提出日：2024/6/7

プログラミング演習　第８回演習レポート

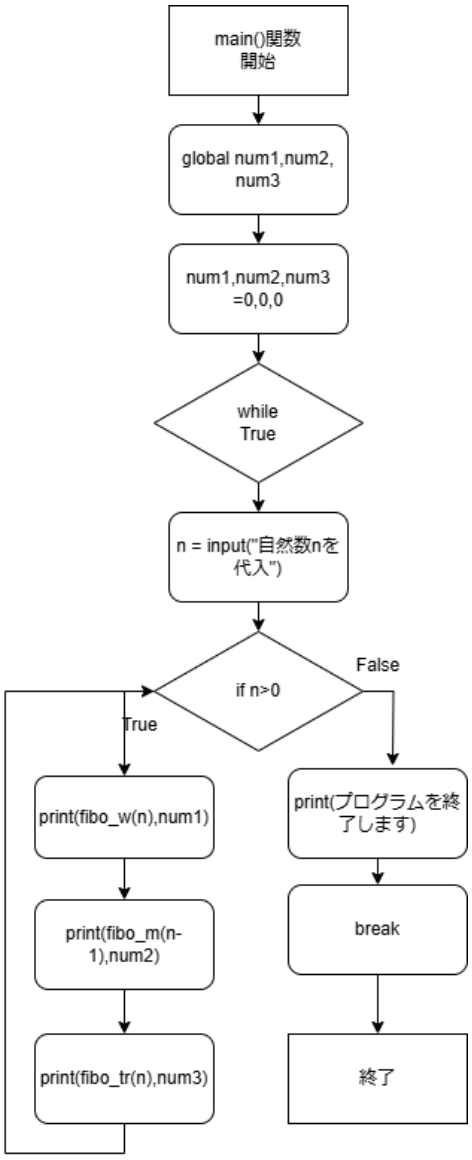
担当教員：杉本　千佳先生
所属：理工学部　数物・電子情報系学科
電子情報システムEP
学年・クラス：２年　Fe1
学籍番号：2364092
氏名：熊田　真歩

（1）課題番号：基本課題4

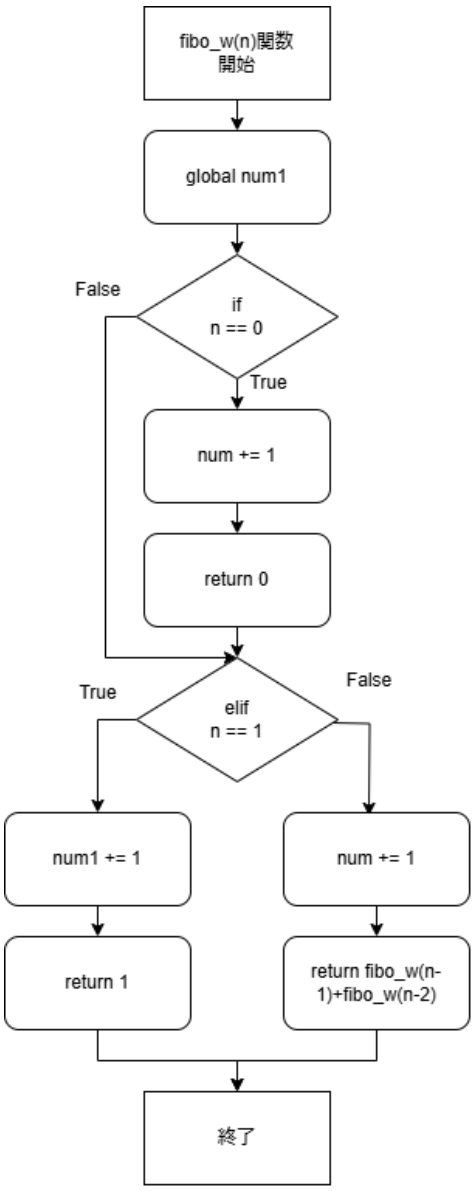
課題名　：フィボナッチ数を求める

（2）プログラムのフローチャート（関数ごと）

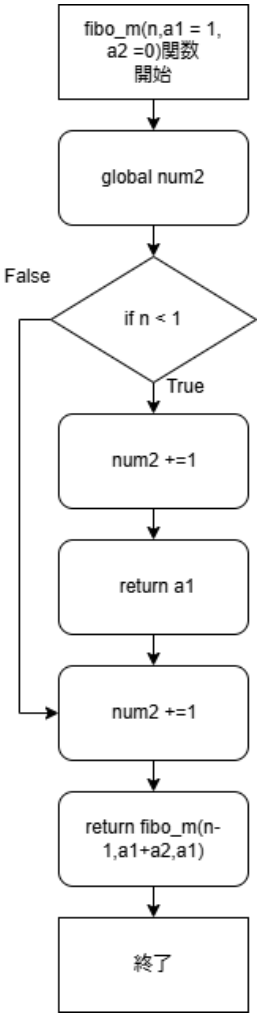
・main（）関数



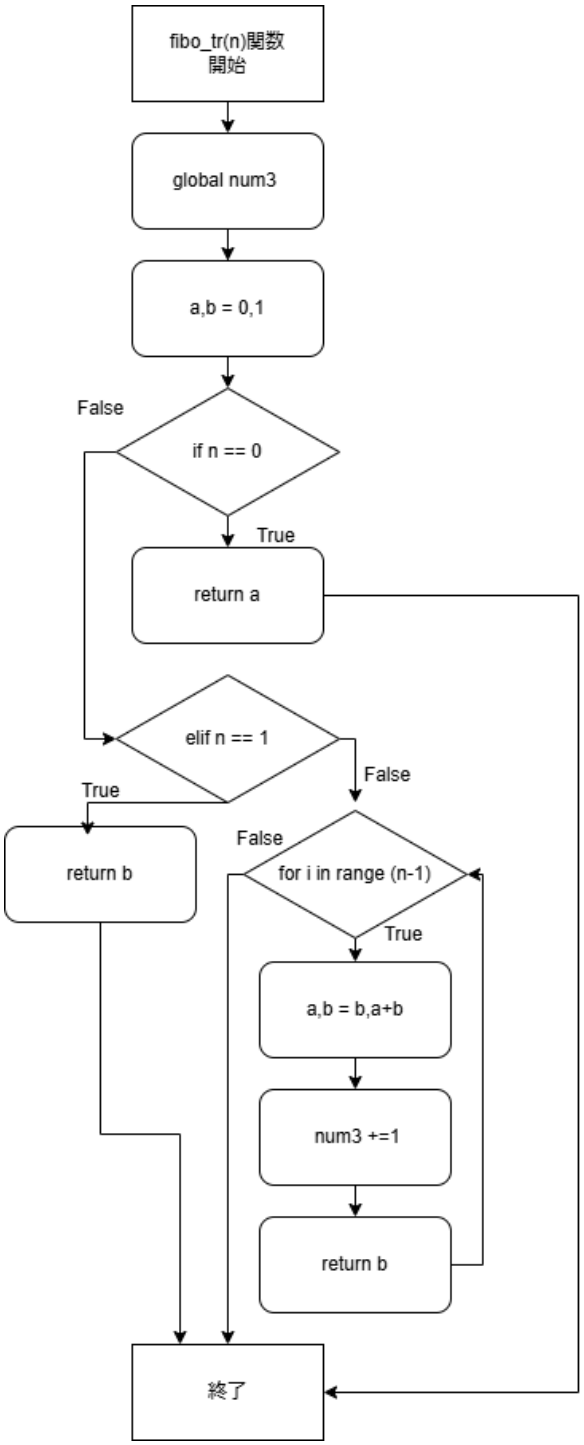
・fibo_w(n)関数



・fibo_m(n,a1=1,a2=0)関数



・fibo_tr(n)関数



（3）アルゴリズムが「正しいこと」である証明

本プログラムでは、二重再帰関数、末尾再帰関数、ループ関数の 3 つの関数を用いて、代入した自然数 n に対してフィボナッチ数列の第 n 項の値を出力し、その際の関数の呼び出しまたは繰り返し数を出力するというプログラミングである。ここで、全ての入力に対してプログラミングは停止することについて考察する。前提条件として本プログラムで入力する n の値は整数値であるする。n にどんな自然数つまり、1 以上の整数を入力してもプログラムは正常に稼働し、再び次の n の値を求める。ここで n にどんな 0 または負の整数値を入力してもプログラミングはメッセージを出力し、終了する。

次にどんな整数値 n を代入してもプログラムは正しい答えを出力することについて、自然数に対しては 3 つのいずれの関数ともある n に対し同じ値を出力しこれはフィボナッチ数列の第 n 項の値に等しい。また、n が 0 以下の値の場合は常に処理が終了することから求める動作を正常に行っているプログラミングと言って問題ない。

したがってある整数値 n に対し、このアルゴリズムは正しいことが示された。

（4）ソース・プログラムの説明

```
def main():        #main()関数の定義
  global num1,num2,num3   #3 つのグローバル変数の定義
  num1,num2,num3=0,0,0    #繰り返し数の初期値は 0
  while True:             #条件を満たす時成立のループ
    n = int(input("¥n 正数 n の値を入力してください(負の数の入力で終了)n="))#n に自然数を代入
    if n > 0:          #n が整数の時
      print("二重再帰:計算結果:",fibo_w(n),"呼び出し回数:",num1)#二重再帰関数の呼び出し
      print("末尾再帰:計算結果:",fibo_m(n-1),"呼び出し回数:",num2)#末尾再帰関数の呼び出
し
      print("繰り返し:計算結果:",fibo_tr(n),"繰り返し回数:",num3)#繰り返し関数の呼び出し
    else:              #n が 0 または負の値の時
      print("プログラムを終了します")   #メッセージを出力
      break          #プログラムを終了
def fibo_w(n):#二重再帰関数
  global num1 #グローバル変数 num1
  if n ==0:    #n の値が 1 の時
    num1+= 1   #呼び出し回数＋1 回
    return 0   #値を返さない(0 を返す)
  elif n == 1:#n の値が 1 の時
    num1+= 1   #呼び出し回数＋1 回
    return 1   #1 を返す
  else:        #n が 0 でも 1 でもない時
```

```
    num1+= 1   #呼び出し回数＋1 回
    return fibo_w(n-1)+fibo_w(n-2)  #fibo_w(n-1)+fibo_w(n-2)を返す


def fibo_m(n,a1 = 1,a2=0):#末尾再帰関数
  global num2  #グローバル変数 num2
  if n < 1:    #n が 1 未満の時
    num2 +=1   #呼び出し回数＋1 回
    return a1  #a1 を返す
  num2 += 1    #呼び出し回数＋1 回
  return fibo_m(n-1,a1+a2,a1) #fibo_m(n-1,a1+a2,a1)を返す


def fibo_tr(n):#ループ関数
  global num3  #グローバル変数 num3
  a,b = 0,1    #a,b=0,1 と初期値設定
  if n == 0:   #n の値が 0 の時
    return a   #a を返す
  elif n == 1: #n の値が 1 の時
    return b   #b を返す
  else:        #n の値が 0 でも 1 でもない時
    for i in range(n-1):#n-1 までの i に対して
      a,b = b,a+b       #a に b の値、b に a+b の値を代入
      num3 +=1          #繰り返し回数＋1 回
    return b            #b に値を返す


main()      #main()関数の実行
```

（5）考察

例題 6.1 分割統治法でフィボナッチ数列の第4 項を求める図を完成させよ

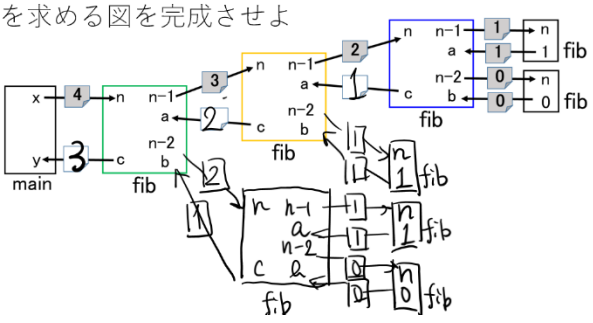


図 1．基本課題 1 フィボナッチ数列の第 4 項

ここではまず、課題の指示通り、３つの関数で採用したアルゴリズムに対し呼び出し回数あるいは繰り返し処理の回数が異なる理由について考察する。上に添付した図、およびプログラムから考察する。はじめに、二重再帰関数について二重再帰関数では繰り返し処理の回数は他の関数に比べて多くなる。これは、fibo_w(n)はfibo_w(n-1)と fibo_w(n-2)を呼び出し、これらの呼び出しはそれぞれ fibo_w(n-2)と fibo_w(n-3)を呼び出すことにより何度も関数を繰り返し処理するためである。つまり、この呼び出し回数は指数関数的に増加する。

次に末尾再帰関数について、末尾再帰関数は変数の設定は多い分、その繰り返し関数は少ない。これはこの 2 変数の効果により同じ計算を避けることが可能であるため、n に比例する呼び出し回数となる。つまり、指数関数的に増加する二重再帰関数に比べ、比較的少ない再帰で処理を完結できる。

最後にループ関数では繰り返す数は n が 1 の時を除き、n -1 回である。これは関数内の for 分の繰り返し範囲が n -1 と設定しているためである。これは、再帰関数でいうと末尾再帰関数の処理回数に近く、効率的な関数といって問題ないだろう。

このように、結果として同じ処理を行うプログラムでもそのアルゴリズムには大きな差が生じる。プログラムを書く際は考えられるアルゴリズムの中でどれが最適なのか熟考する必要がある。これはプログラムが複雑化するほど重要な作業となる。処理数が増えると、一つ一つのアルゴリズムで採用した処理の少なさが大きな差となってくるからだ。また、このように一つの関数のみに注目しても、指数関数的に処理数が増加するものとそうでなくデータ数の処理数で済むものであれば、値が大きくなった時に計り知れない差となることが考えられる。したがって、常にプログラムの処理回数とその時間についての考察は欠かせない項目である。

次に本プログラムは、アルゴリズムが正しいことの証明にもある通り、n に整数値以外の値が入力されることを想定して作成されてたものでない。しかしながら、このプログラムを更に良いものにするためには、n に整数値でない値が入力された場合に四捨五入して整数値に変換してから処理を行ったり、エラーメッセージを出力し再度整数値の入力を促したりするなどの工程があると更に良いプログラムとなったという改善の余地はある。

（６）参考文献

アルゴリズム授業スライドおよびサンプルプログラム

（７）

三つの関数を利用して同じ処理をすることでアルゴリズムの採用の重要性を身に染みて感じることができた。単に最初に思いついたからこのアルゴリズムを採用しようとしていたがこれでは全く良いプログラムを書くことはできないことを実感した。今後プログラムを書く時は自分で考えられるすべての選択肢の中でえ最も処理回数の少ないと考えられるものを採用することが不可欠であると実感した。しかしながら、この方法を実行できるという事はそれだけたくさんのアルゴリズムを思いつく経験と能力が必要である。これらを身に付ける重要性をより強く感じることのできる良い学びとなった。