C-programmering for musikere

Øyvind Hammer

18. september 1996

Innhold

1 Innledning **2**

2 Hallo, verden! **4**
2.1 Et C-program 4
2.2 Kompilering og kjøring 4

3 Variabler, løkker og tester **6**
3.1 Heltall og flyttall, uttrykk og tilordninger 6
3.2 Løkker 7
3.3 Tester 8

4 Algoritmisk komposisjon med CSound **9**
4.1 Score-filer 9

5 Funksjoner, tabeller, datastrukturer **12**
5.1 Funksjoner 12
5.2 Datastrukturer 14
5.3 Tabeller 15

6 Filbehandling **19**
6.1 Innlesing av filer 19

7 Lydbehandling **21**
7.1 Å skrive lydfiler 21
7.2 Lese lydfiler 22

8 Grafikk **24**
8.1 2D-grafikk 24
8.2 3D-grafikk 27

Kapittel 1

Innledning

Hvorfor lære å programmere? Fordi de fleste ferdige musikkprogrammer legger begrensninger på den kunstneriske prosessen. Selv om et ferdig system kan se åpent og generelt ut (som f.eks. *Max*), har programmereren alltid tenkt på en viss måte som nødvendigvis blir påtvunget brukeren. Man støter hele tiden på hindringer av forskjellig slag.

I dette kurset har vi valgt å benytte programmeringsspråket C. C (eller en utvidelse som heter C++) er det overlegent mest brukte språket i dag. C-programmer kjører gjerne raskt, og man har tilgang til store samlinger av ferdig kode for forskjellige deloppgaver.

C har også klare ulemper. C-kode kan se uoversiktlig ut, og det er lett å gjøre feil. Et C-program må også *kompileres*, dvs. oversettes til maskinkode, før det kjøres. Dette gir raske programmer, men det er slitsomt å måtte re-kompilere hele tiden mens man lager og tester et program.

Hvis man først har satt seg inn i C, er det lett å lære seg andre programmeringsspråk. Utfordringen er å lære seg å programmere; hvilket språk man bruker er egentlig av mindre betydning.

Å lære å programmere er imidlertid ingen rask og enkel sak. Du er nok ingen utlært programmerer etter dette kurset! En ting er å lære de grunnleggende bestanddelene i et språk (syntaksen), men så kommer det uendelige fagområdet som handler om *algoritmer*. Hvordan skal man lage et program som spiller sjakk, forutsier været eller etterlikner en fiolin? Slike problemer krever avansert kunnskap om datastrukturer, matematikk og fysikk.

Dessuten har vi alle de såkalte *bibliotekene* med kode som tar seg av spesielle ting som lyd, grafikk, nettverk osv. Man blir aldri ferdig med å lære seg hvordan slike ting skal brukes.

Vi må selvsagt begrense oss kraftig i et kurs som dette. Målet er at du skal lære å lage enkle men nyttige programmer som analyserer og produserer musikk og tildels bilder. Selv det forholdsvis lille vi lærer i dette kurset vil gi deg langt flere muligheter enn de fleste ferdige programmer kan tilby.

De fleste læretekster om programmering bruker eksempler fra ”administrativ” databehandling - sjekking av personnummer og slikt. Dette blir vel litt på siden av hva de fleste er interesserte i, så vi har valgt å lage dette egne, lille kompendiet med mer kreativt orienterte eksempler.

Selve syntaksen i C er beskrevet meget kort, uklart og ufullstendig. I tillegg til denne teksten tren-

ger du derfor en generell referanse til C. Det finnes mange, en klassisk bok er ”The C Programming Language” av Kernighan og Ritchie.

Hvorfor ikke C++? Det er jo så moderne. C++ er en utvidelse av C, som støtter såkalt *objektorientering*. Dette er meget viktig når man skal lage store systemer, men vi har valgt å begrense oss til kjernen av C.

Vi bruker UNIX-arbeidsstasjoner i dette kurset. Slike maskiner er veldig behagelige å programmere. Anskaff gjerne en C-kompilator til din Mac eller PC, men vær forberedt på et mer tungvint programmeringsmiljø.

Kapittel 2

Hallo, verden!

Et kapittel som dette er obligatorisk i alle programmeringsbøker. Hvis man først har klart å skrive og kjøre et program som skriver ”Hello World” på skjermen, har man kommet over en viktig psykologisk terskel.

2.1 Et C-program

Skriv inn følgende program i en teksteditor (f.eks. ”jot”), og lagre med filnavnet ”hallo.c”:

```
#include <stdio.h>

void main(void)
{
  printf("Hallo, verden!\n");
}
```

La oss se litt på de enkelte linjene. Den første linjen forteller at vi vil inkludere en standardpakke (”headerfil”) som tar seg av lesing og skriving. Ta alltid denne linjen med.
”`void main(void)`” markerer (deklarerer) hovedprogrammet. Vi skal komme tilbake til dette.
Krøllparentesene markerer begynnelsen og slutten av hovedprogramblokken.
”`printf`” er en kommando som skriver til skjermen. ”\n” koder for et linjeskift (newline). Et semikolon markerer slutten av en kommando.

2.2 Kompilering og kjøring

Vi kan kompilere C-programmer fra UNIX-kommandolinjen. Åpne et UNIX-skall (Skrivebord:Unix-skall i SGI-menyen) hvis du ikke har gjort det allerede. Kompiler med kommandoen

```
cc hallo.c -o hallo
```

”cc” står for ”C compile”. ”-o hallo” betyr at vi vil lage en kjørbar fil som skal hete ”hallo”. Hvis du får noen feilmeldinger, må du rette opp dette og kompilere på nytt.

Start så programmet ved å skrive ”hallo”.

Oppgave

Modifiser programmet over til å skrive ut flere linjer med en eller annen tekst.

Kapittel 3

Variabler, løkker og tester

3.1 Heltall og flyttall, uttrykk og tilordninger

Når vi lager programmer, trenger vi ”skuffer” som vi kan mellomlagre data i. Slike skuffer kalles *variabler*, og vi gir hver variabel sitt eget *variabelnavn*. Det finnes flere forskjellige *typer* av variabler, avhengig av typen data vi vil lagre i dem. Variablenes navn og type må spesifiseres i såkalte *variabeldeklarasjoner*. Her er to eksempler på slike deklarasjoner:

```
int tall;
```

deklarerer en variabel som kan lagre heltall (integer). Variabelen får navnet ”tall”.

```
float vinkel, lengde;
```

deklarerer to variabler som kan lagre flyttall, dvs. tall med komma (f.eks. 3.141592654).

Vi legger verdier i variablene ved hjelp av *tilordninger*:

```
vinkel = 7.5;
```

Denne kommandoen vil legge tallet 7.5 i variabelen *vinkel*. I tilordninger kan vi også spesifisere matematiske *uttrykk*:

```
vinkel = 7.5*(lengde+4.3);
```

Her er et ubrukelig program som illustrerer bruk av variabler og tilordninger:

```
#include <stdio.h>

int a, b, sum;

void main(void)
{
  a = 1;
  b = a+1;
  sum = a+b;

  printf("Summen er %d\n", sum);
}
```

Det mest forvirrende her er kanskje "printf"-kommandoen. "`%d`" er en såkalt *formateringskode* som angir at her skal det settes inn et heltall. Tallet selv (sum) angis etter kommaet i printf-kommandoen.

3.2 Løkker

Med såkalte *løkker* kan vi la deler av programmet kjøres gjentatte ganger. Den vanligste typen løkke er *for-løkken*. La oss se på et eksempel igjen, denne gangen et litt mer artig program som skriver ut Fibonacci-sekvensen (der hvert tall er summen av de to forrige):

```
#include <stdio.h>

int i, sum, siste, nestsiste;

void main(void)
{
  nestsiste = 0; siste = 1;
  for (i=1; i<=20; i++) {
    sum = siste+nestsiste;
    printf("%2d:    %d\n", i, sum);
    nestsiste = siste;
    siste = sum;
  }
}
```

Det sentrale her er `for`-kommandoen. Vi spesifiserer en løkke der kommandoene mellom krøllparantesene skal gjentas flere ganger. Første gang skal løkkevariabelen i være 1 (`i=1`). "`i++`" betyr at i skal økes med 1 for hver gjennomgang av løkken, dette er bare en kortform for "`i = i+1`". Løkken skal gjentas så lenge i er mindre eller lik 20.

Formateringskoden "`%2d`" i printf-kommandoen angir at hvert tall skal ta opp to posisjoner i utskriften.

Legg dessuten merke til at koden er temmelig "smart" programmert. Prøv å skjønne hva som skjer her! Dette eksemplet illustrerer at programmering innebærer en god del *tenking* i tillegg til det rent syntaktiske.

Oppgave

Lag et program som skriver ut en tabell over kvadratet av alle de tall fra 0 til 1000 som er delelige med 10, slik:

```
   0: 0
  10: 100
  20: 400
...
```

3.3 Tester

Med *if-tester* kan programmet gjøre valg, slik at deler av koden bare utføres dersom et logisk utsagn slår til. Programmet under lager en tilfeldig tallsekvens. Vi bruker den innebygde funksjonen `rand()`, som leverer et tilfeldig tall mellom 0 og 32767. Ved hjelp av en if-test stanser vi alle sprang som er større enn 5000:

```
#include <stdio.h>

int i, tall, siste;

void main(void)
{
  siste = 0; i = 1;
  while (i<=20) {
    tall = rand();
    if (abs(tall-siste)<5000) {
      printf("%2d:    %d\n", i, tall);
      siste = tall;
      i++;
    }
  }
}
```

Her har vi også brukt en ny type løkke: *while-løkken*. Koden inne i while-løkken utføres inntil tellevariabelen i viser at vi har fått skrevet ut 20 tall.

`abs` er en funksjon som gir oss absoluttverdien (verdien uten fortegn) av et uttrykk.

Kapittel 4

Algoritmisk komposisjon med CSound

Vi har nå gått igjennom de grunnleggende deler av C, og vi kan begynne å lage musikk. Vi skal gi endel eksempler på *algoritmisk komposisjon*, der vi bruker C-programmer til å generere notelister. Selve lyden genereres med syntesеprogrammet CSound. Vi har laget noen enkle CSound-instrumenter som man kan bruke til testformål, disse ligger i `/local/ckurs/test.orc`.

4.1 Score-filer

En noteliste som kan leses av CSound består av en rekke enkeltnoter, en på hver linje i en tekstfil. Linjene kan se ut f.eks. slik:

```
;Ins    Start   Dur     Frekv   Amp

i1      0       1       440     80
i1      1       1       660     70
```

Som antydet i kommentarlinjen øverst, angir det første feltet hvilket instrument vi vil bruke (det ligger 3 forskjellige instrumenter i `test.orc`, nummerert fra `i1` til `i3`). Det neste feltet angir notens starttidspunkt i sekunder. Det tredje feltet angir notens varighet i sekunder. De to siste feltene er det opp til instrumentet å tolke, men det er vanlig å bruke dem til frekvens i Hz og amplityde i dB.

Eksempel 1: Granulær syntese

Granulær syntese innebærer at vi bygger opp lydbilder ved hjelp av et stort antall korte, enkle lyder (”plopper”). Kombinasjonen av C og CSound egner seg godt for dette. Følgende program lager en stor noteliste ved hjelp av tilfeldighet med pålagte ”trender”:

```
#include <stdio.h>

int i;
float frek, start;

void main(void)
{
  printf("f1 0 4096 10 1\n"); /* Maa vaere med for CSound */
  for (i=0; i<1000; i++) {
    frek = rand()*655/32768+i;
    start = i*i/100000.;
    printf("i1  %f  0.1  %f  60\n", start, frek);
  }
}
```

Legg merke til *kommentaren*, som er markert med /* og */. I `printf` er det brukt en ny formateringskode, `%f`, som må brukes for flyttall.

Kompiler og kjør dette programmet (kall det f.eks. *granul*). Notelisten blir skrevet ut på skjermen. For å skrive til en fil istedet kan vi bruke et UNIX-knep, nemlig å *redirigere* utskriften fra programmet:

```
granul > granul.sco
```

Utskriften blir da sendt til filen `granul.sco`. For å lage lyd vha. CSound kan man gi kommandoen

```
csound -Ago testlyd /local/ckurs/test.orc granul.sco
```

CSound bruker da orkesterfilen `/local/ckurs/test.orc` og score-filen (notelisten) `granul.sco`, og lager en lydfil som får navnet `testlyd`. Spill lydfilen med kommandoen `sfplay testlyd`.

Hvis du vil eksperimentere med programmet, kan det være lurt å legge alle UNIX-kommandoene (kompilering, kjøring, syntese og avspilling) i en egen *kommandofil*. Kommandofilen må gjøres kjørbar ved hjelp av UNIX-kommandoen

```
chmod +x filnavn
```

Eksempel 2: Kaos

En gammel traver i bransjen er *kaos-musikk*, der vi bruker ulineære dynamiske systemer til å produsere notelistene. Her er et eksempel som bruker den såkalte Verhulst-likningen:

```
#include <stdio.h>

int i;
float lambda, x;

void main(void)
{
  x = 0.5;
  for (i=0; i<200; i++) {
    lambda = i/300.+3.33;
    printf("i2  %f  0.3  %f  80\n", i/8., x*1000.+50.);
    x = lambda*x*(1.-x);
  }
}
```

Oppgaver

1

Bruk CSound og C til å simulere lyden av en *ball som spretter fortere og fortere*, på følgende måte. Velg en fast pitch. Start med et intervall mellom notene på 0.5 sekunder, og multipliser dette intervallet med 0.9 for hver påfølgende note slik at vi får aksellerando. Generer 50 noter.

2

(Litt krevende)

Utvid programmet fra oppgave 1 til å simulere *mange* baller, som starter sprettingen på tilfeldige tidspunkter og med hver sin egen tilfeldige pitch. Husk at CSound ikke krever at notene angis med stigende starttidspunkter, de kan godt ligge hulter til bulter. Tips: Lag en ny løkke utenpå løkken fra oppgave 1!

Kapittel 5

Funksjoner, tabeller, datastrukturer

5.1 Funksjoner

Funksjoner er programbiter som utfører spesielle oppgaver. Det finnes mange innebygde funksjoner i C - vi har allerede sett `printf`, `abs` og `rand`. Men vi kan også definere våre egne funksjoner.

Eksempel 3: Note til frekvens

I eksemplene over har vi oppgitt tonehøyder i Hz. Hvis man heller vil bruke oktav (heltall) og pitchklasse (heltall fra 0 til 11), kan man endre CSound-orkesterfilen slik at den bruker tonehøyder i dette formatet. Men vi kan også lage oss en funksjon i C som foretar konverteringen. Dette krever at vi bruker en matematisk funksjon (`powf`) som ligger i et eget *matematikkbibliotek*.

```
#include <stdio.h>
#include <math.h>

float notefrekvens(int oktav, int pitchklasse)
{
  float frekvens;

  frekvens = 440.*powf(1.059463, (oktav-6)*12+pitchklasse-9);
  return frekvens;
}

void main(void)
{
  printf("%f\n", notefrekvens(6,9));
}
```

Vi har deklarert en funksjon som heter `notefrekvens`, som skal returnere et flyttall. Funksjonen tar to *parametre*, begge av typen `int` (heltall). Variabelen `frekvens` er *lokal* i funksjonen, dvs. at den ikke kan refereres andre steder enn inne i funksjonsdeklarasjonen.

Fordi vi bruker matematikkbiblioteket, må vi "lenke inn" dette når vi kompilerer. Dette gjøres ved å ta med "`-lm`" i kompileringskommandoen, f.eks. slik:

```
cc notefrek.c -lm -o notefrek
```

Oppgave

Lag en funksjon som tar antall timer, minutter, sekunder og tiendedels sekunder som parametre, og returnerer den totale tiden som dette tilsvarer i sekunder. F.eks. skal 2 timer, 14 minutter, 23 sekunder og 3 tiendedeler gi 8063.3 sekunder.

5.2 Datastrukturer

I resten av dette kapittelet skal vi arbeide endel med "tradisjonell" noterepresentasjon. Vi kan betrakte en note som et *objekt* med endel egenskaper, f.eks. starttid, varighet, oktav, pitchklasse, styrke og instrument. Det gir renslige programmer om vi definerer en egen note-datatype der disse egenskapene inngår. Dette er faktisk et enkelt eksempel på en slags objektorientert programmering.

Eksempel 4: Et bibliotek for notebehandling

Her er definisjonen av note-datatypen, og en forbedret versjon av notefrekvens-funksjonen som benytter denne datatypen. I tillegg finnes en funksjon som skriver en CSound-notelinje. Vi skal bruke disse deklarasjonene i senere eksempler. Legg spesielt merke til definisjonen av datatypen `notetype` ved hjelp av en C-konstruksjon som heter `struct`. Nede i `main` kan vi så deklarere en variabel `min_note` av denne typen. Bemerk også hvordan vi refererer til elementene inne i noten med en punktum-notasjon. Vi leser `note.oktav` som "note sin oktav".

```
#include <stdio.h>
#include <math.h>

typedef struct {
  float starttid, varighet, styrke;
  int oktav, pitchklasse, instrument;
} notetype;

float notefrekvens(notetype note)
{
  float frekvens;

  frekvens = 440.*powf(1.059463, (note.oktav-6)*12+note.pitchklasse-
9);
  return frekvens;
}

void skrivnote(notetype note)
{
  printf("i%d  %.3f  %.3f  %.2f %.2f\n",
    note.instrument,
    note.starttid,
    note.varighet,
    notefrekvens(note),
    note.styrke);
}
```

```
void main(void)
{
  notetype min_note;

  min_note.instrument = 1;
  min_note.starttid = 0.0;
  min_note.varighet = 1.0;
  min_note.oktav = 4;
  min_note.pitchklasse = 9;
  min_note.styrke = 80;
  skrivnote(min_note);
}
```

Disse definisjonene og rutinene (bortsett fra `main`) kan vi legge i en egen include-fil som vi kan kalle `notelib.h`, for senere bruk.

Oppgaver

1

Endre note-objektet slik at det takler mikrotonalitet (hvis du først skjønner hva jeg mener med spørsmålet, er dette *fryktelig* lett).

2

(Litt krevende)

Bruk notebiblioteket til å lage et program som spiller tilfeldige tolvtone-treklanger innenfor en fast oktav.

5.3 Tabeller

De variablene vi har sett på hittil kan bare lagre enkelttall og enkeltobjekter. Hvis vi vil arbeide med *lister* av objekter, må vi bruke *tabeller*. En tabell som kan lagre 1000 heltall deklareres slik:

```
int tabell[1000];
```

Enkeltelementer i tabellen kan så refereres ved å *indeksere*. Slik legger vi tallet 13 i element nummer 576:

```
tabell[576] = 13;
```

Indeksene løper fra null, slik at høyeste lovlige indeks her er 999. Hvis man går utenfor denne grensen, vil programmet kræsje.

Eksempel 5: Skalaer

De forsøkene vi har gjort med algoritmisk komposisjon har brukt en kromatisk skala, slik at heltallene som genereres angir et halvtonetrinn direkte. For å spille i en annen skala, kan vi bruke en tabell.

En c-dur-skala kan vi spesifisere slik:

```
int skala[7] = { 0, 2, 4, 5, 7, 9, 11 };
```

Her er angitt de forskjellige halvtonetrinnene innenfor en oktav (pitchklasser) som utgjør skalaen.

Vi kan nå lage et program som spiller tilfeldige melodier i c-dur. Først finner vi et tilfeldig tall mellom 0 og 15 (dette dekker to oktaver). Deretter beregnes oktaven ved hjelp av heltallsdivisjon med 7. Noten innen oktaven bestemmes ved hjelp av *modulo*-operatoren '%', som gir resten ved heltallsdivisjon, slik at f.eks. 9%7=2. Dette siste tallet indekserer så en av de 7 pitchklassene i skalatabellen.

Dessuten blir alle C'er aksentuert med 10 dB ekstra styrke.

```
#include <stdio.h>
#include <math.h>
#include "notelib.h"

int skala[7] = { 0, 2, 4, 5, 7, 9, 11 };

void main(void)
{
  int i, tall;
  notetype cnote;

  for (i=0; i<50; i++) {
    tall=rand()*16/32768;
    cnote.instrument = 2;
    cnote.starttid = i/5.;
    cnote.varighet = 0.3;
    cnote.oktav = tall/7+6;
    cnote.pitchklasse = skala[tall%7];
    if (cnote.pitchklasse==0) cnote.styrke = 90; else cnote.styrke=80;
    skrivnote(cnote);
  }
}
```

Eksempel 6: Serielle operasjoner

For å behandle rekker av noter, kan vi lage en tabell som inneholder note-objekter. Eksemplet under lager en tilfeldig tolvtonerekke. Deretter spilles rekken baklengs, og til slutt med inverterte pitchklasser.

```
#include <stdio.h>
#include <math.h>
#include "notelib.h"

notetype rekke[8];

void main(void)
{
  int i, tall;

  /* Lag og spill rekken */
  for (i=0; i<8; i++) {
    tall=rand()*16/32768;
    rekke[i].instrument = 2;
    rekke[i].starttid = i/4.;
    rekke[i].varighet = 0.3;
    rekke[i].oktav = tall/12+6;
    rekke[i].pitchklasse = tall%12;
    rekke[i].styrke=80;
    skrivnote(rekke[i]);
  }

  /* Spill rekken baklengs */
  for (i=0; i<8; i++) {
    rekke[7-i].starttid = 2.5 + i/4.;
    skrivnote(rekke[7-i]);
  }

  /* Spill rekken invertert */
  for (i=0; i<8; i++) {
    rekke[i].starttid = 5 + i/4.;
    rekke[i].pitchklasse = 11-rekke[i].pitchklasse;
    skrivnote(rekke[i]);
  }
}
```

Oppgaver

1

Endre programmet i eksempel 5 til å spille pentatont.

2

Lag en funksjon som tar et noteobjekt og et intervall (heltall) som parametre, og returnerer en transponerert note. Det blir endel fikling fram og tilbake med oktaver, pitchklasser og halvtonetrinn - det hadde kanskje vært lurere å bruke noteobjekter som representerer pitch med bare et heltall fra 0 til 84?

3

(Krevende)

Lag et program hvor du legger inn en kort melodi som heltall i en tabell, f.eks. kan vi representere "Lisa gikk til skolen" med tallsekvensen `60, 62, 64, 65, 67, 67, 69, 69, 69, 69, 67 ...`. Vi sier at alle notene er like lange.

Spill av denne melodien mange ganger, men bytt om to av notene (tilfeldig valgt) hver gang, slik at melodien blir mer og mer (per)mutert. Å bytte om to tall i en tabell krever litt tenking.

Kapittel 6

Filbehandling

6.1 Innlesing av filer

Hvis man f.eks. vil bruke analysedata fra andre programmer eller ”massere” en eksisterende CSound-scorefil, må man lese data fra filer. Filen må da åpnes og lukkes, og det er egne funksjoner for å lese ulike typer data.

Eksempel 7: Innlesing av tall

Eksemplet under leser inn heltall fra en tekstfil og skriver dem ut som noter:

```
#include <stdio.h>
#include "notelib.h"

void main(void)
{
  FILE *fil;
  notetype note;
  int tall;

  note.instrument = 2;
  note.starttid = 0.0;
  note.varighet = 0.5;
  note.styrke = 80;

  fil = fopen("filnavn", "r");
  while (1) {
    fscanf(fil, "%d", &tall);
    if (feof(fil)) break;
    note.oktav = tall/12+3;
    note.pitchklasse = tall%12;
```

```
    skrivnote(note);
    note.starttid += 0.4;
  }
  fclose(fil);
}
```

Her var det endel nye ting. `FILE *fil` deklarerer en variabel som er et "håndtak" til en fil. `while (1)` betyr at vi skal gå i evig løkke.

Selve innlesingen gjøres med `fscanf`. "%d" er en formateringskode for heltall, som vi før har sett i forbindelse med `printf`. &-tegnet før `tall` vil jeg helst ikke komme inn på, det har å gjøre med *pekere*, som vi holder oss borte fra i dette kurset. Bare husk å ha den med i `fscanf`.

`feof` blir logisk sann når vi har nådd end-of-file, og da hopper vi ut av løkken med `break`.

Oppgave

På `/local/ckurs/dnafil` ligger en sekvens på 52173 nukleotider fra menneskets DNA. Det finnes fire ulike nukleotider, som representeres med bokstavene c (cytosin), a (adenin), g (guanin) og t (thymin). Filen er dermed en lang remse med "cagttcgaatgag" osv.

Kan du lage musikk av dette? For å lese inn bokstaver må du bruke en datatype som heter `char` (character), og formateringskoden `%c`, f.eks. slik:

```
char nukle;

...
fscanf(innfil, "%c", &nukle);
if (nukle=='c') {
  ...
}
...
```

Kapittel 7

Lydbehandling

C-programmer kan lese og skrive lydfiler, og dermed har du direkte tilgang til samplene i lyden og kan lage dine egne syntese- og lydbehandlingsmetoder. Dette trenger ikke være avanserte signalbehandlingsteknikker, enkle og ”naive” algoritmer kan ofte låte minst like spennende.

7.1 Å skrive lydfiler

Vi har laget et eget bibliotek som gjør det enkelt å håndtere lydfiler fra C. La oss prøve kaos-programmet fra et tidligere kapittel igjen, men denne gangen skriver vi ut verdiene direkte som samples.

Eksempel 9: Kaotisk oscillator

```
#include <stdio.h>
#include "/local/ckurs/lydfil.h"

Lydfil *lydfil;

void main(void)
{
  int i;
  float lambda, x;

  lydfil = lydfil_openskriv("test", 16000); /* Filnavn, samplerate */

  x = 0.5;
  for (i=0; i<400000; i++) {
    lambda = i/600000.+3.33;
    lydfil_skrivsample(lydfil, x*20000.);
    x = lambda*x*(1.-x);
  }
```

```
  lydfil_lukk(lydfil);

}
```

Lydfil-funksjonene skulle være selvforklarende. Programmet må kompileres med `-laudiofile` for å lenke inn lydfilbiblioteket:

```
  cc kaoslyd.c -laudiofile -o kaoslyd
```

Oppgave

(Krevende)

Lag en *pulsbreddemodulert* lyd. Dette kan gjøres ved å skrive f.eks. 200 samples (en periode) 200 ganger etter hverandre i en loop. Første gang er første sample i perioden 0, resten 20000. Andre gang er de to første samplene 0, osv., inntil alle 200 samplene er 0.

7.2 Lese lydfiler

Funksjonene for å lese samples fra en lydfil er like enkle. I eksemplet under leser vi inn samples fra en lydfil, ringmodulerer dem (multipliserer med en sinusbølge) og skriver til en annen lydfil. Bemerk &-tegnene i `lydfil_openles`, de brukes på samme måte som vi har sett for `fscanf` og betyr at verdiene for antall samples i lyden og for samplingsfrekvensen skal legges inn i variablene `antsamp` og `srate`.

Eksempel 10: Ringmodulering

```
#include <stdio.h>
#include <math.h>
#include "/local/ckurs/lydfil.h"

Lydfil *innlydfil, *utlydfil;

void main(void)
{
  int i, innsamp, antsamp, srate;
  float modulator;

  innlydfil = lydfil_openles("test", &antsamp, &srate);
  utlydfil = lydfil_openskriv("test2", srate);

  for (i=0;  i<antsamp; i++) {
```

```
    modulator = sin(i*110.*2.*3.14159/srate);
    innsamp = lydfil_lessample(innlydfil);
    lydfil_skrivsample(utlydfil, modulator*innsamp);
  }

  lydfil_lukk(innlydfil);
  lydfil_lukk(utlydfil);
}
```

Oppgave

Lag et program som tar absoluttverdien av en lyd. Dette simulerer en gitareffekt som heter *diodefuzz*.

Kapittel 8

Grafikk

Vi har valgt å se litt på computergrafikk i dette kurset, fordi grafikk er interessant også i musikksammenheng. Visualisering av lyd, og andre kombinasjoner av lyd og bilde (f.eks. musikkvideoer), er aktuelle temaer for tiden.

8.1 2D-grafikk

Grafikk er ikke en integrert del av C, men det finnes mange ulike biblioteker som man kan bruke. Et av de mest avanserte og mest brukte er GL (Graphics Library), som opprinnelig ble utviklet av SGI, men som nå også er tilgjengelig for PC. GL kan håndtere både todimensjonal og tredimensjonal grafikk, men vi skal holde oss til 2-D foreløpig.

Det er endel initialisering og mekk med GL, så vi har laget et eget lite grafikkbibliotek som forenkler ting.

Eksempel 11: Streker

Dette programmet tegner masse streker:

```
#include <stdio.h>
#include "/local/ckurs/grafikk.h"

void main(void)
{
  int i, x, y;

  openwin(800,800);
  for (i=0; i<1000; i++) {
    setcolor(1, i/1000., i/1000.);
    drawline(rand()/41, rand()/41, rand()/41, rand()/41);
  }
```

```
  sleep(100); /* Heng i 100 sekunder (stopp med ctrl-c) */
}
```

Funksjonen `openwin` lager et grafikkvindu med den oppgitte bredde og høyde.

`setcolor` setter en farge. De tre parametrene går fra 0 til 1, og bestemmer hvor mye vi vil ha av hhv. rødt, grønt og blått (RGB).

Programmet må kompileres med `-lgl` for å lenke inn grafikkbiblioteket.

Eksempel 12: Faseplott av lyd

Her følger et litt stort og komplisert eksempel som lager et animert *forsinket faseplan-plott* av en lyd. Programmet leser først en hel lydfil inn i minnet. C-funksjonen `malloc` brukes til å opprette en tabell `lyd` med en størrelse som avgjøres av lengden på lydfilen.

Deretter går programmet i en løkke som tegner ut ett punkt for hvert sample i lyden. X-koordinatet bestemmes av sample-verdien, mens y-koordinatet bestemmes av verdien 100 samples tidligere. For at vi ikke bare skal få flere og flere punkter, fjernes også "gamle" punkter etterhvert, ved å tegne dem i svart.

```
#include <stdio.h>
#include "/local/ckurs/grafikk.h"
#include "/local/ckurs/lydfil.h"

Lydfil *lydfil;
int antsamp;
short int *lyd=NULL;

void lesfil()
{
  int i, R;

  lydfil=lydfil_openles("lydfil", &antsamp, &R);

  /* Lag et array som kan lagre hele lyden */
  lyd=(short int *)malloc(antsamp*sizeof(short int));

  for (i=0; i<antsamp; i++)
    lyd[i]=lydfil_lessample(lydfil);
  lydfil_lukk(lydfil);
}

void tegnpunkt(int n)
{
  int x, y;
```

```
  /* Fjern et 40000 samples gammelt punkt */

  if (n>=40100) {
    x=(lyd[n-40000]/32768.+0.5)*768;
    y=(lyd[n-40000-100]/32768.+0.5)*576;
    setcolor(0., 0., 0.);
    drawpoint(x, y);
  }

  /* Tegn nytt punkt */

  x=(lyd[n]/32768.+0.5)*768;
  y=(lyd[n-100]/32768.+0.5)*576;
  setcolor((n%40000)/40000., 1.0, 1.-(n%40000)/40000.);
  drawpoint(x, y);
}

void main(void)
{
  int n;

  lesfil();
  openwin(768, 576);
  for (n=100; n<antsamp; n++) tegnpunkt(n);
  sleep(10);
}
```

Oppgaver

1

Lag et program som tegner en *logaritmisk spiral*:

```
  x = exp(8*t)*cos(40*t)/10
  y = exp(8*t)*sin(40*t)/10
```

der t løper fra 0 til 1 i fine steg.

2

Plott det kaotiske systemet fra eksempel 2.

8.2 3D-grafikk

Vi skal ikke styre 3D-grafikk direkte på skjermen i dette kurset, det blir for komplisert. Vi skal heller skrive en beskrivelse av 3D-scenen til en fil, som så kan leses av en egen 3D-framviser. Filen kodes i *Inventor*-formatet, som er en standard på SGI. Inventor er også noenlunde kompatibel med *VRML* (Virtual Reality Modelling Language) som er standard-formatet på web. Du kan altså legge filen på hjemmesiden din, og den blir da vist fram i 3D i Netscape.

Eksempel 13: Kulefigur

```
#include <stdio.h>
#include <math.h>

void main(void)
{
  int i;
  float phi;

  printf("#Inventor V2.0 ascii\n");
  printf("Separator {\n");

  for (i=0; i<20; i++) {
    phi = i*2.*3.14159/20.;
    printf("  Separator {\n");
    printf("    Material { diffuseColor [ %f %f 0.7 ] }\n",
      i/20., rand()/32768.);
    printf("    Transform { translation %f %f %f }\n",
      cos(phi)*2, sin(phi)*2, cos(2.*phi)*2);
    printf("    Sphere { radius 1 }\n");
    printf("  }\n");
  }
  printf("}\n");
}
```

Se på filen som dette programmet lager. Definisjonen av hver kule er pakket inn i en *Separator*. Hver kule har et *Material* med farge, matthet etc., en *Transform* med posisjonen i rommet, og en *objekttype* med typen av geometrisk objekt.

Lag en fil som heter f.eks. `kuler.iv`, og bruk programmet `ivview` for å se resultatet (blir best på maskinen "leon"):

```
ivview kuler.iv
```

En komplett beskrivelse av VRML finner du på

```
http://vag.vrml.org/vrml10c.html
```

Oppgave

Skriv et program som lager 100 kuber i tilfeldige posisjoner. Bruk Inventor-objektet *Cube*:

```
Cube { height 1 depth 1 width 1 }
```