Dynamic Implementation
  - Stack **grow in size** as needed, and implemented as **linked lists**
  - Dynamic Implementation is done through pointers
  - The memory is efficiently utilize with Dynamic Implementations

Static Implementation

- Elements are stored in contiguous cells of an array.
- New elements can be inserted to the top of the list.



Static Implementation



Problem with this implementation

- Every PUSH and POP requires moving the entire array up and down.

Static Implementation

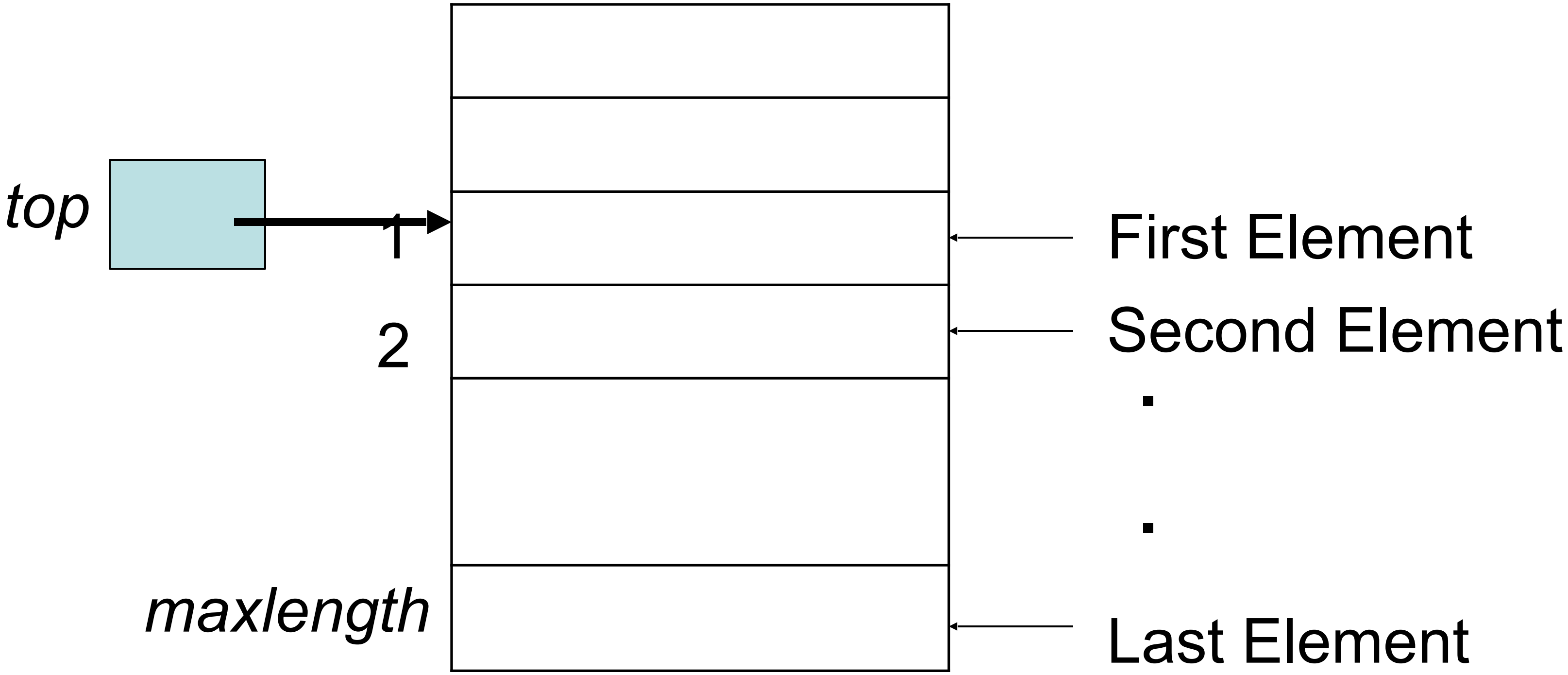
Since, in a stack the insertion and deletion take place only at the top, so…

A better Implementation:

- Anchor the bottom of the stack at the bottom of the array
- Let the stack grow towards the top of the array
- *Top* indicates the current position of the first stack element.

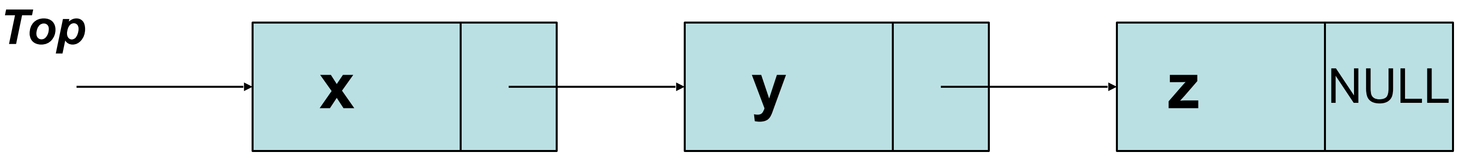
Static Implementation

A better Implementation:



Dynamic Implementation of Stacks

- As we know that dynamic stack is implemented using linked-list.
- In dynamic implementation stack can expand or shrink with each PUSH or POP operation.
- PUSH and POP operate only on the first/top cell on the list.



Stack with Array

```
Type TStack : <stack : Array[1..MAX] of Int
               top : Int>

Function isEmpty(I stk: TStack) -> Boolean {

}

Function isFull(I stk: TStack) -> Boolean {

}
```

Stack with Array

```
Type TStack : <stack : Array[1..MAX] of Int
                    top : Int>

Function isEmpty(I stk: TStack) -> Boolean {
    return (stk.top == 0)
}

Function isFull(I stk: TStack) -> Boolean {
    return (stk.top == MAX)
}
```

Stack with Array : Latihan

```
Procedure push(I stk: TStack, data: Int) {
    2 case : ifFull & bisa push
}

Function pop(I stk: TStack) -> Int {
    2 case : ifEmpty & bisa pop
}
```

Stack with Linked List

```
Type TNode : <data : Int
              next : pointer of TNode>
Type TStack : <top : pointer of TNode>

Function isEmpty(I stk: TStack) -> Boolean {

}

Function isFull(I stk: TStack) -> Boolean {

}
```

Stack with Linked List

```
Type TNode : <data : Int
                    next : pointer of TNode>
Type TStack : <top : pointer of TNode>

Function isEmpty(I stk: TStack) -> Boolean {
    return (stk.top == null)
}

~~Function isFull(I stk: TStack) -> Boolean {~~
    TIDAK MUNGKIN FULL
}
```

Stack with Linked List

```
Procedure push(I stk: TStack, data: Int) {
    2 case : ifEmpty (setHead) & bisa push (addFirst)
}

Function pop(I stk: TStack) -> Int {
    2 case : ifEmpty & bisa pop (getFirst)
}
```