CS4092 Thursday 9th January 2014

Lecture 1 : Case Study - SoloSnap

Shuffle Deck
- Deal Cards one by one until either
  - Snap - Player Wins
  - run out of cards - Player loses.

- 3 Classes
  - Card
  - DeckOfCards
  - SoloSnap

Left - Justified Array.

Achieve accurate, guaranteed Random, then choose 10 times the number of items (eg Cards) so Rand 520.

/~Kieran/cs2504 website

CS4092 Tuesday 14th January 2014

Lecture 2 : Abstract Data Types

Abstraction : Idealised model that provides a simplified conceptual model that cloaks a more complex underlying reality.

eg. DVD player : plays movies, supports behaviours -> play, pause, stop, eject etc. End user version, dont have to know about how it does it, just how to use it.

The Stack :- Push, Pop, LIFO
eg cafeteria trays ≡ c b a -> c, b, a

E/t Type = refers to the type of the items (ints, strings etc)

stk.push(num); /* add to stack */
stk.pop(); /* remove from stack */

CS4092 Thursday 16th January 2014

ADT Queue → FIFO → disciplined ~~queue~~ queue

2 ended → added at back → removed at front.

△ enqueue (o): insert item o at rear of queue

△ dequeue (): remove and return item at front of queue
illegal if queue is empty.

size ()

isEmpty ()

front (): return but do not remove

Java Example

```
{
    QueueOfInt q = new QueueOfInt (); int num;
    for (num = 1; num <= 6; num ++)
    {
        SOP* - ("Enqueuing" + num);
        q . enqueue (num);
    }
    while (! q . isEmpty ())
    {
        num = q. dequeue ();
        SOP* - ("Dequeuing" + num);
    }
}
```

(SOP* - = System.out.println)

Psudo Code

- Setting up the circle (numbered players)
  for i ← 1 to n do
      circle. enqueue(i)

- Passing the ball clockwise once
  circle. enqueue (circle. dequeue ())

- Remove the victim (ball holder)
  circle. dequeue ()

- Determining the last (only) man standing
  circle. front()

for(int i = 1, i <= circleSize ; i++)
for(i ← 1 to n do) } same

Look out for clean solutions instead of most obvious!

CS4092 Tuesday 21st January 2014

Floor ⌊ ⌋
Ceiling ⌈ ⌉
Brackets [ ]
Braces { }
Parentheses ( )

} Mathematical Expression

Example.

$$\overbrace{[\underbrace{(5+x)}_{B} - \underbrace{(y+z)}_{C}]}^{A}$$

△ Many "syntax checking" algorithms (eg programming language parsers) are stack based

○ Unbalanced
1. Mismatch
2. Close paren is encountered and the stack is empty
3. The stack is non-empty at end.
Otherwise it is balanced!

Expression	Stack
([( )])	(
([( )])	([
([( )])	([(
([( )])	([
([( )])	(
([( )])	

OK!

parenStack

<u>Postfix</u>

1 + 2 Standard "Infix"

1 2 + Operator after value / subexpression - Postfix

(STACK Example)

1 + 2	1 2 +	[1][2]
1 + 2 + 3	1 2 + 3 +	3 + [1][2][3] → +
1 * 2 + 3	1 2 * 3 + 4 +	[1][5]
(1 + 2) * 3	1 2 + 3 * (first)	4 - [1][5][4] → -
		[1][1]

1, 2, 3 etc operand

"-", "+", "*", "/" etc operator

The stuff on the stack are intermediate results.
They stay on the stack until we need them.

Token - constituent part. eg operands or/and operators

ExprScanner

- hasNext()
- next()
- currTkn (number or 4 operators)
  -> if (+) elseif (-) elseif (*) elseif (/) else. operator

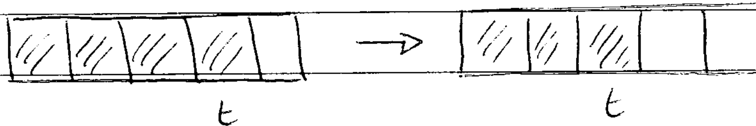
  ↓ Stack PUSH, ↓ Stack PUSH, ↓ Stack PUSH, ↓ Stack PUSH

If operator Pop 2 items from stack, secondOpnd
firstOpnd.

CS4092 Thursday 23rd January 2014

size()	return t+1	S = Stack
isEmpty()	return (t < 0)	t = top
top()	return S[t]	

pop()




```
pop(),
e ← S[t]          /* Whatever is on top of the stack, put in e */
t ← t - 1         /* Decrement t by 1 */
return e          /* Return the item stored in e */

push(o),          /* o is the item we are adding */
t ← t + 1         /* Increment t by 1 */
S[t] ← o          */ Put item in new Stack top */
```

```
public int pop()
{
    int elt = theElts[top];
    top--;
    return elt;
}
```

```
public void push(int element)
{
    top++;
    theElts[top] = element
}
```

```
// Other methods
public static final int ~~Capacity~~ CAPACITY = 100;
private int capacity;
private int theElts[];
private ~~top~~ top = -1;
```

Issues

- No error checking
- Size-limited (CAPACITY = 100)
- Int-Specific - what if ^(you wanted) it contained STRING?

→ Error Checking

```
public int pop ()
{
    if (this.isEmpty ())
    {
        sop (" empty ");
        System.exit (1);
    }
    int elt = theElts [top];
    top--;
    return elt;
}
```

SOP = SYSTEM.OUT.PRINT()

A better way is to use exceptions i.e. throws exception.

→ Size - limited

solution - create a new array, copy all items from original array into it and continue using the new array. So CAPACITY = 400

CAPACITY_2 = 200.

Then we simply point S to the new array.

```
if (this.size() == capacity)     <= Array is full.
{
  New Array
   int temp[] = new int [2 * capacity];

   for (int i = 0 ; i < capacity ; i++)
   {
        temp [i] = theElts [i];
   }

   Setting theElts to new array

    theElts = temp;
```

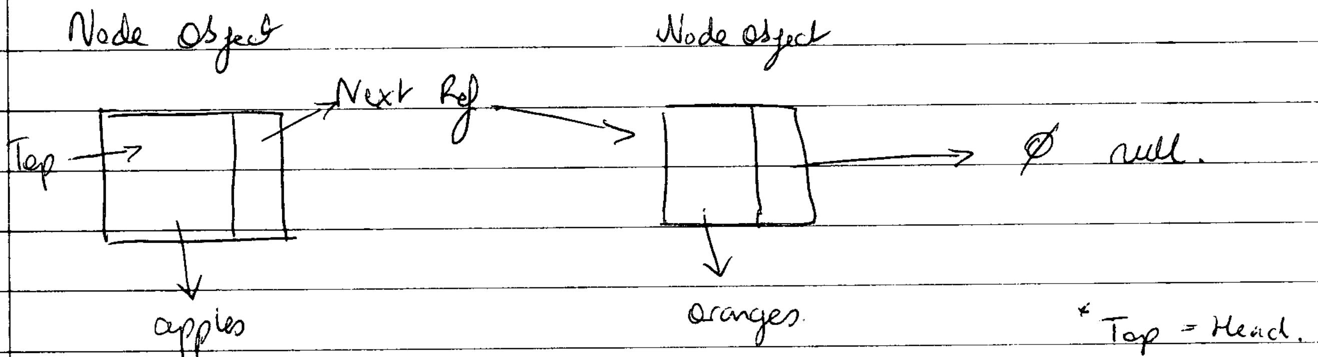
Point to
the new array!

```
   capacity = 2 * capacity
  }
  top ++;
  theElts [top] = element;
}
```

Doubling the size of the array has a far cheaper cost than just adding a new item to the stack, i.e. stack capacity + 1 (repeat, repeat, repeat ...)

CS4092 Thursday 13th February 2014

Linked Lists & ADT's Stack and Queue.



The nodes linked together give us a node list container
Manipulation is possible ie add to list, remove from list.

Must have a separate int to manage list leght. (size)

<u>Push</u>
Node is added to the top of the stack / list.

<u>Pop</u>
Node is removed from the top of the stack / list.

<u>Pg 7/22</u>

top = null

↓

private LLNode <EltType> top;

↓

" " " next;

These two together set up the list.

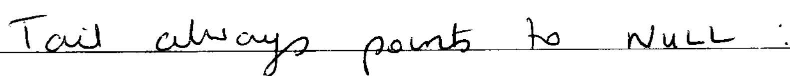
~~add to~~ next setter + getter

When some node is popped from the container, it gets garbage collected at some stage.

ADT Queue

A queue can be manipulated from either the front or rear (head or tail) so we need to keep track of both the head and tail in our program ie. FIFO / LIFO.

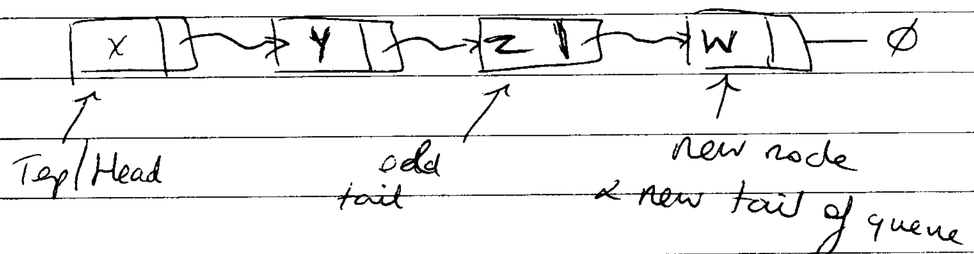
Tail always points to NULL :



Head always points to the next node in the queue :

Enqueue

As a queue works on the principle of adding items to the end of a queue (FIFO)



Dequeue

Remove the front item (top) so this is easier than enqueue.

Implementation

```
head = null;
tail = null;
size = 0;
```

node.setNext(null) – because this new node is the last node it should be set to null (ie reference null as it points to no other node.)

CS4092 Thursday 20th February 2014

Lecture 12: Reasoning About Recursive Algorithms

△ Proposition
for all $n \geq 1$
sum(n) returns the sum of the first n natural numbers

△ Proposition
for all $n \geq 1$, fib(n) returns f_n (the nth Fibonacci number)

○ IH = Inductive Hypothesis
By IH fib(n') returns $f_{n'}$ for all $n' < n$
For example consider fib(n): fib(n-2) returns f_{n-2}

Algorithm Computation