

Wrocław, 6 grudnia 2019

Dominika Poręba, 241126  
Krystyna Łosieczka, 241328  
Jakub Superczyński, 241381  
PT TN 13:30

Prowadzący: dr inż. Dominik Żelazny

Laboratorium Urządzeń Peryferyjnych

## **LABORATORIUM 5**

### **Ćwiczenie 8 – drukowanie kodów paskowych**

## 1. Cel ćwiczenia

Celem ćwiczenia jest napisanie programu generującego kody paskowe w standardzie EAN13 reprezentujące kody wpisane przez użytkownika. Program powinien sprawdzać czy kod jest poprawny (tzn. czy zawiera 12 cyfr) a następnie wyliczyć sumę kontrolną. Mając cały kod, program powinien generować z niego grafikę zawierającą szereg białych i czarnych pasków wraz z cyframi, które te paski kodują. Na końcu program powinien pozwalać uruchamiać menu kontekstowe do wydruku wygenerowanego kodu na drukarce.

## 2. Przebieg ćwiczenia

Program napisaliśmy w języku C# wykorzystując środowisko Visual Studio. Do obsługi kodów paskowych wykorzystaliśmy bibliotekę *Spire.Barcode*, czyli darmową bibliotekę do generowania i skanowania kodów kreskowych dla technologii .NET. Do wydrukowania kodów wykorzystaliśmy *Windows Forms*.

Na początku ćwiczenia należy zapoznać się ze standardem EAN13 i sposobem kodowania informacji. W tym standardzie kody paskowe kodują 12 cyfr plus jedną cyfrę sumy kontrolnej. Pierwszym etapem kodowania jest obliczenie sumy kontrolnej. Robi się to w następujący sposób: każdej cyfrze po kolei nadaje się na przemian wagę 1 i 3, sumuje się wszystkie cyfry, następnie oblicza modulo 10. Otrzymany wynik odejmuje się od 10 i ponownie dzieli modulo 10.

Gdy mamy już kompletny ciąg wraz z sumą kontrolną, należy go zakodować, tzn. przekształcić na ciąg zer i jedynek. Aby przekodować cyfrę zgodnie ze standardem EAN13, należy posłużyć się tabelką:

cyfra	kodowanie lewostronne		kodowanie prawostronne
	nieparzysty (A)	parzysty (B)	wszystkie znaki
0	0001101	0100111	1110010
1	0011001	0110011	1100110
2	0010011	0011011	1101100
3	0111101	0100001	1000010
4	0100011	0011101	1011100
5	0110001	0111001	1001110
6	0101111	0000101	1010000
7	0111011	0010001	1000100
8	0110111	0001001	1001000
9	0001011	0010111	1110100

Jak widać, ostatnie 6 cyfr kodu mają jednoznaczną reprezentację. Bardziej skomplikowane jest kodowanie lewej części. Skąd wiadomo czy należy cyfrę zakodować w sposób A czy B? Zależy to od pierwszej cyfry kodu (pierwszej cyfry nie kodujemy) i należy sprawdzić w tabeli parzystości:

Pierwsza cyfra systemu numerowego	Sposób kodowania - kombinacja parzystości					
	Druga cyfra systemu numerowego	znaki lewej połowy (kod wytwórcy)				
		1	2	3	4	5
0 (UPC-A)	N	N	N	N	N	N
1	N	N	P	N	P	P
2	N	N	P	P	N	P
3	N	N	P	P	P	N
4	N	P	N	N	P	P
5	N	P	P	N	N	P
6	N	P	P	P	N	N
7	N	P	N	P	N	P
8	N	P	N	P	P	N
9	N	P	P	N	P	N

Jeśli mamy już odpowiednio zakodowany ciąg, dalszy ciąg ćwiczenia jest prosty. Odpowiednie biblioteki dostarczają właściwie gotowych rozwiązań do zamiany ciągu bitowego na czarno-białe paski oraz do wydruku, my musimy jedynie odpowiednie zdefiniować parametry i wywołać odpowiednie funkcje.

### 3. Najważniejsze fragmenty kodu

Generowanie sumy kontrolnej

```
public static int ControlSum(string kod)
    // liczenie sumy kontrolnej - wagi kolejnych cyfr to 131313...,
    // suma robiona modulo przez 10, i wynik odejmowany od 10 - to jest suma kontrolna
    {
        int suma_kontrolna = 0;
        for (int i = 0; i < kod.Length - 1; i += 2)
        {
            suma_kontrolna += Int32.Parse(kod[i] + "");
            suma_kontrolna += 3 * Int32.Parse(kod[i + 1] + "");
        }

        suma_kontrolna %= 10;
        return 10 - (suma_kontrolna % 10);
    }
```

Reprezentacja tabeli kodowania. Tabela jest reprezentowana przez dwuwymiarową tablicę, którą należy odczytywać w następujący sposób: pierwszy indeks to cyfra, którą kodujemy; drugi indeks to sposób kodowania, gdzie 0 oznacza lewostronne A, 1 lewostronne B, natomiast 2 prawostronne. Np. słownik[4][1] zawiera kod cyfry „4” w przypadku kodowania lewostronnego B.

```
private static string[, ] słownik =
    {
        //lewostronne A i B      //prawostronne
        { "0001101", "0100111", "1110010" }, //0
        { "0011001", "0110011", "1100110" }, //1
        { "0010011", "0011011", "1101100" }, //2
        { "0111101", "0100001", "1000010" }, //3
        { "0100011", "0011101", "1011100" }, //4
        { "0110001", "0111001", "1001110" }, //5
    }
```

```

        { "0101111", "0000101", "1010000" }, //6
        { "0111011", "0010001", "1000100" }, //7
        { "0110111", "0001001", "1001000" }, //8
        { "0001011", "0010111", "1110100" } //9
    };

```

Reprezentacja tabeli parzystości. Jest ona reprezentowana poprzez jednowymiarową tabelę. Indeks oznacza wartość pierwszej cyfry kodu, natomiast zawartość tabeli pokazuje w jaki sposób należy traktować po kolei cyfry w kodzie. 0 oznacza że daną cyfrę zapisujemy w sposób nieparzysty, 1 – w sposób parzysty.

```

private static string[] parzystosc = { "000000",
                                       "001011",
                                       "001101",
                                       "001110",
                                       "010011",
                                       "011001",
                                       "011100",
                                       "010101",
                                       "010110",
                                       "011010" };

```

Generowanie kodu paskowego z kodu podanego przez użytkownika. Wykorzystujemy tu zakodowane wcześniej tabelki. Dodatkowo dodajemy paski ochronne.

```

public static string Generate(string kod)
{
    kod += Char.ConvertFromUtf32(ControlSum(kod) + '0'); // dodanie
    znaku konca linii

    int pierwsza = Int32.Parse(kod[0] + ""); //zamiana pierwszej
    cyfry na int
    string kodowanie = parzystosc[pierwsza]; //sprawdzamy pierwsza
    cyfre

    string kod_paskowy = "";

    kod_paskowy += "101"; //lewe paski ochronne
    for (int i = 1; i < 7; i++)
    {
        if (kodowanie[i - 1] == '0') //lewy nieparzysty (A)
        {
            kod_paskowy += slownik[Int32.Parse(kod[i] + ""), 0];
        }
        if (kodowanie[i - 1] == '1') //lewy parzysty (B)
        {
            kod_paskowy += slownik[Int32.Parse(kod[i] + ""), 1];
        }
    }
    kod_paskowy += "01010"; //srodkowe paski ochronne
    for (int i = 7; i < 13; i++)
    {
        kod_paskowy += slownik[Int32.Parse(kod[i] + ""), 2];
    }
    //prawy
    kod_paskowy += "101"; //prawe paski ochronne

    return kod_paskowy;
}

```

Funkcja sprawdzająca poprawność kodu wpisanego przez użytkownika, tzn. czy ma 12 znaków.

```
bool IsCodeValid(string code)
    //funkcja sprawdzająca czy wprowadzony przez użytkownika ciąg ma
    12 znaków
    {
        if (code.Length != 12)
        {
            MessageBox.Show("Wpisz dokładnie 12 cyfr");
            return false;
        }

        return code.All(c => char.IsNumber(c));
    }
```

Funkcja generująca z kodu binarnego kod paskowy. Pochodzi ona z biblioteki Spire.Barcode i daje dodatkowe opcje, takie jak ustawienie kolorów, zdecydowanie czy należy wyświetlać cyfry oprócz pasków czy określenie typu kodu paskowego. My wybieramy standard EAN13 oraz kolory standardowe – czarny i biały. Funkcja jest wywołana po naciśnięciu przycisku.

```
private void btnGenerateToBarcode_Click(object sender, EventArgs e)
{
    var code = tbCodeToGenerate.Text.Trim();
    if (!IsCodeValid(code)) return;

    //generowanie kodu kreskowego z cyfr od użytkownika + wyliczenie
    sumy kontrolnej
    var codeToGenerateWithChecksum = code +
    Char.ConvertFromUtf32(Ean13CodeGenerator.ControlSum(code) + '0');

    var barcodegenerator = new Spire.Barcode.BarCodeGenerator(new
    BarcodeSettings()
    {
        Type = BarCodeType.EAN13,
        BackColor = Color.White,
        ForeColor = Color.Black,

        Data = codeToGenerateWithChecksum,
        ShowText = true,
        ShowTextOnBottom = true,
    });

    barcodeImg = barcodegenerator.GenerateImage();
    barcodeImg = cropImage(barcodeImg, new Rectangle(0, 15,
    barcodeImg.Width, barcodeImg.Height - 15));

    pbBarcode.Image = barcodeImg;
    pbBarcode.Update();
    pbBarcode.Visible = true;
}
```

Funkcje odpowiedzialne za wygenerowanie rysunku i uruchomienia menu drukowania. Funkcje te również odpowiednio przycinają rysunek, mamy możliwość manipulowania rozmiarem wydrukowanego kodu kreskowego.

```
private static Image cropImage(Image img, Rectangle cropArea)
    // tworzenie bitmapy na ktorej bedzie tworzony kod kreskowy
    {
        Bitmap bmpImage = new Bitmap(img);
        return bmpImage.Clone(cropArea, bmpImage.PixelFormat);
    }

private void btnPrintBarcode_Click(object sender, EventArgs e)
    {
        if (barcodeImg == null)
        {
            MessageBox.Show("obrazek jest pusty");
            return;
        }

        ImageConverter _imageConverter = new ImageConverter();
        byte[] xByte = (byte[])_imageConverter.ConvertTo(barcodeImg,
typeof(byte[]));

        PrintDocument pd = new PrintDocument();
        pd.PrintPage += new PrintPageEventHandler(PrintPage);
        PrintDialog pdi = new PrintDialog();
        pdi.Document = pd;
        if (pdi.ShowDialog() == DialogResult.OK)
        {
            pd.Print();
        }
        else
        {
            MessageBox.Show("drukowanie anulowane");
        }
    }

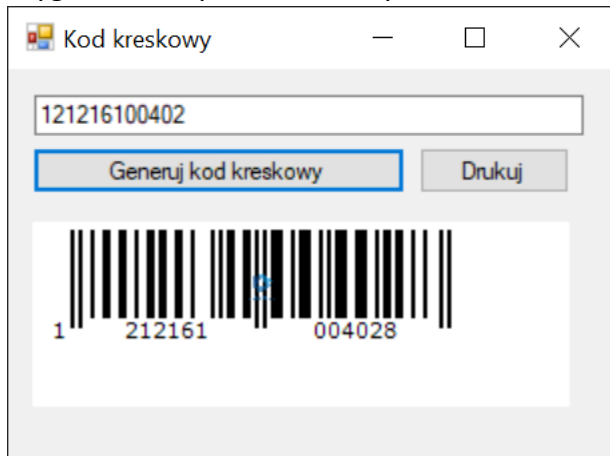
//drukowanie
private void PrintPage(object o, PrintPageEventArgs e)
    {
        try
        {
            //dopasuj rozmiar obrazka do drukowania
            Rectangle m = e.MarginBounds;
            m.Height = 100;
            m.Width = 300;

            //funkcja, drukujaca kod kreskowy
            System.Drawing.Image image = barcodeImg;
            e.Graphics.DrawImage(image, m);
        }
        catch (Exception ex)
        {
            throw ex;
        }
    }
}
```

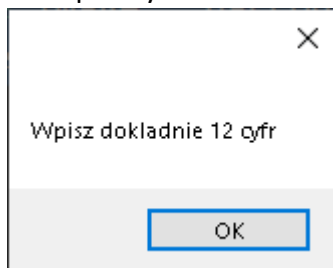
#### 4. Działanie programu

Program spełnia swoje funkcje. Pozwala wpisać własny kod, następnie wyświetla podgląd wygenerowanego kodu paskowego, a następnie pozwala wydrukować kod na drukarce.

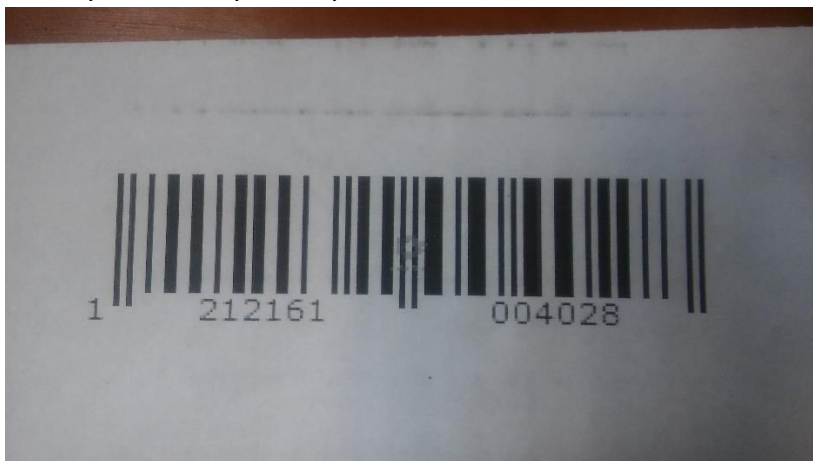
Wygenerowany kod kreskowy:



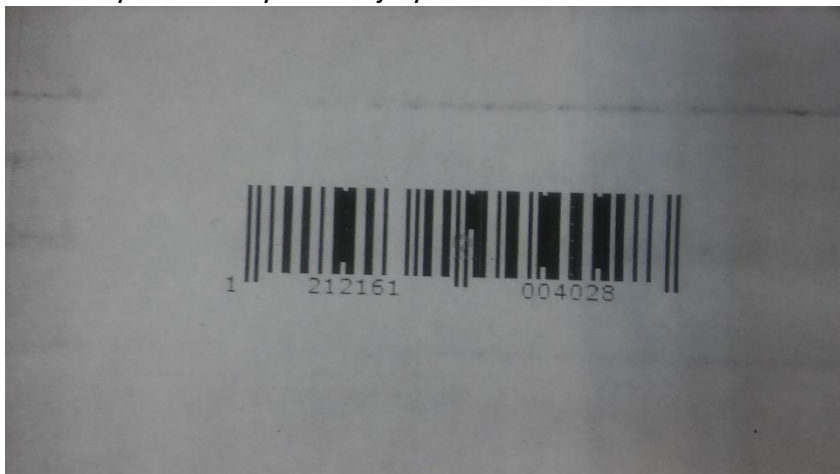
Źle wpisany kod:



Kod wydrukowany w dużym rozmiarze:



Kod wydrukowany w mniejszym rozmiarze:



## **5. Wnioski**

Udało się w całości zrealizować zadanie laboratoryjne. Jedyna trudność w generowaniu kodów kreskowych polega na tym, że nie istnieje biblioteka automatycznie przekodowująca cyfry dziesiętne na kod w odpowiednim standardzie. Należało więc najpierw dobrze zrozumieć kodowanie w standardzie EAN13, a następnie samodzielnie zaimplementować je w programie. Dalsza część jest stosunkowo łatwa dzięki gotowej bibliotece Spire.Barcode. Problem pojawiał się przy drukowaniu kodów paskowych o małym rozmiarze. Czasami czarne paski niejako „wylewały” się na pola, które powinny pozostać białe, co mogłoby by stanowić utrudnienie w odczytywaniu takich kodów. Podejrzewamy jednak że była to wina drukarki: przy dużych rozmiarach taki problem się nie pojawiał, tak samo nie można go było zaobserwować na podglądzie wygenerowanego kodu.