# Försättsblad till skriftlig tentamen vid Linköpings universitet

| | |
|---|---|
| **Datum för tentamen** | 2017-05-31 |
| **Sal (2)** | TER2(16) TERE(1) |
| **Tid** | 8-12 |
| **Kurskod** | TDDA69 |
| **Provkod** | TENA |
| **Kursnamn/benämning** **Provnamn/benämning** | Data- och programstrukturer Tentamen |
| **Institution** | IDA |
| **Antal uppgifter som ingår i tentamen** | 5 |
| **Jour/Kursansvarig** Ange vem som besöker salen | Cyrille Berger |
| **Telefon under skrivtiden** | 076-777 28 70 |
| **Besöker salen ca klockan** | ca kl. 09:00 |
| **Kursadministratör/kontaktperson** (namn + tfnr + mailaddress) | Anna Grabska Eklund, ankn. 2362, anna.grabska.eklund@liu.se |
| **Tillåtna hjälpmedel** | inga |
| **Övrigt** | |
| **Antal exemplar i påsen** | |

# Försättsblad till skriftlig tentamen vid Linköpings universitet

| | |
|---|---|
| **Datum för tentamen** | 2017-05-31 |
| **Sal (2)** | TER2(16) TERE(1) |
| **Tid** | 8-12 |
| **Kurskod** | TDDA69 |
| **Provkod** | TENA |
| **Kursnamn/benämning** **Provnamn/benämning** | Data- och programstrukturer Tentamen |
| **Institution** | IDA |
| **Antal uppgifter som ingår i tentamen** | 5 |
| **Jour/Kursansvarig** Ange vem som besöker salen | Cyrille Berger |
| **Telefon under skrivtiden** | 076-777 28 70 |
| **Besöker salen ca klockan** | ca kl. 09:00 |
| **Kursadministratör/kontaktperson** (namn + tfnr + mailaddress) | Anna Grabska Eklund, ankn. 2362, anna.grabska.eklund@liu.se |
| **Tillåtna hjälpmedel** | inga |
| **Övrigt** | |
| **Antal exemplar i påsen** | |

**TDDA69**
**Spring 2017**
**2017-05-31 8-12**
**Time Limit: 4 hours**

Examiner: Cyrille Berger
Tel: 076-777 28 70

This exam contains 3 pages (including this cover page) and 5 questions.
Total of points is 38p, the minimum for passing the exam is 19p, to get a four it is 25p and to get a five it is 31p.

**No assistance.**

**Good luck!**

1. (10 points) Programming paradigms.
   (a) (3 points) Draw a diagram showing the relation between the following programming paradigm:
   - First-order functional programming
   - Functional programming
   - Logic programming
   - Imperative programming
   - Sequential object-oriented programming

   The relation between those programming paradigms could be (not all of them are necesserary usefull, and some might appear several times in the diagram):
   - +procedure
   - +closure
   - +cell(state)
   - +unification
   - +thread
   - +search
   - +port

   (b) (1 point) What is a pure function? Tell if pure and non-pure functions can be used in functional or imperative programming.

   (c) (3 points) For the following applications, select which programming paradigm (between functional or imperative) you would choose to use, and give an explanation of your choices:
   1. querying a database
   2. distributed numerical computations
   3. game

2. (5 points) Recursion:
   (a) (2 points) Write a recursive function *has_digit(k, d)* that test if a number $k$ contains the digit $d$ at least once.

   Examples of use:

```
>>> has_digit(2147, 7)
True
>>> has_digit(2149, 7)
False
>>> has_digit(2747, 4)
True
>>> has_digit(123393, 4)
False
```

(b) (3 points) What is tail recursion? Give an example of a tail-recursive procedure, and another of a recursive but not-tail recursive procedure. What can be gained by treating tail recursion as a special case (and do tail call optimization)?

3. (14 points) Environment model.

   (a) (2 points) Expressions can be evaluated in *normal* or *applicative* orders. Explain both orders. Do they always give the same result? If not, give an example that differentiates them.

   (b) (2 points) What are the problems with the *substitution* model and how is it solved by the *environment* model?

   (c) (1 point) Assume the expression below is evaluated in the order it is given.

```
function f(x)
{
  var y = g(2)(-1,1);
  g = h(g)
  return g(x+y)(4, 5);
}
function g(x)
{
  return function(y,z) { return z + (y * x); }
}
function h(f)
{
  return function(x) { return f(x+1); }
}
f(3)
```

   What will the result be?

   (d) (3 points) Draw a diagram that captures what is going on according to the environment model of evaluation.

   (e) (2 points) Mark the important structures and explain why, and in what order, they are created and (can be) removed.

   (f) (2 points) Use the diagram to show the result of the evaluation.

   (g) (2 points) Does *f(3)* always return the same value? Explain your answer.

4. (4 points) Macros.

   (a) (2 points) What are macros?

(b) (2 points) What are the benefits and inconvenients of macros?

5. (5 points) Stack machines.

In this question, we use a stack machine with the following instruction set:

- *PUSH [constant_value]*: push the constant on the stack
- *POP [number]*: pop a certain numbers of variables from the stack
- *MUL*: pop two arguments from the stack, push the result of multiplying them
- *SUB*: pop two arguments from the stack, push the result of subtracting them
- *EQUAL*: pop two arguments from the stack, push true if they are equal, or false otherwise
- *LOAD [varname]*: push the value of variable
- *DCL [varname]*: declare the variable
- *STORE [varname]*: get the value, store the result and push the value
- *JMP [idx]*: jump to execute instruction at the given index
- *IFJMP [idx]*: pop the value and if true jump to [idx]
- *CALL [arguments]*: pop the function object and call it with the given number of arguments
- *RET*: return from a function call

(a) (2 points) Given the following factorial function:

```
var factorial = function(n)
{
  if(n == 0) {
    return 1
  } else {
    return n * factorial(n - 1);
  }
}
```

Write the list of instructions that would define the factorial function on a stack machine with the provided instruction set.

Write the list of instructions that would call the factorial function.

For clarity, you should provide a number for each instruction in your answer, as shown in the following example:

1. LOAD 'k'
2. PUSH '5'
3. MUL
4. JMP 1

(b) (1 point) Explain what happen during a *CALL* instruction and how the *RET* instruction knows where to return.

(c) (2 points) What is the maximum depth of the stack for a call to `function(5)` ? List all the values in the stack.