TDT4205 Problem Set 5, Spring 2012

Part 1 - Theory due Wed Mar. 21^{th}, 20:00.
Part 2 - Programming due Wed Mar. 28^{th}, 20:00.

Part 1 needs to be passed in order for Part 2 to be evaluated!
ALL answers are to be submitted to *itslearning*

ALL OF THIS ASSIGNMENT IS TO BE DONE INDIVIDUALLY.
Cheating ("koking"), including using partial solutions from other students, will cause a failing grade for the whole course. We will be checking for this using a plagerism detecting as well as looking for suspiciously similar submissions.

All submitted source code MUST be able to compile and run on asti. **This assignment is PASS/FAIL.** Please read the assignment guidelines on itslearning before starting to work on the assignment. Requests for clarifications can be posted on the itslearning forum.

What to turn in
When turning in assignments, please turn in these files:

- (your_username)_answers.pdf : Answers to non-programming questions (Part 1)
- (your_username)_code.zip,tar.gz,tgz : All your code for this assignment, including makefiles and other necessary code (Part 2)

Part 1 - Theory

Task 1.1 - Data flow analysis

The code for bubble sort is given as

```
for(int x=0; x<n; x++)
{
    for(int y=0; y<n-1; y++)
    {
        if(array[y]>array[y+1])
        {
            int temp = array[y+1];
            array[y+1] = array[y];
            array[y] = temp;
        }
    }
}
```

Assume that the variables *array* and n has been defined earlier, and indices are valid from 0 to n-1.

1. Write equivalent three-address code for this code fragment. Note: You do NOT have to take offsets into account when indexing arrays. In other words, you can write array lookups as $array[t]$, not $array[4*t]$
2. Using the three-address code, draw the flow graph.
3. Perform the following optimizations on the flow graph, and draw the resulting flow graph:
   - Global and local common subexpression elimination
   - Copy propagation
   - Constant folding (if any)
   - Dead code elimination

   Write down every change you make.

Task 1.2

1. What are dominators and what are they used for in the context of this course?
2. What is the difference between forward and backwards analysis with respect to the topic of compiler optimization?