12 章 “再帰呼び出し” の仕組み.

Risa/Asir ドリル 2022, p.249, 24 章 “Risa/Asir プログラマのための C 言語入門” も参照.

コード 1: ベクトルの和, C 言語版 vecadd.c

```
#include <stdio.h>
#define N 3
void vecadd(int A[N],int B[N],int C[N]) {
  int I;
  for (I=0; I<N; I++) {
    C[I]=A[I]+B[I];
  }
}
int main() {
  int A[N] = {1,2,3};
  int B[N] = {3,2,1};
  int C[N];
  int i;
  vecadd(A,B,C);
  printf("C=");
  for (i=0; i<N; i++) printf("%d␣",C[i]);
  printf("\n");
}
```

コード 2: ベクトルの和 asir 版 vecadd.rr

```
def vecadd(A,B,C) {
  N=length(A);
  for (I=0; I<N; I++) {
    C[I]=A[I]+B[I];
  }
}

A=newvect(3,[1,2,3]);
B=newvect(3,[3,2,1]);
C=newvect(3);
vecadd(A,B,C);
printf("C=%a\n",C)$

def vecadd2(A,B) {
  N=length(A);
  C=newvect(N);
  for (I=0; I<N; I++) {
    C[I]=A[I]+B[I];
  }
  return C;
}
end$
```

以下の実行方法:

```
emacs sample1.c &
```

でソースコードを入力して, shell で下記のようにコンパイル, 実行.

```
cc sample1.c glib5.c -lX11 -lm
./a.out
```

glib5.c, glib5.h は講義のページからダウンロード. -l で C 言語のライブラリを指定. -lX11 は glib5.c ライブラリを用いて graphic を描く時に必要. -lm は sin, cos などを使うときに必要.

コード 3: sample1.c

```
#include <stdio.h>  // かならず書く
#include "glib5.h"  // かならず書く
int main() {
  glib_open();
  glib_loop();
}
/* 描画の手続きはこの中に書く. ここを自由に修正
   .
*/
void glib_draw() {
  double y;
  glib_window(0.0, 0.0, 1.0, 1.0);

  glib_color(NULL,0xff0000);  // Red
  glib_line(0,0,0.7,0.1);

  glib_color(NULL,0x00ff00);  // gleen
  glib_line(0,0,0.7,0.5);

  glib_color(NULL,0x0000ff);  // blue
  for (y=0.5; y<1.0; y = y+0.01) {
    glib_putpixel(y,y);
  }

  glib_color("black",0);
  glib_text(0.1,0.1,"Hello");
  glib_flush();
}
```

注意: asir の | color=色 構文は使えません.

コード 4: sample2.c

```
#include <stdio.h>  // かならず書く
#include <stdlib.h>   // random を使う時.
#include <math.h>   // exp, log, cos, sin
  などを使う
  時
#include "glib5.h"  // かならず書く
/*超入門の円を描くプログラムを
 C 言語で書いた例.

cc sample2.c glib5.c -lX11 -lmでコンパイル
.
./a.out で実行
*/
int main() {
  /** main は変更しない. 描画の手続きは下記の
      glib_draw (call back 関数となる)に書く.
  **/
  glib_open();
  glib_loop();
}
/*関数宣言は
   glib_draw での描画の前に.
*/

/*
  glib_line(X1,Y1,X2,Y2 | color=C) は
  myglib_line(X1,Y2,X2,Y2,C);
*/
void myglib_line(double x,double y,double
  x2, double y2, int color) {
  glib_color(NULL,color);
  glib_line(x,y,x2,y2);
}

/*円を描く関数
  .
*/
```

```
void circle(double X,double Y,double R,int Color) {
  double E,T;
  double Px,Py,Qx,Qy;
  E=0.1;
  glib_color(NULL,Color);
  for (T=0; T<=2*3.14; T = T+E) {
    Px = X+R*cos(T);
    Py = Y+R*sin(T);
    Qx = X+R*cos(T+E);
    Qy = Y+R*sin(T+E);
    glib_line(Px,Py,Qx,Qy);
  }
  glib_flush();
}

/* 全体描画の手続きはこの中に書く.
*/
void glib_draw() {
  double y;
  double CC,P;

  glib_clear();
  glib_window(-1,-1,1,1);
  CC = 0xff0000;
  for (P=0.4; P<0.5; P = P+0.01) {
    circle(0,0,P,CC);
    CC = random()%0x1000000;
  }
  glib_flush();
}
```

sample3.c はありません.

コード 5: sample4.c

```
#include <stdio.h>
#include <math.h>  // pow(X,A) : X^A の計算関数, を使う時は必ず必要.
#include "glib5.h"  // かならず書く
/*講義のページから
 glib5.c, glib5.h をダウンロードしてこのファイルと同じフォルダに置いておく
.
cc sample4.c glib5.c -lX11 -lmでコンパイル
.
./a.out で実行

*/
int main() {
  /** main は変更しない. 描画の手続きは下記の
      glib_draw (call back 関数となる)に書く.
  **/
  glib_open();
  glib_loop();
}

/*
  glib_line(X1,Y1,X2,Y2 | color=C) は
  myglib_line(X1,Y2,X2,Y2,C);これはこのまま使う
  .
*/
void myglib_line(double x,double y,double x2, double y2, int color) {
  glib_color(NULL,color);
  glib_line(x,y,x2,y2);
}

/*次関数のグラフを描く例
   2.
   gib_draw の名前を変えてはいけない.
*/
void glib_draw() {
  double X,Y,X1,Y1;
  glib_window(-2,-2, 2,2);
  myglib_line(-2,0,2,0 ,0x0000ff);
  myglib_line(0,-2,0,2 ,0x0000ff);
  for (X=-2.0; X< 2.0; X = X+0.1) {
    Y = pow(X,2)-1; // Y = X*X-1;
    X1 = X+0.1;
    Y1 = pow(X1,2)-1; //Y1 = X1*X1-1;
    glib_line(X,Y, X1,Y1);
  }
  glib_flush();
}
```