2　構造化プログラミングのすすめ

2.1　プログラムをどうやって作るか

確実で、拡張性のあるプログラムを早く作成する方法として構造化プログラミングがある。以下ではこの構造化プログラミングについて説明する。

プログラミングする際、まず対象とする系の (無次元化された) 方程式をよみ、プログラムコードの作成に入るのだが、まず気をつけて欲しいのは、

はじめから、プログラムの詳細部分の作成に入ってはいけない。[注1]

ということである。このことは、作成しようとするシミュレーションの規模 (ソフトウェアとしての汎用度) にもよるのだが、(1) 非常大きな規模のプログラム、あるいは、(2) 非常に汎用的で計算の流れが非常に複雑な場合、また、(3) 複数人でそのような大規模で複雑な計算の流れを取り扱うプログラムソースコートを作成するような場合であれば、フローチャートを紙面上で作成し、それに基づいてプログラムを作り上げる必要があろう。この場合は、紙面上に作成したフローチャートに基づいてプログラミングを進めて行くことになる。一方、それほど計算のフローが複雑でない場合、例えば、ある特定の系だけのシミュレーションを取り扱うプログラムを作成するような場合、フローチャートの作成を別紙で行い、それを用いるといった大がかりな準備は必要はないであろうと考える。後者の場合、紙面でフローチャートを考えるのではなく、直接プログラムソースコードを書く領域に、フローチャートを考えて書くとすれば、そのコメントの構造そのものが、そのままソースコードの骨組み **”スケルトン”** になるので、非常に便利である。つまり、まずは、

計算の流れを抽象化して、計算の流れを構成する一つ一つを意味の分かる短い文で表し、それをコメントとしてプログラムコードの中に箇条書にして書き込んで行くことを行う。

ということになる。

ここでは、次の方程式

$$\frac{\partial \psi}{\partial t} = \Delta \mu \tag{1}$$

$$\Delta \mu = -\psi + \psi^3 - \Delta \psi \tag{2}$$

を周期境界条件の下で解く場合を例として考えよう。

上で述べた内容に従って、まず、メイン関数の中に、処理の流れをコメントで列挙するようにする。以下に例を挙げる。プログラム名を PhaseSeparation.c とする

[注1] もっとも作成するプログラムの内容が非常に簡単な場合は、このことは当てはまらない。

構造化プログラミング Step 1 : PhaseSeparation.c

```
#include<stdio.h>
int main()
{
  /* Version の表示 */
  /* psi の初期値設定 */
  for( step=0; step<MAXSTEP; step++ ){
    /* 場 psi の境界条件設定 */
    /* 場 mu の計算 */
    /* 場 mu の境界条件設定 */
    /* 新しい時刻の psi の更新 */
    /* ある時間間隔でデータをファイルへの出力 */
  }
}
```

次に、このコメントを中身を書いていない空の関数 (**関数のスケルトン**) をソースコードに書き込む。

この時点で書く関数は一時的なものと考えておいてほしい。

ただし、名前は重要である。コメントから空の関数を作成する時、これらの関数の引数は、とりあえず、void にしておくことにする. また返り値は、関数が正常終了したときに 0 を返し、何らかの異常が生じた場合には 1 を返すことにするために、int 型 とする。これらの関数スケルトンは、あとで関数に応じて引数を必要とするもの、返り値が、int 型以外のものになる場合もあるだろう。また、この関数スケルトン作成作業と同時に同時にプロトタイプ宣言を main 関数の前に書いておくことにする。(後で、ヘッダファイル (.h の付いたファイル) へ移動させる) 関数スケルトンの作成時に気を付けて欲しい事は、

関数の名前は、その関数名からその関数内部で行われている内容がある程度が分かるものにする

ということである。これは後で、ソースコードを読んだときに自然に計算の流れが追えるようにするためである。また、名前は英単語にすべきである。これは、趣味の問題でもあるが、日本語のローマ字表記は非常に読みにくいし、時によっては、外国人と一緒に仕事をするかもしれないので、下手な英語でもよいから、英単語を用いるべきである. あとは、プロトタイプ宣言した全ての関数スケルトン群を main 関数の後に付け加える。

もう一度、第二段階での作業内容をまとめると

1. **第一段階でメイン関数の中に書き込んだ計算の流れを書いたコメントの箇条書を、int 型の返り値で、引数 void の関数を作成し置き換える。このとき関数名はその関数内部で計算する内容が簡単に分かるような名前で、かつ、英単語を使って作成する。**
2. **1 で作成した関数スケルトンのプロトタイプ宣言を main 関数の前に羅列する。**
3. **2 プロトタイプ宣言した全ての関数スケルトン群を main 関数の後に付け加える。**

以下に、例を示す。**構造化プログラミング Step 1** と比較してほしい。

構造化プログラミング Step 2 : PhaseSeparation.c

```
#include<stdio.h>
int ShowVersion( void );
int SetInitValueOfVolumeFraction( void );
int SetPeriodicBCForVolumeFraction( void );
int EvaluationOfChemicalPotential( void );
int SetPeriodicBCForChemicalPotential( void );
int TimeEvolutionOfVolumeFraction( void );
int WriteData( void );
int main()
{
  ShowVersion();                          /* Version の表示 */
  SetInitValueOfVolumeFraction();         /* psi の初期値設定 */
  for(step=0; step<MAXSTEP; step++ ){
    SetPeriodicBCForVolumeFraction();     /* 場 psi の境界条件設定   */
    EvaluationOfChemicalPotential();      /* 場 mu の計算            */
    SetPeriodicBCForChemicalPotential();  /* 場 mu の境界条件設定    */
    TimeEvolutionOfVolumeFraction();      /* 新しい時刻の psi の更新 */
    if(step\%IntervalStepOfDATA == 0 ){ WriteData(); }
                                          /* ある時間間隔で Data を file へ出力 */
  }
  return(0);
}
int ShowVersion( void ){
   return(0);
}
int SetInitValueOfVolumeFraction( void ){
   return(0);
}
int SetPeriodicBCForVolumeFraction( void ){
   return(0);
}
int EvaluationOfChemicalPotential( void ){
   return(0);
}
int SetPeriodicBCForChemicalPotential( void ){
   return(0);
}
int TimeEvolutionOfVolumeFraction( void ){
   return(0);
}
int WriteData( void ){
   return(0);
}
```

次に、とりあえずマクロ変数 L,M を定義し、グローバル変数として ψ と μ に対する変数領域の確保を行う。

この段階で、とりあえず ψ と μ に対する変数をグローバルとするのは、関数の引数に何を渡すかを考える必要を無くし関数を書きやすくするためである。

後で、これらの変数を int main 関数の中に移し、領域確保は、malloc,calloc(C) か new(C++) を用いて行うように変更するが、ここではグローバル変数としておく。

構造化プログラミング Step 3 : PhaseSeparation.c

```
#include<stdio.h>
#define L 100
#define M 100
#define MAXSTEP              10000
#define IntervalStepOfDATA   100
int ShowVersion( void );
int SetInitValueOfVolumeFraction( void );
int SetPeriodicBCForVolumeFraction( void );
int EvaluationOfChemicalPotential( void );
int SetPeriodicBCForChemicalPotential( void );
int TimeEvolutionOfVolumeFraction( void );
int WriteData( void );
double psi[2][L][M];
double mu[2][L][M];
int    step;
int main()
{
  ....
  ....
}
int ShowVersion( void ){
  ....
  ....
```

次にヘッダファイル PhaseSeparation.h を作成し次に、main 関数の前にある include 文、マクロ変数定義文、関数プロトタイプ宣言部を PhaseSeparation.h へ移動させる。

構造化プログラミング Step 4 : PhaseSeparation.h

```
#ifndef __PHASE_SEPARATION_H__
#define __PHASE_SEPARATION_H__
#include<stdio.h>
#define L 100
#define M 100
#define MAXSTEP              10000
#define IntervalStepOfDATA   100
int ShowVersion( void );
int SetInitValueOfVolumeFraction( void );
int SetPeriodicBCForVolumeFraction( void );
int EvaluationOfChemicalPotential( void );
int SetPeriodicBCForChemicalPotential( void );
int TimeEvolutionOfVolumeFraction( void );
int WriteData( void );
#endif // __PHASE_SEPARATION_H__
```

この移動と同時に PhaseSeparation.c を次に用に変更する。

構造化プログラミング Step 5 : PhaseSeparation.c

```
#include "PhaseSeparation.h"
double psi[2][L][M], mu[2][L][M];
int    step;
int main()
{
  ShowVersion();                          /* Version の表示 */
  SetInitValueOfVolumeFraction();         /* psi の初期値設定 */
  for(step=0; step<MAXSTEP; step++ ){
    SetPeriodicBCForVolumeFraction();     /* 場 psi の境界条件設定   */
    EvaluationOfChemicalPotential();      /* 場 mu の計算            */
    SetPeriodicBCForChemicalPotential();  /* 場 mu の境界条件設定    */
    TimeEvolutionOfVolumeFraction();      /* 新しい時刻の psi の更新 */
    if(step%IntervalStepOfDATA == 0 ){ WriteData(); }
                                          /* ある時間間隔で Data を file へ出力 */
  }
  return(0);
}
int ShowVersion( void ){ return(0); }
int SetInitValueOfVolumeFraction( void ){ return(0); }
int SetPeriodicBCForVolumeFraction( void ){ return(0); }
int EvaluationOfChemicalPotential( void ){ return(0); }
int SetPeriodicBCForChemicalPotential( void ){ return(0); }
int TimeEvolutionOfVolumeFraction( void ){ return(0); }
int WriteData( void ){ return(0); }
```

次に、Makefile を作成する。Makefile の作成に関しては、適当なテキストブックを参考にして欲しい以下にサンプル用の Makefile を書く。

構造化プログラミング Step 6 : Makefile

```
CC=gcc
OPT=-O3
LFLAGS=-lm
all: PhaseSeparation.c PhaseSeparation.h
        $(CC) PhaseSeparation.c $(OPT) $(LFLAGS)

clean:
        rm -f core *~ *.o a.out
```

ここで、気を付けて欲しいのは all: の次の行の先頭の空白は単なる空白ではなく、タブであるということで

る。これを間違うと Makefile は正常に動作しない。

構造化プログラミング Step 7 : コンパイルのテスト

```
prompt> ls
Makefile  PhaseSeparation.c  PhaseSeparation.h
prompt> make
prompt> ls
Makefile  PhaseSeparation.c  PhaseSeparation.h  a.out
prompt> a.out
prompt>
```

を実行すると実行ファイル (a.out) が作成されるはずである。コンパイルに失敗したら、まずはそれを訂正し、a.out が作成されるようになるまで、訂正を繰り返す。これが出来たら、今作成しようとしているプログラムの完全なスケルトンが出来上がったことになる。後はこれに関数を書き込んで行けばよい。そして最後に、グローバル変数を関数の引数渡しに変更し領域確保を malloc もしくは calloc で行うように変更すれば、完成！.

後はあなた次第！ Good Luck！ さあプログラムを書こう！

3　数値計算を用いた研究における注意点

1. 対象としている物理系に対する実験についてよく調べたか。
   a. システムサイズ、
   b. 使われている物理定数の値
2. 方程式の意味
   各項の物理的を理解する
3. 方程式の無次元化
   a. 空間の単位、時間の単位がなにか。
   b. その単位の物理的意味を理解しているか。
4. 境界条件
   a. 数学的表式を理解しているか。なぜそのような数式になっているか説明できるか
   b. 数値計算上でその境界条件を実現する方法
5. 実際に計算に用いた定数
   システムサイズ
   無次元化した DT
   無次元化した パラメータの値 $\Rightarrow$ 実際の値との関連
6. 実験結果との定性的、定量的比較
   無次元化した空間のみならず、実際の値ではどれくらいかも比較できると良い。
7. 何を目的として、計算を行い、最終的にどのような結果が得られたか。