Norges teknisk–naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

TDT4105 IT Grunnkurs
Høst 2014

Øving 5

1 Teori

a) Hva er den binære ASCII-verdien av bokstaven E (stor e)? Hva er den i titallssystemet?

b) Bruk ASCII tabellen til å oversette følgende til bokstaver:
01001101 01000001 01010100 01001100 01000001 01000010

c) Hva er metadata?

2 Månedskalender

I denne øvingen skal du implementere funksjonen `printCalendar`. Funksjonen skal vise en gyldig månedskalender, gitt et månednummer og et år som parameter. Her er et forslag til utskrift for oktober 2014:

```
   Oktober 2014
ma ti on to fr lø sø
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

For å kunne skrive denne funksjonen trenger vi å definere flere hjelpefunksjoner som løser delproblemer, på veien til den endelige løsningen. Hvis du vil ha en ekstra utfordring, kan du prøve å dele opp problemet selv, og løse øvingen uten å se på resten av oppgavene. Men du må uansett gjøre oppgave 3a. Når du gjør denne øvingen, kan det være smart å se på kalender.no for å kunne sjekke verdiene du får ut av funksjonene du lager. For å løse oppgaven er det som nevnt viktig å dele opp problemet. Hvis man ser på utskriften kan man se at denne har tre deler:

1. Månedsnavn og årstall (første rad)
2. Dagsnavn (andre rad)
3. Datoer for alle dager på riktig plass (tredje til syvende rad)

Punkt 1) her er ganske greit. Tatt parameterene månedsnummer og år i betraktning, trenger man her kun å gjøre et månedsnummer om til et månedsnavn, og så kan `printCalendar` skrive ut resultatet.

a) I filen `getMonthName.m` finner du hjelpefunksjonen `getMonthName`, som tar inn et månedsnummer som parameter og returnerer navnet på måneden.

Skriv et skript som tester denne funksjonen, kalt `getMonthNameTest.m`. (Vedlegget viser en litt mer avansert måte å teste funksjoner på)

For punkt 2) trenger vi ikke å lage noen hjelpefunksjoner, siden dagsnavnene alltid er de samme.

3 Kodeforståelse

På punkt 3) er det desto mer jobb som må gjøres. Ved å se på utskriften kan man legge merke til at den ser ut som en matrise med tall. Dette er en fin måte å representere datoene på, og vi velger en 7x6 matrise hvor kolonnene representerer ukedager og radene uker. For eksempel slik for oktober 2014:

```
month =
     0     0     1     2     3     4     5
     6     7     8     9    10    11    12
    13    14    15    16    17    18    19
    20    21    22    23    24    25    26
    27    28    29    30    31     0     0
```

Tallet 0 representerer dager som ikke finnes i måneden.

Neste steg er å definere en funksjon som lager denne matrisen. Det gjør følgende funksjon.

```
function calendar = getCalendar( year, month )

    startDay = getMonthStartDay( year, month );
    numDays = daysInMonth( year, month );

    calendar = zeros( 6, 7 );

    date = 0;

    for week = 1:6

       for day = 1:7

            if ( date == 0 )

                if ( week == 1 && day == startDay )
                    date = 1;
                end

            else

                if ( date < numDays )
                    date = date + 1;
                else
                    date = 0;
                end

            end

            calendar(week, day) = date;

       end

    end

end
```

Her returnerer funksjonen `daysInMonth` antall dager i måned, `getMonthStartDay` returnerer nummeret på den første ukedagen i måneden (1 = Mandag, 2 = Tirsdag, ..., 7 = Søndag). I eksempelet med oktober 2014, returnerer `daysInMonth` 31 og `getMonthStartDay` 3.

a) Gitt at startDay = 3 og numDays = 31. Gå gjennom funksjonen over og skriv ned verdien til variablene `week`, `day` og `date` for hver iterasjon i den indre for-løkken.

Iterasjon	week	day	date
1	1	1	0
2	...	...	...
...	...	...	...
42	...	...	...

4 Hjelpefunksjoner

Legg inn funksjonen `getCalendar` (ligger vedlagt i filen getCalendar.m). For at denne funksjonen skal kunne kjøre, trenger vi altså å definere funksjonene `daysInMonth` og `getMonthStartDay`. I øving 3 lagde du førstnevnte funksjon, så det skal du slippe å gjøre igjen. For å løse denne oppgaven må den ta hensyn til skuddår. Du kan bruke filene `daysInMonth.m` og `isLeapYear.m`

For å kunne definere funksjonen `getMonthStartDay` trenger vi først å vite hvilken dag året starter på.

a) Lag funksjonen `getYearStartDay`. Funksjonen skal ta inn et årstall som parameter og returnere hvilken ukedag året starter på (1 = Mandag, 2 = Tirsdag, ..., 7 = Søndag).

Tips: Året 1900 startet på en mandag. For hvert år flytter startdagen seg med en dag. Det vil si at 1901 startet på en tirsdag. Hvis året er et skuddår, flytter startdagen i neste år seg med to dager.

For å teste denne funksjonen kan du kjøre følgende skript:

```
getYearStartDay(1900) % skal skrive ut 1
getYearStartDay(1901) % skal skrive ut 2
getYearStartDay(1902) % skal skrive ut 3
getYearStartDay(1903) % skal skrive ut 4
getYearStartDay(1904) % skal skrive ut 5
getYearStartDay(1905) % skal skrive ut 7
getYearStartDay(1906) % skal skrive ut 1
getYearStartDay(2012) % skal skrive ut 7
```

b) Lag funksjonen `getMonthStartDay`. Den skal ta inn et årstall og et månedsnummer som parameter (1 = Januar, 2 = Februar, ..., 12 = Desember). Funksjonen skal returnere hvilken ukedag måneden starter på (1 = Mandag, 2 = Tirsdag, ..., 7 = Søndag).

```
getMonthStartDay(2012, 0) % skal skrive ut 0
getMonthStartDay(2012, 1) % skal skrive ut 7
getMonthStartDay(2012, 2) % skal skrive ut 3
getMonthStartDay(2012, 3) % skal skrive ut 4
getMonthStartDay(2012, 4) % skal skrive ut 7
getMonthStartDay(2012, 5) % skal skrive ut 2
getMonthStartDay(2012, 6) % skal skrive ut 5
getMonthStartDay(2012, 7) % skal skrive ut 7
getMonthStartDay(2012, 8) % skal skrive ut 3
getMonthStartDay(2012, 9) % skal skrive ut 6
getMonthStartDay(2012, 10) % skal skrive ut 1
getMonthStartDay(2012, 11) % skal skrive ut 4
getMonthStartDay(2012, 12) % skal skrive ut 6
getMonthStartDay(2012, 13) % skal skrive ut 0
```

5 Utskriftfunksjoner

Nå har vi laget alle funksjonene vi trenger for å regne ut riktige verdier. Da er det bare utskrift som er igjen.

For å ha god kontroll på hvordan utskriften ser ut, bruker vi `fprintf`, som gir oss mulighet til å formatere den akkurat som vi vil.

Man kaller på funksjonen slik:

```
a = 1; % en variabel med et heltall
b = 2; % en variabel med et heltall

fprintf('dette er et tall: %d, det er dette ogsaa: %d\n', a, b);
```

Denne koden skriver ut 'dette er et tall: 1, det er dette ogsaa: 2' og et linjeskift til skjermen.

- %d betyr: her skal det settes inn et tall.
- \n betyr: her skal det settes inn et linjeskift.

Det må være like mange %d tegn i kontrollstrengen (parameter 1) som det er parametre etterpå. I eksempelet er det to %d og to ekstra parameter.

Siden det å skrive ut punkt 3 er litt mer komplisert enn å skrive ut punkt 1 og 2, lønner det seg å skille dette ut i en egen funksjon.

a) Lag funksjonen `printDays`. Den skal ta inn matrisen som funksjonen i oppgave 3 lager som parameter, og skrive ut datoene som vist i toppen av øvingen. Denne funksjonen skal ikke returnere noen verdi. Funksjonshodet kan dermed se slik ut:

```
function printDays( days )

end
```

b) Lag funksjonen `printCalendar`. Denne skal ta inn år og månedsnummer som parameter, og kalle på `getMonthName` og `getCalendar`. Deretter skal den skrive ut de to første linjene av kalenderen og kalle `printDays` slik at den skriver ut resten. Denne funksjonen skal ikke returnere noe.

6

Du er som alle rasjonelle (?) mennesker meget bekymret hver gang det er fredag 13. Du ønsker derfor å lage en funksjon som sjekker hvilke måneder i løpet av et gitt år som inneholder en fredag 13. Funksjonen skal ta inn et årstall og returnere en liste med 12 verdier, én for hver måned, som angir hvorvidt den måneden inneholder en fredag 13.

For eksempel skal `hasFriday13(2014)` returnere følgende liste: `[0 0 0 0 0 1 0 0 0 0 0 0]`, siden 13. juni 2014 er en fredag (6. elementet i listen). Her angir altså 0 og 1 den boolske verdien `false` og `true`, henholdsvis.

1 Vedlegg: Testing (ikke pensum)

En mer grundig og *enklere* måte å teste en funksjon på er ved å bruke funksjonen `assert`. Den har blandt annet følgende funksjonshoder:

```
assert(utrykk);
assert(utrykk, feilmelding);
```

Hvor `utrykk` er et boolsk utrykk (altså noe som er `true` eller `false`) og `feilmelding` er en tekststreng som forteller hva som er feil.

Assert er ikke som vanlige funksjoner. Dens oppgave er å sjekke om `utrykket` er `true`, hvis ikke stopper den programmet med en gang og skriver ut feilmeldingen. Den kan for eksempel brukes slik:

```
assert( 1 + 1 == 2, ' 1 + 1 er ikke lik 2');
```

Tankegangen her er at hvis 1 + 1 ikke er lik 2 så kommer sannsynligvis ikke resten av programmet til å oppføre seg som det skal, så det er best avslutte med en gang og gi beksjed om hva som er feil.

Dette kan vi bruke på følgende måte: Gitt funksjonen i oppgave 2a, hvis den ikke returnerer riktig navn for alle månedene så kommer ikke `printCalendar` til å vise riktig månedsnavn heller. Da kan vi lage et skript med følgende setninger.

```
assert( strcmp( getMonthName(0), 'Ikke en gyldig maaned'),
  'getMonthName(0) skriver ikke ut riktig verdi');
assert( strcmp( getMonthName(1), 'Januar' ),
  'getMonthName(1) skriver ikke ut Januar' );
assert( strcmp( getMonthName(2), 'Februar' ),
  'getMonthName(2) skriver ikke ut Februar' );
assert( strcmp( getMonthName(3), 'Mars' ),
  'getMonthName(3) skriver ikke ut Mars' );
assert( strcmp( getMonthName(4), 'April' ),
  'getMonthName(4) skriver ikke ut April' );
assert( strcmp( getMonthName(5), 'Mai' ),
  'getMonthName(5) skriver ikke ut Mai' );
assert( strcmp( getMonthName(6), 'Juni' ),
  'getMonthName(6) skriver ikke ut Juni' );
assert( strcmp( getMonthName(7), 'Juli' ),
  'getMonthName(7) skriver ikke ut Juli' );
assert( strcmp( getMonthName(8), 'August' ),
  'getMonthName(8) skriver ikke ut August' );
assert( strcmp( getMonthName(9), 'September' ),
  'getMonthName(9) skriver ikke ut September' );
assert( strcmp( getMonthName(10), 'Oktober' ),
  'getMonthName(10) skriver ikke ut Oktober' );
assert( strcmp( getMonthName(11), 'November' ),
  'getMonthName(11) skriver ikke ut November' );
assert( strcmp( getMonthName(12), 'Desember' ),
  'getMonthName(12) skriver ikke ut Desember' );
assert( strcmp( getMonthName(13), 'Ikke en gyldig maaned'),
  'getMonthName(13) skriver ikke ut riktig verdi');
```

Her brukes `strcmp` for å sammenligne to tekststrenger, den returnerer `1` (altså `true`) hvis tekststrengene er identiske og `0` hvis ikke.

Det fine med denne måten å teste på, er at når man er ferdig med å skrive testen, så slipper man den manuelle delen ved å teste. Skriptet kan kjøres for eksempel hver gang du gjør en liten endring i funksjonen. På den måten får du direkte tilbakemelding på hva som fungerer og ikke i funksjonen. Den gir i tillegg tilbakemelding kun på hva som er feil og ikke hva som er riktig.

En enkel måte å utvide funksjonaliteten til `assert` på er å skrive `assertEqual`. (Denne versjonen fungerer kun for tall)

```
function assertEqual(actual, expected)

  assert(actual == expected,
    'Expected %d to be equal to %d', actual, expected);

end
```

Da kan `isLeapYear` testes slik:

```
assertEqual(isLeapYear(2000), 1);
assertEqual(isLeapYear(1900), 0);
assertEqual(isLeapYear(2004), 1);
assertEqual(isLeapYear(2001), 0);
```

Intill videre kan det være greit å lage *et* skript som tester *en* funksjon, og hvis funksjonen ligger i filen foo.m så kan testene ligge i filen fooTest.m. Du kan nå skrive om all testkoden i denne øvingen ved hjelp av denne metoden.