第3学年　電気電子工学実験実習報告書

6　数値計算（1）

実験日 令和 4 年 10 月 27 日 (木)

班	学生番号	氏名
2	3322	高橋広旭

共同実験者名

提出日			備考	評価
予定日　11/17				
提出日				

東京都立産業技術高等専門学校
電気電子工学コース

1　目的

本実験では、

- 回路図より連立方程式を立てられる。
- C 言語で（繰り返し、条件分岐、配列を用いた）基本プログラムが書ける。

ことを目的とする。

2　原理

2.1　C 言語について

自然言語（日本語や英語）は多義で曖昧さを持っているため、機械には理解するのが難しい。このため、計算して欲しい内容をコンピュータに伝えるには、専用の言語を使わなくてはならない。そのような言語をプログラミング言語といい、C 言語の他に Java、C++、Visual Basic、Perl、Python、Fortran、Pascal、Cobol、BASIC、Ruby など、様々な言語が様々な用途のために開発されている。C 言語は、B 言語をもとに AT&T のケン・トンプソンとデニス・リッチーによって、UNIX という OS を書くための言語として開発が開始された。1973 年ごろに最初の完成を見ており、その後何度も仕様が拡張変更されている。

2.2　関数

C 言語は手続き型言語と呼ばれ、ひとまとまりの手続きを関数という形でまとめて実装する。関数は複数の “引数” を受け取り、一つの値を返すものである（ただし、値を返さない void 関数もある）。関数の記述法をプログラム 1 に示す。

プログラム 1: 関数の形

```
1 返り値の型 関数名 (引数 1 の型 引数 1の名, 引数 2 の型 引数 2の名, ...){
2         型定義; // 必要な場合
3         文 1;
4         文 2;
5         文 3;
6         return 式; // 必要な場合
7 }
```

この関数を使う場合には、プログラム 2 のように記述する。関数に渡すのは単に式だけを記述する。

プログラム 2: 関数の利用法

```
1 代入する変数 = 関数名 (関数に渡す式 1, 関数に渡す式 2, ...)
```

関数は呼び出し側と同じファイルに書かれていない場合、呼び出し側で関数名を書いても、コンパイラは関数について知ることができずにエラーとなる。このため、通常の関数を用いたプログラムでは以下のような構成を取ることが多い。

(i) メインプログラムのファイル（◎◎.c）

- main 関数を持つ
- 作成した関数を呼び出す
- 関数のプロトタイプ宣言が書かれた (iii) のヘッダファイルを読み込む

(ii) 関数の書かれたファイル（○○.c ）

- main から呼ばれる関数が書かれる
- 機能ごとに小分けにして、関数内から別の関数を呼び出すこともある
- メインプログラムでも必要となる共通の設定ファイルは (iii) のヘッダファイルに分けて記述する

(iii) ヘッダファイル（○○.h）

- ○○.c に関連するプロトタイプ宣言・定数・構造体などを記述する
- 通常、関数本体などは記述しない

ここで出てきたプロトタイプ宣言とは、関数の諸元を説明するものである。プログラム 3 に示すように、プログラム 1 の 1 行目を持ってきて、最後に「;」を付けるだけで良い。

プログラム 3: プロトタイプ宣言

```
1 返り値の型 関数名 (引数 1 の型 引数 1の名, 引数 2 の型 引数 2の名, ...);
```

2.3 テストファーストプログラミング

2.2 章 (i) のメインプログラムを書く時点では、(ii) で書かれた関数は完全に動作するものであることが望ましい。そこで、(ii) の関数群が完全に動作することを保証するために、通常は以下のテストファーストプログラムを (ii) より先に作成する。

(iv) テストプログラム（text ○○.c）

- 複数の引数の組み合わせを使って関数を呼び出す
- 関数の返値や関数によって変化する値を確認する
- 特殊な処理をする値がある場合には、その前後の境界値でテストをすることが望ましい

ここでは、二つの浮動小数点型引数 (x, y) を受け取り、原点からの距離 R を返す関数 `getR` のテストを例に説明する。テスト関数はプログラム 4 のように記述できる。このように基本的に二つ以上はテストを記載すること。

プログラム 4: testGetR.c の一部分

```
1 void testGetR() {
2         testStart("testGetR");
3         assertEqualsDboule(getR(2.0, 1.0), sqrt(2.0*2.0 + 1.0*1.0));
4         assertEqualsDboule(getR(3.0, 4.0), sqrt(3.0*3.0 + 4.0*4.0));
5 }
```

この時点では、`getR` という関数が分からないため、コンパイルエラー（関数が分からないという警告）になる。この問題を解決するため、(iii) のプロトタイプ宣言が書かれたファイル（getR.h）を書く。ここには関数の諸元を示すプロトタイプ宣言を記載する。二つの引数を受け取り、一つの値を返すのでプログラム 3 を参考に、プログラム 5 のように書けば良い。この時、testGetR.c に include の追加も忘れない。

プログラム 5: getR.h

```
1 double getR(double x, double y);
```

コンパイルをすると警告は出なくなるが、リンカで「_getR」が無いというエラーになる。すぐに実装が思いつかない場合には、プログラム 4 のどちらかのテストが通るようにつじつまを合わせて中身を仮で書けば良い。

プログラム 6: getR.c

```
1 double getR(double x, double y) {
2         return 2.0*2.0 + 1.0*1.0;
3 }
```

このようにすることで、二つの assert 文のうち片方だけは通る実装を書くことができる。2.0 と 1.0 という具体的な場合の実装が書けているので、抽象的な x、y の場合に差し替える作業を行えば良く、ゼロの状態から実装を書くことよりも難易度を下げることができる。

3 方法

3.1 テスト環境の準備

1) testCommon.c に要素の値を並べた配列と比較する関数を追加した。

2) testCommon.h に要素の値を並べた配列と比較する関数のマクロを追加した。

3.2 テストファーストプログラミングの練習

1) teatMatrix.c に 1 次元配列のデータを 2 次元配列にマッピングする setValues という関数のテストを作成した。

2) make test を実行し、警告が出力されることを確認した。

3) matrix.h に setValues のプロトタイプ宣言を追加した。

4) make test を実行し、エラーが出力されることを確認した。

5) matrix.c に setValues の雛形を追加した。

6) make test を実行し、テストに失敗することを確認する。

7) matrix.c に setValues の実装を記述した。

8) make test を実行し、テストを通過したことを確認する。

3.3 行列の足し算

1) 行列の足し算を行う関数 addMatrix を 3.2 と同様の手順で実装した。

3.4 行列の掛け算

1) 2 つの行列の掛け算を行う関数 mulMatrix を 3.2 と同様の手順で実装した。

3.5 行列の変形

1) r 行 c 列の行列 matpp の n 行目を k 倍する関数 rouCmulMatrix を 3.2 と同様の手順で実装した。

2) r 行 c 列の行列 matpp の n1 行目と n2 行目を入れ替える関数 rowConvertMatrix を 3.2 と同様の手順で実装した。

3) r 行 c 列の行列 matpp の n1 列目と n2 列目を入れ替える関数 colConvertMatrix を 3.2 と同様の手順で実装した。

4) r 行 c 列の行列 matpp の n1 行目を k 倍したものを n2 行目に加算する関数 rowCmulAddMatrix を 3.2 と同様の手順で実装した

4 結果

4.1 実装したプログラム

3.3 の行列の足し算を行うプログラムをプログラム 7、3.4 の行列の掛け算を行うプログラムをプログラム 8、3.5 の行列の変形を行うプログラムをプログラム 9〜12 に示す。

プログラム 7: addMatrix

```
1  void addMatrix(double anspp[N][N], double in1pp[N][N], double in2pp[N][N],int r, int c) {
2          int i, j;
3          for (i = 0; i < r; i++) {
4                  for (j = 0; j < c; j++) {
5                          anspp[i][j] = in1pp[i][j] + in2pp[i][j];
6                  }
7          }
8  }
```

プログラム 8: mulMatrix

```
1  void mulMatrix(double anspp[N][N], double in1pp[N][N], int r1, int c1,double in2pp[N][N], int r2,
       int c2) {
2          int i, j, k;
3          for (i = 0; i < r1; i++) {
4                  for (j = 0; j < c2; j++) {
5                          anspp[i][j] = 0.0;
6                                  for (k = 0; k < c1; k++) {
7                                  anspp[i][j] += in1pp[i][k] * in2pp[k][j];
8                          }
9                  }
10         }
11 }
```

プログラム 9: rowCmulMatrix

```
1  void rowCmulMatrix(double matpp[N][N], int r, int c, int n, double k) {
2      int i;
3      for (i = 0; i < c; i++) {
4          matpp[n][i] = matpp[n][i] * k;
5    }
6  }
```

プログラム 10: rowConvertMatrix

```
1  void rowConvertMatrix(double matpp[N][N], int r, int c, int n1, int n2) {
2    double storage;
3    for (int i = 0; i < c; i++) {
```

```
4         storage = matpp[n1][i];
5         matpp[n1][i] = matpp[n2][i];
6         matpp[n2][i] = storage;
7     }
8 }
```

プログラム 11: colConvertMatrix

```
1 void colConvertMatrix(double matpp[N][N], int r, int c, int n1, int n2) {
2   double storage;
3   for (int i = 0; i < r; i++) {
4         storage = matpp[i][n1];
5         matpp[i][n1] = matpp[i][n2];
6         matpp[i][n2] = storage;
7   }
8 }
```

プログラム 12: rowCmulAddMatrix

```
1 void rowCmulAddMatrix(double matpp[N][N], int r, int c, int n1, double k,int n2) {
2         for (int i = 0; i < c; i++) {
3                 matpp[n2][i] = matpp[n2][i] + matpp[n1][i] * k;
4         }
5 }
```

4.2 プログラムの実行結果

プログラムを実行した結果を以下に示す。

プログラム 13: 実行結果

```
1         == Test testSetValues ==
2         == Test addMatrix ==
3         == Test mulMatrix ==
4         == Test rowCmulMatrix ==
5         == Test rowConvertMatrix ==
6         == Test colConvertMatrix ==
7         == Test rowCmulAddMatrix ==
8         All tests are Ok. [# of Tests = 80, # of pass = 80 (100%)]
```

プログラム 13 より、作成したプログラムは正しく動作しているといえる。

5 考察

5.1 何故テストファーストプログラミングをする必要があるのか

テストファーストプログラミングを用いることによって関数の実装ミスに早い段階で気づくことができる。複雑な関数を作成する場合エラーが発生しやすくなる。また、関数内に別の関数を用いるとエラー発生時に作成中の関数と関数内で使用している関数の確認をする必要があるため原因究明に時間を要する場合がある。このような事態を防ぐために、1 つの関数を作成する間に何度もテストを行うテストファーストプログラミングを用いるべきだと言える。

5.2 for 文の利点

for 文は同じ処理を何度も繰り返す際によく用いられている。例えば階乗の計算を行う場合、4!であれば 4 つの値を掛けるだけなので for 文を用いない場合でも現実的な記述量である。しかし、それが 100!

になった場合、全ての値を入力すると膨大な量を記述する必要がある。一方、for 文を用いると記述量を大幅に減らすことが可能である。また、繰り返す回数等の条件変更を行う際には一部の定数や変数を変更するだけで対応できる。これらのことから for 文を用いることによって記述量を減らしつつ条件変更に柔軟に対応できることが for 文利点としてあげられる。

5.3　この実習で得られた知見

今回の実験を通してテストファーストプログラミングに関する知見を深めることが出来た。実際に出力されるエラーを解読し、修正する過程の中でテストファーストプログラミングを用いると原因究明が容易であることを確認できた。しかし、テストに用いる計算が間違っていると関数が正常に動作している場合でもテストを通過できないことを確認できた。従って、テストに用いる計算は慎重に行うべきだと痛感した。

5.4　実習から浮かんだ技術的な疑問点

#ifdef とはどのような働きをするものか。

5.5　上の疑問点を調査し解決せよ

#ifdef は#ifdef[識別子名] と記述し、識別子が定義されているかどうかの判定するものである [1]。今回の実験では testCommon.c で用いられている。識別子名の COMPLEX はコメントアウトされているためこの下に記述されている#include ”complex.h”は処理されない。

6　結論

今回の実験を通して C 言語で（繰り返し、条件分岐、配列を用いた）基本プログラムが書くことができた。

参考文献

[1] 長谷川裕行,“主なプリプロセッサ指令”, グレープシティ株式会社,https://dev.grapecity.co.jp/support/powernews/column/clang/014/page02.htm,2022 年 11 月 16 日

前試問

プログラム 14: ソース 1

```
1  #include <stdio.h>
2
3  int main() {
4          int i;
5
6          printf("整数を入力してください.␣\n");
7          scanf("%d", &i);
8
9          return 0;
10 }
```

プログラム 15: ソース 2

```
1  #include <stdio.h>
2
3  int main() {
4          int i = 1;
5          float a = 2.0;
6
7          printf("␣i␣=␣%d\n", i);
8          printf("␣a␣=␣%f\n", a);
9
10         return 0;
11 }
```

プログラム 16: ソース 3

```
1  #include <stdio.h>
2
3  int main() {
4          int i, s = 0;
5
6          for (i = 1; i <= 10; i++) {
7                  s = s + i;
8          }
9
10         printf("␣1␣から␣10␣の和は␣%d␣です.␣\n", s);
11
12         return 0;
13 }
```

プログラム 17: ソース 4

```
1  #include <stdio.h>
2
3  int main() {
4          int i;
5
6          printf("整数を入力してください.␣\n");
7          scanf("%d", &i);
8
9          if (i == 0) {
10                 printf("入力された整数は␣0␣です.␣\n");
11         } else if (i > 0) {
12                 printf("入力された整数は正です.␣\n");
13         } else {
14                 printf("入力された整数は負です.␣\n");
15         }
16
17         return 0;
18 }
```

プログラム 18: ソース 5

```
1  #include <stdio.h>
2
3  int main() {
4          int i, a[10];
5
6          for (i = 0; i < 10; i++) {
7                  a[i] = i;
8                  printf("␣a␣は␣%d␣です.␣\n", a[i]);
9          }
10
11         return 0;
12 }
```