Programowanie pod Windows
Zbiór zadań

Wiktor Zychla
Instytut Informatyki
Uniwersytetu Wrocławskiego

Wersja 2013.03.01

Spis treści

Wprowadzenie

Szanowni Państwo!

Niniejszy zbiór zadań przeznaczony jest dla słuchaczy wykładu **Programowanie pod Windows .NET**, który mam przyjemność prowadzić w Instytucie Informatyki Uniwersytetu Wrocławskiego od roku akademickiego 2002/2003. Zbiór stanowi uzupełnienie podręcznika, pozycji **Windows oczami programisty** [2], dostępnej w wersji akademickiej jako **Programowanie pod Windows**.

Zadania zebrano w dwie grupy, z których pierwsza pozwala zapoznać się z podsystemami Windows, interfejsem Win32 oraz technologią COM, 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
wzychla@ii.uni.wroc.pl

Zestawy zadań

Zestaw 1

1. 1.1.1
2. 1.1.2
3. 1.1.3
4. 1.1.4
5. 1.1.5
6. 1.1.6
7. 1.1.7
8. 1.1.8

Zestaw 2

1. 1.2.1
2. 1.2.2
3. 1.2.3
4. 1.2.4
5. 1.2.6
6. 1.2.7

Zestaw 3

1. 1.3.1
2. 1.3.2
3. 1.3.3
4. 1.3.4
5. 2.1.1
6. 2.2.1

Zestaw 4

1. 3.1.1
2. 3.1.2
3. 3.1.3
4. 3.1.4
5. 3.1.5
6. 3.1.6
7. 3.2.1
8. 3.2.2
9. 3.2.3
10. 3.2.4
11. 3.3.1

Zestaw 5

1. 3.3.2
2. 3.3.3
3. 3.3.4
4. 3.3.5
5. 3.3.6
6. 3.3.7
7. 3.3.8
8. 3.3.9
9. 3.4.1
10. 3.4.2

Zestaw 6

1. 3.5.1
2. 3.5.2
3. 3.5.3
4. 3.5.4
5. 3.5.5

Zestaw 7

Zestaw 8

Zestaw 9

Zestaw 10

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.

Wskazówka. Przejrzeć dokumentację i znaleźć komunikat, który wysyłany jest do okna tuż przed jego zamknięciem, a którego obsługa da możliwość zapytania użytkownika o zgodę na zamknięcie i ewentualne anulowanie zamknięcia.

[**1p**]

1.1.2 Wykresy funkcji

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.

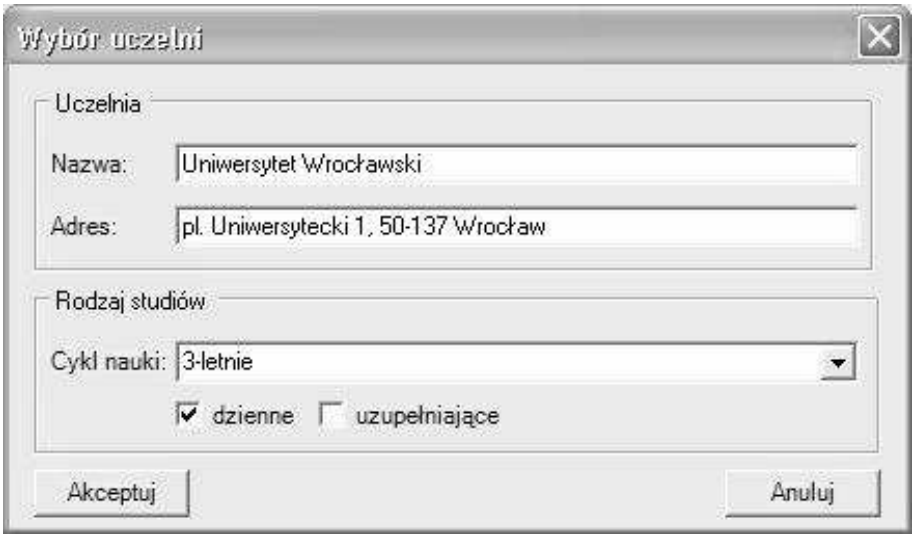
[**1p**]

1.1.3 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**]



Rysunek 1.1: Wygląd okna do zadania [1.1.4]

Rysunek 1.2: Informacja dla użytkownika do zadania [1.1.4]

1.1.4 Okno dialogowe

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).

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**]

1.1.5 Szablon okna dialogowego

Powtórzyć funkcjonalność programu z zadania [1.1.4] 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, zmiast `RegisterClass`*,* `CreateWindow` *i jawnej pętli obsługi komunikatów użyć funkcji* `DialogBox`*.*

[**2p**]

1.1.6 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**]

1.1.7 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, ponieważ odpowiedni WM_COMMAND trafia do funkcji obsługi komunikatów panelu grupującego, a nie okna głównego (panel grupujący jest bezpośrednim rodzicem przycisku).

Przygotować rozwiązanie, w którym własna funkcja obsługi komunikatów panelu grupującego przekazuje trafiający do niej WM_COMMAND "poziom wyżej", czyli do bezpośredniego rodzica panelu grupującego. Dzięki temu logika przetwarzania WM_COMMAND będzie mogła znaleźć w funkcji obsługi komunikatów okna głównego, a trafi tam bez względu na złożoność hierarchii zagnieżdżonych paneli grupujących.

[**1p**]

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.

Wskazówka. Nie ma programowego sposobu na rejestrowanie skojarzeń dla rozszerzeń nazw plików. Odpowiednia informacja znajduje się w rejestrze systemu, należy tylko sprawdzić gdzie powinna się znajdować i jak być zbudowana.

[**1p**]

1.2 Inne podsystemy Windows

1.2.1 Plik tekstowy na pulpicie, powłoka

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 sterujacej powłoką **ShellExecute**.

[**1p**]

1.2.2 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\...

[**2p**]

1.2.3 Komunikacja międzyprocesowa

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

Wskazówka: szkielet kodu serwera TCP/IP znajduje się w podręczniku. Należy go rozbudować o obsługę protokołu komunikacyjnego, tu: HTTP. Klientem będzie przeglądarka internetowa, która po wpisaniu adresu naszego serwera wyśle do niego żądanie - pierwszym żądaniem będzie GET nazwazasobu. Obsługę protokołu HTTP można więc ograniczyć wyłącznie do obsługi żądań GET - to wystarczy do nawiązania komunikacji z przeglądarką internetową.

[**2p**]

1.2.4 Problem golibrody

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

[**2p**]

1.2.5 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).

[**2p**]

1.2.6 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**.

Wskazówka: implementowanie interfejsu w C++ wygląda bardzo podobnie jak implementowanie interfejsu w C# czy Javie. Przykłady implementacji tego konkretnego interfejsu można znaleźć w sieci.

[**2p**]

1.2.7 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/5.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. Komplet narzędzi SDK dla Winampa należy pobrać ze strony `http://www.winamp.com/`

[**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 (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**]

1.3.4 Informacje o systemie

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**]

Rozdział 2

COM Component Object Model

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

2.1 Klient COM

2.1.1 Klient COM aplikacji MS Office

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**]

2.2 Serwer COM

2.2.1 Prosty serwer COM

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**]

Rozdział 3

.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#.

3.1 Język C# 1.0

3.1.1 Prosty algorytm

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

[**1p**]

3.1.2 Indeksery

Zaimplementować klasę `Grid` z dwoma indekserami:

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

```
...
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**]

3.1.3 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 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 );
```

[**1p**]

3.1.4 Atrybuty

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

Następnie sprośród 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**]

3.1.5 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 SandCastle Help File Builder (`http://shfb.codeplex.com/`) do zbudowania pomocy stylach HTML Help i MSDN-online.

[**1p**]

3.1.6 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 (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?

[**2p**]

3.2 Rozszerzenia języka C# 2.0

3.2.1 Kontenery generyczne

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

[**1p**]

3.2.2 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**]

3.2.3 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**]

3.2.4 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.

```
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 );
}
```

[**2p**]

3.3 Rozszerzenia języka C# 3.0

3.3.1 Metoda rozszerzająca klasę `System.String`

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**]

3.3.2 LINQ to Objects, sortowanie, filtrowanie

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**]

3.3.3 LINQ to Objects, grupowanie

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**]

3.3.4 LINQ to Objects, anagramy

Dany jest plik tekstowy zawierający zbiór słów w kolejnych liniach.

Napisać wyrażenie LINQ, które zwróci największą w sensie liczebności grupę anagramów z podanego zbioru.

Uwaga! Anagramami nazywamy słowa, które zawierają te same litery, tylko w innej kolejnosci, na przykład *shore* i *horse*.

Wskazówka: należy użyć operatora grupowania słów, przy czym kryterium grupowania powinno wyrażać właściwość "normy anagramu", czyli zbioru liter wspólnych dla grupy anagramów.

[**2p**]

3.3.5 LINQ to Objects, agregowanie

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`.

[**2p**]

3.3.6 LINQ to Objects, Join

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**]

3.3.7 LINQ to Objects, analiza logów serwera

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**]

3.3.8 Lista obiektów anonimowych

Listy generyczne ukonkretnieniamy 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, jesli mają tę samą liczbę składowych tego samego typu w tej samej kolejnosci.

[**1p**]

3.3.9 Rekursywne anonimowe delegacje

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 anonimowej delegacji określonej rekursywnie:

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

[**2p**]

3.4 Rozszerzenia języka C# 4.0

3.4.1 Wydajność podsystemu DLR

Przeprowadzić testy porównawcze kodu, w którym metoda będzie miała parametr raz typu konkretnego, a drugi raz - dynamicznego. Jak bardzo wolniejsze jest wykonywanie kodu dynamicznego 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**]

3.4.2 Łatwa automatyzacja w C#

Napisać w 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".

[**1p**]

3.5 .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.

3.5.1 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**]

3.5.2 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.

[**2p**]

3.5.3 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**]

3.5.4 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.*

[**3p**]

3.5.5 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).

[**5p**]

3.6 .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.

3.6.1 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**]

3.6.2 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`.

[**1p**]

3.6.3 Własne kolekcje

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**]

3.6.4 Składanie strumieni

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

Napisać kolejny program, który odszyfruje wskazany strumień GZip.

[**1p**]

3.6.5 Prosty strumień pośredni

Zaimplementować klasę strumienia `NegStream`, który będzie działał jak strumień pośredni (wymaganym parametrem konstruktora będzie inny strumień, na którym operował będzie `NegStream` przy czym odczytując i zapisując dane będzie **negował** poszczególne bity danych. Przykład użycia:

```
FileStream fileToWrite = File.Create( ... );
NegSteram negToWrite = new NegStream( fileToWrite );

negToWrite.Write( ... );

FileStream fileToRead = File.Open( ... );
NegStream negToRead = new NegStream( fileToRead );

negToRead.Read( ... );
```

[**1p**]

3.6.6 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.

[**2p**]

3.6.7 Protokoły sieciowe

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

[**1p**]

3.6.8 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**]

3.6.9 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.

[**2p**]

3.6.10 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**]

3.6.11 Active Directory

Używając obiektów `DirectoryEntry` i `DirectorySearcher` znaleźć za pomocą protokołu LDAP (Lightweight Directory Access Protocol) użytkowników (imię, nazwisko, e-mail) usługi katalogowej.

Zrobić to samo za pomocą obiektu `PrincipalSearcher`.

Który sposób dostępu do usługi katalogowej jest łatwiejszy?

Uwaga! Usługi katalogowe są dostępne w naszej instytutowej sieci lokalnej, a co za tym idzie - napisany kod łatwo sprawdzić. Alternatywnie, można posłużyć się jakąś lekką implementacją, na przykład OpenLDAP czy OpenDS.

[**2p**]

3.6.12 Usługa systemowa

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**.

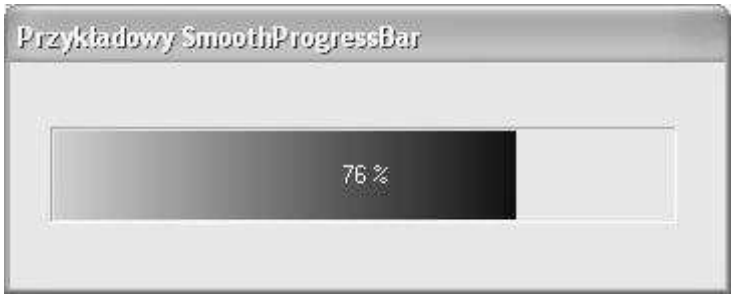
[**2p**]

3.6.13 Zewnętrzny plik w zasobach aplikacji

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.

```
ResourceHelper Helper = new ResourceHelper( Assembly.GetExecutingAssembly() );
Stream PlikzZasobow = Helper.RetrieveResource( "dane.xml" );
```

[**1p**]



Rysunek 3.1: Przykładowy `SmoothProgressBar`

3.7 Biblioteka `System.Windows.Forms`

3.7.1 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! W bibliotece System.Windows.Forms zaprojektowano do tego celu właściwe zdarzenie. Za wersję, która do tego celu obsługuje odpowiedni komunikat będą przyznawane punkty **ujemne***!*

[**1p**]

3.7.2 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**]

3.7.3 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).

[**1p**]

3.7.4 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**]

3.7.5 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**]

3.7.6 Pomoc kontekstowa

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**]

3.8 Inne zagadnienia .NET

3.8.1 Wielojęzykowość .NET

Napisać program złożony z co najmniej 2 zestawów (assembly), z których **co najmniej jeden** będzie napisany w innym języku niż C#.

[**1p**]

3.8.2 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**]

3.8.3 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**]

3.8.4 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**]

3.8.5 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**]

3.9 Programowanie urządzeń mobilnych

3.9.1 Gra planszowa

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

[**5p**]

3.10 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.

3.10.1 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**]

3.10.2 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**]

3.10.3 XML + XSD

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

[**1p**]

3.10.4 XML - serializacja

Napisać prostego klienta struktury XML z zadania 3.10.3, który pliki XML czyta i zapisuje mechanizmem serializacji do struktur danych zamodelowanych odpowiednimi atrybutami.

[**1p**]

3.10.5 XML - DOM

Napisać prostego klienta struktury XML z zadania 3.10.3, który pliki XML czyta i zapisuje za pomocą modelu DOM (`XmlDocument`).

[**1p**]

3.10.6 XML - strumienie

Napisać prostego klienta struktury XML z zadania 3.10.3, który pliki XML czyta i zapisuje za pomocą mechanizmów strumieniowych (`XmlTextReader, XmlTextWriter`).

[**1p**]

3.10.7 XML - LINQ to XML

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**]

3.10.8 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**]

3.10.9 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.

[**2+1p**]

3.11 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`).

3.11.1 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**]

3.11.2 DBMS

Uwaga! Tego zadania nie wolno oddawać samodzielnie, tylko w połączeniu z którymś z następnych zadań, do których ono się odnosi.

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

Model bazy danych zawiera dwie tabele, tabelę `STUDENCI` z polami Imię, Nazwisko, DataUrodzenia i PESEL oraz tabelę `ADRESY` z polami Miejscowość, KodPocztowy, Poczta, NumerDomu.

Tabele `STUDENCI` i `ADRESY` połączone są relacją jeden-do-wielu.

[**1p**]

3.11.3 Data Access Layer

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

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

[**2p**]

3.11.4 Object-Relational Mapping

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

[**2p**]

3.11.5 LINQ to SQL

Zbudować model obiektowy bazy danych z poprzednich zadań za pomocą narzędzia `sqlmetal.exe`. Pokazać w jaki sposób za pomocą LINQ to SQL można dodawać, modyfikować i usuwać dane w bazie danych.

Pokazać w jaki sposób LINQ to SQL tłumaczy kwerendy obiektowe na zapytania SQL.

[**1p**]

3.11.6 LINQ to DataSet

Przygotować arkusz Excela zawierający skoroszyt z dwiema kolumnami danych: PESEL i NumerKonta.

Pokazać w jaki sposób można przeglądać dane zapisane w skoroszycie za pomocą mechanizmu LINQ to DataSet. Ściślej - dane ze skoroszytu wczytać do obiektu DataSet za pomocą mechanizmu wykorzystanego w zadaniu 3.11.1, a wyrażenie LINQ odnosić do napełnionej struktury DataSet.

[**1p**]

3.11.7 LINQ, łączenie różnych źródeł danych

Napisać wyrażenie LINQ, które połączy dane osobowe z bazy danych z zadania 3.11.2 z danymi ze skoroszytu z zadania 3.11.6.

Ściślej - użyć klauzuli `join` do połączenia danych osobowych z bazy danych z danymi o kontach ze skoroszytu. Polem łączącym jest pole PESEL.

[**2p**]

3.12 Biblioteka ASP.NET. ASP.NET WebServices

3.12.1 Rejestr odwiedzin

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

[**1p**]

3.12.2 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**]

3.13 Bezpieczeństwo platformy .NET

3.13.1 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?

[**2+1p**]

3.13.2 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**]

3.13.3 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*