Ada.O8.11: Färgbildsfiler (PPM)

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 data från 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 5 delar.

Del 1:

Denna del går att testa mot automaträttningen innan resten av delarna är klara. Här ska programmet ta emot och kontrollera kommandoradsargument.

Del 2:

Detta motsvarar i princip en Ada.P3-uppgift och 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.

Del 3:

I denna del ska en bildvariabels innehåll skrivas ut i textformat. I denna bör man tänka att man ska ha ett utskriftsunderprogram per datatyp man själv har skapat.

Del 4:

I denna del ska innehållet från en bildfil läsas till en bildvariabel. Här bör man tänka att man ska ha ett inläsningsunderprogram per datatyp man skapat.

Del 5:

I denna del ska en bildvariabels innehåll skrivas ut som en bild. Här ska TJa-biblioteket användas för att hantera färger och utskrifter av lite annan typ.

I denna uppgift kommer du endast att hantera färgbilder av en viss storlek, 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 svartvita bilder samt hantera genomskinlighet.

Datafiler med bilder finns på uppgiftens sida, under de givna filerna. För att få tag på dessa filer är det enklast att göra "download", 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 är korrekt antal angivna.

Tanken är att ditt program skall kunna ta emot både två och fyra kommandoradsargument. Det första av dessa är ett filnamn som representerar en bild. Denna bild skall senare läsas in från filen till programmet. Det andra argumentet anger vilket sätt bilden ska skrivas ut på i terminalen. Om tredje och fjärde argumenten anges är det givet att andra argumentet är ett jämnt tal. Dessa två sista argument anger var i terminalen den inlästa bilden ska skrivas ut.

Du kan anta att programmet körs ifrån samma mapp som den angivna filen finns i. Du kan även anta att det första argumentet anger en existerande fil och det andra argumentet anger ett heltal. Du kan även anta att det tredje och fjärde argumenten anges som heltal och är korrekt inmatade på kommandoraden. 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 N [X Y]
```

Körexempel 2 (Fel antal kommandoradsargument, annat programnamn):

```
Terminalens prompt %  ./other_program_name object.ppm
Error! Incorrect number of arguments!
Usage: ./other_program_name IMAGE_FILENAME N [X Y]
```

Körexempel 3 (Fel antal kommandoradsargument, annat programnamn):

```
Terminalens prompt %  ./another_name object.ppm 2 10
Error! Incorrect number of arguments!
Usage: ./another_name IMAGE_FILENAME N [X Y]
```

Körexempel 4 (för många kommandoradsargument):

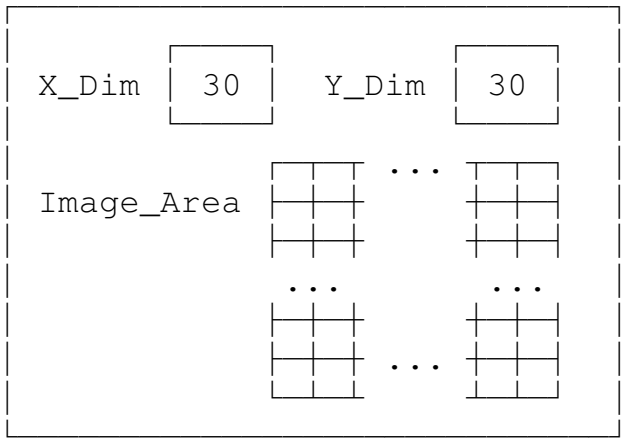
```
Terminalens prompt % ./image_program object.ppm 2 10 3 hello
Error! Incorrect number of arguments!
Usage: ./image_program IMAGE_FILENAME N [X Y]
```

Del 2

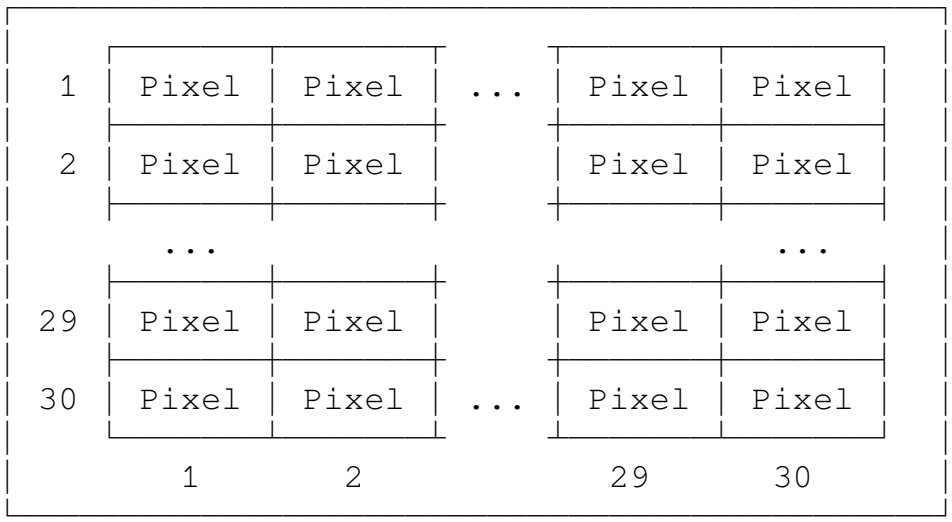
I denna del skall du skapa ett paket som innehåller en datastruktur (datatypen `Image_Type`) representerade en bild. Det är självklart så att `Image_Type` ska vara privat i paketet. Paketet ska ha namnet `Image_Handling`.

Formatet på datastrukturen skall vara enligt följande bilder:

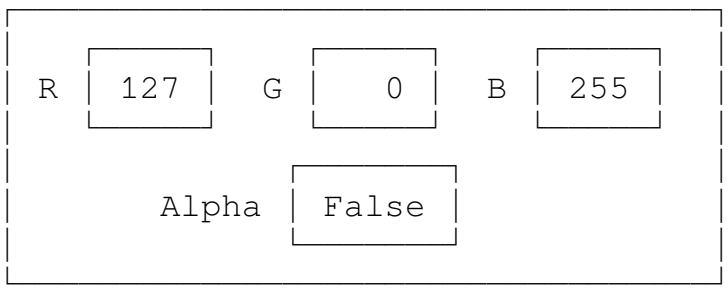
`type Image_Type:`



`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

I denna del ska du lägga till funktionalitet för att skriva ut en bilds innehåll på textformat. Om användaren anger ett ojämnt heltal som det andra kommandoradsargumentet ska denna typ av utskrift ske. Du ska skapa 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` motsvarar huruvida en pixel är genomskinlig (`True␣`) eller ej (`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.

I denna uppgift kommer vi inte använda bilder som är genomskinliga på något sätt utan detta är en förberedelse inför efterföljande uppgifter.

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

Del 4

På hemsidan för denna uppgift finns material som beskriver hur bildfiler av formaten PPM (färgbilder) ser ut. Läs detta material innan du går vidare med uppgiften. Vi börjar med lite allmän information för detta format.

I denna uppgift ska ditt program hantera ett bildformat (PPM). Datat ifrån filen som angavs som första kommandoradsargument skall läsas in och lagras 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.

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. :-)

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 datastrukturen från del 2. När detta är gjort skall en utskrift av innehållet i din datastruktur skrivas ut på ett av två olika sätt beroende på vilken siffra användaren angav som andra kommandoradsargument (se del 3 och del 5).

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.

I denna uppgift kan du anta att alla bildfiler har 256 "färgnivåer" för RGB-värdena så ingen omvandlig av dessa värden krävs.

Del 5

I denna del ska du lägga till funktionalitet för att skriva ut en bildvariabels innehåll som en bild i terminalen.

Om användaren anger ett jämnt heltal som det andra kommandoradsargumentet ska denna typ av utskrift ske. Du ska skapa ett underprogram som skall heta `Print` (i paketet du skapade i del 2) som ska skriva ut innehållet i datastrukturen ifrån del 2 i terminalen som en bild. Underprogrammet ska ta tre parameterar: Själva bilden som ska skrivas ut samt två heltal vilka anger vart på skärmen denna bild ska skrivas ut. Heltalen anger X- och Y-led (kolumn och rad, i den ordningen) där bildens övre vänstra hörn ska skrivas ut. Dessa heltal fås ifrån kommandoraden och matas in som de tredje och fjärde kommandoradsargumenten.

Vi koncentrerar oss på själva utritningen av en bild. I detta fall innebär "utritning" att vi skriver ut tecken i terminalen, men ser till att modifiera vilken färg dessa tecken skrivs ut med. För att klara uppgiften behöver du titta på `TJa`-biblioteket (finns beskrivet på uppgiftens sida i kurskartan) och se vad som finns i detta.

I denna uppgift ska programmet skriva ut två punkter ('`.`') per pixel i bilden. Detta för att underlätta felsökning för er då en punkt syns bättre än ett blanktecken (även om blanktecken kan ge "snyggare" bilder).

Antag att vi har en 4x4-pixel stor bild i formatet `PPM` som ser ut enligt följande (på filen `bild.ppm`):

```
P3
# Minibild
4 4
255
  0   0   0   0   0   0 254   1   1 255 255 255
254   1   1 254   1   1   2 253   2   3   3 252
  0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0
```

I pixeldatat ovan har vi satt följande RGB-värden för respektive färg för att det skall gå att finna var de olika delarna finns:

- `[0, 0, 0]       => svart (S)`
- `[254, 1, 1]     => rött  (R)`
- `[2, 253, 2]     => grönt (G)`
- `[3, 3, 252]     => blått (B)`
- `[255, 255, 255] => vitt  (V)`

Antag att vi skall skriva ut bilden så att det övre vänstra hörnet skall börja på position (10, 3) i terminalen, d.v.s. kolumn 10 och rad 3 från överkanten i terminalen. Detta skulle då se ut enligt följande (punkter motsvarar positioner i terminalen utanför bilden):

```
..........................................................................
..........................................................................
..................SSSSRRVV................................................
..................RRRRGGBB................................................
..................SSSSSSSS................................................
..................SSSSSSSS................................................
..........................................................................
..........................................................................
..........................................................................
..........................................................................
..........................................................................
```

Vi funderar lite på vad som kommer att behövas för att lösa detta.

För att skriva ut 1:a pixeln (i första raden i bilden):
Positionera markören i terminalen på (10, 3). Sätt färg på det som skrivs ut till färg (0, 0, 0). Skriv ut punkt på nuvarande position.

För att skriva ut 2:a pixeln (i första raden i bilden):
Positionera markören i terminalen på (11, 3). Sätt färg på det som skrivs ut till färg (0, 0, 0). Skriv ut punkt på nuvarande position.

För att skriva ut 3:e pixeln (i första raden i bilden):
Positionera markören i terminalen på (12, 3). Sätt färg på det som skrivs ut till färg (254, 1, 1). Skriv ut punkt på nuvarande position.

Vi ser att detta kommer att bli enkelt att utföra lite mer algoritmiskt genom att låta positionen i x-led öka för varje pixel i bilden och man gör sen samma sak för varje pixel. Funderar man vidare på detta ser man att det sen blir så att man ökar y-värdet när raden är slut och vi kommer att få ut hela bilden med några få rader kod.

Ditt program ska fungera för bilder som är 30x30 pixlar stora, men själva tänket är detsamma som ovan.

*Om man tittar lite på vad som görs i denna enkla, men ineffektiva algoritm, så finns det några saker som tar "tid". T.ex. att flytta markören till en viss position på skärmen. Det tar förstås också tid att byta färg varje gång man är på väg att skriva ut en pixel. Antag att det tar tiden T att verkligen skriva ut en punkt. Att bara förflytta sig till en ny position för markören tar då minst 6*T (troligen mer). Att byta färg tar som minst 13*T. För vår lilla bild (med 20 pixlar) skulle detta som minst ta 4 * 5 * (6*T + 13*T + T) = 400*T att skrivas ut (och det är endast 20 pixlar). Tänk en stor bild med kanske 500x400 pixlar (200.000 pixlar) där det blir 4_000_000*T.*

Körexempel 5:

```
Terminalens prompt % ./image_program bild_30x30.ppm 2 10 3
```