Compulsory assignment 1

INF4140, Fall term 2012

Final delivery date

Friday September 28 at 18.00.

How to deliver

- Your solution should be delivered online (https://devilry.ifi.uio.no/).
- Program examples should be commented in order to make them understandable for the group teacher or lecturer.

Who delivers

Everyone is required to deliver a solution. You may work alone or together with one other student. (Notice that you are not allowed to work together with more than one student.) In this case the two of you should deliver the same solution. The solution must be marked with name and email address of the contributing students.

Read the departmental guidelines for written assignments before you start.

Evaluation

This assignment is graded *pass* or *fail.*
You must pass this assignment in order to take the final exam.

Linked list

A queue is often represented using a linked list. Assume that two variables, `head` and `tail`, point to the first and last elements of the queue. A null link is represented by the constant `null`.

Each element in the queue contains a data field (`val`) and a link to the next element of the queue (`next`). Each element in the queue is thereby represented by two variables in the program. For convenience, we use dot-notation and write `el.val` and `el.next` for the variables representing the values `val` and `next` for the queue element `el`. In the initial state, `head` == `tail` == `null`.
The routine `find(d)` finds the first element of the queue containing the data value `d`.

```
find ( d ) {
   i=head; # The variable i is local to the current method instance
   while (i!=null and i.val != d) i=i.next;
}
```

The routine `insert (new)` inserts a new element at the end of the queue. Both the `head` and `tail` pointer must be updated if the queue is empty. (Assume that `new.next == null`.)

```
insert( new ) {
   if (tail == null) {  # empty list
      head = new;
   } else {
      tail.next = new;
   }
   tail = new;
}
```

The routine `delfront` deletes the first element of the queue (pointed to by `head`). The variable `tail` must be updated if the queue contains only one element.

```
delfront {
   if (head != null) {               # the list is not empty
      if (head == tail) tail = null;  # only one element in the list
      head = head.next;
   }
}
```

Do the following:

1. Identify the $\mathcal{V}$ and $\mathcal{W}$ sets (see lecture slides) of the shared variales in the three routines. You may use dot-notation in the variable representation for `val` and `next`. For example: variable `tail.next` is in the $\mathcal{W}$ set of the routine `insert`.

2. Now assume that several processes access the linked list. Consider all six different combinations of two and two routines. Which combinations of routines can be executed in parallel without interference? Which combinations of routines must be executed one at a time? Remember to consider the parallel execution of each routine against itself.

Hint: Remember that two *disjoint* routines A and B do not interfere with each other. If A and B are not disjoint, we may use the "At-Most-Once Property" to decide if A and B interfere. When interpreted on routines, "At-Most-Once Property" may be formulated as follows: Consider all possible results of executing A and B sequentially (i.e. A; B and B; A). The routines A and B interfere with each other if parallel execution of A and B may lead to any *new results* not possible from a sequential execution.

3. Add `await` statements to program the synchronization code in the routines. Such that non-interference of concurrent executions is enforced. Try to make your atomic actions as small as possible, and do not delay a routine unnecessarily.

   Hint: Syntax of the `await` statements: <await (B);> or <await (B) S;>
   Hint: It might be helpful to use more than one locks in order to rule out the interfering combinations of executions.