C#の基礎文法

suzusime

平成 28 年 4 月 23 日

はじめに

KMC へようこそ！ この文書は、新入生プロジェクト「C#でゲームを作ろう 2016」の補助資料です。この文書にプログラミング言語 C#の基本的なことをまとめました。これから C#でプログラムを書いて行くにあたって、分からなくなったらいつでも読み返してください。また、よく分からないことがあったら、私や他の KMC 部員に聞いて下さい。では、皆さん、めくるめくプログラミングの世界への旅を楽しみましょう。

1　変数と流れ(フロー)制御

1.1　はじめてのプログラム

```
class Test {
      public static void Main(){
            System.Console.WriteLine("Hello,␣world!");
      }
}
```

まず、ここでは 3 行目だけに注目します。`System.Console.WriteLine();` は、「括弧の中の文字列を画面に出力せよ」という命令文です。ここでは “Hello, world!” という文字列が画面に出力されます。

他の部分はとりあえず今は考えないでおきましょう。おいおい説明します。今覚えておくべきは `Main()` のあとの中括弧 {} の中に命令文を並べていくと、それが実行されるということだけです。以下では断りなく `Main()` の中だけを示すことが多くなります。

```
System.Console.WriteLine("Hello,␣world!");
System.Console.WriteLine("はにゃーん");
```

このように 2 行並べれば、“Hello, world” のあとに改行されて “はにゃーん” と出力されます。プログラムの命令文は基本的に上から順番に実行されていきます。

1.2　変数

```
int a;
a=28;
System.Console.WriteLine(a);
```

1 行目は「int という型の a という名前の変数をこれから使います」という**宣言**です。**変数**とは数学で言う x や y のような、なにかを入れておく箱のようなものです。数学でベクトルを入れる箱 $\vec{v}$ や函数を入れる箱 f があるように、変数にも何を入れるものかという種類があります。それを**型**といいます。int は整数を入れる型です。また、変数に付ける名前はここでは a ですが、ここは基本的になんでも好きなように付けることが出来ます（ただし、特別な意味をもつ文字などは使えません。詳しくは「識別子に使える文字」などで検索してください）。分かりやすい名前を付けましょう。2 行目は「a に 28 を代入する」という命令です。“=” は代入を表します。

代入演算子 “=” はあくまで「左の変数に右の数の値を代入する」という意味であって、等号ではないということに注意してください。

```
int a, b;
a=28;
b=a;
a=a-17;
```

このプログラムの実行後は $a = 11, b = 28$ となります。

主な型には次のようなものがあります。

整数型	int
実数型	float
倍精度実数型	double
文字列型	string
真理値型	bool

また、宣言と代入は同時に行うこともできます。

```
int a=28;
float r=1.44f; //float 型に代入するときは数字の後に f をつけます
double r2=1.44;
string test="キルミーベイベー"; //string 型に代入するときは文字列を""で囲みます
```

1.3 演算子

int 型などの整数型に対する**演算子**には次のようなものがあります。実数型もこれに準じます。

種類	記号	例（int a=28 とする）	結果（b に代入される値）
加算	+	int b=a+3;	31
減算	-	int b=a-3;	25
乗算	*	int b=a*5;	140
除算	/	int b=a/5;	5
剰余	%	int b=a%5;	3
インクリメント (後置)	++	a++;	a に 29 が代入される
デクリメント (後置)	--	a--;	a に 27 が代入される
符号反転	-	int b=-a;	-28
等しい	==	bool b=a==3;	false
等しくない	!=	bool b=a!=3;	true
より大きい	>	bool b=a>3;	true
以上	>=	bool b=a>=3;	true
より小さい	<	bool b=a<3;	false
以下	<=	bool b=a<=3;	false

string 型の場合には次のようなものしかありません。但し、代入演算子 “=” は文字列でも（それに限らず全ての型で）使えます。

種類	記号	例（string a="ゆゆ式"とする）	結果（b に代入される値）
文字列結合	+	b=a+"は最高";	"ゆゆ式は最高"

bool 型の場合は次のようなものがあります。

種類	記号	例（a=true,b=false とする）	結果（c に代入される値）
論理積（かつ）	&&	bool c=a&&b;	false
論理和（または）	||	bool c=a||b;	true
排他的論理和（どちらか一方だけ真であるか）	^	bool c=a^b;	true
論理否定	!	bool c=!a;	false

1.4 条件分岐

条件分岐は if-else 構文を用いて行います。

```
int a=2;
if(a<0){
      System.Console.WriteLine("負の数です");
} else if(a<10){
      System.Console.WriteLine("1桁です");
} else {
      System.Console.WriteLine("2桁以上です");
}
```

この場合、$a = 2$ なので、$a < 0$ の真理値は偽となり、1 つめの if の後のブロック（= {} に囲まれた部分）は無視され、次の else の先を実行します。else の次には if があるので、そこでまた評価が行われ、今度は $a < 10$ が真になるため、ブロックの中が実行されて “1 桁です” と表示されます。

1.5 繰り返し（ループ）

while 構文を用いると繰り返しを表現できます。

```
int i=0;
while(i<5){
      System.Console.WriteLine(i);
      i++;
}
```

このコードを実行すると 0 から 4 までを出力します。while に辿り着いたときは、if と同じように評価が行われ、今回は $i < 5$ が真なのでブロックの中身が実行されます。if と異なるのは、ブロックの中の処理が終了した後、最初に戻ってまた評価がなされるということです。ブロックの中で i++; が実行されていますので、2 回目の i の値は 1 になっています。このときは $i < 5$ は真ですから、ブロックの中身がまた実行されます。これを 5 回目まで繰り返すと i の値が 5 になるので、最初に戻ったときに $i < 5$ が偽となり、ブロックの中身は実行されず、次へ行くことになります。

continue 文や break 文を用いると、繰り返しをもう少し柔軟に操作することが出来ます。

```
int i=0;
while(i<8){
      System.Console.WriteLine(i);
      if(i==1){ i=i+3; continue; }
      if(i==6){ break; }
      i++;
}
```

このコードを実行すると、0,1,4,5,6 を出力します。continue 文は、そこに来た時点でブロックの中の処理が終了したことにして while の最初に戻るという命令です。戻った後に再び括弧の中の値の真偽が評価されます。対して break 文はそこでブロック内の処理を終了したことにするのは同じですが、while の最初に戻ることはなく、そのまま次に進みます。

練習 1（易） 1 から 100 までの数字のうち、3 でわりきれるものだけを出力するプログラムを書け。

練習 2（標準） 1 行目に “Working”、2 行目に “Working!”、3 行目に “Working!!”、……とひとつずつ感嘆符を増やしていき、30 行出力するプログラムを書け。

1.6 標準入出力

出力には `System.Console.WriteLine()`（改行あり）または `System.Console.Write()`（改行無し）、入力には `System.Console.ReadLine()` を使います。

```
string str;
str = System.Console.ReadLine(); //"うさぎ"と入力すると
System.Console.WriteLine("ご注文は"+str+"ですか？ "); //"ご注文はうさぎですか？ "と出力される
```

`System.Console.ReadLine()` の結果は文字列なので、整数を受け取りたいときには `int.Parse()` を用いて文字列を整数に変換します。

```
string str;
str = System.Console.ReadLine();//4を入力すると
int num=int.Parse(str);
System.Console.WriteLine(num*num);//16が出力される
```

練習 3（易） 入力された整数 x 以上の 8 の倍数のうち、最小のものを出力するプログラムを書け。

練習 4（標準） 入力された 2 以上の整数 x を素因数分解するプログラムを書け。但し x は int 型に収まる範囲（約 21 億以下）とする。

練習 5（標準） 入力された整数 x 以上が 3 の倍数であるか、10 進数で表したときにどこか 1 つ以上の桁が 3 である場合にだけ “あたいったら天才ね！ ” と出力するプログラムを書け。

1.7 参考サイト

「C# によるプログラミング入門｜ ++C++; // 未確認飛行 C」（`http://ufcpp.net/study/csharp/`）がお勧めです。今回の資料では省略した C#の詳しい文法事項も分かりやすく解説されています。

2 函数

2.1 函数の定義

数学でいう函数と同じようなものをプログラムでも用いることができます。

```
class Test {
      static int max(int a, int b){
            if(a<b){ return b; }
            else { return a; }
      }
      public static void Main(){
            int x = int.Parse(System.Console.ReadLine());
            int y = int.Parse(System.Console.ReadLine());
            int z = max(x,y);
            System.Console.WriteLine(z);
      }
}
```

`static int max(int a, int b){}` は、int 型の変数 a と int 型の変数 b の 2 つの変数をとり、int 型の値を結果として返す函数です。数学っぽく書くと $\max : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ ですね。ここで、a、b のような函数が外部から受け取るものを**引数**（ひきすう）、返す値を**返値**（かえりち）といいます。今回は引数も返値も全て整数ですが、どの型の値を何個変数にとって何の値を

返すかは函数を定義する人の自由です。ただし、返値の数は 1 つだけです。また、返値が不要の場合は、返値の型として void を指定します。なお、C#の文脈では、函数のことを**メソッド**と呼ぶことがあります。

函数 `int max(int,int)` は次のように作られています。まず、第 1 引数を変数 a に、第 2 引数を変数 b に代入します。これらの変数は、函数の内部でのみ自由に使うことができます。そして b が a より大きいときは `return b;` で返値として b を返し、それ以外の場合は a を返しています。

練習 6（易） 引数に 1 つの文字列 s と 1 つの整数 n をとり、s を n 回繰り返したあとに「!」を付加した文字列を返す函数を書け。

練習 7（難） 引数として 2 つの整数をとり、その 2 整数の最大公約数と返す函数を書け。ただし、引数のどちらかが負の数であったときは-1 を返すようにせよ。［ヒント：再帰函数で検索すると良いかも］

2.2 値渡しと参照渡し

```
class Test {
      static int inc(int a){
            a=a+1;
            return a;
      }
      public static void Main(){
            int x = 6;
            int y = inc(x);
            System.Console.WriteLine(x.ToString() + "␣" + y.ToString());
      }
}
```

このプログラムで x と y は何と出力されるでしょう？ 実は、$x = 6, y = 7$ になります。函数 `int inc(int)` の中では `a=a+1;` とされていますが、その a の値の元であるはずの x には 1 が足されていません。このような振る舞いをする理由は、この函数の最初で `int a=x;` と書かれていると考えれば分かります。a は x とはあくまで別の変数であり、最初に x の値が代入されただけなのです。このような引数の渡し方を**値渡し**といいます。

一方、次のように書くとどうなるでしょう。

```
class Test {
      static int inc(ref int a){
            a=a+1;
            return a;
      }
      public static void Main(){
            int x = 6;
            int y = inc(ref x);
            System.Console.WriteLine(x.ToString() + "␣" + y.ToString());
      }
}
```

今度は $x = 7, y = 7$ になります。このように引数の前に ref を付けて宣言すると、外部から渡された引数の変数そのものが内部で使われる引数になります。つまり、この場合は a は x そのものです。なので `a=a+1;` は `x=x+1;` が実行されたのと同じようになり、x の値も変化することになります。このような引数の渡し方を**参照渡し**といいます。

3 データ構造

3.1 クラス

クラスは自分が作る新しい型です。

```
class Enemy {
      public int HP;
```

```
        public int MP;
}

class Test {
        public static void Main(){
                Enemy test1;
                test1 = new Enemy();
                test1.HP = 100; test1.MP = 10;
        }
}
```

`class Enemy{}` のところで、Enemy 型を定義しています。ここでは「int 型の HP と MP を持つ型です」と宣言しています。`Enemy test1;` では Enemy 型の変数を宣言しています。`test1 = new Enemy();` は**初期化**という処理です。これで、新しく作られた Enemy 型の値が test1 に入ります（*cf.* 参照型）。クラスの型の変数（ここでいう test1）を特に**実体**（インスタンス）と呼びます。test1.HP は、test1 の中の変数 HP を指します。ちょうど、ベクトルの一つの成分を取り出しているようなものです。この HP のように、クラスの中にある変数を**メンバ変数**と言います。

3.2 値型と参照型

```
class Test {
        static void damage(Enemy e){
                e.HP = e.HP - 5;
        }
        public static void Main(){
                Enemy test1 = new Enemy();
                test1.HP=100;
                damage(test1);
                System.Console.WriteLine(test1.HP);
        }
}
```

これを実行すると、test1.HP は 95 になります。つまり Li 参照渡しをした訳ではないのに、まるで参照渡しをしたかのような振る舞いをします。これは Enemy 型が**参照型**であるからです。対して今まで出てきた int などの「普通の」型を**値型**と言います。値型の変数は実体そのものを持っているのに対し、参照型の変数は実体のある場所（メモリアドレス）の情報を持っています（他の言語で言うポインタにあたります）。参照型の変数の場合、函数の値渡しのときにコピーされるのはアドレスの値なので、内部の変数（ここでいう e）は外部から渡される変数（test1）とは別のものではありますが、両方とも同じ実体のアドレスを持っているので、内部でその変数に行った変更もきちんと外部に反映されます。

`Enemy test1;` のように参照型の変数を宣言した時点では実体は生成されません。宣言した後に初期化する際に初めて実体が生成されます（初期化のときに new という文を使うのは、新しい実体を生成するという意味です）。それ以前は “null”（どこにもない）という値を持っているので、もし初期化せずに test1 のメンバ変数を使おうとすると “NullReferenceExeption” という**例外**（エラーのようなもの）がでてプログラムが落ちてしまいます。

今までにでてきた int や double、bool などの基本的な型は値型です。string は参照型ですが、代入演算子を使ったときに新しい実体が作られるなど、値型のような振る舞いをします。クラスは参照型なので注意して下さい。

3.3 配列

```
Enemy[] enemies;
enemies = new Enemy[3];
for(int i=0; i<3; i++){
        enemies[i] = new Enemy();
        enemies[i].HP = 50;
}
```

配列は同じ型をいくつかまとめて作った新しい型です。[i] を使うことで、その i 番の成分を取り出して扱うことが出来ます。0 が始まりであることに注意して下さい。

3.4 List<>クラス

配列は初期化するときに長さを決めなければなりません。が、それだと不便なので、必要になったときに長さを勝手に伸ばしてくれると便利です。そういうときは List<>クラスを使います。

```
List<Enemy> enemies;
for(int i=0; i<10; i++){
      Enemy temp = new Enemy();
      temp.HP=50;
      enemies.Add(temp);
}
```

List<Enemy>は Enemy 型を中に入れる List<>型です。<>は**雛型**（テンプレート）**クラス**（**総称**（ジェネリック）**クラス**）の記法ですが、ここでは深く触れません。

このような便利なデータ構造を実現するクラスを**コンテナ**と言います。C#では他にも Dictionary や Queue、Stack などのコンテナがあるので、目的とするデータ構造に応じて使い分けると便利です。

4 クラスの各種機能

4.1 メンバ函数

```
class Character {
      private string name;
      public string GetName(){
            return name;
      }
      public Character(string name){
            this.name = name;
      }
      public static void Hello(){
            System.Console.WriteLine("この限りなく無意味な函数");
      }
}

class Test {
      public static void Main(){
            Character yasuna = new Character("折部やすな");
            Character akari = new Character("赤座あかり");
            System.Console.WriteLine(yasuna.GetName());
            System.Console.WriteLine(akari.GetName());
            Character.Hello();
      }
}
```

クラスには先述したメンバ変数の他に、**メンバ函数**（C#ではよくメソッドと言います）を含ませることができます。メンバ函数の利点は、その実体の中に含まれている情報を使った処理ができることです。今回の場合、GetName() 函数はその実体のもつメンバ変数 name を返す函数になっています。

クラスの名前と同じで返値がない函数 Character(string) がありますが、これは**構築函数**（コンストラクタ）と呼ばれる特殊なメンバ函数です。この函数は、実体の初期化のときに呼ばれます。new Character("折部やすな") というところがありますが、ここで引数に渡したものが、Character(string name) の name になっているのです。一般に構築函数は、クラスの実体を生成する際に各メンバ変数を初期化するために用います。

public や private というのは、そのメンバ変数やメンバ函数を呼び出せる範囲を制限するためのものです。public がついたものは外から呼び出せますが、private のついたものはそのクラスの中からしか呼び出せません。

クラスのメンバ函数、メンバ変数に static をつけて定義すると、それらは**静的な**メンバ変数、メンバ函数になります。これらはクラスの実体ではなくクラス全体で共有される変数、函数です。呼び出すときも実体名の後ろではなくクラス名の後ろに . をつけます。

練習 8（易） 3 次元実数ベクトルを表すクラスをつくれ。メンバ函数にそのベクトルの長さを返す函数をもたせよ。また、静的メンバ函数として 2 つのベクトルを引数にとり、それらの内積を返す函数、及びそれらの外積を返す函数をもたせよ。

練習 9（標準） トランプのカードをランダムに配るプログラムを書け。配る人数を入力として受け取り、その人数にできる限り枚数が等しくなるように配ったときのそれぞれのプレイヤーの手札一覧を出力すると良い。なお、ランダムな処理には乱数というものを使う。C#での乱数の扱い方は各自調べてほしい。

ヒント（実装の一例）：カードを表すクラスとプレイヤーを表すクラスを作る。プレイヤーには手札を示すカードの List をもたせる。Main() 函数の中では入力された数字に合わせた数のプレイヤーを入れる List を作る。そのプレイヤーを入れた List でループを回して全部のカードをランダムにうまく配る。

練習 10（難） 簡単な戦闘シュミレーションを作ってみよ。

例えば次のようなモデルを考える。2 人のキャラクターがいて、それぞれ体力と攻撃力をもっている。お互いは交互に攻撃し、相手の体力に攻撃力ぶんだけダメージを与える。ただし、確率で回避することができ、その確率は体力が多いほど上がる。これを繰り返しどちらかの体力が 0 未満になったところで終了する。

あるいは次のようなモデルを考える。2 つの国の軍隊が戦っている。両軍の最初の兵士の数と武器の性能を決める。両軍は同時に攻撃しあう。最初、1 時間経過後、2 時間経過後、……というふうに時間を区切って考え、時間が進む際にその前の時点での相手の軍隊の残存兵士数と武器の性能を掛けたぶんだけの兵士を失うとする。これを続けていきどちらかの兵士が一定数以下になった時点で終了とする。このとき、初期値を色々変えて、武器の性能と兵士の数がそれぞれどれほど勝敗に影響するかを確かめてみると面白いかもしれない（*cf.* ランチェスターの法則）。

5 補遺

5.1 名前空間

```
using System;

namespace Hoge {
      class Piyo{
            public int a;
      }
}

class Test{
      public static void Main(){
            Hoge.Piyo fuga= new Hoge.Piyo();
            Console.WriteLine("あ＾〜␣こころがぴょんぴょんするんじゃぁ＾〜〜〜");
      }
}
```

このように namespace で囲った中でクラスを定義すると、**名前空間**の中にあるものとして周りからは区別され、扱う際には名前空間名 . クラス名のように指定しないといけなくなります。これをいちいち書かなくて良いようにするためには using 文を使います。例えば、今まで散々使ってきた System.Console.WriteLine() 函数は、実は System 名前空間の中の Console というクラスの静的メンバ函数 WriteLine() だったので、`using System;` と書くことで、System. を省くことが出来るようになります。