CS2205: Theory of Computation
Problem Class II

1. A *vowel* is one of the letters $\{a, e, i, o, u\}$.

   For each of the following *problem specifications*, how many *problem instances* are there?

   (a)

   Problem II.1.
   Parameters: *A vowel, v, and another vowel, w*
   Returns: *YES if v equals w, NO otherwise*

   (b)

   Problem II.2.
   Parameters: *A vowel, v, and a pair of vowels, $\langle w_1, w_2 \rangle$*
   Returns: *YES if v is equal to one or other of the members of the pair, NO otherwise*

   (c)

   Problem II.3.
   Parameters: *A vowel, v, and a finite-length list of vowels, L*
   Returns: *YES if v is a member of list L, NO otherwise*

   (d)

   Problem II.4.
   Parameters: *A vowel, v*
   Returns: *YES if v is a vowel, NO otherwise*

   (e)

   Problem II.5.
   Parameters:
   Returns: *YES if 'a' is a vowel, NO otherwise*

2. Which of the problems specified in question 1 are *decision problems*?

3. Rewrite the following expressions, inserting additional parentheses to make explicit the way that the operator precedence and associativity conventions given in the lecture notes would affect the evaluation order.

   (a) $x + 4 \times 5$
   (b) $x \times 4 + 5$
   (c) $x + 4 + 5$
   (d) $x \operatorname{div} 4 \operatorname{div} 5$
   (e) $x + 3 = 6 - 1$
   (f) $\neg p \wedge q$
   (g) $\neg(p \wedge q)$
   (h) $0 \leq x \leq 10$
   (i) $y = 3 \vee 4$

4. Use your answers to the last two parts of question 3 to answer the following.

   (a) A student is writing a DECAFF algorithm. If the value of x lies between 0 and 10 inclusive, then 1 is added to the contents of x. S/he writes:

   ```
   if (0 ≤ x ≤ 10)
   {    x := x + 1;
   }
   ```

   What's wrong with this? (Note: The answer I'm after has nothing to do with the fact that in Java you'd write `<=` rather than DECAFF's $\leq$.)

   (b) What should s/he write?

   (c) Later s/he needs to increment y but only if it contains 3 or 4. S/he writes:

   ```
   if (y = 3 ∨ 4)
   {    y := y + 1;
   }
   ```

   What's wrong with this? (Note: The answer I'm after has nothing to do with the fact that in Java you'd write `||` rather than DECAFF's $\vee$.)

   (d) What should s/he write?

5. In Darjeeling (which is a language that I've just made up), the only operators are those whose precedence and associativity conventions are shown below.

Precedence	
$+, \mathrm{mod}$	highest
div	
$\times$	
$-$	
$=$	lowest

Associate to the left	Associate to the right
$+, \mathrm{mod}, \times, =$	div, $-$

Say whether the following expressions evaluate to true or false according to these bizarre conventions, i.e. does the left hand side evaluate to the same number as the right hand side?

(a) $3 + 4 \times 5 = 3 + (4 \times 5)$

(b) $5 - 4 - 3 - 2 = 2$

(c) $10 \,\mathrm{div}(12 \,\mathrm{div}\, 3) \,\mathrm{div}\, 2 = 5 \,\mathrm{mod}\, 1 \,\mathrm{mod}\, 3 + 1 \times 5$

6. In Assam (another language that I've just invented), the only operators are those whose precedence and associativity conventions are shown below.

Precedence	
mod	highest
$\times$, div	
$+, -$	lowest

Associate to the left	Associate to the right
$\times$, div, $+$	mod, $-$

Explain using an example why this set of conventions is problematic.

7. Let B be a Boolean expression and C a command.

Suppose a programming language provides only one form of unbounded iteration, a **do** ... **until** loop, as follows:

```
do
    C;
until B;
```

do ... **until** loops are post-test, exit-when-**true**-loops.

Show how you would reduce the following kinds of unbounded loop to a combination of sequence, conditionals and **do** ... **until** loops.

(a) Post-test, exit-when-**false**

(b) Pre-test, exit-when-**true**

(c) Pre-test, exit-when-**false**

8. In the lectures, we discussed pre-test and post-test loops. Sometimes, however, we want an exit test in the middle of the body of the loop! Such a loop is sometimes called a *loop-and-a-half*.

Here's an example:

Algorithm: LINEARSEARCH($x, a, lower, upper$)

Parameters: x is an integer; $a[lower \ldots upper]$ is an array of integers; $0 < lower \leq upper$.

Returns: The position of the first occurrence of x in a if found, otherwise **fail**.

```
{   for i := lower upto upper
    {   if a[i] = x
        {   return i;
        }
    }
    return fail;
}
```

This has the potential to jump out of the middle of the loop (using the **return** command). (General remark. In the above algorithm, **return** is used to jump out of the middle of the loop. Many languages, including Java, supply extra commands especially for jumping out of the middle of loops. In the case of Java, C and C++, this command is the *break* command. You may want to read up about this command in a textbook. However, as the rest of this exercise demonstrates, it is possible to completely avoid jumping out of the middle of loops.)

Show how you would reduce the above algorithm to one that does not jump out of the middle of the loop. (Hint: you may wish to use a **while** loop in place of the **for** loop).

9. It is very easy to make what are sometimes called *off-by-one* errors, where a loop body is executed one time too few or one time too many. This question tests your ability to avoid off-by-one-errors. Use reasoning, not guesswork!

In the TOC-Bank, your deposit earns annual interest. At the end of each year, the interest is credited to your account.

For example, suppose at the start of the first year your account contains €1000 and the rate of interest is 5%. At the end of the first year, you have earned 5% of 1000, i.e. €50. So in the second year, your account starts with $1000 + 50 = 1050$ and earns 5% of €1050, i.e. €52. In the third year, it starts with $1050 + 52 = 1102$ and earns 5% of 1102, i.e. €55. And so on.

Here's an (incomplete) algorithm for solving this problem:

Parameters: *balance*, a positive integer denoting the amount in your account at the start of year 1; *rate*, a positive integer denoting the rate of interest.
Returns: The minimum number of years it takes before the amount in your account has at least doubled.

```
{   targetBalance := balance × 2;
    years := Ⓐ;
    while balance Ⓑ targetBalance
        balance := balance + (balance × rate) div 100;
        years := years + 1;
    }
    return years;
}
```

(a) What should be put in place of Ⓐ? Should it be 0 or 1?

(b) What should be put in place of Ⓑ? Should it be $<$ or $\leq$?

10. Variable x stores a positive integer which denotes the strength on the Richter scale of a recent earthquake.

The following algorithms test the value of x and, depending on its value, return a textual description of the severity of the earthquake.

Algorithm: A()

```
if x < 5
    return "Earthquake? What earthquake?";
if x < 6
    return "I think we've lost the cow shed again, mother.";
if x < 7
    return "I think we've lost Conakilty again, mother.";
return "All is lost. Where's mother?"
```


Algorithm: B()

```
if x < 5
    return "Earthquake? What earthquake?";
if x < 6
    return "I think we've lost the cow shed again, mother.";
if x < 7
    return "I think we've lost Conakilty again, mother.";
else
    return "All is lost. Where's mother?"
```

Algorithm: C()

```
if x > 7
    return "All is lost. Where's mother?"
else if x > 6
    return "I think we've lost Conakilty again, mother.";
else if x > 5
    return "I think we've lost the cow shed again, mother.";
else
    return "Earthquake? What earthquake?";
```

Algorithm: D()

```
if x < 5
    return "Earthquake? What earthquake?";
else if x < 6
    return "I think we've lost the cow shed again, mother.";
else if x < 7
    return "I think we've lost Conakilty again, mother.";
return "All is lost. Where's mother?"
```

Algorithm: E()

$s :=$ "";
if $x < 5$
  $s :=$ "Earthquake? What earthquake?";
else if $x < 6$
  $s :=$ "I think we've lost the cow shed again, mother.";
else if $x < 7$
  $s :=$ "I think we've lost Conakilty again, mother.";
else
  $s :=$ "All is lost. Where's mother?"
return s;

Algorithm: F()

$s :=$ "";
if $x < 5$
  $s :=$ "Earthquake? What earthquake?";
if $x < 6$
  $s :=$ "I think we've lost the cow shed again, mother.";
if $x < 7$
  $s :=$ "I think we've lost Conakilty again, mother.";
if $x > 7$
  $s :=$ "All is lost. Where's mother?"
return s;

Algorithm: G()

$s :=$ "";
if $x < 5$
  $s :=$ "Earthquake? What earthquake?";
if $x \geq 5 \wedge x < 6$
  $s :=$ "I think we've lost the cow shed again, mother.";
if $x \geq 6 \wedge x < 7$
  $s :=$ "I think we've lost Conakilty again, mother.";
if $x \geq 7$
  $s :=$ "All is lost. Where's mother?"
return s;

Which of the algorithms B-G are equivalent to algorithm A (in the sense of producing the same result as algorithm A)?

11. Consider the following (deliberately poorly-indented) algorithm:

if $x \bmod 4 = 0$ { **if** $x \bmod 8 = 0$ **return** 1; **else return** 2; } **return** 3;

What value will be returned by this algorithm...

(a) ...when x contains 12?
(b) ...when x contains 16?
(c) ...when x contains 10?

12. Repeat question 11 for the following algorithm:

if $x \bmod 4 = 0$ { **if** $x \bmod 8 = 0$ **return** 1; } **else return** 2; **return** 3;

13. Repeat question 11 for the following algorithm:

if $x \bmod 4 = 0$ **if** $x \bmod 8 = 0$ **return** 1; **else return** 2; **return** 3;

14. Consider execution of the left-hand DECAFF algorithm whose parameter, n, is an integer, $n \geq 0$. In DECAFF, the left-hand algorithm is equivalent to the right-hand one, and this may help you to answer this question.

for $i := 1$ **upto** n
{ C;
}

$i := 1$; A
while $i \leq n$ B
{ C; C
  $i := i + 1$; D
}

(a) How many times is i initialised (i.e. how many times is A executed)?
(b) How many times is i tested (i.e. how many times is B executed)?
(c) How many times is the body, C, executed (i.e. how many times is C executed)?
(d) How many times is i incremented (i.e. how many times is D executed)?