Ada.O8.11: Svartvita bildfiler (PBM och PBMa)

I denna uppgift kommer du att lära dig hur man hanterar textfiler (och kanske en del om hur bilder kan lagras, läsas, etc. i program och på filer).

Mål

I denna uppgift är målet att du skall lära dig:

- hur man öppnar och stänger textfiler.
- hur man läser och skriver till textfiler.
- hur man hanterar radslut och filslut i textfiler.

Uppgift

I denna uppgift ska du skapa ett program som hanterar bilder. Uppgiften är uppdelad i 4 delar.

Del 1:
Denna del går att testa mot automaträttningen innan resten av delarna är klara.

Här använder vi delar av det som gjordes i Ada.O11.2 (kommandoradsargument). Kolla där för mer information och kopiera de delar som är relevanta här till en ny mapp för just denna uppgift.

Del 2:
Detta motsvarar i princip en Ada.P3-uppgift, men bygger givetvis på uppgifterna Ada.O3.1 och Ada.O3.2. Kolla där för mer information om du inte kommer ihåg hur man hanterar poster och fält (inklusive matriser).

Observera att du måste skicka med alla ".adb"- och ".ads"-filer när du skickar in uppgiften till automaträttningen.

Delarna 3.1 och 3.2:
I dessa bör man tänka att man ska ha ett utskriftsunderprogram per datatyp man själv har skapat.

Delarna 4, 4.1 och 4.2:
Här bör man tänka att man ska ha ett inläsningsunderprogram per datatyp man skapat.

I denna uppgift kommer du endast att hantera svartvita bilder, men det gäller att bygga detta på ett generellt sätt så att man kan gå vidare till Ada.O8.12 (som bygger HELT på Ada.O8.11, men där vi också skall klara av att läsa in bilder i färg).

Det är viktigt att kunna hantera bilder i denna uppgift som är av två format. Dels sådana som är rent svartvita och dels sådana som har en såkallad "alpha-kanal" (är genomskinliga).

Datafiler med bilder finns på uppgiftens sida, under de givna filerna. För att få tag på dessa är det enklast att göra "download" på dessa, alltså att högerklicka på filnamnet och sedan välja "`download as...`", "`ladda ner som...`", "`spara som...`" eller liknande.

Bildfilerna kan öppnas i emacs, men när de öppnas kommer dessa visas som bilder istället för text. För att se texten i filen kan du i emacs ge kommandot: `M-x text-mode`

Del 1

I denna del skall du skapa de delar i programmet som tar emot data från kommandoraden och kontrollera att det inte är några felaktigheter i dessa.

Tanken är att ditt program skall ta emot ett eller två kommandoradsargument och att det första av dessa är ett filnamn som representerar en bild som senare skall läsas in från filen till programmet. Det andra argumentet är det filnamn som resultatet av programmet skall skrivas ut till. Om det andra kommandoradsargumentet saknas skall utskriften av resultatet istället ske direkt i terminalen.

Du kan anta att programmet körs ifrån samma mapp som textfilerna finns i och skrivs till. Observera att man kan välja att skriva "`./`" framför filnamn. Detta betyder att "filen ligger i den här mappen". Nedan följer ett antal exempel på hur själva kommandoradshanteringen skall fungera.

Körexempel 1 (för få kommandoradsargument):

```
Terminalens prompt %  ./image_program
Error! Incorrect number of arguments!
Usage: ./image_program IMAGE_FILENAME [OUTPUT_FILENAME]
```

Körexempel 2 (för många kommandoradsargument):

```
Terminalens prompt % ./image_program object.pbm result.txt file.txt
Error! Incorrect number of arguments!
Usage: ./image_program IMAGE_FILENAME [OUTPUT_FILENAME]
```

Körexempel 3 (försöker läsa programmet självt som bildfil):

```
Terminalens prompt % ./program program
Error! Input file "program" cannot be same as the program itself!
```

Körexempel 4 (försöker läsa programmet självt som bildfil):

```
Terminalens prompt % ./program ./program
Error! Input file "./program" cannot be same as the program itself!
```

Körexempel 5 (utdatafil samma som bildfil):

```
Terminalens prompt % ./program ORIGINAL.TXT ./ORIGINAL.TXT
Error! Output file "./ORIGINAL.TXT" cannot be same as input file!
```

Körexempel 6 (utdatafil samma som programmet självt):

```
Terminalens prompt % ./program ./ORIGINAL.TXT program
Error! Output file "program" cannot be same as the program itself!
```

Körexempel 7 (bildfilen existerar ej):

```
Terminalens prompt % ./program none.pbm
Error! Input file "none.pbm" does not exist!
```

Körexempel 8 (bildfilen existerar ej):

```
Terminalens prompt % ./program none.pbm OUTPUT.TXT
Error! Input file "none.pbm" does not exist!
```

Körexempel 9 (utdatafil existerar redan):

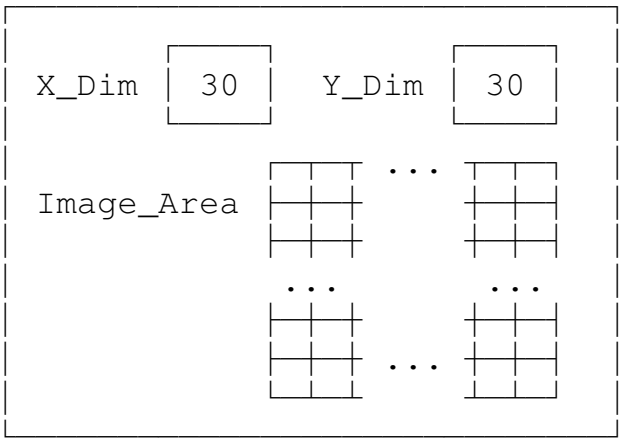
```
Terminalens prompt % ./program object.pbm OUTPUT.TXT
Error! Output file "OUTPUT.TXT" already exists!
```

Del 2

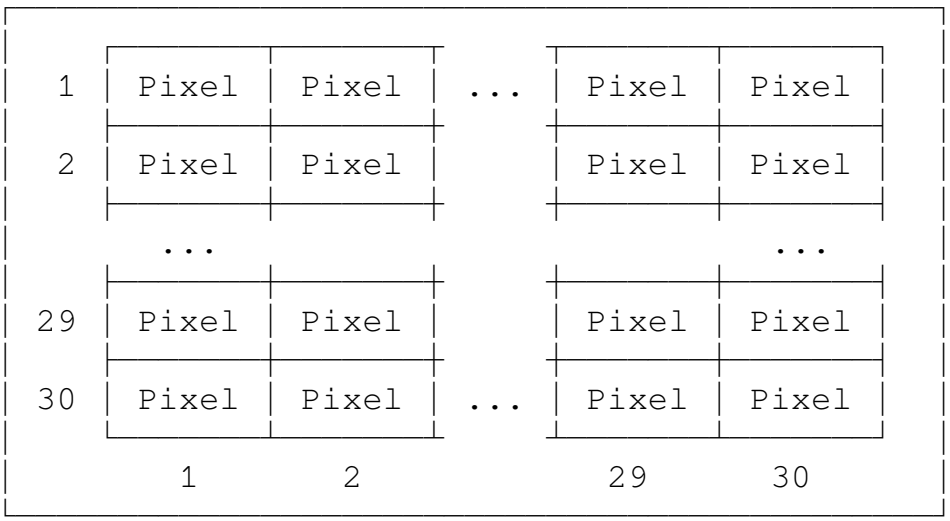
I denna del skall du skapa ett paket som innehåller en datastruktur (datatypen `Image_Type`) representerade en bild. Det är självklart så att `Image_Type` ska vara privat i paketet. Ett lämpligt namn på paketet är `Image_Handling`.

Formatet på datastrukturen skall vara enligt följande bilder:

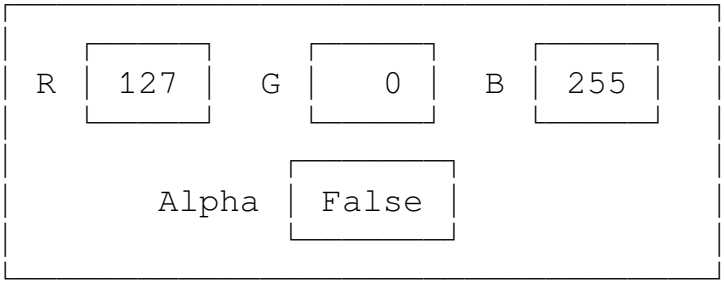
`type Image_Type:`



`type Image_Area_Type:`



`type Pixel_Type:`



I denna uppgift kan du anta att bilderna som ska hanteras består av exakt 30 x 30 pixlar.

Del 3.1

I denna del ska du lägga till ett underprogram som skall heta `Print_Image_Information` (i paketet du skapade i del 2) som ska skriva ut innehållet i datastrukturen ifrån del 2 i terminalen. Varje pixel i datastrukturen ska skrivas ut på formatet:

```
␣RRR␣GGG␣BBB␣ALPHA
```

Självklart skall `RRR` ersättas med värdet i `R` för en pixel och på motsvarande sätt för `GGG` och `BBB`. `ALPHA` skall ersättas med antingen "`True␣`" eller "`False`". Tecknet '`␣`' ersätter ni förstås med blanktecken, dessa står med i exemplet ovan (och nedan) för att ni enklare ska kunna se antalet blanktecken som skall skrivas ut.

Alla pixlar som motsvarar en rad i bilden skall förstås skrivas ut som en rad i terminalen och därefter skall ett radslut (ENTER) skrivas ut. Om vi antar en bild med två pixelrader där varje rad består av två pixlar skulle utskriften kunna se ut på följande sätt:

```
␣␣12␣100␣␣␣0␣False␣100␣200␣255␣True␣
␣111␣␣␣5␣␣45␣True␣␣␣␣8␣␣10␣␣␣0␣False
```

I ditt program kommer utskriften förstås att bestå av 30 pixelrader där varje rad består av 30 pixlar.

Du skall endast ha en parameter till detta underprogram och det är själva bildvariabeln.

OBS! Självsklart ska ditt program använda innehållet i datastrukturen för att avgöra hur mycket data som ska skrivas ut (variablerna `X` och `Y` i datastrukturen anger dimensionerna i x- respektive y-led).

Del 3.2

I denna del ska du lägga till ytterligare ett underprogram `Print_Image_Information` (dv.s. med samma namn som det som nu redan finns i paketet du skapade i del 2) som ska skriva ut innehållet i datastrukturen ifrån del 2 på en textfil. Det ska vara samma format på utskriften till textfilen som utskriften i del 3.1 ovan.

Det skall vara givet att filen är öppen för skrivning innan du anropar denna version av `Print_Image_Information` (du skall alltså INTE öppna filen inuti detta underprogram). Du skall endast ha två parametrar till detta underprogram, filvariabeln och bildvariabeln.

OBS! Textfilen du skall spara utskriften på är det andra kommandoradsargumentet ifrån del 1 (om detta finns).

Observera att duplicering av kod inte är ok inne i paketet. Detta gör att du behöver strukturera om lite i din programkod i värsta fall, men ändra inte saker som gör att det bryter mot tidigare deluppgifter.

Del 4

På hemsidan för denna uppgift finns material som beskriver hur bildfiler av formaten PBM (rena svartvita bilder) och PBMa (svartvita bilder med genomskinliga pixlar) ser ut. Läs detta material innan du går vidare med uppgiften. Vi börjar med lite allmän information för båda dessa format.

Det finns alltså två olika bildformat (PBM och PBMa) som ska hanteras i denna uppgift. Oavsett vilket format som ligger på filen du skall läsa in gäller att lagringen av bilden skall ske i datastrukturen du skapade i del 2. Tanken är att du skall dela upp ditt program i "vettiga" underprogram som utför lämpliga delar av uppgiften.

Det är viktigt att det endast finns ett underprogram som kan anropas från huvudprogrammet när man skall läsa in en bild (oavsett vilket format som är i filen man läser in). Detta gör att du behöver göra vissa delar av programmet med två olika inläsningsvarianter då PBMa har lite extra data som skall behandlas vid själva inläsningen. OBS! Det är INTE ok att ha all kod duplicerad så när man väl läst in data från filen skall man sen stoppa in datat i datastrukturen med generella underprogram som inte skiljer sig åt beroende på vad det var för format på filen.

OBS! För att få olika färger skall man se till att lagra värden i `R`, `G` och `B` i en pixel. En kort lista av värden `(R, G, B)` som ger olika färger kommer här:

```
(255, 255, 255)  vit
(255,   0,   0)  röd
(  0, 255,   0)  grön
(  0,   0, 255)  blå
(  0,   0,   0)  svart
```

Om man har värden mellan 0 och 255 blir färgen inte lika stark och blandar man färgerna blir det andra färger. Du kan skapa egna filer med data för att få till just en bild du gillar. :-)
I denna uppgift hanterar vi endast svartvita bilder men programmet ska vara skrivet så denne även kan hantera bilder med andra färger.

När man sätter värdet `True` i variabeln `Alpha` i en pixel motsvarar detta att den pixeln är transparent (genomskinlig). Värdet `False` motsvarar att det är en icke transparent pixel. I PBM är `Alpha` alltid `False` medan i PBMa kan detta värde variera.

Ditt program skall från kommandoraden ta emot vilken bildfil som skall läsas (se del 1 för detaljer). Därefter skall alltså just den filen läsas och "omvandlas" till data i ovanstående datastruktur. När detta är gjort skall en utskrift av innehållet i din datastruktur skrivas ut antingen på en angiven textfil (om det andra kommandoradsargumentet är angivet) eller i terminalen (om det andra kommandoradsargumentet utelämnats).

Inläsningen av bilden skall göras i ett underprogram som heter `Read` som har två parameterar, filnamnet där bilden finns att hämta (läsa in) och bildvariabeln där bilden skall lagras. Underprogrammet `Read` ska förstås ligga i paketet som skapades i del 2.

OBS! Det är innehållet i filen som avgör formatet för bilden, inte filnamnet. Det är också viktigt att läsa data ifrån filen och fylla datastrukturen på korrekt sätt. Programmet ska inte anta att det är exakt 30 x 30 pixlar (även om detta är givet i denna uppgift) utan programmet ska hantera detta generellt utifrån filinnehållet. ÄVEN om det är så att vi endast kommer att ha bilder som är 30 x 30 pixlar i just denna uppgift. Man skall alltså tänka på framtiden när man skriver sina program.

Del 4.1

I denna del skall du endast lösa problemet för bilder i PBM-formatet (d.v.s. rent svartvita bilder utan transparens). Tänk på att dela upp ditt program (dina underprogram i paktet) så att det är specifika delar som löses i varje underprogram. Underprogrammen kan med fördel anropa nya underprogram för att lösa deluppgifter istället för att lösa stora delar i samma underprogram.

Testfallen som kommer efter de i Del 1 ovan hanterar PBM-filer. Dessa testfall följs av testfall som hanterar PBMa-filer (Del 4.2 nedan). När du kommer till testfallen som handlar om Del 4.2 måste du lösa den delen för att kunna fortsätta.

Del 4.2

I denna del skall du skapa "kopior" av de underprogram som behöver modifieras för att klara transparensen som finns i formatet PBMa-filerna. Du skall även detektera vilket format som finns i filen i din `Read` för att anropa rätt version av inläsning av data.

OBS! Duplicering av kod kommer inte att godkännas så nu gäller det att tänka till vad som behöver "kopieras" och vad som är generellt. Diskutera ram via "papper och penna" när ni funderar på hur detta skall göras så sparar ni säkert tid.