UPPSALA UNIVERSITET

Department of Information Technology

INSTRUCTIONS

Please check that you have the correct exam!

This sheet should always be submitted, even if you haven't solved any of the exam problems.

~~Each solution should be written on a separate paper.~~

Write your exam code on each paper.

Please use only *one side* of the papers and do not use a pencil with red colour.

Sort the solutions in question order, with question 1 first, before you submit them.

FRONT SHEET FOR EXAMS	DATE:
Course name	
Your exam code	
Semester when you were first registered to the course:	Programme:
Time for submitting the exam:	Table number:

					–					–			

No.	Solved problems (mark with X)	Points earned
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		
21		
22		
23		
24		
25		
26		
27		
28		
29		
30		

Comments from the teacher

$\sum$

☐ Exam with bonus points: Grade is not shown.[1]

5 $\geq$ ☐ 4 $\geq$ ☐ 3 $\geq$ ☐

Exam grade:

1 The final result (points including bonus points and grade) can be found in Ladok after certification.

Final Exam (Part 1)

Program Design and Data Structures (1DL201)

Teachers: Eva Darulova and Johannes Borgström

2024-01-11 / 14:00 - 18:00

Instructions

Read and follow these instructions carefully to increase your chance of getting good marks.

- This is a closed-book exam. You may use a standard English dictionary. Otherwise, **no notes, calculators, mobile phones, or other electronic devices are allowed**.
- Write your **exam code** at the top right corner of **each page**.
- There are **16 questions**. Each question awards a number of points, as listed in the question heading. You can obtain a maximum of **40 points**.
- Some questions ask for a justification of your answer; your justification needs to relate the information given in the question, the course content, and your answer.
- Some questions ask you to write code; this code needs to conform to the coding convention. You only need to write function documentation if the question text states so.
- Write your answers directly on the question sheet. You may use English and Swedish.
- Eva and/or Johannes will come to the main exam hall around 15:00 to answer questions (and a bit later to other exam locations).

Question 1: Substitution model (2 points)

What is the result of the following program?

```
function add(x, y) {
  return x + y;
}
const input = 3;
add(input, 3 * input);
```

Answer:

Show all evaluation steps using the **substitution model**:

Question 2: Recursive functions (3 points)

Write a recursive function that calculates the sum of the digits in the base-10 representation of a non-negative integer (tvärsumman).

Hint: Recall that the `%` operator computes the remainder when its left argument is divided by the argument on the right.

Answer:

```
// @example digit_sum(347) === 14
function digit_sum(n) {

}
```

Does your function generate an inductive process or not? Why?

Answer:

Question 3: Orders of growth (2 points)

Which of the following four statements are true?

1. $n \in \mathrm{O}(n^2)$
2. $n^2 \in \Omega(n \log n)$
3. $2^n \in \Theta(n\, 2^n)$
4. $n \in \mathrm{O}(2^n)$

Answer:

Question 4: Solving recurrences (3 points)

Recall the Master theorem (for divide-and-conquer recurrences):
Assume that $T(n) = a\,T(n/b) + f(n)$ for some $a > 0$, $b > 1$, and non-negative f. We let $E = \frac{\log a}{\log b}$.

1. If there is ε such that $f(n) \in \mathrm{O}\left(n^{E-\varepsilon}\right)$ then $T(n) \in \Theta\left(n^E\right)$.
2. If there is k such that $f(n) \in \Theta\left(n^E\,(\log n)^k\right)$ then $T(n) \in \Theta\left(n^E\,(\log n)^{k+1}\right)$.
3. If there is ε such that $f(n) \in \Omega\left(n^{E+\varepsilon}\right)$ and f satisfies the regularity condition (there is $c > a$ and n_0 such that for all $n > n_0$, $f(n) \geq c\, f(n/b)$) then $T(n) \in \Theta(f(n))$.

Use this theorem to solve the recurrence $T(n) = 8\,T(n/8) + n^2$. Show all steps of your solution.

Answer:

Question 5: Higher-order functions (3 points)

We aim to automate the grading system for the autumn homework module. There are three homeworks and each is graded on a scale of 0 to 100 points. A homework is passed if its points are 50 or above. A grade of "G" is given for the whole module if all three homeworks are passed and the average score is from 50 to 74, and a grade "VG" for an average score 75 and above. Write two **lambda expressions** `is_passing` and `average_grade` that can be used as follows:

```
function grade_student(pass_func, avrg_func, score1, score2, score3) {
    if(pass_func(score1) && pass_func(score2) && pass_func(score3)) {
        return avrg_func(score1, score2, score3);
    } else {
        return "U";
    }
}
grade_student(is_passing, average_grade, 75, 45, 95); // returns "U"
grade_student(is_passing, average_grade, 50, 75, 65); // returns "G"
grade_student(is_passing, average_grade, 80, 75, 95); // returns "VG"
```

Answer:

```
const is_passing =

const average_grade =
```

Question 6: List principle (2 points)

What is printed by the two calls to `display_list` in the following program?

```
const ls = list(3, 7, 5);
display_list(tail(tail(ls)));
display_list(pair(tail(ls), pair(9, null)));
```

Write down the answer using **list notation**, i.e. use the notation `list(...)` for any structure that satisfies the list principle (i.e. is a list as defined in the lecture).

Answer:

Question 7: Function specifications (3 points)

Write a **full** function specification for the following function, including **at least two examples** that explain the behavior of the function:

```
function lcp(list1, list2) {
    return is_null(list1) || is_null(list2)
               ? null
           : head(list1) !== head(list2)
               ? null
           : pair(head(list1), lcp(tail(list1), tail(list2)));
}
```

Answer:

Question 8: Variants (2 points)

Give a variant for the function `lcp` from the previous Question.

Answer:

Justify your answer:

Question 9: List processing abstractions (3 points)

The function `average_above` takes as input a list of numbers `lst` and an integer `threshold`:

```
function average_above(lst, threshold) {
    return accumulate((acc, x) => acc + x,
                      0,
                      filter(x => x >= threshold, lst));
}
```

What is the result of the following code?

```
const lst = list(2, 5, 7, 1, 3, 0, 3, 7, 3);
average_above(lst, 5);
```

Answer:

Rewrite the function `average_above` using the higher-order abstraction functions `accumulate` and `map`, but **without** `filter`.

Answer:

```
function average_above(lst, threshold) {

}
```

Question 10: Memoization (2 points)

Is the function `average_above` from the previous Question a good candidate for memoization? **Explain your answer.**

Answer:

Question 11: Trees (3 points)

Recall the binary tree interface with the functions `make_empty_tree()`, `is_empty_tree(tree)`, `make_tree(value, left, right)`, `right_branch(tree)`, `left_branch(tree)`, `entry(tree)`.
Write a function `min_value` that takes as input a **binary search tree** constructed using the binary tree interface, and returns the smallest value in the tree.
You should assume that the input tree is non-empty, and all values in the tree are integers.
Your function should be as efficient as possible and **avoid any unnecessary computation**.
Answer:

```
function min_value(tree) {









}
```

Question 12: Environment model (3 points)

What are the values of `a` and `b` at the end of the following code?

```
let a = 3;
let b = 5;

function foo(b, c) {
    a = b;
    b = c;
}
foo(4, 7);

// values of a and b here?
```

Answer:

Explain your answer **using the environment model**:

Question 13: Arrays (2 points)

What is the contents of the array `arr` at the end of this code?

```
const arr = [[1, 2, 3],
             [3, 5, 2],
             [3, 6, 2]];

function add(A) {
    for(let i = 0; i < array_length(A); i = i + 1) {
        let x = 0;
        let j = 0;
        for(j = 0; j < array_length(A[i]); j = j + 1) {
            x = x + A[i][j];
        }
        A[i][j] = x;
    }
}

add(arr);
// contents of arr here?
```

Answer:

Question 14: Sorting (2 points)

Recall the **Selection Sort** sorting algorithm for sorting values in increasing order and its **in-place** implementation using arrays. Show the contents of the array as it changes during the run of Selection Sort after each step of the algorithm, i.e. after each step of the outer loop of the algorithm.

Answer:

14	5	12	7	9	(input array)
					(array after last step)

Question 15: Streams (3 points)

Write a function `blank_every_third` that takes as input an **infinite stream** without side-effects and returns a stream that inserts a '0' into every third position. For example, recall the infinite streams of 1's (`ones`) and integers (`integers`). Applying `blank_every_third` to them will return the following streams:

```
// 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, ...
blank_every_third(ones);

// 1, 2, 0, 3, 4, 0, 5, 6, 0, 7, 8, 0, 9 ...
blank_every_third(integers);
```

Your implementation does not have to be particularly optimized.

Answer:

```
function blank_every_third(stream) {

}
```

Question 16: Development methodology (2 points)

In this question we will extend a course registration system for a university. As in Homework 6, students (modelled as `string`s) can apply to take various courses. Each course has a fixed number of slots, and if the course is full there may also be students on a waitlist ordered by their application time. You can assume, if needed, that the accessor functions `name(course)`, `credits(course)`, `admitted(course)`, `waitlist(course)` exist.

One problem is that students who are already admitted to courses may remain on the waitlists of other courses. Therefore, we are investigating how to write a function that given a student and a list of courses, checks if they are already admitted to courses adding up to the maximum number of credits (45), and if so, removes the student from all waitlists.

Begin implementing this function using **bluffing** as one of the first steps. You do not need to develop your solution further than to complete the bluffing step.

This is the final page.