プログラミング演習I第4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1

i. 3 次元ベクトルの構造体を用い、この構造体を引数として 外積を返す関数を作成する。

B. 基本課題 2

i. 2つの自然数を引数にして呼び出すと、再帰処理で最大公約数を返す関数を作成する。2 つの自然数の値をキーボードから入力しこの関数を利用して入力した 2つの自然数の最大公約数を表示する。

C. 基本課題 3

i. 任意の自然数（ n <= 2,147,483,647 ）を引数に呼び出されると、再帰処理を使って 3桁ごとに「 ,（ カンマ ）」で区切って出力する関数を作成

1. 入力値 n が 3 桁以下ならば、入力された値をそのまま表示する。そうでなければ、n/1000 を値を n として関数を再帰的に呼び出す。(つまり、下 3 桁を除いた上記桁を n として呼び出す)呼び出した関数から戻ってきた後に、「,」と再帰呼び出しの際に除いた下 3桁（ n%1000 ）を表示する。（ 例えば、12035007 なら再帰呼び出しで 12,035 を表示してから「,007」を表示する）

D. 基本課題 4

i. 任意の自然数（ n <= 4,294,967,295 ）を引数に呼び出されると、再帰処理を使わず 繰り返し処理を使って 3桁ごとに「 ,（ カンマ ）」で区切って出力する関数を作成する。この関数では配列を宣言し、表示する下 3 桁（ n%1000 ）ごとに配列に保存し、 最上位の桁まで繰り返し処理を行う。main 関数では、任意の自然数をキーボードから入力し、この関数を利用して、 3桁ごとに「 , 」で区切って表示する。

(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1

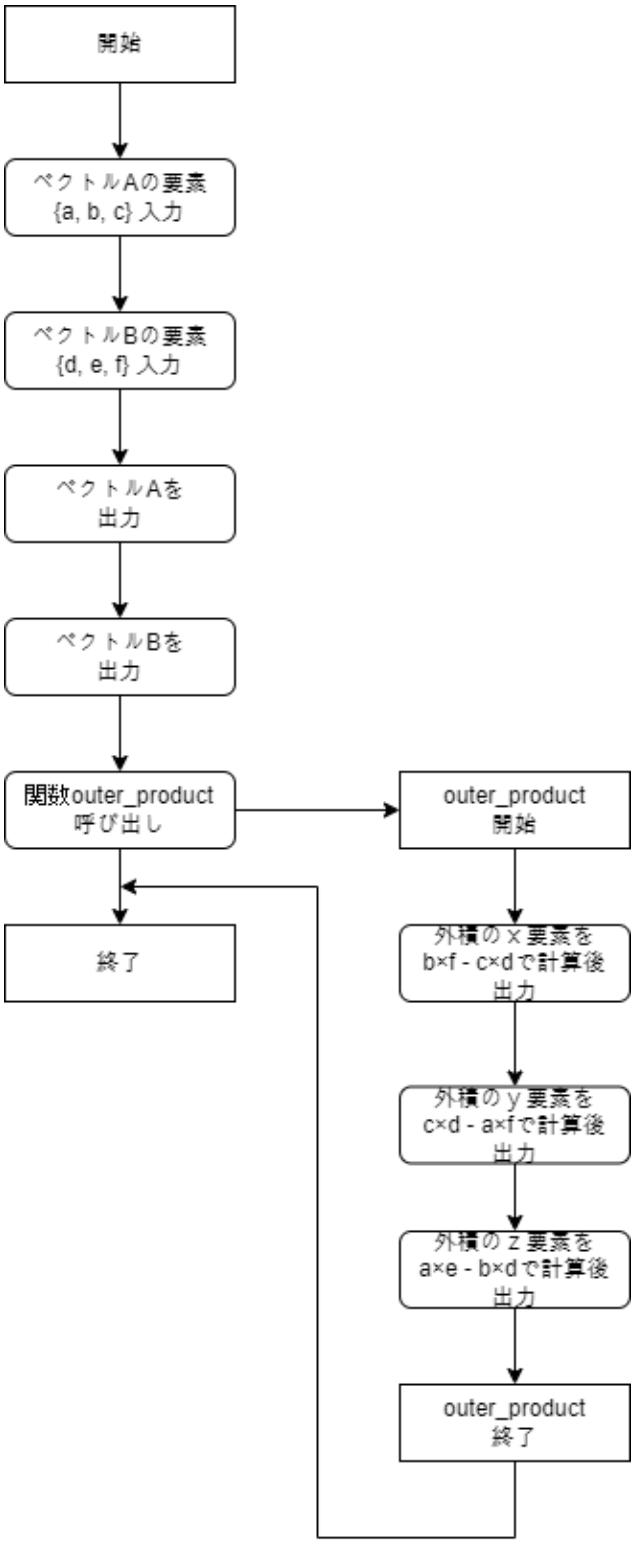


図 1。課題 1 フローチャート

B. 基本課題 2

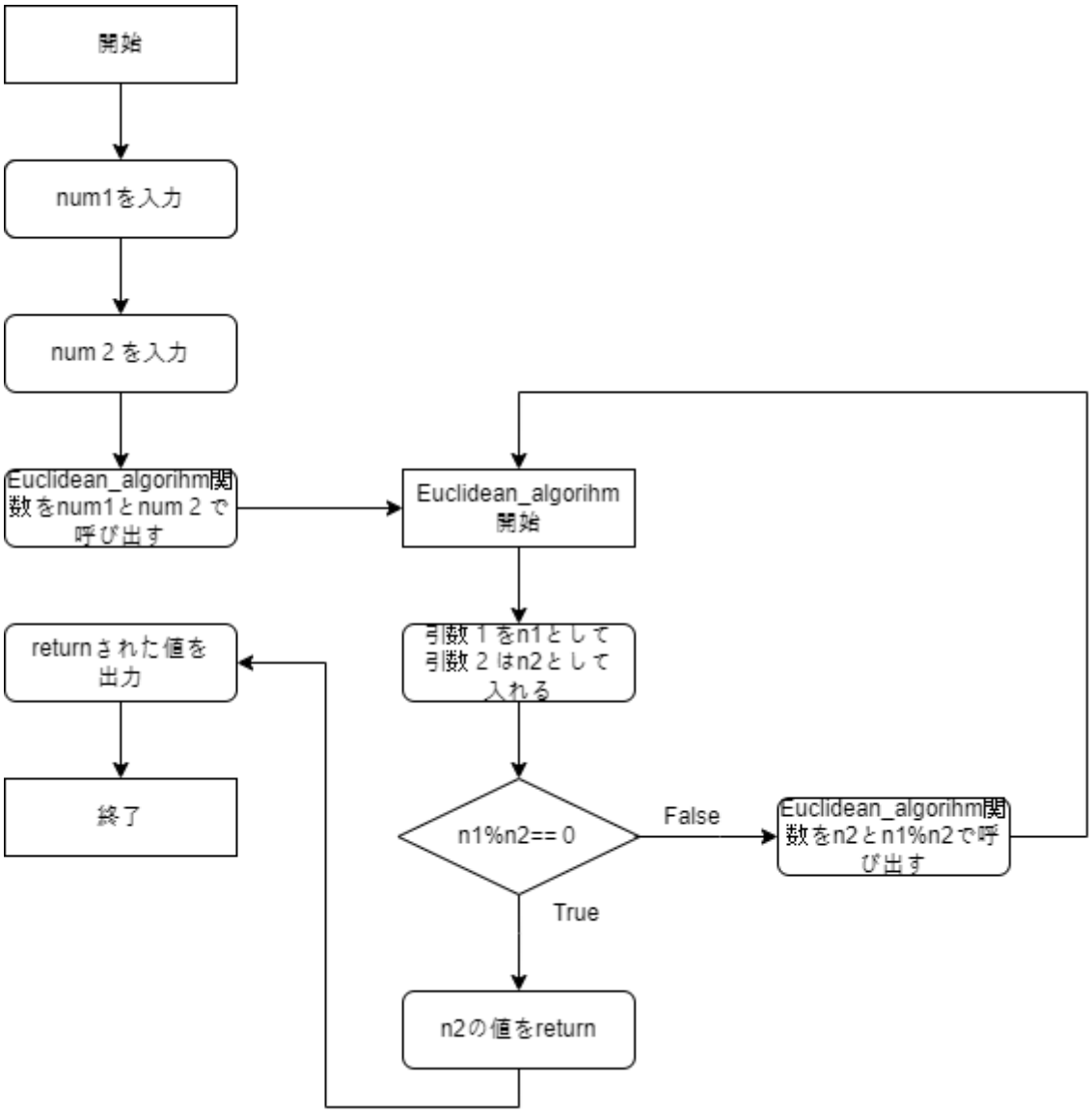


図 2。課題 2 フローチャート

C. 基本課題 3

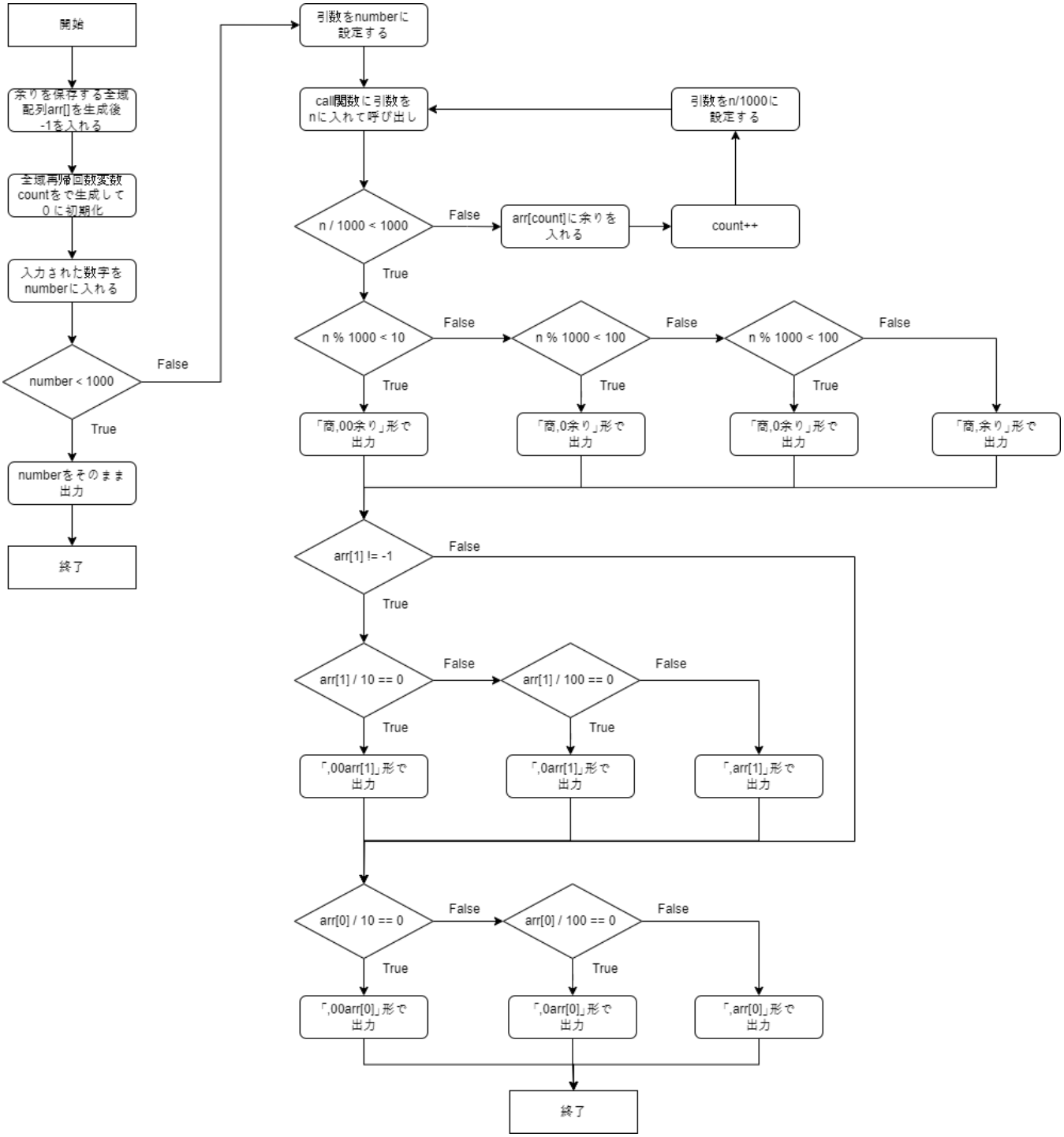


図 3。課題 3 フローチャート

D. 基本課題 4

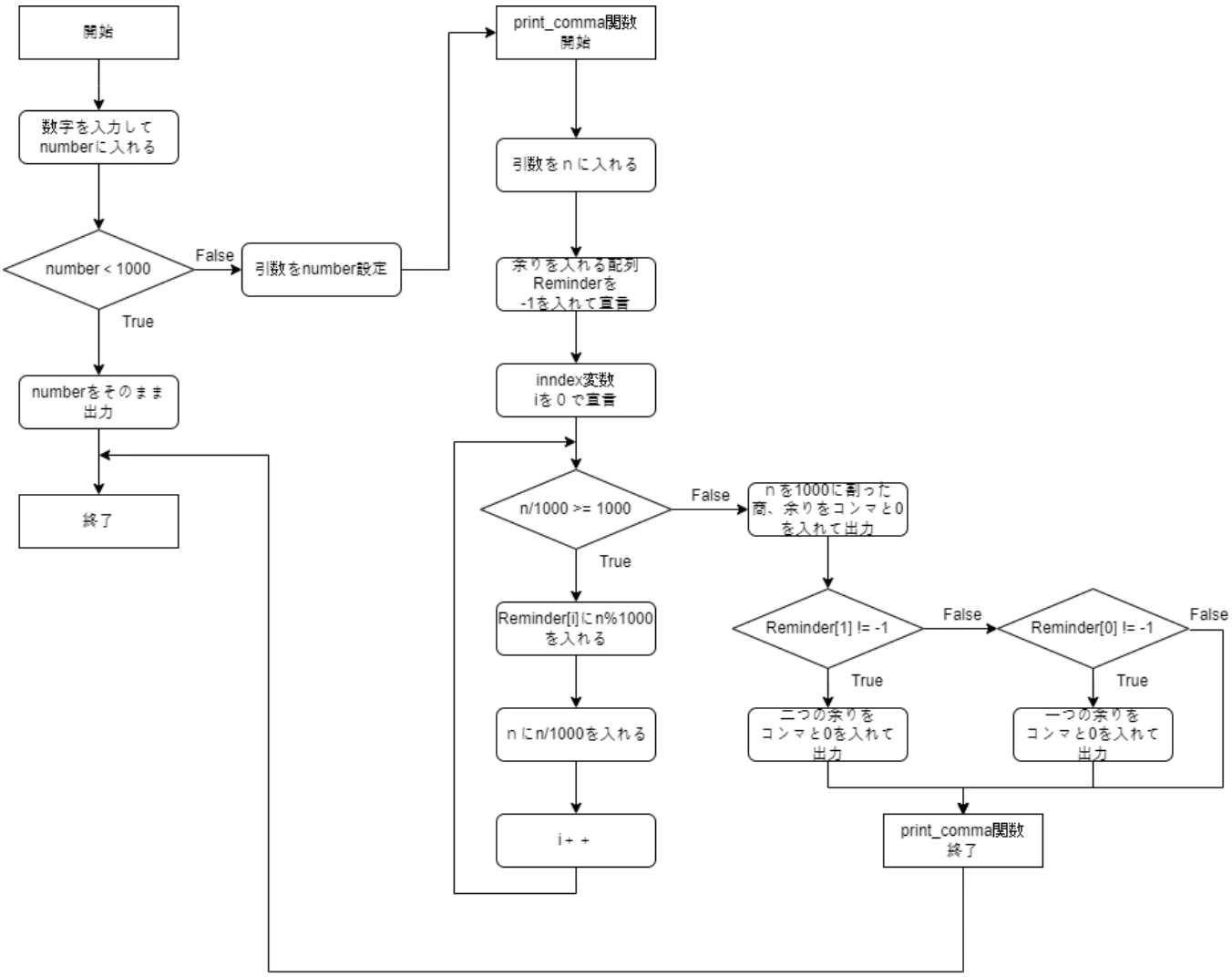


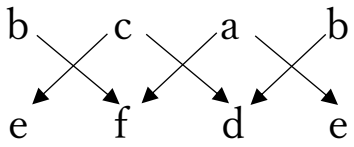
図 4。課題 4 フローチャート

(3) アルゴリズムが正しいことの説明

A. 基本課題１

i. 正当性

1. 理論的な外積の計算は　$\vec{A} = \{a, b, c\}$, $\vec{B} = \{d, e, f\}$ の場合次のような方法で求めることができる。



$$A \times B = \{bf - ce,\ cd - af,\ ae - bd\}$$

2. フローチャートでも書いているように外積の x の計算はbf − ceとして y の計算は cd − afとして z の計算はae − bdとして行ったのでこの計算結果は正当だと考えることができる。

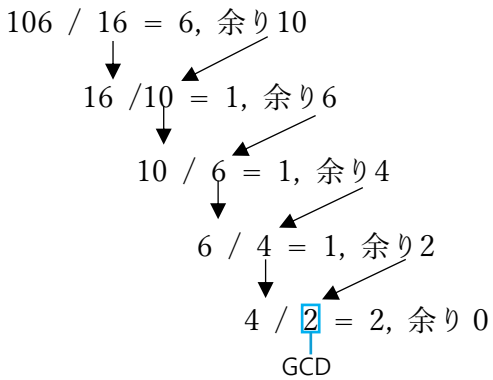
ii. 停止性

1. このプログラムでは反復処理を使ってないので無限ループに入ることはなくすべてのコードを実行すると停止することがわかるので。停止性も満足していると考えられる。

B. 基本課題２

i. 正当性

1. 理論的なユークリッドの互除法を考えてみると例えば、106と16の最大公約数を求めるときは



このような方法で求めることがユークリッドの互除法である。つまり

n1とn2の場合n1をn2で割ってn2をn1に入れて割った余りをn2に入れて余りが 0 になるまで続けて 0 になった瞬間のn2が最大公約数である。

2. 図2で確認できるようにEuclidean_algorihm関数で最大公約数を求める二つの数字を貰ってその余りが 0 であることを確認して 0 ではない場合はn1にn2を入れてn2に余りを入れる作業を余りが 0 になるまで繰り返している。よって、理論的なユークリッドの互除法と同じ作業を行うので正当だと言える。また、n1とn2に大きさが反対の場合は小さい数字を大きい数字で割ったので余りは小さい数字になるのでお互いに入れ替える作業になるので問題なく正しい結果が得られる。

ii. 停止性

1. この場合Euclidean_algorihm関数の中で自分の関数を呼び出しているので無限ループに入る可能性がある。しかし、制限を置いたので無限ループ入ることは起こらない。 図2を見ると毎回n1%n2が 0 になるかを確認している。n2はn1とn2の余りを入れるのでその値は繰り返す度に小さくなる。特に、余りが 0 にならなかった場合はn2が 1 になるまで小さくなるのでn2が1なのでn1とn2の余りは 0 以外の数字になることは起こらないので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ように設計されていると考えられる。

C. 基本課題 3

i. 正当性

1. 3桁の場合
   A. number＜1000の場合はそのまま出力するように設計されているのでコンマ無しで出力する。よって、正当な結果が得られる。
2. 3桁以上の場合
   A. 1000に割った商が3桁になるまで余りは配列に保存して商は続けて1000で割る作業をする。そして、商が3桁になったときは余りが3桁か2桁か1桁かを調べてそれに応じて 0 を追加して出力する。そして、再帰するたびに保存しておいた消えた余りを桁に合わせて 0 を追加して出力する。 nが2,147,483,647の場合再帰するたびに消える可能性たがある余りは483と647の二つのあまりである。よって、483（arr[1]），647(arr[0])順に出力する必要があるのでまずarr[1]が存在するかを確認して存在するとまた同じく

桁数に合わせてコンマを入れて 0 を追加して出力する。その後やarr[1]が存在しない場合はarr[0]が存在するか確認する。そして、arr[0]を桁数に合わせてコンマと 0 を入れて出力する。このような方法を使うと再帰で消えてしまう余りも出力することができるので正しくコンマが入った数字列が表現できる。よって、このアルゴリズムは正当だと言える。

ii. 停止性

1. この再帰関数では入力された数字を1000で割ったときにその商が3桁になるまで再帰する関数である。そして、3桁以上の場合は商を入れて関数を呼んでいる。そのため関数に入れる数字は再帰するたびに小さくなるのでいつか商が3桁になり正しく停止すると言える。

$$1回目：n$$

$$2回目：\frac{n}{1000}$$

$$3回目：\frac{n}{1000}\times\frac{1}{1000}$$

$$\cdots$$

$$n回目：\frac{n}{1000^n}$$

$$\lim_{n\to\infty}\frac{n}{1000^n}=0$$

D. 基本課題 4

i. 正当性

1. 3桁以下の場合
   A. 入力された数字が3桁かを確認してコンマを入れることなく出力するので正しい結果が得られる。
2. 3桁か以上6桁未満の場合
   A. この場合1000に割った商と余りにコンマを入れることで正しい結果が得られる。例えば、abcdefの数字の場合1000で割った商はabcになり余りはdefになる。よって、消える余りが存在しないのでそのまま商と余りを利用してabc,defの形で表現すると正当な結果が得られる。また、各数字が2桁や1桁の場合は0を追加して出力するようにすることで問題なくコンマを入れることができる。
3. 6桁以上の場合

A. この場合は繰り返し文に入って計算を行うので前回の計算の余りが消える可能性がある。1000に割った商の場合は次に繰り返しに引数として与えるので問題ないが余りは消えてしまう。よって、それを配列に入れることで保存して出力するときに適切に出力することで正しい結果が得られると思う。

ii. 停止性

1. この場合課題３と同じ条件n/1000　>=1000という1000に割った商が3桁以下になっているかを確認して繰り返しているのでｎの値は繰り返す度に小さくなるので無限ループに入ることはなく正しく停止すると考えられる。

(4) ソース・プログラムの説明

A. 基本課題 1

ソースコード

main.c

```c
#include <stdio.h>

struct vector{//3dimensions vector
    double x;
    double y;
    double z;
};

void outer_product(struct vector arr1[], struct vector arr2[]);//outer_product function

int main(void){
    struct vector v1;
    struct vector v2;
    printf("3次元ベクトルAのｘｙｚの値を入力してください。（ｘｙｚ）：");//input vector A value
    scanf("%lf %lf %lf", &v1.x, &v1.y, &v1.z);
    printf("3次元ベクトルBのｘｙｚの値を入力してください。（ｘｙｚ）：");//input vector B value
    scanf("%lf %lf %lf", &v2.x, &v2.y, &v2.z);
    printf("ベクトル A : %lf %lf %lf\n", v1.x, v1.y, v1.z);//print vector A
    printf("ベクトル B : %lf %lf %lf\n", v2.x, v2.y, v2.z);
    printf("ベクトルAとベクトルBの外積：");
    outer_product(&v1, &v2);

    return 0;
}

void outer_product(struct vector arr1[], struct vector arr2[]){
    printf("%lf ", (arr1->y * arr2->z) - (arr1->z * arr2->y));
    printf("%lf ", (arr1->z * arr2->x) - (arr1->x * arr2->z));
    printf("%lf ", (arr1->x * arr2->y) - (arr1->y * arr2->x));
}
```

実行結果

```
3次元ベクトルAの ｘ ｙ ｚ の値を入力してくださ  （ｘ ｙ ｚ）：2 3 5
3次元ベクトルBの ｘ ｙ ｚ の値を入力してください。（ｘ ｙ ｚ）：9 18 4
ベクトル A : 2.000000 3.000000 5.000000
ベクトル B : 9.000000 18.000000 4.000000
ベクトルAとベクトルBの外 : -78.000000 37.000000 9.000000

...Program finished with exit code 0
Press ENTER to exit console.
```

i. `ソースコードの説明

1. main関数を呼ぶ前に構造体の形式を宣言する。vectorという構造体を宣言して要素として座標が入るので座標の場合実数も入ることができるのでdouble型で宣言する必要がある。そして、main関数を呼び出して構造体v1とv2を宣言する。そして、Userから各ベクトルの要素を入力してもらう。その後確認として各ベクトルを出力する。そして、外積を計算して出力する予めプロトタイプで宣言した関数outer_productを呼び出す。outer_productの中では外積計算によって出てくるｘｙｚの要素を各自計算して出力する。

B. 基本課題２

ソースコード

main.c

```c
#include <stdio.h>

int Euclidean_algorithm(int n1, int n2);

int main(void){
    int num1, num2 = 0;

    printf("二つの数字を入力するしてください。\n");

    printf("一つ目の数字：");
    scanf("%d", &num1);

    printf("二つ目の数字：");
    scanf("%d", &num2);

    printf("%d と %d の最大公約数は%dです。", num1, num2, Euclidean_algorithm(num1, num2));

}

int Euclidean_algorithm(int n1, int n2){
    if(n1 % n2 == 0){//余りが０のとき
        return n2;
    }
    else{
        return Euclidean_algorithm(n2, n1%n2);
    }
}
```

実行結果

```
二つの数字を入力する◆てください。
一つ目◆数字：1024
二つ目の数字：16
1024 と 16 の最大公約数◆16です。

...Program finished with exit code 0
Press ENTER to exit console.
```

i. ソースコードの説明

1. ユークリッドの互除法の計算は関数で行うので予めプロトタイプとしてEuclidean_algorihm関数宣言する。そして、main関数の中でUserから二つの数字を入力してもらう。その後printfで最大公約数の値としてEuclidean_algorihm関数に二つの数字を入力した後に戻り値を出力するようにする。Euclidean_algorihm関数の中では無限ループに入らないようにするためのif文を利用してユークリッドの互除法は余りが０の時のn2の値が最大公約数なのでif文を利用して余りが０の時にn2の値をreturnするようにした。また、余りが０ではない場合はEuclidean_algorihm関数をn2の値と余りを入れて呼び出すことで繰り返して計算を作った。

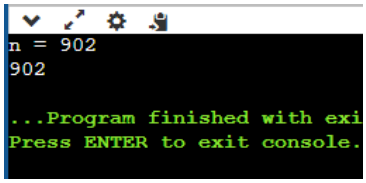
C. 基本課題３

ソースコード

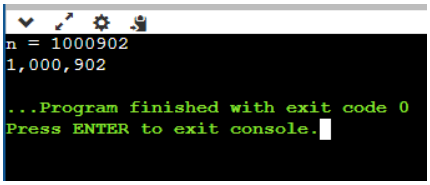
main.c

```c
#include <stdio.h>

static int arr[] = {-1, -1};//余りを保存する変数要素がない時の識別数字-1
static int count = 0;//何回再帰関数を読んだか確認する変数

int call(int n);

int main(void){
    int number = 0;

    printf("n = ");
    scanf("%d", &number);

    if(number < 1000){
        printf("%d", number);
        return 0;
    }
    else{
        call(number);
    }


}

int call(int n){
    if(n/1000 < 1000){
        if(n%1000 < 10){//余りが1桁の場合
            printf("%d,00%d", n/1000, n%1000);
        }
        else if(n%1000 < 100){//余りが2桁の場合
            printf("%d,0%d", n/1000, n%1000);
        }
        else{//余りが3桁の場合
            printf("%d,%d", n/1000, n%1000);
        }
        //もし、2,147,483,647の場合は
        if(arr[1] != -1){//483の部分
            if(arr[1]/10 == 0){//消えた余りが1桁の場合
                printf(",00%d",arr[1]);
            }
            else if(arr[1]/100 == 0){//消えた余りが2桁の場合
                printf(",0%d", arr[1]);
            }
            else{//消えた余りが3桁の場合
                printf(",%d", arr[1]);
            }
        }
        //647の部分
        if(arr[0]/10 == 0){
            printf(",00%d",arr[0]);
        }
        else if(arr[0]/100 == 0){
            printf(",0%d", arr[0]);
        }
        else{
            printf(",%d", arr[0]);
        }
    }
    else{//消える余りが存在するとき
        arr[count] = n%1000;//保存
        count++;
        return call(n/1000);
    }
```

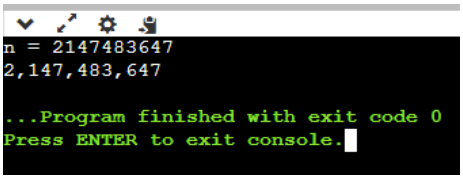
実行結果1



実行結果 2



実行結果3



i. ソースコードの説明

1. 再帰で消える余りを保存する配列arr[]を要素-1で宣言する。この時全域変数として宣言することでどこの関数でも参照できるようにする。また要素-1は配列が空いているかを確認する数字である。そして、再帰するたびに増えるcount全域変数を宣言する。Userから数字を入力してもらってすぐに入力された数字が1000より小さいのかを確認する。3桁の場合はコンマが必要ないからである。もし、1000以上の場合はcall関数に数字を入れて呼び出す。call関数では3桁ごとに調べる必要があるので1000に割った商が3桁以下なのか以上なのかをif文を利用して確認する。もし、3桁以上の場合はもう一回1000で割る必要があるのでもう一回1000に割った商を入れて再帰する。しかし、この時余りが消えるので余りを配列に入れてcount++をする。このようにして商が3桁になるまで繰り返して3桁になると一番最上位の桁なのでその桁の前にはコンマや2桁になって 0 が必要になったりしないのでそのまま出力してコンマを入れて余りを桁数をif文を利用して確認して 0 を加えて出力する。その後、消えていった余りを出力するために配列の要素が存在するか確認する。printfの場合左から順に出力するので後で配列に入った要素を先に調べる必要があるのでarr[1]から存在を確認して桁数に合わせて出力する。その後、arr[1]が存在するとarr[0]は必ず存在するのでarr[0]は存在を調べることなく桁数に合わせて出力する。

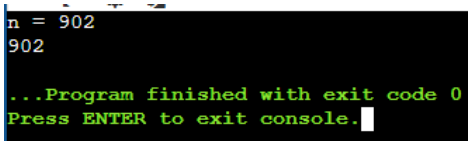
D. 基本課題４

ソースコード

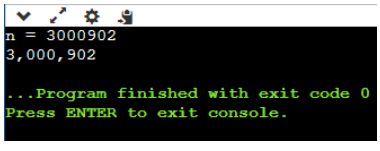
main.c

```c
#include <stdio.h>

void print_comma(unsigned int n);

int main(void){
    unsigned int number = 0;

    printf("n = ");
    scanf("%u", &number);
    if(number < 1000){//入力された数字が3桁の場合
        printf("%u", number);
    }
    else{//3桁の以上の場合
        print_comma(number);
    }
    return 0;

}
void print_comma(unsigned int n){
    int Remainder[] = {-1, -1};//余りを保存する配列

    for(int i = 0; n/1000 >= 1000; i++){//余りを保存する作業
            Remainder[i] = n%1000;
            n /= 1000;
    }
    printf("%d,%03d", n/1000, n%1000);//最上位の桁と余りを出力

    if(Remainder[1] != -1){//消えた余りが二つの場合の出力
        printf(",%03d,%03d", Remainder[1], Remainder[0]);
    }
    else if(Remainder[0] != -1){//消えた余りが一つの場合の出力
        printf(",%03d",Remainder[0]);
    }

}
```

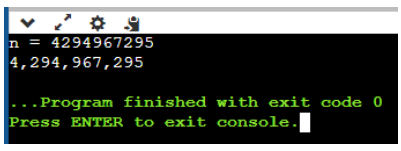
実行結果1



実行結果２



実行結果３



i. ソースコードの説明

1. この課題の n の範囲はn<=4294967295である。この値はint型の表現できる数字を超えているためただのint型ではなくunsigned int型で変数を宣言して数字を入れる必要がある。よって、Userからもらう数字をunsigend int型のnumberに入れる。そして、入力された数字が3桁以下であるかを確認するためにif文をでnumber＜1000条件を満たした場合コンマを入れることなくそのまま出力するようにコードを作成した。もし、3桁以上の場合はプロトタイプ宣言しておいたprint_comma関数にnumberを入れて呼び出しをする。print_comma関数を宣言する際にnumberと同じくint型で宣言すると引数を渡す際に問題が生じるのでunsigend int型で宣言する。そして、引数としてもらった数字を n に入れて繰り返し文を利用して最上位の桁の数字を探索する。3桁以上6桁未満の場合は繰り返し無しで割り算と余りの計算でコンマを入れて出力することができるので繰り返し条件はn/1000 >= 1000にする必要がある。つまり、1000で割った商が4桁以上の場合は場合である。そして、繰り返しながら余りを配列に入れる。配列は何も入らなかったことを認識するように識別コード-1によって初期化されている。そして、最上位の数字が判明した場合最上位の商と余りはコンマと0を加えて出力して配列に保存されている残りの余りを出力する。この時printfは左から出力されるので配列の一番最後に入った数字から出力する必要がある。よって、1番目から存在を確認して出力する。この際1番目が存在すると 0 番目もまた存在するのでそのまま 0 番目も出力する。そして、1番目が存在しない場合は 0 番目の存在を確認して確認して出力する。

(5)考察

A. 基本課題 1

i. この外積の計算ではｘｙｚの要素の計算を一つ一つ計算をして出力をしたが、この計算は同じ作業の反復なのでfor文を利用して作ることが可能だと思う。しかし、この場合構造体の要素を参照することが連続的ではない為for文を利用して作ることは余計な条件が付くので非効率的である。よって、この外積計算は公式に従って直接該当する構造体の要素を呼び出す必要がある。また、ｘｙｚの要素を単純な配列に入れることもまた可能であるが、構造体を利用した理由としては配列を使った場合は何番目の要素の意味していることが分からなくなるのでコードの作成に困難が生じる可能性があるので入力した数字がどのような意味の変数に入ったかを確認できる構造体のほうがよりいいと思う。

B. 基本課題 2

i. ユークリッドの互除法を反復処理ではなく再帰関数を利用して作ったことの利点について考えたいと思う。ユークリッドの互除法では前回の計算結果の余りを次の計算で使うので前回の計算との連動性が必要である。このような処理を反復処理を利用して作るときには前回の計算結果である余りを入れる変数が必要である。しかし、再帰関数を利用すると前回の計算結果を引数として次の計算で利用することができる。また、この時に他の変数は必要としない。前回の計算の余りは全体的のプログラムで見ると計算のために使われる変数で必須ではない。また、変数を宣言するとメモリーを使うことになるので変数の宣言を減らせる方法があると減らした方がより効率的である。よって、余りを次の計算に送る必要があるユークリッドの互除法では再帰関数を使うことがより効率的だと考える。また、変数の数を減らせるのでメモリー活用的な面でも効率的だと考える。

ii. また、ユークリッドの互除法を利用することで得られる計算の効率性を考えたいと思う。もし、ユークリッドの互除法を利用することなく最大公約数を求めるときには二つの数字の約数を全部求めてそこから同じ約数を見つけてその中で一番大きい公約数を見つける必要がある。このような方法で最大公約数を求めるとたくさんの反復処理が必要であるため計算結果が出るまでの時間が長くなる可能性がある。しかし、ユークリッドの互除法アルゴリズムを利用すると約数を求める必要も公約数を見つける作業も公約数の中で一番大きい公約数を見つける作業も必要ないので計算の速度は比べものにならないと思う。よって、ユークリッドの互除法は最大公約数を求めるときにとても効率的な計算方法だと考えられる。

C. 基本課題 3

i. 再帰関数を利用して得られる利点について考えたいと思う。課題 2 でも言っているように再帰関数の場合次の計算との連動性が長けているのでこのような次の再帰関数の引数として前回の割った商を渡すのでより直観的なコードが作れる。

よって、この問題の場合は次に計算結果を渡す必要があるので繰り返し処理より再帰関数を利用した計算がより効率的だと考える。また、この問題は桁数に合わせて0を追加して出力するコードを作成したが、%03dを利用することでより簡単なコードを作れるので。桁数に合わせて0を追加するときは%[追加したい文字][何桁]dを利用することがより効率的である。そして、そのようなコードを作成すると無駄に長いコードが短くなるので読みやすくなるので効率的だと考える。

D. 基本課題4

i. 基本課題4は基本課題3の再帰関数を繰り返し文に変えて入力できる数字の範囲が2倍に増えている。基本課題3では再帰関数を利用するので繰り返す度に関数を呼び出している。よって、関数の中だけで使う配列を使うことが困難である。よって、課題3では全域配列と全域変数を利用して余りの保存と余りのインデックスを保存していた。しかし、繰り返し文を利用すると繰り返す度に新しい関数になることはなく関数の中だけで使える変数の宣言が可能である。よって、課題4のprint_comma関数を見ると関数の中で配列を宣言して出力してprint_comma関数が終了すると消えるように設計されている。これはメモリー空間を効率的に使える方法だと思う。また、配列以外にも課題3で使われたcount全域変数も必要なくなるのでこの問題では再帰関数より繰り返し文を利用することがより効率的だと言える。

ii. また、課題4は課題3に比べて入力できる数字の範囲が2倍である。これはint型とunsigend int型の違いだと考えられる。int型とunsigend int型は両方とも4バイトのデータ型である。4バイトのデータ型で表現できる数字は-2,147,483,648 ~ 2,147,483,647である。しかし、課題3と4は自然数を入力するので-2,147,483,648から-1までの数字が使われない。unsigend int型は負数の部分は表現することなく自然数だけを表現するのでint型の無駄な負数の部分がなく正数の部分だけになっている。これによって考えられるunsigendの意味は符号を取り除く意味を含めていると考えられる。つまり、一番最上位のビットは符号ビットとして使われているがそのビットを取り除くことで符号が無くなり保存できる数字の範囲が負数の範囲が加われ2倍になっていると考えられる。よって、このような自然数を入力する問題ではただのint型ではなくunsigend int型で変数を宣言するほうが保存できる範囲も2倍になるのでより効率的だと考えられる。それ以外に、範囲を増やしたいと思うなら入力を数字ではなく配列で入力してもらうことでより多くの数字が表現できると思う。つまり、入力された数字を数字ではなく文字列として配列にいれることである。このような方法を使うと配列の大きさが表現できる桁数になるのでより多くの数字を表現することができると思う。しかし、3桁ごとにコンマを入れる作業がより複雑になる可能性があるが表現できる数字の範囲で見るとより効率的なコードになると考えられる。

(6) 感想

A. 基本課題 1

i. 普段知っていた外積を実際にプログラムを利用して求めることができたので数学的な理論のプログラムとして作る方法がわかるようになった。また、ネットにある外積計算機もまたこのような方法で作られたプログラムであることを知ってプログラムの仕組みついてわかるようになった。

B. 基本課題 2

i. ユークリッドの互除法アルゴリズムを利用して少ない計算で最大公約数を求めることができるのでアルゴリズムの効率性を学ぶことができた。また、同じ反復処理であっても再帰関数と反復文の違いについて考えることができた。

C. 基本課題 3

i. 桁数に合わせて 0 を表現するときには%03dという方法があることを知った。また、再帰関数を使うことで得られる利点について考える機会になった。

D. 基本課題 4

i. unsigendの意味が符号ビットを取り除く役割であることを知った。そして、その影響として保存できる数字の範囲が2倍になっていることを確認でき、条件によって変数の宣言仕方に注意をするようになった。