CS180 - Stacks

Note Title 9/8/2010

Announcements

- Program 1 due today
  - Don't forget:
    - readme file
    - comments
    - make sure it compiles!
- HW2 is out, due Monday the 20th

float var = 5.6
int(5.6) = 5
int(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 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

```
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 & 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

```
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?

```
void funct1()
{
  //body
}
```

(no list of exceptions) can throw any exception

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

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(o)

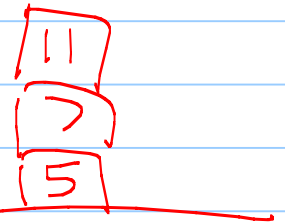
Insert object o at top of stack

2 - pop()

Remove top object from stack & return it

mystack. push(5)
push(7)
push(0)
pop()
push(11)

picture:



Additional behaviors

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

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

5 - 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

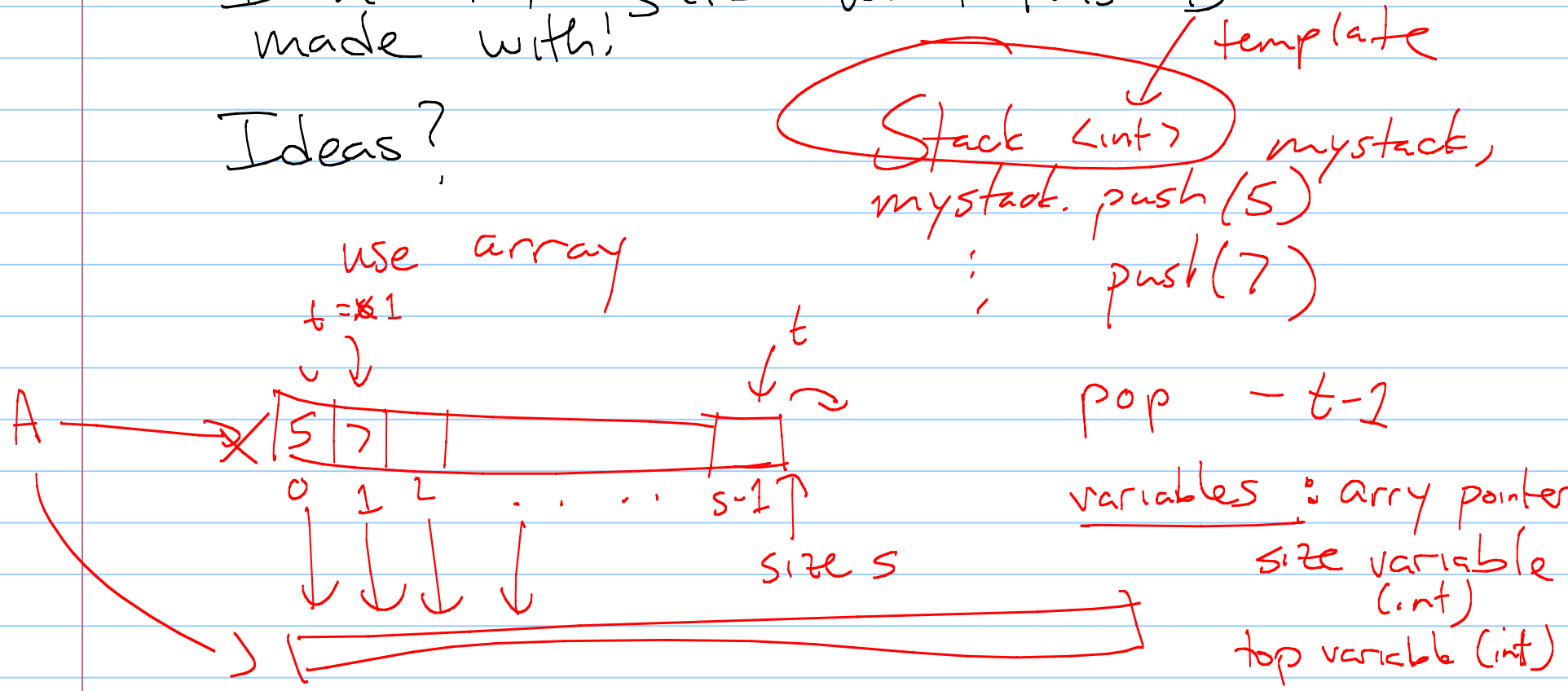
Documentation is available online.

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

Notice:

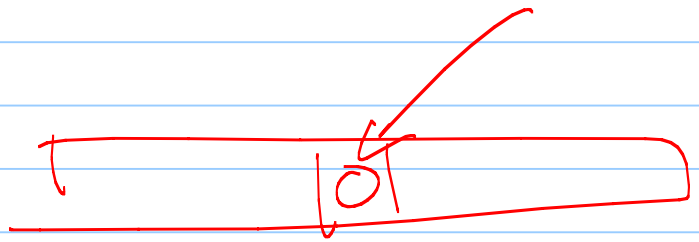
I haven't said what this is made with!

Ideas?



One complication: how should we return objects?

Should pop + top be different?



could return value or could just remove

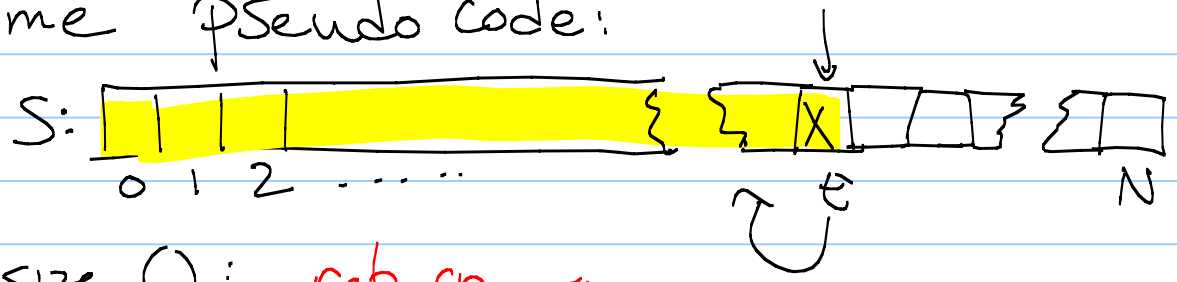
if reference, allow use to change top element of stack (without popping)

returntype myfunction ( ) {
const

Our interface:
Always worth planning the functions ahead of time...
(.h file)

```
template <typename Object>
class Stack {
    void push(const Object& obj);

    bool isEmpty() const;

    const Object& top() const;

    Object pop();

    int size() const;
};
```

Some pseudo code:




```
private:
  Array* A  //array
  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
         t--;
         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];
```