Stack ADT

(PEZ® candy dispenser)
ADTs

• An Abstract Data Type is an abstraction of a data structure: no coding is involved.

• The ADT specifies:
  - what can be stored in it
  - what operations can be done on/by it.

• There are lots of formalized and standardized ADTs (in Java).
ADTs

• For example, if we are going to model a bag of marbles as an ADT, we could specify that
  - this ADT stores marbles
  - this ADT supports putting in a marble and getting out a marble.

• In this course we are going to learn a lot of different standard ADTs. (stacks, queues, trees...)

• (A bag of marbles is not one of them.)
Stack

- A stack is a container of objects that are inserted and removed according to the last-in-first-out (LIFO) principle.
- Objects can be inserted at any time, but only the last (the most-recently inserted) object can be removed.
- Inserting an item is known as "pushing" onto the stack.
- “Popping” off the stack is synonymous with removing an item.
Stack

• A stack is an ADT that supports two main methods:
  - _push(o)_: Inserts object _o_ onto top of stack
  - _pop():_ Removes the top object of stack and returns it; _if the stack is empty then an error occurs_.

• The following support methods should also be defined:
  - _size():_ returns the number of objects in stack
  - _isEmpty():_ returns a boolean indicating if stack is _empty_.
  - _top():_ returns the top object of the stack, without removing it; _if the stack is empty then an error occurs_.

**Examples:**

- `push(3)`
- `push(6)`
- `push(4)`
- `push(1)`
- `pop()`
- `push(5)`
- `pop()`
- `pop()`
Examples:

\[3 + (4 - x) \times 7 + (y - 2 \times (2 + x)).\]
Examples:
Examples:

\[
( ( [ ) ] ) [ [ ] ] \} \} ]
\]

--- --- --- --- --- X since next symbol is ")" which doesn't match top
Examples:

Computer Science COMP 250
Introduction to Computer Science

Winter 2016, Schedule: TTh 13:05-14:25; STBIO S1/4

HW-1 due-date postponed to Feb 4th.

Description: (3 credits; 3 hours) An introduction to the design of computer algorithms, including basic data structures, analysis of algorithms, and establishing correctness of programs. Overview of topics in computer science.

Prerequisite: Familiarity with a high level programming language and CEGEP level mathematics. Computer Science Major and Honours students are advised to take MATH 240 simultaneously with COMP 250 or with COMP 251. Although it not a prerequisite either, COMP 202 will provide you a solid background for programming in Java.

Instructor: Prof. Claude Crépeau
Office Hours: Friday 12:00-15:00, McConnell 110N, (514) 398-4716
avoid sending me HomeWork questions directly; uses links below for faster replies...

FOR ALL CLASS MATTERS CONTACT US
via e-mail: cs250@cs.mcgill.ca
or Discussion board: cs250ganda.cs.mcgill.ca/ NOW WORKING; you will need a cs.mcgill.ca account to use it.
Examples:

Computer Science COMP 250
Introduction to Computer Science

Description: (3 credits; 3 hours)
An introduction to the design of computer algorithms, including basic data structures, analysis of algorithms, and establishing correctness of programs.

Overview of topics in computer science.

Prerequisite: Familiarity with a high level programming language and CEGEP level mathematics. Computer Science Major and Honours students are advised to take MATH 240 simultaneously with COMP 250 or with COMP 251. Although it not a prerequisite either, COMP 202 will provide you with a solid background for programming in Java.

Instructor:
Prof. Claude Crepeau<br><a href="http://www.cs.mcgill.ca/~crepeau">Prof. Claude Crepeau</a><br>Office Hours: Friday 12:00-15:00, McConnell 110N, (514) 398-4716

Avoid sending me HomeWork questions directly; uses links below for faster replies...

FOR ALL CLASS MATTERS CONTACT US
via e-mail:
<a href="mailto:cs250@cs.mcgill.ca">cs250@cs.mcgill.ca</a><n

NOW WORKING; you will need a cs.mcgill.ca account to use it.
Examples:

<!>  <HTML>  <HEAD>  <TITLE>  </TITLE>  </HEAD>  <BODY>  <center>  <H1>  </H1>  </center>  <HR>  <P>  <B>  </B>  <BR>  <HR>  <UL>  <P>  <A>  </A>  <BR>  <I>  </I>  <I>  <A>  </A>  </I>  <br>  . . .
Examples:

\[3 + (4 - 1) \times 7 + (6 - 2 \times (2 + 3))\]
Examples:

\[3 + (4 - 1) \times 7 + (6 - 2 \times (2 + 3))\]
3 + (4 - 1) * 7 + (6 - 2 * (2 + 3))
3 + (4 - 1) * 7 + (6 - 2 * (2 + 3))
Processing arithmetics

```plaintext
 t=gettoken()
 while type(t) ≠ eol do
   if type(t) = number then
   if type(t) = operator then
   if t = "(" then
   if t = ")" then
   t=gettoken()

 while not isemptyO() do
   op=popO()
   arg2=popA()
   arg1=popA()
   pushA(exec(arg1,op,arg2))
 return popA()
```
if type(t)=number then pushA(t)

if type(t)=operator then
  if prio(t) ≤ prio(topO())
    then op = popO()
    arg2 = popA()
    arg1 = popA()
    pushA(exec(arg1, op, arg2))
  pushO(t)
if \( t="(\) \) then pushO(t)

if \( t=")\) then
    op=popO()
    while op\(\neq\)"(" do
        arg2=popA()
        arg1=popA()
        pushA(exec(arg1,op,arg2))
        op=popO()
t=gettoken()
while type(t)≠eol do
  if type(t)=number then pushA(t)
  if type(t)=operator then
    if prio(t)≤prio(topO())
      then op=popO()
        arg2=popA()
        arg1=popA()
        pushA(exec(arg1,op,arg2))
      pushO(t)
  if t="(" then pushO(t)
  if t=")" then
    op=popO()
    while op≠"(" do
      arg2=popA()
      arg1=popA()
      pushA(exec(arg1,op,arg2))
      op=popO()
  t=gettoken()
while not isemptyO() do
  op=popO()
  arg2=popA()
  arg1=popA()
  pushA(exec(arg1,op,arg2))
return popA()
TI vs HP
Examples:

\[ 3 + (4 - 1) \times 7 + (6 - 2 \times (2 + 3)) \]