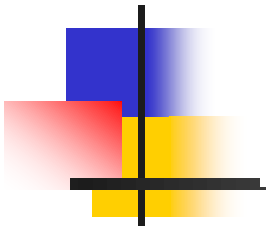


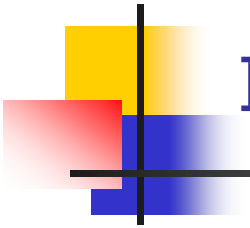
INF 1000 – høsten 2011
Uke 11: 2. november

Grunnkurs i Objektorientert Programmering

Institutt for Informatikk

Universitetet i Oslo

Kursansvarlige: Arne Maus og Siri Moe Jensen



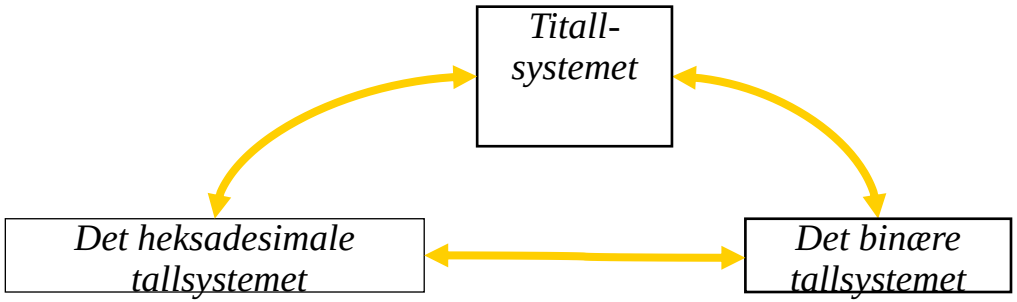
Info

- Obligene skal være kommentert, her blir kravene skjerpet i oblig 4
  - Se f eks guide under ”Ressurser” på kurssiden (nb eksempler i Phyton, prinsippene uavhengig av språk)
- Prøve-eksamen planlagt 22. november, 12-18.

  Mer info når alt er på plass, følg med på kursside og blog
- Oblig-status skal nå være oppdatert i Devilry
  - http://devilry.ifi.uio.no/
  - Gi beskjed til gruppelærer v/ problemer eller feil

Innhold – uke 11

- Binære og heksadesimale tall, konvertering mellom tallsystemer:



- Prinsippene for hvordan tegn og tekst kan representeres ved hjelp av bits og bytes, noen sentrale standarder:
  - ASCII, ISO 8859, Unicode

- Deler av eksamensoppgave

Mål for uke 11:

** Forstå typiske utfordringer og løsninger for digital representasjon av ulike typer data*

** Mer erfaring med programmering og eksamensoppgaver*

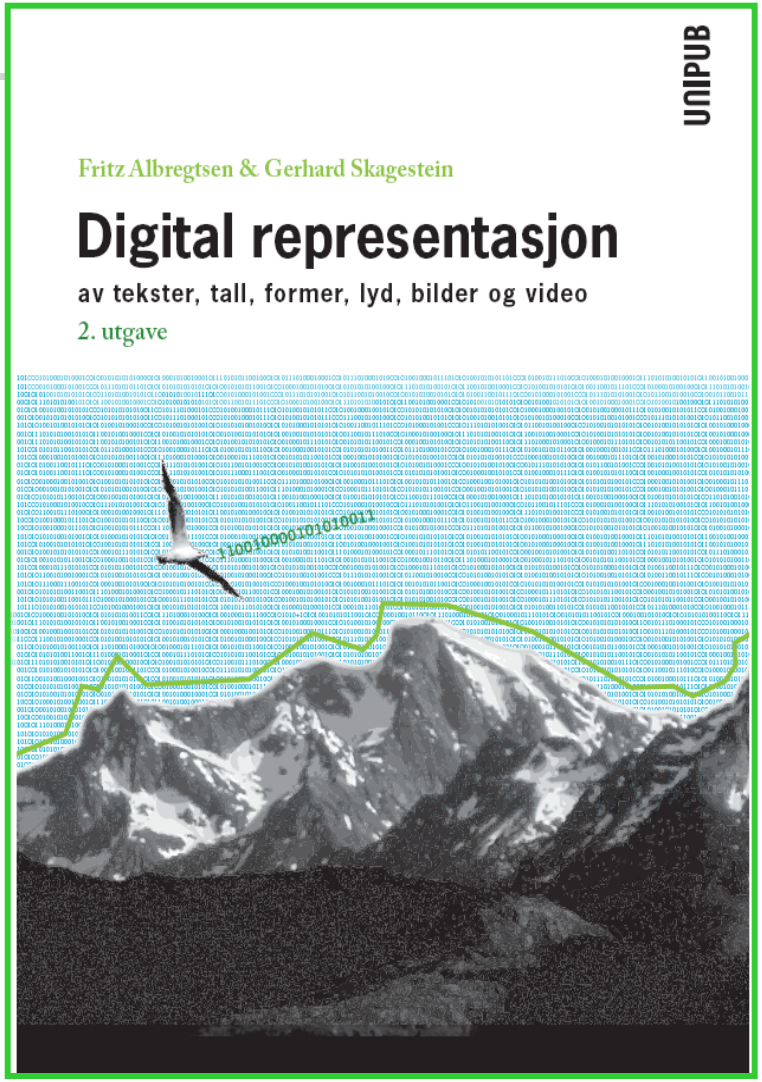
Hovedkilde

Fritz Albregtsen &
Gerhard Skagestein:

Digital representasjon
av tekster, tall, former,
lyd, bilder og video

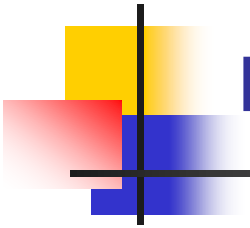
Kapittel 1,2,6 og 7.

(Lærebok i tidligere INF1040)



Hva jobber datamaskiner *egentlig* med?

- I våre dager: Digitale datamaskiner
  - Digital (fra Bokmålsordboka): *..som gjengir fysiske størrelser med diskrete tegn (oftest siffer)*
- Baserer all lagring og prosessering av data på det binære tallsystemet med kun to siffere: 0 og 1
- 0 og 1 representeres på laveste nivå gjerne som spenning/ ikke spenning. Eller enda mer nøyaktig:
  - 0-0.8 volt = ”lav” = 0
  - 2-5 volt = ”høy” = 1
- Også lys, magnetisme, hull/ ikke hull, ….



Datamaskinverdenen er binær digital

- 2 diskrete verdier, 0 og 1
- 0 og 1 kalles binære sifre – binary digits – bits

- Alt i datamaskinen er representert ved sekvenser av bits – bitmønstre

- Moderne datamaskiner arbeider gjerne med grupper på 8 bits
  - En slik gruppe på 8 bits kalles en byte.

Hva betyr bitmønsteret?

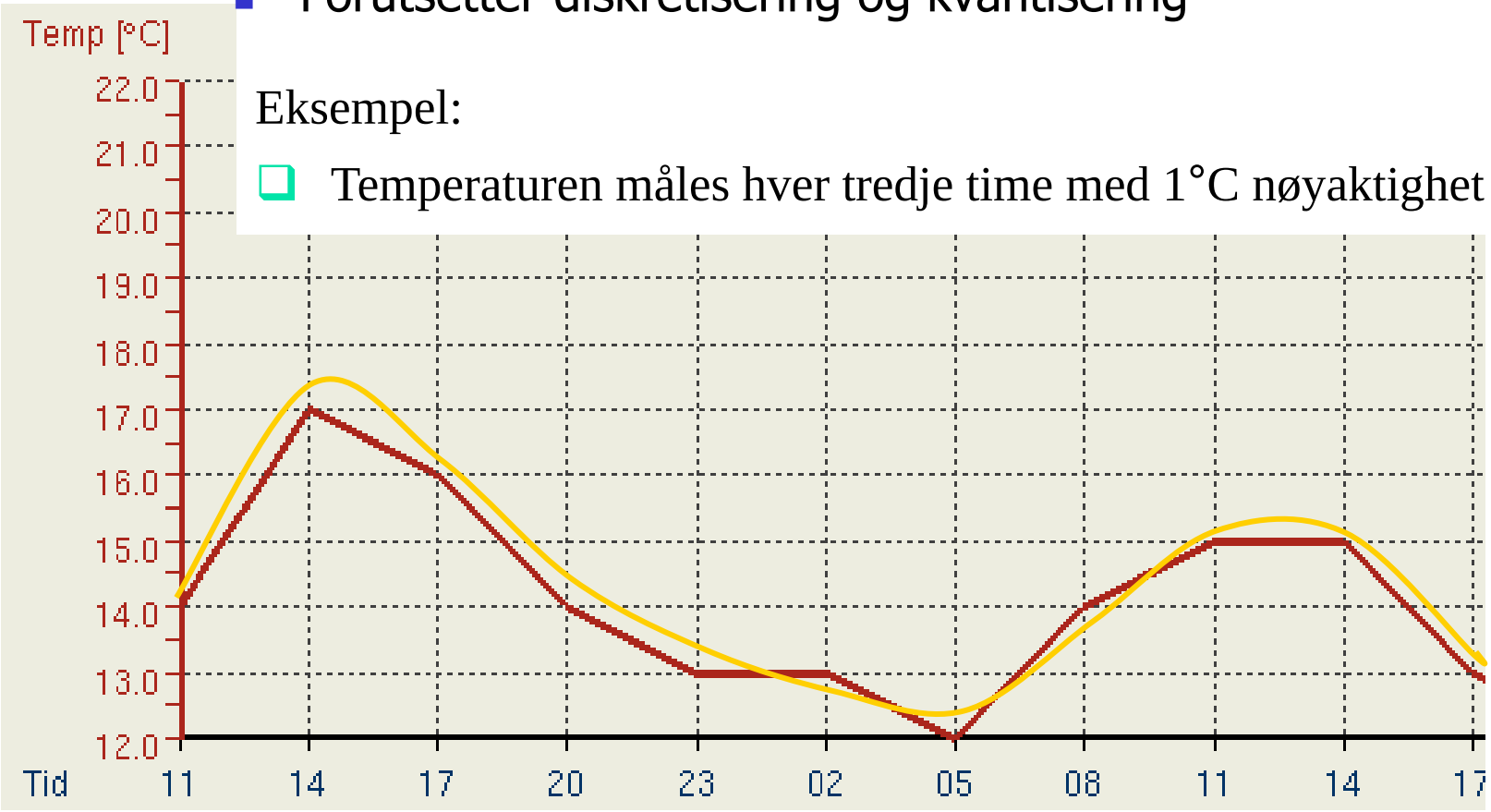
- Bitmønsteret 10100100 kan bl.a. være:
  - 164 (tolket som binærtall)
  - -36 (negativt binærtall med fortegnsbit)
  - -90 (negativt tall på toerkomplementsform)
  - € (ISO 8859-15)
  - ¤ (Windows Codepage 1252)
  -  (som gråtone)
  - …

Analog virkelighet – digital representasjon

- Analog: ”..basert på fysiske, kontinuerlig varierbare størrelser”
- Digital: ”..som gjengir fysiske størrelser med diskrete tegn”
- Forutsetter diskretisering og kvantisering

Eksempel:

- Temperaturen måles hver tredje time med 1°C nøyaktighet



BINÆRTALL - TALLSYSTEMER

Potensregning – kort repetisjon

- $\text{grunntall}^{\text{eksponent}} = \underbrace{\text{grunntall} * \text{grunntall} * \ldots * \text{grunntall}}_{\text{eksponent antall ganger}}$

- Spesialregel: $\text{grunntall}^0 = 1$

- Eks:
  - $10^3 = 10*10*10 = 1000$
  - $2^4 = 2 * 2 * 2 * 2 = 16$
  - $5^0 = 1$

Hva er et tallsystem?

- Trenger noen symboler (siffere), og en mekanisme for å kombinere disse slik at vi kan håndtere ”uendelig” store tall
- I vårt (desimale, dekadiske) titallsystem har vi 10 siffere (0-9), og bruker sammensetninger av disse for å lage tall større enn 9 vha mekanismen *posisjonssystemet*.

$$19 = 1*10^1 + 9*10^0$$

- Det binære tallsystem (totallsystemet) har bare 2 siffere – men bruker samme mekanisme (posisjonssystemet) for å håndtere tall større enn 1

$$1\ 0\ 1\ 0\ 1 = 1*2^4 + 1*2^2 + 1*2^0$$

- Eksempel på en annen mekanisme: Romertall. Har symboler som I, V, X og L – her omfatter reglene for å kombinere til nye tall bla. subtraksjon.

$$X\ I\ X = 10 + (10-1)$$

Titallsystemet – et posisjonssystem

- I titallsystemet (desimalsystemet) har vi
  - Grunntall 10
  - 10 sifre: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

- Større tall konstrueres ved hjelp av posisjonssystemet:

10^6 (1 000 000)	10^5 (100 000)	10^4 (10 000)	10^3 (1 000)	10^2 (100)	10^1 (10)	10^0 (1)

Det binære tallsystemet

- Det binære tallsystemet (totallsystemet) har
  - grunntall 2
  - 2 sifre: 0 og 1

2^7 (128)	2^6 (64)	2^5 (32)	2^4 (16)	2^3 (8)	2^2 (4)	2^1 (2)	2^0 (1)

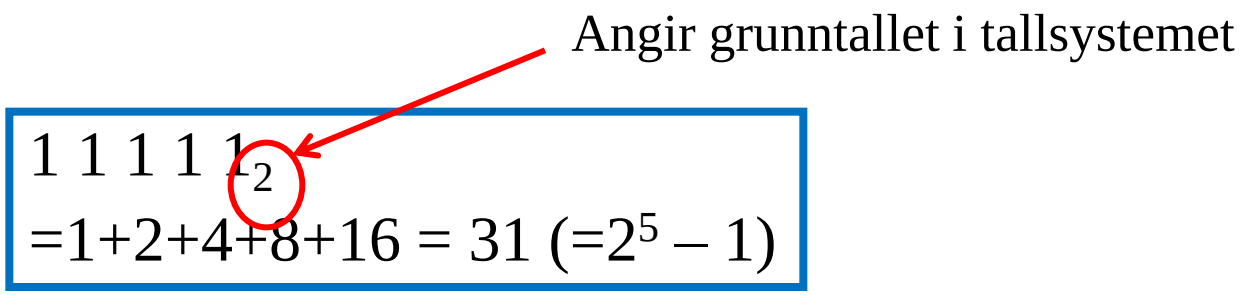
Titallsystemet → totallsystemet (binærtall)

- Gitt et tall x i titallsystemet.
  - Start på posisjon 0.
  - Hvis x er oddetall, sett 1 i denne posisjonen (ellers 0).
  - Sett x lik x heltallsdividert med 2.
  - Fortsett med neste posisjon til venstre, så lenge x ≠ 0.

```
int[] tilBintall(int x) {
    int[] bintall = new int[8];

    int pos = 0;


    do {
        if (x % 2 == 1) {
            bintall[pos] = 1;
        } else {
            bintall[pos] = 0;
        }
        x = x/2;
        pos++;

    } while (x != 0);

    return bintall;
}
```

Bitposisjoner og bitmønstre

- For et tall x i titallsystemet, hvor mange sifre (bits) trenger det tilsvarende binærtallet?
- Det største tallet som kan representeres med b bits er $2^b - 1$.
- Vi må altså finne den minste b'en slik at $x \leq 2^b - 1$.



Det heksadesimale tallsystemet

- I det heksadesimale tallsystemet har vi
  - grunntall 16
  - 16 sifre: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

16^5 (1 048 576)	16^4 (65 536)	16^3 (4 096)	16^2 (256)	16^1 (16)	16^0 (1)

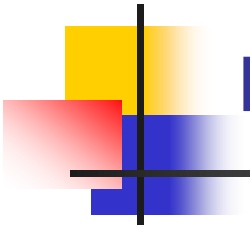
Konvertering binært ↔ heksadesimalt

Binært	Heksadesimalt
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7

Binært	Heksadesimalt
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

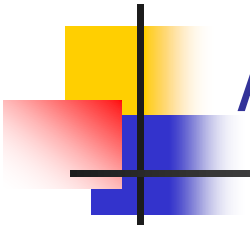
- For å angi at noe er skrevet på heksadesimal form kan vi enten
  - føye til 16 som subskript, f.eks. $A1_{16}$, eller
  - føye til 0x i forkant, f.eks. 0xA1

TEGN OG TEKSTER



Problemstilling

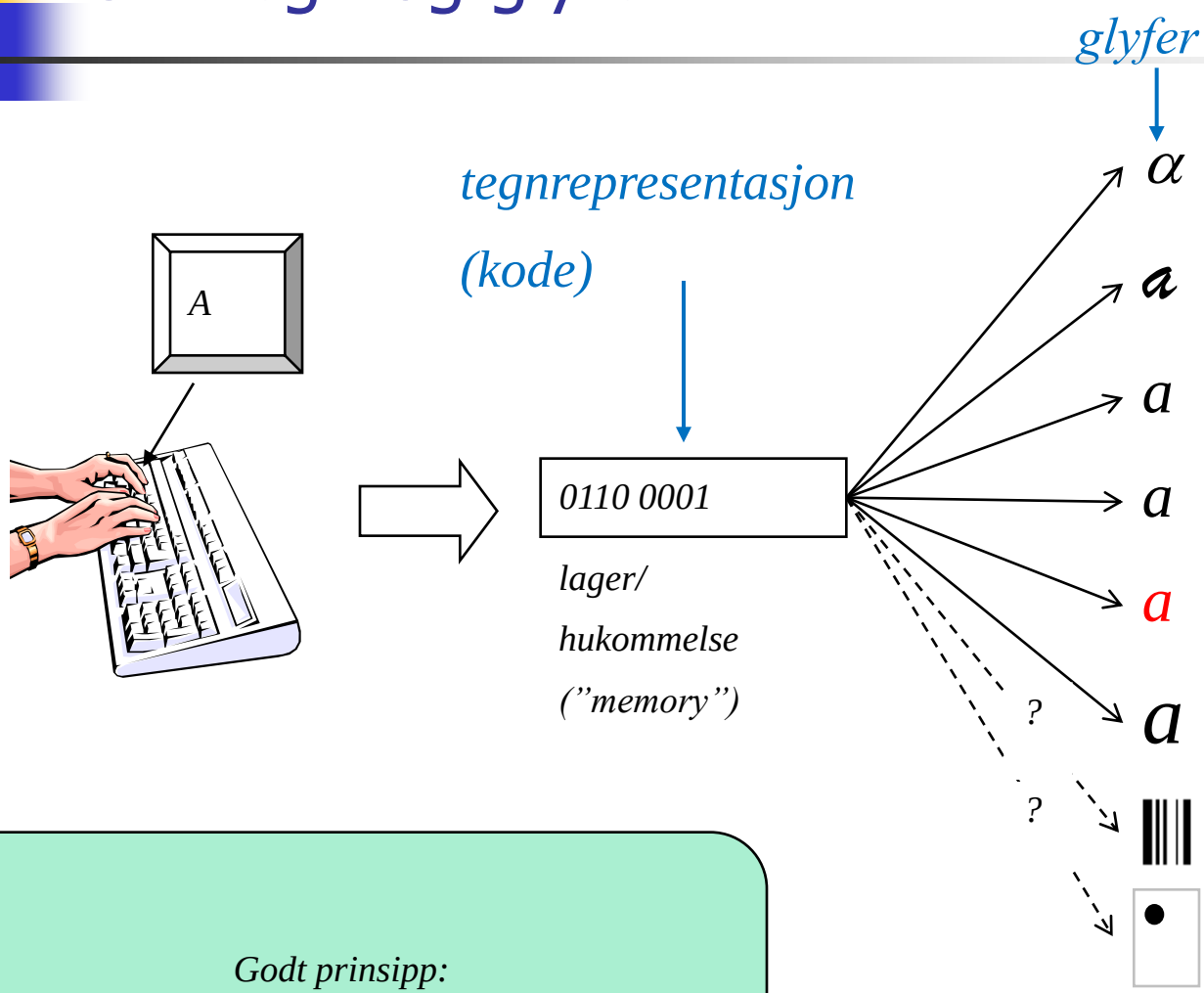
- Hvert tegn i teksten representeres av et unikt bitmønster.
- Eksempel:
  - E = 01000101
  - H= 01001000
  - I = 01001001
  - HEI = 01001000 01000101 01001001
- Sender (skriver) og mottaker (leser) må være enige om kodingen.
  - Vi trenger standarder!



Aktuelle spørsmål

- Hvilke tegn skal representeres?
- Hvor mange bits per tegn?
- Fast eller variabelt antall bits per tegn?
- Bør det være noen form for systematikk i bitmønstrene?
- Ulike svar i ulike standarder – gis i *kodetabeller*

Om tegn og glyfer



Godt prinsipp:

Skill representasjon fra fremvisning (”rendering”)

Kodetabeller

- Hvert tegn som inngår i tegnsettet tilordnes et *kodepunkt* (ofte angitt på heksadesimal form)
  - Tegnets ”numeriske” verdi
  - Det som står i kodetabellen
- Representasjon:
  - Hvordan tegnet representeres som et bitmønster

adderes for å få kodepunktet

$=25_{16}$

+	00	10	20	30	40	50	60	70
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x

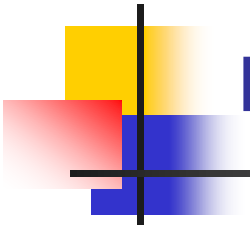
glyf = %

<utsnitt av ascii-tabell>

ASCII kodetabell

- American Standard Code for Information Interchange
- *1963 →*

	00	10	20	30	40	50	60	70
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	”	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL



Linjeskift: LF og CR

- LF (Line Feed):
  - 0x0A
  - "Indicates movement of the printing mechanism or display cursor to the next line."
- CR (Carriage Return):
  - 0x0D
  - "Indicates movement of the printing mechanism or display cursor to the starting position of the same line."
- Linjeskift representeres i dag på ulike måter:
  - PC: CR + LF
  - Mac: CR
  - UNIX: LF

ISO 646-60 kodetabell

- Bygger på ASCII.
- *[|] { | } er ofret til fordel for Æ Ø Å æ ø å*
- Lignende tilpasninger er gjort i tilsvarende standarder for andre språkmiljøer.

	00	10	20	30	40	50	60	70
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	”	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	Æ	k	æ
C	FF	FS	,	<	L	Ø	l	ø
D	CR	GS	-	=	M	Å	m	å
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

ISO 8859-1 (Latin-1) kodetabell

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
0	NUL	DLE	space	0	@	P	`	p			no break space	°	À	Đ	à	ð
1	SOH	DC1	!	1	A	Q	a	q			¡	±	Á	Ñ	á	ñ
2	STX	DC2	”	2	B	R	b	r			¢	²	Â	Ò	â	ò
3	ETX	DC3	#	3	C	S	c	s			£	³	Ã	Ó	ã	ó
4	EOT	DC4	$	4	D	T	d	t			¤	´	Ä	Ô	ä	ô
5	ENQ	NAK	%	5	E	U	e	u			¥	µ	Å	Õ	å	õ
6	ACK	SYN	&	6	F	V	f	v	*un-defined*		¦	¶	Æ	Ö	æ	ö
7	BEL	ETB	'	7	G	W	g	w			§	·	Ç	×	ç	÷
8	BS	CAN	(	8	H	X	h	x			¨	¸	È	Ø	è	ø
9	HT	EM	)	9	I	Y	i	y			©	¹	É	Ù	é	ù
A	LF	SUB	*	:	J	Z	j	z			ª	º	Ê	Ú	ê	ú
B	VT	ESC	+	;	K	[	k	{			«	»	Ë	Û	ë	û
C	FF	FS	,	<	L	\	l	|		¬	¼	Ì	Ü	ì	ü
D	CR	GS	-	=	M	]	m	}			-	½	Í	Ý	í	ý
E	SO	RS	.	>	N	^	n	~			®	¾	Î	Þ	î	Þ
F	SI	US	/	?	O	_	o	DEL			¯	¿	Ï	ß	ï	Ÿ

ETSI GSM 03.38 kodetabell for SMS

	00	10	20	30	40	50	60	70
0	@	Δ	space	0	¡	P	¿	p
1	£	_	!	1	A	Q	a	q
2	$	Φ	"	2	B	R	b	r
3	¥	Γ	#	3	C	S	c	s
4	è	Λ	¤	4	D	T	d	t
5	é	Ω	%	5	E	U	e	u
6	ù	Π	&	6	F	V	f	v
7	ì	Ψ	'	7	G	W	g	w
8	ò	Σ	(	8	H	X	h	x
9	Ç	Θ	)	9	I	Y	i	y
A	LF	Ξ	*	:	J	Z	j	z
B	Ø	ESC	+	;	K	Ä	k	ä
C	ø	Æ	,	<	L	Ö	l	ö
D	CR	æ	-	=	M	Ñ	m	ñ
E	Å	undef	.	>	N	Ü	n	ü
F	å	É	/	?	O	§	o	à

...pluss disse 10 "escape"-sekvensene

€	ESC e
FF	ESC LF
[	ESC <
\	ESC /
]	ESC >
^	ESC Λ
{	ESC (
|	ESC @
}	ESC)
~	ESC =

Den endelige løsning? Unicode og ISO 10646

- 17 ”plan” med 256 *256 kodepunkter i kodetabellen
  - Ca 130 000 private
  - Ca 870 000 ennå ikke brukt
- Kodepunktet representeres med 21 bit, evt innledende 0’er
- Første 256 tegn identisk med ISO 8859-1
- For hvert tegn finnes:
  - en representativ glyf
  - kodepunktet
  - et navn
  - klassifisering
  - skriveretning
- Vedtatte tegn med kodepunkter skal aldri endres

se http://www.unicode.org/charts/

String-metoden compareTo i Java

- Basert på Unicode-verdiene til hvert enkelt tegn i de to tekstene:

```
void sammenlign(String s1, String s2) {
    if (s1.compareTo(s2) < 0) {
        System.out.println(s1 + " " + s2);
    } else if (s1.compareTo(s2) = 0) {
        System.out.println("Like!");
    } else { // s1.compareTo(s2) > 0
        System.out.println(s2 + " " + s1);
    }
}
```

NB: Hva skjer hvis tekstene inneholder de norske tegnene æøå?

Representasjon av kodepunkter i Unicode

- I Unicode representeres ikke de 21 biters kodepunktene direkte, men styres av kodinger:

  Unicode Transformation Formats (UTF)

- UTF-8, UTF-16 og UTF-32: Tallet angir *minimum* antall bit som benyttes til representasjonen

- UTF-8: ”litt komplisert, men genial”
  - 8, 16, 24 eller 32 bit representasjon av hvert tegn
  - alle tegn fra ASCII representeres direkte: bruker 8 bit og innledes med 0
  - “bakoverkompatibel” med ASCII og ISO 8859-1

Big endian vs. Little endian

- I representasjoner som krever mer enn én byte, finnes det to mulige rekkefølger av bytene:
  - Starte med den mest signifikante (”Big endian”)
  - Starte med den minst signifikante (”Little/small endian”)
- Eksempel:
  UTF-16 Big endian for A er 0x 00 41
  UTF-16 Little endian for A er 0x 41 00
- Begge muligheter blir brukt i praksis, og dette kan gi problemer når data overføres fra et maskinmiljø til et annet!

Byte order mark (BOM)

- Et ”Byte order mark” (BOM) er tegnet ”Zero width no-break space” med kodepunkt U+FEFF i begynnelsen av en Unicode-fil.
- Siden det ikke finnes noe tegn med kodepunkt FFFE, kan BOM brukes til å finne filformatet (UTF-32, UTF-16, UTF-8 og Big eller Small endian):

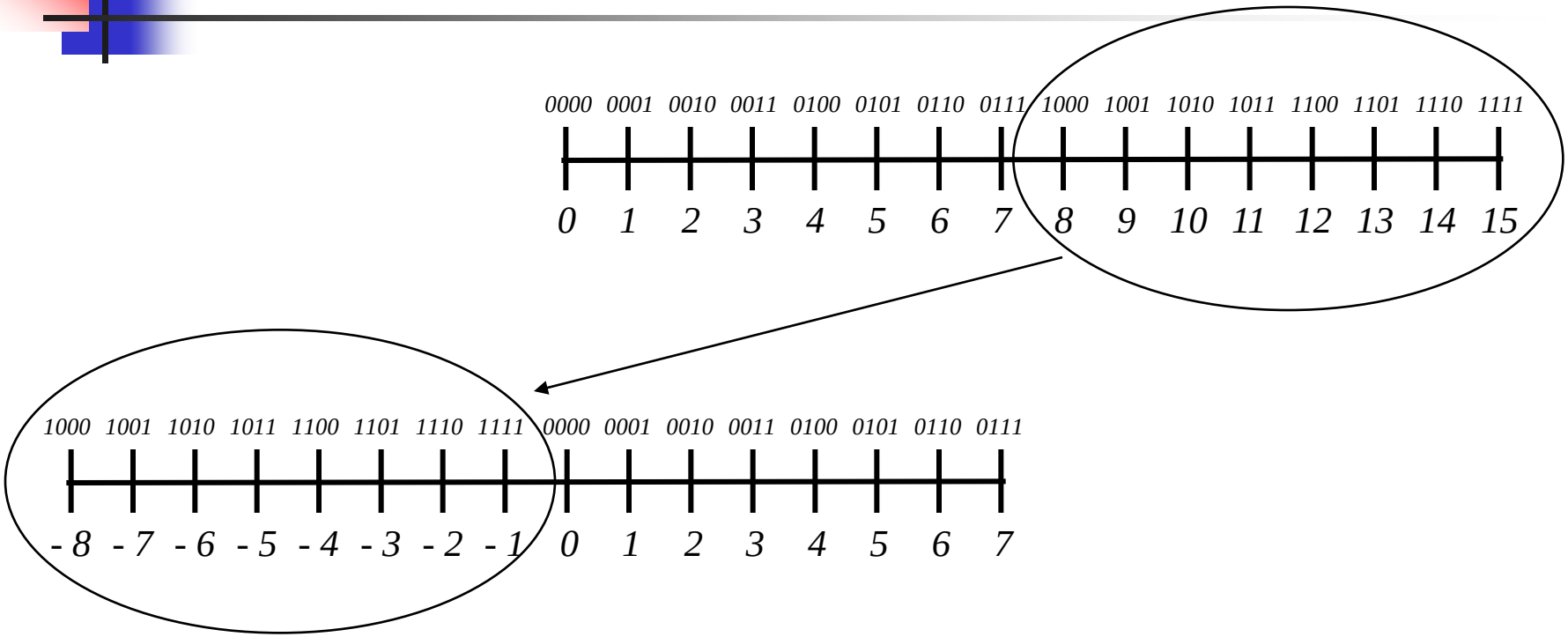
Koding	BOM-bitmønster
UTF-32, big-endian	0x 00 00 FE FF
UTF-32, little-endian	0x FF FE 00 00
UTF-16, big-endian	0x FE FF
UTF-16, little-endian	0x FF FE
UTF-8	0x EF BB BF

NEGATIVE OG REELLE TALL

Ulike klasser tall

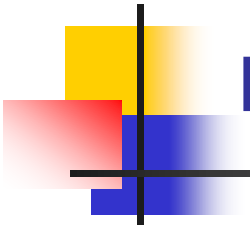
- De naturlige tallene:
  $\mathbb{N} = \{ 1, 2, 3, \ldots \}$
- De hele tallene:
  $\mathbb{Z} = \{ \ldots, -2, -1, 0, 1, 2, \ldots \}$
- De rasjonale tallene:
  $\mathbb{Q}$ = alle tall som kan skrives som en brøk
- De reelle tallene:
  $\mathbb{R}$ = alle tallene på tallinjen

Negative tall: toer-komplement

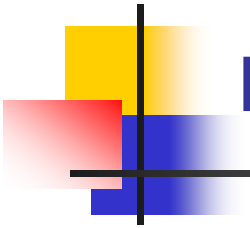


- For å negere et tall:
  1. Snu alle bit'ene i tallet (0 ↔ 1).
  2. Legg til 1.

Negative tall: Bias



- Et alternativ er å legge til en konstant bias til alle tallene.
- Med 8 bitposisjoner og bias 128 kan tallene fra -128 til 127 representeres ved hjelp av tallene fra 0 til 255.
- Eksempler:
  53: -21:

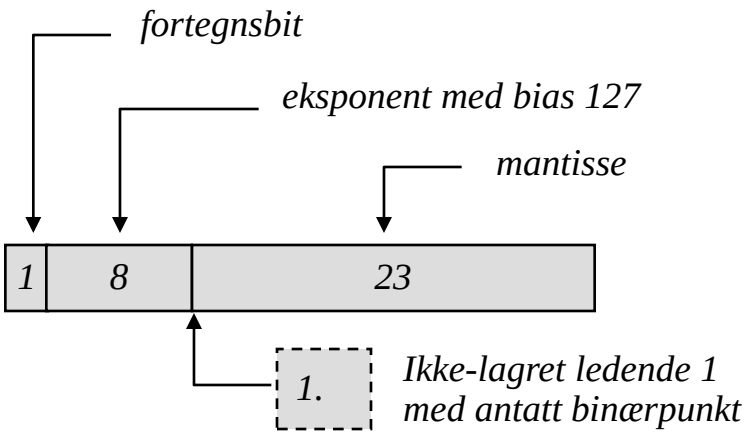


Flyttall

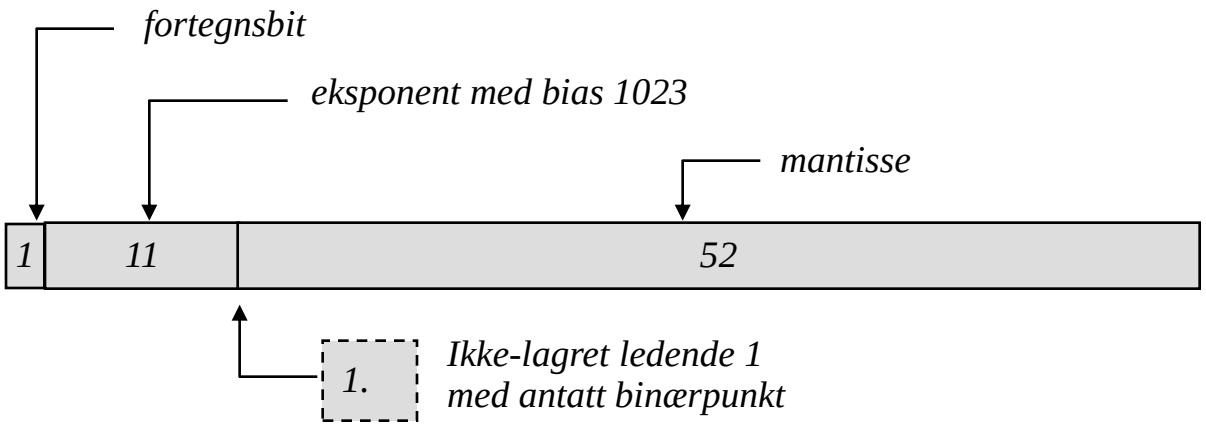
- I titallsystemet kan et tall skrives på formen
  $\text{mantisse} * 10^{\text{eksponent}}$
- Eksempler:
  $0.5 * 10^2 = 0.5 * 100 = 50$
  $0.5 * 10^0 = 0.5 * 1 = 0.5$
  $0.5 * 10^{-1} = 0.5 * 0.1 = 0.05$
  $-5 * 10^{-1} = -0.5 * 0.1 = -0.05$

- Tilsvarende kan vi skrive binære flyttall på formen
  $\text{mantisse} * 2^{\text{eksponent}}$
- For flyttall må vi altså representere både eksponent og mantisse.
  Begge må kunne være positive, negative og null.

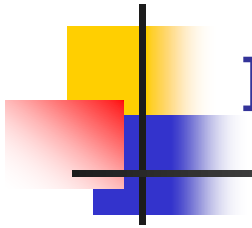
Binære flyttall: IEEE 754 single precision



Binære flyttall: IEEE 754 double precision

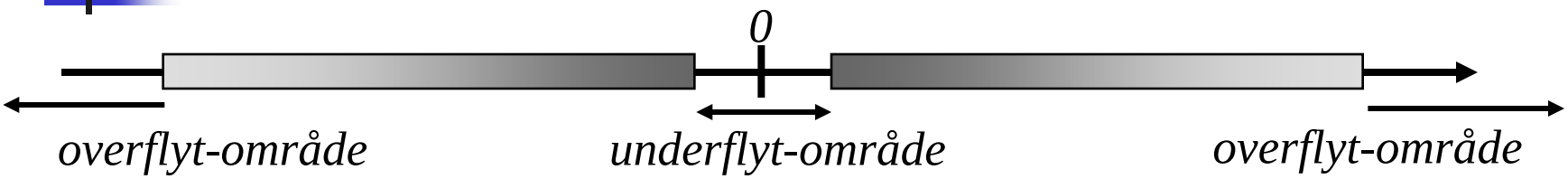


IEEE 754: Spesielle verdier

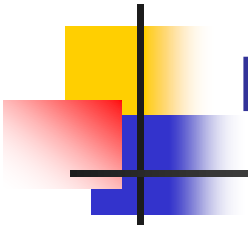


- Null: Både eksponent og mantisse er 0
- Uendelig: Eksponent med bare 1ere, mantisse med bare 0ere
- Not A Number: Eksponent med bare 1ere, mantisse ≠ 0
  - Mantisse som starter med 1 : Resultat av en udefinert operasjon (eksempel: 0/0)
  - Mantisse som starter med 0: Resultat av en ulovlig operasjon (eksempel: N/0)

Flyttallsområder i IEEE 754 og i Java



datatype	antall biter	minste positive tall	største positive tall
float	32	$2^{-126} \approx 10^{-44,85}$	$(2 - 2^{-23}) * 2^{127} \approx 10^{38,53}$
double	64	$2^{-1022} \approx 10^{-323,3}$	$(2 - 2^{-52}) * 2^{1023} \approx 10^{308.3}$



Eksamen INF1000 vår 2005

- Oppgave 1:
  - Kodeforståelse
- Oppgave 2:
  - Kodeforståelse, UML
  - Delegering av utskrift v/ metoder i flere klasser
  - NB, Java 1.4 HashMaps
- Oppgave 3
  - Metoder, klasser, lese fra fil

Oppgavetekst

Løsningsforslag

Oppgave 1 Kortsvarsoppgave (vekt 30%)

Denne oppgaven består av flere små deloppgaver, som hver teller like mye i vurderingen.

1a: Studer koden nedenfor:

```
class SkrivHei {
  public static void main (String[] args) {
    for (int i = 0; i < 5; i++) {
      for (int j = 0; j < 2; j ++) {
        i = i + 4;
        System.out.println("hei");
      }
    }
  }
}
```

Hvor mange ganger blir teksten hei skrevet ut når programmet SkrivHei kjøres?

1b: Hva skrives ut på skjermen når programmet nedenfor kjøres?

```
class EnkelRegning {
  public static void main(String[] args) {
    int a = 4;
    int b = 9;
    a = b;
    b++;
    System.out.println("a = " + a);
    System.out.println("b = " + b);
  }
}
```

1c - Hva skrives ut på skjermen når programmet nedenfor kjøres?

```
class SkrivUt2{
  public static void main (String[] args){
    String s = "abcdefghijklm";
    String t = s.substring(0,3);
    System.out.println("t er nå:" + t);
    int lengde = s.length();
    int kvart = lengde/4;
    int halv = lengde/2;
    String u = s.substring(halv - kvart, halv + kvart);
    System.out.println("u er nå:" + u);
    if (s.indexOf("deF")>(-1)) {
      System.out.println("deF finnes");
    } else
      System.out.println("deF finnes ikke");
    if (s.endsWith("klm")){
      System.out.println("s ender med klm");
} } }
```

1d: Hva skrives ut på skjermen når programmet nedenfor kjøres?

```
class Beregninger{
  public static void main(String[] args){
    int x = 9;
    int y = 3;
    int z = x - y;
    System.out.println("verdien til z er " + z);
    if ((x-z) == y)
      System.out.println("like");
    else
      System.out.println("ulike");
    if ((x > 8) || (y < 2))
      System.out.println("sann");
    else
      System.out.println("usann");
    x++;
    y += x;
    int d = --x + y--;
    System.out.println("d = " + d);
} }
```

1e:

Hva skrives ut på skjermen når programmet nedenfor kjøres?

```
class WhileTest {
  public static void main (String[] args) {
    boolean fortsett = true;
    int k = 3;
    while (fortsett) {
      k += 3;
      fortsett = !fortsett;
      System.out.println("k = " + k);
    }
  }
}
```

1f: Studer kodelinjene nedenfor:

```
int[] a = new int[50];
... // fyller inn tall i arrayen a

int s = 1;
for (int i = 0; i < a.length; ++i)
  s *= a[i];
```

Hva blir innholdet i variabelen s etter at denne koden er utført?

Oppgave 3a: DNA-sekvenser

- Gitt DNA-sekvensen AATGGATC.
- Denne sekvensen består av 8 symboler, hvorav 3 forekomster av A.
- Dermed er den relative frekvensen av A i sekvensen 3/8 = 0,375.
- Skriv metoden

```
double[] symbolFrekvens(String sekvens) {
    ...
}
```

som tar en String som parameter og returnerer en array med 4 verdier (frekvensen til A, frekvensen til T, frekvensen til C, frekvensen til G).

```
double[] symbolFrekvens(String sekvens) {
    int antA = 0;
    int antT = 0;
    int antC = 0;
    int antG = 0;

    int lengde = sekvens.length();

    for (int i = 0; i < lengde; i++) {
        char b = sekvens.charAt(i);
        switch (b) {
            case 'A': antA++; break;
            case 'T': antT++; break;
            case 'C': antC++; break;
            case 'G': antG++; break;
        }
    }

    double[] frekvens = { (double) antA/lengde,
                          (double) antT/lengde,
                          (double) antC/lengde,
                          (double) antG/lengde };
    return frekvens;
}
```

Oppgave 3b

- Lag en klasse DNAsekvens som inneholder følgende informasjon:
  - Navnet på sekvensen
  - Selve sekvensen
  - Lengden på sekvensen (antall symboler)
  - Frekvensen av A, T, C og G
- Lag en konstruktør i klassen slik at et nytt DNAsekvens objekt kan opprettes ved kodesetningen

```
DNAsekvens a = new DNAsekvens(navn, sekvens);
```

der argumentene navn og sekvens begge er av type String.

```
class DNAsekvens {
    String navn;
    String sekvens;
    double[] frekvens;
    int lengde;

    DNAsekvens(String navn, String sekvens) {
        this.navn = navn;
        this.sekvens = sekvens;
        lengde = sekvens.length();
        frekvens = symbolFrekvens(sekvens);
    }

    double[] symbolFrekvens(String sekvens) {
        // se forrige oppgave...
    }
}
```

Oppgave 3c

- Eksempel på filen DNA.txt:

```
3
AY1231 AAATCAGAAG
AY5432 GGAATCCAGTAAAA
AY3234 GGAGTCGATGA
```

Antall DNA-sekvenser i filen

Navnet på sekvensen

Selve sekvensen

- Lag en metode

```
DNAsekvens[] lesSekvenserFraFil(String filnavn){
    ...
}
```

som leser inn alle DNA-sekvensene i en array bestående av DNA-sekvens-objekter og returnerer denne.

```
DNAsekvens[] lesSekvenserFraFil(String filnavn) {
    In innfil = new In(filnavn);

    int antall = innfil.InInt();
    DNAsekvens[] sekvensene = new DNAsekvens[antall];

    for (int i = 0; i < antall; i++) {
        String navn = innfil.inWord();
        String sekv = innfil.inWord();
        sekvensene[i] = new DNAsekvens(navn, sekv);
    }
    return sekvensene;
}
```