Ada.O8.12: Bildfiler, forts. (PPM, PBM, PPMa och PBMa)

Denna uppgift är en förlängning av Ada.O8.11 för hantering av bilder.

Mål

I denna uppgift är målet att du skall lära dig:

- hur man modifierar/lägger till/effektiviserar saker i ett paket.
- återanvändning av releventa delar i redan befintlig kod.

Uppgift

I denna uppgift ska du lägga till och modifiera det paket du skapade i Ada.O8.11 för att även klara av svartvita bilder och bilder av olika storlekar samt lägga till lägga till ytterligare funktionalitet. Programmet ska förutom detta hantera genomskinlighet samt skalning av pixlarnas RGB-värden. De bildformat som skall hanteras (PPM, PPMa, PBM, PBMa) beskrivs mer detaljerat på hemsidan. **Läs på om de olika formaten på denna uppgifts sida i kurskartan innan du börjar på själva uppgiften.**

Uppgiften är uppdelad i 5 delar enligt följande:

Del 1 - Bilderna kan nu ha andra dimensioner än just 30x30 pixlar. Bildfilerna innehåller information om bildens storlek och denna information ska användas i programmet för att anpassa inläsning och utskrift. Datastrukturen ska modifieras för att kunna hantera olika bilddimensioner.

Del 2 - Utskrift av bilder ska göras smartare vad gäller byte av färg för pixlarna samt markörförflyttning.

Del 3 - Fyra bildformat ska hanteras:

`PPM` - färgbildfiler (mestadels hanterad i tidigare uppgift)
`PPMa` - färgbildfiler med genomskinliga pixlar
`PBM` - svartvita bilder
`PBMa` - svartvita bilder med genomskinliga pixlar

Formaten PPMa och PBMa introducerar genomskinliga pixlar. Programmet ska ta hänsyn till genomskinlighet vid inläsning och utskrift av bilder.

Del 4 - I färgbildsfiler finns en rad som beskriver hur många "färgnivåer" som finns representerade i filen. Programmet behöver nu ta hänsyn till detta vid inläsningen då nivån kan vara annat än just 255. Detta innebär att du behöver skala upp de pixelvärden som finns i bildfilen så att värdena i din datastruktur sprids över hela spannet [0, 255].

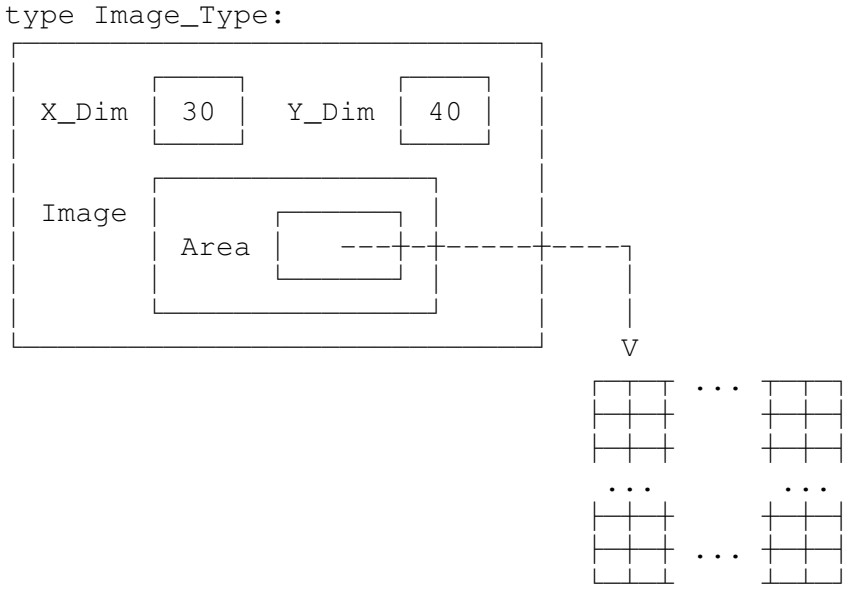
Del 5 - Paketet ska utökas med lite ny funktionalitet.

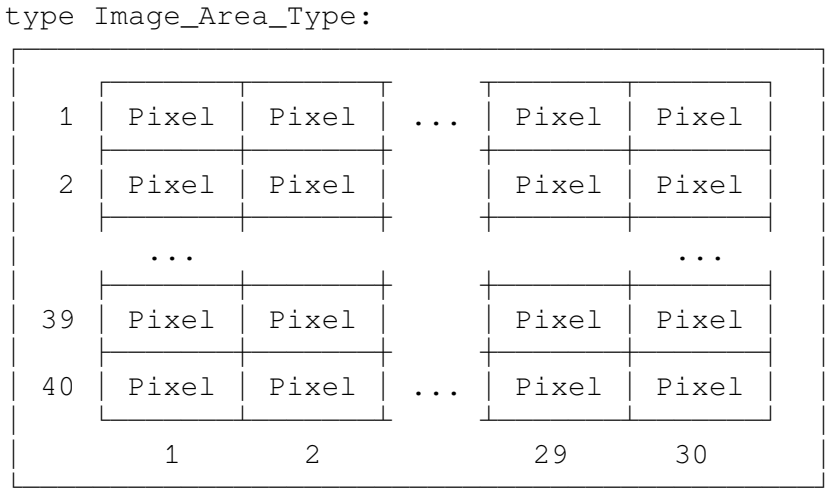
OBS! För att underlätta och kunna komma tillbaka till det gamla om det går snett någonstans är det lämpligt att skapa en ny mapp för denna uppgift och kopiera de filer som är relevanta från tidigare uppgift till denna nya mapp.

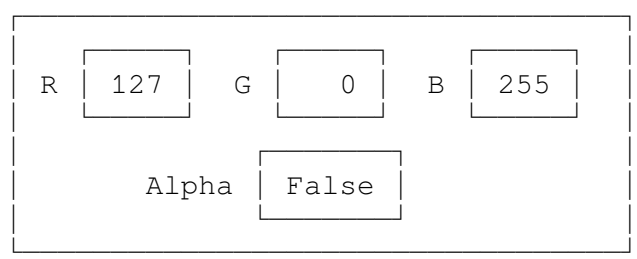
Del 1 - Uppdateringar av datastrukturen

Modifiering av datastrukturen för att klara av bilder av olika storlekar behöver göras. Detta kommer att kräva att du modifierar lite i både “ads” och “adb” gällande paketet som läser in bilder. I Ada.O8.11 var förutsättningen att det alltid var bilder med storleken 30x30 pixlar men nu kan det vara bilder av olika storlekar som ligger på filerna.

Datatstrukturen kommer att se ut på följande sätt när du är klar med modifieringarna (jämför med bilden på datastrukturen i Ada.O8.11 för att se var du behöver ändra saker).







För att lösa detta behövs att man alltså använder pekare och att man skapar själva matrisen efter det att man läst in dimensionen från bildfilen. Det behövs alltså några extra “lager” i datastrukturen vi hade tidigare.

OBS! `Image` är en post som innehåller `Area` som är en pekare som refererar till (pekar på) en `Image_Area_Type` när vi väl skall läsa in eller när vi läst in en bild. `Area` kan förstås vara `null` om man inte har en bild att tillgå.

Anledningen till att ha en post med en pekare inuti är att man direkt kan sätta ett skönsvärde (defaultvärde) på “delvariablerna” i poster vilket kommer att visa sig vara bra att kunna göra när det gäller dessa bilder.

Efter dessa ändringar behöver du också modifiera lite i inläsningsdelen av bilderna.

En viktig sak att tänka på är också att du kommer att få en minnesläcka i ditt program om du inte lägger till ett nytt publikt underprogram `Delete` som “lämnar tillbaka” (avallokerar) minnet som bilden tar upp. Detta är väsentligt att tänka på för varje bild man läser in.

Lägg till ett publikt underprogram `Image_Exists` som kontrollerar om en `Image_Type` innehåller en bild eller ej (alltså kontrollerar om `Area` är `null` eller ej).

Tips 1: Tänk också på vad som skulle hända om du läser in en bild och sen direkt läser in en ny bild till samma bildvariabel eller om man försöker skriva ut en bild som inte finns. Vad skall göras då? Kanske ett undantag skall kastas? Kanske skall man ta bort den gamla bilden innan man läser in en ny? Skall detta ske automatiskt eller skall det göras av “huvudprogrammet”?

KRAV: Om du skall använda detta paket senare i ett projekt i kursen är det ett krav att lösa minnesläckorna i `Read` enligt tips ovan.

Körexempel 1:

```
Terminalens prompt % ./image_program alternating_30_30.ppm 1
   0   0   0 False 255 255 255 False   0   0   0 False 255 255 255 False   0   0   0
False 255 255 255 False   0   0   0 False 255 255 255 False   0   0   0 False 255 255
255 False   0   0   0 False 255 255 255 False   0   0   0 False 255 255 255 False   0
0   0 False 255 255 255 False   0   0   0 False 255 255 255 False   0   0   0 False 255
255 255 False   0   0   0 False 255 255 255 False   0   0   0 False 255 255 255 False
0   0   0 False 255 255 255 False   0   0   0 False 255 255 255 False   0   0   0 False
255 255 255 False
... (29 rader till)
```

Körexempel 2:

```
Terminalens prompt % ./image_program right_arrow.ppm 1
 255 255 255 False 255 255 255 False 255 255 255 False 255 255 255 False 255 255 255
False 255 255 255 False 255 255 255 False 255 255 255 False 255 255 255 False 255 255
255 False 255 255 255 False 255 255 255 False 255 255 255 False 255 255 255 False 255
255 255 False 255 255 255 False 255 255 255 False 255 255 255 False 255 255 255 False
255 255 255 False 255 255 255 False 255 255 255 False 255 255 255 False 255 255 255
False 255 255 255 False 255 255 255 False 255 255 255 False 255 255 255 False 255 255
255 False 255 255 255 False 255 255 255 False 255 255 255 False 255 255 255 False 255
255 255 False 255 255 255 False 255 255 255 False 255 255 255 False 255 255 255 False
0   0   0 False   0   0   0 False 255 255 255 False 255 255 255 False 255 255 255 False
255 255 255 False 255 255 255 False 255 255 255 False 255 255 255 False 255 255 255
False 255 255 255 False 255 255 255 False
... (10 rader till)
```

Körexempel 3:

```
Terminalens prompt % ./image_program alternating_4_4.ppm 1
   0   0   0 False 255 255 255 False 255   0   0 False   0 255   0 False
   0   0 255 False 255 255   0 False 255   0 255 False   0 255 255 False
   0   0   0 False 255 255 255 False 127 127 127 False 200 100  45 False
 255   0   0 False   0 255   0 False   0   0 255 False 255 255   0 False
```

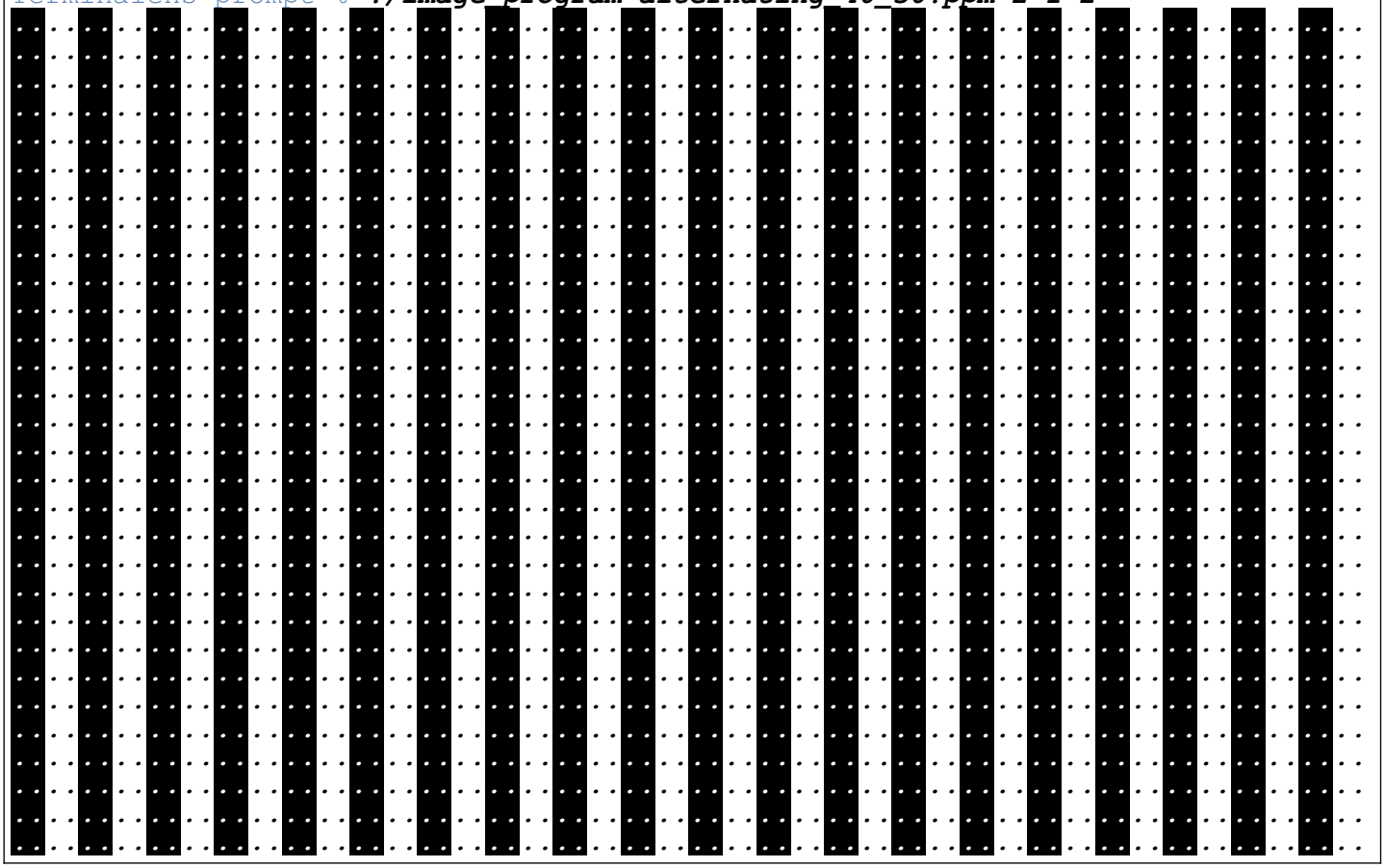
Del 2 - Smartare bildutskrift

Utskriften av bilder ska göras något smartare vad gäller byte av färger och markörförflyttningar. Färgbyte ska endast utföras om nuvarande pixel har annan färg jämfört med färgen på den tidigare utskrivna pixeln och markörförflyttningar ska endast utföras då detta verkligen behövs. I detta läge gäller att markörförflyttningar endast behöver utföras vid utskrift av första pixeln i bilden samt vid byte av rad då markören förflyttas framåt automatiskt då text skrivs ut på skärmen.

Anledningen till ovan ändringar är för att minska antalet instruktioner som utförst då en bild skrivs ut. Se tidigare uppgift för exempel på hur kostsam en bildutskrift kan vara.

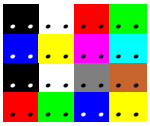
Körexempel 4:

```
Terminalens prompt % ./image_program alternating_40_30.ppm 2 1 2
```



Körexempel 5:

```
Terminalens prompt % ./image_program alternating_4_4.ppm 2 1 2
```

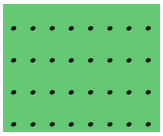


Körexempel 6:

```
Terminalens prompt % ./image_program one_color_1_4.ppm 2 1 2
```

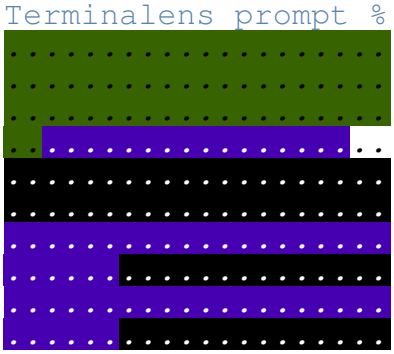
Körexempel 7:

```
Terminalens prompt % ./image_program one_color_4_4.ppm 2 1 2
```



Körexempel 8:

```
Terminalens prompt % ./image_program color_blocks_10_10.ppm 2 1 2
```



Del 3 - Olika bildfilsformat och genomskinlighet

Programmet ska kunna hantera 4 olika bildformat:

- `PPM` - färgbildfiler (mestadels hanterad i tidigare uppgift)
- `PPMa` - färgbildfiler med genomskinliga pixlar
- `PBM` - svartvita bilder
- `PBMa` - svartvita bilder med genomskinliga pixlar

De nya formaten kan läggas till i vilken ordning som helst, men vi rekommenderar att lägga till dessa i samma ordning som listan ovan. **OBS!** Lägg till en i taget och testa programmet allt eftersom för att snabbare komma i mål.

Bildfilerna innehåller information om vilket bildformat som finns lagrad i filen. Programmet ska använda denna information för att läsa in filernas innehåll korrekt. Detta skall förstås hanteras på ett lämpligt sätt (d.v.s. utan alltför mycket duplicering etc.). Det är alltså informationen som är lagrad i filen som bestämmer formatet på bilden, inte filnamnet eller filändelsen.

Del 3a:
Följande gäller PPMa, men ditt program/paket ska givetvis fortfarande fungera för PPM.

För att lägga till det nya formatet måste ny programkod läggas till eller modifieras i ditt program. Vi exemplifierar detta genom att bara prata om `Print` i denna del. Ni måste förstås lösa även övriga delar i paketet såsom t.ex. `Read`.

Tips: För att undvika duplicering av kod kan man skapa nya underprogram för likartade kodavsnitt. Nyttja detta.

Då bilderna nu kan innehålla genomskinliga pixlar behöver utskiftsprogrammet `Print` ta hänsyn till detta. Alla pixlar som är genomskinliga ska "hoppas över" (inte skrivas ut) och nästa efterföljande pixel som inte är genomskinlig ska skrivs ut på korrekt position på skärmen.

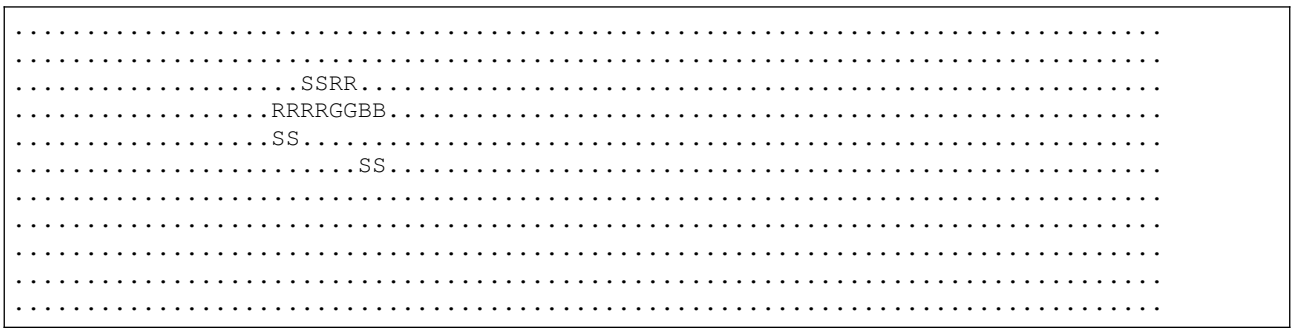
Antag att vi har en 4x4 pixel PPMa-bild som ser ut enligt följande (på filen `bild.ppma`):

```
P3a
# Minibild
4 4
255
0 0 0 1 0 0 0 0 254 1 1 0 255 255 255 1
254 1 1 0 254 1 1 0 2 253 2 0 3 3 252 0
0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1
0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0
```

Följande RGB-värden används i bilden:

- `[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. Då vissa pixlar är markerade som genomskinliga kommer bilden innehålla "hål" där inget har skrivits ut (för denna bild):



Om två bilder skrivs ut ovanpå varandra kommer alltså den underliggande bilden synas vid de genomskinliga pixlarna. Vi funderar lite på vad som kommer att behövas för att lösa detta.

För 1:a pixeln (rad 1, pixel 1): Hoppa över denna pixel då denna är genomskinlig. Inget färgbyte eller positionsbyte ska utföras.

För 2:a pixeln (rad 1, pixel 2): 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 3:e pixeln (rad 1, pixel 3): Markören ska inte positioneras då denna redan står på rätt position efter föregående utskrift. Sätt färg på det som skrivs ut till färg (254, 1, 1). Skriv ut punkt på nuvarande position.

För 4:e pixeln (rad 1, pixel 4): Hoppa över denna pixel då denna är genomskinlig. Inget färgbyte eller positionsbyte ska utföras.

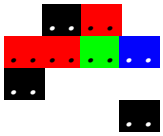
För 5:e pixeln (rad 2, pixel 1): Positionera markören i terminalen på (10, 4). Inget färgbyte ska utföras då denna pixel har samma färg som den tidigare utskrivna pixeln. Skriv ut punkt på nuvarande position.

Del 3b och Del 3c:
Dessa två delar motsvarar Del 3a men med de två sista bildformaten.

Pixlarna är representerade på annat sätt i svartvita bildfiler jämfört med färgbildsfiler vilket leder till att du behöver tänka till om hur du gör inläsning från fil utan att modifiera eller duplicera alltför mycket kod.

Körexempel 9:

```
Terminalens prompt % ./image_program bild.ppma 2 1 2
```

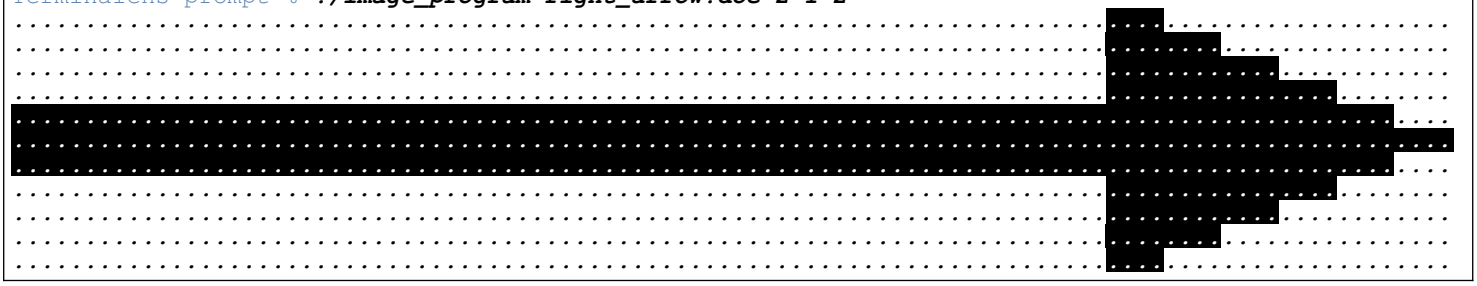


Körexempel 10:

```
Terminalens prompt % ./image_program right_arrow.pbm 2 1 2
```

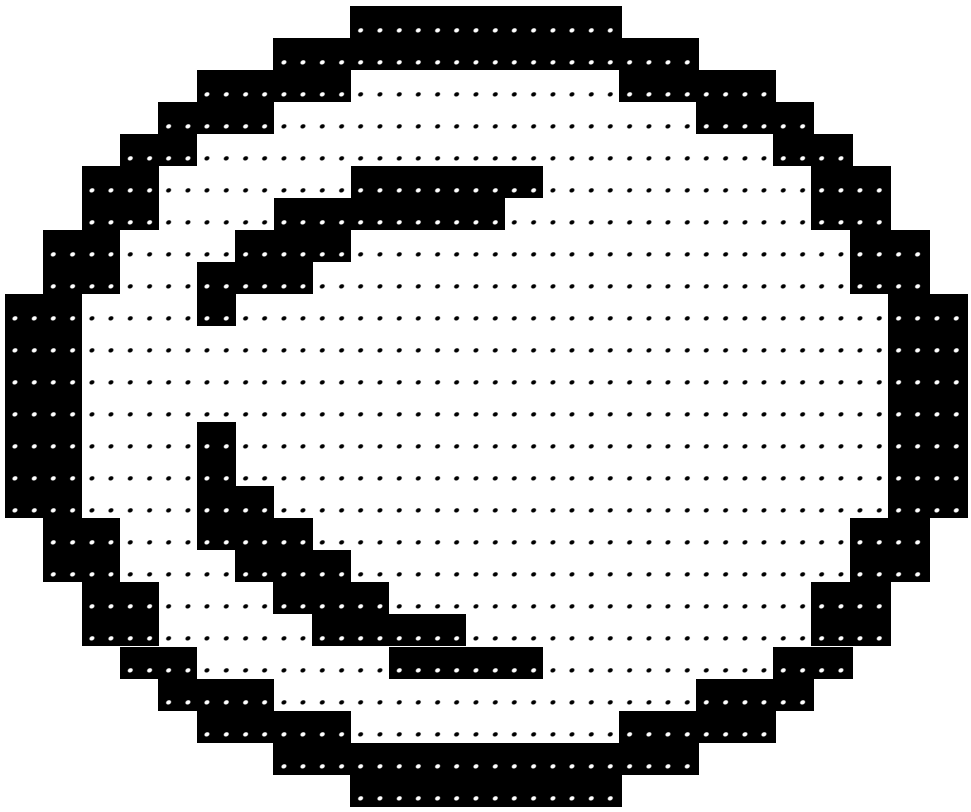
Körexempel 11 (PBM, annan filändelse):

```
Terminalens prompt % ./image_program right_arrow.doc 2 1 2
```



Körexempel 12:

```
Terminalens prompt % ./image_program object_P1a.pbma 2 1 2
```



Del 4 - Olika färgnivåer för bildformaten PPM (och även PPMa)

I denna typ av bildfil finns en rad som talar om hur många “nivåer” av färgnyanser för varje färg det finns i bilden. Denna gäller för alla färger (röd, grön, blå) i en pixel och är ett heltal mellan 1 och 255 (det skulle också kunna vara 0, men det innebär att det endast kan vara en helt svart bild som går att lagra vilket är teoretiskt möjligt, men ni behöver inte ta hänsyn till denna).

Om det står en 1:a betyder detta att det i princip är en bild med binära data (helt röd, helt grön och helt blå för respektive del i pixeln eller “ingenting” för den aktuella färgen). Om det står talet 15 innebär det att det finns 16 olika nivåer (0-15) som motsvarar färgnyanserna från 0-255.

Detta leder till att man måste räkna om de färgvärden som finns för R, G och B i bildfilen till motsvarande tal på skalan 0-255. Nedan följer några exempel på hur detta skall vara (du behöver fundera ut vilken formel du skall använda för att få till detta, det är en del i den problemlösning som programmerare behöver göra). Vilka värden skall man få i skalan 0-255 utifrån de färgvärden som står i filen är alltså beroende på hur många nivåer det är.

Exempel 1: Det står **1** som antal nivåer. Vad motsvarar färgvärdena i skalan 0-255?

Färgvärde	Skalan 0-255
0	0
1	255

Exempel 2: Det står **3** som antal nivåer. Vad motsvarar färgvärdena i skalan 0-255?

Färgvärde	Skalan 0-255
0	0
1	85
2	170
3	255

Exempel 3: Det står **7** som antal nivåer. Vad motsvarar färgvärdena i skalan 0-255?

Färgvärde	Skalan 0-255
0	0
1	36
2	73
3	109
4	146
5	182
6	219
7	255

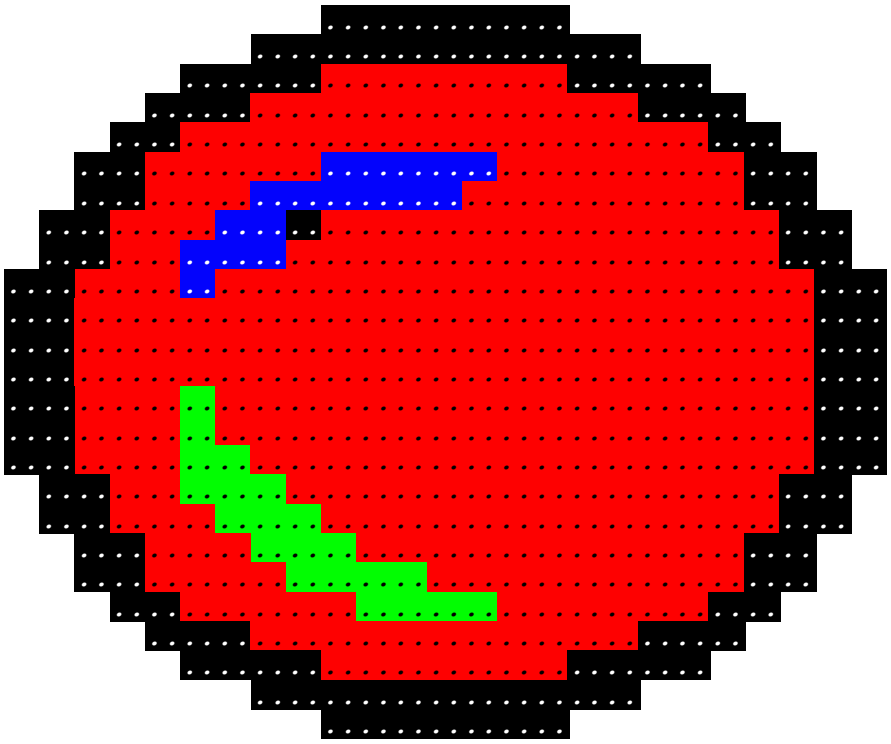
Exempel 4: Det står **12** som antal nivåer. Vad motsvarar färgvärdena i skalan 0-255?

Färgvärde	Skalan 0-255
0	0
1	21
2	43
3	64
4	85
5	106
6	128
7	149
8	170
9	191
10	213
11	234
12	255

Tips: Skriv en funktion som utför denna beräkning så att du senare kan anropa denna och endast behöver rätta till saker på ett ställe om det råkar vara så att du “räknade fel” i första läget.

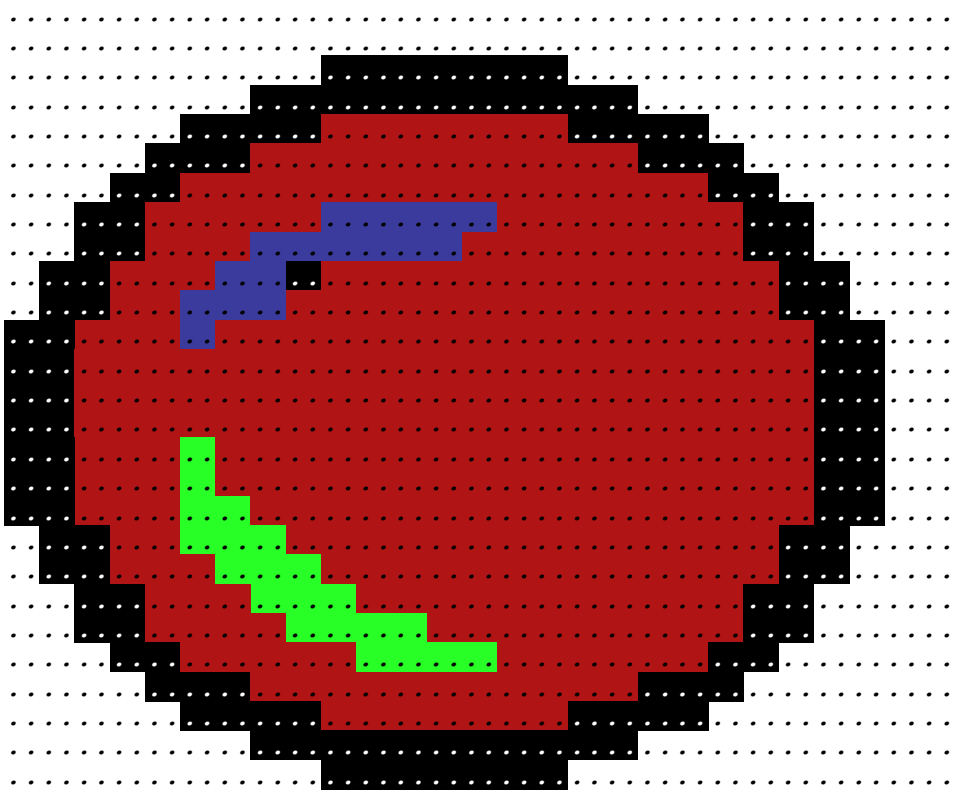
Körexempel 13 (PPMa, 15 nivå):

```
Terminalens prompt % ./image_program object_few_colours.ppma 2 1 2
```



Körexempel 14 (PPM, 13 nivå):

```
Terminalens prompt % ./image_program object_max_level_13.ppm 2 1 2
```



Del 5 - Utökning av funktionalitet

Nya bilder ska kunna skapas utifrån delar av andra bilder. Skapa ett underprogram `Slice` som tar emot en bild och koordinater för två motstående hörn (i en rektangel) i bilden som parametrar. Underprogrammet ska returnera den delbild koordinaterna spänner upp i den ursprungliga bilden. Den nya bildens x- och y-dimensioner ska börja på 1 och sluta på delbildens bredd och höjd. Om någon av koordinaterna ligger utanför originalbildens dimensioner ska undantaget `Image_Constraint_Error` kastas av underprogrammet.

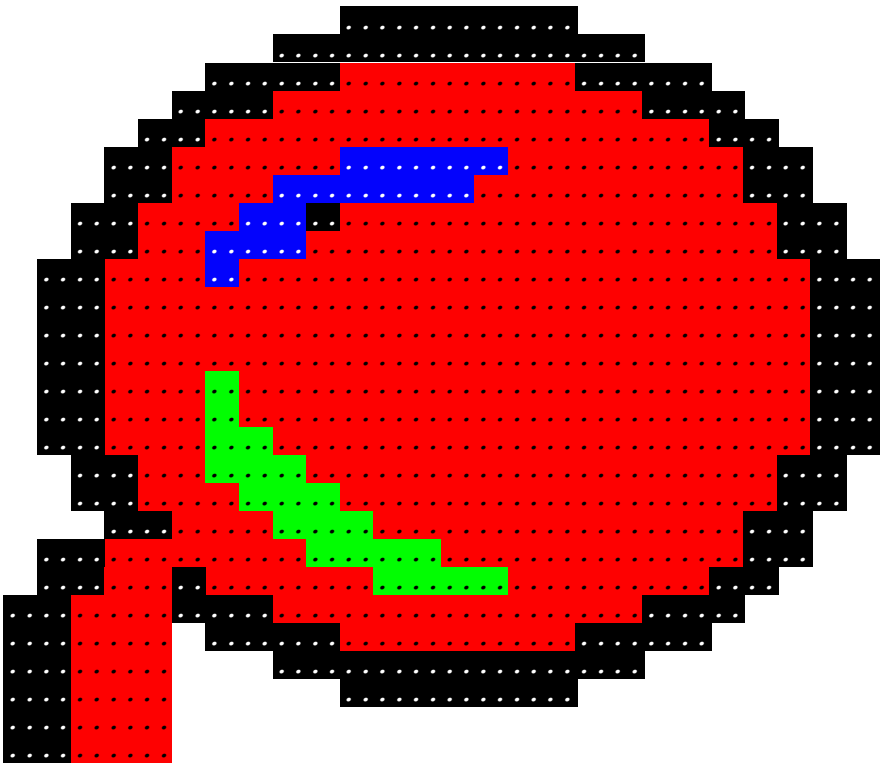
För att köra det första testfallet för denna del anger användaren filen "`slice_test_1.ppma`" och välja bildutskrift via kommandoraden (ett jämnt heltal). Programmet ska för detta testafall skriva ut originalbilden på koordinat (1, 1) samt använda `Slice` för att plocka fram en delbild. Delbilden spänns upp av rektangeln mellan koordinaterna (3, 10) och (8, 17) i originalbilden och ska skrivas ut på den via kommandoraden angivna koordinaten.

För att köra det andra testfallet för denna del anger användaren filen "`slice_test_2.ppma`" och välja bildutskrift via kommandoraden (ett jämnt heltal). Programmet ska för detta testafall skriva ut originalbilden på koordinat (1, 1) samt använda `Slice` för att plocka fram en delbild. Delbilden spänns upp av rektangeln mellan koordinaterna (3, 10) och (8, 17) i originalbilden men originalbilden är för liten så undantaget ska kastas här.

OBS! `Slice` ska självklart fungera generellt vad gäller bilder och koordinater.

Körexempel 15:

```
Terminalens prompt % ./image_program slice_test_1.ppma 2 2 22
```



Körexempel 16:

```
Terminalens prompt % ./image_program slice_test_2.ppma 2 15 10
Error! Slice outside image.
```