
1. Quiz 2
   a. Average: 85%
   b. Return quizzes and go over solution.
   c. Rehash linked list methods:
      i. See TraversingTheLinkedList_part1-dsf.2.pdf
      ii. See TraversingTheLinkedList_part2-dsf.pdf
   d. Re-grades to Dawn and appealable to Professor Potts.

2. Homework 6
   a. Due Monday June 25 at 11:45pm with a grace period until June 26 at 7:00am
   b. Pairs that do not turn in a pair programming agreement AND post their pair to the coweb will be graded individually with an automatic 10% deduction.
   c. View your grading TAs at http://coweb.cc.gatech.edu/cs1316/713
      i. If you did not post your pair, you did not get assigned.
   d. Questions?

3. Abstract data structures: Stacks and Queues
   a. Stack \( \rightarrow \leftarrow |data| \)
      i. A list where removal and addition occur at the same end (usually the head). Frequently known a LIFO (Last-In-First-Out) structure.
      ii. A good example in real life is a stack of trays in a cafeteria.
      iii. Stack methods (assuming we insert and remove at the head)
          1. push(anObject) – Adds a new object onto the top of the stack
          2. pop() – Removes the top (head) object off the stack.
          3. peek() – Gets the top of the stack, but does not remove it from the stack.
          4. size() – Return the size(aka the length) of the stack
   b. Queue \( \leftarrow |data| \leftarrow \)
      i. Pronounced “Q,” a list where removal occurs at one end (usually the head) and addition occurs at the other end (usually the tail). Frequently known a FIFO (First-In-First-Out) structure.
      ii. A good example in real life is a line at a movie theatre.
      iii. Queue methods (assuming we insert at the end and remove from the head)
          1. enqueue(anObject) - Adds a new object at the end of the queue. Pronounced “en-q.”
          2. dequeue() – Removes an object from the head. Pronounced “de-q.”
          3. size() – Return the size(aka the length) of the stack
   c. Stacks and Queues in action
      i. Stack
         1. push
            Data: 1 2 3 4
            Original Stack: \( \rightarrow \leftarrow |9 \: 10| \)
a. push(1): no return
   →←|1 9 10|
b. push(2): no return
   →←|2 1 9 10|
c. push(3): no return
   →←|3 2 1 9 10|
d. push(4): no return
   →←|4 3 2 1 9 10|

2. pop
   Original Stack: →←|4 3 2 1 9 10|
   a. pop(): returns 4
      →←|3 2 1 9 10|
   b. pop(): returns 3
      →←|2 1 9 10|
   c. pop(): returns 2
      →←|1 9 10|
   d. pop(): returns 1
      →←|9 10|
   e. pop(): returns 9
      →←|10|
   f. pop(): returns 10
      →←| |

ii. Queue

1. enqueue
   data: 1 2 3 4
   original queue: ←|9 10|←
   a. enqueue(1): no return
      ←|9 10 1|←
   b. enqueue(2): no return
      ←|9 10 1 2|←
   c. enqueue(3): no return
      ←|9 10 1 2 3|←
   d. enqueue(4): no return
      ←|9 10 1 2 3 4|←

2. dequeue
   original queue: ←|9 10 1 2 3 4|←
   a. dequeue(): returns 9
      ←|10 1 2 3 4|←
   b. dequeue(): returns 10
      ←|1 2 3 4|←
   c. dequeue(): returns 1
      ←|2 3 4|←
d. dequeue(): returns 2
   ←| 3 4|←
e. dequeue(): returns 3
   ←| 4|←
f. dequeue(): returns 4
   ←| |←

d. Big O – a measure of efficiency commonly used in computer science
   i. Why would a stack be good to use for reverse method? Think about how stacks
      remove and how they insert new data.
   ii. Conventional reverse method: \( O(n^2) \).
       See intro-to-stack.ppt p16.
   iii. New reverse method using a stack: \( O(2n) \Rightarrow O(n) \).
       See intro-to-stack.ppt p20.
e. Abstract data type (ADT) - An abstract type is a description of the methods that a data
   structure knows and what the methods do.
f. We can actually write programs that use the abstract data type without specifying the
   implementation. For example:
   ```java
   public void push(Object element){
       elements.addFirst(element);
   }
   ```

   Because every class in Java extends Object in some way or another, instances of classes
   are always Objects (Hence Java is considered an object-oriented language). Thus this
   stack may contain any Object. Remember that int, double, etc are considered primitives
   and are not Objects, but there are ways around this limitation by using the Integer,
   Double, etc classes available in Java.