Stacks and Queues

Dr. Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

Input and Output

Your Objectives:

- You already know about these! But...
- We will cover the STL implementations.
- Also linked lists....

- Last in First out — $\mathcal{O}(1)$ access to top element only.
- Use in many algorithms. E.g., brace matching, strongest connected components.
- C++ built-in: `push(x)`, `pop()`, `top()`, `empty()`.

```
// stack::emplace -- from cplusplus.com/reference/stack
#include <iostream>       // std::cin, std::cout
#include <stack>          // std::stack
#include <string>         // std::string, std::getline(string)

int main () {
  std::stack<std::string> mystack;
  mystack.emplace ("First sentence");
  mystack.emplace ("Second sentence");
  std::cout << "mystack contains:\n";
  while (!mystack.empty()) {
    std::cout << mystack.top() << '\n';
    mystack.pop();
  }
```


Queues

- FIFO — used for BFS, scheduling, etc.
- STL Queues use a *doubly ended* underlying implementation by default.
- This gives you $\mathcal{O}(1)$ access to the front and the back.
- View: `back()` and `top()`
- Insert: `push()`. If you need explicit access, use `deque` directly.
- Do **not** use pointer arithmetic to access elements!

Lists

- There is a STL `list` class.
- For most problems you will **not** want to use this!
- Important exception: if you must do fast insertions in the middle of the list.

That's it!

- There will be a problem set. Focus on two skills:
  - Deciding quickly which data structure is appropriate,
  - Using the STL versions of the data structures.