講義メモ
・p.185「フィボナッチ数列を求める」から

p.181 階乗を計算する(再掲載)

・階乗とは、ある正の整数において、その数から1までの全整数の積
　※ 実質的にはその数から2までの全整数の積になる
・整数nの階乗を「n!」で表し、例えば、2!は2、3!は6、4!は24、5!は120、…
・これを逆順で展開すると
```
 5! = 5×4×3×2×1
 4! =    4×3×2×1
 3! =       3×2×1
```
・なので、5!=5×4!、4!=4×3! であることから「n! = n×(n-1)!」が導ける。
・これをメソッドFactにすると、
```
 int CalcFact(int n) { //nの階乗
  return n * CalcFact(n - 1); //n×(n-1)!を返す(再帰する)
 }
```
・これに、再帰の終了条件として「0の階乗は1」を加えると良い
```
 int CalcFact(int n) { //nの階乗
  return (n > 0) ? n * CalcFact(n - 1) : 1; //nが0超ならn×(n-1)!を返す(再帰する)でなければ1を返す
 }
```
・p.182 fact01.csは上記を展開したメソッドになっている

提出フォロー:アレンジ演習:p.182 fact01.cs

・CalcFact(int n)メソッドを上記を用いてシンプルにしよう

作成例

```
//アレンジ演習:p.182 fact01.cs
using System;
class Fact {
    public long CalcFact(int n) { //n!を返す
        return (n > 0) ? n * CalcFact(n - 1) : 1; //nが0超ならn×(n-1)!を返す(再帰する)でなければ1を返す
    }
}
class fact01 {
    public static void Main() {
        Fact f = new Fact();
        for (int i = 0; i <= 20; i++) {
            Console.WriteLine("{0}! = {1}", i, f.CalcFact(i));
        }
    }
}
```

p.185 フィボナッチ数列を求める

・フィボナッチ数列は統計やシミュレーションなどに用いる増加速度をもつ数字の並び。
・先頭2値が1で、それ以降は前2値の和になる
・実例: 1,1,2,3,5,8,13,21,34,55,…
・よって、n番目のフィボナッチ数はn-1番目とn-2番目の和として、再帰で表現できる

p.185 fibonacci.cs

```
//p.185 fibonacci.cs
using System;
class fibo {
    public long CalcFibo(int n) { //フィボナッチ数列のn番目を返す
        long fb;
        if (n == 1 || n == 2) { //先頭2要素は1固定(再帰の終了条件でもある)
            fb = 1;
        } else {
            fb = CalcFibo(n - 1) + CalcFibo(n - 2); //3値目以降は前2値の和
        }
        return fb;
    }
}
class fibonacci {
    public static void Main() {
        fibo f = new fibo();
        for (int i = 1; i <= 30; i++) {
            Console.WriteLine("f({0}) = {1}", i, f.CalcFibo(i));
        }
    }
}
```

アレンジ演習:p.185 fibonacci.cs

・条件演算子を用いて記述をシンプルにしよう

作成例

```
//アレンジ演習:p.185 fibonacci.cs
using System;
class fibo {
    public long CalcFibo(int n) { //フィボナッチ数列のn番目を返す
        return  (n == 1 || n == 2) ? 1 : CalcFibo(n - 1) + CalcFibo(n - 2); //3
値目以降は前2値の和
    }
}
class fibonacci {
    public static void Main() {
        fibo f = new fibo();
        for (int i = 1; i <= 30; i++) {
            Console.WriteLine("f({0}) = {1}", i, f.CalcFibo(i));
        }
    }
}
```

p.188 refとout(値渡しと参照渡し)

・C#においてメソッドの引数は値渡しが基本であり、値のコピーが行われる
・よって、メソッド内で引数を変更しても、呼び出し側の指定した引数の値は変わらない
・p.189 swap01.csが失敗例で、Mainメソッドで定義して引数に指定しているxとyは、Swap関数の仮引数xとyとは(たとえ同名でも)無関係
・C/C++では参照型の場合に参照渡しによりこれを解決できるが、C#では参照型でも値渡しになる

・なお、配列などのようにデータ構造を渡した場合は、構造の専用位置を示す値が値渡しされるので、受け取った側で同じオブジェクトを用いるため、参照渡しになる

p.191 charngearray01.cs

```
//p.191 charngearray01.cs
using System;
class change {
    public void modify(int[] array) { //引数が配列なので実質的に参照渡し
        int n = array.Length; //要素数を得て
        for (int i = 0; i < n; i++) { //全要素について繰返す
            array[i] *= 2; //要素値を2倍にする
        }
    }
}
class changearray01 {
    public static void Main(){
        change c = new change();
        int[] myarray = new int[3]{1, 2, 3};
        Console.WriteLine("----modifyメソッド実行前----");
        int i = 0;
        foreach (int x in myarray) { //配列myarrayの全要素について繰返す
            Console.WriteLine("myarray[{0}] = {1}", i, x); //値を表示
            i++;
        }
        c.modify(myarray); //要素値を2倍にする
        Console.WriteLine("----modifyメソッド実行後----");
        i = 0;
        foreach (int x in myarray) { //配列myarrayの全要素について繰返す
            Console.WriteLine("myarray[{0}] = {1}", i, x);
            i++;
        }
    }
}
```

p.192 refキーワード

・値渡しとなる引数と仮引数の両方に「ref」を前置すると、参照渡しに変更される
・なお、この仮引数は事前に初期化が必要

p.193 swap03.cs

```
//p.193 swap03.cs
using System;
class myclass {
    private int temp;
    public void swap(ref int x, ref int y) { //仮引数xとyは参照渡しにする
        temp = x;
        x = y;    //ここでxを書き換えた結果が引数に反映する
        y = temp; //ここでyを書き換えた結果が引数に反映する
    }
}
class swap03 {
    public static void Main() {
```

```
        myclass s = new myclass();
        int x = 10, y = 20;  //引数用の変数を初期化
        s.swap(ref x, ref y); //参照渡しにより、呼び出す
        Console.WriteLine("x = {0}, y = {1}", x, y);
    }
}
```

p.194（outキーワード）

※ テキストではrefの代わりに無条件にoutが利用できるような説明になっているが、制限もある
・refに代わりにoutを指定すると、事前の初期化が不要になる
・ただし、out指定の仮引数は初期化されていないことから、代入の右辺には記述できない
・よって、p.193 swap03.csの「ref」を「out」にするとエラーになる
・また、古いバージョンのC#では利用できない

p.194 outkeyword01.cs

```
//p.194 outkeyword01.cs
using System;
class MyClass{
    public void Square(double x, double y, out double s) { //仮引数sは参照渡し
        s = x * y; //よってsをreturnする必要はない
    }
}
class outkeyword01 {
    public static void Main() {
        double a = 125.3, b = 16.25, c;
        MyClass mc = new MyClass();
        //cには値を代入していません
        mc.Square(a, b, out c); //初期化していないcを参照渡しできる
        Console.WriteLine("縦{0}m, 横{1}mの長方形の面積は{2}平方メートル", a, b, c);
    }
}
```

補足：p.194 outkeyword01.csについて

・Squareメソッドは1値を返すようになっているので、outは必須ではない。下記で可能。

```
    public double Square(double x, double y) {
        return x * y;
    }
```

アレンジ演習：p.194 outkeyword01.cs

・引数xとyの和と積を返すメソッド public void AddMul(double x, double y, out double add, out double mul) にしよう

作成例

```
//アレンジ演習：p.194 outkeyword01.cs
using System;
class MyClass{
    public void AddMul(double x, double y, out double add, out double mul) { //仮引数add,mulは参照渡し
        add = x + y; //よってreturnする必要はない
```

```
        mul = x * y; //よってreturnする必要はない
    }
}
class outkeyword01 {
    public static void Main() {
        double a = 125.3, b = 16.25, c, d;
        MyClass mc = new MyClass();
        mc.AddMul(a, b, out c, out d); //初期化していないc、dを参照渡しできる
        Console.WriteLine("和は{0}, 積は{1}", c, d);
    }
}
```

p.195 メソッドのオーバーロード

・p.168で説明のとおり、引数の型や数が異なるコンストラクタを記述できることをオーバーロード
・これは、メソッドでも可能なので、同じ意味の処理を行うメソッドは同じ名前にすると良い
　※ オーバーロードができないC言語では関数名が長くなるというデメリットがある
・コンストラクタと同様にシグニチャが異なればOKだが、メソッドの戻り値型はシグニチャに含まれないので注意
・これは戻り値型だけが異なるメソッドは呼び出し時に区別できないため

p.195 overload01.cs

```
//p.195 overload01.cs
using System;
class manymethods {
    public int Method(int x) { //メソッド①
        Console.WriteLine("第1のバージョンが呼ばれました");
        return x + 10;
    }
    public double Method(double x) { //メソッド②
        Console.WriteLine("第2のバージョンが呼ばれました");
        return x * 2;
    }
    public string Method(string x) { //メソッド③
        Console.WriteLine("第3のバージョンが呼ばれました");
        return x += "です";
    }
    public int Method(int x, int y) { //メソッド④
        Console.WriteLine("第4のバージョンか呼ばれました");
        return x + y;
    }
}
class overload01 {
    public static void Main() {
        manymethods m = new manymethods();
        Console.WriteLine("その戻り値は「{0}」です", m.Method(3)); //①を呼ぶ
        Console.WriteLine("その戻り値は「{0}」です", m.Method(3.2)); //②を呼ぶ
        Console.WriteLine("その戻り値は「{0}」です", m.Method("粂井")); //③を呼ぶ
        Console.WriteLine("その戻り値は「{0}」です", m.Method(5, 6)); //④を呼ぶ
    }
}
```

アレンジ演習：p.195 overload01.cs

・下記のシグニチャを持ち、最大値を返すMaxメソッドのオーバーロードに書き換えよう
① int Max(int, int)
② int Max(int, int, int)
③ double Max(double, double, double, double)
④ double Max(double[])

作成例

```
//アレンジ演習：p.195 overload01.cs
using System;
class manyMaxs {
    public int Max(int x, int y) { //メソッド①
        return (x > y) ? x : y; //大きい方を返す
    }
    public int Max(int x, int y, int z) { //メソッド②
        return (x > y) ? ((x > z) ? x: z) : ((y > z) ? y : z); //最大値を返す
    }
    public double Max(double x, double y, double z, double w) { //メソッド③
        double max1 = (x > y) ? x : y; //xとyの大きい方を得る
        double max2 = (z > w) ? z : w; //zとwの大きい方を得る
        return (max1 > max2) ? max1 : max2; //上記2者の大きい方を返す

    }
    public double Max(double[] d) { //メソッド④
        double max = double.MinValue;
        foreach (var w in d) { //全要素について作業変数wに取り出しながら繰返す
            if (w > max) {
                max = w; //最大値更新
            }
        }
        return max;
    }
}
class overload01 {
    public static void Main() {
        manyMaxs m = new manyMaxs();
        Console.WriteLine("max(3, 5)：{0}", m.Max(3, 5)); //①を呼ぶ
        Console.WriteLine("max(3, 7, 1)：{0}", m.Max(3, 7, 1)); //②を呼ぶ
        Console.WriteLine("max(3.14, 5.25, 7.79, -2.1)：{0}", m.Max(3.14, 5.25,
7.79, -2.1)); //③を呼ぶ
        double[] da = {3.14, 5.25, 7.79, -2.1, 9.25};
        Console.WriteLine("max(3.14, 5.25, 7.79, -2.1, 9.25)：{0}", m.Max(da));
//④を呼ぶ
    }
}
```

アレンジ演習：p.195 overload01.cs

・最大値を求める処理を1つにしよう
・①②③の引数をdouble型の配列に格納して④を呼ぶと良い

作成例

```
//アレンジ演習：p.195 overload01.cs
```

```
using System;
class manyMaxs {
    public int Max(int x, int y) { //メソッド①
        double[] wa = {x, y}; //引数を配列化する
        return (int)Max(wa); //メソッド④を呼ぶ
    }
    public int Max(int x, int y, int z) { //メソッド②
        double[] wa = {x, y, z}; //引数を配列化する
        return (int)Max(wa); //メソッド④を呼ぶ
    }
    public double Max(double x, double y, double z, double w) { //メソッド③
        double[] wa = {x, y, z, w}; //引数を配列化する
        return Max(wa); //メソッド④を呼ぶ
    }
    public double Max(double[] d) { //メソッド④
        double max = double.MinValue;
        foreach (var w in d) { //全要素について作業変数wに取り出しながら繰返す
            if (w > max) {
                max = w; //最大値更新
            }
        }
        return max;
    }
}
class overload01 {
    public static void Main() {
        manyMaxs m = new manyMaxs();
        Console.WriteLine("max(3, 5):{0}", m.Max(3, 5)); //①を呼ぶ
        Console.WriteLine("max(3, 7, 1):{0}", m.Max(3, 7, 1)); //②を呼ぶ
        Console.WriteLine("max(3.14, 5.25, 7.79, -2.1):{0}", m.Max(3.14, 5.25,
7.79, -2.1)); //③を呼ぶ
        double[] da = {3.14, 5.25, 7.79, -2.1, 9.25};
        Console.WriteLine("max(3.14, 5.25, 7.79, -2.1, 9.25):{0}", m.Max(da));
//④を呼ぶ
    }
}
```

別解（メソッド①②③の中身を1行に）

```
//アレンジ演習:p.195 overload01.cs
using System;
class manyMaxs {
    public int Max(int x, int y) { //メソッド①
        return (int)Max(new double[]{x, y}); //メソッド④を呼ぶ
    }
    public int Max(int x, int y, int z) { //メソッド②
        return (int)Max(new double[]{x, y, z}); //メソッド④を呼ぶ
    }
    public double Max(double x, double y, double z, double w) { //メソッド③
        return Max(new double[]{x, y, z, w}); //メソッド④を呼ぶ
    }
    public double Max(double[] d) { //メソッド④
        double max = double.MinValue;
        foreach (var w in d) { //全要素について作業変数wに取り出しながら繰返す
```

```
            if (w > max) {
                max = w; //最大値更新
            }
        }
        return max;
    }
}
class overload01 {
    public static void Main() {
        manyMaxs m = new manyMaxs();
        Console.WriteLine("max(3, 5):{0}", m.Max(3, 5)); //①を呼ぶ
        Console.WriteLine("max(3, 7, 1):{0}", m.Max(3, 7, 1)); //②を呼ぶ
        Console.WriteLine("max(3.14, 5.25, 7.79, -2.1):{0}", m.Max(3.14, 5.25,
7.79, -2.1)); //③を呼ぶ
        double[] da = {3.14, 5.25, 7.79, -2.1, 9.25};
        Console.WriteLine("max(3.14, 5.25, 7.79, -2.1, 9.25):{0}", m.Max(da));
//④を呼ぶ
    }
}
```

p.197 Mainメソッドのオーバーロード

・Mainメソッドは特殊なメソッドであり、プログラムの動作開始位置の指定を行う
・Mainメソッドはpublic staticであり、戻り値はvoidまたはint、引数は無しまたは文字列配列であれば良い
・戻り値をintにすると、プログラムの呼び出し側に整数値を返すことができる
・プログラムの呼び出し側がOSの場合は正常終了した0を、でなければ0以外をreturnすると良い
・この値はシステム変数「errorlebel」で参照できるので、直後にコマンド「echo %errorlebel%」を実行すると得られる
・戻り値を文字列配列にすると、呼び出し時に0個以上の文字列を与えることができ、メソッド内で受け取って利用できる
・この文字列をコマンドライン引数といい、Visual Studioではデバッグのプロパティで指定可能
・コマンドプロンプトなどの外部処理からプログラムを実行可能であり、この時はプログラム名に続けてコマンドライン引数を指定可能
・なお、コマンドライン引数の数は不定なので、Lengthプロパティで要素数を得て用いると良い

p.198 main01.cs

```
//p.198 main01.cs
using System;
class main01 {
    public static int Main(string[] s) { //コマンドライン引数を受け取り整数を返す
        int n;
        n = s.Length; //コマンドライン引数の数を得る
        Console.WriteLine("引数の個数は{0}個です", n);
        if (n != 0) { //コマンドライン引数が指定されている？
            for (int i = 0; i < n; i++) { //全引数について繰返す
                Console.WriteLine("引数{0} : {1}", i + 1, s[i]);
            }
        }
        return 0; //正常終了を返す
    }
}
```

補足：コマンドラインからの実行方法

①「ツール」「コマンドライン」「開発者用コマンドプロンプト」
② cd プロジェクト名\bin\debug（例： cd chap8\bin\debug）
③ プロジェクト名に続けてコマンドライン引数を記述する（例： chap8 cat dog apple）
※ この環境では「\」が「/」の反対（バックスラッシュ）になるが同じ文字コードなので問題ない

p.200（バッチファイル）

・複数のコマンドをまとめて実行したい場合に、これらを記述して「●.bat」として保存するとバッチファイルとみなされる
・コマンドプロンプトでバッチファイル名をそのまま記述するだけで、内容が実行される

補足：バッチファイルの作成例

・開発者用コマンドプロンプトを起動して「notepad a.bat」「はい」でバッチファイルが作成可能になる
・下記を記述して「ファイル」「保存」して閉じる

```
ver
dir
date
```

・「a.bat」と入力すると、バージョン情報、ファイル一覧、日付確認の順に動作する
※日付確認はEnterキーを押すだけでOK

p.199 main02.csの実行方法

p.199「main02.cs」は、p.200のバッチファイル「main02test.bat」で実行する
①「main02.cs」は「デバッグなしで実行」は行わず「ビルド」「ソリューションのビルド」のみを行い、実行可能にしておく
②「ツール」「コマンドライン」「開発者用コマンドプロンプト」
③ cd プロジェクト名\bin\debug（例： cd chap8\bin\debug）
④ notepad main02test.bat
⑤ p.200のバッチファイル「main02test.bat」の内容をコピーペースト
⑥ この中の「main02」を全てプロジェクト名に書き換える
⑦「ファイル」「名前をつけて保存」して閉じる（このときエンコードを「ANSI」に変更すること）
⑧ main02test.bat

p.200のバッチファイル「main02test.bat」

```
@rem main02test.bat
@echo off
main02
echo %errorlevel%
echo 「main02.exe」の呼び出し結果です
main02
echo %errorlevel%
echo 「main02.exe」の呼び出し結果です
main02 a
echo %errorlevel%
echo 「main02.exe a」の呼び出し結果です
main02 20 30
echo %errorlevel%
echo 「main02.exe 20 30」の呼び出し結果です
```

```
main02 30
echo %errorlevel%
echo「main02.exe 30」の呼び出し結果です
```

提出：main02.cs

次回予告：p.201「引数が可変個のメソッド」から