- Ces implémentations sont-elles wait free ?

Supposons qu'un objet g de la classe *Ginc* soit initialisé à 0 et que plusieurs threads appellent une seule fois *ginc*() sur cet objet.

5. Combien parmi eux retourneront la valeur 1?

6. En supposant qu'il n'y a que deux threads, quelles sont les valeurs qu'ils obtiendront de l'objet dans un appel de *ginc*?

On rappelle qu'un objet *n-consensus* est un objet qui a un état interne S initialisé à $\perp$ et une méthode *propose* avec pour argument une valeur v et dont la spécification séquentielle est la suivante

$\{S = \perp\}$ *propose*(v) $\{S = v\}$ return S
$\{S \neq \perp\}$ *propose*(v) $\{\}$ return S

7. Peut-on réaliser un wait-free 2-consensus avec des objets *Ginc* et des registes (donner un algorithme ou faire une preuve d'impossibilité)

8. Peut-on réaliser un wait-free 3-consensus avec des objets *Ginc* et des registres (donner un algorithme ou faire une preuve d'impossibilité)

9. Peut-on avoir une implémentation wait-free d'objets de *Ginc* en n'utilisant que des registres?

Exercice 3.— (6 points). On veut réaliser une implémentation linéarisable d'une pile d'entiers partagée entre n threads ($n \geq 2$) *int pop*() retire l'élément au sommet de la pile et retourne la valeur de celui-ci; *push*(*int* v) ajoute l'élément v à la pile.

1.
```
public class Node {
     public int value;
     public Node next;
     public Node(int value) {
         this.value=value;
         next=null;
        }
     }
  public class PileUn {
     private Node top;
     public boolean compareAndSet(Node old, Node n)
       { if (top==old){top=n; return true;}
        else return false;
     }
  public void push(int value) {
      Node node = new Node(value);
      Node oldTop = top;
      node.next =  oldTop;
```

ANNEXE

```
public class AtomicReference<V>
extends Object
implements Serializable
```

An object reference that may be updated atomically. See the `java.util.concurrent.atomic` package specification for description of the properties of atomic variables.

Since:

1.5

See Also:

Serialized Form

Constructor Summary

Constructors

Constructor and Description
AtomicReference() Creates a new AtomicReference with null initial value.
AtomicReference(V initialValue) Creates a new AtomicReference with the given initial value.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
V	**accumulateAndGet**(V x, BinaryOperator<V> accumulatorFunction) Atomically updates the current value with the results of applying the given function to the current and given values, returning the updated value.
boolean	**compareAndSet**(V expect, V update) Atomically sets the value to the given updated value if the current value == the expected value.
V	**get**() Gets the current value.
V	**getAndAccumulate**(V x, BinaryOperator<V> accumulatorFunction) Atomically updates the current value with the results of applying the given function to the current and given values, returning the previous value.
V	**getAndSet**(V newValue) Atomically sets to the given value and returns the old value.
V	**getAndUpdate**(UnaryOperator<V> updateFunction) Atomically updates the current value with the results of applying the given function, returning the previous value.
void	**lazySet**(V newValue) Eventually sets to the given value.
void	**set**(V newValue) Sets to the given value.
String	**toString**() Returns the String representation of the current value.
V	**updateAndGet**(UnaryOperator<V> updateFunction) Atomically updates the current value with the results of applying the given function, returning the updated value.
boolean	**weakCompareAndSet**(V expect, V update) Atomically sets the value to the given updated value if the current value == the expected value.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait