G1. 分割コンパイル

各ソース・ファイルを，コマンド

```
tateyama~$ gcc -c xxxxx.c
```

で (狭義) コンパイルする．このコマンドを実行するとソース・ファイルにエラーがなければ，xxxxx.o と言うファイル名のオブジェクト・ファイルが生成される．必要な全てのソース・ファイルをコンパイルした後，生成されたオブジェクト・ファイルを，コマンド

```
tateyama~$ gcc -o 出力ファイル名 xxxxx.o yyyyy.o   ..
                                                ...  zzzzz.o
```

によってリンクし，実行型ファイルを生成する．出力ファイル名の後に必要な全てのオブジェクト・ファイルを，スペースで区切って書き並べる．

リンクのコマンドは，オブジェクト・ファイルが多数に及ぶ場合，コマンド入力が煩わしいので，次に示す Makefile をオブジェクト・ファイルが保存されているディレクトリー内に作成し，そのディレクトリーでコマンド make を実行してもよい．

〈 **Makefile** 〉

```
出力ファイル名 : xxxxx.o yyyyy.o  .....  zzzzz.o
; gcc -o 出力ファイル名 xxxxx.o yyyyy.o  .....
                                                zzzzz.o
```

以下の [1] から [5] のプログラム例に対して，それぞれ別のディレクトリーを作成する．そのディレクトリー内に各ソースファ

イルを保存し，コンパイル・リンクもそのディレクトリー内で実行する．

[1] 行列の積・ポインター版 (matrix_multi_ptr)

　このプログラムは，ポインターを用いて行列を処理している．ポインターの指し示す変数のアドレスを理解する為の例題で，やや不自然で強引な使用法を敢えて採っている．特に，関数 output は，配列変数に格納された値を参照しているだけであるのに，ポインターを引数に取っている．行列の無理のない扱いは [2] の構造体版で示す．

〈 **matrix.h** 〉

```
int input_dim(void);
void input(float *x, int d, int m, char mn,
                                        char en);
void output(float *x, int d, int m, char mn);
```

〈 **main.c** 〉

```
#include<stdio.h>
#include"matrix.h"
#define MAXDIM 10

main()
{
     float a[MAXDIM][MAXDIM], b[MAXDIM][MAXDIM],
                              c[MAXDIM][MAXDIM];
     int dim;
```

```
    dim = input_dim();

    input(&a[0][0], dim, MAXDIM, 'A', 'a');
    printf("\n");
    output(&a[0][0], dim, MAXDIM, 'A');
    printf("\n");

    input(&b[0][0], dim, MAXDIM, 'B', 'b');
    printf("\n");
    output(&b[0][0], dim, MAXDIM, 'B');
    printf("\n");

    multi(&a[0][0], &b[0][0], &c[0][0], dim,
                                        MAXDIM);

    printf("Product of matrices A and B = \n");
    output(&c[0][0], dim, MAXDIM, 'C');

    return 0;
}
```

⟨ **input_dim.c** ⟩

```
#include<stdio.h>

int input_dim()
{
```

```
     int d;

     printf("Input dim = ?");
     scanf("%d", &d);

     return d;
}
```

⟨ **input.c** ⟩

```
#include<stdio.h>

void input(float *x, int d, int m, char mn, char en)
{
     int i, j;

     printf("Input matrix %c\n", mn);

     for(i=1;i<=d;i++){
          for(j=1;j<=d;j++){
               printf("Input %c[%d][%d] = ?", en, i,
                                                  j);
               scanf("%f", x);
               if(j != d){
                    x = x + 1;
               }
          }
          x = x + (m - d + 1);
```

```
    }
}
```

⟨ **output.c** ⟩

```
#include<stdio.h>

void output(float *x, int d, int m, char mn)
{
    int i, j;

    if(mn != 'C'){
        printf("Matrix %c =\n", mn);
    }

    for(i=1;i<=d;i++){
        for(j=1;j<=d;j++){
            printf("%f", *x);
            if(j != d){
                x = x + 1;
                printf("\t");
            }
        }
        if(i != d){
            x = x + (m - d) + 1;
        }
        printf("\n");
    }
```

```
}
```

⟨ **multi.c** ⟩

```
#include<stdio.h>

void multi(float *x, float *y, float *z, int d,
                                                int m)
{
     int i, j, k;
     float *orga_x, *orga_y;

     orga_x = x;
     orga_y = y;

     for(i=1;i<=d;i++){
          for(j=1;j<=d;j++){
               x = orga_x + (i - 1)*m;
               y = orga_y + (j - 1);
               *z = 0;
               for(k=1;k<=d;k++){
                    *z = *z + ((*x) * (*y));
                    if(k != d){
                         x = x + 1;
                         y = y + m;
                    }
               }
               if(j != d){
```

```
                    z = z + 1;
                }
            }
            if(i != d){
                z = z + (m - d) + 1;
            }
        }
}
```

G2. 構造体

構造体とは，変数の組で構成され，プログラマーがその目的に応じて自由に構成可能な，一般化変数である．例えば，行列を扱うには，２次元配列 m[10][10]，その行列の次元を格納する整数型変数 dim，行列に与えられた名前 (識別子) を格納する文字型変数name 等を，ひとまとめに格納可能な変数があれば便利である．C 言語では，このような変数の組を定義することが出来る．この例であれば，定義しようとしている変数の組に Matrix と名付けるとして，

```
struct Matrix{
    float m[10][10];
    int dim;
    char name;
}
```

と，main 関数の外側で宣言すればよい．

この宣言は，新たな変数 (の組の) 型を定義したのであり，その型を持つ変数を使用するためには，各関数の中で通常の変数と

同様に変数宣言する必要がある．例えば，

```
struct Matrix ma, mb;
```

などと宣言すると，上で定義された内部構造を持つ変数 ma，mb が主記憶上に確保される．関数の引数，戻り値としても，通常の変数と全く同様に使用可能であるが，構造体の型名のみでなく，上の変数宣言の場合と同様に struct を型名の前に明示する必要がある．例えば，引数に Matrix を取り，戻り値にも Matrix を取る関数 func の場合，struct Matrix func(struct Matrix mx); とプロトタイプ宣言する．

ma の内部の変数，例えば配列の $(2, 3)$ 要素を参照または操作するには，識別子 ma.m[2][3] を用いる．つまり ma.m[2][3] が, 配列の $(2, 3)$ 要素のための変数である．内部変数 dim を参照・操作するには，識別子 ma.dim を用いる．

[2] 行列の積・構造体版 (matrix_multi_str)

〈 **matrix.h** 〉

```
#define MAXDIM 10

struct Matrix{
     float m[MAXDIM][MAXDIM];
     int dim;
     char mname, ename;
};

int input_dim(void);
struct Matrix input(int dim, char mn, char en);
```

```
void output(struct Matrix mx);
struct Matrix multi(struct Matrix mx, struct Matrix
                                                    my);
```

⟨ **main.c** ⟩

```
#include<stdio.h>
#include"matrix.h"

main()
{
     struct Matrix am, bm, cm;
     int d;

     d = input_dim();

     am = input(d, 'A', 'a');
     printf("\n");
     output(am);
     printf("\n");

     bm = input(d, 'B', 'b');
     printf("\n");
     output(bm);
     printf("\n");

     cm = multi(am, bm);
```

```
    printf("Product of matrices A and B = \n");
    output(cm);

    return 0;
}
```

⟨ **input_dim.c** ⟩

```
#include<stdio.h>

int input_dim()
{
    int d;

    printf("Input dim = ?");
    scanf("%d", &d);

    return d;
}
```

⟨ **input.c** ⟩

```
#include<stdio.h>
#include"matrix.h"

struct Matrix input(int dim, char mn, char en)
{
    struct Matrix mx;
    int i, j;
```

```
    mx.dim = dim;
    mx.mname = mn;
    mx.ename = en;

    printf("Input matrix %c\n", mn);

    for(i=1;i<=dim;i++){
        for(j=1;j<=dim;j++){
      printf("Input %c[%d][%d] = ? ", en, i, j);
            scanf("%f", &mx.m[i-1][j-1]);
        }
    }

    return mx;
}
```

⟨ **output.c** ⟩

```
#include<stdio.h>
#include"matrix.h"

void output(struct Matrix mx)
{
```

課題１.

```
}
```

⟨ **multi.c** ⟩

```
#include<stdio.h>
#include"matrix.h"

struct Matrix multi(struct Matrix mx,
                                    struct Matrix my)
{
     int i, j, k, d;
     struct Matrix mz;

     d = mx.dim;
     mz.dim = d;
     mz.mname = 'C';
     mz.ename = 'c';

     for(i=1;i<=d;i++){
          for(j=1;j<=d;j++){
               mz.m[i-1][j-1] = 0;
               for(k=1;k<=d;k++){
                    mz.m[i-1][j-1] = mz.m[i-1][j-1]
                  + mx.m[i-1][k-1] * my.m[k-1][j-1];
               }
          }
     }

     return mz;
}
```

[3] 掃出し法・逆行列 (matrix_inv)

⟨ **matrix.h** ⟩

```
#define MAXDIM 10

struct Matrix{
     float m[MAXDIM][MAXDIM];
     int dim;
     char mname, ename;
};

int input_dim(void);
struct Matrix input(int dim, char mn, char en);
void output(struct Matrix mx);
struct Matrix unit_ini(int dim, char mn, char en);
float select(struct Matrix mx, int i, int j);
float det_culc(float x, float d);
struct Matrix div(struct Matrix mx, float x, int i);
struct Matrix cancel(struct Matrix mx, float x,
                                        int i, int j);
```

⟨ **main.c** ⟩

```
#include<stdio.h>
#include<math.h>
#include"matrix.h"
#define ZERO 0.00000000000001
```

```
main()
{
     struct Matrix ma, mb;
     float x, det;
     int dim, i, j;

     printf("We culculate an inverse matrix.\n");

     dim = input_dim();
     ma = input(dim, 'A', 'a');
     output(ma);

     mb = unit_ini(dim, 'B', 'b');
     det = det_culc(1, 1);

     for(i=1;i<=dim;i++){
          x = select(ma, i, i);
          det = det_culc(x, det);

          if(fabs(x) <= ZERO){
               printf("We cannot solve this problem.
                                          Sorry.\n");
               return 0;
          }

          ma = div(ma, x, i);
```

```
            mb = div(mb, x, i);

            for(j=1;j<=dim;j++){
                if(j != i){
                    x = select(ma, j, i);
                    ma = cancel(ma, x, i, j);
                    mb = cancel(mb, x, i, j);
                }
            }
        }

        printf("\n");
        printf("det A = %f\n", det);
        printf("\n");
        printf("Inverse matrix B of A is\n");
        output(mb);

        return 0;
}
```

〈 **input_dim.c** 〉

行列の積・構造体版と同じ

〈 **input.c** 〉

行列の積・構造体版と同じ

〈 **output.c** 〉

行列の積・構造体版と同じ

〈 **unit_ini.c** 〉

```
#include<stdio.h>
#include"matrix.h"

struct Matrix unit_ini(int dim, char mn, char en)
{
     struct Matrix mx;
     int i, j;

     mx.mname = 'B';
     mx.ename = 'b';
     mx.dim = dim;

     for(i=1;i<=dim;i++){
          for(j=1;j<=dim;j++){
               if(i == j){
                    mx.m[i-1][j-1] = 1;
               }
               else{
                    mx.m[i-1][j-1] = 0;
               }
          }
     }

     return mx;
}
```

⟨ **cancel.c** ⟩

```
#include<stdio.h>
#include"matrix.h"

struct Matrix cancel(struct Matrix mx, float x,
                                        int i, int j)
{
     int k;

     for(k=1;k<=mx.dim;k++){
          mx.m[j-1][k-1] = mx.m[j-1][k-1]
                              + (-x) * mx.m[i-1][k-1];
     }

     return mx;
}
```

⟨ **det_culc.c** ⟩

```
#include<stdio.h>
#include"matrix.h"

float det_culc(float x, float d)
{
     d = x * d;

     return d;
}
```

⟨ **div.c** ⟩

```
#include<stdio.h>
#include"matrix.h"

struct Matrix div(struct Matrix mx, float x, int i)
{
     int j;

     for(j=1;j<=mx.dim;j++){
          mx.m[i-1][j-1] = mx.m[i-1][j-1]/x;
     }

     return mx;
}
```

⟨ **select.c** ⟩

```
#include<stdio.h>
#include"matrix.h"

float select(struct Matrix mx, int i, int j)
{
     return mx.m[i-1][j-1];
}
```

[4] ファン・デル・ポール・標準出力版 (vdp_eul)

⟨ **vdp_eul.h** ⟩

```
struct Vector{
```

```
    float x, y;
};

struct Vector ini_v();
float input_step();
int data_l();
struct Vector euler(struct Vector vo, float t);
```

⟨ **main.c** ⟩

```
#include <stdio.h>
#include <math.h>
#include "vdp_eul.h"
#define BOUND 10000000

main()
{
    struct Vector v_old, v_new;
    float t;
    int i, dl;

    v_old = ini_v();
    t = input_step();
    dl = data_l();

    for(i=1;i<=dl;i++){
         v_new = euler(v_old, t);
         if(fabs(v_new.x) >= BOUND || fabs(v_new.y)
```

```
                                                >= BOUND){
            printf("The solution has diverged !
                                                    \n");

            return 0;
        }
        printf("%f %f\n", v_new.x, v_new.y);
        v_old = v_new;
    }

    return 0;
}
```

⟨ **ini_v.c** ⟩

```
#include <stdio.h>
#include "vdp_eul.h"

struct Vector ini_v()
{
    struct Vector v;

    printf("Input an initial value.\n");
    printf("   x = ?");
    scanf("%f", &v.x);
    printf("   y = ?");
    scanf("%f", &v.y);
```

```
    return v;
}
```

⟨ **data_l.c** ⟩

```
#include <stdio.h>

int data_l()
{
    int d;

    printf("How many times do you iterate ?\n");
    printf("Input n = ?");
    scanf("%d", &d);

    return d;
}
```

⟨ **input_step.c** ⟩

```
#include <stdio.h>

float input_step()
{
    float t;

    printf("Input a step time = ?");
    scanf("%f", &t);
```

```
    return t;
}
```

〈 **euler.c** 〉

```
#include <stdio.h>
#include "vdp_eul.h"
#define EPSILON 0.5

struct Vector euler(struct Vector vo, float t)
{
    struct Vector vn;

    vn.x = vo.x + t * vo.y;
    vn.y = vo.y + t * (EPSILON * (1 - vo.x * vo.x)
                                  * vo.y - vo.x);

    return vn;
}
```

G3. ファイル入出力

C 言語においてファイルを扱うには，ファイル・ポインターを宣言する必要がある.

```
FILE *fp
```

と宣言すると，fp と名付けられたファイル・ポインター(主記憶上に展開されたファイルを操作する為のポインター)が，変数の場合と同様に確保される．ファイルにデータを出力するには,

まず，ファイルに名前を与え，その名前のファイルを開く (主記憶上に展開する) 必要がある．

```
fp = fopen("data.txt", "w");
```

によって，data.txt と名付けられたファイルが書き込み可能な状態で，主記憶上に展開される．上の"w" が書き込み用にファイルが開かれることを示している．fopen の戻り値は，fp が指し示すべきアドレスとなっている．実際の書き込みは，scanf と printf を併せたような書式で，書き込み先のアドレス (fp) と書き込むデータの型を参照して実行する．

```
fprintf(fp, "%f", x);
```

は，浮動小数点数型の変数 x に格納されている値をアドレス fp の (主記憶上に展開された) ファイルに書き込むことを指示している．

fclosed(fp) によって，主記憶上に展開されていたファイルが，主記憶上から消える．

[5] ファン・デル・ポール・ファイル出力版 (vdp_eul_f)
標準出力版の ⟨ **main.c** ⟩ を ⟨ **main_f.c** ⟩ に変更

⟨ **main_f.c** ⟩

```
#include <stdio.h>
#include <math.h>
#include "vdp_eul.h"
#define BOUND 10000000


main()
```

```
{
    struct Vector v_old, v_new;
    float t;
    int i, dl;
    FILE *fp;

    fp = fopen("data.txt", "w");

    v_old = ini_v();
    t = input_step();
    dl = data_l();

    fprintf(fp, "%f   %f\n", v_old.x, v_old.y);

    for(i=1;i<=dl;i++){
         v_new = euler(v_old, t);
         if(fabs(v_new.x) >= BOUND || fabs(v_new.y)
                                          >= BOUND){
              printf("The solution has diverged!
                                                \n");

              return 0;
         }
         v_old = v_new;
         fprintf(fp, "%f  %f\n", v_old.x, v_old.y);
    }
```

```
    fclose(fp);

    return 0;
}
```