Programowanie pod Windows
Zbiór zadań

wersja 0.3

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

Wiktor Zychla

Instytut Informatyki
Uniwersytetu Wrocławskiego

Wrocław 2006

Spis treści

Wprowadzenie

Niniejszy zbiór zadań przeznaczony jest dla słuchaczy wykładu **Programowanie pod Windows** prowadzonego w Instytucie Informatyki Uniwersytetu Wrocławskiego i stanowi uzupełnienie podręcznika, pozycji **Windows oczami programisty** ([2]).

Zadania zebrano w dwie grupy, z których pierwsza pozwala zapoznać się z podsystemami Windows i interfejsem Win32, druga zaś to przegląd języków, bibliotek i technologii platformy .NET. Duża liczba i różnorodność oraz fakt, iż suma punktów za wszystkie zadania przekracza maksymalną referencyjną liczbę punktów dla kryteriów punktowych (100 punktów), mają stanowić dla studenta zachętę do wybierania zadań interesujących i pouczających. Zachęcam do zachowania takich proporcji w wyborze zadań z obu grup, jakie wynikają z ich liczności.

Wiktor Zychla

Rozdział 1

Win32 Application Programming Interface

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 Win32API. Tam gdzie to możliwe należy wybierać funkcje z Win32API zamiast ich odpowiedników z C (na przykład przy obsłudze systemu plików czy allokacji pamięci). Do tworzenia i obsługi okien **nie wolno** wykorzystywać żadnych interfejsów pośrednich (WTL, MFC, wxWidgets, GTK).

1.1 Elementy interfejsu użytkownika

1.1.1 Potwierdzenie zamknięcia okna

Napisać program, który podczas próby zamknięcia okna poprosi użytkownika o potwierdzenie (*"Czy jesteś pewien, że chcesz zakończyć program?"*) i w razie odpowiedzi odmownej zrezygnuje z zamykania okna.

[**1p**]

1.1.2 Sinus, cosinus

Napisać program, który tworzy okno i w jego obszarze roboczym rysuje wykresy funkcji $f(x) = sin(x)$ i $f(x) = cos(x)$ (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**]

1.1.3 Poruszające się kółko

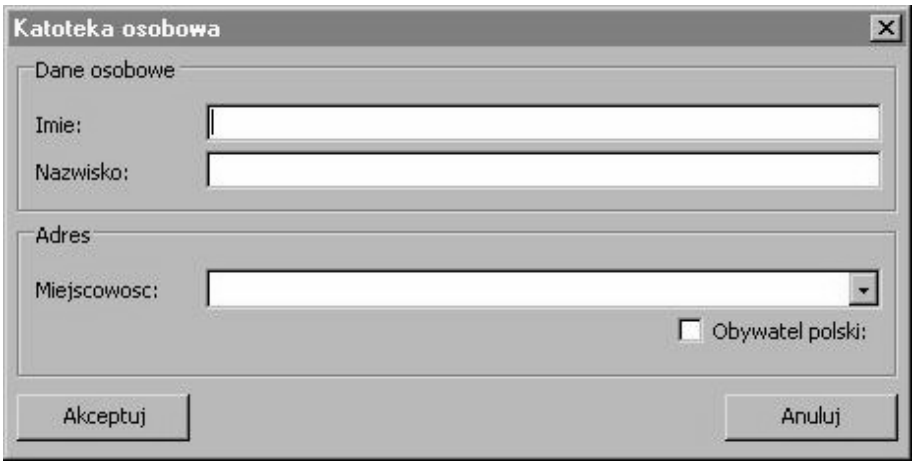
Napisać program, który w obszarze roboczym okna pokaże poruszające się i odbijające się od ramki okna kółko.

Kółko powinno poprawnie reagować na skalowanie rozmiarów okna przez użytkownika.

[**1p**]

1.1.4 Okno dialogowe

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



Rysunek 1.1: Wygląd okna do zadania [1.1.4]

Rysunek 1.2: Informacja dla użytkownika do zadania [1.1.4]

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

Lista rozwijalna pola wyboru powinna być wypełniona nazwami kilku przykładowych miejscowości.

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

Naciśnięcie przycisku **Anuluj** powinno zakończyć program.

Uwaga! Formanty potomne należy inicjować bezpośrednio przez `CreateWindow`.

[**2p**]

1.1.5 Wybrane składniki `Common Controls`

Napisać program, który zademonstruje działanie trzech wybranych komponentów biblioteki Common Controls (ListView, TreeView, Animate Control, 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 Bara itp.).

[**2p**]

1.1.6 Komunikaty formatów zagnieżdżonych

Napisać program, który w oknie umieści panel grupujący (*Group Box*), a wewnątrz niego przycisk. Próba obsługi zdarzenia kliknięcia przycisku w funkcji obsługi komunikatów okna głównego nie uda się (*dlaczego? gdzie trafia odpowiedni WM_COMMAND?*).

Zastanowić się jak poradzić sobie w takiej sytuacji. Czy da się wymyślić rozwiązanie ogólne, tzn. radzące sobie przy dalszym zagnieżdżaniu paneli grupujących?

[**1p**]

1.1.7 Funkcje powłoki

Stare 16-bitowe aplikacje pozbawione są w systemie Windows możliwości drukowania przez nowe typy drukarek (np. drukarki USB). Rozwiązać ten problem.

Ściślej: napisać konsolowy program, Win32 który z linii poleceń przyjmuje nazwę pliku tekstowego i używa powłoki do wydrukowania tego pliku (**ShellExecute**).

Przygotować 16-bitowy program (na przykład za pomocą Borland C++ 3.1), który będzie wywoływał program do drukowania z zadanym parametrem, wskazującym plik, który należy wydrukować.

[**2p**]

1.1.8 Domyślne skojarzenia powłoki

Sprawdzić jak powłoka obsługuje typowe akcje (`open, print`) dla kilku typowych rozszerzeń plików (`txt, exe, doc, rtf, html`).

Zarejestrować w systemie własne rozszerzenie plików, `*.ppwin` i skojarzyć je z przykładową aplikacją tak, aby po wykonaniu przez powłokę akcji `open` lub `print`) wskazany dokument otwierał się w przykładowej aplikacji lub był przez nią drukowany.

[**1p**]

1.1.9 Szablon okna dialogowego

Powtórzyć funkcjonalność programu z zadania [1.1.4] używając tym razem wizualnego generatora szablonów okien do zbudowania interfejsu użytkownika. Zamiast `RegisterClass`, `CreateWindow` i jawnej pętli obsługi komunikatów użyć funkcji `DialogBox`.

[**1p**]

1.2 Inne podsystemy Windows

1.2.1 Kopiowanie dyskietek

Napisać okieknowy program do kopiowania dyskietek.

Interfejs programu powinien składać się z jednego, dwóch przycisków. Użytkownik powinien być proszony o podanie dyskietki źrodłowej, której zawartość po przeczytaniu powinna zostać odwzorowana na dyskietce docelowej.

W programie należy korzystać wyłącznie z funkcji Win32. Idea niskopoziomowego dostępu do nośnika omówiona jest w podręczniku.

[**2p**]

1.2.2 Plik tekstowy na pulpicie

Napisać program, który na pulpicie bieżącego zalogowanego użytkownika umieści plik tekstowy z bieżącą datą systemową. Do pobrania nazwy foldera użyć funkcji **SHGetFolderPath**.

Zapoznać się z innymi możliwościami tej funkcji.

[**1p**]

1.2.3 Rozmiar okna w rejestrze

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\...

[**1p**]

1.2.4 Komunikacja między instancjami aplikacji

Napisać program, który będzie potrafił przekazywać komunikaty między wieloma instancjami siebie samego.

Program powinien

- użyć funkcji **RegisterWindowMessage** do zarejestrowania w systemie wiadomości wspólnej dla wszystkich instancji
- po naciśnięciu przycisku powinien rozgłaszać jakąś tekstową wiadomość (**GetSystemTime, GetTimeFormat**) do wszystkich okien (**SendMessage** z parametrem HWND_BROADCAST)
- po odebraniu nadanej wiadomości aplikacja powinna zareagować zmianą tytułu belki na (**Odebrano wiadomość: m**), gdzie zamiast **m** powinna pojawić się treść wiadomości.

[**2p**]

1.2.5 Komunikacja między procesowa

Napisać prosty serwer WWW obsługujący minimalny pozdbiór protokołu HTTP umożliwiający użytkownikowi przeglądarki internetowej obejrzenie zawartości przykładowej witryny.

[**2p**]

1.2.6 Problem golibrody

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

[**3p**]

1.2.7 Statystyka połączeń TCP/UDP

Napisać konsolowy (lub okienkowy) program do szczegółowego diagnozowania stanu połączeń TCP lub UDP na lokalnej maszynie. Wykorzystać w tym celu funkcje **GetTcpStatistics** i **GetTcpTable** (lub **GetUdpStatistics** i **GetUdpTable**) z biblioteki **IP Helper** (iphlpapi.h).

[**1p**]

1.2.8 Biblioteka `Url Monikers`

Napisać konsolowy (lub okienkowy) program do pobierania danych z sieci Internet za pomocą funkcji **UrlDownloadToFile** z biblioteki **Url Monikers** (urlmon.h).

Podczas pobierania użytkownik powinien być informowany o postępie. W tym celu poprawnie zaimplementować interfejs **IBindStatusCallback**.

[**2p**]

1.2.9 Wiggle

Napisać prosty program OpenGL animujący w czasie rzeczywistym sześcian obracający się dookoła swojego środka. Podsystem OpenGL musi być inicjowany bezpośrednio z poziomu Win32, bez użycia interfejsów pomocnicznych (AUX, GLUT).

[**2p**]

1.3 Win32 Varia

1.3.1 Wtyczka DSP do Winampa 2.x

Wzorując się na przykładzie z podręcznika, napisać wtyczkę DSP do Winampa 2.x realizującą efekt zamiany lewego i prawego kanału dźwiękowego.

Uwaga! Winamp 2.x i 5.x mają ten sam interfejs programowania wtyczek.

[**1p**]

1.3.2 Skrypty powłoki

Napisać skrypt powłoki (JScript lub VBScript), który na pulpicie bieżącego zalogowanego użytkownika umieści plik tekstowy z bieżącą datą.

[**1p**]

1.3.3 Internet Explorer jako host dla aplikacji okienkowych

Napisać aplikację HTA, 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 (podać co najmniej dwa powody)?

[**1p**]

1.3.4 Informacje o systemie

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

- Model procesora oraz częstotliwość taktowania (1 dodatkowy punkt za wersję działającą na W95 i W98)
- Ilość pamięci operacyjnej (wolnej, całej)
- Wersję systemu operacyjnego wraz z wersją uaktualnienia (1 dodatkowy punkt za wersję językową)
- 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**]

Rozdział 2

.NET Framework

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

2.1 Język C#

2.1.1 Składnia języka

Składnia C# wzorowana jest na składni C/C++, jednak semantyka pewnych wyrażeń w C# bywa zaskoczeniem dla programistów używających wcześniej C/C++.

Bardzo charakterystycznym przykładem jest lubiany przez programistów kod, powodujący zamianę wartości dwóch zmiennych całkowitoliczbowych bez użycia pomocniczej zmiennej.

```
int x, y;
x ^= y ^= x ^= y;
```

Wyjaśnij dlaczego taki sam kod skompilowany w C i w C# daje różne wyniki. Podaj inny przykład wyrażenia, które ewaluuje się inaczej w C/C++ i w C#.

[**1p**]

2.1.2 Liczby pierwsze

Napisać program, który wyznacza zbiór liczb pierwszych między 0 a 100000. Zastosować metodę najprostszą algorytmicznie, niekoniecznie wydajną obliczeniowo (za wydajny algorytm nie będzie dodatkowych punktów).

[**1p**]

2.1.3 Konstruktory, dziedziczenie

Wytłumaczyć działanie poniższego programu:

```
using System;

class A {
  public A() {
    Console.WriteLine( "A" );
  }
}
```

```
class B {
  public B() {
    Console.WriteLine( "B" );
  }
}

class C : A {
  B b = new B();
}

public class M {
  public static void Main() {
    C c = new C();
  }
}
```

[**1p**]

2.1.4 Polimorfizm

Utworzyć prostą hierarchię 2-3 klas i zademonstrować efekt **polimorficznego** wywołania metod.

Jaki efekt (i dlaczego) będzie miała zamiana w jednej z klas potomnych `override` na `new` przy którejś z metod? Jaki efekt (i dlaczego) będzie miała próba przeciążenia metody, która nie jest oznakowana jako `virtual`?

[**2p**]

2.1.5 Dopasowania metod przeciążonych

Rozważmy następujący kod.

```
using System;

public class A
{
  public virtual void Q( int k )
  {
    Console.WriteLine( "A::Q(int)" );
  }
}

public class B : A
{
  public virtual void Q( double d )
  {
    Console.WriteLine( "B::Q(double)" );
  }
}

public class C
{
  public virtual void Q( double k )
```

```
  {
    Console.WriteLine( "C::Q(int)" );
  }
}

public class D : C
{
  public virtual void Q( int d )
  {
    Console.WriteLine( "D::Q(double)" );
  }
}

class Start
{
  public static void Main()
  {
    B b = new B();
    b.Q( 1.0 );
    b.Q( 1 );

    D d = new D();
    d.Q( 1.0 );
    d.Q( 1 );
  }
}
```

Jaki będzie efekt działania tego kodu i jak go wytłumaczyć?
[**1p**]

2.1.6 Inicjowanie zmiennych statycznych

Objaśnić działanie następującego kodu z dwoma statycznymi zmiennymi odwołującymi się do siebie nawzajem.

```
public class A
{
  public static int a = B.b + 1;
}

public class B
{
  public static int b = A.a + 1;
}

public class CMain
{
  public static void Main()
  {
    Console.WriteLine( "A.a={0}, B.b={1}", A.a, B.b );
```

```
  }
}
```

[**1p**]

2.1.7 Indeksery

Napisać klasę **CPolska** z dwoma indekserami:

- jednowymiarowym, zwracającym tablicę miejscowości leżących na zadanej szerokości geograficznej, tak aby klient klasy mógł napiasć:

```
...
CPolska  Polska = new CPolska();
string[] miejsc = Polska[52.1];
```

- dwuwymiarowym, zwracającym nazwę miejscowości leżącej najbliżej odpowiednich współrzędnych geograficznych, tak aby klient klasy mógł napisać:

```
...
Polska CPolska = new CPolska();
string  miejsc = Polska[15.1, 52.4];
```

Oba indeksery powinny przyjmować jako parametry liczby rzeczywiste.

W wewnętrznych strukturach klasy przechować kilka przykładowych miejscowości (zainicjowanych statycznie lub dynamicznie w konstruktorze).

[**1p**]

2.1.8 Refleksja - składowe prywatne

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 kliencji powinien uzyskać dostęp do składowych prywatnych za pomocą refleksji.

Jeden dodatkowy punkt za porównanie szybkości dostępu do składowej publicznej w zwykły sposób i za pomocą refleksji.

[**1+1p**]

2.1.9 Dokumentowanie kodu

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

Wygenerować dokumentację w postaci pliku XML podczas kompilacji. Użyć narzędzia NDoc (`http://ndoc.sourceforge.net`) do zbudowania pomocy stylach HTML Help i MSDN-online.

[**1p**]

2.1.10 Dekompilacja kodu

Napisać w C# dowolny program demonstrujący użycie klas (metod, pól, propercji, indekserów, delegacji i zdarzeń) oraz podstawowych konstrukcji składniowych (pętle, instrukcje warunkowe, `switch`) i zdekompilować go do wybranego przez siebie języka wysokiego poziomu za pomocą narzędzia **Reflector** (`http://www.aisto.com/roeder/dotnet`).

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?

[**2p**]

2.2 .NET ⇔ Win32, Platform Invoke, COM Interoperability

Możliwości platformy .NET byłyby mocno ograniczone, gdyby niemożliwa była współpraca z kodem niezarządzanym. Podobnie jednak jak istnieją dwa różne typy niezarządzanych bibliotek, bibilioteki 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

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

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.

[**1p**]

2.2.3 P/Invoke + DLL + wskaźniki na funkcje/delegacje

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

To zadanie składa się z 3 części:

1. Napisać bibilotekę 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.*

[**2p**]

2.2.5 COM Interop, COM ⇒ .NET dla istniejących typów

Utworzyć opakowanie istniejącej w systemie biblioteki COM (może to być na przykład biblioteka programu Microsoft Excel) i napisać program kliencki, demonstrujący jej użycie (dodawanie nowego arkusza oraz tworzenie kilku przykładowych wierszy danych).

Napisać analogiczny program używając późnego wiązania w C# oraz jeszcze jeden, w VB.NET.

Scharakteryzować różnice między tymi trzema technikami (wczesne wiązanie, późne wiązanie w C#, późne wiązanie w VB.NET).

[**2p**]

2.2.6 COM Interop, .NET ⇒ COM

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 rejstrowania bibliotek .NET jako komponentów COM. Przy okazji obejrzeć 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**]

2.2.7 Enterprise Interop, Java ⇒ .NET

Wywołać usługę obiektu stworzonego w Javie z poziomu platformy .NET **bez** użycia technik komunikacji międzyprocesowej.

Dodatkowy punkt za oryginalność lub technologiczną elegancję rozwiązania.

[**1+1p**]

2.2.8 Enterprise Interop, .NET ⇒ Java

Wywołać usługę obiektu stworzonego w .NET z poziomu Javy **bez** użycia technik komunikacji międzyprocesowej.

Dodatkowy punkt za oryginalność lub technologiczną elegancję rozwiązania.

[**1+1p**]

2.3 .NET Base Class Library

Biblioteka standardowa platformy .NET zawiera komplet funkcji do komunikacji z usługami systemu operacyjnego Windows. Ponieważ w przyszłych wersjach systemu operacyjnego Windows interfejs BCL ma szansę stać się natywnym interfejsem programowania Windows, warto szczegółowo zapoznać się z jego możliwościami.

2.3.1 Liczby zespolone

Napisać klasę do obsługi liczb zespolonych. Dodać odpowiednie konstruktory, przeciążyć odpowiednie operatory. Porównać wydajność obliczeń z użyciem zaprojektowanej klasy z obliczeniami przy użyciu szablonu `complex` z C++ (napisać podobny kawałek kodu z przykładowymi obliczeniami i porównać czas wykonania).

Rozszerzyć tę klasę o własne formatowane. Ściślej, 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**]

2.3.2 Operacje na obiektach typu `string`

Zmierzyć czas działania kodu:

```
int    k = 1000;
string s = string.Empty;

for ( int i=0; i<k; i++ )
  s += i.ToString();
```

dla rosnących `k` (powiedzmy co 1000 do 50000).
Jakie zjawisko możemy zaobserwować? Jak mu zaradzić?
[**1p**]

2.3.3 Kodowanie napisów

Napisać program do konwersji kodowania plików tekstowych.

Program powinien przyjmować jako parametry: nazwę pliku wejściowego i wyjściowego oraz kodowanie wejściowe i wyjściowe.

Przykładowe wywołanie programu:

```
encoding-converter.exe in.txt out.txt windows-1250 iso-8859-2
```

oznacza przekodowanie pliku in.txt w kodowaniu `windows-1250` do pliku out.txt w kodowaniu `iso-8859-2`.

Dodatkowy punkt zostanie przyznany za program, który potrafi samodzielnie rozpoznać kodowanie pliku źródłowego (o ile jest to możliwe).

[**2+1p**]

2.3.4 Własne kolekcje

Zaimplementować kolekcję `Set` działającą jak zbiór, odrzucający duplikaty elementów. Które kolekcje wbudowane mogą być użyte jako bazowe dla `Set`?

[**1p**]

2.3.5 Enumeratory opakowujące

Przetestować w praktyce składanie enumeratorów opakowujących. Ściślej, uzupełnić szkic kodu ze str. 211 z podręcznika, tak aby kod zadziałał zgodnie z sugestią.

Czy podobna technika może/musi być użyta dla kontenerów generycznych? Dlaczego?

[**2p**]

2.3.6 Składanie strumieni

Napisać program, który zawartość wskazanego pliku tekstowego zapisze do zaszyfrowanego wybranym algorytmem strumienia GZip.

Napisać kolejny program, który odszyfruje wskazany strumień GZip.

Uwaga! Zamiast bibliotecznego GZip można użyć Zip z biblioteki **SharpZipLib** *za dodatkowy punkt.*

[**1+1p**]

2.3.7 Golibroda w .NET

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

[**3p**]

2.3.8 Protokoły sieciowe

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

[**2p**]

2.3.9 Własna usługa sieciowa + serializacja

Problem obiektowych systemów rozproszonych polega na konieczności przesyłania obiektów między odległymi platormami.

Zademonstrować możliwość serializowania obiektów po stronie serwera, przesyłania ich za pomocą protokołu TCP do klienta (używając do tego celu `TcpClient` i `TcpListener`) i odtwarzania ich stanu u klienta przez deserializację.

Która metoda serializacji jest do tego celu najlepsza, a która najgorsza?

[**2p**]

2.3.10 Komunikacja międzyprocesowa - MSMQ

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.

Scharakteryzować różnice między komunikatami MSMQ, a komunikatmi Windows.

[**2p**]

2.3.11 Globalizacja

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**]

2.3.12 Przeglądanie Active Directory

Używając obiektów `DirectoryEntry` i `DirectorySearcher` znaleźć za pomocą protokołu LDAP (Lightweight Directory Access Protocol) listę dostępnych usług Active Dictory, a następnie pokazać listy ich użytkowników (imię, nazwisko, e-mail).

Dwa dodatkowe punkty za uniwersalną przeglądarkę usług katalogowych, umożliwiającą połączenia do dowolnych dostawców usługi ADSI (Active Directory Services Interface).

Jakie inne usługi, oprócz LDAP, dostarczają implementacji ADSI?

[**2+2p**]

2.3.13 Usługi systemowe

Napisać usługę dla systemu NT, która będzie co minutę wysyłać listę uruchomionych aplikacji na pewien ustalny adres e-mail. Dodatkowo, każdy wysłany komunikat powinien być odnotowany w systemowym rejestrze zdarzeń (Event Log).

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**.

[**3p**]

2.4 Rozszerzenia platformy .NET 2.0

2.4.1 Kontenery generyczne

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

[**1p**]

2.4.2 Drzewo binarne

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?

[**2+2p**]

2.4.3 Anonimowe delegacje `Predicate`, `Action`, `Comparison`, `Converter`

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

[**1p**]

2.4.4 Algorytmy biblioteczne

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.

Uwaga! Kod nie powinien być wzorowany na oryginalnej implementacji.

```
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 );
}
```

[**3p**]

2.5 Biblioteka `System.Windows.Forms`

2.5.1 Potwierdzenie zamknięcia okna

Napisać program, który podczas próby zamknięcia okna poprosi użytkownika o potwierdzenie (*Czy jesteś pewien, że chcesz zakończyć program?*) i w razie odpowiedzi odmownej zrezygnuje z zamykania okna.

Uwaga! Za wersję, która do tego celu obsługuje odpowiedni komunikat będą przyznawane punkty **ujemne***!*

[**1p**]

2.5.2 Podsystem `GDI+`

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.

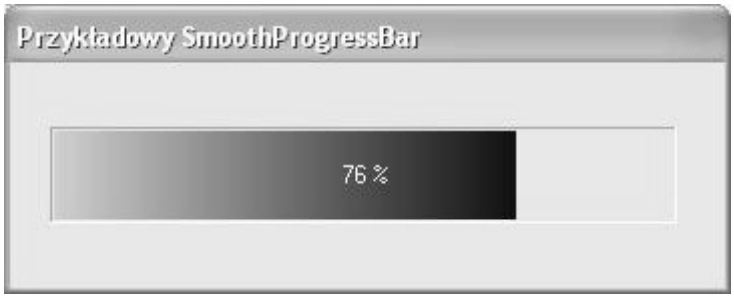
Wykorzystać funkcje do rysowania z GDI+.

[**2p**]

2.5.3 Przeglądarka plików graficznych

Napisać program będący przeglądarką do plików graficznych (wykorzystać możliwości klasy `Image`).

Interfejs programu powinien umożliwiać



Rysunek 2.1: Przykładowy `SmoothProgressBar`

- otwieranie plików graficznych
- konwersję do innych formatów graficznych
- zmianę rozdzielczości obrazków
- obrót wokół osi pionowej lub poziomej
- kilka prostych filtrów graficznych, np.
  - konwersję kolorów w odcienie szarości
  - rozmycie obrazu

Aplikacja powinna pozwalać na jednoczesną pracę w wielu oknach roboczych (MDI).
[**3p**]

2.5.4 Formant `SmoothProgressBar`

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).

[**2p**]

2.5.5 Formant `Grid`

Zaimplementować własny komponent `Grid`, który będzie udostępniał funkcjonalność siatki.

Poszczególne pola siatki powinny być reprezentowane przez obiekty typu `GridCell`.

Interfejs klasy `GridCell`:

```
Color  BackColor { get; set; }
Color  ForeColor { get; set; }
Font   Font { get; set; }
string Text { get; set; }
int    Width { get; set; }
int    Height { get; set; }

Size   GetRequiredSize(Graphics g);
```

Interfejs klasy `Grid`:

```
Color  CellBackColor { get; set; }
Color  CellForeColor { get; set; }
Font   Font { get; set; }
int    Rows{ get; }
int    Cols{ get; }

GridCell this[int x, int y] { get; }
GridCell CellUnderMouse { get; }

void     Redim( int x, int y );
void     Clear();
void     AutosizeCells();
```

Przykładowy kod klienta:

```
class TestForm : Form
{
  ...
  private Grid grid;

  void InitializeComponent()
  {
    ...
    grid = new Grid();
    grid.Size = new Size( 250, 100 );
    grid.Location = new Point( 0, 0 );
    grid.Font = new Font( "Tahoma", 12 );
    this.Controls.Add( grid );
  }

  void SetupGrid()
  {
    int R = 10, C = 10;

    grid.Redim( R, C );

    for ( int r=0; r<R; r++ )
      for ( int c=0; c<C; c++ )
      {
        grid[r,c].Font      = new Font( "Tahoma", r+c+6 );
        grid[r,c].BackColor = Color.Yellow;
        grid[r,c].Text      = string.Format( "[{0},{1}]", r,c );
      }
  }
```

[**3p**]

2.5.6 Windows Media Player ActiveX

Napisać program, który użyje techniki hostowania w aplikacjach .NET formantów ActiveX i udostępni użytkownikowi formant Windows Media Player.

Udostępniony formant powinien umożliwiać otwieranie i odtwarzanie dowolnych plików multimedialnych oraz przerywanie odtwarzania na życzenie użytkownika.

[**1p**]

2.6 .NET Varia

2.6.1 Wielojęzykowość .NET

Napisać program złożony z co najmniej 3 modułów, z których **co najmniej jeden** będzie napisany w innym języku niż C#.

[**1p**]

2.6.2 Nieoczekiwany problem sortowania

Poniższy kod powinien był posortować kolekcję `ArrayList` tak, aby utrzymać pozycję ustalonego, ostatniego elementu. Jednakże uruchomienie programu pod .NET 1.1 powoduje jego zawieszenie się.

```
using System;
using System.Collections;


class TestComparer : IComparer
{
  public int Compare(object x, object y)
 {
  if (!( x is string )) return 0;
  if (!( y is string )) return 0;

  string xs = x as string;
  string ys = y as string;

  if ( ys == "fixed" )
    return -1;
  if ( xs == "fixed" )
    return 1;

  return xs.CompareTo( ys );
 }
}


class Test
{
  public static void Main()
  {
    ArrayList test = new ArrayList();
```

```
    test.Add( "test2" );
    test.Add( "test1" );
    test.Add( "fixed" );
    test.Add( "test5" );
    test.Add( "test4" );
    test.Add( "test3" );

    foreach ( string s in test )
      Console.WriteLine( s );

    Console.WriteLine();
    test.Sort( new TestComparer() );

    foreach ( string s in test )
      Console.WriteLine( s );
  }
}
```

Należy odpowiedzieć na następujące pytania:

1. dlaczego powyższy kod zawiesza się pod kontrolą .NET 1.1?
2. dlaczego powyższy kod nie zawiesza się pod kontrolą .NET 2.0?
3. dlaczego powyższy kod nie zawiesza się pod kontrolą .NET 1.1 gdy ustalonym elementem będzie pierwszy element listy:

```
if ( ys == "fixed" )
  return 1;
if ( xs == "fixed" )
  return -1;
```

4. Jak poprawić kod tak, aby działał poprawnie pod kontrolą .NET 1.1?

[**2p**]

2.6.3 Tablica o małym zużyciu pamięci

Napisać klasę `TArray`, która będzie pewną specjalną implementacją tablicy elementów.

Wewnątrz obiektu klasy dane powinny być przechowywane na drzewie, w którym każdy węzeł ma 10 synów, oznaczonych indeksami od 0 do 9. Aby dostać się do elementu o indeksie $i = \sum_{j=0}^{k} 10^j * i_j$ przechodzimy drzewo, na poziomie j przechodząc do syna i_j.

Na przykład chcąc uzyskać dostęp do elementu o indeksie 7 wybieramy 7 syna korzenia drzewa i odczytujemy zapamiętany w nim element. Aby uzyskać dostęp do elementu o indeksie 145 przechodzimy kolejno przez 5-ego, 4-ego i 1-ego syna kolejnych węzłów począwszy od korzenia.

Odpowiednie gałęzie drzewa powinny być budowane tylko wtedy, kiedy do tablicy dodawny jest element o odpowiednim indeksie. Na przykład dodanie do tablicy elementu o indeksie 10000000000 powinno spowodować powstanie tylko jednej długiej gałęzi od 0-ego syna korzenia, przez dziesięciu kolejnych synów kolejnych węzłów. Bezpośrednio po zainicjowaniu korzeń drzewa powinien mieć tylko 10 pustych referencji na kolejnych synów.

Dzięki takiej konstrukcji użytkownik będzie mógł dodać na przykład element o indeksie 1 i element o indeksie 10000, a w drzewie będą przechowane tylko te 2 elementy (plus oczywiście puste referencje na pozostałe elementy w kolejnych węzłach). Tablica nie będzie więc (jak zwykła tablica liniowa) zużywać miejsca na wszystkie brakujące elementy między 1 a 10000.

Zdefiniować odpowiedni indekser, tak aby do elementów tablicy można było odwoływać się w "zwykły" sposób, na przykład:

```
TArray a   = new TArray();
a[17]      = 5;
a[1000000] = 176;
```

Zdefiniować odpowiedni enumerator, tak aby elementy tablicy można było przeglądać w "zwykły" sposób, na przykład:

```
TArray a   = new TArray();
a[17]      = 5;
a[1000000] = 176;
foreach ( int i in a )
...
```

Porównać wydajność (tworzenie, przeglądanie):

- TArray
- ArrayList
- zwykłych tablic

[**4p**]

2.6.4 Asekwencyjność kolekcji asocjacyjnych

Wykazać (pisząc odpowiedni program), że zarówno `Hashtable` jak i `Dictionary<T,K>` **nie** są kolekcjami sekwencyjnymi, czyli nie zachowują kolejności umieszczanych w nich elementów.

[**1p**]

2.6.5 Sekwencyjna generyczna kolekcja asocjacyjna

Zaimplementować generyczną kolekcję asocjacyjną, która będzie miała własność sekwencyjności, tzn. przeglądanie kolekcji `Keys` i `Values` zwróci elementy w kolejności w jakiej były do kolekcji dodawane.

Wskazówka: generyczna kolekcja asocjacyjna musi implementować interfejs `IDictionary<T,K>`. *Zarówno same elementy jak i ich kolejność zapamiętać w pomocniczych kolekcjach wewnętrznych (jakich?).*

[**2p**]

2.6.6 Lekser, parser, rekursja

Napisać program wykonujący symboliczne obliczanie pochodnej. Wykorzystać rekurencyjne wzory:

$$\begin{aligned}(f+g)' &= f'+g' \\ (f-g)' &= f'-g' \\ (fg)' &= fg'+f'g \\ (\frac{f}{g})' &= \frac{f'g-fg'}{g^2} \\ (ax^n)' &= nax^{n-1}\end{aligned}$$

Program powinien z linii poleceń przyjmować wyrażenie, które następnie należy sparsować i pokazać wynik. Analizę leksykalną i składniową oprzeć na dowolnej bibliotece do automatycznego tworzenia lekserów i paserów, np:

- **CSTools** (`http://cis.paisley.ac.uk/crow-ci0`)
- **GOLD Parser** (`http://www.devincook.com/GOLDParser/index.htm`)
- **ANTLR** (`http://www.antlr.org/`)

[**3p**]

2.6.7 Informacje o systemie w .NET

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 i wersją językową
- Numer wersji środowiska uruchomieniowego, którym kompilowano program
- Numer wersji środowiska uruchomieniowego, które nadzoruje wykonanie bieżącego programu
- Nazwę sieciową komputera i nazwę aktualnie zalogowanego użytkownika
- Ustawienia rozdzielczości i głębi kolorów pulpitu
- Listę drukarek podłączonych do systemu
- Numery wersji aplikacji
  - Internet Explorera
  - Microsoft Worda

[**3p**]

2.7 Programowanie urządzeń mobilnych

2.7.1 Gra planszowa

Napisać program, który pozwala dwóm użytkownikom na rozegranie partii dowolnej gry planszowej (kółko-krzyżyk, warcaby, Othello, Link 5) na urządzeniu mobilnym.

Punktacja za rozwiązane zadanie zależy od tego czy binarium jest przenaszalne pomiędzy platformami mobilnymi a Win32. Jeśli nie - rozwiązanie jest warte 5 punktów, jeśli tak - 10 punktów.

Kolejne dodatkowe 5 punktów można uzyskać za staranność wykonania programu. Punkty te będą przyznawane według subiektywnej oceny prowadzącego. *Uwaga! Kółko-krzyżyk nie kwalifikuje się do tej kategorii dodatkowych punktów!*

Wskazówka: techniki programowania interfejsu użytkownika w tego typu programach można podejrzeć w przykładowym programie **Trilma.NET***, którego kod źródłowy (również w wersji mobilnej, przenaszalnej między platformami) można pobrać ze strony Internetowej.*

[**5/10p+5**]

2.8 eXtensible Markup Language

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

2.8.1 XML

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.

[**1p**]

2.8.2 XSD

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**]

2.8.3 XML + XSD

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

[**1p**]

2.8.4 XML - serializacja

Napisać prostego klienta struktury XML z zadania 2.8.3, który pliki XML czyta i zapisuje mechanizmem serializacji do struktur danych zamodelowanych odpowiednimi atrybutami.

[**1p**]

2.8.5 XML - DOM

Napisać prostego klienta struktury XML z zadania 2.8.3, który pliki XML czyta i zapisuje za pomocą modelu DOM (`XmlDocument`).

[**1p**]

2.8.6 XML - strumienie

Napisać prostego klienta struktury XML z zadania 2.8.3, który pliki XML czyta i zapisuje za pomocą mechanizmów strumieniowych (`XmlTextReader, XmlTextWriter`).

[**1p**]

2.8.7 XML - analiza

Porównać trzy metody obsługi XML z poprzednich zadań. Porównanie powinno uwzględniać:

1. czas odczytu/zapisu
2. łatwość implementacji odczytu/zapisu
3. podatność na konserwację (np. ewolucję struktury)

Ponadto rozważyć model aplikacji, w której **nie ma** żadnych pośrednich struktur danych w których przechowywane byłyby dane z XML, a warstwa logiki biznesowej korzysta bezpośrednio ze struktury XML przechowywanej w pamięci na przykład w obiekcie DOM.

Rozwiązanie zadania powinno mieć formę pisemną i nie powinno przekraczać 150 słów.

[**1p**]

2.8.8 XML jako protokół komunikacyjny

Napisać prostego okienkowego klienta protokołu RSS (w dowolnej wersji).

Klient powinien nawiązywać połączenie sieciowe do wskazanego źródła danych i udostępniać listę publikowanych informacji. Wybór linka przez użytkownika powinien powodować pobranie zawartości wskazanego artykułu i zaprezentowanie go użytkownikowi w uruchomionej z boku nowej instancji przeglądarki internetowej lub ([**+1p**]) w kontrolce ActiveX Internet Explorera hostowanej w obrębie aplikacji.

Uwaga! Do połączeń HTTP użyć gotowych klas z przestrzeni nazw `System.Net`.

Uwaga! Mechanizm hostowania kontrolki IE omówiony był na wykładzie.

[**3+1p**]

2.9 Biblioteka ADO.NET

Bibiloteka ADO.NET udostępnia spójny interfejs do obsługi różnych rodzajów źródeł danych. Informacje o właściwym inicjowaniu parametrów połączenia (`ConnectionString`) powinny być oczywiście dostępne w dokumentacji źródła danych, jednak dla typowych źródeł danych parametry te są ogólnie znane (np. `http://www.connectionstrings.com`).

2.9.1 DataReader

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 w ListView.

[**1p**]

2.9.2 DBMS

Przygotować bazę danych zawierającą dane osobowe i adresy przykładowej grupy studentów (dwie tabelki `STUDENCI` i `ADRESY` połączone relacją jeden-do-wielu) w dowolnym relacyjnym systemie bazodanowym (Access, MySql, MSSQL, itp.).

[**1p**]

2.9.3 Data Access Layer

Napisać prostą aplikację okienkową, która udostępnia dane z bazy z poprzedniego zadania użytkownikowi w trybie w trybie *do odczytu i zapisu*.

Wybrać dowolny model obsługi danych po stronie aplikacji klienckiej - dane odczytywać do lokalnych struktur i odsyłać odpowiednie zapytania lub użyć DataAdapterów i DataSetów powiązanych z DataGrid.

[**3p**]

2.9.4 Object-Relational Mapping

Zapoznać się z dowolną implementacją ORM (NHibernate, Sooda, Wilson ORM itd.) dla platformy .NET i zademonstrować jej działanie na bazie danych z poprzednich zadań.

[**3p**]

2.9.5 Własny ORM, atrybuty, refleksja

Zaprojektować zbiór atrybutów i napisać funkcję do automatycznego tworzenia kwerend SQL dla modelu obiektowego bazy danych z poprzednich zadań. Automatyczny generator kwerend powinien odczytać informację o polach obiektu, wydobyć te, które oznakowane są odpowiednimi atrybutami, i na ich podstawie budować kwerendy CRUD (Create, Retrieve, Update, Insert).

```
[ORMClass( tablename="STUDENCI")]
class Student
{
  [ORMIdentity( fieldname="ID")]
  public int ID;

  [ORMField( fieldname="Name", maxlength=80 )]
  public string Nazwisko;

  [ORMField( fieldname="DataUr" )]
  public DateTime DataUr;

  ...
}

class SimpleORM
{
  ...

  public static List<T> RetrieveAll<T>();
```

```
   public static T RetrieveOne<T>( object id );
   public static void Update<T>( T t );
   public static void Insert<T>( T t );
   public static void Delete<T>( T t );
}
```

Dodatkowe 2 punkty za ładne użycie wzorca **Strategy** dla tworzenia wymienialnych generatorów kwerend dla co najmniej dwóch różnych DBMSów:

```
SimpleORM<OracleORMProvider> oracle_orm = new SimpleORM<OracleORMProvider>();
SimpleORM<SQLORMProvider> sql_orm = new SimpleORM<SQLORMProvider>();
```

[**3+2p**]

2.10 Biblioteka ASP.NET. ASP.NET WebServices

2.10.1 Rejestr odwiedzin

Przygotować stronę, która każde odwiedziny zarejestruje w pliku tekstowym, zapisując datę i numer IP komputera klienta.

[**1p**]

2.10.2 Statystyka odwiedzin

Przygotować stronę, która udostępni statystykę rejestru odwiedzin z poprzedniego zadania. Statystyka powinna zawierać numery IP uporządkowane według liczby połączeń, przy każdym numerze IP powinna znajdować się liczba połączeń oraz data ostatniego połączenia.

[**2p**]

2.10.3 ASP.NET+ADO.NET

Przygotować stronę, która w DataGrid/GridView wyświetli zbiór rekordów z dowolnego zewnętrznego źródła danych wczytany za pomocą ADO.NET do DataSet.

[**1p**]

2.10.4 Uwierzytelnianie

Zaproponować i zrealizować w dowolny sposób system uwierzytelniania w ASP.NET. Może być to albo jeden z udostępnianych przez ADO.NET (Forms, Windows, Passport) albo własny, oparty np. o informacje przechowywane w zmiennych sesji czy ciasteczkach.

Realizacja praktyczna powinna składać się z 2 stron: `login.aspx` i `main.aspx`. Próba dostępu do `main.aspx` bez autoryzacji powinna powodować przekierowanie do `login.aspx`. Poprawna autoryzacja powinna objawiać się ujawnieniem w treści strony `main.aspx` nazwy zalogowanego użytkownika.

[**1p**]

2.10.5 WebService - autentykacja

Przygotować WebService udostępniający metodę, która dla podanej dodatniej liczby naturalnej k zwraca najmniejszą liczbę pierwszą nie mniejszą niż k. WebService powinien autentykować użytkownika, który wywołuje funkcję i zwrócić wyjątek w wypadku błędnej autentykacji.

Wskazówka: autentykacja WebServices może odbywać się na kilka różnych sposobów, m.in:

- *uwierzytelnianie zintegrowane, tożsamość przekazana w propercji* `CredentialCache` *obiektu klienta*
- *własny nagłówek SOAP (*`SOAPHeader`*), po stronie serwera explicite sprawdzenie tożsamości*
- *jawna funkcja logująca, która w obiekcie sesji WebService umieści odpowiednie ciasteczko* `FormsAuthentication`

[**2p**]

2.10.6 WebService - dane binarne

Przygotować WebService który będzie potrafił przesyłać dane binarne (na przykład pliki).

Jak rozwiązać problem pojawiający się, gdy rozmiar przesyłanych danych jest stosunkowo duży (na przykład przesyłanie pliku wielkości 100MB)?

[**2p**]

2.10.7 WebService - kompresja

Przygotować rozszerzenie SOAP (`SOAPExtension`), które w zdarzeniu `AfterSerialize` po stronie serwera spakuje treść (Body) komunikatu SOAP dowolnym algorytmem kompresji, a w zdarzeniu `BeforeDeserialize` po stronie klienta rozpakuje treść komunikatu.

Porównać zysk na wielkości przesyłanych danych dla typowych obiektów, np. `DataSet`.

[**3p**]

2.11 Bezpieczeństwo platformy .NET

2.11.1 Weryfikacja poprawności MSIL i metadanych

Zdekompilować prosty program napisany w C# do MSIL i zmodyfikować kod tak, aby któraś z funkcji powodowała nadmiar/niedomiar stosu. Skompilować program za pomocą `ildasm.exe` i uruchomić.

Obejrzeć szczegółowy raport diagnostyczny narzędzia `PEVerify.exe`.

(Dodatkowy punkt) Czy i jak wykrywana jest niezgodność głębokości/typów wartości na stosie dla pętli?

[**1+1p**]

2.11.2 Polisa bezpieczeństwa aplikacji

Korzystając z narzędzia konfiguracyjnego platformy .NET zdefiniować nową grupę kodu (Code Groups) dla bieżącego użytkownika, dla której regułą przynależności będzie lokalizacja w systemie plików (np. `C:/Sandbox`), następnie zdefiniować dla tej grupy nowy zbiór reguł bezpieczeństwa (Permission Sets), który da aplikacji nieograniczony dostęp tylko do plików w jej folderze.

Następnie napisać program, który spróbuje przeczytać plik w innej lokalizacji niż bieżący katalog, umieścić go w folderze utworzonej grupy kodu i uruchomić.

Omówić efekt działania programu.
[**1p**]

2.11.3 Szyfrowanie kodu aplikacji

Bibliotekę zawierającą logikę aplikacji zaszyfrować dowolnym algorytmem kryptograficznym platformy .NET.

W głównym module aplikacji prosić użytkownika o podanie kodu, użyć go do rozkodowania biblioteki do pamięci, a następnie użyć refleksji do uruchomienia głównej części logiki aplikacji.

Kiedy taki scenariusz mógłby być użyteczny?
[**2p**]

2.11.4 Silny podpis kodu aplikacji

Bibliotekę zawierającą logikę aplikacji silnie podpisać na etapie kompilacji kluczem wygenerowanym programem `sn.exe`.

Przygotować duplikat biblioteki z logiką aplikacji, o tej samej nazwie i tej samej zawartości (w sensie sygnatur), ale nie posiadającej silnego podpisu.

Jak zachowuje się moduł główny, uruchomiony z duplikatem biblioteki zamiast z oryginałem? Kiedy taki scenariusz mógłby być użyteczny?
[**1p**]

Bibliografia

[1] *http://msdn.microsoft.com*

[2] Wiktor Zychla *Windows oczami programisty, Mikom*

[3] Archer T., Whitechapel A. *Inside C#, Microsoft Press*

[4] Eckel B. *Thinking in C#, http://www.bruceeckel.com*

[5] Gunnerson E. *A Programmer's Introduction to C#*

[6] Lidin S. *Inside Microsoft .NET IL Assembler, Microsoft Press*

[7] Petzold Ch. *Programming Windows, Microsoft Press*

[8] Reilly Douglas J. *Designing Microsoft ASP.NET Applications*

[9] Scott Mitchel *ASP.NET Data Web Controls Kick Start*

[10] Nikhil Kothari, Vandana Datje *Developing Microsoft ASP.NET Server Controls and Components*

[11] Andrew Troelsen *COM and .NET Interoperability*