MATEMATISKA INSTITUTIONEN
STOCKHOLMS UNIVERSITET
Avd. Beräkningsmatematik
Kursledare och examinator: Anders Mörtberg

Tentamensskrivning i
DA4003 Programmeringsparadigm
7.5 hp
2022-03-14

- **Del 1** består av 8 flervalsfrågor där minst ett svarsalternativ är korrekt. Om man svarar fel eller inte har exakt rätt antal alternativ får man 0 poäng på frågan.
- **Del 2** består av ett antal frågor med varierande antal poäng vilka ska lösas genom att man skriver kod i de olika programmeringsspråken i kursen.
- **Skriv tydligt.** Svårlästa svar riskerar 0 poäng.
- Inga externa bibliotek får användas om det inte står explicit i uppgiften.
- Skriv bara på en sida av varje papper.
- För att få godkänt måste man ha minst 4 poäng på Del 1, har man inte det rättas inte Del 2.
- **Hjälpmedel:** Ett A4 med så mycket information du vill. Du får skriva på båda sidorna.
- **Betygsgränser:** E: 15, D: 18, C: 21, B: 24, A: 27, av maximala 30.

Del 1: flervalsfrågor (1p per fråga, 8p totalt)

Var snäll och samla svaren på del A på ett svarspapper. Varje fråga på del 1 är värd 1 poäng.

1. Vad är en ”metod” inom objektorienterad programmering?

A. En funktion som inte tillhör någon klass.
B. Ett speciellt sätt att skriva objektorienterad kod på.
C. En variabel som tillhör en specifik instans av klassen
D. En funktion som är definierad i en klass.
E. Det finns inget som heter ”metod” inom objektorientering.

2. Vad blir resultatet av `f [1,2,3]` givet koden till höger?

A. `[1,1,2,2,3,3]`
B. `[1,2,3]`
C. `[1,1,2,2,2,2,3,3,3,3,3,3]`
D. `[1,1,1,1,1,1]`
E. `[]`

```
f :: [Int] -> [Int]
f [] = []
f (x:xs) = x : x : f xs
```

3. Vilka av följande påståenden stämmer för JavaScript?

A. JavaScript utgör en av grundpelarna inom webbprogrammering.
B. Det finns pekare i JavaScript.
C. JavaScript är en dialekt av Java.
D. JavaScript måste kompileras för att kunna köras.
E. Variabler i JavaScript kan byta typ efter att man deklararerat dem.

4. Vilka av följande uttryck motsvarar en fas i en typisk implementation av ett programmeringsspråk?

A. Sorter
B. Lexer
C. Classifier
D. Reader
E. Parser

5. Hur många lösningar kommer Prolog generera för ett anrop till `p(X,Y)` givet koden nedan till höger?

A. 1
B. 2
C. 3
D. 4
E. 5

```
p(X,Y):- q(X), !, r(Y).

q(1).
q(2).
q(3).

r(4).
r(5).
```

6. Vad skrivs ut av koden till höger?

A. `%p`
B. `1`
C. `2`
D. Minnesaddressen där `1` i arrayen `x` har sparats.
E. Minnesaddressen där `2` i arrayen `x` har sparats.

```
int x[5] = {1,2,3,4,5};

int *ptr = &x[1];

printf("%p", ptr);
```

7. Vad är typen på Haskellfunktionen `f g xs = map (fst . g) xs` ?

A. `(a -> b) -> [a] -> [b]`
B. `[(a,b)] -> [a] -> [b]`
C. `(a -> (b, c)) -> [a] -> [b]`
D. `(a -> (b, c)) -> [a] -> [c]`
E. `(a -> (b, c)) -> a -> [b]`

8. Vilka av följande termer hänger främst ihop med Prolog?

A. Attribut
B. Händelser
C. Unifikation
D. Backtracking
E. Typklasser

Del 2: kodfrågor (12p totalt)

Var snäll använd ett papper till varje uppgift i del 2.

9. **Imperativ programmering i C**

Vi kan definiera en matris med värden och en tom array för att spara summan av varje rad i matrisen genom:

```
int arr[3][4] = {{4,2,10,9},{0,8,1,3},{5,5,9,1}};
int sums[3];
```

Betrakta nu följande kodsnutt som beräknar summorna av raderna i `arr` och sparar i `sums`:

```
for (int i = 0; i < 3; i++) {
  int s = 0;

  for (int j = 0; j < 4; j++)
    s += arr[i][j];

  sums[i] = s;
}
```

Uppgiften är nu att skriva om kodsnutten enligt instruktionerna nedan (så man ska inte ändra deklarationen av `arr` och `sums`, utan man ska bara ändra kodsnutten med `for` loopar). Deluppgifterna (a)–(c) är oberoende av varandra, men bör skrivas på samma papper.

(a) Skriv om koden så att den använder sig av `while` loopar istället för `for` loopar. (1p)

(b) Skriv om koden så att den använder sig av `goto` istället för `for` loopar (så `for`, `while` och `do-while` är inte tillåtet). (2p)

(c) Skriv om koden i `for` looparna med pekararitmetik istället för array notation (så man får använda `for` looparna, men inte `arr[i][j]` och `sums[i]` för att komma åt och skriva till arrayerna). (2p)

Obs: för poäng på deluppgifterna måste man skriva om `for` looparna på lämpligt sätt så att `sums` har samma innehåll som om man hade kört koden ovan. Man får inte hårdkoda värdet på `sums` utan det ska sättas enligt instruktionen i varje deluppgift.

10. Objektorienterad programmering i Java

På en gammal tenta skulle man skriva en klass för tärningar. Ni ska nu implementera en variant av detta där man istället sparar nuvarande värde på tärningen som en attribut. För att göra detta ska ni skriva en klass `Dice` för att representera 6-sidiga tärningar. Lösningarna kan skrivas på samma papper och den funktionalitet som ni ska implementera är:

(a) Klassen ska ha ett privat instansattribut `face_value` av typ `int` och en konstruktor som låter användaren sätta detta till ett tal mellan 1 och 6. Om talet man försöker sätta `face_value` till inte råkar vara i detta intervall ska `face_value` sättas till 1. (2p)

(b) En metod `int get_face_value()` som returnerar värdet på tärningen. (1p)

(c) En metod `void roll_die()` som rullar tärningen och uppdaterar värdet på `face_value`. (2p)

Ni får använda er av funktioner från `java.util.Random` för att generera slumpmässiga tal till (c). För att få ett tal `num` mellan 0 och 5 skriver man:

```
Random rand = new Random();
int num = rand.nextInt(6);
```

11. Funktionell programmering i Haskell

(a) Tribonacci talen är en talsekvens som påminner på Fibonacci talen, men istället för att börja med 0 och 1 och sen addera de två tidigare talen i sekvensen så börjar man med 0, 0 och 1, och sen adderar man de *tre* tidigare talen för att få nästa i sekvensen. De första 10 talen i sekvens är alltså:

$$0, 0, 1, 1, 2, 4, 7, 13, 24, 44, \ldots$$

Skriv en funktion `tribonacci :: Int -> Int` som returnerar det n:e Tribonacci talet räknat från `0`.
Exempelanvändning: (3p)

```
> map tribonacci [0..9]
[0,0,1,1,2,4,7,13,24,44]
```

(b) Skriv en högre ordningens funktion `take_while :: (a -> Bool) -> [a] -> [a]` som tar in en funktion `p` som returnerar `True` eller `False`, samt en list `xs`. Funktionen ska sedan returnera de första elementen i `xs` som gör att testfunktionen `p` blir `True`. Så fort testfunktionen `p` blir `False` för något element i `xs` ska alltså funktionen avslutas och inga fler värden returneras. (3p)
Exempelanvändning:

```
> take_while (\x -> x > 3) ([5,8,4,3,10,20])
[5,8,4]
> take_while (\x -> x > 3) ([2,5,8,4,3,10,20])
[]
```

Obs: för poäng får man inte använda de inbyggda Haskell funktionerna `takeWhile` och `dropWhile`.

12. Logikprogrammering i Prolog

(a) Ett binärt träd med tal kan representeras i Prolog som antingen ett löv `leaf(X)` där `X` är ett tal eller en förgrening `branch(X,L,R)` där `X` är ett tal och `L` och `R` är underträd. Två exempel på binära träd på detta format är:

```
branch(2, leaf(4), leaf(8)).
branch(2, branch(3, leaf(4), leaf(8)), leaf(10))
```

Skriv ett predikat `sum_tree(T,N)` där `T` är ett binärt träd och `N` är summan av alla tal i trädet. (3p)

Exempelkörning:

```
?- sum_tree(branch(2, leaf(4), leaf(8)),N).
N = 14.

?- sum_tree(branch(2, branch(3, leaf(4), leaf(8)), leaf(10)),N).
N = 27.
```

(b) Skriv predikatet `swap_tree(T1,T2)` där `T1` är ett binärt träd och `T2` är ett binärt träd där *alla* vänster och höger delträd till `branch` delarna i `T1` har bytt plats. (3p)

Exempelkörning:

```
?- swap_tree(branch(2, leaf(4), leaf(8)),T2).
T2 = branch(2, leaf(8), leaf(4)).

?- swap_tree(branch(2, branch(3, leaf(4), leaf(8)), leaf(10)),T2).
T2 = branch(2, leaf(10), branch(3, leaf(8), leaf(4))).
```

Obs: för full poäng måste *alla* höger och vänster delträd bytt plats (se det andra exemplet ovan).