Agenda

Coming up:

- Midterm 1:
  - Friday, Feb 21st, 5-7PM
  - Location: DUAN G1B20
  - **Info + practice exam on Moodle**
  - **No lecture next Friday**

This week:

- Assignment 4:
  - due this Sunday, 6PM (Feb. 16)

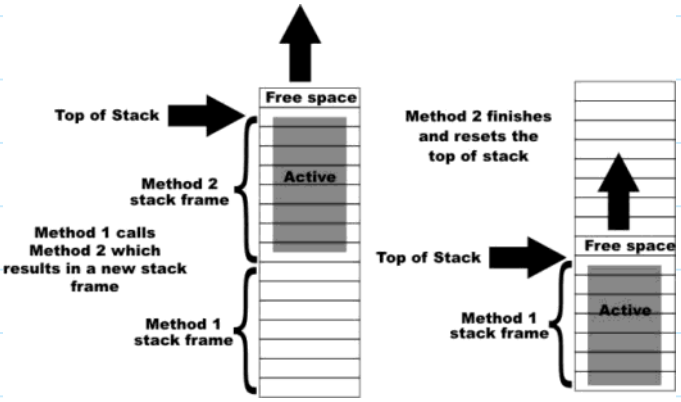
Today:

- The Stack Data Structure
- The Queue DS

The Stack Data Structure

- Last In First Out data structure (LIFO)
  A "limited access" DS
    - can only add to the top
    - can only remove from the top
    - depends on implementation, can a hard limit on the size
  Usage examples:
    - call stack; during program execution, keeping track of active subroutines (automatic variables)
    - word processor "undo"

example:



credit: https://www.i-programmer.info

example:

paste text
insert image
delete "there"
type "there"
type "hello"

Stack ADT

abstracting the implementation: what functionality do we want from this DS?

private:

top - keeps track of the top element (int or ptr?)
maxSize - limit on total size of stack (optional - depends on implementation)
count - current number of elements in stack

public:

initialize - (constructor)
destroy - (destructor)
isFull() - optional depending on implementation
isEmpty()
push(item) - add a new item to top
pop() - discard from top
peek() - shows the top item
display() - optional

note: no generic insert or delete

Note that the ADT does not specify anything about the implementation.

- *linked list (singly)*
- *array*

Stack SLL implementation 0

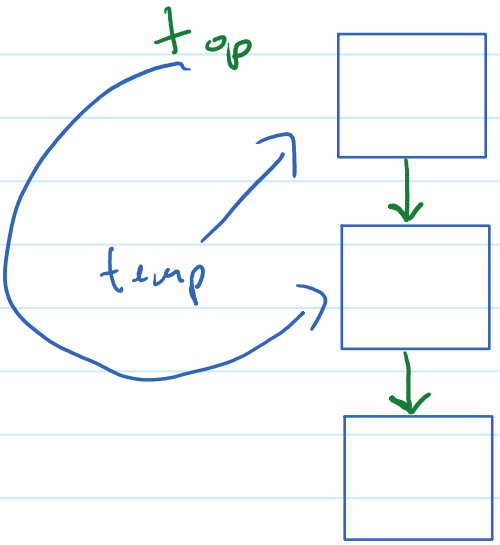
Thursday, February 13, 2020 2:28 PM

```
struct Node{
   std::string item;
   Node *next;
};
class Stack{
private:
   // pointer to top of stack
   Node *top;
   // number of nodes currently in stack
   int count;
public:
   Stack(); // constructor
   ~Stack(); // destructor
   bool isEmpty();
   void push( string newItem );
   // Precondition: newItem parameter is a string type
   // Postcondition: dynamically allocate a new nodea and push onto stack
   void pop();
   // Precondition: none
   // Postcondition: remove the node from top of stack and deallocate the
   // node's memory

   Node* peek();
   // Precondition: none
   // Postcondition: return a pointer to the node that corresponds to the
   // top of stack

   void disp();
   // Precondition: none
   // Postcondition: display the contents of entire stack
};
```

Stack SLL impementation 1

Tuesday, February 5, 2019 5:25 PM

```
Stack::Stack(){
    top = nullptr;
}

bool Stack::isEmpty(){
    if(top==nullpt)
        return true;
    else
        return false;

}

// isFull - not needed

void Stack::push( string newItem ){
    Node *temp = new Node;
    temp->item = newItem;

    if(isEmpty()){
        top = temp;
        top->next = nullptr;
    }
    else{
        temp->next = top;
        top = temp;

    }
// big-O = O(1)
}

void Stack::pop(){
   Node *temp;
    if(!isEmpty()){
        temp = top;
        top = top->next
        delete temp;
    }
    else{
        cout << "stack underflow";
    }
```

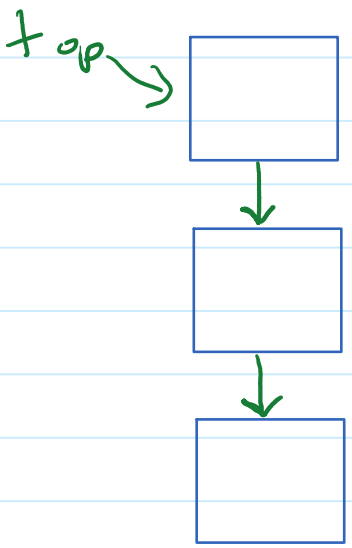


```
// big-O = O(1)

}
```

Stack SLL impementation 2

Tuesday, February 5, 2019 5:25 PM

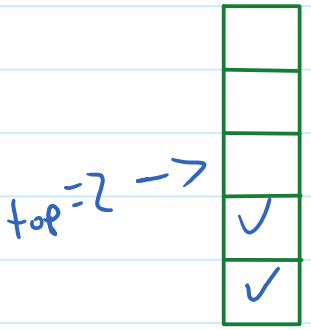
```
Node* Stack::peek(){
   return top;
}

void Stack::disp(){
    // optional method (maybe for debug)
   Node *current = top;
   cout << "Top of the stack: " << endl;
   while( current != nullptr ){
      cout << current->item << endl;
      current = current->next;
   }
   cout << endl;
}


Stack::~Stack(){
   Node *current;
   while( !isEmpty() ){
      current = top;
      top = top->next;
      delete current;
   }
}
```



Stack - array implementation

Monday, September 23, 2019 11:53 AM

```
private:

    int top; // index of next available element
    int count; // current number of elements
    string s[MAX_SIZE];// the stack array

public:
    bool isFull(){
        return top==MAX_SIZE;
    }

    bool isEmpty(){
         return top==0;
    }

    string peek(){
        if(!isEmpty)
            return s[top-1];
    }

    void push(string newItem){
        if(!isFull)
            s[top] = newItem;
            top++;
        else
            "stack overflow"
            // add array doubling
        }

    void pop() {

    }
```



big-O for push, pop, and peek? *O(1)*

Stack implementations summary

Monday, September 23, 2019 11:53 AM

In summary: we can implement a stack with an underlying array or Linked List.
How do we decide which one to choose?

Array based:

Pros:

- memory efficient
- fast

Cons:

- fixed size
- if using dynamic memory, not linear speed

LL based:

Pros:

- no size limitation

Cons:

- slower (access the heap every time)