Stacks

Maunendra Sankar Desarkar

IIT Hyderabad

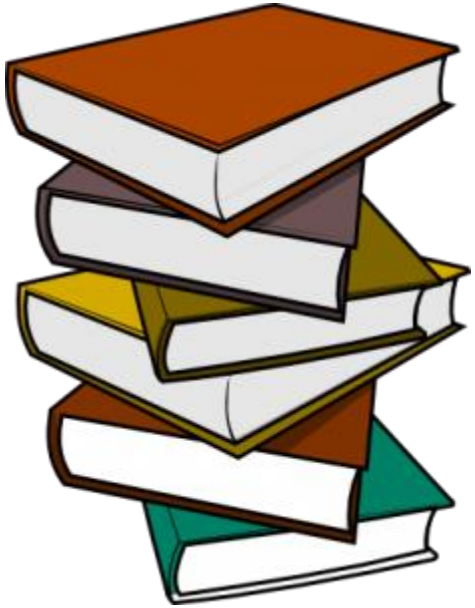
CS1353: Introduction to Data Structures

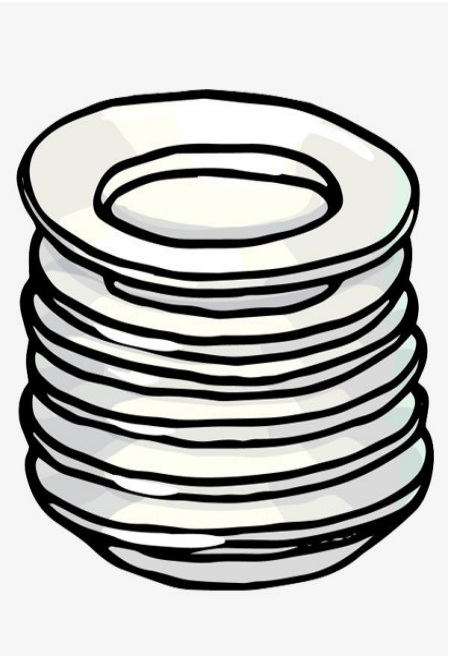
Acknowledgements

- Slides are adapted versions of other publicly available slides on the same topic.

Stacks

- A stack is a last in, first out (LIFO) data structure
  - Items are removed from a stack in the reverse order from the way they were inserted





Recursion: example

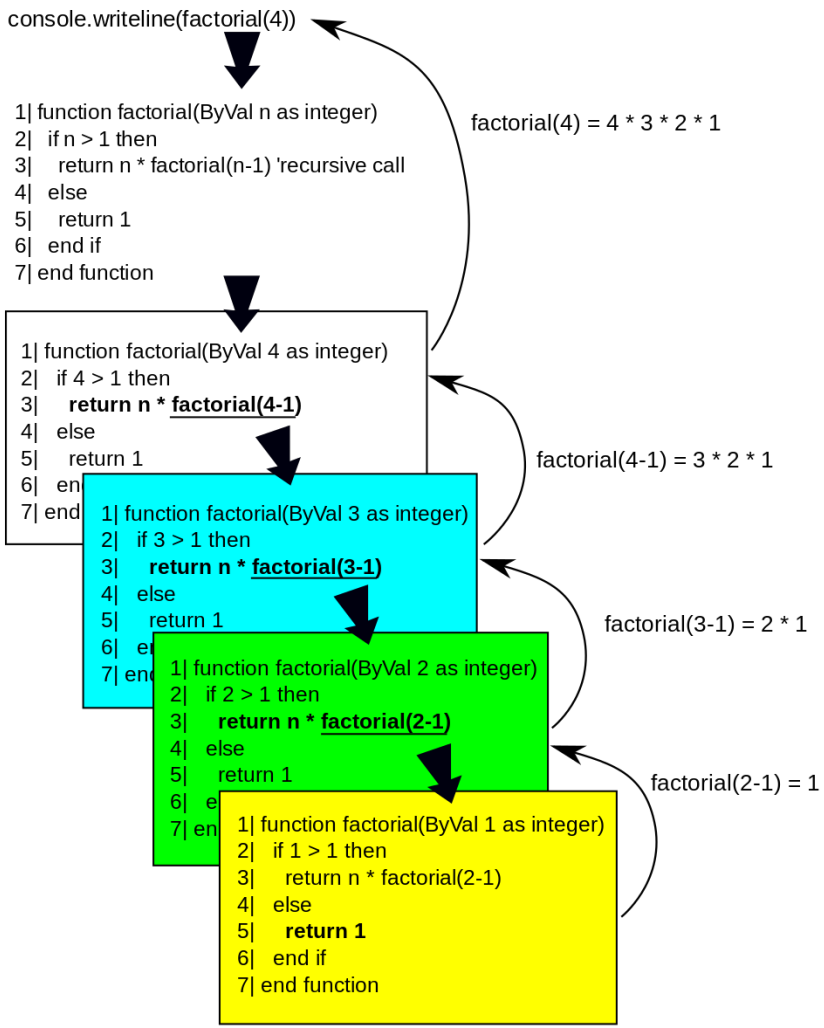


Figure from Wikipedia

Nested calls: example

```
void func1() {
    int a=1;
    cout<<"a="<<a<<". Inside function 1."
        <<"About to call function 2\n";
    func2();
    cout<<"a="<<a<<". Function 2 ended, "
         <<"about to exit from function 1.\n";
}
```

```
void func2() {
    int a=2;
    cout<<"a="<<a<<". Inside function 2. "
        <<"About to call function 3\n";
    func3();
    cout<<"a="<<a<<". Function 3 ended, "
         <<"about to exit from function
2.\n";
}
```

```
void func3() {
    int a=3;
    cout<<"a="<<a<<". Inside function 3 ."
        <<"About to call function 4\n";
    func4();
    cout<<"a="<<a<<". Function 4 ended, "
         <<"about to exit from function
3.\n";
}
```

```
void func4() {
    int a=4;
    cout<<"a="<<a<<". Inside function 4. "
        <<"No other nested calls\n";
    cout<<"a="<<a<<". About to exit from "
         <<"function 4.\n\n--------------------\n\n";
}
```

```
int main() {
        func1();
        return 0;
}
```

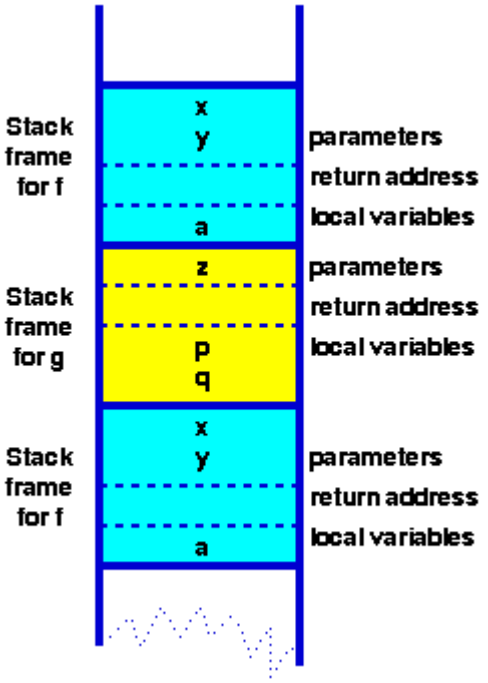
Sample output

```
a=1. Inside function 1. About to call function 2
a=2. Inside function 2. About to call function 3
a=3. Inside function 3. About to call function 4
a=4. Inside function 4. No other nested calls
a=4. About to exit from function 4.

------------------

a=3. Function 4 ended, about to exit from function 3.
a=2. Function 3 ended, about to exit from function 2.
a=1. Function 2 ended, about to exit from function 1.
```

Call stack

Call stack

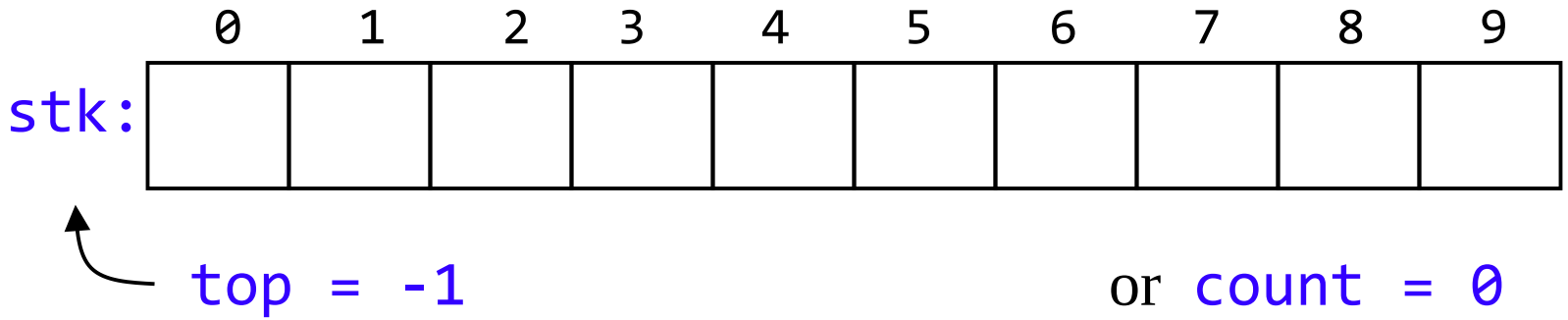
Nr	Address	Function
0		func4 ()
1	0x40175d	func3()
2	0x4017de	func2()
3	0x40185f	func1()
4	0x4018a8	main()



Array implementation of stacks

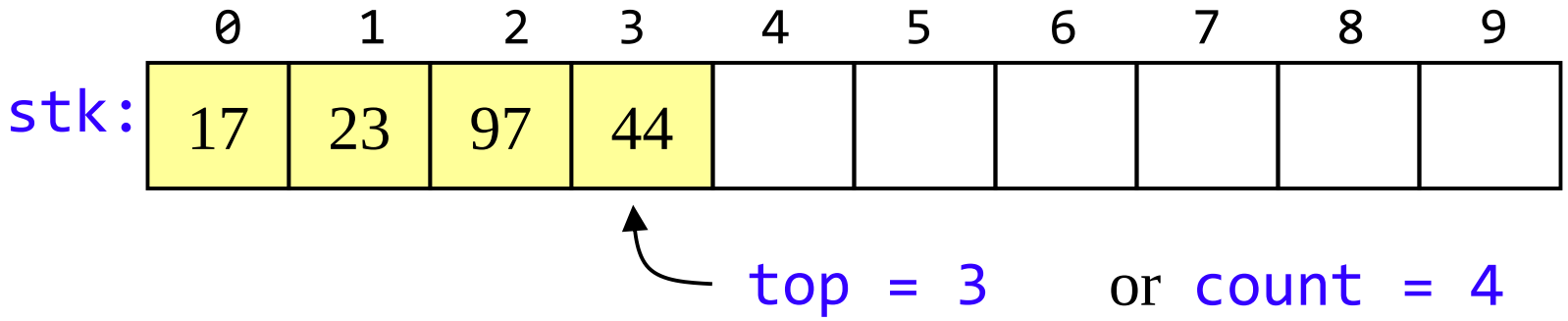
- Items are inserted and removed at the same end
  - Generally called the top
- To use an array to implement a stack, we need: an array and an integer
  - Array
    - The container: contains all elements of the stack
  - Integer
    - Helps to identify the top of the stack, or
    - How many elements are in the stack

Pushing and popping



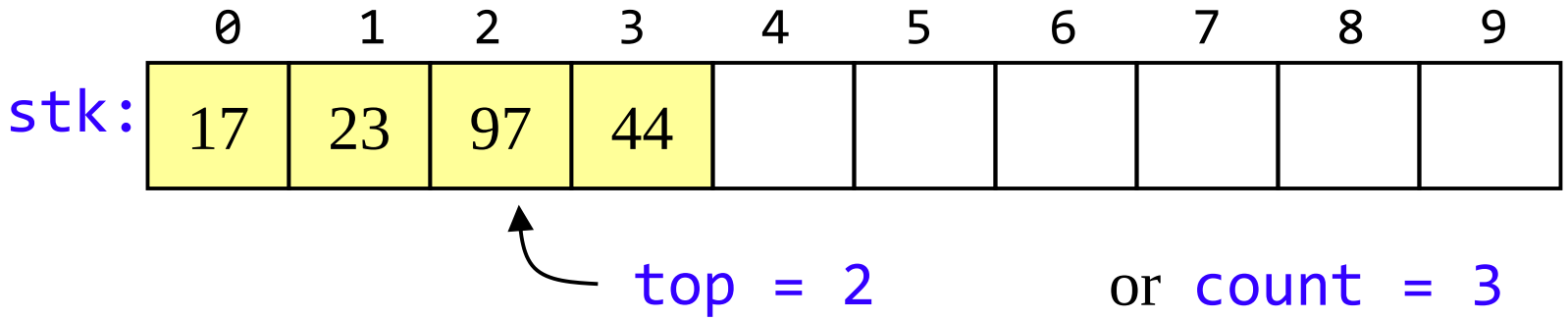
- If the bottom of the stack is at location `0`, then an empty stack is represented by `top = -1` or `count = 0`

Pushing and popping



- To add (push) an element, either:
  - Increment `top` and store the element in `stk[top]`, or
  - Store the element in `stk[count]` and increment `count`
- To remove (pop) an element, either:
  - Get the element from `stk[top]` and decrement `top`, or
  - Decrement `count` and get the element in `stk[count]`

After popping



- When you pop an element, do you just leave the "deleted" element sitting in the array?
- The surprising answer is, *"it depends"*
  - If this is an array of primitives, *or* if you are programming in C or C++, *then* doing anything more is just a waste of time
  - If you are programming in Java, and the array contains objects, you should set the "deleted" array element to `null`
  - Why? To allow it to be garbage collected!

Code snippets

- Next, we will have some code snippets
- Some of those snippets are wrong/have limitations
- You need to identify the issues with those snippets (if any)

Push: inserting element to stack

```
int a[MAX];
int top=-1;


int push(int x)
{
        // What if array is full?
        // Check overflow
        a[++top] = x;
        cout<<x <<" pushed into
    stack\n";
        return 1;
}
```

```
int a[MAX];
int top=-1;
int isFull();
int push(int x)
{
    // What if array is full?
    if (!isFull()) {
        a[++top] = x;
        cout<<x <<" pushed into
    stack\n";
        return 1;
    }
}
int isFull() {
    if(top>=MAX-1)
        return 1;
    else
        return 0;
}
```

Push: inserting element to stack

```
int a[MAX];
int top=-1;


_______ pop(_____)
{
    // What if array is empty?
    // Check underflow
    top = top-1;
    return a[top+1];
}
```

```
int a[MAX];
int top=-1;

int pop()
{
    // What if array is empty?
    // Check underflow
    return a[top--];
}
```

Push: inserting element to stack

```
int a[MAX];
int top=-1;

int pop(_________)
{
    // What if array is empty?
    // Check underflow
    return a[--top];
}
```

```
int a[MAX];
int top=-1;
int isEmpty();
int pop()
{
    // What if array is empty
    if (!isEmpty()) {
        int x=a[top];top--;
        return x;
    }
}
int isEmpty() {
    if(top<0)
        return 1;
    else
        return 0;
}
```

Error checking

- There are two stack errors that can occur:
  - **Underflow**: trying to pop (or peek at) an empty stack
  - **Overflow**: trying to push onto an already full stack
- Throw error messages/statements whenever the above conditions are hit