

# Programowanie pod Windows

## Zbiór zadań

Uwaga: zbiór zadań jest w fazie ciągłego rozwoju. Wszelkie prawa autorskie zastrzeżone. Dokument może być rozpowszechniany wyłącznie w celach edukacyjnych, z wyłączeniem korzyści materialnych.

**Wiktor Zychła**  
Instytut Informatyki  
Uniwersytetu Wrocławskiego

Wersja 2014.01.01



# Wprowadzenie

Szanowni Państwo!

Niniejszy zbiór zadań przeznaczony jest dla słuchaczy wykładu **Programowanie pod Windows .NET**, który mam przyjemność prowadzić w Instytucie Informatyki Uniwersytetu Wrocławskiego w kolejnych semestrach letnich od roku akademickiego 2002/2003. Celem wykładu jest zapoznanie słuchaczy z praktyką programowania systemów operacyjnych rodziny Windows.

Zbiór zadań stanowi uzupełnienie podręcznika, pozycji *Windows oczami programisty* [1], dostępnej w wersji akademickiej jako skrypt *Programowanie pod Windows*.

Zadania zebrano w trzy grupy. Pierwsza część to spojrzenie na fundamenty systemu operacyjnego. Druga część to podróż przez historię rozwoju języka C#, współczesnego referencyjnego języka programowania systemów Windows. Trzecia część to przegląd biblioteki standardowej platformy .NET.

Kontynuacją wykładu **Programowanie pod Windows** jest prowadzony w semestrach zimowych wykład **Projektowanie aplikacji ASP.NET**, który jest w całości poświęcony podsystemowi ASP.NET, dedykowanemu rozwijaniu aplikacji internetowych. Z tego też powodu wykład **Programowanie pod Windows** świadomie całkowicie pomija ten obszar technologiczny.

Naturalną kontynuacją każdego wykładu technologicznego, w tym tego, jest również wykład dotyczący zasad projektowania obiektowego. Język i technologia są bowiem tylko sposobem wyrażania aplikacji, a od ich poznania do dobrego projektowania aplikacji jest jeszcze daleka droga. Podobnie jest gdzie indziej - daleka droga wiedzie od poznania gramatyki obcego języka do pisania w nim książek albo od poznania tego jak zamieszać zaprawę murarską do budowania domów. Dlatego serdecznie zapraszam na wykład **Projektowanie obiektowe oprogramowania** gdzie moment w którym kończy się **Programowanie pod Windows** jest początkiem opowieści o tym jak **naprawdę** projektuje i wytwarza się oprogramowanie.

Wiktor Zychla  
wzychla@ii.uni.wroc.pl



# Spis treści

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Win32 API (24)</b>   | <b>9</b>  |
| 1.1      | Elementy interfejsu użytkownika (8)   | 9         |
| 1.1.1    | Wykresy funkcji (1)   | 9         |
| 1.1.2    | Poruszające się kółko (1)   | 9         |
| 1.1.3    | Okno dialogowe (2)  | 9         |
| 1.1.4    | Szablon okna dialogowego (2)  | 11        |
| 1.1.5    | Wybrane składniki <code>Common Controls</code> (2)  | 11        |
| 1.2      | Inne podsystemy Windows (10)  | 11        |
| 1.2.1    | Plik tekstowy na pulpicie, powłoka (1)  | 11        |
| 1.2.2    | Rozmiar okna w rejestrze (2)  | 11        |
| 1.2.3    | Problem golibrody (2)   | 11        |
| 1.2.4    | Internet Explorer jako host dla aplikacji okienkowych (2)   | 12        |
| 1.2.5    | Informacje o systemie (3)   | 12        |
| 1.3      | Component Object Model (6)  | 12        |
| 1.3.1    | Klient COM w C++ (2)  | 12        |
| 1.3.2    | Serwer COM (3)  | 13        |
| 1.3.3    | Klient COM w VBA (1)  | 13        |
| <b>2</b> | <b>Język C# (36)</b>  | <b>15</b> |
| 2.1      | Język C# 1.0 (6)  | 15        |
| 2.1.1    | Prosty algorytm (1)   | 15        |
| 2.1.2    | Indeksery (1)   | 15        |
| 2.1.3    | Refleksja - składowe prywatne (1)   | 16        |
| 2.1.4    | Atrybuty (1)  | 16        |
| 2.1.5    | Dokumentowanie kodu (1)   | 17        |
| 2.1.6    | Dekompilacja kodu (1)   | 17        |
| 2.2      | .NET $\Leftrightarrow$ Win32, Platform Invoke, COM Interoperability (12)  | 17        |
| 2.2.1    | P/Invoke, Win32 $\Rightarrow$ .NET (1)  | 18        |
| 2.2.2    | P/Invoke + DLL (2)  | 18        |
| 2.2.3    | P/Invoke + DLL + wskaźniki na funkcje/delegacje (2)   | 18        |
| 2.2.4    | COM Interop, COM $\Rightarrow$ .NET, early/late binding (3)   | 18        |
| 2.2.5    | COM Interop, .NET $\Rightarrow$ COM (4)   | 19        |
| 2.3      | Rozszerzenia języka C# 2.0 (6)  | 19        |
| 2.3.1    | Kontenery generyczne (1)  | 19        |
| 2.3.2    | Drzewo binarne (2)  | 20        |
| 2.3.3    | Anonimowe delegacje <code>Predicate</code> , <code>Action</code> , <code>Comparison</code> , <code>Converter</code> (1) | 20        |
| 2.3.4    | Algorytmy biblioteczne (1)  | 20        |
| 2.4      | Rozszerzenia języka C# 3.0 (10)   | 20        |
| 2.4.1    | Metoda rozszerzająca klasę <code>System.String</code> (1)   | 20        |

|          |  |           |
|----------|--|-----------|
| 2.4.2    | LINQ to Objects, sortowanie, filtrowanie (1) | 21        |
| 2.4.3    | LINQ to Objects, grupowanie (1)              | 21        |
| 2.4.4    | LINQ to Objects, agregowanie (1)             | 21        |
| 2.4.5    | LINQ to Objects, Join (1)                    | 21        |
| 2.4.6    | LINQ to Objects, analiza logów serwera (2)   | 22        |
| 2.4.7    | Lista obiektów anonimowych (1)               | 22        |
| 2.4.8    | Rekursywne anonimowe delegacje (2)           | 22        |
| 2.5      | Rozszerzenia języka C# 4.0 (2)               | 23        |
| 2.5.1    | Wydajność podsystemu DLR (1)                 | 23        |
| 2.5.2    | Łatwa automatyzacja w C# (1)                 | 23        |
| <b>3</b> | <b>.NET Framework (40)</b>                   | <b>25</b> |
| 3.1      | Biblioteka standardowa .NET (15)             | 25        |
| 3.1.1    | Własna kompletna klasa usługowa (1)          | 25        |
| 3.1.2    | Własna kolekcja danych (1)                   | 26        |
| 3.1.3    | Składanie strumieni (1)                      | 26        |
| 3.1.4    | Golibroda w .NET (2)                         | 26        |
| 3.1.5    | Protokoły sieciowe (1)                       | 26        |
| 3.1.6    | Serializacja i przesyłanie obiektów (2)      | 26        |
| 3.1.7    | Komunikacja międzyprocesowa - MSMQ (2)       | 27        |
| 3.1.8    | Globalizacja (1)                             | 27        |
| 3.1.9    | Usługa systemowa (1)                         | 27        |
| 3.1.10   | Zewnętrzny plik w zasobach aplikacji (1)     | 27        |
| 3.1.11   | Dynamiczne tworzenie kodu (2)                | 27        |
| 3.2      | Podsystem Windows Forms (7)                  | 27        |
| 3.2.1    | Podstawowy interfejs użytkownika (1)         | 27        |
| 3.2.2    | Komponenty dodatkowe (1)                     | 28        |
| 3.2.3    | Podsystem GDI+ (2)                           | 28        |
| 3.2.4    | Własny formant (1)                           | 28        |
| 3.2.5    | Pomoc kontekstowa (2)                        | 28        |
| 3.3      | Podsystemy WPF/Metro (6)                     | 29        |
| 3.3.1    | Podstawowy interfejs użytkownika (2)         | 29        |
| 3.3.2    | Komponenty dodatkowe (2)                     | 29        |
| 3.3.3    | Komponenty dodatkowe (2)                     | 29        |
| 3.4      | eXtensible Markup Language (6)               | 29        |
| 3.4.1    | XML (0)                                      | 29        |
| 3.4.2    | XSD (1)                                      | 29        |
| 3.4.3    | XML + XSD (1)                                | 29        |
| 3.4.4    | XML - serializacja (1)                       | 29        |
| 3.4.5    | XML - DOM (1)                                | 30        |
| 3.4.6    | XML - strumień (1)                           | 30        |
| 3.4.7    | XML - LINQ to XML (1)                        | 30        |
| 3.5      | Relacyjne bazy danych (6)                    | 30        |
| 3.5.1    | DataReader (1)                               | 30        |
| 3.5.2    | Baza danych (0)                              | 30        |
| 3.5.3    | Linq2SQL (1)                                 | 30        |
| 3.5.4    | Entity Framework (1)                         | 31        |
| 3.5.5    | Interfejs użytkownika dla danych (3)         | 31        |

# Rozdział 1

## Win32 API (24)

Zadania z tej grupy mają zapoznać słuchaczy z fundamentami architektury systemów Windows: oknami, uchwytami i przepływem komunikatów. Poznajemy także kilka wybranych podsystemów Win32 i interfejs ich programowania (*Win32API Application Programming Interface*). Współcześnie interfejsu tego prawie nie używa się do wytwarzania nowych aplikacji, co nie zmienia faktu, że nadal stanowi on fundament całego systemu operacyjnego. Poznanie Win32 to więc tak naprawdę zrozumienie jak działają systemy operacyjne z rodziny Windows.

Rozwiązanie zadań w tym rozdziale polega na napisaniu programów w języku C, przy czym w programach wolno korzystać wyłącznie z funkcji bibliotek standardowych C oraz Win32 API. Tam gdzie to możliwe należy wybierać funkcje z Win32API zamiast ich odpowiedników z C (na przykład przy obsłudze systemu plików czy alokacji pamięci).

### 1.1 Elementy interfejsu użytkownika (8)

#### 1.1.1 Wykresy funkcji (1)

Napisać program, który tworzy okno i w jego obszarze roboczym rysuje wykresy funkcji  $f(x) = |x|$  i  $f(x) = x^2$  (z osiami). Oba wykresy powinny być narysowane różnymi kolorami i różnymi stylami pędzli.

Wykresy powinny automatycznie dopasowywać się do nowych rozmiarów okna podczas skalowania okna.

[1p]

#### 1.1.2 Poruszające się kółko (1)

Napisać program, który w obszarze roboczym okna pokaże poruszające się i odbijające się od ramki okna kółko.

Kółko powinno poprawnie reagować na skalowanie rozmiarów okna przez użytkownika.

[1p]

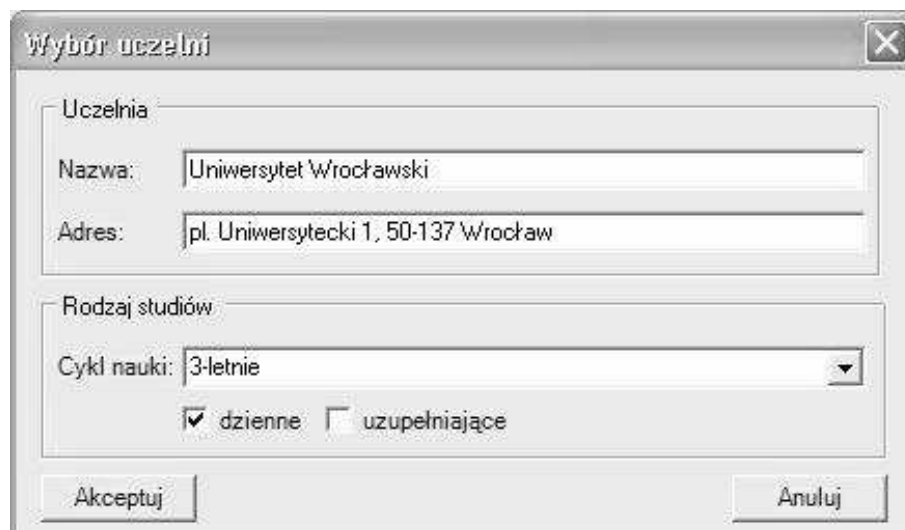
#### 1.1.3 Okno dialogowe (2)

Napisać program, który odtworzy następujący wygląd okna z rysunku 1.1.

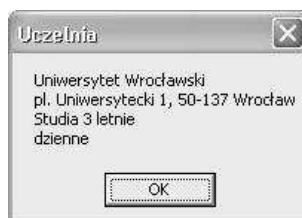
Okno zawiera dwie ramki grupujące (*Group Box*). Pierwsza ramka zawiera dwa pola tekstowe (*Edit Box*), druga zawiera pole wyboru (*Combo Box*) oraz dwa przyciski stanu (*Check Box*).

Lista rozwijalna pola wyboru powinna być wypełniona przykładowymi nazwami.

Po wybraniu przez użytkownika przycisku **Akceptuj**, wybór powinien zostać zaprezentowany w oknie informacyjnym (rysunek 1.2).



Rysunek 1.1: Wygląd okna do zadania [1.1.3]



Rysunek 1.2: Informacja dla użytkownika do zadania [1.1.3]



Naciśnięcie przycisku **Anuluj** powinno zakończyć program.

*Uwaga! Formanty potomne należy inicjować bezpośrednio przez `CreateWindow`. Komunikat w oknie informacyjnym zależy oczywiście od danych wprowadzonych przez użytkownika na formularzu głównym.*

[2p]

#### 1.1.4 Szablon okna dialogowego (2)

Powtórzyć funkcjonalność programu z zadania [1.1.3] używając tym razem edytora zasobów i wbudowanej w niego wizualnego funkcji wizualnej edycji szablonu okna do zbudowania interfejsu użytkownika.

*Uwaga! W przypadku tworzenia okna z szablonu zapisanego w zasobach, zamiast `RegisterClass`, `CreateWindow` i jawnej pętli obsługi komunikatów użyć funkcji `DialogBox`.*

[2p]

#### 1.1.5 Wybrane składniki Common Controls (2)

Napisać program, który zademonstruje działanie trzech wybranych komponentów biblioteki formantów Common Controls (`ListView`, `TreeView`, `Progress Bar`, `Status Bar`, `Tool Bar`, itd.). Demonstracja ma polegać na obsłudze kilku wybranych właściwości komponentów (na przykład wypełnieniu `ListView` kilkoma elementami, zmianie wartości i stylu `Progress Bar`a itp.).

[2p]

## 1.2 Inne podsystemy Windows (10)

### 1.2.1 Plik tekstowy na pulpicie, powłoka (1)

Napisać program, który na pulpicie bieżącego zalogowanego użytkownika umieści plik tekstowy z bieżącą datą systemową. Następnie plik ten skieruje do wydruku.

Do pobrania nazwy foldera użyć funkcji `SHGetFolderPath`. Do skierowania dokumentu do wydruku użyć funkcji sterującej powłoką `ShellExecute`.

[1p]

### 1.2.2 Rozmiar okna w rejestrze (2)

Napisać okienkowy program, który zapamięta w rejestrze systemu rozmiary swojego okna. Rozmiary te powinny być odtwarzane przy każdym uruchomieniu i zapamiętywane przy zamykaniu okna programu.

Zaprojektować format zapisu do rejestru. Zapisywać pod kluczem:

`HKEY_CURRENT_USER\Software\Programowanie pod Windows\...`

[2p]

### 1.2.3 Problem golibrody (2)

Napisać konsolowy program, który rozwiązuje klasyczny problem golibrody lub problem "palaczy tytoniu" za pomocą jeden z metod synchronizacji wątków udostępnianej przez Win32:

- muteksy
- semafony
- zdarzenia

[2p]

### 1.2.4 Internet Explorer jako host dla aplikacji okienkowych (2)

Napisać aplikację HTA (HTML Application), która w głównym oknie programu pozwoli wpisać imię, nazwisko i datę urodzenia, a po naciśnięciu przycisku "OK" zapisze dane do wybranego przez użytkownika pliku tekstowego.

Dlaczego, mimo budowania interfejsu w HTML ta technologia nie może być użyta do budowy aplikacji internetowych?

[2p]

### 1.2.5 Informacje o systemie (3)

Napisać program do diagnozowania komponentów komputera i systemu operacyjnego. Raport powinien obejmować m.in.

- Model procesora oraz częstotliwość taktowania
- Ilość pamięci operacyjnej (wolnej, całej)
- Wersję systemu operacyjnego wraz z wersją uaktualnienia
- Nazwę sieciową komputera i nazwę aktualnie zalogowanego użytkownika
- Ustawienia rozdzielczości i głębi kolorów pulpitu
- Listę drukarek podłączonych do systemu
- Obecność i numery wersji
  - platformy .NET
  - Internet Explorera
  - Microsoft Worda

[3p]

## 1.3 Component Object Model (6)

Rozwiązanie zadań w tym rozdziale polega na napisaniu programów w języku C++, korzystając z wbudowanych w Visual Studio szablonów projektów bibliotek COM.

### 1.3.1 Klient COM w C++ (2)

Napisać w C/C++ aplikację konsoli, która za pośrednictwem usługi COM aplikacji MS Word otworzy nową instancję tej aplikacji, a w niej otworzy nowy dokument, do którego wstawi tekst "Programowanie pod Windows". Następnie dokument zostanie zapisany na dysku pod nazwą "ppw.doc".

[2p]

### 1.3.2 Serwer COM (3)

Przygotować w C++ serwer COM, udostępniający funkcję `int IsPrime( int n )` umożliwiającą sprawdzenie, czy podana liczba jest liczbą pierwszą. Funkcja powinna zwracać zero dla argumentu będącego liczbą złożoną i dowolną niezerową wartość dla argumentu będącego liczbą pierwszą.

*Wskazówka: w Visual Studio należy rozpocząć od projektu C++/ATL Project. Następnie w widoku Class View użyć funkcji Add/ATL COM+ 1.0 Component. Dalsze kroki postępowania zmierzającego do zbudowania serwera COM zostaną zaprezentowane na wykładzie.*

[3p]

### 1.3.3 Klient COM w VBA (1)

Napisać w Visual Basic for Applications (język skryptowy Microsoft Office) funkcję wykorzystującą serwer COM z poprzedniego zadania. Rozwiązanie nie musi posiadać żadnego interfejsu użytkownika do wygodnego wprowadzania argumentu funkcji.

[1p]



## Rozdział 2

# Język C# (36)

Fundamentem współczesnego środowiska wywrtarzania aplikacji musi być nowoczesny język programowania. Język C# czerpie z doświadczeń Smalltalka, C, C++, Pascala i Javy z każdego biorąc to co najlepsze. Do tego jest w mądry i przemyślany sposób rozwijany, dodaje się kolejne elementy, równocześnie dbając o czystość języka.

W rozdziale tym oswajamy się z językiem i w kolejnych zadaniach przechodzimy historię jego rozwoju przez te ponad 10 ostatnich lat.

### 2.1 Język C# 1.0 (6)

#### 2.1.1 Prosty algorytm (1)

Napisać program, który wyznacza zbiór wszystkich liczb naturalnych 1 a 100000, które są podzielne zarówno przez każdą ze swoich cyfr z osobna jak i przez sumę swoich cyfr.

[1p]

#### 2.1.2 Indeksery (1)

Zaimplementować klasę siatki dwuwymiarowej, `Grid`, z dwoma indeksami:

- jednowymiarowym, zwracającym listę elementów zadanego wiersza tablicy, tak aby klient klasy mógł napisać:

```
...
Grid grid = new Grid( 4, 4 );
int[] rowdata = grid[1]; // akcesor "get"
```

- dwuwymiarowym, zwracającym określony element tablicy, tak aby klient klasy mógł napisać:

```
...
Grid grid = new Grid( 4, 4 );

elem[2, 2] = 5;           // akcesor "set"
int elem = grid[1, 4];   // akcesor "get"
```

Oba indeksery powinny przyjmować jako parametry liczby całkowite. Konstruktor klasy powinien przyjmować jako parametry liczbę wierszy i liczbę kolumn siatki.

[1p]

### 2.1.3 Refleksja - składowe prywatne (1)

Napisać program, który zademonstruje możliwość dostępu z zewnątrz do prywatnych składowych klasy.

Kod programu powinien składać się z przykładowej klasy z co najmniej jedną prywatną metodą i właściwością. Kod kliencki powinien uzyskać dostęp do składowych prywatnych za pomocą refleksji.

Należy ponadto porównać szybkość dostępu do składowej publicznej w zwykły sposób i za pomocą refleksji.

*Wskazówka: mierzenie czasu działania bloku kodu najprościej wykonać następująco:*

```
DateTime Start = DateTime.Now;
/* tu blok kodu */
DateTime End = DateTime.Now;

TimeSpan Czas = Start-End;
Console.WriteLine( Czas );
```

*Należy jedynie pamiętać o **powtórzeniu** bloku kodu w pętli tak długo, aż pomiar czasu będzie miał jakikolwiek sens - w przypadku kodu wykonywanego kilka/kilkanaście milisekund powyższa metoda zastosowana do jednokrotnie wykonanego bloku kodu zwróci po prostu 0 jako czas wykonania. Przykład:*

```
int LiczbaProb = 1000;
DateTime Start = DateTime.Now;

for ( int proba=0; proba<LiczbaProb; proba++ )
{
    /* tu blok kodu */
    DateTime End = DateTime.Now;
}

TimeSpan Czas = Start-End;
Console.WriteLine( Czas );
```

Należy również pamiętać o odrzuceniu wyników kilku pierwszych pomiarów - wyniki mogą być zaburzone przez rozruch silnika JIT przygotowującego aplikację do wykonania w systemie operacyjnym.

[1p]

### 2.1.4 Atrybuty (1)

Napisać funkcję, która jako parametr przyjmuje dowolny obiekt i wyszukuje wszystkie jego publiczne, niestacyjne metody zwracające wartość typu **int** i mające pustą listę parametrów.

Następnie spośród tych metod, funkcja wywoła i wypisze na konsoli wynik wywołania wszystkich tych funkcji, które są oznakowane atrybutem **[Oznakowane]**.

Przykładowo, w poniższym fragmencie kodu na konsoli powinna pojawić się tylko wartość z funkcji **Bar**.

```
public class Foo
{
    [Oznakowane]
    public int Bar()
    {
        return 1;
    }

    public int Qux()
    {
        return 2;
    }
}
```

[1p]

### 2.1.5 Dokumentowanie kodu (1)

Zdokumentować (przez umieszczenie odpowiednich komentarzy w kodzie) jeden dowolny program z bieżącej sekcji.

Wygenerować dokumentację w postaci pliku XML podczas kompilacji. Użyć narzędzia SandCastle Help File Builder (<http://shfb.codeplex.com/>) do zbudowania pomocy stylach HTML Help i MSDN-online.

[1p]

### 2.1.6 Dekompilacja kodu (1)

Napisać w C# dowolny program demonstrujący użycie klas (metod, pól, propercji, indeksów, delegacji i zdarzeń) oraz podstawowych konstrukcji składniowych (pętle, instrukcje warunkowe, **switch**) i zdekompilować go do wybranego przez siebie języka (VB.NET lub CIL) za pomocą narzędzia **ILSpy** (<http://ilspy.net/>).

Otrzymany kod skompilować odpowiednim kompilatorem, aby otrzymać plik wynikowy. Plik ten następnie zdekompilować na powrót do języka C#.

Porównać otrzymane w ten sposób pliki z kodem źródłowym. Jak objawiają się i z czego wynikają różnice?

[1p]

## 2.2 .NET ⇔ Win32, Platform Invoke, COM Interoperability (12)

Możliwości platformy .NET byłyby mocno ograniczone, gdyby niemożliwa była współpraca z kodem niezarządzanym. Podobnie jednak jak istnieją dwa różne typy niezarządzanych bibliotek, biblioteki natywne i biblioteki COM, tak istnieją dwa różne mechanizmy do współpracy z

nimi, **Platform Invoke** do konsumpcji bibliotek natywnych oraz **COM Interoperability** do konsumpcji i produkcji usług COM.

Współpraca z już istniejącym kodem niezarządzanym oznacza tak naprawdę możliwość stopniowego wprowadzania platformy .NET do już istniejących projektów, bez konieczności kosztownego jednorazowego przenoszenia ich do .NET w całości. To również szansa na współpracę .NET zarówno z technologiami, które z jakichś powodów nigdy nie zostaną przeniesione do kodu zarządzanego jak i z innymi technologiami przemysłowymi.

### 2.2.1 P/Invoke, Win32 ⇒ .NET (1)

Napisać w C# program, w którym zostanie wywołana funkcja Win32 `GetUserName`, a jej wynik zostanie wyprowadzony w oknie informacyjnym, wywołanym przez funkcję Win32 `MessageBox`.

*Wskazówka: użyć atrybutów `DllImport`, zadeklarować obie funkcje jako `extern`.*

[1p]

### 2.2.2 P/Invoke + DLL (2)

Napisać w języku C bibliotekę natywną, która udostępnia funkcję `int IsPrimeC`, sprawdzającą czy podana 32-bitowa liczba jest pierwsza.

Napisać program w C#, który wywoła tę funkcję z parametrem podanym przez użytkownika z konsoli.

[2p]

### 2.2.3 P/Invoke + DLL + wskaźniki na funkcje/delegacje (2)

Napisać w języku C bibliotekę natywną, która udostępnia funkcję `int ExecuteC` przyjmującą dwa parametry: 32-bitową wartość `n` i wskaźnik na funkcję o sygnaturze `int f(int)`. Funkcja `Execute` jako wynik powinna zwracać wartość `f(n)`.

Napisać program w C#, który oprócz funkcji `Main` będzie zawierał funkcję `int IsPrimeCs` i który użyje funkcji `ExecuteC` (zastosowanej do funkcji `IsPrimeCs`) do sprawdzenia czy podana przez użytkownika z konsoli liczba jest pierwsza.

Czy możliwe było przeniesienie kodu funkcji `IsPrimeC` z poprzedniego zadania jako funkcji `IsPrimeCs`?

[2p]

### 2.2.4 COM Interop, COM ⇒ .NET, early/late binding (3)

To zadanie składa się z 3 części:

1. Napisać bibliotekę COM, która będzie zawierała klasę `PrimeTester`, a w niej metodę `int IsPrime`. Napisać skrypt powłoki, w którym ta metoda zostanie wywołana, a wynik pokazany w oknie informacyjnym.

*Wskazówka: tworzenie bibliotek COM zostało omówione na wykładzie. Zastosować zaproponowaną tam metodę: projekt C++ typu **ATL Library**, do niego dodana klasa **ATL COM+ 1.0 Component**.*

2. Napisać program w C#, w którym zostanie wywołana funkcja `IsPrime` z poprzedniego zadania. Użyć klasy opakowującej (utworzonej automatycznie lub ręcznie).
3. Napisać program w C#, w którym zostanie wywołana funkcja `IsPrime` z poprzedniego zadania. Zamiast klasy opakowującej użyć refleksji.



Jakie są wady i zalety wczesnego i późnego wiązania (łatwość użycia, bezpieczne typowanie)? Czy użycie wczesnego wiązania jest zawsze możliwe?

*Wskazówka: nauczyć się korzystać z `regsvr32.exe` do rejestrowania i wyrejestrowywania komponentów COM. Nauczyć się korzystać z `tlbimp.exe` do tworzenia klas .NET opakowujących klasy COM.*

[3p]

### 2.2.5 COM Interop, .NET ⇒ COM (4)

Napisać w C# bibliotekę, która będzie zawierała klasę `PrimeTesterCS`, a w niej metodę `int IsPrime`. Zarejestrować tę bibliotekę jak bibliotekę COM. Napisać w C++ niezarządzanego klienta COM, zwykłą aplikację konsoli, która skorzysta z tej biblioteki.

Jakie warunki muszą być spełnione, aby klasa .NET mogła być zarejestrowana jako biblioteka COM?

*Wskazówki:*

1. *Nauczyć się korzystać z atrybutu `GuidAttribute`. Dlaczego warto użyć go do oznaczenia klasy `PrimeTesterCS`? Co stałoby się, gdyby nie został on użyty?*
2. *Nauczyć się korzystać z `sn.exe` do tworzenia plików z sygnaturami cyfrowymi. Silnie cyfrowo osygnować bibliotekę, umieszczając odpowiedni atrybut w `AssemblyInfo.cs`. Dlaczego trzeba silnie sygnować biblioteki przeznaczone do COM Interop?*
3. *Nauczyć się korzystać z `gacutil.exe` do zarządzania GAC. Dodać bibliotekę do GAC.*
4. *Nauczyć się korzystać z `regasm.exe` do rejestrowania bibliotek .NET jako komponentów COM. Przy okazji obejrzyć efekt działania `regasm.exe` z parametrem `/regfile`. Zarejestrować bibliotekę dla COM Interop.*
5. *Nauczyć się korzystać z `tlbexp.exe` do eksportowania informacji z bibliotek .NET do współpracy z COM. Dlaczego trzeba eksportować informacje o typach do pliku `*.tlb` (`TypeLib`)?*
6. *Nauczyć się korzystać z dyrektywy `#import` do tworzenia klientów COM w niezarządzanym C++. Dlaczego dyrektywy tej należy użyć wskazując jako parametr ścieżkę do pliku `*.tlb`, a nie do biblioteki `*.dll`?*

*Uwaga! Ze względu na pewną trudność zadania, za częściowe rozwiązania będą wyjątkowo przyznawane punkty pośrednie (między 1 a 4).*

[4p]

## 2.3 Rozszerzenia języka C# 2.0 (6)

### 2.3.1 Kontenery generyczne (1)

Porównać wydajność (dodawanie elementów, przeglądanie, usuwanie) par kontenerów: `ArrayList` - `List<T>` oraz `Hashtable` - `Dictionary<T,K>`.

[1p]

### 2.3.2 Drzewo binarne (2)

Napisać klasę `BinaryTreeNode<T>`, która będzie modelem dla węzła drzewa binarnego. Węzeł powinien przechowywać informację o danej typu `T` oraz swoim lewym i prawym synu.

Klasa powinna zawierać dwa enumeratory, dla przechodzenia drzewa w głąb (i wszerz, za dodatkowe punkty), zaprogramowane w dwu wariantach: z wykorzystaniem słowa kluczowego `yield` i bez.

Który sposób implementacji enumeratora jest łatwiejszy? Dlaczego?

[2p]

### 2.3.3 Anonimowe delegacje Predicate, Action, Comparison, Converter (1)

Zademonstrować w działaniu metody `ConvertAll`, `FindAll`, `ForEach`, `RemoveAll` i `Sort` klasy `List<T>` używając anonimowych delegacji o odpowiednich sygnaturach.

[1p]

### 2.3.4 Algorytmy biblioteczne (1)

W klasie `ListHelper` zaprogramować statyczne metody `ConvertAll`, `FindAll`, `ForEach`, `RemoveAll` i `Sort` o semantyce zgodnej z odpowiednimi funkcjami z klasy `List<T>` i sygnaturach rozszerzonych względem odpowiedników o instancję obiektu `List<T>` na którym mają operować.

```
public class ListHelper
{
    public static List<TOutput> ConvertAll<T, TOutput>(
        List<T> list,
        Converter<T, TOutput> converter );
    public static List<T> FindAll<T>(
        List<T> list,
        Predicate<T> match );
    public static void ForEach<T>( List<T>, Action<T> action );
    public static int RemoveAll<T>(
        List<T> list,
        Predicate<T> match );
    public static void Sort<T>(
        List<T> list,
        Comparision<T> comparison );
}
```

[1p]

## 2.4 Rozszerzenia języka C# 3.0 (10)

### 2.4.1 Metoda rozszerzająca klasę System.String (1)

Zaimplementować metodę `bool IsPalindrome()` rozszerzającą klasę `string`. Implementacja powinna być niewrażliwa na białe znaki i znaki przestankowe występujące wewnątrz napisu ani na wielkość liter. Klient tej metody powinien wywołać ją tak:

```
string s = "Kobyła ma mały bok.";
bool ispalindrome = s.IsPalindrome();
```

[1p]

### 2.4.2 LINQ to Objects, sortowanie, filtrowanie (1)

Dany jest plik tekstowy zawierający zbiór liczb naturalnych w kolejnych liniach. Napisać wyrażenie LINQ, które odczyta kolejne liczby z pliku i wypisze tylko liczby większe niż 100, posortowane malejąco.

```
from liczba in [liczby]
  where ...
  orderby ...
  select ...
```

Przeformułować wyrażenie LINQ na ciąg wywołań metod LINQ to Objects:

```
[liczby].Where( ... ).OrderBy( ... )
```

Czym różnią się parametry operatorów **where/orderby** od parametrów funkcji **Where, OrderBy**?

[1p]

### 2.4.3 LINQ to Objects, grupowanie (1)

Dany jest plik tekstowy zawierający zbiór nazwisk w kolejnych liniach.

Napisać wyrażenie LINQ, które zwróci zbiór **pierwszych** liter nazwisk uporządkowanych w kolejności alfabetycznej. Na przykład dla zbioru (Kowalski, Malinowski, Krasicki, Abacki) wynikiem powinien być zbiór (A, K, M).

*Wskazówka: zgodnie z tytułem zadania użyć operatora `group .. by .. into ...`*

[1p]

### 2.4.4 LINQ to Objects, agregowanie (1)

Napisać wyrażenie LINQ, które dla zadanego foldera wyznaczy sumę długości plików znajdujących się w tym folderze.

Do zbudowania sumy długości plików użyć funkcji **Aggregate**. Listę plików w zadanym folderze wydobyć za pomocą odpowiednich metod z przestrzeni nazw **System.IO**.

[1p]

### 2.4.5 LINQ to Objects, Join (1)

Dane są dwa pliki tekstowe, pierwszy zawierający zbiór danych osobowych postaci (Imię, Nazwisko, PESEL), drugi postaci (PESEL, NumerKonta). Kolejność danych w zbiorach jest przypadkowa.

Napisać wyrażenie LINQ, które połączy oba zbiory danych i zbuduje zbiór danych zawierający rekordy postaci (Imię, Nazwisko, PESEL, NumerKonta). Do połączenia danych należy użyć operatora **join**.

[1p]

### 2.4.6 LINQ to Objects, analiza logów serwera (2)

Rejestr zdarzeń serwera IIS 5.5 ma postać pliku tekstowego, w którym każda linia ma postać:

```
08:55:36 192.168.0.1 GET /TheApplication/WebResource.axd 200
```

gdzie poszczególne wartości oznaczają czas, adres klienta, rodzaj żądania HTTP, nazwę zasobu oraz status odpowiedzi.

Napisać aplikację która za pomocą jednego (lub wielu) wyrażeń LINQ wydobędzie z przykładowego rejestru zdarzeń IIS listę adresów IP trzech klientów, którzy skierowali do serwera aplikacji największą liczbę żądań.

Wynikiem działania programu powinien być przykładowy raport postaci:

```
12.34.56.78 143
23.45.67.89 113
123.245.167.289 89
```

gdzie pierwsza kolumna oznacza adres klienta, a druga liczbę zarejestrowanych żądań.

[2p]

### 2.4.7 Lista obiektów anonimowych (1)

Listy generyczne ukonkretniamy typem elementów:

```
List<int> listInt;
List<string> listString;...
```

Z drugiej strony, w C# 3.0 mamy typy anonimowe, które nie są nigdy jawnie nazwane:

```
var item = new { Field1 = "The value", Field2 = 5 };
Console.WriteLine( item.Field1 );
```

Czy możliwe jest zadeklarowanie i korzystanie z listy generycznej elementów typu anonimowego?

```
var item = new { Field1 = "The value", Field2 = 5; };
List<?> theList = ?
```

W powyższym przykładzie, jak utworzyć listę generyczną, na której znalazłby się element **item** w taki sposób, by móc następnie do niej dodawać nowe obiekty takiego samego typu?

*Obiekty typu anonimowego mają ten sam typ, jeśli mają tę samą liczbę składowych tego samego typu w tej samej kolejności.*

[1p]

### 2.4.8 Rekursywne anonimowe delegacje (2)

Cechą charakterystyczną anonimowych delegacji, bez względu na to czy zdefiniowano je przy użyciu słowa kluczowego **delegate**, czy też raczej jako lambda wyrażenia, jest brak "nazwy", do której można odwołać się w innym miejscu kodu.

Zadanie polega na zaproponowaniu takiego tworzenia anonimowych delegacji, żeby w jednym wyrażeniu możliwa była rekursja. W szczególności, poniższy fragment kodu powinien się kompilować i zwracać wynik zgodny ze specyfikacją.

```
List<int> list = new List<int>() { 1,2,3,4,5 };

foreach ( var item in
    list.Select( i => [...] ) )

    Console.WriteLine( item );
}
```

W powyższym fragmencie kodu, puste miejsce ([...]) należy zastąpić definicją ciała anoni-mowej delegacji określonej rekursywnie:

$$f(i) = \begin{cases} 1 & i \leq 2 \\ f(i-1) + f(i-2) & i > 2 \end{cases}$$

**[2p]**

*Wskazówka* W języku C# można z powodzeniem zaimplementować operator punktu stałego **Y**, wykorzystywany do definicji funkcji rekurencyjnych. Zadanie to można rozwiązać więc definiując taki operator i za jego pomocą implementując funkcję rekurencyjną. Istnieje jednak zaskakujący i o wiele prostszy sposób rozwiązania wymagający jednak trochę nagięcia specyfi-kacji. Oba rozwiązania będą przyjmowane.

## 2.5 Rozszerzenia języka C# 4.0 (2)

### 2.5.1 Wydajność podsystemu DLR (1)

Przeprowadzić testy porównawcze kodu, w którym metoda będzie miała parametr raz typu konkretnego, a drugi raz - dynamicznego. Jak bardzo wolniejsze jest wykonywanie kodu dyna-micznego w tym konkretnym przypadku?

```
int Foo( int x, int y )
{
    // jakieś obliczenia na x i y
}

dynamic Foo( dynamic x, dynamic y )
{
    // te same obliczenia na x i y
}
```

**[1p]**

### 2.5.2 Łatwa automatyzacja w C# (1)

Napisać w C# aplikację konsoli, która za pośrednictwem usługi COM aplikacji MS Word otwo-rzy nową instancję tej aplikacji, a w niej otworzy nowy dokument, do którego wstawi tekst "Programowanie pod Windows". Następnie dokument zostanie zapisany na dysku pod nazwą "ppw.doc".

**[1p]**



## Rozdział 3

# .NET Framework (40)

.NET Framework to współczesna platforma aplikacyjna dla systemów rodziny Windows. Wnosi całe mnóstwo nowoczesnych języków i technologii wytwarzania aplikacji. Również coraz większa część systemu operacyjnego jest obecnie dostarczana w postaci kodu celującego w środowisko .NET.

Rozwiązanie zadań w tym zestawie polega na napisaniu programów w językach platformy .NET. Jeśli nie jest to podane jawnie, sugerowanym językiem jest C#.

### 3.1 Biblioteka standardowa .NET (15)

Biblioteka standardowa platformy .NET bardzo szybko się rozwija. Współcześnie obejmuje właściwie większość możliwych aspektów technologicznych, przez różne podsystemy interfejsu użytkownika, podsystemy graficzne, usługi sieciowe, systemy plików, podsystemy kryptograficzne, usługi kolejkowe i katalogowe, komunikację z systemami relacyjnych i nierelacyjnych baz danych oraz programowanie serwerów aplikacyjnych. Jest to ogromny i fascynujący świat, który ma to do siebie, że obojętnie jak dobrze się go zna, zawsze można znaleźć tu coś nowego i zajmującego.

Z uwagi na ograniczone ramy czasowe, przegląd biblioteki standardowej ma tu charakter wybiórczy.

Ponieważ w przyszłych wersjach systemu operacyjnego Windows interfejs BCL ma szansę stać się natywnym interfejsem programowania Windows, warto szczegółowo zapoznać się z jego możliwościami.

#### 3.1.1 Własna kompletna klasa usługowa (1)

Napisać klasę do obsługi liczb zespolonych. Dodać odpowiednie konstruktory, przeciążyć odpowiednie operatory.

Rozszerzyć tę klasę o własne formatowane. Ścisłej, zaimplementować interfejs `IFormattable` i obsługiwać dwa rodzaje formatowania:

- domyślne (brak formatowania lub `d`) powinno dawać wynik  $a + bi$
- wektorowe (format `w`) powinno dawać wynik  $[a, b]$ .

Przykładowy kawałek kodu:

```
Complex z = new Complex( 4, 3 );
Console.WriteLine( String.Format( "{0}", z ) );
Console.WriteLine( String.Format( "{0:d}", z ) );
Console.WriteLine( String.Format( "{0:w}", z ) );
```

powinien dać wynik

```
4+3i
4+3i
[4, 3]
```

[1p]

### 3.1.2 Własna kolekcja danych (1)

Zaimplementować niegeneryczną kolekcję `Set` działającą jak zbiór, odrzucający duplikaty elementów.

*Wskazówka:* są trzy możliwości - albo dziedziczenie jakiejś kolekcji bibliotecznej, albo zaimplementowanie własnej kolekcji, która wewnętrznie będzie wykorzystywała jakąś kolekcję biblioteczną, wreszcie zaimplementowanie własnej kolekcji nie dziedziczącej z żadnej kolekcji bibliotecznej ani nie wykorzystującej wewnętrznie żadnej kolekcji bibliotecznej. Ta ostatnia możliwość ma niewielki sens - należy uczyć się korzystania z biblioteki standardowej i wykorzystywać jej komponenty we własnym kodzie, a nie wyważać otwarte drzwi implementując już istniejące mechanizmy samemu.

[1p]

### 3.1.3 Składanie strumieni (1)

Napisać program, który zawartość wskazanego pliku tekstowego zapisze do **zaszyfrowanego** algorytmem AES **skompresowanego** strumienia GZip.

Napisać kolejny program, który odszyfruje wskazany skompresowany strumień GZip.

[1p]

### 3.1.4 Golibroda w .NET (2)

Napisać konsolowy program, który rozwiązuje klasyczny problem golibrody lub problem "palaczy tytoniu" za pomocą którejkolwiek z metod synchronizacji wątków udostępnianej przez .NET BCL.

[2p]

### 3.1.5 Protokoły sieciowe (1)

Zademonstrować działanie klas `FtpWebRequest`, `HttpWebRequest`, `HttpListener`, `TcpListener`, `TcpClient`, `SmtpClient`.

[1p]

### 3.1.6 Serializacja i przesyłanie obiektów (2)

Wybrać jeden z omawianych sposobów serializacji (binarne, XML, SOAP) i przygotować dwa moduły: klienta i serwera.

Klient serializuje wskazany obiekt z danymi (jakaś współdzielona prosta klasa) i przesyła go do serwera. Serwer deserializuje obiekt i zapamiętuje go w pliku.

*Wskazówka.* Do implementacji komunikacji klient-serwer można użyć klas `TcpClient` i `TcpListener`

[2p]



### 3.1.7 Komunikacja międzyprocesowa - MSMQ (2)

Korzystając z MSMQ (`System.Messaging`) utworzyć dwukomponentowy system, w którym jeden z komponentów będzie co pewien czas tworzył dużą liczbę komunikatów, a drugi komponent będzie regularnie opróżniał kolejkę komunikatów, wykonując dla każdego z nich jakąś kilkusekundową akcję.

[2p]

### 3.1.8 Globalizacja (1)

Napisać program, który korzystając z informacji z odpowiedniej instancji obiektu `CultureInfo` wypisze pełne i skrótowe nazwy miesięcy i dni tygodnia oraz bieżącą datę w językach: angielskim, niemieckim, francuskim, rosyjskim, arabskim, czeskim i polskim.

[1p]

### 3.1.9 Usługa systemowa (1)

Napisać usługę systemową (*System Service*), która będzie co minutę zapisywać listę uruchomionych aplikacji do pliku tekstowego.

*Uwaga! Po skompilowaniu usługa musi zostać zarejestrowana w systemie za pomocą programu `installutil.exe`. Zarządzanie usługami odbywa się z poziomu panelu **Zarządzanie komputerem**, sekcja **Usługi i aplikacje**.*

[1p]

### 3.1.10 Zewnętrzny plik w zasobach aplikacji (1)

Umieścić dowolny plik w zasobach aplikacji (w projekcie plik powinien mieć właściwość *Embedded Resource*). Następnie napisać klasę, która po podaniu nazwy zasobu umożliwi wydobycie pliku z zasobów zestawu.

Osadzanie plików (tekstowych, binarnych) w zasobach aplikacji przydaje się wtedy kiedy aplikacja jest dystrybuowana do środowiska klienckiego. Zamiast plików wykonywalnych i dodatkowych plików zasobów, klient dostaje pliki wykonywalne w zasobach których zaszyte są pliki z danymi.

[1p]

### 3.1.11 Dynamiczne tworzenie kodu (2)

Napisać program, który w czasie działania powoła do życia instancję kompilatora C# i użyje go do skompilowania fragmentu kodu funkcji, wprowadzonej przez użytkownika do konsoli. Następnie skompilowany fragment zostanie włączony do aktualnie wykonywanego programu i wykonany.

*Wskazówka. Obiekt kompilatora to klasa `Microsoft.CSharp.CSharpCodeProvider`.*

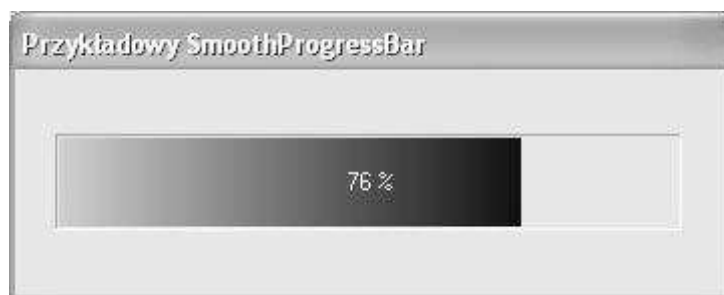
[2p]

## 3.2 Podsystem Windows Forms (7)

### 3.2.1 Podstawowy interfejs użytkownika (1)

Powtórzyć w Windows Forms zadanie według specyfikacji 1.1.3 ze str.9.

[1p]



Rysunek 3.1: Przykładowy SmoothProgressBar

### 3.2.2 Komponenty dodatkowe (1)

Powtórzyć w Windows Forms zadanie według specyfikacji 1.1.5 ze str.11. *Uwaga!* Komponenty pochodzą z podsystemu Windows Forms.

[1p]

### 3.2.3 Podsystem GDI+ (2)

Przedstawiony w skrypcie program rysujący w oknie bieżący czas przerobić na wzór zegarka systemowego Windows, to znaczy tak, żeby bieżąca godzina była przedstawiana na tarczy zegara analogowego a nie cyfrowego.

Wykorzystać funkcje do rysowania z GDI+.

[2p]

### 3.2.4 Własny formant (1)

Zaimplementować własny komponent `SmoothProgressBar`, który będzie imitować zachowanie standardowego komponentu `ProgressBar` ( pasek postępu).

Komponent powinien mieć co najmniej 3 propercje: `Min`, `Max` i `Value`, pozwalające określić odpowiednio minimalną, maksymalną i bieżącą wartość paska postępu. Mając te informacje, `SmoothProgressBar` w zdarzeniu `Paint` powinien rysować gładki (w przeciwieństwie do oryginalnego, który jest złożony z "kafelków") pasek postępu o odpowiedniej długości (według zadanych proporcji).

[1p]

### 3.2.5 Pomoc kontekstowa (2)

W dowolny sposób przygotować plik pomocy kontekstowej w formacie CHM.

Następnie przykładową aplikację rozszerzyć o obsługę pomocy kontekstowej. Należy pokazać, że dla różnych formantów interfejsu użytkownika, przywołanie pomocy kontekstowej przywołuje właściwy temat pliku pomocy.

*Wskazówka: do wiązania formantów z tematami pomocy można użyć bibliotecznego komponentu `HelpProvider` lub jego alternatyw w rodzaju*

<http://netpl.blogspot.com/2007/08/context-help-made-easy-reloaded.html>.

[2p]

### 3.3 Podsystemy WPF/Metro (6)

#### 3.3.1 Podstawowy interfejs użytkownika (2)

Powtórzyć w WPF zadanie według specyfikacji 1.1.3 ze str.9.

[2p]

#### 3.3.2 Komponenty dodatkowe (2)

Powtórzyć w WPF zadanie według specyfikacji 1.1.5 ze str.11. *Uwaga!* Komponenty pochodzą z podsystemu WPF.

[2p]

#### 3.3.3 Komponenty dodatkowe (2)

Powtórzyć zadanie według specyfikacji 1.1.5 ze str.11. *Uwaga!* dla systemu Windows Phone 8.

[2p]

### 3.4 eXtensible Markup Language (6)

Poniższe problemy skomponowano w sposób maksymalnie atomowy, nic nie stoi jednak na przeszkodzie aby kilka kolejnych powiązanych zadań połączyć w jednej większej aplikacji.

#### 3.4.1 XML (0)

Zaprojektować prostą strukturę XML do przechowywania danych o studentach. Każdy student reprezentowany jest **co najmniej** przez podstawowy zbiór atrybutów osobowych, ma dwa adresy (stały i tymczasowy) oraz listę zajęć na które uczęszcza wraz z ocenami.

[0p]

#### 3.4.2 XSD (1)

Schemat struktury z poprzedniego zadania wyrazić w postaci XSD. Zadbać o poprawne opisane reguł walidacji zakresu danych (pewne dane mogą być opcjonalne) i ich zawartości (pewne dane mogą przyjmować wartości o konkretnym formacie).

[1p]

#### 3.4.3 XML + XSD (1)

Napisać program, który używa zaprojektowanego w poprzednim zadaniu schematu XSD do walidacji wskazanych przez użytkownika plików XML i raportuje ewentualne niezgodności.

[1p]

#### 3.4.4 XML - serializacja (1)

Napisać prostego klienta struktury XML z zadania 3.4.3, który pliki XML czyta i zapisuje mechanizmem serializacji do struktur danych zamodelowanych odpowiednimi atrybutami.

[1p]

### 3.4.5 XML - DOM (1)

Napisać prostego klienta struktury XML z zadania 3.4.3, który pliki XML czyta i zapisuje za pomocą modelu DOM (`XmlDocument`).

[1p]

### 3.4.6 XML - strumienie (1)

Napisać prostego klienta struktury XML z zadania 3.4.3, który pliki XML czyta i zapisuje za pomocą mechanizmów strumieniowych (`XmlTextReader`, `XmlTextWriter`).

[1p]

### 3.4.7 XML - LINQ to XML (1)

Napisać wyrażenie LINQ to XML, które z dokumentu XML z poprzednich zadań wybierze dane osobowe studentów o nazwiskach rozpoczynających się na wskazaną literę (wybór litery powinien być możliwy jakkolwiek bez rekompilacji programu).

[1p]

## 3.5 Relacyjne bazy danych (6)

Biblioteka ADO.NET udostępnia spójny fundament obsługi różnych rodzajów źródeł danych. W kolejnych latach do platformy .NET nie tylko migrowały uznane technologie mapowania obiektowo-relacyjnego (nHibernate) ale także powstały rozwiązania natywne (Linq2SQL, Entity Framework), które mają duży wpływ na rozwój technologii poza .NET. Równocześnie wraz z udoskonalaniem narzędzi typu ORM obserwuje się odwrót od rozwiązań typu `DataSet`.

### 3.5.1 DataReader (1)

Przygotować arkusz Excela zawierający dane osobowe (kilka wybranych atrybutów) przykładowej grupy studentów.

Połączyć się do arkusza odpowiednio zainicjowanym połączeniem `OleDb` (`OleDbConnection`), przeczytać zbiór rekordów za pomocą `DataReadera` (`OleDbDataReader`) i pokazać je na liście.

[1p]

### 3.5.2 Baza danych (0)

Przygotować bazę danych Microsoft SQL Server zawierającą dane osobowe i adresy przykładowej grupy studentów.

Model bazy danych zawiera dwie tabele, tabelę **Student** z polami `Imię`, `Nazwisko`, `DataUrodzenia` oraz tabelę **Miejscowosc** z polem `Nazwa`.

Obie tabele połączone są relacją jeden-do-wielu (jak łączy się tabele relacją jeden-do-wielu?).

*Uwaga! Do wykonania tego zadania wystarczy darmowy SQL Server Express Edition albo nawet deweloperski SQL Server Local DB.*

[0p]

### 3.5.3 Linq2SQL (1)

Zbudować model obiektowy dla bazy danych z zadania 3.5.2 za pomocą narzędzia `sqlmetal.exe`.

Pokazać w jaki sposób za pomocą Linq2SQL można dodawać, modyfikować i usuwać dane w bazie danych. W szczególności pokazać jak w jednym bloku kodu dane do **obu** tabel - kod powinien dodać do bazy **nową** miejscowość i **nowego** studenta z tej nowej miejscowości.

[1p]

#### 3.5.4 Entity Framework (1)

Powtórzyć poprzednie zadanie w technologii Entity Framework.

W szczególności - zbudować model obiektowy dla bazy danych z zadania 3.5.2 ręcznie lub za pomocą inżynierii odwrotnej (Reverse Engineer Code First).

Pokazać w jaki sposób za pomocą EF można dodawać, modyfikować i usuwać dane w bazie danych. W szczególności pokazać jak w jednym bloku kodu dane do **obu** tabel - kod powinien dodać do bazy **nową** miejscowość i **nowego** studenta z tej nowej miejscowości.

[1p]

#### 3.5.5 Interfejs użytkownika dla danych (3)

Napisać prostą aplikację okienkową, która udostępnia dane z bazy z poprzedniego zadania. Aplikacja powinna pozwalać na przeglądanie listy studentów, dodawanie, modyfikację i usuwanie.

Do dostępu do danych wybrać Linq2SQL lub Entity Framework.

[3p]



# Bibliografia

- [1] Wiktor Zychla *Windows oczami programisty*, Mikom
- [2] Archer T., Whitechapel A. *Inside C#*, Microsoft Press
- [3] Eckel B. *Thinking in C#*, <http://www.bruceeckel.com>
- [4] Gunnerson E. *A Programmer's Introduction to C#*
- [5] Lidin S. *Inside Microsoft .NET IL Assembler*, Microsoft Press
- [6] Petzold Ch. *Programming Windows*, Microsoft Press
- [7] Solis D. *Illustrated C#*