Katedra **I**nżynierii **K**omputerowej Laboratorium z Podstaw Programowania – semestr II	

Cel i zakres zajęć:

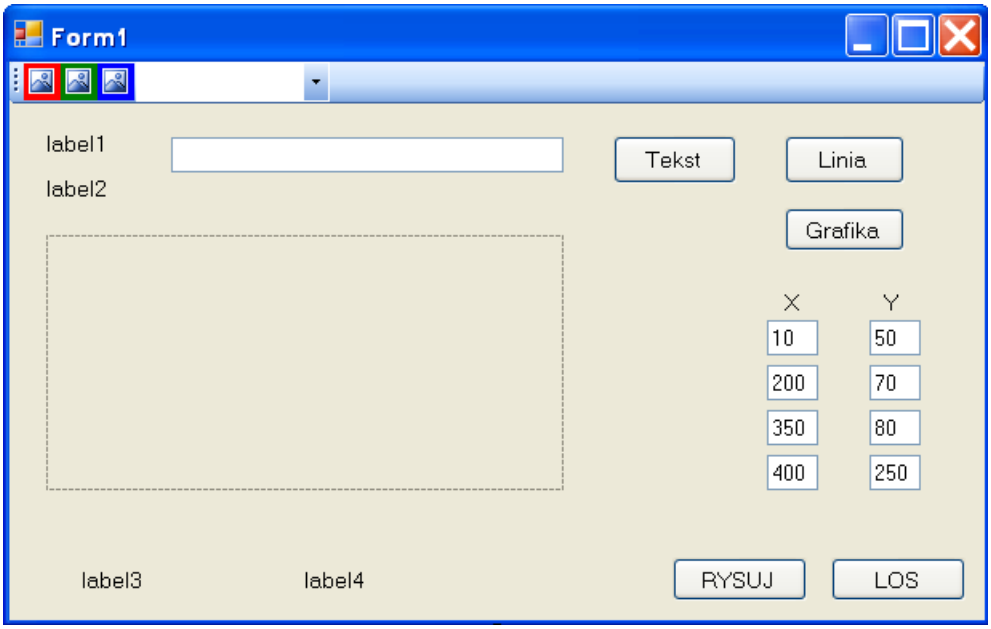
Głównym celem zajęć jest zapoznanie się z właściwościami i zdarzeniami podstawowych komponentów w środowisku *Microsoft Visual C++ 2010 Express Edition*. Przeprowadzenie ćwiczenia pozwala na poznanie projektów typu *Windows Forms Application*, naukę składni podstawowych komponentów w interakcji z użytkownikiem.

Przebieg ćwiczenia:

1. Proszę stworzyć nowy projekt typu: *Windows Forms Application*.

2. Rozmieścić komponenty zgodnie z widokiem zaprezentowanym na rysunku nr 1 (*Button*, *Label*, *TextBox*, oraz *ToolStrip*). W pasku *ToolStrip*, po jego umieszczeniu na formatce, domyślnie znajduje się obiekt typu *SplitButton* składający się z przycisku po lewej stronie i trójkątnej strzałki po stronie prawej. Każdorazowe naciśniecie przycisku powoduje dodanie do paska komponentu *Button*, natomiast kliknięcie strzałki rozwija listę wyboru z pozostałymi komponentami. Dodaj do paska trzy komponenty *Button* oraz jeden *ComboBox*.
   Dodatkowo w projekcie dołączyć należy następujące przestrzenie nazw: *System::Data*, *System::Drawing* oraz *System::IO*.

```
using namespace System::Data;
using namespace System::Drawing;
using namespace System::IO;
```



Rys. 1. Widok formatki projektu.

3. Za pomocą uprzednio zmodyfikowanego paska zadań *ToolStrip* należy przełączać kolory tekstu w etykietach *Label1* oraz *Label2*. Wyboru etykiety, w której zmieniany jest kolor dokonuje się poprzez zaznaczenie właściwej pozycji na liście rozwijanej paska zadań. Przyciski w pasku narzędzi mają zmieniać kolor etykiet na czerwony, zielony i niebieski. W celu zmiany koloru tła przycisków należy zmienić im właściwość *BackColor*, wybierając zakładkę *Custom* oraz stosowny kolor (*Red*, *Green* lub *Blue*). Dla zdarzenia *Click* przycisków należy zaimplementować kod zaprezentowany poniżej. Proszę nie powielać kodu dla każdego z przycisków, a jedynie zaimplementować funkcjonalność jednemu z nich a pozostałym wskazać w wywołaniu metodę zdefiniowanego przycisku.

```
if(toolStripComboBox1->Text=="Etykieta 1")
{
 if(((ToolStripButton^)sender)->Name=="toolStripButton1")
   label1->ForeColor=System::Drawing::Color::Red;
 if(((ToolStripButton^)sender)->Name=="toolStripButton2")
     label1->ForeColor=System::Drawing::Color::Green;
 if(((ToolStripButton^)sender)->Name=="toolStripButton3")
   label1->ForeColor=System::Drawing::Color::Blue;
}
if(toolStripComboBox1->Text=="Etykieta 2")
{
 if(((ToolStripButton^)sender)->Name=="toolStripButton1")
   label2->ForeColor=System::Drawing::Color::Red;
 if(((ToolStripButton^)sender)->Name=="toolStripButton2")
    label2->ForeColor=System::Drawing::Color::Green;
 if(((ToolStripButton^)sender)->Name=="toolStripButton3")
    label2->ForeColor=System::Drawing::Color::Blue;
}
```

4. Wyświetl w oknie aplikacji pełne dane dotyczące aktualnej daty i czasu. Spraw, aby

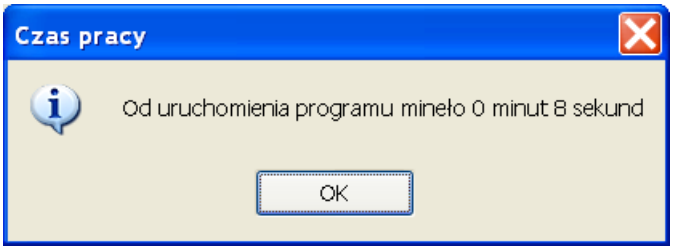
dane były aktualizowane co sekundę. W celu rozwiązania problemu umieść na formatce komponent *Timer*. Ustaw właściwość *Interval* na 1000 (jest to czas próbkowania podany w milisekundach). Przejdź do metody timer1_Tick i dokonaj stosownej implementacji. Uruchomienie *Timera* ma następować przy starcie aplikacji, stąd w zdarzeniu *Load* formatki należy dokonać wpisu: *timer1->Start();*

```
DateTime czas = DateTime::Now;

label3->Text=czas.Hour.ToString("D2")+ ":" + czas.Minute.ToString("D2")+
   ":" + czas.Second.ToString("D2");

label4->Text=czas.Day.ToString("D2")+ "-" +  czas.Month.ToString("D2")+
"-" + czas.Year.ToString("D2")+ " " + czas.DayOfWeek.ToString();
```

5. Oprogramuj funkcjonalność program w taki sposób, aby po zakończeniu pracy (podczas zamykania okna programu) użytkownik poznał czas jaki spędził nad pracą z aplikacją – rys.2.



Rys. 2. Okno informujące o czasie korzystania z aplikacji.

Zadeklaruj w klasie *Form1* obiekt typu *DateTime* do zapisu czasu:

```
private:
        DateTime czas_start;
```

Dla zdarzenia uruchomienia formatki zaimplementuj odczytanie aktualnego czasu systemowego:

```
czas_start=DateTime::Now;
```

Teraz analogicznie oprogramuj zdarzenie *FormClosing* dbając o to, aby użytkownik poznał różnice pomierzonych czasów.

```
TimeSpan^ czas_dzial=gcnew TimeSpan();
czas_dzial=DateTime::Now-czas_start;

MessageBox::Show("Od uruchomienia programu mineło "+
  czas_dzial->Minutes.ToString() + " minut "+
  czas_dzial->Seconds.ToString() + " sekund ",
  "Czas pracy", MessageBoxButtons::OK,
  MessageBoxIcon::Information);
```

6. Większość komponentów wizualnych w VC++ posiada właściwość *Graphics*, dzięki której można rysować, wypisywać tekst oraz umieszczać na nich grafiki w postaci bitmapy. Właściwość *Graphics* porównać można do płótna na którym użytkownik

może tworzyć grafikę. Na początku po utworzeniu komponentu jego właściwość *Graphics* jest pusta. Przed rozpoczęciem rysowania należy stworzyć obiekt typu *Graphics* i „podpiąć" do tej właściwości. Do stworzenia obiektu *Graphics* służy metoda *CreateGraphics()* wywoływana na komponencie, na którym chcemy rysować. Pod przyciskiem *Linia* zaimplementuj rysowanie ukośnej, niebieskiej linii prostej od punktu (10,10) do (100,100) według współrzędnych formatki.

```
Graphics^ linia = this->CreateGraphics();
Pen^ pioro = gcnew Pen(System::Drawing::Color::Blue);
linia->DrawLine(pioro,100,100,300,100);
```

7. Po wciśnięciu przycisku *Tekst* w oknie formatki wyświetl tekst podany w polu tekstowym *TextBox*. Nie używaj żadnego dodatkowego komponentu, a jedynie korzystaj z pędzla typu *SolidBrush*. Tekst powinien być koloru zielonego, pisany czcionka *GenericSansSerif* o rozmiarze 14.

```
Graphics^ napis = this->CreateGraphics();
napis->Clear(System::Drawing::Color::FromName("Control"));
SolidBrush^ pedzel = gcnew
SolidBrush(System::Drawing::Color::DarkGreen);
System::Drawing::Font^ czcionka = gcnew
System::Drawing::Font(System::Drawing::FontFamily::
GenericSansSerif,14, FontStyle::Regular);
napis->DrawString(textBox1->Text,czcionka,pedzel,100,70);
```

8. Umieść na formatce komponent *PictureBox*. Pod przyciskiem *Grafika* Wyświetl na tym komponencie plik *rysunek.bmp,* który uprzednio umieść w katalogu projektu. Utwórz obiekt *Image* zawierający plik, następnie użyj metody *DrawImage()*. Parametry tej funkcji definiują położenie lewego górnego rogu obrazu.

```
Graphics^ obrazek=pictureBox1->CreateGraphics();
Image^ bmp = Image::FromFile("rysunek.bmp");
obrazek->DrawImage(bmp,25,25);
delete obrazek;
```

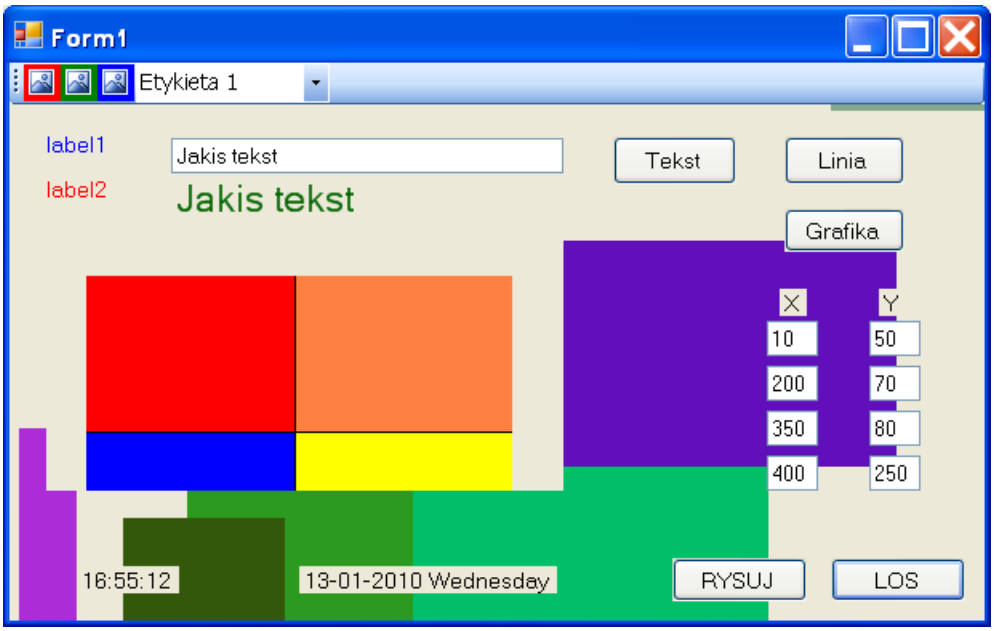
9. Po naciśnięciu przycisku *Rysuj* wyświetl na formularzu wykres ze współrzędnych podanych w polach tekstowych (X,Y). Współrzędne będą pobierane z pól tekstowych i przekazywane do tablicy obiektów *System::Drawing::PointF*, które reprezentują punkty na płaszczyźnie. Wykres narysować należy za pomocą metody *DrawCurve().*

```
Graphics^ wykres = this->CreateGraphics();
wykres->Clear(System::Drawing::Color::FromName("Control"));
array<System::Drawing::PointF>^ punkty = gcnew
array <System::Drawing::PointF>(4);
punkty[0].X=Convert::ToSingle(textBox2->Text);
punkty[0].Y=(this->Height-30)-Convert::ToSingle(textBox3->Text);
punkty[1].X=Convert::ToSingle(textBox5->Text);
punkty[1].Y=(this->Height-30)-Convert::ToSingle(textBox4->Text);
punkty[2].X=Convert::ToSingle(textBox9->Text);
```

```
punkty[2].Y=(this->Height-30)-Convert::ToSingle(textBox8->Text);
punkty[3].X=Convert::ToSingle(textBox7->Text);
punkty[3].Y=(this->Height-30)-Convert::ToSingle(textBox6->Text);
Pen^ pioro = gcnew Pen(System::Drawing::Color::DarkBlue);
wykres->DrawCurve(pioro,punkty);
```

10. Przy każdorazowym wciśnięciu przycisku *Los*, na formatce wyrysuj prostokąt o losowych współrzędnych, losowych wymiarach i losowym kolorze. Aby wygenerować współrzędne użyj generatora liczb losowych *Random*. Należy najpierw utworzyć obiekt, a następnie można pobierać z niego liczby za pomocą metody *Next()* tego obiektu. Po wygenerowaniu liczb, prostokąt rysujemy za pomocą metody *FillRectangle()*. Kolor pędzla definiujemy poprzez trzy liczby losowe z zakresu <0;255>, określające składowe RGB.

```
Graphics^ prostokat = this->CreateGraphics();
Random^ los = gcnew Random();
System::Int32 x= los->Next(this->Width-100);
System::Int32 y= los->Next(this->Height-100);
SolidBrush^ pedzel = gcnew SolidBrush(
System::Drawing::Color::FromArgb(los->Next(255),
los->Next(255), los->Next(255)));
prostokat->FillRectangle(pedzel,x,y,x+30,y+30);
```



Rys. 2. Przykładowy widok ukończonej aplikacji.