LABORATION 3

Läs in metadata från bildfiler (.bmp & .png)

Inledning

Vi vill skapa en console applikation som tar ett (sökväg)filnamn som argument. Programmet ska sedan, genom att läsa in binärdata, avgöra om filen man angivit är en .bmp-fil eller en .png-fil, samt ange höjd och bredd på bilden i pixlar. Om filen varken är en .bmp eller .png så skriver vi det.

Exempel output:

“File not found.”
”This is not a valid .bmp or .png file!”
“This is a .png image. Resolution: 800x600 pixels.”

ASCII vs Binärdata

Om vi till exempel vill lagra information om en bilds upplösning så skulle vi kunna skriva en textsträng ”1280x720” till en textfil. Den datan kan vi sedan läsa in igen i ett program och omvandla till heltalen 1280 och 720, men den har också fördelen att vi kan öppna filen i en texteditor och direkt se att det står ”1280x720”.

Om man vill ha så små filer som möjligt så är detta dock inte det mest effektiva sättet att lagra data eftersom varje (ASCII) tecken är 1 byte, medan vi vet att vi hade kunnat lagra det som två 16 bitars tal, och kommit undan med totalt 4 byte om vi lagrar det som binär data. (Nackdelen är att det blir svårläst i en texteditor).

Eftersom bilder innehåller mycket data, så lagras dessa nästan alltid i något binärt format. Vanliga bildformat är t.ex. gif, jpg, bmp, och png. Hur den binära informationen ska tolkas (vilka bytes betyder vad?) bestäms av de som skapat formatet och går att hitta om man läser specifikationen.

PNG (Portable Network Graphics)

Om man googlar och läser specifikationen för .png formatet så hittar man att dessa filer alltid startar med en 8 byte lång signatur, som alltid är samma och därför kan användas för att identifiera png filer.

Signaturen följs sedan av ett godtyckligt antal ”chunks” beroende på filens innehåll. Man kan i spec:en hitta att varje sådan chunk har en viss struktur, bland annat kan man läsa ut hur lång en chunk är, och därmed hitta hur många byte framåt man behöver läsa i filen för att hitta nästa chunk (de ligger efter varandra, men kan vara olika långa, och ha olika innehåll).

Själva bilddatan i en png-fil är komprimerad och kan vara uppdelad över flera chunks. Att läsa ut den kräver att man implementerar komplicerade algoritmer och är långt utanför ambitionsnivån på den här uppgiften. Det vi istället ska göra är att försöka hitta och läsa ut så kallad meta-data (data om datan).

Uppgiften blir att försöka läsa sig till i specifikationen och försöka hitta i vilken chunk information om bildens bredd och höjd ligger. Det är en betydligt enklare uppgift, och den informationen tillsammans med ID:t är allt du behöver för att lösa uppgiften.

Andra delen av uppgiften

Programmet ska även kunna identifiera och skriva ut upplösning på .bmp filer. Se om ni kan hitta specifikationen för formatet med hjälp av google.

.bmp filer är enklare uppbyggda och använder inte chunks så som png gör, utan har istället en ”header”. En header är ett (enligt spec.) givet antal byte som oftast talar om var i filen man hittar de andra delarna man söker efter. Till skillnad från chunks kan man alltså hoppa direkt till det man letar efter istället för att gå igenom varje chunk för att hitta nästa. Både headers och chunks förekommer i olika form även i många andra filformat.

Försök läsa er till i specifikationen för .bmp-formatet hur man kan identifera att det är en .bmp-fil, samt var man kan hitta metadata om bredd och höjd.

VG-delen av uppgiften

För VG så ska programmet, när det identifierat en .png, (förutom att skriva ut upplösning) även lista alla chunks (typ och storlek) i den ordning som de förkommer i filen.

Tips & Hjälp

Skapa bilder i Paint i olika storlekar och spara som .bmp eller .png för att ha filer att testa på.

Ni kan även ladda ner .bmp eller .png från nätet. För att verifiera upplösningen, högerklicka på filen i windows, välj egenskaper och fliken ”Information”.

Använd Visual Studios inbyggda hexeditor för att visa binära filer. Där kan ni se filernas innehåll byte för byte och jämföra med specifikationen. Ett alternativ till Visual Studios hexeditor är Notepad++ med tillägget *NPP_HexEdit*. Dessutom kan Windows Kalkylatorn vara användbar, när man har den i läget **Programmerare**.

Med offset avses oftast positionen i antal bytes räknat från filens början (när ni läser specifkationer).

Ofta nämns i specifikationen i vilken ordning byte ska läsas för att läsa in t.ex. en 32-bitars integer. Man pratar då om MSB (Most Significant Byte) och LSB (Least Significant Byte). MSB är alltså den byte (av fyra) i ett 32 bitars tal som är värd mest. LSB är den byte som är värd minst.

Använd klassen FileStream för att läsa in binära data från en fil.

Redovisning

Lämna in uppgiften på ithsdistans med en kommentar med GitHub-länken.

Jag föreslår att ni jobbar i grupper om 2–3 personer så ni kan hjälpas åt att förstå hur man ska läsa filerna och diskutera ihop er om lösning. I så fall måste alla i gruppen skicka in länken till GitHub, samt vem/vilka ni jobbat med.

Betygskriterier

För godkänt:

- Koden ska fungera enligt ovan beskrivning.
- Den ska identifiera .bmp och .png filer (även om filändelsen inte stämmer).
- Den ska skriva ut bredd och höjd (i pixlar) på bilderna den identifierar.
- Man får inte använda Image eller andra bildbibliotek för att läsa ut data.

För väl godkänt krävs även:

- Programmet ska även lista typ (t.ex. IDAT) och storlek på alla chunks i .png filer. Storleken ska skrivas ut i antal bytes (decimalt) och gäller hela chunkens storlek (inte bara storlek på data).
- Koden ska vara väl strukturerad och lätt att förstå
- Lösningen ska inte innehålla massa onödig kod.
- Allting ska fungera vid inlämning. Gör det inte det så kommer ni få tillbaks uppgiften med möjlighet att fixa det som krävs för godkänt. Men ni har alltså bara en chans på er att få VG. Så var noga med testning.
- Inlämning sker före deadline.

TIPS

- Läs på ordentligt innan ni ens börjar att fundera över koden
- Gör en ordentlig design av er lösning innan ni börjar att koda
    - Visual Studio har ett inbyggt verktyg för att skapa klassdiagram. Det är dock inte tillgängligt automatiskt i alla Core projekttyper. Lösningen på det problemet finns beskrivet på

https://fmoralesdev.com/2019/05/16/generate-class-diagram-vs2019-net-core/

- För den officiella beskrivningen av PNG, gå till https://www.w3.org/TR/PNG/
    - https://www.w3.org/TR/PNG-Introduction.html och efterföljande sidor beskriver filformatet i mer detalj