MATEMATISKA INSTITUTIONEN
STOCKHOLMS UNIVERSITET
Avd. Beräkningsmatematik
Kursledare: Anders Mörtberg
Examinator: Lars Arvestad

Tentamensskrivning i
DA4003 Programmeringsparadigm
7.5 hp
2021-04-19

- **Del 1** består av ett antal frågor med varierande antal poäng (totalt 32) vilka ska lösas genom att man skriver kod i de olika programmeringsspråken i kursen.
- **Del 2** ges muntligen och består av flervalsfrågor där minst ett svarsalternativ är korrekt. Om man svarar fel eller inte har exakt rätt antal alternativ får man 0 poäng på frågan.
- **Skriv tydligt:** onödigt svårläst eller komplicerad kod leder till poängavdrag, så indentera och kommentera koden på ett lämpligt sätt.
- Inga externa bibliotek får användas om det inte står explicit i uppgiften.
- **Hjälpmedel:** Del 1 är öppen bok då det är hemtenta och samma regler kring hjälpmedel gäller som för labbarna. På Del 2 får man ha papper och penna.
- För att få godkänt måste man ha minst 4 poäng på Del 2.
- **Betygsgränser:** E: 20, D: 24, C: 28, B: 32, A: 36, av maximala 40.
- **Inlämning:** lösningarna ska lämnas in på inlämningsmodulen under ”Tentamen” längst ner på kurssidan. Man ska ha en fil per uppgift och filnamnen ska vara de som står i uppgifterna. Börja alla filer med att skriva en kommentar med den anonyma koden som ni fått när ni registrerade er på tentan i Ladok. Denna kod är på formen på formen `XXX-XXXX-XXX`. Man ska **inte** skriva sitt riktiga namn någonstans i filen!
- Frågor om tentan ställs via mail till Anders (`anders.mortberg@math.su.se`). Han kommer vara tillgänglig under hela tentan och svara på frågor så fort som möjligt. Eventuella korrektioner och förtydliganden kommer publiceras på kursforumet.

Del 1: kodfrågor (32p totalt)

1. **Imperativ programmering:** Lösningarna på den här delen ska vara skrivna i C och läggas i `uppgift1.c`. På båda uppgifterna får man använda `stdio.h` och `stdlib.h`.

   a) På den ordinarie tentan 2021-03-15 handlade en uppgift om att kryptera strängar med hjälp av matriser. Det gick till så man skriver bokstäverna i en matris med bestämt antal kolumner och sen läser man ut det krypterade ordet genom att läsa kolumnvis.

   Exempel: för att kryptera `Secret text` med `3` kolumner skapar vi matrisen:

S	e	c
r	e	t
	t	e
x	t	

   Den krypterade texten är då `Sr xeettcte`.

   Uppgiften är nu att skriva en funktion för att avkryptera meddelanden. Detta ska göras genom funktionen `void matrix_decrypt(char* s, int n, int cols)` där `s` är strängen vi ska avkryptera, `n` är längden på strängen och `cols` är hur många kolumner matrisen ska ha. (2p)

   Tester: om man kompilerar och kör

   ```
   int main() {
     matrix_decrypt("Sr xeettcte",11,3);
     matrix_decrypt("Sce eterttx",11,2);
     matrix_decrypt("Stetcerxet",10,5);
   }
   ```

   så ska utdata bli

```
Secret text
Secret text
Secrettext
```

Obs: om längden på `s` inte är jämnt delbar med `cols` så ska man inte få några extra mellanslag eller andra tecken i slutet av den ackrypterade strängen.

b) Skriv en version av `matrix_decrypt` som heter `matrix_decrypt_goto` som endast använder `goto` istället för loopar.

Tester: om man kompilerar och kör

```
int main() {
  matrix_decrypt_goto("Sr xeettcte",11,3);
  matrix_decrypt_goto("Sce eterttx",11,2);
  matrix_decrypt_goto("Stetcerxet",10,5);
}
```

så ska man få samma utdata som ovan. (3p)

2. **Objektorienterad programmering:** Lösningarna på den här delen ska vara skrivna i Java och läggas i `uppgift2.java`. Man får använda `java.util.Map` och `java.util.HashMap` för att lösa uppgiften.

   På den här uppgiften ska ni implementera klasser för algebraiska uttryck. Dessa ska kunna vara konstanter, variabler och addition av två algebraiska uttryck.

   Börja med att lägga in en *abstrakt* klass `AlgExpr` med en `abstrakt` metod `eval(Map<Character,Integer> d)` som returnerar en `Integer`. (1p)

   Lägg sedan in följande klasser vilka alla ska ärva från `AlgExpr`:

   - `Constant` med ett *privat* heltalsvärde som attribut.
   - `Var` med ett *privat* attribut av typ `Character`.
   - `Add` med två *privata* attribut av typ `AlgExpr`.

   Dessa tre klasser ska ha konstruktorer samt metoderna `toString()` och en implementation av `eval`. (6p)

   Tester: För att testa era algebraiska uttryck kan ni använda följande kodsnutt:

```
public class uppgift2 {
    public static void main(String[] args) {
        AlgExpr e =
          new Add(new Var('x'),
                  new Add(new Constant(42),new Var('y')));

        System.out.println(e);

        Map<Character,Integer> d = new HashMap<Character,Integer>();
        d.put('x',2);
        d.put('y',-5);
        System.out.println(e.eval(d));

    }
}
```

   Med detta ska utdata bli:

```
x + 42 + y
39
```

3. **Webb och händelsestyrd programmering:** Kod som krävs för att lösa uppgiften finns att hämta på kurshemsidan i samma mapp som tentan. Så börja med att ladda ner filerna `uppgift3.html`, `style.css` och `uppgift3.js`. Lösningen på den här delen ska vara skriven i JavaScript och er uppgift är att skriva klart `uppgift3.js`. När ni är klara ska endast denna fil lämnas in (så ni ska inte ändra i HTML och CSS filerna). Skriv gärna en kommentar i toppen av filen vilken webbläsare ni testat lösningen med.

Uppgiften går ut på att implementera en variant av det lilla spelet från ordinarie tentan. Den här varianten utspelar sig på ett 5×5 bräde med nollor och ettor. När man börjar spelet är alla nollor låsta och målet är att klicka på den etta med högst summa på raden och kolumnen som den ligger på (utan att ettan man klickat på räknas). Lyckas användaren göra detta byts värdet i rutan ut mot 0 och rutan går inte att klicka på igen. Man ska sedan hitta den etta som nu har högst summa på raden och kolumnen som den ligger på (utan att räkna ettan själv). Detta fortsätter tills spelaren klickat på alla rutor i rätt ordning och ett lämpligt meddelande skrivs sedan ut. Misslyckas man med att klicka på rätt ruta får man ett meddelande som säger det och får försöka igen.

Filen `uppgift3.js` innehåller strukturen på programmet och de funktioner som finns implementerade är:

- `get_grid()`: hämta nuvarande bräde som en array av arrayer.
- `best_value(g)`: beräkna det bästa värdet i `g` (dvs den maximala summan av en rad och en kolumn utan att man räknar rutan de möts på).
- `klick()`: funktionen som ska köras när användaren klickar på en ruta.

De funktioner ni ska lägga till är:

a) `init(g)`: rita startbrädet enligt `g`. Alla rutor med `1` ska kunna klickas på och då ska funktionen `klick()` köras. Alla rutor med `0` ska inte gå att klicka på och de ska ha bakgrundsfärgen `#b6d9ee`. (2p)

b) `sum_row_col(g,r,c)`: beräkna summan av de 8 celler på raden och kolumnen som cellen `g[r][c]` ligger på. Här antas `g` vara ett bräde (dvs array av arrayer) och `r` samt `c` är heltal. (2p)

c) `is_complete(g)`: har användaren klickat på alla rutor i `g` i rätt ordning? Dvs, är värdet på alla de 25 rutorna 0? (1p)

4. **Funktionell programmering:** Lösningarna på den här delen ska vara skrivna i Haskell och läggas i `uppgift4a.hs`, `uppgift4b.hs` och `uppgift4c.hs`.

a) Skriv en funktion `mask :: String -> String -> String` som tar in två strängar `s1` och `s2` som byter ut bokstäverna till `*` i `s1` i den ordning de förekommer i `s2`.

Exempel:

```
> mask "Hello, how are you Anders?" "ehoaAr"
"H*llo, **w *re you *nde*s?"
> mask "Hello, how are you Anders?" "H,?A"
"*ello* how are you Anders*"
```

Notera att i det första exemplet byts inte `o` ut i `you` då det redan använts för att byta ut `o` i `how`. (2p)

b) Implementera en högre ordningens funktion `mapFilter :: (a -> b) -> (b -> Bool) -> [a] -> [b]` som fungerar så att `mapFilter f p xs` applicerar `f` på alla värden i `xs` och sen bara sparar de värden som uppfyller `p` (dvs för vilka `p` evaluerar till `True` för).

För poäng för man inte använda några inbyggda Haskellfunktioner som `map` eller `filter`. (2p)

c) Implementera matriskryptering från C delen på förra ordinarie tentan 2021-03-15 som en funktion `matrixEncrypt :: String -> Int -> String`. Den ska fungera så att `matrixEncrypt s n` krypterar strängen `s` med hjälp av en matris med `n` kolumner.

Tips: skriv hjälpfunktioner som använder sig av Haskellfunktionerna `take` och `drop`. (5p)

Exempel:

```
> matrixEncrypt "Secret text" 3
"Sr xeettcte"
> matrixEncrypt "Secret text" 2
"Sce eterttx"
> matrixEncrypt "Secrettext" 5
"Stetcerxet"
```

5. **Logikprogrammering:** Lösningarna på den här delen ska vara skrivna i Prolog och läggas i `uppgift5a.pl` och `uppgift5b.pl`.

   a) Implementera ett predikat `mask(XS,YS,ZS)` som fungerar så att `ZS` är den maskerade varianten av `XS` enligt `YS`. (3p)

   Tips: ASCII koden för `*` är `42`.

   Exempel:

```
?- string_to_list("Hello, how are you Anders?",XS),
   string_to_list("ehoaAr",YS),
   mask(XS,YS,ZS),
   string_to_atom(ZS,WS).
...
WS = 'H*llo, **w *re you *nde*s?'
?- string_to_list("Hello, how are you Anders?",XS),
   string_to_list("H,?A",YS),
   mask(XS,YS,ZS),
   string_to_atom(ZS,WS).
...
WS = '*ello* how are you Anders*'
```

   b) Molekylmassan (i Dalton) för de fyra kvävebaserna i DNA är

Adenin (A)	313.2
Cytosin (C)	289.2
Guanin (G)	329.2
Tymin (T)	304.2

   Skriv ett predikat `dna_mass(S,M)` som relaterar en sträng `S` med bokstäverna A, C, G och T med summan av dess molekylmassa `M`. Om `S` innehåller andra bokstäver än A, C, G och T, så ska dessa inte räknas med i summan. (3p)

   Tips: ASCII koden för `A` är `65`, `C` är `67`, `G` är `71`, and `T` är `84`.

   Exempel:

```
?- dna_mass("AGATACGTA",M).
M = 2808.7999999999997 .
?- dna_mass("AGxAYTxACGxTXA",M).
M = 2808.7999999999997 .
```

Del 2: flervalsfrågor (1p per fråga, 8p totalt)

6. Vilka termer hänger specifikt ihop med objektorienterad programmering?

   A. Rekursion
   B. Klasser
   C. Högre ordningens funktioner
   D. Snitt
   E. Arv

7. Vad blir resultatet av följande listomfattning `[ [ x !! i | x <- [[1,2,3],[4,5,6]] ] | i <- [0,1,2] ]` i Haskell?

   A. `[[0,1,2],[0,1,2]]`
   B. `[]`
   C. `[[1,2,3],[4,5,6]]`
   D. `[[1,4],[2,5],[3,6]]`
   E. `[1,2,3,4,5,6]`

8. Vad används ”`!`” till i Prolog?

A. Snitt
B. Negation
C. Att ta bort en regel
D. Att markera slutet på en rad
E. Det finns ingen ”`!`” i Prolog

9. Vad är ett ”klassattribut” inom objektorienterad programmering?

A. En funktion som delas av alla instanser av klassen
B. En funktion som tillhör en specifik instans av klassen
C. En variabel som delas av alla instanser av klassen
D. En variabel som tillhör en specifik instans av klassen
E. Det finns inget som heter ”klassattribut” inom objektorientering

10. Vad returnerar en parser?

A. Ett syntaxträd
B. En lexikografiskt sorterad lista av ord
C. Rad och kolumnnummer för alla ord i en fil
D. En exekverbar fil
E. En lista med tokens

11. Vad kommer skrivas ut om man kör C-koden till höger?

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

int *p = &x[2];

printf("%d", *(p+1));
```

A. `%d`
B. `3`
C. `4`
D. `5`
E. Det går inte att säga då `*(p+1)` är en godtycklig minnesadress.

12. Vad är typen på Haskellfunktionen:

```
f x y = y ++ snd x
```

A. `(a,[b]) -> c -> (a,[c])`
B. `(a,[b]) -> [b] -> (a,[b])`
C. `(a,b) -> [b] -> (a,b)`
D. `(a,[b]) -> [b] -> [b]`
E. Ingen då koden leder till ett typfel.

13. Vilka påståenden nedan är sanna?

A. JavaScript är objektorienterat
B. Typfel hittas vid kompilering för dynamiskt typade språk
C. En polymorf funktion är en funktion som är skriven med rekursion
D. `X` och `p(X)` kan unifieras med `=` i Prolog
E. Man kan returnera pekare från funktioner i C