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 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。

(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

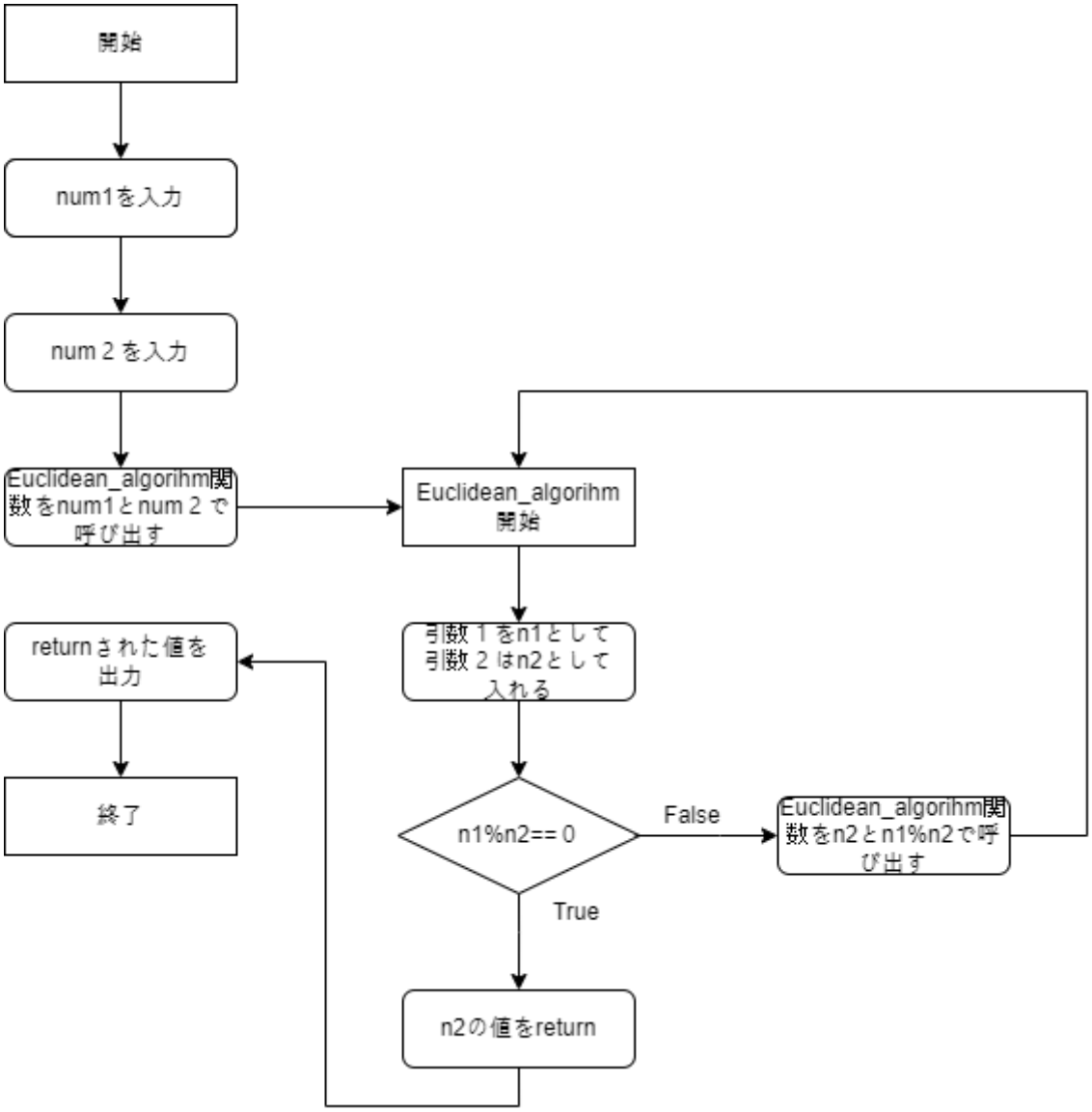
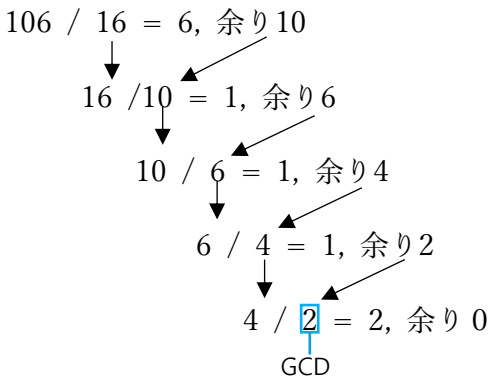


図 1。フローチャート

(3) アルゴリズムが正しいことの説明

A. 正当性

i. 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。

ii. (2)のフローチャート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。

B. 停止性

i. 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。フローチャートを見ると毎回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。

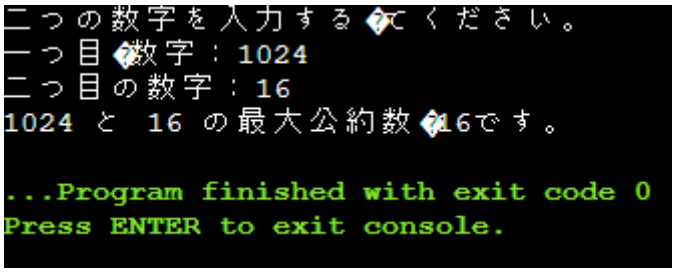
(4) ソース・プログラムの説明

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＞



A. ソースコードの説明

i. 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。

(5) 考察

A. 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。

B. 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。

(6) 感想

A. 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。