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

Tentamensskrivning i
DA4003 Programmeringsparadigm
7.5 hp
2025-08-29

- **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 1 på ett svarspapper.

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

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

2. Vilka av följande uttryck motsvarar en fas i en typisk implementation av ett statiskt typat programmeringsspråk?

A. Organizer
B. Lexer
C. Typechecker
D. Optimizer
E. Parser

3. Vad skrivs ut av koden till höger?

```
int *p = malloc(3 * sizeof(int));

for (int i = 0; i < 3; i++)
    *(p + i) = i * sizeof(int);

p += 2;

printf("%d", *p);
```

A. `0`
B. `1`
C. `2`
D. `4`
E. `8`

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

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

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

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

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

5. Vad är typen på Haskellfunktionen `foo` nedan?

```
foo f g xs = map (f . g . f) xs
```

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

6. Vad blir värdet på `YS` om man anropar `f([1,2,3],YS)` givet koden nedan?

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

```
f(XS,YS) :-
    append(ZS,[_|WS],XS),
    !,
    append(ZS,WS,YS).
```

7. Vad blir resultatet av `f [(1,2),(-3,4),(5,6)]` givet koden nedan?

A. `1`
B. `3`
C. `4`
D. `10`
E. `15`

```
f [] = 0
f ((x,_):xs) = x + g xs

g [] = 0
g ((_,x):xs) = x + f xs
```

8. Hur skapar man en funktion `foo` som tar ett heltal `x` och ett flyttal `y` och returnerar en sträng i JavaScript?

A. `def foo(x,y):`
B. `function foo(x,y)`
C. `function foo(var x, var y)`
D. `string function foo(int x,float y)`
E. `string foo(int x,float y)`

Del 2: kodfrågor (22p totalt)

Var snäll använd ett papper till varje uppgift i del 2. Lösningarna på deluppgifterna för varje programmeringsspråk kan skrivas på samma papper.

9. **Imperativ programmering i C**

Ett tal är *perfekt* om det är samma som summan av dess delare. Om vi tar 6 som exempel så är delarna 1, 2, och 3. Summerar vi dessa får vi $1 + 2 + 3 = 6$. Tar vi 8 istället så är delarna 1, 2 och 4, vilket ger $1 + 2 + 4 = 7 \neq 8$. Vi har alltså att 6 är perfekt, men 8 är inte det. På den här uppgiften ska ni skriva funktioner för att testa om tal är perfekta eller inte.

(a) Börja med att skriva en funktion med typ `int* divisors(int n)` som returnerar en pekare till början av en minnessekvens med delarna till `n`. För enkelhets skull ska ni allokera `n` tal i minnet och när det är slut på delare kan ni fylla på med nollor (man får använda `malloc` om man vill). Givet `6` ska alltså `divisors` returnera en pekare till en minnessekvens med `1,2,3,0,0,0`. För poäng måste man använda sig av `for`-loopar på den här uppgiften. (2p)

(b) Skriv om `divisors` till en funktion `int* divisors_goto(int n)` som bara använder sig av `goto` istället för `for`-loopar. (2p)

(c) Skriv en funktion `int is_perfect(int n)` som använder sig av `divisors` för att testa om `n` är perfekt eller inte. Är talet perfekt ska `1` returneras och `0` annars. (1p)

Exempelanvändning:

Om man kör följande kodsnutt:

```
int* p1 = divisors(6);
for (int i = 0; i < 6; i++)
    printf("%d ",*(p1 + i));

printf("\n");

int* p2 = divisors_goto(6);
for (int i = 0; i < 6; i++)
    printf("%d ",*(p2 + i));

printf("\n%d\n%d\n",is_perfect(6),is_perfect(8));
```

så ska resultatet vara

```
1 2 3 0 0 0
1 2 3 0 0 0
1
0
```

10. Objektorienterad programmering i Java

Här ska ni skriva klasser för att representera (enkelt länkade) listor som innehåller `Integers`. Man ska kunna ta längden av listor med hjälp av en metod `length` och en lista är antingen `Cons` av ett heltal och en till lista, eller `Empty`, dvs tomma listan.

(a) Börja med att skriva en *abstrakt* klass `List` med en abstrakt metod `length`. (2p)

(b) Lägg till en klass `Empty` som ärver `List` och har en konstruktor som inte tar några argument. Implementera även `length` metoden för `Empty` listor. (1p)

(c) Lägg till en klass `Cons` som ärver `List` och vilken har en konstruktor som tar en `Integer` och en lista och sätter lämpliga instansattribut. Implementera även `length` metoden för `Cons` listor. (2p)

Exempelkörning: om man testar klasserna med koden

```
class uppgift10 {

  public static void main(String[] args) {
    // Tomma listan []
    List e = new Empty();

    // Listan [1,4,5]
    List l = new Cons(1,new Cons(4,new Cons(5,new Empty())));

    System.out.println(e.length());
    System.out.println(l.length());
  }
}
```

så ska utskriften bli:

```
0
3
```

11. Funktionell programmering i Haskell

(a) Skriv en funktion `foo :: [Int] -> [Int]` som tar in en lista med tal och multiplicerar med `5` och sen adderar `3` till varje tal. Man får använda `map` om man vill. (1p)

(b) Skriv en funktion `prefix :: String -> String -> Bool` som returnerar `True` om första strängen är ett prefix till den andra strängen. (2p)

Exempelanvändning:

```
> prefix "hej" "hej"
True
> prefix "hej" "hejhopp"
True
> prefix "hej" "nej"
False
```

(c) Skriv en datatyp `Sign` som antingen är `Negative`, `Zero` eller `Positive`. Skriv även en funktion `sign` av typ `Int -> Sign` som returnerar om talet är negativt, noll, eller positivt. (3p)

Exempelkörning:

```
> sign (-2)
Negative
> sign 0
Zero
> sign 4
Positive
```

12. Logikprogrammering i Prolog

(a) Vi kan använda konstanterna `cons(X,XS)` och `empty` i Prolog för att representera listor precis som vi gjorde i Java ovan. Skriv en relation `length_list(L,N)` så att `N` är längden på `cons`/`empty`-listan `L`. (2p)

Exempelanvändning:

```
?- length_list(empty,N).
N = 0.
?- length_list(cons(1,cons(4,cons(5,empty))),N).
N = 3.
```

(b) Skriv `append_list(XS,YS,ZS)` för denna typ av listor så att `ZS` är `XS` följt av `YS`. (2p)

Exempelanvändning:

```
?- append_list(empty,cons(1,empty),ZS).
ZS = cons(1, empty).
?- append_list(cons(1,empty),empty,ZS).
ZS = cons(1, empty).
?- append_list(cons(1,empty),cons(2,cons(3,empty)),ZS).
ZS = cons(1, cons(2, cons(3, empty))).
```

(c) Skriv `reverse_list(XS,YS)` för denna typ av listor så att `YS` är `XS` baklänges. (2p)

Tips: man får använda sig av `append_list` ovan även om man inte löst den uppgiften.

Exempelanvändning:

```
?- reverse_list(cons(1,cons(4,cons(5,empty))),YS).
YS = cons(5, cons(4, cons(1, empty))).
```