NTNU
Kunnskap for en bedre verden

Institutt for datateknikk og informasjonsvitenskap

Eksamensoppgave i TDT4105 Informasjonsteknologi – grunnkurs, med Matlab

Faglig kontakt under eksamen:	Rune Sætre	Mobil: 452 18103
	Anders Christensen	Mobil: 918 97181
	Terje Rydland	Mobil: 957 73463
	Benjamin A. Bjørnseth	Mobil: 478 32483

Eksamensdato:	**2015-12-16**
Eksamenstid (fra-til):	**09:00 – 13:00**
Hjelpemiddelkode/Tillatte hjelpemidler:	**Godkjent kalkulator**

Annen informasjon:

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 løse oppgavene. Disponer tiden godt! Gjør rimelige antagelser der du mener oppgaveteksten er ufullstendig, skriv kort hva du antar.

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

Målform/språk:	**Bokmål**
Antall sider:	**19 (inkl. Forside, svarark og appendiks)**

Innhold:

- Oppgave 1: Flervalgsoppgave (25%)
- Oppgave 2: Kodeforståelse (15%)
- Oppgave 3: Programmering reisetid (20%)
- Oppgave 4: Programmering sensur (40%)
- Appendiks: Nyttige funksjoner
- Svarark til Flervalgsoppgave (2 eksemplarer)

Kontrollert av:

9.des. 2015	Alf Inge Wang
Dato	Sign

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 Random Access Memory?
    a. At det er tilfeldig hvor maskinen lagrer informasjon.
    b. At hukommelsescellene kan aksesseres direkte i tilfeldig rekkefølge.
    c. Hukommelsen er plassert på forskjellige, tilfeldige, plasser på hovedkortet.
    d. At det kan oppstå tilfeldige feil i deler av hukommelsen.
2. Når brukes fotolitografi i produksjonen av datamaskiner?
    a. Når man etser inn navnene på portene bak på maskinen.
    b. Under produksjon av integrerte kretser.
    c. Når man lager bildene som skal inn i brukermanualen.
    d. Når man monterer integrerte kretser på kretskortene.
3. Hva er «pipelining»?
    a. Et uttrykk for det som skjer når man skriver mye data til harddisken samtidig.
    b. En teknikk der man sender data mellom de forskjellige delene i maskinen i «pipes».
    c. En teknikk der en CPU kan utføre flere instruksjoner parallelt.
    d. En teknikk som fungerer som en ”sikker tunnel” mellom din maskin og en tjener.
4. Hva finner alle burst-feil med lengde n bit, gitt en n-bit maske, men er uegnet til kryptografi?
    a. Enkel sjekksum
    b. HASH-funksjoner
    c. Paritet
    d. Syklisk sjekksum (CRC)
5. I TCP/IP-protokollen …
    a. sendes alle pakkene langs den samme ruten til mottakeren.
    b. brukes pakkesvitsjing.
    c. mottas ingen ting før siste IP-pakke er framme.
    d. er det mindre interferens pga. at mindre biter sendes hver for seg.
6. Hva blir det binære tallet `10101010` desimalt?
    a. 170
    b. 180
    c. 190
    d. 200
7. Hvilken av følgende RGB-fargekodinger gir blått?
    a. `f1faf0`
    b. `120012`
    c. `0000ff`
    d. `884311`

8. Vi har en liste av navn, à la **liste = { 'Jo Å', 'Geir Li', 'Ine By' }** men i praksis med vesentlig flere navn enn dette. Lista er ikke sortert og kan inneholde duplikater (dvs. samme navn kan stå flere sted i lista). Vi skal skrive en funksjon **antall (liste, navn)** som skal returnere et heltall som sier hvor mange ganger navnet forekommer i lista. Vi kladder følgende pseudokode:

```
function antall (liste, navn):
        antall ← 0
        la n gå fra og med første til og med siste element i liste:
                hvis n == navn:
                        antall ←antall + 1
        returner antall
```

   Spørsmål: Kjøretidskompleksiteten til pseudokoden over vil være?
   a. $\Theta(n)$
   b. $\Theta(n \log n)$
   c. $\Theta(\log n)$
   d. $\Theta(n^2)$

9. I ei sortert liste kunne vi brukt binærsøk i stedet for løkka **"la n gå… "** i pseudokoden fra spørsmål 8. En alternativ algoritme som først sorterer lista, og deretter bruker binærsøk for å finne navnet, vil ha…
   a. lavere kompleksitet (dvs. være raskere) enn pseudokoden gitt over.
   b. høyere kompleksitet enn pseudokoden over.
   c. samme kompleksitet som pseudokoden over.
   d. høyere kompleksitet hvis navnet fins null eller én gang i lista, lavere hvis det fins flere ganger.

10. Funksjonen **antall** for ei usortert liste, som beskrevet i spørsmål 8, kan i Matlab implementeres ved de innebygde funksjonene **sum** og **strcmp**, som gjør at funksjonskroppen kan skrives som en eneste kodelinje. For eksempel

```
function ant = antall (liste, navn)

        ant = sum( strcmp( liste, navn) )
```

    Spørsmål: Hvilken kjøretidskompleksitet vil denne koden ha?
    a. $\Theta(1)$
    b. $\Theta(\log n)$
    c. $\Theta(n)$
    d. $\Theta(n^2)$

11. Hvorfor ønsker man å bruke en SSD heller enn en vanlig magnetisk harddisk?
    a. En SSD øker minnet på grafikkortet slik at spill og lignende flyter bedre.
    b. I en SSD lagres data i elektroniske kretser. Det er ingen bevegelige deler, og dermed blir disken raskere og mer pålitelig.
    c. Det er lettere å lagre data permanent på en SSD.
    d. En SSD er ikke så utsatt for strømtopper og tåler derfor mer enn en magnetisk disk.

12. Hva er motivasjon til fagfeltet systemutvikling?
    a. Raskere kode.
    b. Utvikle programvare med best mulig kvalitet uavhengig av budsjett og tid.
    c. Legge til rette for at all programvare skal utvikles i spesifiserte faser etter hverandre.
    d. Utvikle programvare med god nok kvalitet innen tid og budsjett.

13. Hva betyr modulering i f.eks. FM og AM?
    a. Det beskriver hvordan man kan sende et signal over en bærebølge.
    b. Det beskriver hvordan man kan få oversikt over hele internett.
    c. Det beskriver hvordan man kan øke strømstyrken slik at flere får tilgang.
    d. Det beskriver hvordan man kan gruppere internett i hensiktsmessige biter.

14. Dersom tekststreng 'Hallo' i ASCII representeres som følgende tall heksadesimalt:
    `48 61 6c 6c 6f`, hvordan representerer man på samme måte 'Morna'?
    a. `4e 65 69 64 61`
    b. `4e 54 4e 55 21`
    c. `4d 6f 72 6e 61`
    d. `55 66 6g 7h 61`

15. En fordel med vannfallsmodellen kan være:
    a. Enklere å håndtere øyeblikkelige krav fra kunder.
    b. Enklere å følge fremgang i forhold til prosjektplan for prosjektleder.
    c. Systemet reflekterer en gradvis bedre forståelse av brukernes behov.
    d. Gir raskere levering og kortere tid til å ta i bruk fungerende deler av systemet.

16. Hvor mange bytes trengs for å lagre et 24-bits bilde med 1280x1024 piksler uten komprimering?
    a. Ca. 3,8MB
    b. Ca. 1,2MB
    c. Ca. 24MB
    d. Ca. 24GB

17. Hva er den første aktiviteten i ”requirement engineering”-prosessen i følge læreboka?
    a. Gjennomførbarhetsstudie.
    b. Kravinnhenting og analyse.
    c. Kravspesifisering.
    d. Validering av krav.

18. Hva er akseptansetesting?
    a. Teste hvordan ulike deler av systemet fungerer sammen.
    b. Teste at hver enkelt funksjon i systemet fungerer som den skal.
    c. Teste at operativsystemet aksepterer systemet på plattformen.
    d. Teste med kundedata for å sjekke om systemet møter kundens behov.

19. Hvilken av følgende teknikker er en tapsløs komprimering?
    a. Run-length encoding.
    b. Analog-to-digital conversion.
    c. JPEG-encoding.
    d. Check-sum generation.

20. Hva er det Boehm’s spiralmodell inneholder, som man ikke finner i vannfallsmodellen eller inkrementell utvikling?
    a. Risikoanalyse.
    b. Testing/Validering.
    c. Kravspesifisering.
    d. Vedlikehold.

Oppgave 2 Kodeforståelse (15%)

Oppgave 2a (5%)
Hva blir skrevet ut til skjerm når du kjører funksjonen **main** i koden vist under? (3%)
Forklar med en setning hva funksjonen **`mystery`** gjør (2%)

```
function main()
A='SUNEAILSUN';
B='JALTNCSAES';
D=mystery(A,B);
disp(D);
end

function z = mystery(x,y)
    z='';
    for i = 1:length(x)
        if  mod(i,2)
            z(i) = y(i);
        else
            z(i) = x(i);
        end %if
    end %for
end %function
```

Oppgave 2b (5%)
Hva blir returnert hvis koden **`compute(1)`** som vist under blir kjørt? (3%)
Forklar med en setning hva funksjonen **`compute`** gjør (2%)

```
function c = compute(x)
    if x<10
        c = x*compute(x*2);
    else
        c = 1;
    end
end %function
```

Oppgave 2c (5%)
Hva blir returnert hvis koden **`a([2,5,3,8,6,1,7])`** som vist under blir kjørt? (3%)
Forklar med en setning hva funksjonen **`a`** gjør (2%)

```
function c = a( c )
   for b=1:length(c)
      d = b ;
      for e=b+1:length(c)
         if c(e)>c(d)
            d = e ;
         end
      end
      f = c(b) ;
      c(b) = c(d) ;
      c(d) = f ;
   end
end
```

Oppgave 3 Programmering reisetid (20%)

Skriv funksjonene slik at du kan gjenbruke dem. Hvis du ikke klarer å løse en oppgave kan du likevel gjenbruke funksjoner fra den oppgaven i en senere oppgave.

Oppgave 3a (5%)

Lag funksjonen **readTime** (ingen input-parametere) som spør brukeren om å skrive inn et klokkeslett med time, minutt og sekund angitt separat som i eksemplet under. Funksjonen skal sørge for at brukeren skriver inn et korrekt klokkeslett (man kan anta at bruker skriver inn et tall og ikke bokstaver). Den skal gi feilmelding ved feil brukerinput, og spørre brukeren å oppgi time, minutt eller sekund på nytt hvis tallet ikke er korrekt. Funksjonen skal returnere en vektor med tre elementer [hour, minute, sec].

Eksempel på kjøring og på hva som skal skrives ut til skjerm (brukerinput er vist med understrekning og fet skrift):

```
>> time = readTime()
Enter hour: 24
- ERROR: Hour must be between 0 and 23!
Enter hour: 12
Enter minute: -5
- ERROR: Minute must be between 0 and 59!
Enter minute: 34
Enter second: 65
- ERROR: Second must be between 0 and 59!
Enter second: 20
time =
    12, 34, 20
>>
```

Oppgave 3b (5%)

Lag funksjonen **convertTime** som tar inn to parametere **time** og **mode**. Parameteren **mode** kan ha to verdier: ’time’ eller ’sec’. Hvis mode har verdien ’time’, skal funksjonen konvertere tid angitt i sekunder til tid angitt som en vektor på formatet [hour, min, sec] og returnere dette. Hvis mode har verdien ’sec’, skal funksjonen returnere antall sekunder vektoren på formatet [hour, min, sec] tilsvarer. Parameteren **time** kan altså enten være et heltall eller en vektor med heltall avhengig av verdien til **mode**.

Eksempel på kjøring:

```
>> convertTime(3857, 'time')
ans =
     1     4    17
>> time = convertTime(3857,'time')
time =
     1     4    17
>> time = convertTime([1,4,17],'sec')
time =
        3857
>>
```

Oppgave 3c (5%)
Lag funksjonen **travelTime** som hverken har input parametere eller returnerer noe. Funksjonen skal først spørre brukeren om et starttidspunkt og deretter et sluttidspunkt for en reise. Funksjonen skal sørge for at det blir lagt inn gyldige tidspunkt. Hvis brukeren prøver å legge inn et sluttidspunkt som er tidligere enn starttidspunktet, skal funksjonen skrive følgende feilmelding: *"- ERROR: Arrival time must be later than Departure time"*, samt spørre brukeren om et nytt sluttidspunkt. Funksjonen trenger ikke å ta hensyn til reiser som varer over midnatt. Funksjonen skal ikke returnere noe, men skrive ut reisetid angitt i timer, minutter og sekunder som vist under.

Eksempel på kjøring (bruker-input er vist med understrekning og fet font):

```
>> travelTime()
Give departure time in hour, minute and second:
Enter hour: 26
- ERROR: Hour must be between 0 and 23!
Enter hour: 15
Enter minute: 20
Enter second: 20
Give arrival time in hour, minute and second:
Enter hour: 13
Enter minute: 15
Enter second: 39
- ERROR: Arrival time must be later than Departure time
Give arrival time in hour, minute and second:
Enter hour: 18
Enter minute: 59
Enter second: 59
Traveltime: 3 hr, 39 min, 39 sec
>>
```

Oppgave 3d (5%)
Lag funksjonen **analyzeBusRoutes** som tar inn en parameter **busTables** som er en todimensjonal tabell av heltall der hver rekke inneholder følgende: nummer på bussrute, starttidspunkt (angitt med time, minutt) og sluttidspunkt (angitt med time, minutt). Funksjonen skal ikke returnere noe, men skal skrive ut til skjerm nummer og reisetid på bussruten som tar lengst tid og nummer og reisetid på bussruten som tar kortest tid som vist i eksempel på kjøring under. Hvis det finnes flere bussruter med samme kjøretid, skal funksjonen skrive ut rutenummer på den første den finner.

Hvis man kjører funksjonen med følgende opplysninger:

- Buss nr. 1, starttid 15:00, stopptid: 15:19
- Buss nr. 3, starttid 15:32, stopptid: 16:45
- Buss nr. 4, starttid 15:45, stopptid: 16:23
- Buss nr. 5, starttid 15:55, stopptid: 16:11

vil eksempel på kjøring bli som følger:

```
>> busses=[1,15,0,15,19;
        3,15,32,16,45;
        4,15,45,16,23;
        5,15,55,16,11];
>> analyzeBusRoutes(busses)
The slowest bus route is bus nr. 3 and it takes 1 hour, 13 min.
The fastest bus route is bus nr. 5 and it takes 0 hour, 16 min.
>>
```

Oppgave 4 Programmering Sensur (40%)

Hvert år tar ca. 2000 studenter ITGK-eksamen, og mye tid går med til å sensurere og sette karakterer. Derfor er det bestemt at det skal programmeres et system som skal hjelpe til med dette. For å sikre at andre universiteter/fag også kan få nytte av programmet bør det gjøres en del generaliseringer.

Du skal lage et system for registrere sensur av eksamen. Eksamen består vanligvis av fire oppgaver som hver teller 25%, 15%, 20% og 40% av karakteren. Den andre oppgaven har tre underoppgaver, den tredje har fire og den fjerde har åtte underoppgaver. Underoppgavene skal vektes slik at hver teller et bestemt antall prosent av hundre.

Det er en fordel å gjenbruke tidligere del-oppgaver der det er mulig. Du kan anta at alle funksjonene mottar gyldige argumenter (inn-verdier), hvis ikke annet er oppgitt.

Oppgave 4 a) (5%)

Lag starten på hovedprogrammet **grades**. Starten skal bare inneholde de konstantene (variablene) som programmet trenger.

Eksempel på oppstart av programmet og hva som skrives ut:

```
>> grades
NTNU_scores =
    89    77    65    53    41     0
NTNU_letters =
    'A'    'B'    'C'    'D'    'E'    'F'
TASKS =
  Columns 1 through 8
    '1'    '2a'    '2b'    '2c'    '3a'    '3b'    '3c'    '3d'
  Columns 9 through 16
    '4a'    '4b'    '4c'    '4d'    '4e'    '4f'    '4g'    '4h'
WEIGHTS =
  Columns 1 through 12
    25     5     5     5     5     5     5     5     5     5     5     5
  Columns 13 through 16
     5     5     5     5
```

Oppgave 4 b) (5%)

Lag funksjonen **`makeArray`** som har to inn-parametere (**`numbers`** og **`texts`**), der **numbers** er ei liste av tall, og **texts** er ei liste av tegn og/eller bokstaver (begge listene har samme lengde). Funksjonen skal returnere en CellArray som inneholder all informasjonen fra de to inn-parameterne som vist i eksemplet på kjøring under.

Eksempel på kjøring av funksjonen og hva den skal returnere (med input fra Oppgave 4a)):

```
>> limitLetters = makeArray( NTNU_scores, NTNU_letters )
limitLetters =
    [89]    'A'
    [77]    'B'
    [65]    'C'
    [53]    'D'
    [41]    'E'
    [ 0]    'F'

>> weightTasks = makeArray( WEIGHTS, TASKS )
weightTasks =
    [25]    '1'
    [ 5]    '2a'
    [ 5]    '2b'
... Skipping some lines here ... Vi hopper over noen linjer her ...
    [ 5]    '4g'
    [ 5]    '4h'
```

Oppgave 4 c) (5%)

Sensor retter eksamen og setter poeng på hver oppgave med en poengsum som går fra og med 0 til og med 10, der 0 er dårligst (0% score) mens 10 er best (100% score). Sensur av en eksamen består av en vektor med ett tall for hver deloppgave, i samme rekkefølge som de kommer (dvs. `1`, `2a`, `2b`, osv.)

Skriv en funksjon **computeScore** med inn-parameteren **Points** som er en vektor med poengsummer for deloppgavene, og konstanten **WEIGHTS** fra Oppgave 4 a). Funksjonen skal regne ut en totalscore i prosent for en eksamen, basert på vektingen av oppgavene.
Eksempel på kjøring av funksjonen og hva den returnerer:

```
>> computeScore([10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10], WEIGHTS)
ans =
   100
>> computeScore([10 0 0 0 10 10 10 10 0 0 0 0 0 0 0 0], WEIGHTS)
ans =
    45
>> computeScore([5 0 0 0 10 10 10 10 0 0 0 0 0 0 0 0], WEIGHTS)
ans =
   32.5000
>> computeScore([4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4], WEIGHTS)
ans =
    40
```

Oppgave 4 d) (5%)

Skriv en funksjon **score2Letter** med to inn-parametere **scoreSum** (prosentvise total-poengsummen for en kandidat fra Oppgave 4 c)) og **limitLetters**-tabellen fra Oppgave 4b).
Funksjonen skal returnere den høyeste bokstavkarakteren som kandidaten har nok poeng til (uten avrunding). For eksempel vil 40.9 poeng gi F, mens 41.0 poeng gir E.
Eksempel på kjøring av funksjonen og hva den returnerer:

```
>> score2letter( 88.9, limitLetters )
ans =
B
```

Oppgave 4 e) (5%)

Skriv en funksjon **addCandidate** som tar inn tre parametere: **candidateNumber**, **Scores** (en vektor), og konstanten **WEIGHTS** fra Oppgave 4a). Funksjonen skal legge til følgende (adskilt med tabulator) som en linje på slutten av en fil med navn '**eksamen.txt**':

candidateNumber kommer først på linjen, så alle del-poengsummene fra **Scores**, og til slutt den prosentvise scoren med én desimals nøyaktighet og et linjeskift. Dersom fila ikke finnes, skal den opprettes. Dersom den ikke kan opprettes skal funksjonen avsluttes med en feilmelding.

Eksempel på kjøring av funksjonen og hva den returnerer:

```
>> addCandidate(12392, [10 0  0  0 10 10 10 10 10 0 0 0 0 0 0 0], WEIGHTS)
>> addCandidate(33322, [0 10 10 10 0 0 0 0 0 10 10 10 10 10 10 10], WEIGHTS)

% Etter at minnepinnen som inneholder filen er fjernet ...
>> addCandidate(12492, [0 10 10 10 0 0 0 0 0 10 10 10 10 10 10 10], WEIGHTS)
Error using addCandidate (line 4)
Kan ikke åpne filen eksamen.txt
>>

% Innholdet i filen eksamen.txt på minnepinnen er nå:
12392 10    0     0     0     10    10    10    10    10    0     ...   0     50.0
33322 0     10    10    10    0     0     0     0     0     10    ...   10    50.0
```

Oppgave 4 f) (5%)

Skriv en funksjon **readResultFile** som har en inn-parameter **filename**. Funksjonen skal lese innholdet i fila med navn **filename** som er formatert som beskrevet i Oppgave 4 e) og legge innholdet i en to-dimensjonal tabell der hver rekke inneholder kandidatnummer, alle del-poengsummene og den prosentvise scoren. Kandidatnummer og del-poengsummene skal være av typen heltall, mens den prosentvise scoren skal være av typen flyttall. Du får ikke bruke load-funksjonen i Matlab i denne deloppgaven. Dersom filen ikke finnes, skal funksjonen stoppes med en feilmelding.

Filen eksamen2.txt har følgende innhold der tallene er separert med tabulator

```
12300   0   10  10  10  0   0   0   0   0   10  10  10  10  10  10  10 50.0
44400   4   4   11  0   0   0   0   0   0   0   0   0   0   0   0   19.0
12300   9   0   0   0   10  10  10  10  10  0   0   0   0   0   0   0  47.5
```

Eksempel på kjøring med fila eksamen2.txt:

```
>> readResultFile('eksamen2.tekst')
Error using readResultFile (line 4)
Kunne ikke åpne fila
>>
>> readResultFile('eksamen2.txt')
ans =
  Columns 1 through 6
       12300            0           10           10           10            0
       44400            4            4           11            0            0
       12300            9            0            0            0           10
  Columns 7 through 12
           0            0            0            0           10           10
           0            0            0            0            0            0
          10           10           10           10            0            0
  Columns 13 through 18
          10           10           10           10           10           50
           0            0            0            0            0           19
           0            0            0            0            0         47.5
>>
```

Oppgave 4 g) (5%)

Skriv en funksjon **checkResultOK** som tar inn to parametere **filename** (navnet på fila med eksamensresultater som skal sjekkes) og **WEIGHTS** (fra Oppgave 4a)). Funksjonen skal lese inn fila og returnerer **true** hvis følgende er oppfylt:
- ingen kandidat er listet mer enn en gang i samme fil
- ingen har fått mindre enn 0 poeng eller mer enn 10 poeng på noen oppgave
- den prosentvise poengsummen for alle kandidatene er korrekt utregnet som i Oppgave 4c) over.

Funksjonen skal skrive ut feilmeldinger hvis den finner feil.

Eksempel på kjøring av funksjonen, som viser hva den skriver ut og hva den returnerer:

```
>> checkResultOK( 'eksamen.txt', WEIGHTS )
ans =
     1
```

```
>> checkResultOK( 'eksamen2.txt' , WEIGHTS )
ERROR: Candidate 44400 scores are not between 0-10!
ERROR: Candidate 44400 has wrong total score!
ERROR: Candidate 12300 appears more than once!
ans =
     0
```

Oppgave 4 h) (5%)

Skriv en funksjon **listAll** med inn-parametere **filename** og **limitLetters**. Funksjonen skal lese inn resultatene fra **filename** med samme formatet som i Oppgave 4e), og skrive ut en liste der hver linje inneholder femsifret kandidatnummer, prosentvis poengsum med ett siffers nøyaktighet og tilhørende bokstavkarakter etter reglene i Oppgave 4d). Kolonnene skal være justert vertikalt som vist her.

```
10004   7.4 F
10000  75.4 C
10098  87.4 B
10008  88.1 B
10017  94.9 A
10019 100.0 A
```

Utskriften skal være sortert etter prosentvis poengsum i stigende rekkefølge. Funksjonens returverdi skal være antallet kandidater som ble skrevet ut.

Eksempel på kjøring av funksjonen, hva den skriver ut, og hva den returnerer:

```
>> listAll('eksamen2.txt', limitLetters)
44400  19.0 F
12300  47.5 E
12300  50.0 E
ans =
     3
```

Appendix: Some useful functions

FIX Round towards zero.

FIX(X) rounds the elements of X to the nearest integers towards zero.

FLOOR Round towards minus infinity.

FLOOR(X) rounds the elements of X to the nearest integers towards minus infinity.

FCLOSE Close file.

ST = FCLOSE(FID) closes the file associated with file identifier FID, which is an integer value obtained from an earlier call to FOPEN. FCLOSE returns 0 if successful or -1 if not.

FEOF Test for end-of-file.

ST = FEOF(FID) returns 1 if the end-of-file indicator for the file with file identifier FID has been set, and 0 otherwise.
The end-of-file indicator is set when a read operation on the file associated with the FID attempts to read past the end of the file.

FGETL Read line from file, discard newline character.

TLINE = FGETL(FID) returns the next line of a file associated with file identifier FID as a MATLAB string. The line terminator is NOT included. Use FGETS to get the next line with the line terminator INCLUDED. If just an end-of-file is encountered, -1 is returned.

FIND Returns the linear indexes of non-zero elements in a matrix.

FIND([0 1 0 1 0]) returns [2 4]. If the first parameter has more than one row, a column vector containing the linear indexes of non-zero elements are returned. An optional second parameter set the maximum number of indexes to return.

FOPEN Open file.

FID = FOPEN(FILENAME,PERMISSION) opens the file FILENAME in the mode specified by PERMISSION:

'r'	open file for reading
'w'	open file for writing; discard existing contents
'a'	open or create file for writing; append data to end of file
'r+'	open (do not create) file for reading and writing
'w+'	open or create file for reading and writing; discard existing contents
'a+'	open or create file for reading and writing; append data to end of file

FPRINTF Write formatted data to file.

COUNT = FPRINTF(FID,FORMAT,A,...) formats the data in the real part of array A (and in any additional array arguments), under control of the specified FORMAT string, and writes it to the file associated with file identifier FID. COUNT is the number of bytes successfully written. FID is an integer file identifier obtained from FOPEN. It can also be 1 for standard output (the screen) or 2 for standard error. If FID is omitted, output goes to the screen.

FORMAT is a string containing ordinary characters and/or C language conversion specifications. Conversion specifications involve the character %, optional flags, optional width and precision fields, optional subtype specifier, and conversion characters d, i, o, u, x, X, f, e, E, g, G, c, and s.

The special formats \n,\r,\t,\b,\f can be used to produce linefeed, carriage return, tab, backspace, and formfeed characters respectively. Use \\ to produce a backslash character and %% to produce the percent character.

INPUT Read a value from the keyboard and into a variable
ANSWER=INPUT(STR) prints STR as a prompt, reads a number and assigns it to ANSWER. If character string are to be read, use the optional second parameter 's'.

ISEMPTY - Determine whether array is empty
This MATLAB function returns logical 1 (true) if A is an empty array and logical 0 (false) otherwise.
TF = isempty(A)

LENGTH The length of vector.
LENGTH(X) returns the length of vector X. It is equivalent to MAX(SIZE(X)) for non-empty arrays and 0 for empty ones.

LOAD Loads data from filename.
load(filename) loads data from filename. If filename is a MAT-file, then load(filename) loads variables in the MAT-File into the MATLAB® workspace. If filename is an ASCII file, then load(filename) creates a double-precision array containing data from the file.

MAX finds the highest element in a vector, or the highest element in each column of a matrix.

MIN finds the lowest element in a vector, or the lowest element in each column of a matrix.

MOD Modulus after division.
MOD(x,y) is x - n.*y where n = floor(x./y) if y ~= 0.

RANDI Pseudorandom integers from a uniform discrete distribution.
R = RANDI(IMAX,N) returns an N-by-N matrix containing pseudorandom integer values drawn from the discrete uniform distribution on 1:IMAX.
RANDI(IMAX,M,N) or RANDI(IMAX,[M,N]) returns an M-by-N matrix.

REM Remainder after division.
REM(x,y) is x - n.*y where n = fix(x./y) if y ~= 0.

ROUND Rounds to nearest decimal or integer
Y = round(X) rounds each element of X to the nearest integer. If an element is exactly between two integers, the round function rounds away from zero to the integer with larger magnitude.
Y = round(X,N) rounds to N digits

SIZE The size of array.
D = SIZE(X), for M-by-N matrix X, returns the two-element row vector
D = [M,N] containing the number of rows and columns in the matrix.

SORTROWS Sort array rows
This MATLAB function sorts the rows of A in ascending order, based on column.
B = sortrows(A)
B = sortrows(A,column)

SQRT Square root.

SQRT(X) is the square root of the elements of X.

SSCANF Extracts values from a string according to a format string. Opposite of FPRINTF.
A=SSCANF('12/11-2014','%d/%d-%d') returns a column vector containing the values 12, 11, and 2014.

STRSPLIT Splits the first (string) parameter into a cell array of substrings, according to the delimiter string given as the second parameter. STRSPLIT('one, two, three', ', ') results in {'one', 'two', 'three'}. Multiple alternative delimiters can be specified using a cell array as the second parameter.

STRTOK separates the first token of a string from the rest of that string.
[TOKEN, REST]=STRTOK(' first second', DELIM) sets TOKEN to 'first' and REST to ' second'. The optional parameter DELIM contains a list of delimiter characters – where the space character is default. Any delimiter characters before the first token are ignored.

STR2NUM Convert string matrix to numeric array.
X = STR2NUM(S) converts a character array representation of a matrix of numbers to a numeric matrix. For example, S=['12'; '34'] str2num(S) => [12; 34]

SUM The sum of elements.
S = SUM(X) is the sum of the elements of the vector X. If X is a matrix, S is a row vector with the sum over each column.

Denne siden er med hensikt blank!

Svarskjema flervalgsoppgave

Kandidatnummer: _____ Program: _____

Fagkode: _____ Dato: _____

Antall sider: _____ Side: _____

Oppgavenr	***A***	***B***	***C***	***D***
1.1				
1.2				
1.3				
1.4				
1.5				
1.6				
1.7				
1.8				
1.9				
1.10				
1.11				
1.12				
1.13				
1.14				
1.15				
1.16				
1.17				
1.18				
1.19				
1.20				

Denne siden er med hensikt blank!

Svarskjema flervalgsoppgave

Kandidatnummer: _____ Program: _____

Fagkode: _____ Dato: _____

Antall sider: _____ Side: _____

Oppgavenr	*A*	*B*	*C*	*D*
1.1				
1.2				
1.3				
1.4				
1.5				
1.6				
1.7				
1.8				
1.9				
1.10				
1.11				
1.12				
1.13				
1.14				
1.15				
1.16				
1.17				
1.18				
1.19				
1.20				