Institutionen för Datavetenskap. Lunds Universitet. Dat501: FP, VT03

Prov på delkurs Funktionsprogrammering

Tentamen 21. mars 2003

Tillåtna hjälpmedel: Litteratur, egna anteckningar

Tentamen består av 3 uppgifter, varje uppgift kan ge 10 poäng. Deluppgifternas poäng anges inom [hakparenteser]. Observera att poänggivningen inte nödvändigtvis avspeglar deluppgiftens svårhetsgrad. Uppgifterna kan besvaras på svenska eller engelska.

Behandla högst en uppgift per papper (det går bra att behandla deluppgifter på samma papper). Skriv bara på ena sidan och markera varje sida med dina initialer. Skriv läsligt.

- a)[0] Hur många uppgifter får man behandla per papper?
- b)[0] Får man skriva på baksidan?

Det går bra att referera till kursböckerna av Schmidt, Hansen och Rischel eller Rosen. Referenser måste ha formen [S: *något*], [HR: *något*] eller [R: *något*], där *något* är ett avsnittsnummer, en kodbit, en räknad uppgift, etc. Skriv till exempel "genom att använda metoden **validate** från [S: Figure 7 i Chapter 6]". Vill du referera till något annat, måste du ange bokens titel, författare, och förlag. Skriv till exempel "genom att använda metoden *validate* från [Barry Cornelius: Understanding Java, Addison–Wesley 2001, p. 254]".

Ibland hänger vissa uppgifter ihop. Observera att det går bra att referera till svaret på tidigare uppgifter, även om du inte har löst dem.

Exercise 1

On the planet Voralon VII, each year consists of 12 months of 30 days each. A date is given by a triple (*day*, *month*, *year*).

- a)[1] Define a type `voralonDate` for Voralon dates.
- b)[1] Define a reasonable invariant

  ```
  valid:  voralonDate -> bool
  ```

  that accepts only legal Voralon dates. For example, we want the value of the expression `valid(12,4,543)` to be true, but `valid(31,0,~543)` to be false.

  (Let's agree that the first year of the modern Voralon calendar was year 1. Don't worry about enormous years, dates far in the future are perfectly acceptable.)

  In the rest of the exercise you can assume that the given dates satisfy the invariant.
- c)[2] Declare an SML function

  ```
  before:  voralonDate * voralonDate -> bool
  ```

  that checks if one date precedes (*kommer före*) another. For instance, `before((12,3,54), (6,2,55))` is true.
- d)[2] Declare an SML function

  ```
  tomorrow:  voralonDate -> voralonDate
  ```

  that computes the day after a given date.
- e)[2] Declare an SML function

  ```
  add:  voralonDate * int -> voralonDate
  ```

  such that `add(`d`,`i`)` is the date that "comes i days after" date d. For example, `add((29,3,245), 3)` is `(2,4,245)`. Your implementation doesn't have to be efficient (fast). You may assume that $i \geq 0$.
- f)[2] Devise a set of tests for the function in your previous exercise. Your tests have to be few but complete.

Exercise 2

This exercise studies polynomials with integer coefficients, i.e., functions like $x^3 + 2$ or $2x^4 + 10x^3 + x^2 + 4x - 5$.

We will use the type `int list` for this, wo that we represent the polynomial $a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$ by the list $[a_0, a_1, a_2, \cdots, a_n]$. For instance, $x^3 + 2$ is represented by $[2, 0, 0, 1]$, and $2x^4 + 10x^3 + x^2 + 4x - 5$ is represented by $[-5, 4, 1, 10, 2]$.

► a)[2] Declare an SML function

```
intMult:  int list * int -> int list
```

for multiplying a polynomial with an integer. For instance, the product of the polynomial $x^3 + 2$ and the integer 7 is the polynomial $7x^3 + 14$.

► b)[2] Declare an SML function

```
xMult:  int list -> int list
```

for multiplying a polynomial with x. For instance, the product of the polynomial $x^3 + 2$ and x is the polynomial $x^4 + 2x$.

► c)[3] Declare an SML function

```
polyAdd:  int list * int list -> int list
```

for adding two polynomials. For instance, the sum of the polynomials $x^3 + 2$ and $2x^4 + 10x^3 + x^2 + 4x - 5$ is the polynomial $2x^4 + 11x^3 + x^2 + 4x - 3$.

► d)[3] Declare an SML function

```
polyMult:  int list * int list -> int list
```

for multiplying two polynomials. For instance, the product of the polynomials $x^3 + 2$ and $2x^2 + x$ is $2x^5 + x^4 + 4x^2 + 2x$ (you can check that by hand if you want to).

A useful formula for the product of two polynomials is the following. If $Q(x)$ is a polynomial then

$$0 \cdot Q(x) = 0$$

$$(a_0 + a_1x + \cdots + a_nx_n) \cdot Q(x) = a_0 \cdot Q(x) + x \cdot (a_1 + a_2x + \cdots + a_nx_{n-1}) \cdot Q(x)$$

Exercise 3

This exercise studies genealogical trees (*släktträd*). A *person* has a name and zero or more children, who are themselves persons.

```
datatype person = Childless of string
         | Parent of string * person list
```

► a)[1] Create an value corresponding to the person 'Finwë', who has two children 'Finarfin' and 'Fingolfin'. Neither of the children have children of their own.

► b)[2] Declare an SML function

```
numGrandchildren:  person -> int
```

that returns the number of a given person's grandchildren.

The next two questions (and only those) consider the following function:

```
fun f (s, Parent (t,x::xs)) = s=x orelse f(s,Parent(t,xs))
|   f _ = false;
```

► c)[1] What is the type of `f`?

► d)[1] Explain what `f` computes. (Unless you are sure that you are exceptionally good at explaining stuff, you probably want to give an example).

A person is *lonely* if he has exactly one child.[1]

► e)[2] Write a method

```
hasLonelySuccessor:  person-> bool
```

so that `hasLonelySuccessor(`p`)` is true if and only if there is a lonely person among p's successors (*efterkommande*).

► f)[3] Write a method

```
makeHappy:  person->person
```

that adds a new child (whose name is 'TV') to *every* lonely person among a person's successors.

[1] Persons with no children at all are assumed to be very young or have full-time jobs, and are not considered lonely in this exercise