NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultetet for informasjonsteknologi, matematikk og elektroteknikk

Institutt for datateknikk og informasjonsvitenskap

BOKMÅL

Sensurfrist: 21. januar 2011

LØSNINGSFORSLAG:

Avsluttende eksamen i TDT4110/IT1102 Informasjonsteknologi, grunnkurs

Tirsdag 21. desember 2010
9.00 – 13.00

Faglig kontakt under eksamen:
Alf Inge Wang (922 89 577)

Hjelpemidler: C
Tilleggshefte I, "Introduksjon til HTML, CSS, JSP og MYSQL"
Bestemt, enkel kalkulator: HP 30S eller Citizen SR270-X

Sensur:
Resultater gjøres kjent på http://studweb.ntnu.no.

Oppgavesettet inneholder 4 oppgaver. Det er angitt i prosent hvor mye hver oppgave og hver deloppgave teller ved sensur. Les igjennom hele oppgavesettet før du begynner å lage løsning. Disponer tiden godt! Gjør rimelige antagelser der du mener oppgaveteksten er ufullstendig, skriv kort hva du antar.

En liste over metoder som fritt kan brukes i alle programmeringsoppgavene er vedlagt.

Svar kort og klart, og skriv tydelig. Er svaret uklart eller lenger enn nødvendig trekker dette ned.

Lykke til!

Innhold:

Oppgave 1: Flervalgsoppgave (25 %)

Bruk de to vedlagte svarskjemaene for å svare på denne oppgaven (ta vare på det ene selv). Du kan få nytt ark av eksamensvaktene dersom du trenger dette. Kun ett svar er helt riktig. For hvert spørsmål gir korrekt avkryssing 1 poeng. Feil avkryssing eller mer enn ett kryss gir −1/2 poeng. Blankt svar gir 0 poeng. Du får ikke mindre enn 0 poeng totalt på denne oppgaven. Der det er spesielle uttrykk står den engelske oversettelsen i parentes.

1) Hva betyr det at et dataprogram er opphavsrettslig beskyttet?
   a) Det kan ikke kopieres og brukes uten at rettighetshaver får betalt.
   b) Det kan ikke kopieres og brukes uten at rettighetshaver blir informert.
   c) Det kan ikke kopieres og brukes uten rettighetshavers samtykke.

2) Hva defineres som en sensitiv personopplysning?
   a) Informasjon om kjøp i en nettbutikk.
   b) Medlemskap i et idrettslag.
   c) Både a og b.
   d) Verken a eller b.

3) Kari bryter seg inn på brukerkontoen til Ola, kopierer musikk som Ola har laget, og selger denne musikken på internett. Hvilke lover har Kari brutt?
   a) Lov om opphavsrett til åndsverk og Straffeloven.
   b) Straffeloven.
   c) Lov om opphavsrett til åndsverk.
   d) Ingen lover.

4) 01010101 i binærtallsystemet, er det samme som (i titallsystemet)?
   a) 105
   b) 95
   c) 85

5) 4095 (i titallsystemet) skal kodes heksadesimalt (16-tallsystemet). Hvor mange siffer blir det i resultatet?
   a) 5
   b) 4
   c) 3

6) En ER-modell:
   a) Viser hvilke entiteter og relasjoner som faktisk er lagret i en database.
   b) Beskriver informasjonsstrukturen i en database.
   c) Beskriver strukturen i et ERP-system (Enterprice Resource Planning System).

7) Anta at karakterene har følgende fordeling: A (7 %), B (20 %), C (35 %), D (25 %), E (8 %) og F (5 %). Hva er riktig Huffmannkoding for A-F?

a)	**11110 (A)**	**110 (B)**	**0 (C)**	**10 (D)**	**1110 (E)**	**11111 (F)**
b)	001 (A)	010 (B)	011 (C)	100 (D)	101 (E)	111 (F)
c)	110 (A)	01 (B)	0 (C)	1 (D)	11 (E)	111 (F)

8) I JSP, hva blir resultatet av: 1+2/2–1?
   a) 0,5
   b) 1
   c) 3

9) En melding består av resultatet av 100 myntkast (mynt/krone). En annen melding består av resultatet av 100 terningkast (1–6 øyne). Hvilken melding har høyest entropi?
   a) Mynt-meldingen.
   b) Terning-meldingen.
   c) De har samme entropi.

10) Hvilket av alternativene er gyldig CSS?
   a) style="font-weight=bold"
   b) style="font-weight: bold, color: red"
   c) style="font-weight: bold;"

11) Hva er den *viktigste* oppgaven til en nettverkssvitsj?
   a) Sørge for at alle påkoblede maskiner mottar alle datapakker.
   b) Overvåke og stenge ute uønsket nettverkstrafikk.
   c) Fordele datatrafikken slik at datapakkene kommer frem til riktig mottaker.

12) Vi har en sortert liste med 5 000 000 elementer. Ved binærsøking i denne listen, hvor mange sammenligninger må vi i verste fall gjøre?
   a) Omtrent 20
   b) Omtrent 30
   c) Omtrent 40

13) For et større nettsted, hva er den *viktigste* fordelen ved å samle CSS-stiler i en stilfil?
   a) Det tvinger HTML-filene til å bruke de samme stilene og gir et konsistent utseende.
   b) Det tar mindre plass enn å gjenta de samme stilene i mange HTML-filer.
   c) Det gjør det enkelt å ha felles stiler for mange HTML-filer.

14) `<a…>…</a>` i en HTML-fil definerer:
   a) Et adressefelt.
   b) Et avsnitt.
   c) En hyperlenke.

15) Hvilket av disse er et *ikke-funksjonelt* krav?
   a) Systemet skal registrere nye brukere.
   b) Systemet skal kryptere persondata.
   c) Systemet skal gi en oversikt over ledige seter.

16) Testing av programkode vil si at man:
   a) Går gjennom programkoden og prøver å finne feil.
   b) Lar noen utvalgte brukere prøve systemet.
   c) Kjører programkoden og sjekker at forholdet mellom inndata og utdata er som forventet.

17) Hva er fordelen med *asymmetrisk* kryptering fremfor symmetrisk kryptering?
   a) Man kan publisere den ene nøkkelen.
   b) Det er vanskeligere å knekke koden fordi man må finne to ulike nøkler.
   c) Man kan være sikker på identiteten til den som har sendt meldingen (autentisering).

18) Hva er nettverkstjenester?
   a) E-post, filoverføring og trådløst nettverk.
   b) Fildeling, WWW og e-post.
   c) Filoverføring, TCP/IP og WWW.

19) CPU er forkortelse for:
   a) Central Pipelining Unit.
   b) Coordinating Processor Unit.
   c) Central Processing Unit.

20) Et tall av typen double lagres i 8 byte. Omtrent hvor mange slike tall kan lagres i en gigabyte (GB)?
   a) Ca. 134 000.
   b) Over 100 millioner.
   c) Omtrent en milliard.

21) Hvilken betydning har *klasse* (eng: class) i forbindelse med HTML/CSS:
   a) Det definerer klasser av lignende tags, for eksempel klassen av overskrift-tags (<h1>, <h2>, etc.).
   b) Det lar oss skille mellom viktige tags (head, body, etc) og ubetydelige tags (, etc.).
   c) Det lar oss definere en delmengde av alle forekomster av en tag, som vi kan gi spesiell behandling.

22) På hvilken måte skiller harddisk seg fra primærminne (RAM):
   a) Det tar lengre tid å skrive eller lese.
   b) Dataene er sikre mot strømbrudd.
   c) Verken a eller b.
   d) Både a og b.

23) `A && (B || C)` er true (sant) for:
   a) A true, B true, C false
   b) A true, B false, C false
   c) A false, B true, C true

24) Hovedgrunnen til at vi deler opp lengre programmer ved å bruke metoder:
   a) Det går raskere å kjøre programmet.
   b) Programmet får høyere funksjonalitet.
   c) Det er lettere å forstå programmet.

25) Hva stemmer *ikke* om kommentarer i programkode?
   a) Programmet tar større plass på harddisken.
   b) De gjør det lettere å forstå koden.
   c) Programmet kjører langsommere.

Oppgave 2: Programforståelse (10 %)

Følgende metoder er definert:

```
int a(int x) {
  x = x + 1;
  int y = 1 + x * 2;
  return y;
}
```

```
int b(int n) {
  int y;
  if (n<40) {
    y = 2 * n; }
  else if (n<10) {
    y = n; }
  else {
    y = 1; }
  return y;
}
```

```
int c(int w) {
  int z = b(a(w));
  if (z < 10) {
    z = z + w;  }
  return z;
}
```

a) (2 %) Hva er verdien til *x* etter at vi har kjørt `int x = 2; x = a(x);`?
b) (2 %) Hva er verdien til *x* etter at vi har kjørt `int x = 2; x = b(x);`?
c) (2 %) Hva er verdien til *x* etter at vi har kjørt `int x = 2; x = c(x);`?
d) (2 %) Hva er verdien til *x* etter at vi har kjørt `int x = 20; x = b(x);`?
e) (2 %) Hva er verdien til *x* etter at vi har kjørt `int x = 50; x = b(x) + c(x);`?

Løsning:
a) 7
b) 4
c) 14
d) 40
e) 52

Oppgave 3: Programmering (15 %)

I skihopp gis det poeng for hopplengde og stil.

a) (5 %) Poeng for hopplengde beregnes med følgende formel:

distance_points = 60 + (*jump_distance* – *kpoint*) $\star$ *meter_value*

Tallene *kpoint* og *meter_value* er fastsatt for hver hoppbakke.

Hoppbakken i Granåsen har *kpoint* 124 og *meter_value* 1,8. En skihopper som hopper 140 meter i denne bakken vil få 60 + (140 – 124) $\star$ 1.8 = 88.8 lengdepoeng (distance points).

Skriv en metode *distance_points* som tar inn parametrene *distance* (hopplengde), *kpoint* (k-punkt) og *meter_value* (meterverdi), og returnerer lengdepoeng.

Løsning:

```
double distance_points(double distance, double kpoint, double meter_value)
{
      return 60.0 + (distance-kpoint)*meter_value;
}
```

b) (10 %) Et skihopp belønnes med 0–60 stilpoeng. Fem dommere gir 0–20 poeng hver. Den laveste og den høyeste poengsummen strykes, og de tre resterende poengsummene legges sammen og utgjør hoppets stilpoeng.

Hvis et skihopp får poengsummene 17, 17.5, 17.5, 18, 19, strykes 17 og 19, og stilpoengene blir 17.5 + 17.5 + 18 = 53.

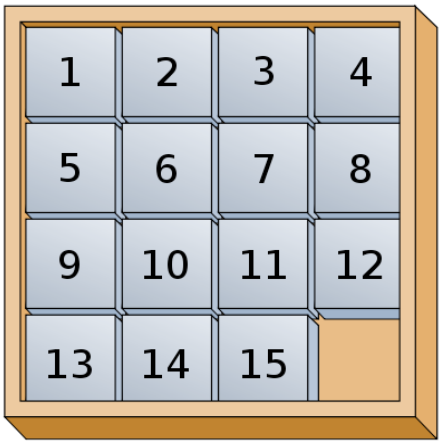
Skriv en metode *style_points* som tar inn en usortert liste *points* med de fem dommerpoengsummene, og returnerer hoppets stilpoeng. (Hint: bruk max og min).

Løsning:

```
double style_points(double[] points) {
     int max=max(points);
     int min=min(points);
     double sum = 0;
     for (int i=0;i<points.length;i++) {
          if (i!=max && i!=min) {
               sum=sum+points[i];
          }
     }
     return sum;
}
```

Oppgave 4: Programmering (50 %)

I denne oppgaven skal du programmere metoder til "15-spillet" som er illustrert i figur 1. Spillet består av et brett med 4x4 ruter med 15 brikker med tallene 1 til 15. En rute er tom, og den kan brukes til å flytte om på brikkene innbyrdes for å endre på rekkefølgen. Spillet starter ved at brikkene står i tilfeldig rekkefølge. Målet med spillet er å få brikkene i riktig rekkefølge fra 1 til 15 med den siste ruten fri slik som vist i figur 1. Spillebrettet skal i koden representeres som en 4x4-tabell med heltall som vist i figur 2.



Figur 1. Spillebrett til "15-spillet".

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	0

Figur 2. Spillebrettet representert som tabell.

I denne oppgaven er det hensiktsmessig å gjenbruke metoder du lager. Du kan *bruke* metoder fra andre deloppgaver selv om du ikke har klart å løse deloppgaven hvor du skal lage metoden.

a) (5 %) Skriv metoden *number_in_list* som tar inn et heltall, *number,* og en liste (endimensjonal tabell) med heltall, *list*. Hvis tallet *number* finnes i listen skal metoden returnere true, ellers false.

Løsning:

```
boolean number_in_table(int number, int[] table) {
     boolean found = false;
     for(int i=0;i<table.length;i++) {
          if (number==table[i]) {
               found = true;
          }
     }
     return found;
}
```

b) (5 %) Skriv metoden *random_list* som tar inn et heltall, *number,* og returnerer en liste (endimensjonal tabell) med heltallene fra og med 1 til og med tallet *number* i tilfeldig rekkefølge. Bruk metoden *number_in_list* fra oppgave a) i løsningen av denne oppgaven.

Gjør man metodekallet *random_list*(15) kan for eksempel følgende tabell returneres:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
3	2	15	13	12	9	1	11	5	14	7	6	4	8	10

Figur 3. Tabell med 15 tall i tilfeldig rekkefølge.

Løsning:

```
int[] random_table(int number) {
     int[] table = new int[number];
     for (int i=0;i<table.length;i++) {
          boolean new_number=false;
          while(!new_number) {
               int random = (int) (Math.random()*number)+1;
               if (!number_in_table(random,table)) {
                    table[i]=random;
                    new_number=true;
               }
          }
     }
     return table;
}
```

c) (5 %) Skriv metoden *new_level* som tar inn en liste (endimensjonal tabell), *list*, bestående av 15 heltall i tilfeldig rekkefølge som vist i figur 3. Metoden skal returnere en 4x4-tabell der tallene i *list* er satt inn fortløpende, radvis, fra rad 0, kolonne 0, til rad 3, kolonne 2. Elementet med indeks 3,3 skal ha verdien 0.

Kalles metoden *new_level* med listen vist i figur 3, skal den returnere tabellen vist i figur 4.

	0	1	2	3
0	3	2	15	13
1	12	9	1	11
2	5	14	7	6
3	4	8	10	0

Figur 4. Et tilfeldig spillebrett i 4x4-tabell.

Løsning:

```
int[][] new_level(int[] table) {
     int[][] level = new int[4][4];
     int x=0;
     int y=0;

     for (int i=0;i<table.length;i++) {
          level[x][y] = table[i];
          x=x+1;
          if (x>3) {
               x=0;
               y=y+1;
          }
     }
     return level;
}
```

d) (15 %) Skriv metoden *move_tile* som tar inn et spillebrett, *level*, og en tekststreng, *direction*, som angir retning der "l" er venstre (left), "r" er høyre (right), "d" er ned (down), og "u" er opp (up). Metoden skal returnere et spillebrett der ruten med tallet 0 har byttet plass med tallet som befinner seg i naboruten i angitt retning hvis et slikt bytte er mulig. Hvis byttet ikke er mulig skal metoden returnere et uendret spillebrett.

I eksemplet vist i figur 4, kan ruten med tallet 0 bytte plass opp (med tallet 6) og til venstre (med tallet 10).

Merk at metoden skal være generell og fungere uansett hvor ruten med tallet 0 er plassert i tabellen. Hvis metoden *move_tile* kalles med tabellen vist i figur 4 og retning "u" (opp) som parametre, så vil tallet 6 bytte plass med tallet 0 i tabellen.

Løsning:

```
// Trenger ikke å returnere level etter som det er en tabell der endringene vil overleve
// et metode kall. Også godkjent å returnere et nytt spillbrett.
void move_tile(int[][] level, String direction) {
     int x=-1;
     int y=-1;

     // Finn tom plass i brett (verdi 0)
     for(int i=0;i<level.length;i++) {
          for(int j=0;j<level.length;j++) {
               if (level[i][j]==0) {
                    x=i;
                    y=j;
               }
          }
     }
     // sjekk retning og om den kan flyttes i den retningen
     if (direction.equals("v") && x>0) {
          level[x][y]=level[x-1][y];
          level[x-1][y]=0;
     } else if (direction.equals("h") && x<3) {
          level[x][y]=level[x+1][y];
          level[x+1][y]=0;
     } else if (direction.equals("o") && y>0) {
          level[x][y]=level[x][y-1];
          level[x][y-1]=0;
     } else if (direction.equals("n") && y<3) {
          level[x][y]=level[x][y+1];
          level[x][y+1]=0;
     }
}
```

e) (5 %) Skriv metoden *correct_place* som tar inn et spillebrett, *level,* og returnerer antall tall som er riktig plassert på spillebrettet. Korrekt plassering av tallene er som vist i figur 2. Kalles metoden *correct_place* med spillebrettet fra figur 4, vil metoden gi svaret 2, ettersom tallene 0 og 2 er korrekt plassert på spillebrettet.

Løsning:

```
int correct_place(int[][] level) {
     int correct=0;
     int counter=1;
     // Sjekk om tallene står på riktig plass i tabellen
     for (int y=0;y<level.length;y++) {
          for (int x=0;x<level[0].length;x++) {
               if(level[x][y]==counter) {
                    correct=correct+1;
               } // End if
          counter++;
          } // End for x
     } // End for y

     // Sjekk om 0 står på riktig plass
     if (level[3][3]==0) {
          correct=correct+1;
     }

     return correct;
}
```

f) (5 %) Skriv metoden *level_to_html* som tar inn et spillebrett, *level*, og returnerer en tekststreng der spillebrettet er formatert som en tabell i HTML. HTML-koden skal formateres slik at tabellen vil se ut som på figur 5 hvis den skrives ut til en nettleser. Avstand fra tallene til strekene i tabellen skal være på 20 piksler.

14	9	7	8
1	10	5	4
2	11	13	3
12	15	6	0

Figur 5. Spillebrett formatert som HTML-tabell.

Løsning:

```
String level_to_html(int[][] level) {
     String html="<table border=\"1\" cellpadding=\"20\">\n";
     for (int y=0;y<level.length;y++) {
          html=html+"<tr>\n";
          for(int x=0;x<level[0].length;x++) {
               html=html+"<td>"+level[x][y]+"</td>\n";
          }
          html=html+"</tr>\n";
     }
     html=html+"</table>\n";
     return html;
}
```

g) (10 %) Lag koden for å utføre følgende i et JSP-skript:

1. Opprett nødvendige variabler.
2. Fyll et spillebrett med tallene 1 til 15 i tilfeldig rekkefølge. Vi ønsker at spillebrettet skal ha minst 10 tall på riktig sted, så skriptet må fylle ut nye brett helt til et slikt spillebrett er funnet.
3. Skriv ut spillebrettet med minst 10 riktig plasserte tall til nettleseren ved hjelp av metoden *level_to_html*.

Løsning:

```
<%
int[] table= new int[15]; // 15 tall i tilfeldig rekkefølge
int[][] level = new int[4][4]; // spillbrettet

boolean solved = false;
while (!solved) {
     table= random_table(15); // fyll inn 15 tall
     level = new_level(table); // fyll ut spillbrettet
     int correct = correct_place(level); // sjekke plassering
     if (correct>9) {
          solved=true;
     }
}

String level_html = level_to_html(level); // Lag HTML-tabell
out.println(level_html); // Skriv ut til nettleser
%>
```

Vedlegg: JSP-metoder til programmeringsoppgavene

Math.random Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.

Eksempel på bruk:
```
int x = (int) (Math.random()*10);
```

Man skulle ikke programmere metodene max og min på eksamen. Koden er kun tatt med for å forklare hva metoden gjør i detalj.

max Returns the index of the largest number in an array of doubles.

Eksempel på bruk:
```
double[] table = {5, 8, 54, 3, 23.2};
int index = max(table); // index = 2;
```

Kode:
```
int max(double[] table) {
      int index=0;
      double max = table[0];
      for (int i=1; i<table.length;i++) {
            if (table[i]>max) {
                  max=table[i];
                  index=i;
            }
      }
      return index;
 }
```

min Returns the index of the smallest number in an array of doubles.

Eksempel på bruk:
```
double[] table = {5, 8, 54, 3, 23.2};
int index = min(table); // index = 3;
```

Kode:
```
int min(double[] table) {
      int index=0;
      double min=table[1];
      for (int i=1; i<table.length;i++) {
            if (table[i]<min) {
                  min=table[i];
                  index=i;
            }
      }
      return index;
 }
```

Svarskjema flervalgsoppgave

Studentnummer: _____ Linje: _____

Fagkode: _____ Dato: _____

Antall sider: _____ Side: _____

Oppgave nr.	*A*	*B*	*C*	*D*
1.1			O	
1.2				O
1.3	O			
1.4			O	
1.5			O	
1.6		O		
1.7	O			
1.8		O		
1.9		O		
1.10			O	
1.11			O	
1.12	O			
1.13			O	
1.14			O	
1.15		O		
1.16			O	
1.17	O			
1.18		O		
1.19			O	
1.20		O		
1.21			O	
1.22				O
1.23	O			
1.24			O	
1.25			O	