比較対照 METAPOST & Asymptote 〜 第 5 回

今回は **path** と **guide** , **picture** と **frame** です。Asymptote の **path** と **guide** は METAPOST の **path** を機能分化させたようなもの，Asymptote の **picture** と **frame** は METAPOST の **picture** を機能分化させたようなものです。

1 path と guide

100 個程度の点 (**pair** 値) からなる曲線 (**path** 値) を描くにはどうしたらいいでしょう。(ちなみに，100 個の点はすでに z[0] 〜 z[99] に設定済みとします。) さすがに 100 個すべてを .. で結ぶのは大変です。ここで for 文を使うことが考えられますが，1 つ問題があります。

```
path p=z[0];  for(int i=1; i<100; ++i){p=p..z[i];}
```

で一見いけそうに見えますが，これでは z[0]..z[1] , z[1]..z[2] , ····· , z[98]..z[99] のブツ切りの ⟨ パス ⟩ をつないだだけで，思ったようにつながるとは限りません[1]。

METAPOST の場合は「for 文で繰り返す内容は文単位でなくてもいい」という，プログラム言語の中では特殊な仕様のおかげで，次のように記述すれば問題は解決です。

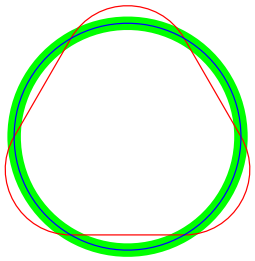
```
for n = 0 upto 98: z[n].. endfor z99;
```

しかし，Asymptote では，C 言語と同様 for 文で繰り返す内容は文単位ですので，この手は使えません。

こんなときに役立つのが **guide** です。**guide** 値は **path** 値と同様に，**pair** 値を .. や -- などでつないで記述しますが，**path** 値が記述と同時に曲線の形状を決めてしまうのに対し，**guide** 値は記述の際は構成する **pair** 値と接続方法のみ記録し，実際に使う段になって初めて曲線を構築します。もう少し詳しく言うと，**guide** 値は draw や fill など，**pair** 値が使える文脈で **pair** 値の代わりに使え，このときに **pair** 値に変換されて[2]実際の曲線の形状が決定されます。

具体例を見てもらった方が早いので，以下例を挙げます。上では話の勢い (?) で 100 個の点と言いましたが，実例でそれは面倒なので 6 個の点にします。METAPOST や Asymptote では同一円周上に等間隔に点をとり，それらを .. で結ぶとほぼ完全な円になります[3]。6 個の点は円周上に等間隔にとってますので，普通に結べば (緑の太線) 円になります。これを for 文ブツ切りでつなぐと (赤の線) 円とは異なるよく分からない曲線になってしまいます。for 文を使うなら **pair** 値ではなく **guide** 値を使うべきで，これで描けば (青の線) 一気に 6 つの点を結んだときと同じ円になります。

```
\begin{ASYpic}<1.5cm>(0,0)(2,-2)
  \sendASY{pair[] z;
    for(int i=0; i<6; ++i){z[i]=dir(60i);}
    path p=z[0]..z[1]..z[2]..z[3]..z[4]..z[5]..cycle;
    path q=z[0];  for(int i=1; i<6; ++i){q=q..z[i];} q=q..cycle;
    guide g=z[0]; for(int i=1; i<6; ++i){g=g..z[i];} g=g..cycle;
    draw(p,green+linewidth(5pt)); draw(q,red); draw(g,blue);
  }
\end{ASYpic}
```



赤の線の形状について少し補足しておきますと，まず，for 文の 1 回目の実行で z[0]..z[1] の ⟨ パス ⟩ が形成されます。この ⟨ パス ⟩ はこの時点では 2 点しか含まれないので線分です。そして，終端 z[1] での ⟨ パス ⟩ の向きは z[0] から z[1] の向きです。円周方向ではありません。次に，for 文の 2 回目の実行で z[1]..z[2] の ⟨ パス ⟩ が追加されます。このとき，新たに追加された部分は前回の終端である z[1] での向きを引き継ぎます。z[2] での向きは直接指定されていませんが，METAPOST や Asymptote で 2 点 (**pair** 値) を .. で結び，曲線の向きを一方の端点のみで与えると，できあがった曲線が「2 点を結ぶ線分の垂直二等分線」に関して対称になるようにもう一方の端点での向きが決まるため，終端 z[2] での ⟨ パス ⟩ の向きは z[2] から z[3] の向きになります。さらに，for 文の 3 回目の実行で z[2]..z[3] の ⟨ パス ⟩ が追加されます。このとき，前回の終端 z[2] での向きが z[2] から z[3] の向きでしたので，追加される部分は線分になります。以下同様です。

ちなみに，METAPOST ではこんな感じになります。

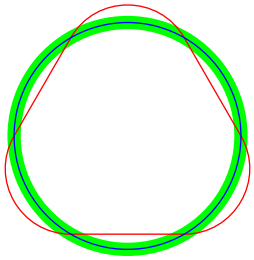
---

[1] 後の実例を見れば分かると思いますが，接合位置での ⟨ パス ⟩ の向きは引き継がれるようです。しかし，それだけでは不十分なことも実例を見れば明らかです。

[2] いわゆる型キャストです。

[3] 4 点程度でも十分な精度です。

```
\begin{MPpic}<1.5cm>(0,0)(2,-2)
  \sendMP{for n=0upto5: z[n]=w*dir60n; endfor
    path p,q,r;
    p:=z0..z1..z2..z3..z4..z5..cycle;
    q:=z0; for n=1upto5: q:=q..z[n]; endfor q:=q..cycle;
    r:=for n=0upto5: z[n].. endfor cycle;}
  \sendMP[5pt]{draw p withcolor green;}
  \sendMP{draw q withcolor red; draw r withcolor blue;}
\end{MPpic}
```



2 picture と frame

METAPOST では描画面（**picture** 値，デフォルトは currentpicture）は初めから PostScript の基本単位である bp を単位長とする座標系ですが，Asymptote では前回説明したように currentpicture も含め，各 **picture** 値はそれぞれ独自の単位長を持てます[4]。そして，別の図に貼り付ける，あるいは PostScript データとして出力するとき，その単位長を反映させます。その単位長を反映させた後のものは，PostScript データと同じく bp 単位の座標系を持つデータであり，これを **picture** 値と区別して **frame** 値と呼びます。ちなみに，前回紹介した .fit() は，実は **picture** 値を **frame** 値に変換する命令です。また,「固定サイズで貼り付け」は，内部的に .fit() の引数で単位長 1bp（拡大縮小なし）に相当する変換を指定して **frame** 値に変換して貼り付けています。さらに currentpicture は，ファイルの最後で，とくに指定されていなくても自動的に .fit() が実行され **frame** 値に変換されて最終的なデータになります。

Asymptote の **frame** 値は,「bp 単位の座標系を持つデータ」という意味では METAPOST の **picture** 値に近いものですが，METAPOST の **picture** 値と異なり，Asymptote では **frame** 値に直接描画するのではなく，**picture** 値（もちろん Asymptote の）に描き込んだ後 **frame** 値に変換するというスタイルになっています。**frame** 値に変換するのは，通常，別の **picture** 値に貼り付けるか最終出力をするときかですので,「変換後直ちに使用する」ため，**frame** 値の変数[5]を直接使う機会はあまりないかも知れません[6]。

《クイズ》 **picture** 値変数 pica , picb を用意し，単位長を次のように設定します。

```
unitsize(pica,0.5,1.5);  unitsize(picb,3);  unitsize(2);
```

つまり，pica は横方向 0.5 倍，縦方向 1.5 倍，picb は縦横とも 3 倍，currentpicture は縦横とも 2 倍です。

ここで，中心 (0,1cm) , 半径 1cm の円を pica に描きます。

```
draw(pica,circle((0,1cm),1cm));
```

その後，pica を picb に貼り付け，さらに picb を currentpicture に貼り付けることを考えます。

以下のそれぞれの場合について，円の中心と半径（楕円のときは長半径と短半径）はいくらになるでしょう。

pica→picb	picb→currentpicture	中心	半径
add(picb,pica);	add(picb);		
	add(picb.fit());		
	add(picb,(w,0));		
add(picb,pica.fit());	add(picb);		
	add(picb.fit());		
	add(picb,(w,0));		
add(picb,pica,(-w,0));	add(picb);		
	add(picb.fit());		
	add(picb,(w,0));		

答は次回。ヒント：サイズが決まるのは最初に .fit() したとき。

[4] unitsize ではなく size を使えば，単位長ではなく図全体のサイズを指定することになりますが，図の拡大縮小という点では同じなので，以下区別しません。適宜読み替えてください。

[5] currentpicture に対応する currentframe はありません。

[6] 引数に **frame** 値を要求する関数（命令）は結構ありますので，**frame** 値を使わないわけではありません。ただ，それらの引数には .fit() などで生成した **frame** 値を直接記述することが多いので。変数に名前を付けて使うことが少ないという意味です。**frame** 値変数をよく使うのは，**frame** 値を操る関数（命令）を定義するときでしょう。