Soundfile Processing

Mattias Cederlund

22-11-2012

1 Uppgiften

Vi har fått givet ett c-bibliotek innehållande verktyg för att manipulera ljudfiler. Uppgiften består i att kompillera och installera biblioteket och använda sig av dess funktioner för att ta fram information om ljudfiler och normalisera dem för att få maximal volym. Vi ska även utvecka en funktion som sätter ihop flera ljudfiler utan att det sprakar i mellan. Alla dessa funktioenr ska argument från kommandoraden.

2 Ansatts

2.0.1 Färdigställa SndReader

I SndReader-klassen ska funktionen next() som flyttar fram pekaren till nästa frame i buffern färdigställas. Funktionen ska returnera true om det finns fler frames tillgängliga och false annars. Om vi nått slutet av buffern ska vi anropa refresh_buffer() för att ladda in nya frames till buffern.

När next anropas flyttar vi först fram cur, positionen för nuvarande frame i buffern ett steg. Sedan undersöker vi om den är mindre än end, sista framen i buffern. Isåfall returneras true. Annars anropar vi refresh_buffer() och sedan kollar vi om end, som blivit uppdaterad är noll, alltså om det inte finns några frames i buffern. Är så fallet returneras false, annars true.

2.0.2 Skriva ut information om en ljudfil

Uppgiften var att skriva ett program som tar en path till en ljudfil som argument. Programmet ska sedan skriva ut namn, samplerate, antal channels, antal samples, längd (h:m:s) och högsta volym.

Argumenten behandlas på samma sätt som i sndcopy.cc. Vi kollar om antalet argument är rätt, om inte avslutas programmet och ett felmeddelande visas. Om antal argument stämmer så antar vi att argument 1 är filnamnet. Vi skapar en reader till ljudfilen och får filnamn direkt från argumentet. Från

funktionerna samplerate() och channels() får vi underförstått efterfrågad information. Antal samples fås genom channels()*frames(), där frames() ger antal frames. Eftersom varje frame innehåller en sample för varje channel har vi antal samples.

Längden i sekunder får vi utav antal frames genom sampleraten. För att skriva ut dessa i timmar, minuter och sekunder använder vi division och modulooperatorn på lämpligt sätt.

Högsta volymen får vi genom att gå igenom alla samples där vi loopar så länge det finns fler frames, vilket ges av next(). Vi loopar igenom alla channels, hämtar samplelns värde med sample(c), där c är channel och jämför med ett tidigare högsta värde. Överstiger samplens värde det högsta sparas absolutbeloppet som det nya högsta påträffade värdet.

2.0.3 Maximera volymen av en ljudfil

Uppgiften var att skriva ett program som tar namn på ljudfiler som argument och som därefter ser till att volymen blir maximal. För att öka volymen av en ljudfil till det maximala, alltså att den högsta volymen är 1 behöver vi skala om alla samples beroende på den högsta uppmätta volymen. Delar vi alla samples på den högsta uppmätta volymen kommer den sample med högst volym bli 1 och resterande kommer fortfarande ha samma proportioner.

Argumenten hanteras på samma sätt som i sndcopy.cc. Vi läser in en path till en fil som ska normaliseras och skapar en reader till den filen. Vi skapar sedan en writer till en ny fil, normalized.wav som kommer bli den normaliserade versionen av input-filen. Därefter tar vi reda på filens högsta volym på samma sätt som beskrivet i föregående avsnitt. Vi går nu tillbaka till början av filen och loopar igenom alla frames och channels. Vi läser av samplen och skriver en ny sample med writern där vi delar det avlästa värdet med det tidigare bestämda högsta värdet. Vi stegar fram writern ett steg och repeterar tills readerns buffer är tom. Därefter stänger vi reader och writer och programmet avslutas.

2.0.4 Slå ihop ljudfiler utan klick

Uppgiften var att skapa ett program som tar namn på ljudfiler som argument från kommandoraden och sätter ihop dem till en fil utan klick mellan ljudfilerna. Funktionen ska också ta en parameter -gap n där n är antal millisekunder som övergången mellan ljudfilerna sker. För att få det klick-fritt lägger vi i övergången till samples som går från nivån vid slutet av den ena ljudfilen till nivån i början av den andra; övergången blir därmed mjuk och vi undviker klick.

Programmet har som argument i main antalet argument som gives på kommandoraden och en pekare till en array innehållande dessa argument. Det första programmet gör är att gå igenom argumenten med en loop där vi undersöker om argumentet är -gap". Isåfall vet vi att nästa efterföljande argument är antalet millisekunder som övergången mellan två ljudfiler ska vara och vi sparar ner detta för senare. Det första argumentet som påträffas som inte är -gap-parametern antar vi är en path till en ljudfil. Vi öppnar denna med hjälp av en SndReader och bestämmer utifrån den vilken samplerate och hur många channels som filen vi skriver till ska ha. Alla filer måste nämligen ha samma samplerate och antal channels.

Nu skapar vi en writer med de värden vi har och går igenom argumenten igen med hjälp av en loop. Påträffas "gap"vet vi att nästa argument är övergångstiden så vi ökar index så att den hoppas över. Alla andra argument antas vara paths till ljudfiler och när en sådan påträffas skapar vi en reader för filen. Därefter kollas det om det finns något sparat värde från en eventuell föregående fil. Finns det ska en mjuk övergång skapas.

Information från slutet av föregående fil sparas i en vector där den sista samplen från varje channel sparats. För att bestämma om en mjuk övergång ska göras kollar vi helt enkelt om denna vectors storlek är större än 0. Isåfall loopar vi igenom vector lika många gånger som antal channels som finns och för varje channel bestämmer vi hur stort hoppet ska vara i ljudstyrka mellan alla samples. Om ett positivt värde sparats är skillnaden mellan den förra ljudfilen och den nya: -vectorns sparade värde minus första samplens värde, är det ett negativt värde tvärt om. Sedan delas skillnaden på antal frames som övergången ska bestå av, dvs. samplerate*n/1000. Därefter skriver vi till frames med (gamla värdet + i * hoppstorleken) där i går upp till samplerate*n/1000. När alla samples för alla channels skrivits clearas vector.

Nu kollar vi om den nya ljudfilen har rätt samplerate och antal channels. Har den inte det avslutas programmet med ett felmeddelande. Om samplerate och antal channels stämmer så skrivs hela ljudfilen över till den nya filen på samma sätt som i sndcopy.cc. Därefter går vi till sista samplen och sparar ner värdet för alla dess channels till vectorn inför nästa övergång. Sedan stänger vi ner readern för den fil vi nyss läst och läser in nästa argument osv. När alla argument lästs in stänger vi writern och programmet är nu klart och avslutas.

3 Utvärdering

Vid ett test av normaliseringsprogrammet undersöktes först b.wav med hjälp av sndinfo.

```
Filename: wavfiles/b.wav
Samplerate: 44100
Number of channels: 1
Number of samples: 88167
Length (h/m/s): 0:0:1
Highest volume level: 0.800049
```

Därefter anropades normaliseringsprogrammet med b.wav som argument. Min normaliserare skriver output till en fil som heter normalized.wav. Anropar vi den med sndinfo får vi följande utskrift.

```
Filename: wavfiles/normalized.wav
Samplerate: 44100
Number of channels: 1
Number of samples: 88166
Length (h/m/s): 0:0:1
Highest volume level: 0.999969
```

Den högsta volymen är nu maximal och därmed verkar programmet fungera. Vid uppspelning av båda ljudfilerna hörs även en volymskillnad.

4 Sammanfattning

Det kanske främsta problemet jag hade i början av labben är att jag kör på en windows-maskin. För att kunna använda biblioteket körde jag en virtualbox med Ubuntu i och då gick det utan problem.

Sedan var det lite trassel med argument-hanteringen i uppgiften där ljudfiler skulle sättas ihop. Vid letande efter -gapbland argumenten fanns det aldrig. Kanske inte så konstigt eftersom att man från början försökte jämföra en sträng med objekt i en char-array. För att argumenten skulle kunna jämföras med strängen -gapbehövde de omvanlas först. Det gjorde jag med string(argv[i]). Detsamma gällde för n, alltså hur många millisekunder övergången skulle vara. Den omvanlades till en int med hjälp av atoi(argv[i]).

Lärandemålet är tydligt, men jag tycker att man även skulle lägga till argument-hanteringen som ett lärandemål då det är en stor del av uppgiften och för mig det som orsakade mest problem.

Ursprungligen hade jag även problem med övergångarna mellan ljudfiler i programmet som ska appenda dem. Jag upptäckte att skillnaden mellan förra filens slutvärde och den nyas startvärde blev olika beroende på om de var negativa eller positiva. Därmed behövde jag dela upp fallen i när sparade värdet var positivt eller negativt för att inte övergången skulle bli bakvänd".