講義メモ
・「【補足】if文等のネスト」の実例を示してから、p.115「for文」に進みます

【補足】if文等のネスト：再掲載

・if文やswitch文の中にさらにif文やswitch文を記述できる。これをネスト(入れ子)という
例：正の数が入力されたら、もう1つの整数を入力させ、どちらも正の数のなら「どちらも正」と表示する。
　あてはまらない場合はなにも表示しない。
　if（正の数？）{ もう1つ入力: if（正の数？）{ 「どちらも正」と表示 } }
※ なお、switch文のネストはどうしても必要な場合を除いて避けた方が良い

ミニ演習 mini114.cs

・上の例をプログラムにしよう

```
//ミニ演習 mini114.cs
using System;
class mini114
{
    public static void Main()
    {
        Console.Write("整数①:"); int n1 = int.Parse(Console.ReadLine()); //整数
入力
        if (n1 > 0) { //正の数？
            Console.Write("整数②:"); int n2 = int.Parse(Console.ReadLine()); //
整数入力
            if (n2 > 0) { //正の数？(ifのネスト)
                Console.WriteLine("どちらも正");
            }
        }
    }
}
```

【補足】if文等のネスト：つづき

・if文等のネストでは「else」や「else if」がどの「if」に対するものなのか見間違いやすくなるので注意
・インデントが正しければif-else、if-elseif-elseが高さで判別できる

悪い例：

```
//ミニ演習 mini114.cs 悪い例
using System;
using System.Diagnostics.Eventing.Reader;

class mini114
{
    public static void Main()
    {
        Console.Write("整数①:"); int n1 = int.Parse(Console.ReadLine()); //整数
入力
        if (n1 > 0) { //正の数？
            Console.Write("整数②:"); int n2 = int.Parse(Console.ReadLine()); //
整数入力
            if (n2 > 0) { //正の数？(ifのネスト)
```

```
                Console.WriteLine("どちらも正");
                } //インデントが正しくないと…
            } //外側のifの終わりのインデントもおかしくなる
            else { //外側のifに対するelseなのだが…
                Console.WriteLine("これはどちらのifのelse？");
        }
    }
}

//ミニ演習 mini114.cs 正しい例
using System;
using System.Diagnostics.Eventing.Reader;

class mini114
{
    public static void Main()
    {
        Console.Write("整数①:"); int n1 = int.Parse(Console.ReadLine()); //整数
入力
        if (n1 > 0) { //正の数？
            Console.Write("整数②:"); int n2 = int.Parse(Console.ReadLine()); //
整数入力
            if (n2 > 0) { //正の数？(ifのネスト)
                Console.WriteLine("どちらも正");
            }
            else { //内側のifに対するelse
                Console.WriteLine("整数①は正の数だが、整数②が正の数ではない");
            }
        }
        else { //外側のifに対するelse
            Console.WriteLine("整数①が正の数ではない");
        }
    }
}
```

p.115 for文

・C#には4種類の繰り返し構造文があり、主に回数指定繰り返しに向くのがfor文
・構文： for(①前処理； ②継続条件式； ③毎回の末尾で行う処理) { ④繰り返し内容 }
・前処理：繰返し開始前に行うこと。主に回数カウンタのクリアがある。省略可。
・継続条件式：bool型を返す式。主にカウンタの値の上限や下限と比較する文。省略可。
・毎回の末尾で行う処理：繰り返し内容の最後の文の後に行うこと。主にカウントアップやカウントアップ。省略可。
・よって、この繰り返しは① ②④③ ②④③ ②④③ …の順で行われ、②の結果によって中断できる
・例：5回繰返す場合： int i; for(i = 0; i < 5; i++) { 繰り返し内容 }
・なお、繰り返し内容が1文しかない時、{}を省略できるが、推奨されないことも多い

p.115 for01.cs

```
//p.115 for01.cs
using System;
class for01
{
    public static void Main()
```

```
    {
        int i; //カウンタ用の変数
        for (i = 0; i < 5; i++) { //①カウンタを0にする ②カウンタが5未満なら④③を繰返す ④カウントアップ
            Console.WriteLine("i = {0}", i); //③繰り返し内容(初回は"i = 0",2回目は"i = 1"…,5回目は"i = 4")
        }
    }
}
```

p.115 for01.csの動作について

01. int i; でカウンタ用の変数が宣言される
02. i = 0;① でカウンタ用の変数が0になる
03. i < 5;② ⇒ 0 < 5; でtrueなので繰り返し継続決定
04. Console.WriteLine("i = {0}", i);③ ⇒ Console.WriteLine("i = 0"); で「i = 0」を表示
05. i++④ でカウンタ用の変数が+1されて1になる
06. i < 5;② ⇒ 1 < 5; でtrueなので繰り返し継続決定
07. Console.WriteLine("i = {0}", i);③ ⇒ Console.WriteLine("i = 1"); で「i = 1」を表示
08. i++④ でカウンタ用の変数が+1されて2になる
09. i < 5;② ⇒ 2 < 5; でtrueなので繰り返し継続決定
10. Console.WriteLine("i = {0}", i);③ ⇒ Console.WriteLine("i = 2"); で「i = 2」を表示
11. i++④ でカウンタ用の変数が+1されて3になる
12. i < 5;② ⇒ 3 < 5; でtrueなので繰り返し継続決定
13. Console.WriteLine("i = {0}", i);③ ⇒ Console.WriteLine("i = 3"); で「i = 3」を表示
14. i++④ でカウンタ用の変数が+1されて4になる
15. i < 5;② ⇒ 4 < 5; でtrueなので繰り返し継続決定
16. Console.WriteLine("i = {0}", i);③ ⇒ Console.WriteLine("i = 4"); で「i = 4」を表示
17. i++④ でカウンタ用の変数が+1されて5になる
18. i < 5;② ⇒ 5 < 5; でfalseなので繰り返し終了決定

p.116 for02.cs

```
//p.116 for02.cs
using System;
class for02
{
    public static void Main()
    {
        int i; //カウンタ用の変数
        for (i = 4; i >= 0; i--) { //①カウンタを4にする ②カウンタが0以上なら④③を繰返す ④カウントダウン
            Console.WriteLine("i = {0}", i); //"i = 4","i = 3","i = 2","i = 1","i = 0",の順に表示
        }
    }
}
```

補足:カウンタ用の変数の初期化の場所について(p.118 最下行)

・カウンタ用の変数をforのブロックの中でのみ用いる場合、forの頭で初期化して良い（予め宣言しなくて良い）
例：
```
for (int i = 4; i >= 0; i--) { //①カウンタを4にする ②カウンタが0以上なら④③を繰返す ④カウントダウン
    Console.WriteLine("i = {0}", i); //"i = 4","i = 3","i = 2","i = 1","i = 0", の順に表示
}
```
・このカウンタ用の変数はforを抜けると無効になるので便利だが、forを抜けた後でも利用したい場合は、予め宣言しておくこと
例：
```
int i;
for (i = 4; i >= 0; i--) { //①カウンタを4にする ②カウンタが0以上なら④③を繰返す ④カウントダウン
    Console.WriteLine("i = {0}", i); //"i = 4","i = 3","i = 2","i = 1","i = 0", の順に表示
}
Console.WriteLine("終了時のi = {0}", i); //-1になる
```

p.112 forループからの脱出

・forを構成する3要素の内、条件式を省略すると無限ループになる
　※ すべてを省略して「for (;;)」としてもOKで、無限ループになる
・この場合、繰り返し内部の任意の場所に「break」を記述することで、ループから脱出できる（繰り返しの後続の処理はスキップされる）
・これは、ループからの脱出であり、その後に記述があれば実行される

p.117 for03.cs

```
//p.117 for03.cs
using System;
class for03
{
    public static void Main()
    {
        int i = 0;
        for ( ; ; ) { //無限ループ!!
            Console.WriteLine("i = {0}", i);
            i++; // WriteLineメソッドが実行されたらiを1増やす
            if (i >= 5) { // iが5以上になったらbreak文で脱出
                break; //forループを抜ける
            }
        }
    }
}
```

アレンジ演習：p.117 for03.cs

・ループ終了時のiの値を表示しよう

作成例

```
//アレンジ演習：p.117 for03.cs
using System;
```

```
class for03
{
    public static void Main()
    {
        int i = 0;
        for ( ; ; ) { //無限ループ!!
            Console.WriteLine("i = {0}", i);
            i++; // WriteLineメソッドが実行されたらiを1増やす
            if (i >= 5) { // iが5以上になったらbreak文で脱出
                break; //forループを抜ける
            }
        }
        Console.WriteLine("終了時のi = {0}", i); //【追加】5になる
    }
}
```

p.118 for04.cs

```
//p.118 for04.cs
using System;
class for04
{
    public static void Main()
    {
        int i;
        for (i = 0; ; i++) { //継続条件式のみ省略できる（無限ループになる）
            if (i >= 5) {
                break; //ループを抜ける
            }
            Console.WriteLine("i = {0}", i);
        }
    }
}
```

アレンジ演習：p.118 for04.cs

・iの値を0から4までカウントアップし、すぐに0までカウントダウンしよう

作成例

```
//アレンジ演習：p.118 for04.cs
using System;
class for04
{
    public static void Main()
    {
        int i;
        for (i = 0; i < 4; i++) { //i=0,1,2,3について繰返す
            Console.WriteLine("i = {0}", i);
        }
        for (; i >= 0; i--) { //i=4,3,2,1,0について繰返す ※事前処理は不要
            Console.WriteLine("i = {0}", i);
        }
    }
```

}

p.119 forループのネスト

・for文の中にfor文を記述できる。これもネスト(入れ子)という
・繰返し文の中に繰返し文を書くことで2重ループが実現する
・forループのネストはN回×M回の繰り返しに便利で、2次元の情報を扱う時に用いることが多い
・主に、外側の繰り返しの用のカウンタにはiを、内側の繰り返しの用のカウンタにはjを使うことが多い
・なお、カウンタを同じ変数にすると想定外の動作になる(通常、無限ループする)ので注意
・実行イメージ

```
 for(int i = 0①; i < 3②; i++⑦) {
  for(int j = 0③; j < 2④; j++⑥) {
   Console.Write("[{0},{1}]", i, j);⑤
  }
 }
 ①②③④⑤[0,0] ⑥④⑤[0,1] ⑥④⑦②③④⑤[1,0] ⑥④⑤[1,1] ⑥④⑦②③④⑤[2,0] ⑥④⑤[2,1] ⑥④⑦②
```

p.119 kuku01.cs

```
//p.119 kuku01.cs
using System;
class kuku01
{
    public static void Main()
    {
        int i, j; //外側用カウンタ、内側用カウンタ
        for (i = 1; i <= 9; i++) { //外側用カウンタで1,2,3,4,5,6,7,8,9について繰返す
            for (j = 1; j <= 9; j++) { //内側用カウンタで1,2,3,4,5,6,7,8,9について繰
返す
                Console.WriteLine("{0} * {1} = {2}", i, j, i * j);
            }
            Console.WriteLine("------------"); //段の区切りを表示
        }
    }
}
```

アレンジ演習:p.119 kuku01.cs

・式を略して、積だけを段ごとに表示し、段の後ろで改行しよう(段の区切りの代わりに改行)
・積は3桁で表示すること

```
  1  2  3  4  5  6  7  8  9
  2  4  6  8 10 12 14 16 18
  3  6  9 12 15 18 21 24 27
  :(略)
```

作成例

```
//アレンジ演習:p.119 kuku01.cs
using System;
class kuku01
{
    public static void Main()
    {
        int i, j; //外側用カウンタ、内側用カウンタ
```

```
        for (i = 1; i <= 9; i++) { //外側用カウンタで1,2,3,4,5,6,7,8,9について繰返す
            for (j = 1; j <= 9; j++) { //内側用カウンタで1,2,3,4,5,6,7,8,9について繰
返す
                Console.Write("{0, 3}", i * j); //積を3桁表示し改行しない
            }
            Console.WriteLine(); //段の区切りで改行
        }
    }
}
```

p.121（Mathクラス）

・C#が提供する算術系クラス(情報の構造体の一種で詳細は7章で)の一つがMathで、その中に算術計算に便利なメソッドや定数などが用意されている
・利用には「Math.」を前置する。
・定数Math.PI：円周率を提供する定数(double型)
・メソッドMath.Sin(ラジアン値)：サインを返すメソッド(double型)。カッコ内に角度を表すラジアン値をdouble型で指定する
※ メソッドの呼び出しにおいてカッコ内に指定する値や式を引数という（後述）
・ラジアン値は180度を円周率とした値で、n度のラジアン値は n / 180 * Math.PI で得られる
※ sin01.csでは4度のラジアン値を 4÷180×円周率 ⇒ Math.Pi / 45.0 として扱っている
※ そして、0度から180度まで4度ずつ繰返すfor文になっている
・メソッドMath.Round(実数)：小数点以下を四捨五入した結果を返すメソッド(double型)。
 例： Math.Round(3.8) ⇒ 4.0 となる

p.121 sin01.cs

```
//p.121 sin01.cs
using System;
class sin01
{
    public static void Main()
    {
        double s; //カウンタとして用いる実数
        //0.0度から180度(Math.PI)まで4度(Math.PI / 45.0)ずつ増やしながら繰返す
        for (double a = 0.0; a <= Math.PI; a += Math.PI / 45.0) {
            s = Math.Sin(a); //サイン値を得る
            Console.Write("{0,7:#.#####}:", s); //7桁の小数点以下5桁で表示
            //サイン値0.2につき"*"を1個表示することでグラフを描く
            for (int i = 1; i <= Math.Round(s * 50); i++) { //サイン値の50倍(小数点
以下四捨五入)だけ繰返す
                Console.Write("*"); //"*"を1個表示(改行しない)
            }
            Console.WriteLine(); //1行分終わったので改行
        }
    }
}
```

補足：p.121 sin01.csの実行方法

・現在のバージョンのVisual Studioでは1ページ分以上の表示を一気に行うと欠けが発生してしまう
・そこで「ビルド」「ソリューションのビルド」までをVisual Studioで行い、実行はコマンドプロンプトで行うと良い
・コマンドプロンプトは「すべてのアプリ」「Windowsツール」にあるのでダブルクリック
・ここで、プログラムのあるプロジェクトのフォルダの中のbin/debugフォルダにある.exeファイルを指定する

例：E:\ha234_C#_akiba\chap4\chap4\bin\Debug\chap4.exe
・実行例：

```
C:\Users\human>E:\ha234_C#_akiba\chap4\chap4\bin\Debug\chap4.exe
        :
 .06976:***
 .13917:*******
 .20791:**********
 .27564:**************
 .34202:*****************
 .40674:*******************
 .46947:************************
 .52992:**************************
 .58779:*****************************
 .64279:*******************************
 .69466:*********************************
 .74314:***********************************
 .78801:*************************************
 .82904:**************************************
 .86603:***************************************
 .89879:****************************************
 .92718:*****************************************
 .95106:******************************************
  .9703:*******************************************
 .98481:*******************************************
 .99452:********************************************
 .99939:********************************************
 .99939:********************************************
 .99452:********************************************
 .98481:*******************************************
  .9703:*******************************************
 .95106:******************************************
 .92718:*****************************************
 .89879:****************************************
 .86603:***************************************
 .82904:**************************************
 .78801:*************************************
 .74314:***********************************
 .69466:*********************************
 .64279:*******************************
 .58779:*****************************
 .52992:**************************
 .46947:************************
 .40674:*******************
 .34202:*****************
 .27564:**************
 .20791:**********
 .13917:*******
 .06976:***
```

アレンジ演習：p.121 sin01.cs

・サイン値の表示がすべて「.00000」から「.99999」の5桁になるようにしよう
・「{0,7:#.#####}:」を「{0,7:#.00000}:」とすれば良い

・実行結果

```
C:\Users\human>E:\ha234_C#_akiba\chap4\chap4\bin\Debug\chap4.exe
 .00000:
 .06976:***
 .13917:*******
 .20791:**********
 .27564:**************
 .34202:******************
 .40674:********************
 .46947:************************
 .52992:**************************
 .58779:*****************************
 .64279:*******************************
 .69466:*********************************
 .74314:***********************************
 .78801:*************************************
 .82904:**************************************
 .86603:***************************************
 .89879:****************************************
 .92718:*****************************************
 .95106:******************************************
 .97030:*******************************************
 .98481:********************************************
 .99452:*********************************************
 .99939:*********************************************
 .99939:*********************************************
 .99452:*********************************************
 .98481:********************************************
 .97030:*******************************************
 .95106:******************************************
 .92718:*****************************************
 .89879:****************************************
 .86603:***************************************
 .82904:**************************************
 .78801:*************************************
 .74314:***********************************
 .69466:*********************************
 .64279:*******************************
 .58779:*****************************
 .52992:**************************
 .46947:************************
 .40674:********************
 .34202:******************
 .27564:**************
 .20791:**********
 .13917:*******
 .06976:***
```

提出：アレンジ演習：p.121 sin01.cs

次回予告：p.122「while文とdo while文」から再開します