Budujemy prosty edytor tekstu pozwalający na wykonywanie operacji na zaznaczonych blokach tekstu, na kolejnych zajęciach zostanie on rozbudowany o system wtyczek. Poniżej opis, jak do zaimplementować podstawowową funkcjonalność edytora.
(LPM – Lewy przycisk myszy, PPM – Prawy przycisk myszy)

1. Uruchamiamy Visual Studio 2017 (lub inne dostępne), wybieramy New Project, spośród języków wybieramy Visual C#, z listy dostępnych projektów Windows Forms Application. Wpisujemy nazwę dla naszego projektu, można wskazać katalog i naciskamy OK.

2. Z poziomu części okna 'Solution Explorer' (po prawej stronie, jeżeli nic nie poprzestawialiśmy) wybieramy formatkę (prawdopodobnie będzie się ona nazywać Form1) na nim PPM i View Designer.

3. Zmieniamy rozmiar projektowanej formatki (ciągnąc za prawy dolny róg), do podobnych jak rozmiar ekranu. Z okna 'Toolbox' (po lewej stronie, o ile oczywiście nic nie przestawialiśmy), z gałęzi 'Common Controls' wybieramy 'RichTextBox' i rysujemy pole 'RichTextBox' na oknie, tak, żeby zostało trochę miejsca na przyciski (np. z prawej strony okna).

4. Uruchamiamy program (F5 lub przycisk Start) i sprawdzamy, czy wszystko działa zgodnie z oczekiwaniami. Od tego miejsca można uruchamiać program po wykonaniu każdego punktu i weryfikowac poprawność działania elementów programu.

5. Następnie dodajemy 2 przyciski ('Button'). Podpisujemy je (okienko 'Properties' w prawym dolnym rogu, pole Text) jako 'Zapisz' i 'Wczytaj' (jak ktoś na tym etapie stwerdzi, że mu za szybko idzie, to prosze dorobić odpowiednie ikonki dla tych przycisków i dołożyc nowe przyciski z funkcjami dodatkowymi typu pogrubienie).

6. Pod przycisk podpisany 'Zapisz' podpinamy odpowiednią funkcję obsługi zdarzenia naciśnięcia przycisku. Najłatwiej to zrobić poprzez dwukrotne kliknięcie na przycisk w widoku 'Designer', powinno nas to przenieś do kodu funkcji: `private void button1_Click(object sender, EventArgs e)`

7. Wewnątrz kodu powyższej funkcji należy umieścić nastepujący kod:

```
    SaveFileDialog saveFile1 = new SaveFileDialog();
    saveFile1.DefaultExt = "*.rtf";
    saveFile1.Filter = "RTF Files|*.rtf";

    if (saveFile1.ShowDialog() == System.Windows.Forms.DialogResult.OK &&
saveFile1.FileName.Length > 0)
        {
            richTextBox1.SaveFile(saveFile1.FileName);
        }
```

8. Podobnie należy przygotować funkcję odczytującą plik ze wskazanego pliku tekstowego i podpiąć go pod przycisk 'Wczytaj'. Kod do umieszczenia:

```
    OpenFileDialog openFile1 = new OpenFileDialog();
    openFile1.DefaultExt = "*.rtf";
    openFile1.Filter = "RTF Files|*.rtf";
    if (openFile1.ShowDialog() == System.Windows.Forms.DialogResult.OK &&
openFile1.FileName.Length > 0)
    {
           richTextBox1.LoadFile(openFile1.FileName);
    }
```

9. Przetestować, przejść do następnego punktu.

   Do istniejącego edytora dorabiamy system wymiennych wtyczek (czyli w umówionym katalogu znajdują sie wtyczki w postaci pliku nazwa.dll, a program główny je wczytuje i na tej podstawie generuje swoje menu i uruchamia funkcje z wtyczek). Poniżej opis, jak do wcześniej zamplementowanego edytora dodać taki system wtyczek.
10. Dodać menu o nazwie "Wtyczki"
11. Na starcie formatki edytora, po wywołaniu funkcji InitializeComponent(), wywołujemy funkcję LoadPlugins (ładowanie wtyczek), którą poniżej pod funkcją InitializeComponent() zaimplementujemy mniej więcej w taki sposób:

```
    void LoadPlugins()
    {
```

```
//kod z kolejnych punktów
}
```

12. Na początek dynamicznie budowane menu, do wcześniej utworzonego menu dodamy w pętli 10 elementów, każdy o nazwie i opisie składających się ze stałego napisu i numeru określającego miejsce w menu. Kod poniższy piszemy w celu uświadomienia sobie faktu, że menu można zbudować także w trakcie wykonywania programu – później ten kod można zakomentować. Dodanie podpozycji do menu wygląda następująco (Pierwszy parametr to opis, drugi wskazanie na obrazek, trzeci na funkcje wykonywaną w momencie naciśnięcia, czwarty to nazwa, którą można się posłużyć do odnajdowania pozycji w menu i identyfikowania, że została ona wciśnięta), kod poniższy umieszczamy we wspomnianej wyżej funkcji LoadPlugins:

```
for(int i = 0 ; i<10 ; i++)
{
    string description = "Opis " + i.ToString();
    string name = "Nazwa" + i.ToString();
    wtyczkiToolStripMenuItem.DropDownItems.Add(new
    ToolStripMenuItem(description, null, null, name));
    }
```

13. Następnie dodajemy pole tekstowe (textBox) i przycisk (button). Po naciśnięciu tego przycisku dodajemy do menu pozycję o takiej nazwie i opisie jaka jest wpisana w polu tekstowym. Jeżeli nazwa jest już dodana, to jej nie dodajemy. Do sprawdzenia, czy nazwy (klucza) NIE ma w menu można się posłużyć funkcją ContainsKey, która sprawdza, czy obiekt o podanym kluczu (w naszym przypadku jest to nazwa) jest już w kolekcji:
Funkcję obsługi zdarzenia pokazujemy dwukrotnie klikając w daną kontrolkę, wewnątrz niej wpisujemy

```
if (!wtyczkiToolStripMenuItem.DropDownItems.ContainsKey("Nazwa"))
{
//kod z punktu 14
}
```

14. Żeby można było obsłużyć naciśnięcie pozycji w menu należy w trzecim parametrze wskazać funkcję która wykona obsługę:

```
wtyczkiToolStripMenuItem.DropDownItems.Add(new ToolStripMenuItem("Opis",
null, MenuHandler, "Nazwa"));
```

15. Funkcja może wyglądać następująco, można ją umieścić na końcu:

```
        private void MenuHandler(object sender, EventArgs e)
        {
             ToolStripMenuItem item = (ToolStripMenuItem)sender;
             MessageBox.Show("Wciśnięto menu.\nKlucz: " + item.Name + " Napis: " + item.Text );
        }
```

16. Kolejnym zadaniem jest dodanie listy plików z określonego katalogu, jako pozycje w menu. Do tego celu należy na początku pliku umieścić dyrektywę using:

```
        using System.IO;
```

17. W funkcji LoadPlugins w pętli dodajemy do menu poszczególne nazwy plików. Do pobrania listy plików z katalogu służy funkcja GetFiles z klasy Directory, zwraca ona tablicę łańcuchów znaków zawierającą poszczególne nazwy plików, można iterować po niej przy pomocy pętli foreach:

```
foreach (string myFilename in
Directory.GetFiles(@"i:\nasza\sciezka\do\ktorej\za\chwile\przegramy\pliki\dll
", "*.dll", SearchOption.AllDirectories))
        {
              ;//miejsce na kod z pkt. 19
        }
```

18. Kolejny element to dodanie listy klas lub metod z dlla do menu. Żeby załadować informacje zawarte w pliku dll (napisanym w .NET), należy użyć klasy Assembly. Na początku pliku umieszczamy:
using System.Reflection;

19. Później w funkcji LoadPlugins zamieniamy kod na poniższy:

```
Assembly plugin = Assembly.LoadFrom("ścieżka_do\ClassLibrary1.dll");
foreach (Type item in plugin.GetTypes()) //lista klas
   foreach (MethodInfo method in item.GetMethods())
   {
        wtyczkiToolStripMenuItem.DropDownItems.Add(new ToolStripMenuItem(item.Name + method.Name,
null, MenuHandler, item.Name + method.Name)); // dodanie nazwy metody w połączeniu z nazwą klasy
        //object result = method.Invoke(null, new object[] { "aaaa" });
```

```
        //result = item.InvokeMember(method.Name, BindingFlags.InvokeMethod, Type.DefaultBinder,
null, new object[] { "aaaa" });
        //object[] methodParams = new object[1];
        //methodParams[0] = "aaaaaaaaaa";
        //result = method.Invoke(null, methodParams); //zakomentowane linijki to inne metody
wywołań
    }
```

20. Kolejny etapem jest utworzenie wtyczek, przyjmujemy, że jeden projekt (jeden dll) to jedna funkcjonalność – 1 klasa z 2 metodami, jedną rejestrującą i jedną wykonującą działanie. Funkcja rejestrująca zawsze nazywa się registerPlugin, druga funkcja ma nazwę dowolną, jej nazwa będzie podana w getPluginName. Klasa zawierająca obie metody może nazywać się dowolnie (przy pisaniu "prawdziwych" pluginów przyjmuje sie też odpowiednie konwencje). Nowy projekt można dodać na 2 sposoby:
    1. wybierając z głównego menu VS.NET opcje File->Add->New Project,
    2. naciskając prawy przycisk na solucji (Solution) w okienku Solution Explorera i wybierając Add->New Project.

    W obydwu przypadkach pojawia sie okno wyboru rodzaju projektu, w nim należy **wybrać Class Library** (C#) W ten sposób proszę dodać **5 projektów**, czyli w sumie ma być ich przynajmniej 6 (5 Class Library + 1 główny Windows Application).

21. Do **każdego** projektu należy dodać klasę o dowolnej nazwie. Klasa może nazywać się dowolnie, jednak przed możliwością pomyłki (o którą łatwo w przypadku duzej liczby projektów) powinno uchronić nazywanie klas podobnie jak projektów (np. jeśli projekty mają liczbę na końcu nazwy – to klasy też powinny mieć liczbę) do których klasy dodajemy. Klasę do projektu dodajemy wciskając w Solution Explorerze prawy przycisk myszy (dalej PPM) na projekcie i wybierając Add->Class.

22. Wszystkie nowo utworzone projekty dodajemy jako referencje do projektu głównego (PPM->Add reference).

23. Do każdej dodanej w pkt. 20 klasy dodajemy metodę getPluginName, która **zwraca nazwę drugiej metody** w tej klasie, którą też dodajemy np:

```
public static string getPluginName()
{
  return "PrecyzyjnaNazwaDrugiejFunkcji";
}

public static string PrecyzyjnaNazwaDrugiejFunkcji(string value)
{
  return "wartość podana do precyzyjnej funkcji to: " + value;
}
```

24. Poszczególne metody wykonujące działania powinny wykonywać różne operacje. Umawiamy się na razie (w jednym z następnych punktów to zmienimy), że wszystkie metody przyjmują jako parametr łańcuch znaków (string) i zwracają ten łańcuch po jakiejś modyfikacji (jak w pkt 20). Przykładowe modyfikacje to: obrócenie łańcucha, zamiana liter małych na wielkie i odwrotnie, operacje dodania znaków z przodu lub tyłu.

25. Kolejnym krokiem jest rejestracja poszczególnych wtyczek na liście, wykorzystujemy do tego kod z punktu 19. - kod wewnątrz pętli dodający pozycję menu wygląda teraz tak:

```
wtyczkiToolStripMenuItem.DropDownItems.Add( new ToolStripMenuItem(method.Name, null,
MenuHandler, myFilename + "|" + item.Name + "|" + method.Name));
```

26. Następnie należy zaimplementować w metodzie MenuHandler wywołanie metody po nazwie (kod z punktu 19 należy zmodyfikować przez dodanie ifa sprawdzającego czy nazwa się zgadza, a funkcję można wywołać dla textu z naszego Richtextboxa. Żeby odszukać metodę do wywołania, można postąpić na 2 sposoby:
    1. wyszukiwać po wszystkich plikach, wszystkich klasach w każdym pliku i na końcu po wszystkich metodach wszystkich klas (pętla podobna jak w LoadPlugins)
    2. w LoadPlugins można połączyć wszystkie nazwy (assembly, klasa, metoda) i w MenuHandler wywołać na tej podstawie odpowiednią metodę (łańcuch znaków dzielimy przy pomocy funkcji Split):

```
    string[] names = item.Name.Split(new char[] {'|'});
    string NazwaAssembly = names[0];
    string NazwaKlasy = names[1];
    string NazwaMetody = names[2];
MessageBox.Show("Wciśnięto menu.\nKlucz: " + item.Name + " Po rozbiciu: " +
NazwaAssembly + "." + NazwaKlasy + "." + NazwaMetody);
//ładowanie dlla o konkretnej nazwie
Assembly plugin = Assembly.Load(NazwaAssembly);
```

```
//ładowanie klasy o konkretnej nazwie
Type item = plugin.GetType(NazwaKlasy)
//ładowanie dlla o konkretnej nazwie
//Assembly plugin = Assembly.Load("NazwaPliku.dll bez dll");
//W niektórych wypadkach:
//Assembly plugin = Assembly.LoadFrom("ścieżka_do\NazwaPliku.dll");
//ładowanie metody o konkretnej nazwie
MethodInfo method = item.GetMethod(NazwaMetody)
object result = method.Invoke(null, new object[] { richTextBox1.Text });
```

27. Po przetestowaniu poprzednich punktów, modyfikujemy wszystkie metody wykonujące (**ich stare wersje zachować np. zakomentowane**), tak żeby przyjmowały RichTextBox jako parametr i nie zwracały nic (swoje działania mają wykonać na Richtexboxie, więc może to być również kolorowanie i pogrubianie). Kod operacji można wzorować na kodzie pod przyciskami robionymi na wcześniejszych zajęciach. Przykładowa funkcja powinna wygladać teraz tak:

```
public static void PrecyzyjnaNazwaDrugiejFunkcji(RichTextBox richBox)
{
  richBox.Text = "wartość podana do precyzyjnej funkcji to: " + richBox.Text;
}
```

28. Wywołanie funkcji powinno wyglądać teraz tak:

```
method.Invoke(null, new object[] { richTextBox1 });
```

29. Następnie modyfikujemy system pluginów tak, żeby rozpoznawał, czy metoda obsługuje string czy RichTextBox. W najprostszej formie można to uzyskać np. przez **doklejenie litery z przodu** nazwy funkcji (s to string, r to RichTextBox). Kod wywołujący później rozpoznaje po tej literze (przechowujemy ją w nazwie menu, ale nie pokazujemy w opisie (przydatna funkcja Substring).

30. Ostatnią modyfikacją jest trzeci wariant funkcji (oznaczany np. literą e), do której przekazujemy wybrany w RichTextBox tekst. Do realizacji samodzielnej na podstawie kodu z poprzednich i wcześniejszych zajęć. ;)