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

Tentamensskrivning i
DA4003 Programmeringsparadigm
7.5 hp
2024-03-08

- **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. Vilket/Vilka påståenden nedan är sanna?

A. Variabler i JavaScript kan byta typ efter att man deklararerat dem.
B. Java är ett dynamiskt typat språk.
C. Det finns pekare i Java.
D. Typfel hittas vid kompilering för statiskt typade språk.
E. En polymorf funktion är en funktion som fungerar för argument med godtyckliga typer.

2. Vad blir resultatet av om man kör följande Prologkod: `2 is 1 + 1.`

A. `true.`
B. `false.`
C. `X = 1 + 1, Y = 2.`
D. `1 + 1 = 2.`
E. Det finns inget som heter ”`is`” i Prolog

3. Vad skrivs ut av C-koden till höger?

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

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

p += 1;

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

A. `0`
B. `1`
C. `2`
D. `3`
E. `4`

4. Vad är typen på den anonyma Haskellfunktionen `\f g xs -> filter (f . g) xs` ?

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

5. Vad gäller för Javakoden till höger?

```
class Volvo extends Car {
    private String owner;

    public Volvo(String o) {
        super("Volvo");
        owner = o;
    }
    public String get_owner() {
        return owner;
    }
}
```

A. Klassen `Car` ärver från `Volvo`
B. Klassen `Volvo` ärver från `Car`
C. Konstruktorn i `Car` tar in en sträng som argument
D. `owner` är en instansvariabel
E. `owner` är en klassvariabel

6. Hur många möjliga lösningar hittas när man kör `f([1,2,3],YS)` givet Prologkoden nedan?

A. 0
B. 1
C. 2
D. 3
E. 4

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

7. Vad blir resultatet av `foo (*2) [1,2,3,4]` givet Haskellkoden nedan?

A. `[1,2,3,4]`
B. `[2,4,6,8]`
C. `[1,4,12,16]`
D. `[1,4,12,24]`
E. `[1,4,12,32]`

```
foo :: (a -> a) -> [a] -> [a]
foo _ [] = []
foo f (x:xs) = x : foo f (map f xs)
```

8. På vilket/vilka sätt kan man skapa en variabel `x` med värdet `42` i JavaScript?

A. `int x = 42`
B. `Int x = 42`
C. `var x = 42`
D. `variable x = 42`
E. `x := 42`

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**

Fibonacci-sekvensen är en talsekvens som börjar med 0 och 1, och nästa tal i sekvensen fås sedan genom att addera de två tidigare talen i sekvensen. De första 10 talen i sekvens är alltså:

0 1 1 2 3 5 8 13 21 34 ...

Den här uppgiften går ut på att skriva olika funktioner i C som beräknar sekvensen.

(a) Skriv en *rekursiv* funktion `int fib_rec(int n)` som returnerar det `n`:e Fibonacci-talet (räknat från noll, så `fib_rec(0)` ska vara `0`, `fib_rec(1)` ska vara `1`, etc). (1p)

(b) En nackdel med `fib_rec` är att den snabbt blir långsam. Skriv en funktion `int fib_while(int n)` som returnerar det `n`:e Fibonacci-talet (räknat från noll) med hjälp av en `while`-loop istället. Lösningen måste vara skriven med en `while`-loop, så varken rekursion eller `for`-loopar ger poäng. (2p)

(c) Istället för att beräkna ett specifikt tal i sevensen kan det vara användbart att beräkna de `n` första talen i sekvensen. Skriv en funktion `int* fib_numbers(int n)` som returnerar en pekare till början av ett minnesblock där de `n` första talen i sekvensen är sparade. På den här uppgiften får man använda `malloc` om man vill. (2p)

Exempelanvändning:

Om man kör följande kodsnutt:

```
int main() {
  for (int i = 0; i < 10; i++)
    printf ("%d ", fib_rec(i));
  printf("\n");

  for (int i = 0; i < 10; i++)
    printf ("%d ", fib_while(i));
  printf("\n");

  int* f = fib_numbers(10);
  for (int i = 0; i < 10; i++)
    printf ("%d ", *(f + i));
}
```

så ska resultatet vara

```
0 1 1 2 3 5 8 13 21 34
0 1 1 2 3 5 8 13 21 34
0 1 1 2 3 5 8 13 21 34
```

10. Objektorienterad programmering i Java

Den här uppgiften går ut på att representera rationella tal, dvs bråktal, med hjälp av klasser i Java.

(a) Skriv en klass `Rational` med två publika instansattribut av typ `Integer` vilka heter `numerator` och `denominator` (täljaren respektive nämnare). Konstruktorn ska låta användaren sätta båda dessa, men om nämnaren försöks sättas till noll så ska även täljare sättas till noll (matematiskt är detta så klart inte korrekt, men det kan vara praktiskt att låta `n/0` vara `0/0` för alla `n`). (2p)

(b) Lägg till en `toString()` metod som returnerar en sträng `n/d` där `n` är täljare och `d` nämnare. (1p)

(c) Skriv en klass `NormalizedRational` som ärver från `Rational`. Konstruktorn ska först anropa konstruktorn för `Rational` och sen ska talet normaliseras. Om varken täljare eller nämnare är 0 ska båda delas med deras största gemensamma delare (SGD). Här får ni anta att det redan finns en funktion med signatur `Integer gcd(Integer x,Integer y)` vilken returnerar SGD av `x` och `y`. För att undvika division med noll så ska $0/0$ ej ändras och $0/n$ för $n \neq 0$ sättas till $0/1$. (2p)

Exempelkörning: om man testar klassen med koden

```
Rational r1 = new Rational(2,3);
Rational r2 = new Rational(2,0);
System.out.println(r1);
System.out.println(r2);

NormalizedRational n = new NormalizedRational(12,36);
System.out.println(n);
NormalizedRational n00 = new NormalizedRational(0,0);
System.out.println(n00);
NormalizedRational n05 = new NormalizedRational(0,5);
System.out.println(n05);
```

så ska utskriften bli:

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

11. Funktionell programmering i Haskell

(a) Skriv `mapOn :: (a -> a) -> (a -> Bool) -> [a] -> [a]` som fungerar så att `mapOn f p xs` applicerar funktionen `f` på de element som gör `p` sann och lämnar de andra oförändrade. (2p)

Exempelanvändning:

```
> mapOn (*2) (\x -> x > 3) [1,2,3,4,5]
[1,2,3,8,10]
```

(b) Betrakta följande datatyp för sten-sax-påse (eng. rock-paper-scissor) spel:

```
data RPS = Rock | Paper | Scissor
  deriving (Eq,Show)
```

Uppgiften är nu att skriva Haskellkod för sten-sax-påse spel.

i. Skriv en datatyp `Winner` som kan vara antingen `Player x` (där `x` är en `Int`), som ska användas om spelare `x` vinner, eller `Draw` om det är oavgjort. (1p)

ii. Lägg även till en funktion `play :: (RPS,RPS) -> Winner` som returnerar vem som vann. Reglerna är som vanligt att papper slår sten, sten slår sax, sax slår papper, och vid samma blir det oavgjort. Första elementet i paret är draget för `Player 1` och andra är `Player 2`. (1p)

iii. Lägg till en funktion `winner :: [(RPS,RPS)] -> Winner` som tar in en lista av spel, applicerar `play` på alla, och returnerar den som vunnit flest spel. Om `Player 1` och `Player 2` vunnit lika många ska `Draw` returneras. (2p)

Exempelkörning:

```
> play (Scissor,Rock)
Player 2
> play (Scissor,Scissor)
Draw
> winner [(Scissor,Rock),(Paper,Rock),(Scissor,Paper)]
Player 1
> winner [(Scissor,Rock),(Paper,Rock),(Scissor,Scissor)]
Draw
```

12. Logikprogrammering i Prolog

(a) Skriv ett predikat `intersect(XS,YS,ZS)` så att `ZS` är snittet av `XS` och `YS` *utan dubletter*. För full poäng ska det ges ett svar direkt och Prolog ska inte låta användaren be om fler lösningar. (4p)

Exempelanvändning:

```
?- intersect([1,2,3],[4,2,5,3],ZS).
ZS = [2, 3].
?- intersect([3,2,1],[4,2,5,3,2],ZS).
ZS = [3, 2].
?- intersect([3,1,1,2,2,3],[4,2,5,3,2],ZS).
ZS = [2, 3].
?- intersect([3,1,1,2,2,3],[4,5,7],ZS).
ZS = [].
```

Tips: här får man använda det inbyggda predikatet `member(X,XS)` som testar om `X` finns i `XS`.

(b) Skriv `common_max(XS,YS,N)` där `XS` och `YS` är listor och `N` ett tal som ska vara det maximala värdet som finns i både `XS` och `YS`. Finns det inga gemensamma värden i `XS` och `YS` ska resultatet bli `false`. (2p)

Exempelanvändning:

```
?- common_max([3,1,1,2,2,3],[4,2,5,3,2],N).
N = 3.
?- common_max([3,1,1,2,2,3],[4,2,5,2],N).
N = 2.
?- common_max([3,1,1,2,2,3],[4,5],N).
false.
```

Tips: här kan man använda lösningen på (a) även om man inte löst den uppgiften. Man kan även använda det inbyggda predikatet `max_list(XS,N)` som relaterar listan `XS` med det maximala värdet `N`.