Programy do ćwiczeń w języku C++, Zestaw 10

1. Program służący do rysowania zbioru Mandelbrota.

   Zbiór Mandelbrota tworzą punkty płaszczyzny zespolonej: $M = \{c \in C : \forall_{n \in N} \ |z_n| < 2\}$, gdzie $z_{n+1} = z_n^2 + c$ dla $z_0 = 0$.

   $z_n = x + iy, \ c = x_0 + iy_0 \Rightarrow z_{n+1} = x^2 + i2xy - y^2 + x_0 + iy_0 = x^2 - y^2 + x_0 + i\,(2xy + y_0)$

   Mamy więc wzór rekurencyjny dla współrzędnych punktu c $(x_0, \ y_0)$ płaszczyzny:

   $z_{n+1} = x_{n+1} + iy_{n+1} \Rightarrow x_{n+1} = x^2 - y^2 + x_0, \ y_{n+1} = 2xy + y_0$

   Kolorowy obraz zbioru Mandelbrota otrzymamy gdy każdemu punktowi c przyporządkujemy kolor odpowiadający ilości iteracji ciągu, dla których jest spełniony warunek $|z_n| < 2$.

   Szerszy opis można znaleźć na stronie: https://en.wikipedia.org/wiki/Mandelbrot_set

   Część praktyczna

   Tworzymy projekt z szablonu C# **Windows Forms Application (.NET Framework)** lub w C++ według instukcji podanej na stronie: http://fizyka.szmaglinski.eu/Forms2.php

   Z przybornika umieszczamy przy prawej ścianie okna naszego formularza trzy kontrolki **VScrollBar** odpowiadające natężeniom barw *RGB*.

   Dodatkowo np. u góry okna kontrolkę **ProgressBar**, ilustrującą postęp obliczeń.

   Dla naszego okna formularza generujemy obsługę zdarzenia **Paint**, dla którego wywołujemy funkcję *Rysuj* (składową klasy naszego formularza), której definicję napiszemy później:

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
   Rysuj();
}
```

   Umieszczamy jako składniki klasy zmienne dotyczące kolorowanego obszaru:

```
double pX = -2, pY = -2, sz = 4, wy = 4; // parametry rysowanego prostokąta
int szerokosc, wysokosc; // rozmiary okna w pikselach
```

W definicji funkcji *Rysuj* tworzymy obiekty do rysowania w naszym oknie i pobieramy jego rozmiary:

```
Graphics g = this.CreateGraphics();
SolidBrush pedzel = new SolidBrush(Color.Black);
g.FillRectangle(pedzel, g.VisibleClipBounds);

szerokosc = (int)g.VisibleClipBounds.Width;
wysokosc = (int)g.VisibleClipBounds.Height;
if (wysokosc < 1 || szerokosc < 1) return;
Bitmap b = new Bitmap(szerokosc, wysokosc);
```

Rysujemy na naszej bitmapie kolorowy obraz zbioru Mandelbrota, korzystając z kodowania *RGB* (zobacz wzór: https://pl.wikipedia.org/wiki/RGB):

```
progressBar1.Value = 0;
double xx = sz / szerokosc, yy = wy / wysokosc; // jednostki na piksel
for (int i = 0; i < szerokosc; i++)
{
   for (int j = 0; j < wysokosc; j++)
   {
      int k = Kolor(pX + i * xx, pY + j * yy);
      int niebieski = k % 256;
      int zielony = k / 256;
      int czerwony = zielony / 256;
      zielony %= 256;
      b.SetPixel(i, j, Color.FromArgb(czerwony, zielony, niebieski));
   }
   progressBar1.Value = 100 * i / (szerokosc - 1);
}

g.DrawImage(b, 0, 0);
```

Definicja funkcji składowej *Kolor* dla liczb $x0,\ y0 \in (-2;\ 2)$ ze zbioru Mandelbrota. Liczba iteracji odpowiada rysowanemu kolorowi. Ucinamy iterację po 255 krokach. Wygląd zbioru można zmieniać przy pomocy pionowych suwaków. Ich wartości ustawiamy w ich właściwościach (początkowe 7, mimimalne 0, maksymalne 16):

```
double x = x0, y = y0, x2, y2;
int n = 0;
do
{
   x2 = x * x; y2 = y * y;
   y = 2 * x * y + y0; // wzór rekurencyjny na y{n+1}
   x = x2 - y2 + x0; // wzór rekurencyjny na x{n+1}
} while (x2 + y2 < 4 && ++n < 256);
return (0x1000 * vScrollBarRed.Value + 0x10 * vScrollBarGreen.Value
        + vScrollBarBlue.Value) * n;
```

Dla kontrolek typu *VScrollBar* uruchamiamy obsługę zdarzenia chwytania myszką **MouseCaptureChanged**, które uruchamia rysowanie gdy zwalniamy uchwyt suwaka:

```
private void vScrollBarRed_MouseCaptureChanged(object sender, EventArgs e)
{
   if (!vScrollBarRed.Capture) Rysuj();
}
```

Kontrolki kotwiczymy z odpowiednimi ściankami okna. Dla okna programu możemy ewentualnie uruchomić obsługę zdarzenia *Resize*, wpisując kod zmieniający wielkość i położenie kontrolek.

Na koniec obsługa kliknięć i przewijania myszką, związana z dwukrotnym powiększaniem i pomniejszaniem obszaru wokół kursora myszki.

Generujemy obsługę zdarzenia kliknięcia myszką **MouseClick** na naszym oknie głównym:

```
private void Form1_MouseClick(object sender, MouseEventArgs e)
{
   if (e.Button == MouseButtons.Left)
   {
      pX += sz * ((double)e.X / szerokosc - .25);
      pY += wy * ((double)e.Y / wysokosc - .25);
      sz /= 2; wy /= 2;
   }
   else if (e.Button == MouseButtons.Right)
   {
      pX += sz * ((double)e.X / szerokosc - 1);
      pY += wy * ((double)e.Y / wysokosc - 1);
      sz *= 2; wy *= 2;
   }
   Rysuj();
}
```

Analogiczny kod wpisujemy po wygenerowaniu obsługi zdarzenia **MouseWheel**.

Może nie występować opcja automatycznego generowania kodu, wtedy trzeba ręcznie wpisać definicję poniższej funkcji:

```
private void Form1_MouseWheel(object sender, MouseEventArgs e)
{
   if (e.Delta > 0)
   { ... }
   else if (e.Delta < 0)
   { ... }
   Rysuj();
}
```

oraz w odpowiednim miejscu np. w pliku *Form1.Designer.cs* wpisać instrukcję wywołującą powyższą funkcję:

```
this.MouseWheel += new System.Windows.Forms.MouseEventHandler(this.Form1_MouseWheel);
```