Announcements

- Program 1 due today
  - Don't forget: /readme file
  - comments
  - make sure it compiles!

- HW2 is out, due Monday the 20th

\[
\text{float var = 5.6} \\
\text{int(5.6) = 5} \\
\text{int(\text{var + .5})}
\]
Exceptions

In C++, exceptions are "thrown" by code that encounters something odd.

(Relatively new addition to C++, so standard template library doesn't always use them.)

Exceptions are often inherited, since you might have a set of similar ones...
Example: Math errors

class MathException {
    private:
        string errMsg;
    public:
        MathException(const string& err)
            & errMsg = err;
        // probably others to access message, etc ...
}

like ValueError
    TypeError
More specific exceptions:

class ZeroDivideException : public MathException {
    
public:
    ZeroDivideException (const string &err) :
        MathException (err) 
    
};

class NegativeRootException : public MathException {
    
public:
    NegativeRootException (const string &err) :
        MathException (err) 
    
};
Throwing & Catching Exceptions

```java
try {
    // ... some computations

    if (divisor == 0) {
        throw ZeroDivideException("Divide by 0 in Module X");
    }

    catch (ZeroDivideException zde) {
        // handle division by 0
    }

    catch (MathException & me) {
        // handle any others
    }
}```
What happens?

- When divisor is equal to 0, it immediately jumps to that “catch”.

- If the exception is not caught, the program just aborts.

- In previous example, if we had thrown a NegativeRootException, it would have been caught by the MathException, since that is closest matching catch.

- catch(...) ← catches all exceptions (like blank except in Python)
How do we recover?
Depends on type of exception.
- Often, print error and end program.
- May require clean-up, such as deallocating memory.
  memory leaks
- May even retry w/ different input
Exceptions in functions

When we declare a function, we should also specify what exceptions might occur.

- lets user know what to expect, so they can handle appropriately

- means we don’t have to handle exceptions - will be passed up
  (see p. 95 of text for details)
Syntax: Exceptions in functions

```java
void calculator() throw (ZeroDivideException, NegativeRootException)
{
    // function body
}
```

- Means we can throw only these 2 exceptions in `calculator` (or any child classes).
What do these mean?

```c
void funct1() // no list of exceptions
{
    //body
}
```

```c
void funct2() throw()
{
    //body
}
```

(can throw any exception)

(can't throw exceptions)
Stack: a way to store a list of data

Ex: Web browser: Store history

hit back button, goes to the last page visited

Ex: Text editors: Store previously executed commands

Undo button - only removes last action
The Stack Abstract Data Type (ADT)

Supports 2 main functions:

1. push(x)
   Insert object x at top of stack

2. pop()
   Remove top object from stack and return it

picture:

```
myStack. push(5)
push(7)
push(0)  push(11)
pop()    pop()
```

Diagrams:

```
5 7
5 7 11
5 7 11 0
```
Additional behaviors

- `size()`: Return # of objects in the stack

- `isEmpty()`: Returns true if stack is empty, false otherwise

- `top()`: Returns top object on stack without removing it
Standard Template Library

Stacks are one of the built-in class in the STL.

Functions: push, pop, top, size, & empty

Documentation is available online.

(We'll use this for lab soon...)
Notice:

I haven't said what this is made with!

Ideas?

```
Stack <int> mystack,
  push(5)
  push(7)
```

Template

```
use array

A
\[ 5 \quad 7 \quad 1 \quad \cdots \quad s-1 \]
```

```
\uparrow \uparrow \uparrow \uparrow \uparrow
\quad t - 1
\quad t
\quad t
```

```
pop = t - 2
```

Variables:
- Array pointer
- Size variable <int>
- Top variable <int>
One complication: how should we return objects?

Should pop and top be different?

- Could return value
- Or could just remove

If reference:
- Allow use to change top element of stack (without popping)
Our interface:

Always worth planning the functions ahead (in the .h file)

```cpp
template <typename Object>
class Stack {

public:
    void push (const Object& obj);
    bool isEmpty () const;
    const Object& top () const;
    Object pop ();
    int size () const;

private:
};
```
Some pseudo code:

Private:
Array A
Int s //size
Int t //top

size (): return s

isEmpty (): return (s == 0);

top (): if isEmpty ()
raise error
else
return A[t];

pop (): if isEmpty ()
raise error
else
return A[t+1];
Our code: available on webpage

Based on code from text (p. 163)
(with a few changes).

Inside constructor

s = new Object[capacity]