1. Introduction.

RASTER

Raster Graphics Library

John Walker

This file is purely a placeholder. It will eventually provide the low-level services for raster image generation, used by `plotting` for charts, and by user analysis programs for other graphical representations of data.

⟨ raster_test.c 1 ⟩ ≡
```
#define REVDATE "29th␣June␣2001"
```
See also section 32.

2. Program global context.

```
#include "config.h"     /* System-dependent configuration */
  ⟨ Preprocessor definitions ⟩
  ⟨ Application include files 4 ⟩
  ⟨ Class implementations 5 ⟩
```

3. We export the class definitions for this package in the external file `raster.h` that programs which use this library may include.

```
⟨ raster.h 3 ⟩ ≡
#ifndef RASTER_HEADER_DEFINES
#define RASTER_HEADER_DEFINES
#include <math.h>     /* Make sure math.h is available */
#include <iostream>
#include <exception>
#include <stdexcept>
#include <string>
#include <vector>
#include <algorithm>
  using namespace std;
#include "bitmap.h"
#include <stdio.h>     /* Needed to define FILE for gifout.h */
#include "gifout.h"
#include "graphics.h"
#include "pstampr.h"
#include "psrtext.h"
  ⟨ Class definitions 6 ⟩
#endif
```

4. The following include files provide access to external components of the program not defined herein.

```
⟨ Application include files 4 ⟩ ≡
#include "raster.h"     /* Class definitions for this package */
```

This code is used in section 2.

5. The following classes are defined and their implementations provided.

```
⟨ Class implementations 5 ⟩ ≡
  ⟨ Colour systems 17 ⟩
```

This code is used in section 2.

6. Colour system parent class: csColour.
This is the parent class of all colour classes.

```
⟨ Class definitions 6 ⟩ ≡
  class csColour {
  public:
    virtual string colourSystemName(void) = 0;      /* Return colour system name */
    virtual void writeParameters(ostream &of)
    {      /* Write description of colour system to output stream */
      of ≪ "Colour␣system:␣" ≪ colourSystemName( ) ≪ "\n";
    }
  };
```

See also sections 7, 8, 9, 10, 11, and 12.

This code is used in section 3.

7. Spectral colour systems parent class: csSpectralColour.

The most general of all our notions of colour is that of an function that maps wavelengths (given in metres) onto intensities. This form allows specification of any radiation in the electromagnetic spectrum. Intensities are normalised to the range zero to one by dividing the intensity for a given frequency by the integrated intensity over the entire electromagnetic spectrum.

```
⟨ Class definitions 6 ⟩ +≡
  class csSpectralColour : public csColour {
  public:     /* Return normalised intensity for a given wavelength in metres */
    virtual double getIntensity(double waveLength) = 0;
      /* Return true if this is a pure (monochromatic) radiator, false */     /* otherwise. */
    virtual bool isMonochromatic(void) = 0;
  };
```

8. Monochromatic spectral colour systems: csMonochromaticColour.

A monochromatic colour is a abstract notion of a source which radiates all its intensity at a single wavelength. These don't really exist in the real world, since fundamental quantum processes will always spread the spectrum of any real radiator.

⟨ Class definitions 6 ⟩ +≡

```
class csMonochromaticColour : public csSpectralColour {
private:
  double wavelength;
public:
  virtual string colourSystemName(void)
  {
    return "monochromatic␣spectral";
  }
  double getIntensity(double waveLength)
  {
    return (waveLength ≡ wavelength) ? 1.0 : 0.0;
  }
  bool isMonochromatic(void)
  {
    return true;
  }     /* Constructors and destructors */
  csMonochromaticColour(void)
  {
    wavelength = 0.0;
  }
  csMonochromaticColour(double waveLength)
  {
    wavelength = waveLength;
  }     /* Class-specific methods */
  void setWavelength(double waveLength)
  {
    wavelength = waveLength;
  }
  double getWavelength(void)
  {
    return wavelength;
  }
  void writeParameters(ostream &of)
  {
    csSpectralColour::writeParameters(of);
    of ≪ "␣␣␣␣Wavelength␣=␣" ≪ getWavelength( ) ≪ "\n";
  }
};
```

9. Black: csSpectralBlack. Black is the absence of colour. We define it as a spectral colour with zero intensity at any wavelength.

```
⟨ Class definitions 6 ⟩ +≡
  class csSpectralBlack : public csSpectralColour {
  public:
    virtual string colourSystemName(void)
    {
      return "black␣spectral";
    }
    double getIntensity(double waveLength)
    {
      return 0.0;
    }
    bool isMonochromatic(void)
    {
      return false;
    }
  };
```

10. White: csSpectralWhite. A theoretical source of white noise has equal power at all wavelengths. Such a source is impossible in reality since the energy flux would be infinite.

```
⟨ Class definitions 6 ⟩ +≡
  class csSpectralWhite : public csSpectralColour {
  public:
    virtual string colourSystemName(void)
    {
      return "white␣spectral";
    }
    double getIntensity(double waveLength)
    {
      return 1.0;
    }
    bool isMonochromatic(void)
    {
      return false;
    }
  };
```

11. Planckian black body: csBlackBody. Define the colour of a Planckian black body radiator with a given colour temperature.

```
⟨ Class definitions 6 ⟩ +≡
  class csBlackBody : public csSpectralColour {
  private:    /* Change temperature to class with system conversions */
    double temperature;    /* Temperature of radiator (°K) */
  public:
    double getIntensity(double waveLength)
    {
      return 0.0;
    }
    bool isMonochromatic(void)
    {
      return false;
    }    /* Constructors and destructors */
    csBlackBody(void)
    {
      temperature = 0.0;
    }
    csBlackBody(double temp)
    {
      temperature = temp;
    }    /* Class-specific methods */
    void setTemperature(double temp)
    {
      temperature = temp;
    }
    double getTemperature(void)
    {
      return temperature;
    }
    double totalFlux(void)    /* Total energy flux in W/m^2 */
    {    /* Total energy flux in Watts per square metre of a black body with temperature T (degrees
        Kelvin) is given by:
                                        C_s T^4
        where C_s is the Stefan-Boltzman constant:
                                    5.67051 × 10^-8 W/(m^2 K^4)
        which, in fundamental terms is:
                                    (π^2 k^4)/(60(h/(2π))^3 c^2)
          */
      return 5.67051 · 10^-8 * (temperature * temperature * temperature * temperature);
    }
    double flux(double wl)    /* Energy flux at a given wavelength */
    {    /* Energy flux from a black body with temperature T (degrees Kelvin), in Watts per square
        metre at wavelength λ (metres) is given by:
                                    M_e = C_1 λ^-5 (e^(C_2/(λ*T)) − 1)^-1
```

where:

$$C_1 = 3.74183e - 16Wm^2$$

$$C_2 = 1.4388e - 2m^\circ K$$

```
    */
  return 3.74183 · 10^-16/((wl * wl * wl * wl * wl) * (exp(1.4388 · 10^-2/(wl * temperature)) - 1));
}
void writeParameters(ostream &of)
{
  csSpectralColour::writeParameters(of);
  of ≪ "␣␣␣␣Temperature␣=␣" ≪ getTemperature( ) ≪ "␣K\n";
}
};
```

12. Device colour systems parent class: csDeviceColour. This class is the parent of all classes which specify colour in a device-specific way, assuming a particular set of illuminants which are summed to form the colour.

```
⟨ Class definitions 6 ⟩ +≡
  class csDeviceColour : public csColour {
  public:
    ⟨ Device colour fundamental methods 13 ⟩
    ⟨ Device colour derived methods 14 ⟩
    ⟨ Device colour system conversion utilities 15 ⟩
  };
```

13. The following three must-implement methods allow retrieving the colour in any of the three fundamental device colour spaces: RGB (additive), CMYK (subtractive), or Greyscale. Typically, a specific device colour class will store the colour in one of these forms and implement the other retrieval methods by converting the colour representation to that form.

```
⟨ Device colour fundamental methods 13 ⟩ ≡
  virtual void asRGB(double &r, double &g, double &b) = 0;
  virtual void asCMYK(double &c, double &m, double &y, double &k) = 0;
  virtual void asGreyScale(double &g) = 0;
```

This code is used in section 12.

14. The following methods allow retrieval of a device colour in other colour mapping spaces. The **csDeviceColour** class implements methods for these functions which provide default definitions which use the subclass *asRGB* method to obtain RGB components which are the converted into the requested colour space.

```
⟨ Device colour derived methods 14 ⟩ ≡
  virtual void asHSV(double &h, double &s, double &v);
  virtual void asHLS(double &h, double &l, double &s);
  virtual void asYIQ(double &y, double &i, double &q);
  virtual void asYUV(double &y, double &u, double &v);
  virtual void asSMPTE(double &y, double &Pb, double &Pr);
  virtual void asCMY(double &c, double &m, double &y);
```

This code is used in section 12.

15. The following **static** methods of the **csDeviceColour** class provide interconversions of common colour systems using the convention that colour components c are **double** values $0 \le c \le 1$.

```
⟨ Device colour system conversion utilities 15 ⟩ ≡
protected:
  static double hlsval(double n1, double n2, double hue);
public:
  static void hsv_rgb(double h, double s, double v, double *r, double *g, double *b);
    /* HSV → RGB */
  static void rgb_hsv(double r, double g, double b, double *h, double *s, double *v);
    /* RGB → HSV */
  static void rgb_hls(double r, double g, double b, double *h, double *l, double *s);
    /* RGB → HLS */
  static void hls_rgb(double h, double l, double s, double *r, double *g, double *b);
    /* HLS → RGB */
  static void rgb_yiq(double r, double g, double b, double *y, double *i, double *q);
    /* RGB → YIQ */
  static void yiq_rgb(double y, double i, double q, double *r, double *g, double *b);
    /* YIQ → RGB */
  static void rgb_yuv(double r, double g, double b, double *y, double *u, double *v);
    /* RGB → YUV */
  static void yuv_rgb(double y, double u, double v, double *r, double *g, double *b);
    /* YUV → RGB */
  static void rgb_smpte_204M(double r, double g, double b, double *y, double *Pb, double *Pr);
    /* RGB → SMPTE 204M */
  static void smpte_204M_rgb(double y, double Pb, double Pr, double *r, double *g, double *b);
    /* SMPTE 204M → RGB */
  static void rgb_cmy(double r, double g, double b, double *c, double *m, double *y);
    /* RGB → CMY */
  static void cmy_rgb(double c, double m, double y, double *r, double *g, double *b);
    /* CMY → RGB */
  static void cmy_cmyk(double c, double m, double y, double *oc, double *om, double *oy, double
      *ok);     /* CMY → CMYK */
  static void cmyk_cmy(double c, double m, double y, double k, double *oc, double *om, double
      *oy);     /* CMYK → CMY */
```

This code is used in section 12.

16. Colour system conversion utilities. The following **static** methods of the **csDeviceColour** class implement interconversions of common colour systems using the convention that colour components c are **double** values $0 \le c \le 1$.

17. `HSV_RGB`: Convert HSV colour specification to RGB intensities. Hue (h) is specified as a **double** value from 0 to 360°, Saturation (s) and Intensity (v) as **double**s from 0 to 1. The (r, g, b) components are returned as **double**s from 0 to 1.

```
⟨ Colour systems 17 ⟩ ≡
  void csDeviceColour::hsv_rgb(double h, double s, double v, double *r, double *g, double *b)
  {
    int i;
    double f, p, q, t;
    if (s ≡ 0) {
      *r = *g = *b = v;
    }
    else {
      if (h ≡ 360.0) h = 0;
      h /= 60.0;
      i = (int) h;
      f = h − i;
      p = v * (1.0 − s);
      q = v * (1.0 − (s * f));
      t = v * (1.0 − (s * (1.0 − f)));
      assert(i ≥ 0 ∧ i ≤ 5);
      switch (i) {
      case 0: *r = v;
        *g = t;
        *b = p;
        break;
      case 1: *r = q;
        *g = v;
        *b = p;
        break;
      case 2: *r = p;
        *g = v;
        *b = t;
        break;
      case 3: *r = p;
        *g = q;
        *b = v;
        break;
      case 4: *r = t;
        *g = p;
        *b = v;
        break;
      case 5: *r = v;
        *g = p;
        *b = q;
        break;
      }
    }
  }
```

See also sections 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, and 31.

This code is used in section 5.

18. `RGB_HSV`: Map r, g, b intensities in the range from 0 to 1 into Hue (h), Saturation (s), and Value (v): Hue from 0 to 360°, Saturation from 0 to 1, and Value from 0 to 1. Special case: if Saturation is 0 (it's a grey scale tone), Hue is undefined and is returned as -1.

This follows Foley *et al.*, section 13.3.4.

```
⟨ Colour systems 17 ⟩ +≡
  void csDeviceColour::rgb_hsv(double r, double g, double b, double *h, double *s, double *v)
  {
    double imax = max(r, max(g, b)), imin = min(r, min(g, b)), rc, gc, bc;
    *v = imax;
    if (imax ≠ 0) *s = (imax − imin)/imax;
    else *s = 0;
    if (*s ≡ 0) {
      *h = −1;
    }
    else {
      rc = (imax − r)/(imax − imin);
      gc = (imax − g)/(imax − imin);
      bc = (imax − b)/(imax − imin);
      if (r ≡ imax) *h = bc − gc;
      else if (g ≡ imax) *h = 2.0 + rc − bc;
      else *h = 4.0 + gc − rc;
      *h *= 60.0;
      if (*h < 0.0) *h += 360.0;
    }
  }
```

19. `RGB_HLS`: Map r, g, b intensities in the range from 0 to 1 into Hue (h), Lightness (l), and Saturation (s): Hue from 0 to 360°, Lightness from 0 to 1, and Saturation from 0 to 1. Special case: if Saturation is 0 (it's a grey scale tone), Hue is undefined and is returned as -1.

This follows Foley *et al.*, section 13.3.5.

⟨ Colour systems 17 ⟩ +≡

```
void csDeviceColour::rgb_hls(double r, double g, double b, double *h, double *l, double *s)
{
  double imax = max(r, max(g, b)), imin = min(r, min(g, b)), rc, gc, bc;
  *l = (imax + imin)/2;
  if (imax ≡ imin) {
    *s = 0;
    *h = −1;
  }
  else {
    if (*l ≤ 0.5) *s = (imax − imin)/(imax + imin);
    else *s = (imax − imin)/(2.0 − imax − imin);
    rc = (imax − r)/(imax − imin);
    gc = (imax − g)/(imax − imin);
    bc = (imax − b)/(imax − imin);
    if (r ≡ imax) *h = bc − gc;
    else if (g ≡ imax) *h = 2.0 + rc − bc;
    else *h = 4.0 + gc − rc;
    *h *= 60.0;
    if (*h < 0) *h += 360.0;
  }
}
```

20. `HLS_RGB`: Convert HLS colour specification to r, g, b intensities. Hue (h) is specified as a **double** value from 0 to 360°; Lightness (l) and Saturation (s) as **double**s from 0 to 1. The RGB components are returned as **double**s from 0 to 1.

```
⟨ Colour systems 17 ⟩ +≡
  double csDeviceColour::hlsval(double n1, double n2, double hue)
  {
    if (hue > 360.0) hue -= 360.0;
    else if (hue < 0.0) hue += 360.0;
    if (hue < 60.0) {
      return n1 + ((n2 - n1) * hue)/60.0;
    }
    else if (hue < 180.0) {
      return n2;
    }
    else if (hue < 240.0) {
      return n1 + ((n2 - n1) * (240.0 - hue))/60.0;
    }
    else {
      return n1;
    }
  }
  void csDeviceColour::hls_rgb(double h, double l, double s, double *r, double *g, double *b)
  {
    double m1, m2;
    if (l ≤ 0.5) m2 = l * (1.0 + s);
    else m2 = l + s - (l * s);
    m1 = 2 * l - m2;
    if (s ≡ 0) {
      *r = *g = *b = l;
    }
    else {
      *r = hlsval(m1, m2, h + 120.0);
      *g = hlsval(m1, m2, h);
      *b = hlsval(m1, m2, h - 120.0);
    }
  }
```

21. `RGB_YIQ` Convert RGB colour specification, r, g, b ranging from 0 to 1, to Y, I, Q colour specification. YIQ is the encoding used in NTSC television.

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.2989 & 0.5866 & 0.1144 \\ 0.5959 & -0.2741 & -0.3218 \\ 0.2113 & -0.5227 & 0.3113 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

⟨ Colour systems 17 ⟩ +≡

```
void csDeviceColour::rgb_yiq(double r, double g, double b, double *y, double *i, double *q)
{
  double ay = (r * 0.2989 + g * 0.5866 + b * 0.1144), ai = (r * 0.5959 + g * -0.2741 + b * -0.3218),
      aq = (r * 0.2113 + g * -0.5227 + b * 0.3113);
  *y = ay;
  if (ay == 1.0) {      /* Prevent round-off on grey scale */
    ai = aq = 0.0;
  }
  *i = ai;
  *q = aq;
}
```

22. `YIQ_RGB`: Convert YIQ colour specification, Y, I, Q given as **double**s: $0 \le Y \le 1$, $-0.6 \le I \le 0.6$, $-0.52 \le Q \le 0.52$, to R, G, B intensities in the range from 0 to 1. The matrix below is the inverse of the `RGB_YIQ` matrix above. YIQ is the encoding used in NTSC television.

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0000 & 0.9562 & 0.6210 \\ 1.0000 & -0.2717 & -0.6485 \\ 1.0000 & -1.1053 & 1.7020 \end{bmatrix} \cdot \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

⟨ Colour systems 17 ⟩ +≡

```
void csDeviceColour::yiq_rgb(double y, double i, double q, double *r, double *g, double *b)
{
  double ar = (y + i * 0.9562 + q * 0.6210), ag = (y + i * -0.2717 + q * -0.6485),
      ab = (y + i * -1.1053 + q * 1.7020);
  *r = max(0.0, min(1.0, ar));
  *g = max(0.0, min(1.0, ag));
  *b = max(0.0, min(1.0, ab));
}
```

23. `RGB_YUV`: Convert RGB colour specification, R, G, B ranging from 0 to 1, to Y, U, V colour specification. YUV is the encoding used by PAL television.

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.2989 & 0.5866 & 0.1144 \\ -0.1473 & -0.2717 & 0.4364 \\ 0.6149 & -0.5145 & -0.1004 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

⟨ Colour systems 17 ⟩ +≡

```
void csDeviceColour::rgb_yuv(double r, double g, double b, double *y, double *u, double *v)
{
  double ay = (r * 0.2989 + g * 0.5866 + b * 0.1144), au = (r * -0.1473 + g * -0.2891 + b * 0.4364),
      av = (r * 0.6149 + g * -0.5145 + b * -0.1004);
  *y = ay;
  if (ay == 1.0) {      /* Prevent round-off on grey scale */
    au = av = 0.0;
  }
  *u = au;
  *v = av;
}
```

24. `YUV_RGB`: Convert YUV colour specification, Y, U, V given as **double**s, to R, G, B intensities in the range from 0 to 1. The matrix below is the inverse of the *rgb_yuv* matrix above. YUV is the encoding used by PAL television.

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0000 & 0.0000 & 1.1402 \\ 1.0000 & -0.3959 & -0.5810 \\ 1.0000 & 2.0294 & 0.0000 \end{bmatrix} \cdot \begin{bmatrix} Y \\ U \\ V \end{bmatrix}$$

⟨ Colour systems 17 ⟩ +≡

```
void csDeviceColour::yuv_rgb(double y, double u, double v, double *r, double *g, double *b)
{
  double ar = (y + u * 0.0000 + v * 1.1402), ag = (y + u * -0.3959 + v * -0.5810),
      ab = (y + u * 2.0294 + v * 0.0000);
  *r = max(0.0, min(1.0, ar));
  *g = max(0.0, min(1.0, ag));
  *b = max(0.0, min(1.0, ab));
}
```

25. `RGB_SMPTE_204M`: Convert RGB colour specification, R, G, B ranging from 0 to 1, to Y, P_b, P_r colour specification according to the SMPTE 204M (1988) specification for HDTV using the SMPTE set of phosphors.

$$\begin{bmatrix} Y \\ P_b \\ P_r \end{bmatrix} = \begin{bmatrix} 0.2122 & 0.7013 & 0.0865 \\ -0.1162 & -0.3838 & 0.5000 \\ 0.6149 & -0.4451 & -0.0549 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

```
⟨ Colour systems 17 ⟩ +≡
  void csDeviceColour::rgb_smpte_204M(double r, double g, double b, double *y, double
          *Pb, double *Pr)
  {
    double ay = (r * 0.2122 + g * 0.7013 + b * 0.0865), aPb = (r * -0.1162 + g * -0.3838 + b * 0.5000),
        aPr = (r * 0.5000 + g * -0.4451 + b * -0.0549);
    *y = ay;
    if (ay ≡ 1.0) {      /* Prevent round-off on grey scale */
      aPb = aPr = 0.0;
    }
    *Pb = aPb;
    *Pr = aPr;
  }
```

26. `SMPTE_204M_RGB`: Convert a colour specified using the SMPTE reference phosphors as given in the SMPTE 204M (1988) specification for HDTV to R, G, B intensities in the range from 0 to 1. The matrix below is the inverse of the `RGB_SMPTE_204M` matrix above.

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0000 & 0.0000 & 1.5755 \\ 1.0000 & -0.2254 & -0.4768 \\ 1.0000 & 1.8270 & 0.0000 \end{bmatrix} \cdot \begin{bmatrix} Y \\ P_b \\ P_r \end{bmatrix}$$

```
⟨ Colour systems 17 ⟩ +≡
  void csDeviceColour::smpte_204M_rgb(double y, double Pb, double Pr, double *r, double
          *g, double *b)
  {
    double ar = (y + Pb * 0.0000 + Pr * 1.5755), ag = (y + Pb * -0.2254 + Pr * -0.4768),
        ab = (y + Pb * 1.8270 + Pr * 0.0000);
    *r = max(0.0, min(1.0, ar));
    *g = max(0.0, min(1.0, ag));
    *b = max(0.0, min(1.0, ab));
  }
```

27. `RGB_CMY`: Convert RGB colour specification, R, G, B ranging from 0 to 1, to C, M, Y colour specification, also ranging from 0 to 1.

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

⟨ Colour systems 17 ⟩ +≡

```
void csDeviceColour::rgb_cmy(double r, double g, double b, double *c, double *m, double *y)
{
  *c = 1.0 - r;
  *m = 1.0 - g;
  *y = 1.0 - b;
}
```

28. `CMY_RGB`: Convert CMY colour specification, C, M, Y ranging from 0 to 1, to R, G, B colour specification, also ranging from 0 to 1.

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$

⟨ Colour systems 17 ⟩ +≡

```
void csDeviceColour::cmy_rgb(double c, double m, double y, double *r, double *g, double *b)
{
  *r = 1.0 - c;
  *g = 1.0 - m;
  *b = 1.0 - y;
}
```

29. *cmy_cmyk*: Convert CMY colour specification, C, M, Y ranging from 0 to 1, to C, M, Y, K colour specification, also ranging from 0 to 1. K is the black ink component in four colour printing processes. We convert C, M, Y by computing $K = \min(C, M, Y)$, then subtracting that value from each of the C, M, Y components.

⟨ Colour systems 17 ⟩ +≡

```
void csDeviceColour::cmy_cmyk(double c, double m, double y, double *oc, double *om, double
        *oy, double *ok)
{
  double k = min(c, min(m, y));
  *oc = c - k;
  *om = m - k;
  *oy = y - k;
  *ok = k;
}
```

30. *cmyk_cmy*: Convert CMYK colour specification, C, M, Y, K ranging from 0 to 1, to C, M, Y colour specification, also ranging from 0 to 1. K is the black ink component in four colour printing processes. We simply add the K component to each of the C, M, Y components.

```
⟨ Colour systems 17 ⟩ +≡
  void csDeviceColour::cmyk_cmy(double c, double m, double y, double k, double *oc, double
          *om, double *oy)
  {
    *oc = c + k;
    *om = m + k;
    *oy = y + k;
  }
```

31. Colour system conversion methods.

The following methods express a colour in a given colour system in a variety of others. The class must implement the *asRGB* method required for each of these conversions.

```
⟨ Colour systems 17 ⟩ +≡
  void csDeviceColour::asHSV(double &h, double &s, double &v)      /* To HSV */
  {
    double r, g, b;
    asRGB(r, g, b);
    rgb_hsv(r, g, b, &h, &s, &v);
  }
  void csDeviceColour::asHLS(double &h, double &l, double &s)      /* To HLS */
  {
    double r, g, b;
    asRGB(r, g, b);
    rgb_hls(r, g, b, &h, &l, &s);
  }
  void csDeviceColour::asYIQ(double &y, double &i, double &q)      /* To YIQ */
  {
    double r, g, b;
    asRGB(r, g, b);
    rgb_yiq(r, g, b, &y, &i, &q);
  }
  void csDeviceColour::asYUV(double &y, double &u, double &v)      /* To YUV */
  {
    double r, g, b;
    asRGB(r, g, b);
    rgb_yuv(r, g, b, &y, &u, &v);
  }
  void csDeviceColour::asSMPTE(double &y, double &Pb, double &Pr)
    /* To SMPTE 204M */
  {
    double r, g, b;
    asRGB(r, g, b);
    rgb_smpte_204M(r, g, b, &y, &Pb, &Pr);
  }
  void csDeviceColour::asCMY(double &c, double &m, double &y)      /* To CMY */
  {
    double r, g, b;
    asRGB(r, g, b);
    rgb_cmy(r, g, b, &c, &m, &y);
  }
```

32. Test program.

```
⟨ raster_test.c 1 ⟩ +≡
  ⟨ Test program include files 35 ⟩;
  ⟨ Show how to call test program 34 ⟩;
  int main(int argc, char *argv[])
  {
    extern char *optarg;      /* Imported from getopt */
    extern int optind;
    int opt;
    ⟨ Process command-line options 33 ⟩;
    return 0;
  }
```

33. We use *getopt* to process command line options. This permits aggregation of options without arguments and both `-d`*arg* and `-d` *arg* syntax.

```
⟨ Process command-line options 33 ⟩ ≡
  while ((opt = getopt(argc, argv, "nu-:")) ≠ −1) {
    switch (opt) {
    case 'u':     /* -u Print how-to-call information */
      case '?': usage();
      return 0;
    case '-':     /* -- Extended options */
      switch (optarg[0]) {
      case 'c':     /* --copyright */
        cout ≪ "This␣program␣is␣in␣the␣public␣domain.\n";
        return 0;
      case 'h':     /* --help */
        usage();
        return 0;
      case 'v':     /* --version */
        cout ≪ PRODUCT ≪ "␣" ≪ VERSION ≪ "\n";
        cout ≪ "Last␣revised:␣" ≪ REVDATE ≪ "\n";
        cout ≪ "The␣latest␣version␣is␣always␣available\n";
        cout ≪ "at␣http://www.fourmilab.ch/eggtools/eggshell\n";
        return 0;
      }
    }
  }
```

This code is used in section 32.

34. Procedure *usage* prints how-to-call information.

```
⟨ Show how to call test program 34 ⟩ ≡
  static void usage(void)
  {
    cout ≪ PRODUCT ≪ "␣␣--␣␣Analyse␣eggsummary␣files.␣␣Call:\n";
    cout ≪ "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣" ≪ PRODUCT ≪ "␣[options]␣[infile]␣[outfile]\n";
    cout ≪ "\n";
    cout ≪ "Options:\n";
    cout ≪ "␣␣␣␣␣␣␣␣␣␣␣--copyright␣␣␣␣␣␣␣Print␣copyright␣information\n";
    cout ≪ "␣␣␣␣␣␣␣␣␣␣␣-u,␣--help␣␣␣␣␣␣␣␣Print␣this␣message\n";
    cout ≪ "␣␣␣␣␣␣␣␣␣␣␣--version␣␣␣␣␣␣␣␣␣Print␣version␣number\n";
    cout ≪ "\n";
    cout ≪ "by␣John␣Walker\n";
    cout ≪ "http://www.fourmilab.ch/\n";
  }
```

This code is used in section 32.

35. We need the following definitions to compile the test program.

```
⟨ Test program include files 35 ⟩ ≡
#include "config.h"      /* Our configuration */      /* C++ include files */
#include <iostream>
#include <exception>
#include <stdexcept>
#include <string>
  using namespace std;
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#ifdef HAVE_GETOPT
#ifdef HAVE_UNISTD_H
#include <unistd.h>
#endif
#else
#include "getopt.h"      /* No system getopt—use our own */
#endif
#include "raster.h"      /* Class definitions for this package */
```

This code is used in section 32.

36. Index. The following is a cross-reference table for `raster`. Single-character identifiers are not indexed, nor are reserved words. Underlined entries indicate where an identifier was declared.

⟨ Application include files 4 ⟩ Used in section 2.
⟨ Class definitions 6, 7, 8, 9, 10, 11, 12 ⟩ Used in section 3.
⟨ Class implementations 5 ⟩ Used in section 2.
⟨ Colour systems 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31 ⟩ Used in section 5.
⟨ Device colour derived methods 14 ⟩ Used in section 12.
⟨ Device colour fundamental methods 13 ⟩ Used in section 12.
⟨ Device colour system conversion utilities 15 ⟩ Used in section 12.
⟨ Process command-line options 33 ⟩ Used in section 32.
⟨ Show how to call test program 34 ⟩ Used in section 32.
⟨ Test program include files 35 ⟩ Used in section 32.
⟨ `raster.h` 3 ⟩
⟨ `raster_test.c` 1, 32 ⟩

RASTER