プログラミング演習I第2回基礎課題３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. 課題番号：基本演習課題３

B. 課題内容

i. 指数関数 exp(x) [eのx乗]を、級数展開「$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!} + \cdots$」を使って求めるプログラムの作成。キーボードから入力した数字を x として級数展開の公式に入れる。また、n 項が 0.0000001（ 1e-7 ）より小さくなったところで計算を打ち切るようにする。 各項ごとに「次項 = 現項 *（ x/n)」で計算する。 x を（ double ）型で宣言し、絶対値をとる数学関数 fabs（ ）で x の絶対値をとり級数展開の公式を計算し、x が負の時は、1/exp（ |x| ）を解としなさい。出力の書式 %.8lf または %.8le など小数点以下 8 桁まで出力するよう指定する。

(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. 次のページ図1に記述している。

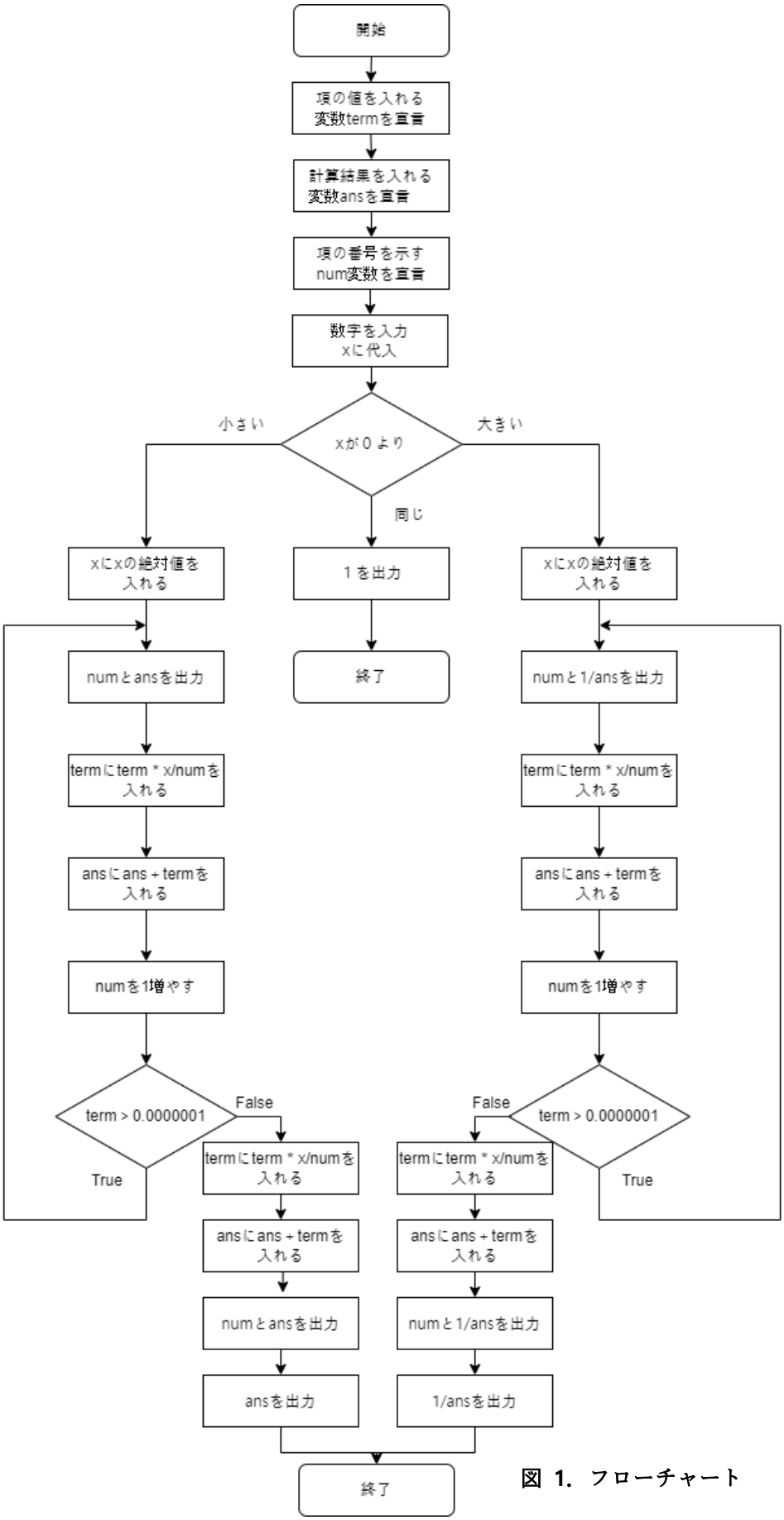
開始
項の値を入れる
変数termを宣言
計算結果を入れる
変数ansを宣言
項の番号を示す
num変数を宣言
数字を入力
xに代入
xが０より
小さい
同じ
大きい
xにxの絶対値を
入れる
１を出力
xにxの絶対値を
入れる
終了
numとansを出力
numと1/ansを出力
termにterm * x/numを
入れる
termにterm * x/numを
入れる
ansにans + termを
入れる
ansにans + termを
入れる
numを1増やす
numを1増やす
term > 0.0000001
False
True
termにterm * x/numを
入れる
ansにans + termを
入れる
numとansを出力
ansを出力
term > 0.0000001
False
True
termにterm * x/numを
入れる
ansにans + termを
入れる
numと1/ansを出力
1/ansを出力
終了

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. 今回の課題の目標はe^xの値を求めるために級数展開「$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!} + \cdots$」を使って求めるプログラムの作成することである。答えとしては項が0.0000001より小さい値になるまで級数展開の計算を行って0.0000001より小さくなった項を一回足した値を求める。

1. 項の計算の正当性

A. 検証

$$a_1 = a_0 \times \frac{x}{1}$$
$$a_2 = a_0 \times \frac{x}{1} \times \frac{x}{2}$$
$$a_3 = a_0 \times \frac{x}{1} \times \frac{x}{2} \times \frac{x}{3}$$
$$\cdots$$
$$a_n = a_0 \times \frac{x}{1} \times \frac{x}{2} \times \ldots \times \frac{x}{n-1} \times \frac{x}{n}$$

B. 一般項が $\frac{x^n}{n!}$ であるので $\frac{x^n}{n!} = \frac{x}{1} \times \frac{x}{2} \times \ldots \times \frac{x}{n-1} \times \frac{x}{n}$ を計算する必要がある。しかし、各項ごとにべき乗計算をするのは非効率的なので 次項 = 現項 $\times \frac{x}{n}$ を利用して計算できることは検証で証明したように 次項 = 現項 $\times \frac{x}{n}$ が一般項が同じなので正当である。

2. 級数展開の正当性

A. 入力された数字（x）の符号によって違う計算を行うので、計算は x の絶対値で行って符号によって違う出力をすることで正当な計算結果が得られる。

i. 入力された数字（x）が陽数の場合

1. 項の計算「term＝term×(x/n)」と 級数展開の計算「ans＝ans＋term」をして回数を1増やすそれからtermの値が0.0000001より大きい場合は項の計算と級数展開の計算に戻る。そしてtermの値が0.0000001より小さいときはもう一回足して終わりにすると問題で求めているe^xが得られる。

ii. 入力された数字（x）が負数の場合

1. $e^{-x} = \frac{1}{e^x}$であるので陽数と同じ計算で計算結果を1/ansとするとe^xが得られる。

iii. 入力された数字（x）が0である場合

1. 理論的に$e^0 = 1$であるので特別な計算を行うことではなく1としてe^xが得られる。

B. 停止性

i. $次項 = 現項 \times \frac{x}{回数}$ の計算は回数が増えることにつれて項の値は小さくなる。よって、条件式term＞0.0000001を満足するときに繰り返して計算を行うので回数が増えると0.0000001より小さい値になりループは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

ソースコード

main.c

```c
#include <stdio.h>
#include <math.h>

int main(void){
    double x = 0;
    double ans = 1.0;
    double term = 1;
    int num = 1;

    printf("input the number : ");
    scanf("%lf", &x);

    if(x > 0){
        x = fabs(x);
        do{
            printf("%d %.8lf\n",num, ans);
            term *= x/(double)num;
            ans += term;
            num++;
        }while(term > 0.0000001);
        printf("%d %.8lf\n",num, ans);

        printf("ans = %.8lf", ans);
    }
    else if(x < 0){
        x = fabs(x);
        do{
            printf("%d %.8lf\n",num, (double)1/ans);
            term *= x/(double)num;
            ans += term;
            num++;
        }while(term > 0.0000001);
        printf("%d %.8lf\n",num, (double)1/ans);

        printf("ans = %.8lf", (double)1/ans);
    }
    else if(x == 0){
        printf("ans = %.8lf", ans);
    }
    else{
        printf("Error!");
        return 0;
    }
```

実行結果

```
input the number : 3
1 1.00000000
2 4.00000000
3 8.50000000
4 13.00000000
5 16.37500000
6 18.40000000
7 19.41250000
8 19.84642857
9 20.00915179
10 20.06339286
11 20.07966518
12 20.08410308
13 20.08521256
14 20.08546859
15 20.08552346
16 20.08553443
17 20.08553649
18 20.08553685
19 20.08553691
ans = 20.08553691

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

```
input the number : -3
1 1.00000000
2 0.25000000
3 0.11764706
4 0.07692308
5 0.06106870
6 0.05434783
7 0.05151320
8 0.05038690
9 0.04997713
10 0.04984202
11 0.04980163
12 0.04979062
13 0.04978787
14 0.04978724
15 0.04978710
16 0.04978707
17 0.04978707
18 0.04978707
19 0.04978707
ans = 0.04978707

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

A. 説明

```
#include <stdio.h>
#include <math.h>   絶対値計算を行うためのライブラリ

int main(void){
    double x = 0;   入力された数字を入れるための変数 0 に初期化
    double ans = 1.0;   計算結果を入れるための変数。1項の値が1なので1に初期化
    double term = 1;   項の計算結果を入れるための変数。同じく、1に初期化
    int num = 1;   項の番号入れるための変数

    printf("input the number : ");
    scanf("%lf", &x);

    if(x > 0){
        x = fabs(x);   x の値を絶対値に変換
        do{
            printf("%d %.8lf¥n",num, ans);   1項を出力するために先に出力
            term *= x/(double)num;   項の計算次項に現項×x/numを入れる
            ans += term;   計算結果の計算各項を足した値
            num++;   次の項に移動するための操作
        }while(term > 0.0000001);   項の値が0.0000001より大きい条件
        printf("%d %.8lf¥n",num, ans);   0.0000001未満になった値をもう一回出力。

        printf("ans = %.8lf", ans);   計算結果
    }
    else if(x < 0){
        x = fabs(x);
        do{
            printf("%d %.8lf¥n",num, (double)1/ans);   負数であるため結果を1/an
sする
            term *= x/(double)num;
            ans += term;
            num++;
        }while(term > 0.0000001);
        printf("%d %.8lf¥n",num, (double)1/ans);

        printf("ans = %.8lf", (double)1/ans);
```

```
        }
        else if(x == 0){   xの値が 0 であるときには理論的に1であるため 1 を出力
            printf("ans = %.8lf", ans);
        }
        else{   理論的に有り得ない条件になるとエラーと出力してプログラム停止するようにする条件
            printf("Error!");
            return 0;
        }
        return 0;
    }
```

(5) 考察

A. do~whileを利用した理由

i. 反復処理の場合for文、while文、do~while文三つがある。この場合項の値が0.0000001小さいときにループを脱出して未満になった値を最後に足して終わりにする必要がある。for文とwhile文の場合繰り返しを行う前に条件に満たしているか確認して繰り返し文の中のコードを実行する。しかし、そのような方法では0.0000001未満になった項の計算を行うことができない。よって、先ずは実行してから繰り返すか確認するdo~while文の方が0.0000001未満の値を足した値を求めるのに適合していると考えられる。

B. 絶対値で計算を行う理由

i. 今回の課題の目的はexp(x)の値を求めることである。そのために級数展開を利用する。しかし、xの値が負数の場合級数展開計算の中で負数が結果に影響を及ぼすのでexp(x)で x が負数の場合の理論を考えると陽数の場合と同じ計算を行って最後に計算結果を（1/答え）することで正しい結果が得られるので両方同じく絶対値を取り計算を行っていると考えられる。

C. 項の計算

i. 項の計算の場合 $\frac{x^n}{n!}$ であるが毎回項の計算を行うと非効率的であるために、前回の項の計算結果を利用することでより簡単な計算を行うことができる。式の性質上前の項の計算結果にx/nをかけるだけなのでより簡単で素早く計算を行うことができる。

(6) 感想

A. 数学的な数値をプログラムを利用して求めることで数学という学問とプログラムの関連性についてよくわかるようになった。また、効率的なコードの作成の仕方を学んだ。そして、反復処理の条件によって使えい分けることができた。