プログラミング演習I第3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. 課題1番号：基本演習課題1

B. 課題1番号：基本演習課題2

C. 課題1内容

i. 繰り返し処理を使って 1, 2, ..., n の 2 乗和 $1^2 + 2^2 + \cdots + n^2$ を計算する。

ii. 2 乗和を求める部分は関数にし、引数 n を受け取ると2 乗和の計算結果を戻り値とする関数にする。

D. 課題2内容

i. 入力された n が 負でない間は、関数で 2 乗和を計算し計算結果を出力するようループを形成。

ii. n に負の値が入力されたらプログラムを終了。

iii. 最後に呼ばれた際の引数と戻り値を記憶しておく(static 宣言を利用する)。

iv. 新たに呼び出されたとき、直前の呼び出しと引数の値が同じ場合、計算をせずに記憶しておいた値を戻り値とする。

v. 再計算の必要があったかどうかを関数内でメッセージとして出力

(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. 課題1は2ページの図1に記述している。

B. 課題2は3ページの図2に記述している。

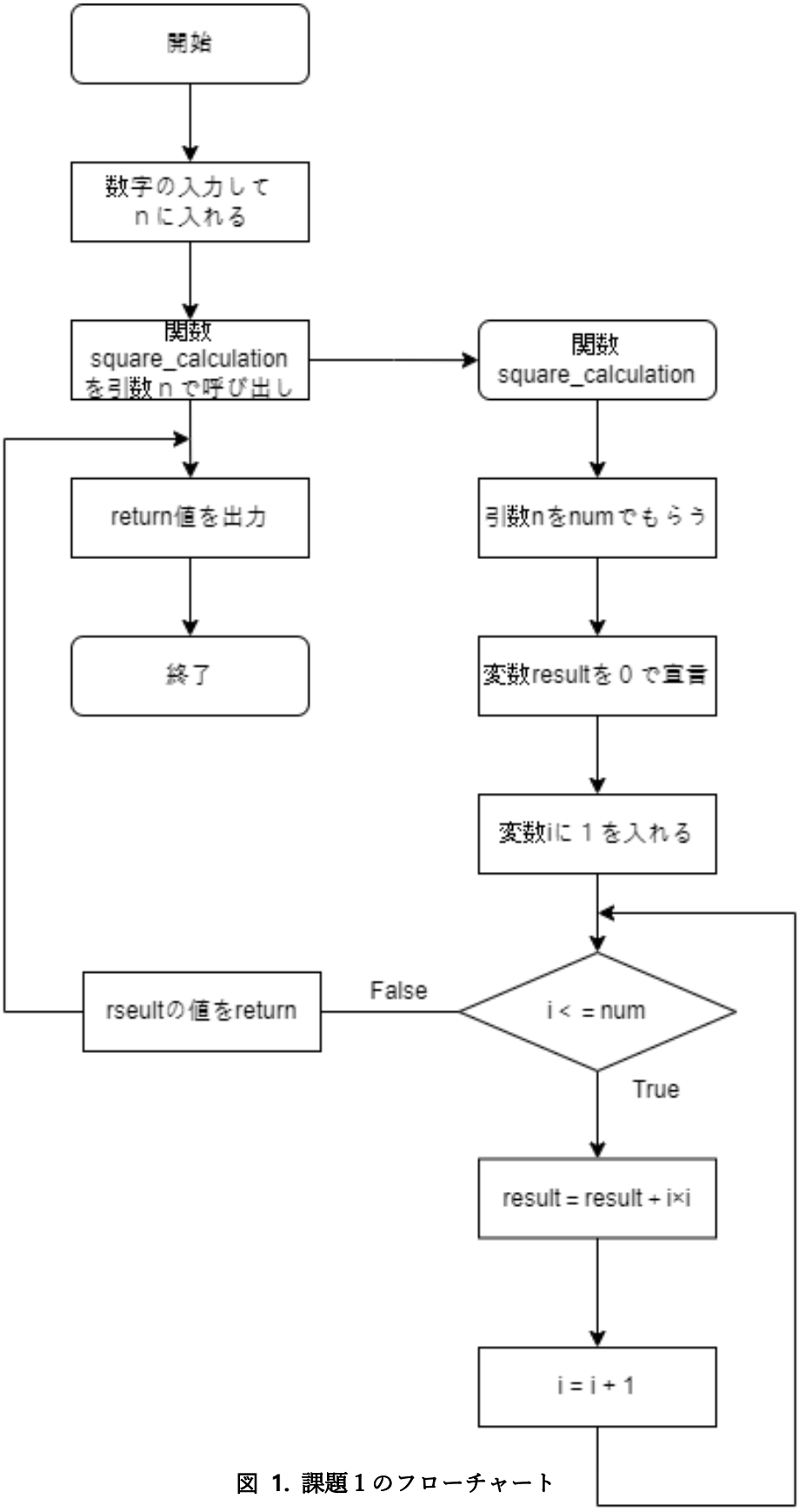

図 1. 課題 1 のフローチャート

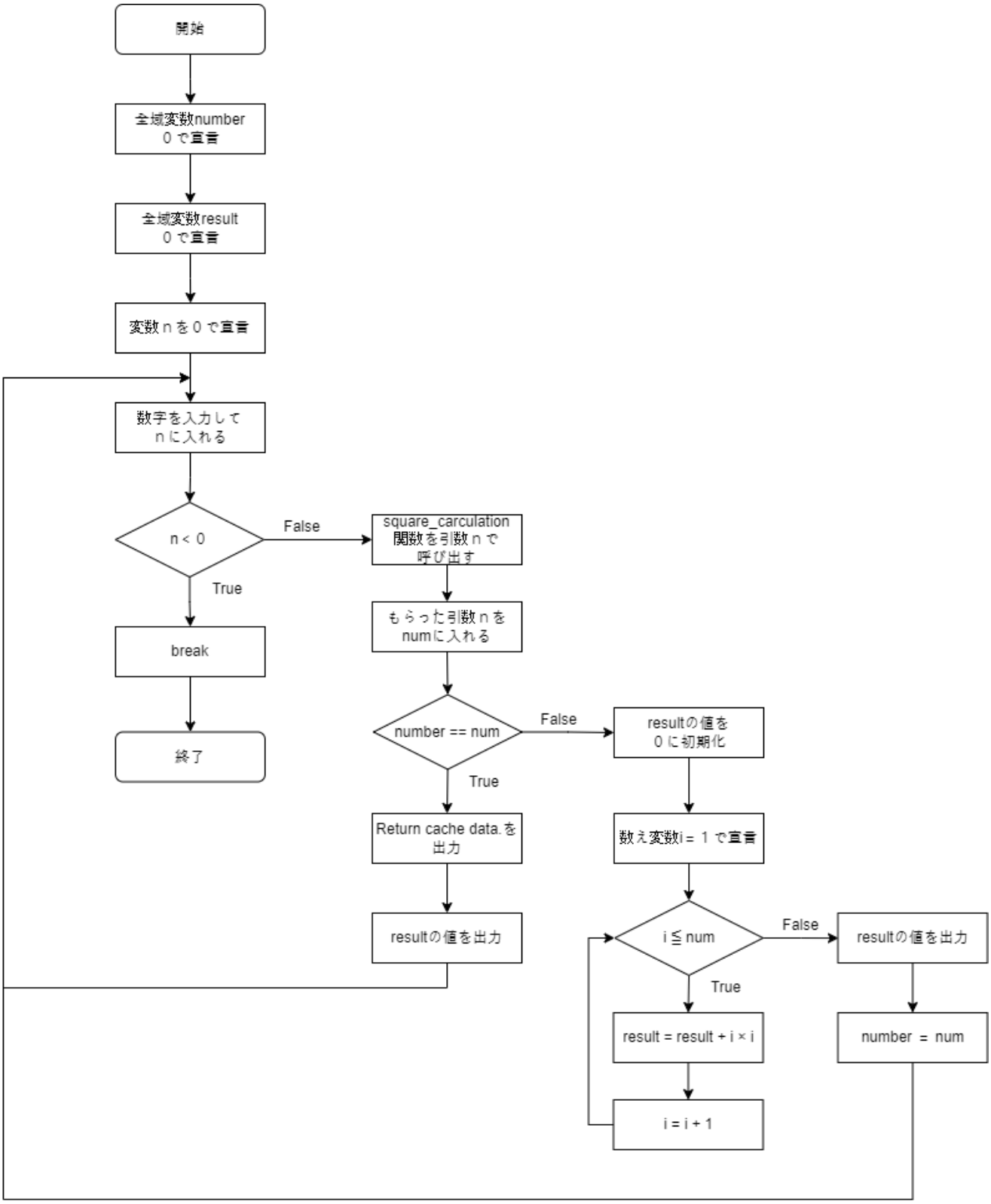

図 2. 課題 2 のフローチャート

(3) アルゴリズムが正しいことの説明

A. 課題１

i. 正当性

1. 目標の計算
   A. $1^2 + 2^2 + 3^2 + \cdots + (n-1)^2 + n^2$
2. プログラムで行われる計算
   A. 引数でもらった数字を反復処理を利用して足し算の計算を行った。

$$1\text{回目}\quad result = 0 + 1 \times 1$$

$$2\text{回目}\quad result = 0 + 1 \times 1 + 2 \times 2$$

$$\cdots$$

$$n\text{回目}\quad result = 0 + 1 \times 1 + 2 \times 2 + \cdots + n \times n$$

$$result = 1^2 + 2^2 + \cdots + n^2$$

結果的に目標の計算と同じであることが検証されたのでこのプログラムは正当だと言える。

ii. 停止性

1. 2乗の和を計算するときに反復処理が使われているが数え変数iを１にとして宣言して、i≦nという条件を立ててiの値を徐々に増加するように設計されている。よって、有限であるnに比べてiは徐々に増加するのでiがnを超える時が必ず存在する。従って、このプログラムは正しく停止するように設計されていると考えられる。

B. 課題２

i. 正当性

1. 目標の計算
   A. $1^2 + 2^2 + 3^2 + \cdots + (n-1)^2 + n^2$
2. プログラムで行われる計算
   A. 前回と同じ場合
      i. 前回行った計算と同じ場合は計算結果が変わらないので予めsaveしておいた計算結果をそのまま出力することで正しい結果が得られると考えられる。
   B. 前回と違う場合
      i. 課題１と同じ計算を行うと目標の計算と一致するので正しい計算結果が得られると考えられる。

ii. 停止性

1. 入力の反復処理

A. Userが負数を入力するまで繰り返す反復処理の停止性について考えてみると、条件n＜0でTureの場合反復処理を停止するように設計されている。よって、Userが反復処理を止めたいときに停止するようになっているので正しく停止することが出来ると考えられる。

2. 計算の反復処理

A. 課題1の同じである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。

(4) ソース・プログラムの説明

A. 課題1

ソースコード

main.c

```c
#include <stdio.h>

int square_calculation(int n);

int main(void){
    int n = 0;
    printf("input number = ");
    scanf("%d", &n);

    printf("%d", square_calculation(n));
}
int square_calculation(int n){
    int result = 0;
    for(int i = 1; i <= n; i++){
        result += i*i;
    }
    return result;
}
```

実行結果

```
input number = 0
0

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

```
input number = 1
1

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

```
input number = 5
55

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

B. 課題２

ソースコード

main.c

```c
#include <stdio.h>

static int number = 0;
static int result = 0;

void square_carculation(int num);

int main(void){
    int n = 0;
    for(;;){
        printf("n = ");
        scanf("%d", &n);
        if(n < 0){
            break;
        }
        else if(n > 0){
            square_carculation(n);
        }
        else{
            printf("Error!");
            return 0;
        }
    }
    return 0;
}

void square_carculation(int num){
    if(number == num){
        printf("Return cache data.\n");
        printf("%d\n", result);
    }
    else{
        result = 0;
        for(int i = 1; i <= num; i++){
            result += i*i;
        }
        printf("%d\n", result);
        number = num;
    }
}
```

実行結果

```
n = 110
449735
n = 10
385
n = 10
Return cache data.
385
n = 5
55
n = 7
140
n = 5
55
n = 5
Return cache data.
55
n = 0
0
n = -10


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

C. 課題1 ソースコードの説明

```
#include <stdio.h>

int square_calculation(int n);    2乗の和の計算を行う関数を予め宣言。

int main(void){
    int n = 0;
    printf("input number = ");
    scanf("%d", &n);

    printf("%d", square_calculation(n));    printfで関数を呼び出してreturn値を出力
}
int square_calculation(int n){
    int result = 0;    和を入れる変数の宣言と初期化
    for(int i = 1; i <= n; i++){
        result += i*i;    2乗の和の計算
    }
    return result;
}
```

D. 課題２ソースコードの説明

```
#include <stdio.h>

static int number = 0;    全域変数としてどこでも使える前回の引数変数numberを宣言
static int result = 0;    全域変数としてどこでも使える前回の計算結果変数resultを宣言

void square_carculation(int num);    2乗の和の結果を出力する関数

int main(void){
    int n = 0;    入力された数字を入れる変数
    for(;;){
        printf("n = ");
        scanf("%d", &n);
        if(n < 0){    入力された数字が負数の場合
            break;    繰り返し文を脱出してプログラムを終了する。
        }
        else if(n >= 0){    入力された数字が０以上の場合
            square_carculation(n);    関数の呼び出し
        }
        else{    例外のエラー処理
            printf("Error!");
            return 0;
        }
    }
    return 0;
}

void square_carculation(int num){
    if(number == num){    前回の引数と一致する場合
        printf("Return cache data.¥n");
        printf("%d¥n", result);    前回の計算結果を出力
    }
    else{    前回の引数と違う場合
        result = 0;    計算結果初期化
        for(int i = 1; i <= num; i++){    2乗の和の計算
            result += i*i;
        }
        printf("%d¥n", result);
        number = num;    前回の引数として全域変数にsaveする。
    }
}
```

(5) 考察

A. 課題 1

i. 2乗の和の計算を別の関数で行う理由として考察したいと思う。今回の場合は2乗の和の計算を行う場所が一つしかいなかったのでmain関数の中で反復処理を利用して2乗の和の計算を行っても問題はない。しかし、2乗の和の計算を何回も行うことプログラムを作るときに毎回for文を利用して計算を行うとコードが複雑になる可能性がある。また、見る人が毎回どのような反復処理なのかをいちいち確認する必要があるのでとても非効率的である。よって、関数を利用してどのような計算を行うと関数であるかを示すことで効率的にコードを把握することができる。また、コードを修正するときにも関数を利用した場合は関数の中を修正すると関数が使われたすべてのところが修正されるが関数を使わなかった場合はすべてのコードを修正する必要があるので非効率的である。よって、よく使われる計算は関数として宣言するのが効率的なので2乗の和の計算を別の関数を利用したと考えられる。

B. 課題 2

i. 前回の計算結果と引数を覚えた理由と全域変数を利用して理由について考察したいと思う。全域変数を利用した理由は今回のプログラムでは2乗の計算を別の関数の中で行っているため関数の中で使われた変数は関数を呼び出す度に初期化される。しかし、全域変数はすべての関数で使える変数なので関数を呼び出す度に初期化されることなく既存の値を保存することができるので全域変数を使っていると考えられる。そして、前回の計算結果を保存した理由として考えられるのは前回と同じ引数の場合無駄な計算を行うことなく計算結果を導くことができるので効率的であると考えられる。特に、今回は計算を行うために反復処理を使っているので毎回反復処理をするとメモリを計算が遅くなる可能性がある。よって、前回の計算を保存して同じ時に前回の計算結果を出力するだけで時間の短縮できるため効率的なコードになると考えられる。

(6) 感想

A. 関数の使い方になれるようになった。また、全域変数を利用して前回の計算結果を保存することで得られるメリットについて考えるようになるいい経験であった。そして、同じ計算を行う時には別の関数を利用することでより見やすいコードが作れることを知ってこれからも頻繁に使われそうな計算は関数を利用してプログラムを作成することが効率的であると思った。