CSC148H Lecture 1

Dan Zingaro
OISE/UT

September 8, 2008

Welcome!

- Welcome to CSC148
- Comments or questions during class?
- Previous experience: programming in an object-oriented language
- Topics: stacks, queues, object-orientation, exceptions, recursion, trees, graphs, and lots of other cool stuff
- "do I seriously have to buy the textbook?"
- Evaluation: four assignments, eight labs, midterm exam, final exam, participation

Huh? I Can't Sleep in Class?

Well, you can, but:

- Participation is worth 5%
- But that's not the most important reason
- Learning is enhanced when we are actively engaged in and questioning the material
- You will have opportunities to discuss solutions to problems instead of just listening to me the whole time

Python Rampup Session

- When: Sat Sept 13 and Sat Sep 20, 10 AM - 5 PM
- Where: BA3185
- To register: `148rampup@cs.utoronto.ca` (let me know the day you prefer)

Top Ten Things that Annoy Your Instructor

- 10. Academic Offenses

Top Ten Things that Annoy Your Instructor

- 10. Academic Offenses
- 9. Disrespecting Other Students

Top Ten Things that Annoy Your Instructor

- 10. Academic Offenses
- 9. Disrespecting Other Students
- 8. Academic Offenses
- 7. Disrespecting Other Students
- 6. Academic Offenses
- 5. Disrespecting Other Students
- ...

Learning with Think-Pair-Share

- TPS: think-pair-share; we'll use this during lecture for you to discuss concepts and questions
- Think: briefly consider what I have asked you to think about
- Pair: turn to your neighbor (or neighbors) and discuss your respective ideas
- Share: one person from the chosen groups debriefs the class on the consensus reached
- Let's talk now ...

What is Computer Science

- Computer science is absolutely *not* programming
- Computer science is the study of problems, problem-solving, and the solutions that come out of the problem-solving process
- It just happens that we often use programming as a way to help us solve problems, once we have devised an algorithm

Steps in Solving a CS Problem — Specification

- Specifications must be clear and precise
- How's this: "I want a wicked-cool easy-to-use interface to this database"

Steps in Solving a CS Problem — Design

- Abstraction: ignoring certain details to make problems easier to solve
- Example: Databases (logical vs. physical)
- Use standard metaphors (abstract data types)
- *Abstract* because there is no mention of implementation
- *Data Type* because it includes data and operations on those data
- Allow us to more directly describe the data we work with in our problems

TPS: Abstract Data Types

With your neighbor, come up with abstract data types (ADTs) that you have previously learned about. What are their data and operations?

Steps in Solving a CS Problem — Analysis

- Reason about (1) efficiency and (2) correctness
- Which of these is more important?
- Any program that works is better than any program that doesn't
- Story time: parts requirements for cars based on customer-selected options (Weinberg, 1975)

Steps in Solving a CS Problem — Analysis

- Reason about (1) efficiency and (2) correctness
- Which of these is more important?
- Any program that works is better than any program that doesn't
- Story time: parts requirements for cars based on customer-selected options (Weinberg, 1975)
- "But your program doesn't work. If the program doesn't have to work, I can write one that takes one millisecond per card — and that's faster than our card reader."

Steps in Solving a CS Problem — Implementation and Testing

- Use programming constructs that allow clean implementation of abstract data types
- Modular, well-designed, easy to understand code
- Test suites: thorough, documented

Stack ADT (2.3)

- A sequence of objects
- Objects are removed in the opposite order that they are inserted
- Last in, first out (LIFO), like putting away and taking out plates
- Object last inserted is at the top

Stack Operations

- **push(o)** Add a new item to the top of the stack
- **pop()** Remove and return top item
- **peek()** Return top item
- isEmpty() Test if stack is empty
- **size()** Return number of items in stack

Stack Example

- Start with empty stack
- Push 5: $[5]$
- Push 8: $[5, 8]$
- Pop: $[5]$ (and returns 8)

Uses For A Stack

- Keep track of pages visited in a browser tab
- Keep track of function calls in a running program
- Check for balanced parentheses

Python Stack Class

```
class Stack:

    def __init__(self):
        self.items = []

    def push(self, o):
        self.items.append(o)

    def pop(self):
        return self.items.pop()

    def peek(self):
        return self.items[-1]

    def isEmpty(self):
        return self.items == []

    def size(self):
        return len(self.items)
```

Queue ADT (2.4)

- A sequence of objects.
- Objects are removed in the same order they are inserted (first in, first out, FIFO)
- **enqueue(o)** Add o to the end of the queue
- **dequeue()** Remove and return object at the front of the queue
- **front()** Return object at the front of queue
- **isEmpty()** test if queue is empty
- **size()** return number of items in queue

Python Implementation of Queue

- Almost the same as the stack class
- We can still use lists, but the front element is now at index 0 instead of index -1
- What does this mean in terms of efficiency? For example:

```
def dequeue(self):
        return self.q.pop(0)

    def front(self):
        return self.q[0]
```

Priority Queue ADT

- A sequence of objects.
- Objects are removed in order of their priority
- Like a line up in a bank where the customer with largest bank account goes to the front
- **insert(o)** Add o to the queue
- **extractMin()** Remove and return object with minimum value
- **min()** Return object with minimum value
- **isEmpty**, **size** ... Same as previously

ADT Puzzle

You're given a list of integers; you're goal is to transform the list into a new list according to the following rule: find the leftmost pair of consecutive numbers in the list whose values are x and $x + 1$, replace them by the single element whose value is $2x + 1$ and repeat the process using this new list. If no pair of integers satisfies this property, the process is complete.

- Example: list $[1, 2, 3, 4]$ is transformed first to $[3, 3, 4]$, and then to $[3, 7]$

ADT Puzzle...

- What is the problem with using the "obvious" algorithm of scanning left to right looking for the next pair of numbers satisfying the condition?
- Example: $[32, 16, 8, 4, 2, 1, 2]$
- Which ADT can we use to speed things up?
- TPS ... and we'll discuss this on the bulletin board and in the next class