Musikbibliotek

De flesta formaten för ljud, bild och video innehåller så kallad metainformation, alltså information *om* informationen som filen i huvudsak innehåller. Det kan vara information om när och var en bild är tagen, hur långt ett filmklipp är, eller vilken artist som spelar en viss låt. Den här uppgiften går ut på att läsa av metainformationen hos ett antal musikfiler och sortera in dem i ett musikbibliotek.

Notera att denna uppgift kan uppfattas som svårare än de andra två i denna sprint!

Partiell lista över innehållet

– I/O från fil
– Minneshantering på bytenivå
– Interaktion med filsystemet
– Stränghantering

Introduktion

MPEG-1 Audio Layer 3 (ofta förkortat MP3) är ett filformat[1] för ljud som länge har varit populärt. Det finns ingen standard för metainformation i MP3-filer, men den överlägset mest använda metoden är att inkludera så kallade ID3-taggar. ID3 finns i flera versioner, men i den här uppgiften kommer vi fokusera på version 2.3 (som är den näst senaste).

ID3v2-taggar ligger ofta i början av ljudfilen och består av en *header* följt av ett antal *frames*. Headern består av 10 bytes och har följande utformning[2]:

```
ID3v2 identifier:  "ID3"
   ID3v2 version:  0x03 0x00
     ID3v2 flags:  0x??
      ID3v2 size:  0x?? 0x?? 0x?? 0x??
```

Först står de tre tecknen ”ID3”, vilket markerar starten på ID3-taggen. Sedan följer två bytes som anger vilken version av ID3v2 som används (här är det ID3v2.3.0). Påföljande byte anger ett antal flaggor som påverkar hur resten av filens metadata ska tolkas. Deras betydelse kan ignoreras helt för denna uppgift. Slutligen följer fyra bytes som anger hur stor (i antal bytes) hela ID3-taggen är, exklusive headern. Se nedan för information om hur man tolkar dessa bytes!

Efter headern följer sedan ett antal poster (frames) som var och en innehåller ett visst stycke information. Varje post inleds med en egen header, också den 10 bytes, med följande utseende:

```
Frame ID:  0x?? 0x?? 0x?? 0x??
    Size:  0x?? 0x?? 0x?? 0x??
   Flags:  0x?? 0x??
```

Först står fyra tecken som anger vilken typ av information som posten innehåller (se nedan). Påföljande fyra bytes anger storleken på posten (*exklusive* postens header) på samma sätt som i ID3-taggens header. Sist ligger två bytes som anger flaggor om den aktuella posten. Deras betydelse kan också ignoreras för den här uppgiften. Efter postens header kommer sedan postens själva innehåll, vars längd har angivits i postheadern.

Frameinnehåll

Viken typ av innehåll som posten innehåller bestäms som sagt av de första fyra tecknen i postens header, som kallas Frame ID. Alla poster vars information består av text har ett Frame ID som inleds med tecknet ”T”. Sådana posters innehåll börjar alltid med en byte som anger teckenkodningen för texten (`0x00` för ISO-8859-1, `0x01` för UCS-2), sedan följer själva texten. I den här uppgiften kommer innehållet alltid vara text. Följande Frame IDn är aktuella:

[1] Egentligen en algoritm för ljudkomprimering
[2] 0x?? anger ett hexadecimalt tal (en byte).

- `TIT2` – Låttitel
- `TPE1` – Huvudartist
- `TALB` – Album
- `TRCK` – Spårnummer
- `TCON` – Genre
- `TYER` – Inspelningsår

Padding

Efter den sista posten lägger man ibland in ”tomt utrymme” bestående av bara nollor. Detta gör man för att lämna plats för fler frames innan filens ljuddata, så att man inte behöver skriva om hela filen om man vill utöka metadatan. Notera att paddingen räknas in i längden som anges i ID3-taggens header!

ID3 size

Storleksangivelserna i ID3-formatet anges på ett lite speciellt sätt. Det är fyra bytes, där endast de sju *minst* signifikanta bitarna i varje byte spelar roll (den mest signifikanta biten sätts till noll). I praktiken är det alltså ett värde på 28 bitar. För att läsa in det till en `int` (till exempel) kan man läsa en byte i taget, ”flytta över” de sju intressanta bitarna till `int`en och skjuta dem sju steg åt vänster för att lämna plats för bitarna i nästa byte. Om storleken till exempel är `0x00 0x00 0x01 0x2A` så borde den resulterande storleken bli `0xAA` (170). Följande bitoperationer kan vara användbara[3]:

- `x & k` – Bitvis ”och” mellan `x` och `k`. `x & 127` ger de sju minst signifikanta bitarna i `x`.
- `x << k` – Skifta bitarna i `x` `k` steg åt vänster (och fyll på med nollor).

Uppgiften

Målet med den här uppgiften är att skriva ett program som kan läsa in ID3-taggen från ett antal MP3-filer och skapa en katalogstruktur där filerna sorteras efter artist och album. Nedan följer ett antal tips för hur uppgiften kan angripas:

- Större delen av den här uppgiften går ut på att förstå hur ID3 fungerar och hur dess information skall läsas från filerna. Följande länkar kan vara av intresse:
  - Wikipedia: `http://en.wikipedia.org/wiki/ID3`
  - ID3v2 made easy: `http://id3lib.sourceforge.net/id3/easy.html`
  - ID3v2.3 informal standard: `http://id3lib.sourceforge.net/id3/id3v2.3.0.html`
  - C file system interface: `http://www.gnu.org/software/libc/manual/html_node/File-System-Interface.html`
- För att få ett hum om hur datan i MP3-filer ser ut kan ni prova att öppna en i Emacs. Det går att läsa en fil hexadecimalt genom att slå på läget `hexl-mode` (använd `M-x`).
- Ni kommer förmodligen vilja lagra information om individuella frames i ert program. Det kan vara en bra idé att skapa en strukt som kan lagra en frame i minnet.
- Försök inte skriva hela programmet på en gång! Se om ni först kan få programmet att skriva ut informationen i ID3-taggens header. Försök sedan skriva ut en fils alla frames.
- Prova funktionerna för interaktion med filsystemet i mindre program innan ni försöker bygga själva musikbiblioteket. När ni känner er säkra kan ni kan prova att först bara byta namn på filen ni läser (till något som anger lämplig metainformation), innan ni skapar några kataloger.

[3] Se även slides om bitmanipulering från IOOPM 2012, föreläsning 19

Musikfiler

I mappen `/it/kurs/imperoopmet/music` ligger ett antal filer i MP3-format. Ni kan använda dessa filer för när ni provkör ert program. De innehåller alla samma ljudinformation, men har olika ID3-taggar. Allt textinnehåll använder vanlig ASCII, och ID3-taggen kommer först i filen. När ni ska testa att bygga upp ert musikbibliotek kan ni kopiera hela mappen till en katalog på ert konto och köra programmet.

Teckenkodning

En teknisk detalj är att textinnehållet i en ID3-frame ibland använder en teckenkodning som inte är direkt kompatibel med ASCII. Det innebär att det kan vara svårt att läsa och använda textinformationen som den är skriven. Om ni vill testa er kod på andra musikfiler än de som nämnts ovan tillhandahåller vi ett bibliotek för att konvertera textinnehållet i en ID3-frame till UTF-8 (där alla tecknen i ASCII-tabellen kodas likadant som ASCII)[4].

Genom att inkludera `convert_content.h` och kompilera programmet tillsammans med `convert_content.c` och flaggan `-liconv` (som länkar in biblioteket som används för kodningskonvertering) får ni tillgång till funktionen `convert_content`. Hur funktionen används står i `convert_content.h`.

Förslag till redovisningar kopplade till denna uppgift

Nedan beskriver vi några möjliga kunskapmål vars uppfyllnad kan redovisas som en del av denna uppgift. Listan är inte på något sätt uttömmande. Det är heller inte så att man måste redovisa de kunskapmål som nämns nedan i samband med denna uppgift, eller att denna uppgift är ”den bästa” att redovisa dessa i samband med.

A1 (Procedurell abstraktion) Förmodligen kommer ni inte vilja ha all kod i en och samma funktion, utan istället ha programmet uppdelat i flera funktioner som alla utför varsin deluppgift. Komplicerade bitoperationer kanske också bör abstraheras i funktioner eller makron för att göra koden mer lättläst.

F14 (Rekursion och iteration) Flera av delfunktionerna i programmet gör någonting upprepade gånger, till exempel läser in en byte eller frame. I ML skulle ni förmodligen ha löst upprepningen med rekursion, medan det i C är vanligare med iteration. Fundera på för- och nackdelar med att använda det ena eller det andra i olika funktioner.

J27 (Stack och heap) Viss information som ni läser kommer ni vilja spara tillfälligt i variabler (på stacken), medan annan information måste sparas på ett mer bestående sätt (på heapen). Fundera på hur långt man kan komma genom att bara allokera data på stacken (och om det är en bra idé).

M37 (Pekare och arrayer) Det är troligt att ni kommer använda både arrayer och egendefinierade structar i ert program. Hur gör ni för att indexera i dessa strukturer? Vad översätter kompilatorn det till? Om ni inte har några egna structar eller arrayer kan ni skriva programmet så att man ger musikfilerna som argument till programmet, och resonera om argumentvektorn `argv` i `main`.

M39 (Pekare till stackvariabler) För att överföra information mellan olika funktionsanrop kan man antingen ta vara på vad funktionen returnerar, eller skicka med en ”behållare” som funktionen fyller själv. I det senare fallet kan man åstadkomma en funktion som ”returnerar” flera värden än ett.

[4] Mer matnyttig läsning om teckenkodning finns i den här bloggposten:
`http://www.joelonsoftware.com/articles/Unicode.html`