比較対照 METAPOST & Asymptote 〜 第 8 回

1 構造化データ

え〜，訳語として正しいかどうか，ちょっと自信がないですが，Asymptote の structure の訳として「構造化データ」という言葉を使うことにします。C++ や Delphi を使っている人には class 型のデータを想像してもらえばいいかと思います[1]。

わかりやすい例として，ある会の会員リストを管理することを考えて下さい。会員 1 人 1 人につき，管理するデータはたとえば「名前 (**string** 値)」「年齢 (**int** 値)」「性別 (**bool** 値？)」など，タイプの違うデータの集合になります。タイプは違いますが，これらはセットで扱うべきものです。新規会員が入ればこれらのデータを一括で入力しますし，脱会すれば一括で削除します。コピーするときも一括です。こういう場合,「組にしたデータとそれらに対する基本操作」をまとめて扱える仕組みがあると便利です。

これが「構造化データ」で，これまでに出てきたものでいうと **picture** 値が構造化データの例になっています[2]。たとえば，**picture** 値変数にはそれぞれ固有の拡大率 (あるいはサイズ) が設定できます。これは **picture** 値変数の内部に `xunitsize` , `yunitsize` , `zunitsize` , `xsize` , `ysize` などの **real** 値が含まれており，たとえば **picture** 値変数 `pic` に対して `unitsize(pic,2);` とすれば `xunitsize` , `yunitsize` , `zunitsize` の 3 つが 2 に設定され，`size(pic,200);` とすれば `xsize` と `ysize` が 200 に設定されます．つまり，**picture** 値変数は単なる画像データではなく，こういった多種多様なデータの集合体になっています。さらに,「**picture** 値変数が内部に持っているデータ」に対する基本操作 (関数) も **picture** 値変数に含まれます。たとえば **picture** 値変数 `pic` を (拡大率を反映した) **frame** 値に変換して `currentpicture` 貼り付けるとき，`add(pic.fit());` としましたが，これは **picture** 値変数が持っている `fit` という関数を (`pic` に対して) 呼び出しているのです。なお，この例でも分かるように，**picture** 値変数 `pic` が内部に持っている関数や変数は `pic.fit()` や `pic.xunitsize` のように，`pic` と関数や変数の名前をピリオドでつないで表します。

2 Label 型変数

さて，前置きが長くなってしまいましたが，前回の続き — 前回保留した **Label** 値変数の説明です。

「ラベル」は図を描いている人から見ると「図に『文字列』を貼り付けた」というイメージでしょうが，内部的なことを考察すればそう単純なものではないことが分かると思います。「図に『文字列』を貼り付け」るためには「文字列」に相当する **string** 値を指定の (あるいは既定の) フォント・サイズ・色で組み，それを図形データとして実現し，必要なら拡大などの変形も施した上で貼り付けるわけですから，単なる文字列 (**string** 値) では表しきれない情報を必要とします。

こういうときに役立つのが前述の構造化データで，**Label** 値変数は「ラベル」関連の情報を一括して扱うための構造化データです。

Label 値変数が内部に持っているデータ (変数) は,

文字列を表す **string** 値変数の `s` ,
貼り付け先の座標を表す **position** 値変数の `position` ,
貼り付け先の点に対しどの方向に貼り付けるかを示す **align** 値変数の `align` ,
文字を描くペンを表す **pen** 値変数の `p` ,
文字列画像の変形を表す **transform** 値変数の `T` ,
後述の「貼り付け後の変形」をどの程度まで許容するかを表す **embed** 値変数の `embed` ,
枠を描いたり塗りつぶしたりするかを表す **filltype** 値変数の `filltype`

などから成ります[3]。これらのうち，`position` , `align` , `p` , `filltype` は，前回紹介した `label` 関数の引数で上書きされる[4]ので，今回は **transform** 値 `T` , **embed** 値 `embed` をいじってみましょう。

---

[1] class 型よりは object 型に近いかも。

[2] Asymptote の **picture** 値の話です。METAPOST の **picture** 値は単なる画像データで，Asymptote の **frame** 値に近いデータタイプです。

[3] 他にもいくつか変数を持ってます。ちなみに，**position** 値や **align** 値なども構造化データです。たとえば，**position** 値は内部に **pair** 値と **bool** 値を持っていて，**pair** 値で座標，**bool** 値でその座標が (曲線弧などに対する) 相対座標か普通の座標かを表します。

[4] **position** 値は **pair** 値の型キャストで設定されます。

3　ラベルの変形

ラベルは拡大縮小や回転，傾斜といった変形を行うことができます。やりかたは **picture** 値や **path** 値に対する変形と同様で，**Label** 値の**左から** **transform** 値をかける（ * ）だけです。これは，**transform** 値 の **Label** 値への積が，**Label** 値の内部にある **transform** 値 T（ デフォルトは恒等変換 ）への積と解釈されるようプログラムが組まれているからです。

ただしこの場合，前回のように **string** 値を型キャストで **Label** 値と解釈するわけにはいかないため，明示的に **Label** 値を生成します。**Label** 値を生成する命令（ 関数 ）は型名と同名の `Label` という関数で，次のような書式で使います。

`Label`(**string** 値 (必須), **align** 値, **pen** 値, **embed** 値, **filltype** 値)

`label` 命令で貼り付けるとき，上記の引数のうち **align** 値，**pen** 値，**filltype** 値は `label` 命令の引数で上書きされるので，これらはデフォルトのまま放置しておけばいいでしょう。**embed** 値引数はラベルを貼り付けた**後**の変形をどの程度許容するかを指定するもので[5]，以下の 5 タイプがあります。

`Shift` — 貼り付け先の図に対する変形のうち平行移動のみ追従。

`Rotate` — 貼り付け先の図に対する変形のうち平行移動と回転のみ追従。デフォルトはこれです。

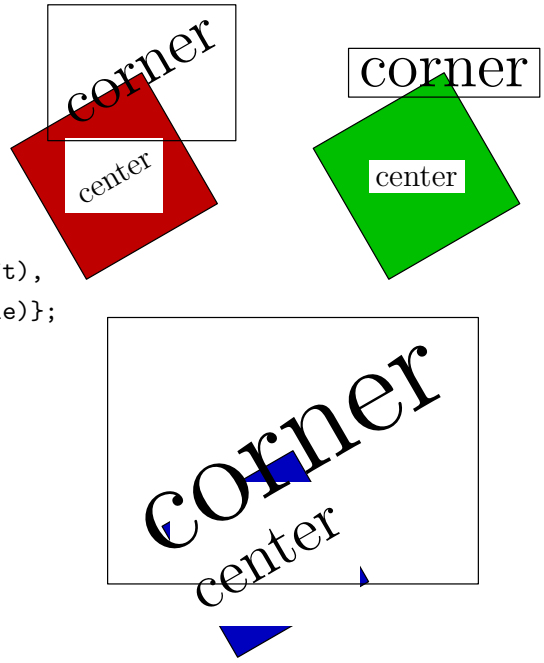
`Slant` — 貼り付け先の図に対する変形のうち平行移動と回転と傾斜と反転のみ追従。

`Scale` — 貼り付け先の図に対する変形（ 平行移動，回転，傾斜，反転，拡大縮小 ）すべてに追従。

`Rotate`(**pair** 値) — 貼り付け先の図の座標系で引数に与えた **pair** 値を解釈し，その方向角だけ回転。

以上を踏まえて次のサンプルを見て下さい。

```
\begin{ASYpic}*<1cm>(0,0)(6,-5.5)
  \sendASY{unitsize(2); picture[] pic;
    pair[] pos={(0,0),(2w,0),(w,-2.5h)};
    pen[] p={heavyred,heavygreen,heavyblue};
    Label[] L = {Label("center"),       Label("corner"),
                 Label("center",Shift), Label("corner",Shift),
                 Label("center",Scale), Label("corner",Scale)};
    for (int i=0; i<3; ++i){
      pic[i] = new picture;
      filldraw(pic[i],scale(w)*unitsquare,p[i],black);
      label(pic[i],L[2i],(0.5w,0.5w),Fill(1bp,1bp,white));
      label(pic[i],scale(2)*L[2i+1],(w,w),Draw(1bp,1bp));
      add(shift(pos[i])*rotate(30)*pic[i]);
    }
  }
\end{ASYpic}
```



このサンプルでは 3 つの **picture** 値変数 `pic[0]` , `pic[1]` , `pic[2]` を用意し，それぞれに赤，緑，青で 1 辺の長さ 1cm の正方形を描いています。さらにこの正方形の中心に center というラベル，右上の頂点に corner というラベルを貼り付けています。corner のラベルは `scale(2)` で 2 倍に拡大されていることにも注意して下さい。そしてこれらを `currentpicture` に貼り付けます。このとき各図を 30° 回転して貼り付けています。`currentpicture` は `unitsize(2)` を指定していますので，最終的に出力される図では 1 辺の長さが 2 cm の 30° 傾いた正方形が描かれます。

さて，**embed** 値引数の効果についてですが，まず，`pic[0]`（ 赤の正方形 ）では **embed** 値引数を指定せずに `Label` 関数を使っています。したがって，デフォルトの `Rotate` が採用され，図の回転にともなってラベルも回転しています[6]。しかし，`currentpicture` 出力時の「2 倍の拡大」の影響は受けていません。

次に，`pic[1]`（ 緑の正方形 ）では **embed** 値引数に `Shift` を指定しましたので，図の回転や拡大の影響は受けていません。もちろん，貼り付け先の図の変形の影響を受けないだけで，ラベル自体の拡大はきっちり反映されています。

最後の `pic[2]`（ 青の正方形 ）では **embed** 値引数に `Scale` を指定しましたので，図の回転だけでなく拡大の影響も受けています。したがって，corner のラベルは `scale(2)` によるラベルの拡大と，貼り付け先の図の拡大を合わせて $2 \times 2 = 4$ 倍の拡大になっていることに注意して下さい。

[5]たとえば，貼り付け先の図を回転させたとき，「図形は回転したいけど，図の注釈たるラベルは回転したくない」という場合，**embed** 値引数で変形を抑制します。

[6]残念ながら背景の矩形は，座標軸と平行な辺を持つものにしかならないようです。