Programowanie pod Windows

Zestaw 4, Biblioteki .NET

28 kwietnia 2003 roku

1. Rozwiązać zadania 6-7 z zestawu 1 i 3 z zestawu 2, nie korzystając (chyba że jest to niezbędne) z funkcji Win32API, tylko z bibliotek .NET

2. Zaprojektować kolekcję `ArrayList2`, która oprócz standardowego enumeratora dostępnego za pomocą metody `GetEnumerator`, będzie zwracać *enumerator dwukierunkowy* za pomocą funkcji `GetBidirectionalEnumerator`. Enumerator dwukierunkowy powinien oprócz trzech elementów interfejsu *IEnumerable* udostępniać metodę `MovePrev`, która będzie cofać enumerację do poprzedniego elementu względem bieżącego.

3. 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.

   Tego rodzaju tablice są dostępne w niektórych językach programowania.

   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:

- TArray
- ArrayList
- zwykłych tablic

4. Zaimplementować kolekcję `Set` działającą jak zbiór.

5. Przygotowaną w poprzednim zestawie zadań klasę `COsoba` wykorzystać do zaimplementowania prostej aplikacji będącej katalogiem osobowym.

   Aplikacja powinna składować dane w postaci pliku XML, zaś do pamięci ładować je do jakiejś kolekcji standardowej (np. ArrayList). Użytkownik powinien mieć możliwość dodawania, modyfikacji i usuwania danych oraz sortowania (dodać kilka różnych porządków sortowania za pomocą interfejsów *IComparable* i *IComparer*) oraz wyszukiwania (za pomocą kilku przykładowych kryteriów, na przykład podanie daty urodzenia, początkowych liter imienia bądź nazwiska).

   Zaprojektować dwa przykładowe arkusze stylów XSL do prezentacji danych z pliku XML. Wewnątrz aplikacji użytkownik powinien mieć możliwość obejrzenia zestawienia powstałego przez zastosowanie arkusza stylów do formatowania danych XML (do otworzenia pliku XML wykorzystać mechanizmy powłoki udostępniane przez biblioteki .NET, to znaczy że po wybraniu funkcji tworzącej zestawienie powinna uruchomić się przeglądarka internetowa, w której bieżącym dokumentem byłby zbiór danych XML sformatowany przy użyciu wybranego arkusza styli XSL).

6. Zrealizować w praktyce omawiany na wykładzie scenariusz dynamicznego kompilowania kodu aplikacji, przechowywanego na dysku w postaci zaszyfrowanej. Aplikacja po starcie powinna poprosić o hasło, odszyfrować kod źródłowy, skompilować go do pamięci i uruchomić.