プログラミング演習I第5回レポート

学籍番号：2364902

名前：キム　ギュソク

(1) 課題番号と課題内容

A. 基本課題 3

i. 2 次元配列で 2 x 2 の正方行列を表現し、逆行列を求める関数を作成する。逆行列を求める関数の引数は元の配列および逆行列用とする。逆行列を求める関数内で行列式を計算し、0 のときは逆行列が存在しないので、 関数を呼び出した側にそれがわかるように、逆行列を求める関数の戻り値を行列式の値とする。

B. 基本課題 4

i. 配列でベクトルを表現し、その内積を求める関数を作成する。内積を求める関数は配列の形式（ a[i] ）で内積を求めるように作成する。なお、この関数ではポインタを動かす形式（ *a++ ）で内積を求めない。関数の戻り値が内積の値となるようにすること内積を求める関数にはベクトルの次数を引数で渡し、 任意の次元のベクトルを扱うことができるようにする。main 関数ではベクトルの次数およびデータをキーボードから入力し、 内積を求める関数へ渡す。

(2) フローチャートまたは疑似言語によるアルゴリズムの記述

A. 基本課題 3

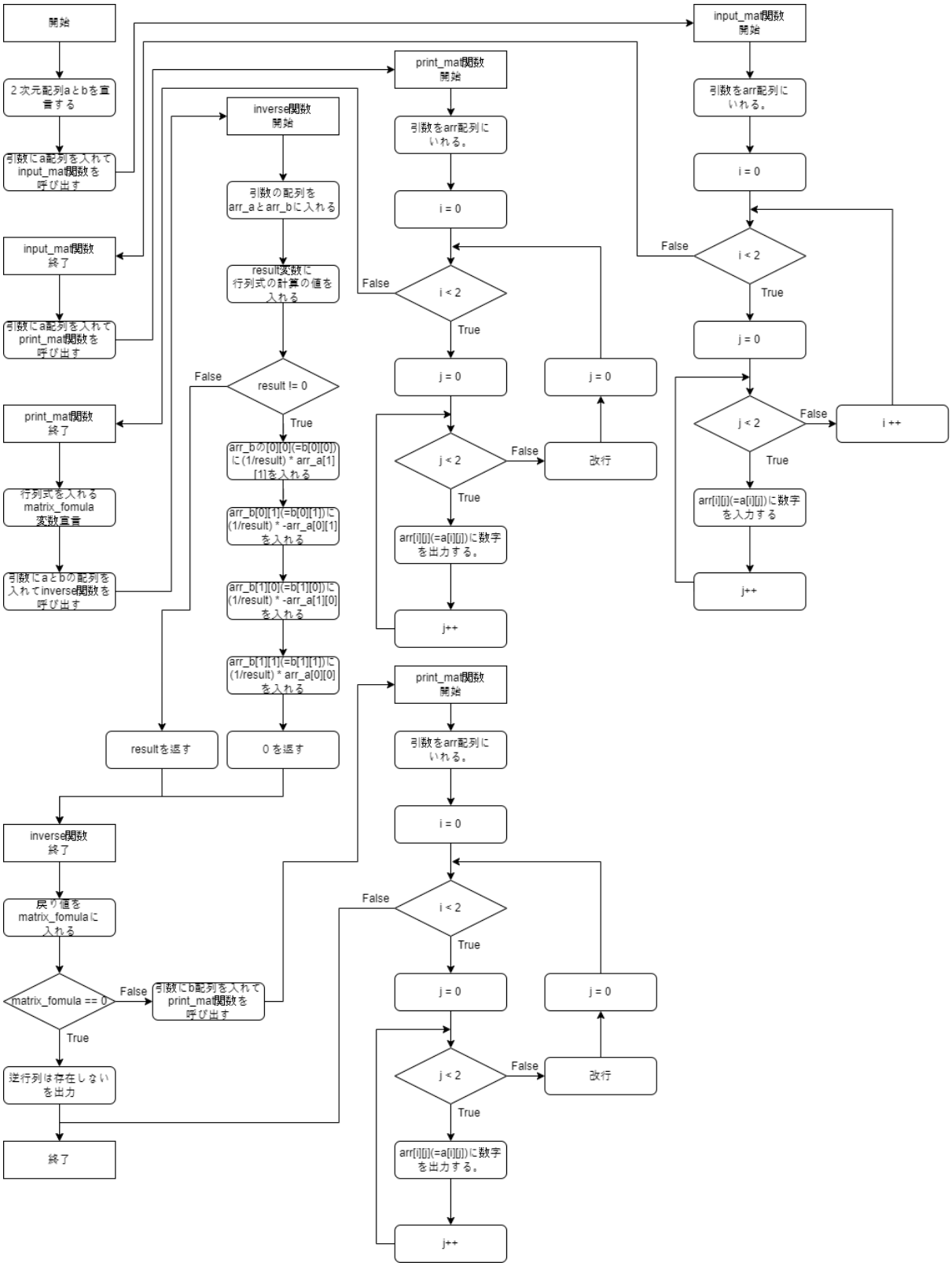


図 1。基本課題 3 フローチャート

B. 基本課題 4

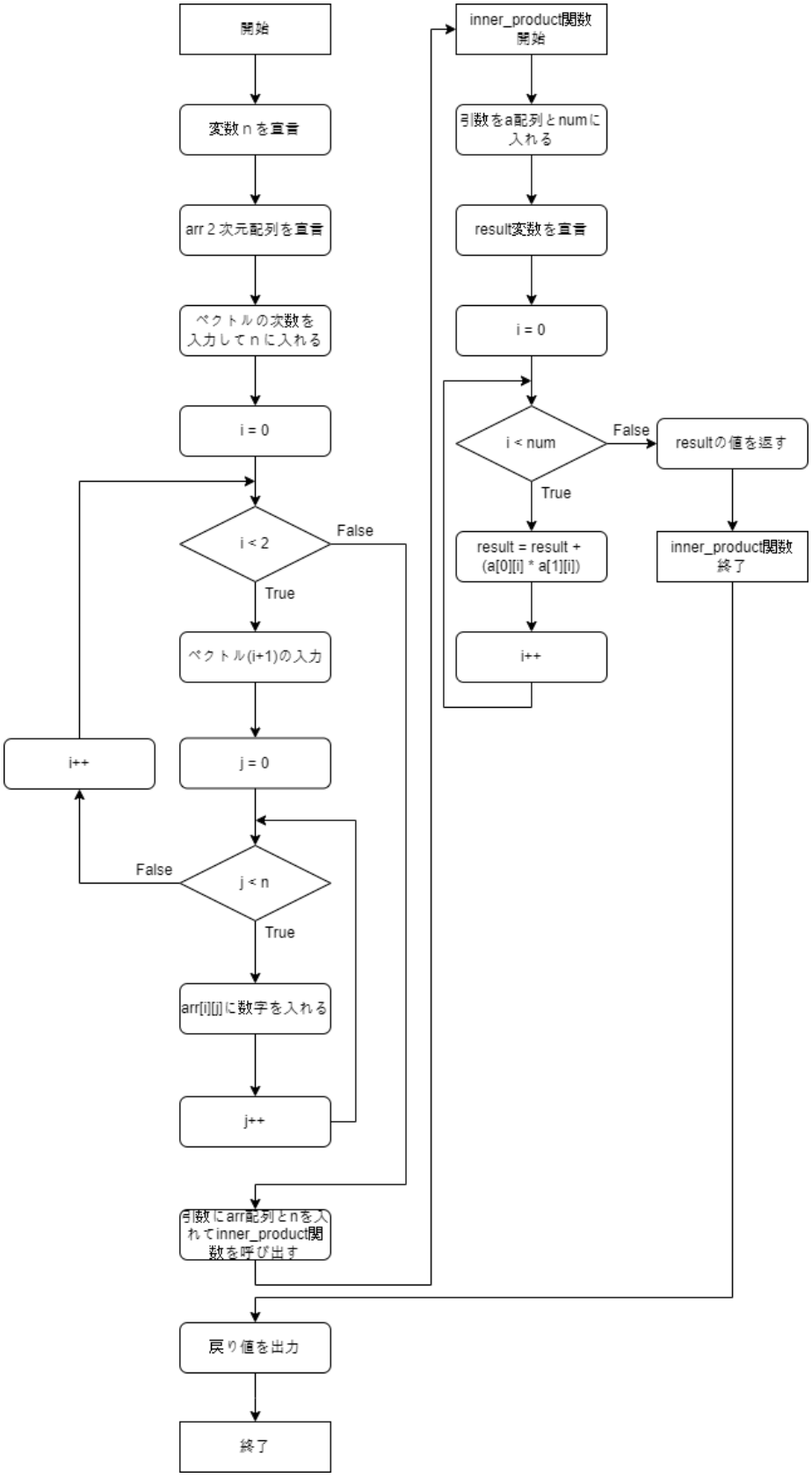


図 2。基本課題 4 フローチャート

(3) アルゴリズムが正しいことの説明

A. 基本課題３

i. 正当性

1. 理論的な逆行列の求め方は行列 $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ の時に求め方は次のとおりである。

   A. 行列式を求める

   $$\det A = ad - bc$$

   B. A行列を余因子行列に変える

   $$adjA = \begin{bmatrix} d & -c \\ -b & a \end{bmatrix}$$

   C. 余因子行列と行列式を利用して逆行列を求める

   $$A^{-1} = \det A \times adjA$$

   $$A^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -c \\ -b & a \end{bmatrix}$$

2. 基本課題３のフローチャート見るとわかるようにinverse関数の中で行列式を求めて、行列式が０ではないときに行列aとdの位置を入れ替えるために逆行列であるb配列の[0][0]の位置にa配列の[1][1]を入れる。そして、残りの数字を余因子行列の方法に合わせて入れ替えると余因子行列を求めることができる。そして、最後に逆行列配列のすべての要素に$\frac{1}{\det A}$をかけることで逆行列bを得られることができる。よって、このアルゴリズムは正しい結果が得られる正当なアルゴリズムだと考えられる。

ii. 停止性

1. 基本課題３では繰り返し文をinput_mat関数とprint_mat関数で2回ずつ使われている。

   A. input_mat関数の停止性

      i. 大きく見て外側のループと内側のループ二つ存在する。行を表すiと列を表すjを利用して行列の全ての要素を入力することができる。基本課題３の行列は２×２の行列なのでiとjを０において２未満の条件をおいて++することで２×２の要素全てに触れて正しく停止することができる。よって、無限ループに入ることはなく正しい結果が得られた時に停止する。

   B. print_mat関数の停止性

      i. print_mat関数もinput_mat関数と同じなので正しく停止するように設計されていると考えられる。

B. 基本課題４

i. 正当性

1. 理論的なベクトルの内積の求め方は以下の通りである。

$ベクトル\vec{A}= (a_1\ a_2 \dots a_n)$、$ベクトル\vec{B} = (b_1\ b_2 \dots b_n)$

$$内積 = a_1b_1 + a_2b_2 + \cdots + a_nb_n$$

2. 基本課題４のフローチャートを見るとわかるように２次元配列を利用して[0][]にAベクトルの要素を入れて[1][]にBベクトルの要素を入れて内積の計算を行う。今回nの値を入力してもらって利用しているので繰り返し文を利用してiをnまで増やしながら[0][i]×[1][i]の計算を全て足すことで内積の計算を行うことができる。段階的に見ると

$$0番目\ \ a_{00} \times a_{10}$$

$$+$$

$$1番目\ \ a_{01} \times a_{11}$$

$$+$$

$$\cdots$$

$$n番目\ \ a_{0n} \times a_{1n}$$

$$a_{00}a_{10} + a_{01}a_{11} + \cdots + a_{0n}a_{1n}$$

よって、理論的な内積と同じく結果になることがわかるので正当なアルゴリズムだと考えられる。

ii. 停止性

1. 配列に数字を入れる時の繰り返し文と内積の計算を行う時の繰り返し文二つ存在する。数字を入力する時の繰り返し文は基本課題３の場合と同じなので正しく停止する。基本課題４では列が入力されたnなのでその以外は基本課題３と同じである。そして、内積の計算を行う時の繰り返し文は列の繰り返しだけなのでiを n まで増やしながら各要素をかけるのでiがnになったら停止するので正しい結果が得られた時に停止するように設計されている。

(4) ソース・プログラムの説明

A. 基本課題 3

ソースコード

main.c

```c
#include <stdio.h>

void input_mat(double arr[][2]);
void print_mat(double arr[][2]);
double inverse(double arr_a[][2], double arr_b[][2]);

int main(void){
    double a[2][2];
    double b[2][2];

    printf("2 x 2 の行列の要素の値を入力してください\n");
    input_mat(a);
    print_mat(a);

    double matrix_formula_a = inverse(a, b);

    if(matrix_formula_a == 0){//when inverse matrix has not existed
        printf("の逆行列は存在しない。");
    }
    else{//when inverse matrix has existed
        printf("の逆行列は\n");
        for(int i = 0; i < 2; i++){
            for(int j = 0; j < 2; j++){
                printf("%lf ", b[i][j]);
            }
            printf("\n");
        }
    }
}

void input_mat(double arr[][2]){
    for(int i = 0; i < 2; i++){
        for(int  j = 0; j < 2; j++){
            printf("a[%d][%d] = ", i, j);
            scanf("%lf", &arr[i][j]);
        }
    }

}
void print_mat(double arr[][2]){
    for(int i = 0; i < 2; i++){
        for(int  j = 0; j < 2; j++){
            printf("%lf ", arr[i][j]);
        }
        printf("\n");
    }
}

double inverse(double arr_a[][2], double arr_b[][2]){
    double matrix_formula = (arr_a[0][0] * arr_a[1][1]) - (arr_a[0][1]*arr_a[1][0]);
    if(matrix_formula != 0){
        arr_b[0][0] = (1/matrix_formula) * arr_a[1][1];
        arr_b[0][1] = (1/matrix_formula) * -arr_a[0][1];
        arr_b[1][0] = (1/matrix_formula) * -arr_a[1][0];
        arr_b[1][1] = (1/matrix_formula) * arr_a[0][0];
    }
    else{
        return matrix_formula;
    }
}
```

実行結果

```
2 x 2 の行列の要素の値を入力してください
a[0][0] = 1
a[0][1] = 2
a[1][0] = 3
a[1][1] = 4
1.000000 2.000000
3.000000 4.000000
の逆行列は
-2.000000 1.000000
1.500000 -0.500000
```

```
2 x 2 の行列の要素の値を入力してください
a[0][0] = 12
a[0][1] = 30
a[1][0] = 6
a[1][1] = 15
12.000000 30.000000
6.000000 15.000000
の逆行列は存在しない。
```

i. ソースコードの説明

1. まず実数の計算を行うのでdouble型で入力された行列を入れる２次元行列aと逆行列を入れるためのbを宣言する。そして、a配列を入力するためのinput_mat関数に引数としてaを入れて呼び出す。関数を宣言するときに２次元配列はa ＝ {{a, b} {c, d}}として定義されているので中の配列と外側の配列が存在する。よって、引数としてarr[][2]として外側の配列をもらうことで配列を引数としてもらうことができる。そして、繰り返し文を外側と内側に二つ作ることで２×２の行列に入れることができる。また、main関数にある配列の要素に入れることは配列aの定義は一番最初の０番目の要素のアドレスなので配列aとarrのアドレスは同じであると考えられる。よって、input_mat関数で入れることでmain関数のa配列に入れることができる。input_mat関数を利用して配列に入れてprint_mat関数に引数aを入れて繰り返し文二つ利用して出力する。そして、最後にinverse関数で行列式の計算を行って０の時は０を返して行列式が０なので逆行列が存在しないことを示す。０じゃないときは逆行列の計算を行ってb配列にb配列逆行列を入れる。そして、main関数の最後にprint_mat関数にb配列を入れて呼び出すことで逆行列を出力する。

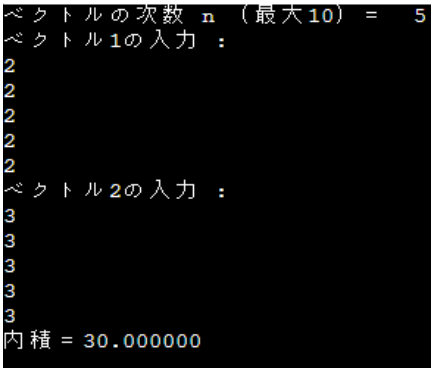
B. 基本課題４

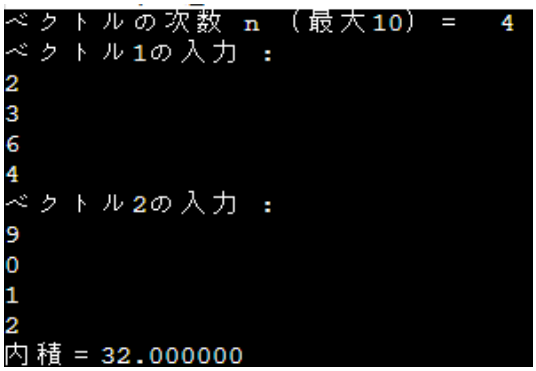
ソースコード

main.c

```c
#include <stdio.h>


double inner_product(double a[][10], int num);

int main(void){

    int n = 0;
    double arr[2][10];//declaration of max array

    printf("ベクトルの次数 n （最大10） =  ");
    scanf("%d", &n);


    for(int i = 0; i < 2; i++){
        printf("ベクトル%dの入力 : \n", i+1);
        for(int j = 0; j < n; j++){
            scanf("%lf", &arr[i][j]);
        }
    }


    printf("内積＝%lf", inner_product(arr, n));

    return 0;
}

double inner_product(double a[][10], int num){
    double result = 0.0;
    for(int i = 0; i < num; i++){
        result += (a[0][i] * a[1][i]);//calculation of inner product
    }
    return result;
}
```

実行結果





i. ソースコードの説明

1. return値が実数なのでdouble型でinner_produt関数のプロトタイプ宣言して次数を入力してもらう変数ｎと二つのベクトルを入れるための２次元配列をarrで宣言する。次数を入力すると繰り返し文二つを利用して[0][]配列を一つ目の配列で[1][]を二つ目の配列として扱う。よって、各ベクトルの要素を入れるためには[i][j]とおいてiは2まで増やしてjはｎまで増やすことで要素を入力する。そして、inner_produt関数にarr配列とｎを入れて内積結果を入れるresultを宣言して繰り返し文を利用して各ベクトルの要素をかけてresultに足して入れる。そして、resultの値を返してprintfで出力する。

(5) 考察

A. 基本課題３

i. 基本課題３で考察したいと思うことは２次元配列を関数の引数として使う方法である。特に、関数の中で数字を入力する場合は配列ではなく変数の場合関数の中だけで使われているので入力された変数をreturnするかポインタを利用してアドレスに接近する必要がある。しかし、配列の場合配列の名前が一番最初の０番目の要素が入っているアドレスである。よって、引数として配列の名前を入れてもアドレスを入れることなのでポインタの仕組みと同じである。また、今回は２次元配列を使うので引数として使うときに1次元の配列と違う方法を取っている。２次元配列の場合列の情報が必要であるため列の部分に情報を入れる必要がある。よって、関数の中で数字を入力してもmain関数の配列に入ることができる理由は引数としてアドレスをやり取りするので[][n]として列の情報を入れることは２次元配列を引数としてもらうときに必要であると考えられる。また、逆行列の計算を行う際に余因子行列を求める際に繰り返し文を利用して作ると行列の大きさが大きくなると問題がないので効率的なコードを作ることができると思う。この時、インデックス変数iとｊ以外にもｍとｎを別に宣言してｍとｎは０まで減りながらa配列の要素を計算することで繰り返し文で計算することができる。そして、if文を利用してiとｊが同じ時は余因子行列の計算でマイナスになるので変えると作ることができる。より大きな行列の計算の時はこの方法が効率的だと考える。

B. 基本課題４

i. ２次元配列の列の値を固定ではなく入力された数字で宣言することについて考察したいと思う。簡単に考えると列の情報をｎに入れて配列を[2][n]で宣言すると簡単だと考える可能性がある。しかし、配列の大きさを変数で宣言することはできない。よって、基本課題４の場合配列の列を最大値である10を入れることでnまでの要素を入れてその後は宣言されているので初期化されてない場合は意味のないデータが入っている。そして、全ての計算や出力をnを利用してn以上は触れることないようにするとまるで入力された数字で配列を宣言したように作ることができる。しかし、この方法だと無駄なアドレス空間を使っているので9以下の数字を入力した時に無駄なアドレス消費が生じて効率的なコードだとは考えにくい。よって、無駄なアドレス消費を避けるためには動的割り当ての方法を使う必要がある。C言語の場合動的割り当て関数mallocを利用することで動的割り当てされた配列を宣言することができる。つまり、基本課題４のように配列の大きさが変数の場合動的割り当て方法で配列を宣言することが効率的なコードになると考えられる。

(6) 感想

A. ２次元配列の仕組みを理解することができた。また、配列の大きさが変数の場合のやり方も学ぶことができた。そして、２次元配列を関数の引数として利用するときの注意点についてもわかるようになった。また、配列の名前の意味がアドレスであることで起こるコードの作成の違いもわかるようになった。