Kolekce

C# má generické kolekce podobně jako jazyk C++

	C++	**C#**
Pole automatické délky	vector	List
Pole s unikátními prvky	set	HashSet
Pole s prvky klíč-hodnota	map	Dictionary
Pole FIFO	queue	Queue
Pole LIFO	stack	Stack

var

C# má též automatický typ `var` (podobně, jako C++ `auto`). Při této deklaraci se typ určí kontextově na základě pravé strany přiřazení. Lze tedy psát

```
Dictionary<char,int> data = new Dictionary<char,int>(); // dlouhý zápis
var data = new Dictionary<char,int>(); // krátký, stejně srozumitelný zápis
```

Prvky s uživatelským typem

Vytvořme si jednoduchý uživatelský typ `Point`:

```
struct Point {
  public double X, Y;
  public Point(double X, double Y) {
    this.X = X;
    this.Y = Y;
  }
}
```

Naplňme kolekci:

```
var points = new HashSet<Point>();
points.Add(new Point(1,1));
points.Add(new Point(1,2));
points.Add(new Point(1,1));
```

Prvky poté můžeme iterovat pomocí `foreach`:

```
foreach(Point point in points) {
  Console.WriteLine("[X={0}, Y={1}]", point.X, point.Y);
}
```

Všimněte si zástupných symbolů `{0}` a `{1}` (podobné Cčkovému `printf`). Řetězec takto můžeme vygenerovat pomocí `String.Format` (podobné Cčkovému `sprintf`). Pohodlněji můžeme objekty vypsat, pokud překryjeme metodu `ToString`:

```
struct Point {
  …
  public override string ToString() {
    return string.Format(("[X={0}, Y={1}]", X, Y);
  }
}
```

Všimněte si klíčového slova `override`, které má stejný význam, jako `virtual` v C++ v rodičovské třídě. V C# o překrývání rozhoduje potomek. Pokud bychom v C# místo toho chtěli metodu *skrýt* (ekvivalent běžné rodičovské metody v C++), nahradíme `override` slovem `new`.

Poté lze iterovat, jako by `Point`s byly řetězce:

```
foreach(Point point in points) Console.WriteLine(point);
```

Generické programování HashSet

Pokud však místo struktur použijeme třídu (u `Point` nahradíme slovo `struct` slovem `class`), kolekce přestane fungovat a v kolekci se objeví dva identické prvky. Třída se kopíruje odkazem, její klíč v množině ve výchozím stavu odpovídá adrese. Toto lze změnit překrytím metod `bool Equals` (výsledek porovnávání objektů) a `int GetHashCode` (klíč např. v `HashSet`):

```
public override bool Equals(object obj) {
  const double little = 0.0000000001;
  Point other = obj as Point;
  if(other==null) return false;
  return Math.Abs(X-other.X)<little && Math.Abs(Y-other.Y)<little;
}

public override int GetHashCode() {
  int hash = 23;
  unchecked {
    hash = hash*31 + X.GetHashCode();
    hash = hash*31 + Y.GetHashCode();
  }
  return hash;
}
```

V metodě `Equals` není vhodné provádět porovnávání dvou desetinných čísel, neboť jejich binární reprezentace obvykle nemá ukončený desetinný rozvoj a není přesná. V hashovací metodě je cílem vyhýbat se kolizím, v uvedeném algoritmu je místo 23 a 31 možné zvolit i jiná nesoudělná čísla, tato empiricky dávají dobré výsledky. Blok `unchecked` ignoruje přetečení datového typu.

Iterace Dictionary

Při iteraci pomocí `foreach` u objektu typu `Dictionary<K,V>` iterovaný typ `KeyValuePair<K,V>`, což jsou v podstatě dvě hodnoty, kde `K` je typ klíče a `V` typ hodnoty. Tento typ obsahuje vlastnosti `.Key` a `.Value`, např.:

```
var data = new Dictionary<char,int>();
data['A'] = 65;
data['B'] = 66;
data['C'] = 67;

foreach(var kv in data) Console.WriteLine("{0}={1}", kv.Key, kv.Value);
```