配列

配列とインスタンス

配列は汎用的なプログラムにおいて欠かせない重要な技術です
実用プログラムでは、実行時までインスタンスの数を把握できないケースがほとんどで
実行中に必要となるメモリを監視し、それを制御する必要があります
これを行うには、配列を作成しこれに各オブジェクトを格納して制御する方法をとります

C# の型は、全て .NET ライブラリが用意する構造体のエイリアスであることを説明しました
このことから、C# の配列は C 言語のような単純な配列とは異なります
実際には、C# 言語の配列は System.Array クラスを基底とするインスタンスを生成します

配列を宣言するには、**型の次に [] を指定します**
C++ は変数名の後に [] を指定しましたが、C# は型の後ろであることに注意してください
これで、**宣言する変数が配列**であるということを指定しす

しかし、この時点で配列は作られていません
先ほど説明したように、配列とは System.Array クラスのインスタンスなのです
配列を作成するには、クラス同様に new 演算子を用いてインスタンスを生成します
例えば、次は x 個の int 型の配列を宣言しています

int[] iValue = new int[x];

これで、iValue は System.Array を継承する配列のインスタンスとなっています
x は、配列の長さを整数型で指定します
配列にアクセスするには要素アクセス式を用いてアクセスします

variable[X , ...];

X にはアクセスする要素番号（添え字）を指定します
もし配列が多次元の場合は、カンマ , で区切ってそれぞれの次元配列を指定します
配列の要素にアクセスする時、CLR は配列が null ではないか、添え字が不正ではないかを調べ
不正なアクセスの場合は「例外」を発生させて処理を強制的に中断します

```
class Test {
 public static void Main() {
  string[] str = new string[3];
  str[0] = "Kitty on your lap";
  str[1] = "Silver Gene";
  str[2] = "Nekoneko Zoo";

  for (int i = 0 ; i < 3 ; i++)
   System.Console.WriteLine(str[i]);
 }
}
```

このプログラムでは string 型の配列 str を作成しています
この配列は 3 つの要素を持ち、それぞれに string 型の文字列を格納しています

配列の数は new string[x] で指定しています。ここで指定しているのは配列の数です
それに対して、要素アクセス式で指定する場合は 0 から数えた要素番号です
つまり、3つの要素を持つ str は str[0] , str[1] , str[2] の3つの空間を持っているのです

それぞれに格納した文字列を出力する時、このプログラムはループを使っています
このようにループによってまとまった処理ができるというのも、配列の魅力です

多次元配列

上のプログラムは、配列が1つの直列した空間を持つ「1次元配列」でした
配列は、さらに並列した配列空間である「n次元配列」にすることが可能です
このように、複数の空間を持つ配列を**多次元配列**と呼びます

一般に多次元配列は「2次元配列」となることが多いです
もちろんそれ以上の配列も可能ですが、複雑になり多くのメモリ空間を消費します

C# で多次元配列を実現するには**次元指定子にカンマ**をつけます
次元指定子とは、型の後に記述した [] のことを表します
次元指定子にカンマがない場合は1次元配列であり、その後1つのカンマにつき1次元増えるのです

表記	次元数	最初の要素へのアクセス方法
int[] value	int 型の1次元配列	value[0]
int[,] value	int 型の2次元配列	value[0 , 0]
int[,,] value	int 型の3次元配列	value[0 , 0 , 0]

これは従来の言語のような value[][] という多次元配列の指定方法とは違うので注意していください
value[][] というような指定は C# はで「ジャグ」と呼ばれる特殊な配列です（後記）

```
class Test {
 public static void Main() {
  string[,] str = new string[2 , 3];
```

```
  str[0 , 0] = "Rena";
  str[0 , 1] = "Yuki";
  str[0 , 2] = "Mimi";

  str[1 , 0] = "Di Gi Charat";
  str[1 , 1] = "Petit Charat";
  str[1 , 2] = "Rabi en Rose";

  for (int i = 0 ; i < 2 ; i++)
   for (int j = 0 ; j < 3 ; j++)
    System.Console.WriteLine(str[i , j]);
 }
}
```

これは、C# による2次元配列の例です
2次元配列を使えば、2次元的な性質を持つ要素をプログラムで表現できます
このような多次元構造を持つ情報は、実社会に多く存在しています

配列の初期化

配列は、通常 new キーワードを使ってインスタンスを生成し
生成した配列に対して初期化を行います
しかし、インスタンスの生成と同時に初期化してしまう方法もあります
配列生成時に要素が決定している場合、こちらの方が効率的です

配列を生成と同時に初期化するには**配列初期化子**を用います
配列初期化子は { と } に囲まれ , で区切られます
例えば、次の文は int 型の配列を配列初期化子で生成しています

int[] iValue = { 1 , 2 , 4 , 8 };
int[] iValule = new int[] { 1 , 2 , 4 , 8 };

これらの場合、インスタンスの長さは初期化子で指定した式の数に調整されます
配列は、インデックス 0 番から昇順に格納されていきます
つまり、上の配列初期化子は次の文と同じ処理を行っています

int[] iValue = new int[4];
iValue[0] = 1; iValue[1] = 2; iValue[2] = 4; iValue[3] = 8;

これを使えば、あらかじめ格納する要素が決定している場合
何行も初期化のために代入演算を行う手間を省くことができます

```
class Test {
 public static void Main() {
  string[] str = new string[] {
   "Kitty on your lap",
   "Silver Gene",
   "Nekoneko Zoo"
  };

  for (int i = 0 ; i < 3 ; i++)
   System.Console.WriteLine(str[i]);
 }
}
```

この場合 str は、指定した3つの文字列を生成と同時に格納しています

多次元配列の場合は、{ } を配列と同じレベルでネストします
もっとも外側の { } は一番左の次元で、もっとも内側の { } は一番右側の次元です

```
class Test {
 public static void Main() {
  string[,] str = {
   {"Rena" , "Yuki" , "Mimi"} ,
   {"Di Gi Charat" , "Petit Charat" , "Rabi en Rose"} ,
  };

  for (int i = 0 ; i < 2 ; i++)
   for (int j = 0 ; j < 3 ; j++)
    System.Console.WriteLine(str[i , j]);
 }
}
```

この原理がわかれば、どんな多次元配列の初期化も可能です
