C 言語の学習
定数・変数とデータ型・配列と文字列

山本昌志*

2004 年 4 月 28 日

1　本日の学習内容

C 言語の基本的な部分を学習する。内容は、教科書の 2〜4 章である。各章の内容と理解すべきことは、以下の通りである。

2 章　定数

- 整数と実数の定数の書き方が分かる。
- エスケープシーケンスについては、水平タブと改行の使い方が分かること。

3 章　変数とデータ型

- 変数宣言の意味が分かる。
- 文字型 `char` と整数型 `int`、倍精度実数型 `double` 型が使える。

4 章　配列と文字列

- 配列がどのようなものか理解できる。
- 整数型や実数型の配列が使える。

2　定数 (教科書の 2 章)

教科書の p.20 を見ると分かるようにいろいろな定数がある。しかし、この講義は主に数値計算について学習するので、使う定数は決まっている。もっとも多く使われるのが、整数を表す整数定数と実数を表す浮動小数点定数である。文字定数や文字列リテラルは使用頻度が少ない。その他のものはほとんど使われない。

*独立行政法人　秋田工業高等専門学校　電気工学科

2.1 整数型と実数型

整数型と実数型の定数を変数に代入して、画面へ出力するソースをプログラム 1 に示す。各行の内容は以下の通りである。

1, 2, 11, 12 行 とりあえずおまじないと思って欲しい。

3 行 整数型の変数 `seisu` を宣言。詳細は 3 節で述べる。

4 行 倍精度実数型の変数 `jisu` を宣言。詳細は 3 節で述べる。

6,7 行 変数に定数を代入。コンピューター言語で値を代入する場合、左辺の変数に右辺の計算結果を代入することになる。必ず、左辺は変数で、右辺は数値となる。

9 行 ダブルクォーテーション[1]で囲まれた部分中の`%d` の部分に変数 `seisu` の値を 10 進数 (decimal) で、`%e` の部分に変数 `seisu` を e タイプで置き換えディスプレイに表示する。この`%d` や`%e` を変換仕様という (教科書 p.322)。`\n` は改行である。

プログラム 1: 定数の学習プログラム

```
1  #include <stdio.h>
2  int main(){
3    int seisu;
4    double jisu;
5
6    seisu = 65;
7    jisu = -69.53e-7;
8
9    printf("seisu = %d     jisu = %e\n", seisu, jisu);
10
11   return 0;
12 }
```

プログラム 1 を直して、以下の練習問題を実行させよ。

[練習 1] 変数 `seisu` に -1234 を `jisu` に -6.987×10^{-68} を代入するプログラムを作成せよ。

[練習 2] 変数 `seisu` に -6.987×10^{-68} を `jisu` に -1234 を代入するプログラムを作成せよ。そして、実行結果の内容を考察せよ。

[練習 3] 変数 `seisu` に $-10/3$ を `jisu` に $-10/3$ を代入するプログラムを作成せよ。そして、実行結果の内容を考察せよ。

2.2 エスケープシーケンス

教科書の表 2-4(p.28) のものをエスケープシーケンスと言う。これは 2 つあるいはそれ以上の文字列で表す特殊文字である。それらの機能は表に書いてあるとおりであるが、数値計算で重要なものは、`\n` と `\t` である。とりあえず、この 2 つの動作を理解せよ。

プログラム 1 の 9 行目の `printf` 関数のダブルクォーテーション内を直す以下の練習問題を実行させよ。

[1]記号 "をダブルクォーテーションと言う。

[**練習 1**] \n と適当に挿入して、その動作を確認せよ。挿入は、1 個のみならず、2〜3 個それを挿入した場合も確認せよ。

[**練習 2**] \t と適当に挿入して、その動作を確認せよ。挿入は、1 個のみならず、2〜3 個それを挿入した場合も確認せよ。

3　変数とデータ型 (教科書の 3 章)

C 言語の変数は、数学で使われる変数とよく似ており、そこに数値を代入することができる。実際には、コンピューターのメモリーの一部の記憶場所を示しており、そこに数値を記憶するようになっている。イメージは、図 1 に示しているとおりで、変数とは数値を入れる箱のようなものである。整数型と倍精度実数型の変数は、数学の変数と全く同じである。

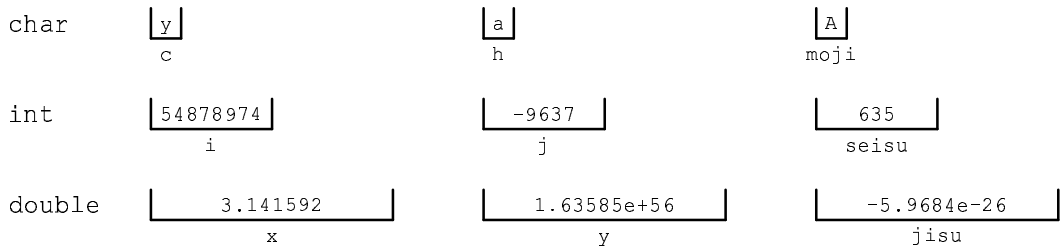

図 1: 変数のイメージ。変数とはデータを入れる箱のようなもの。

図の左端に書かれている char と int、double はその変数の型を示しており、それぞれ文字型、整数型、倍精度実数型を表す。これは変数に代入できる値の種類が決まっていると考える。文字型の変数 (c, h, moji) には 1 文字が、整数型の変数 (i, j, seisu) には整数が、倍精度実数型の変数 (x, y, jisu) には実数を入れることができる。このように変数には、型と変数名があることを理解して欲しい。プログラム中では、この変数に入れられた値を操作するのである。

型と変数名の指定は、プログラム中の中括弧 で囲まれた部分の最初に

```
char c, h, moji;
int i, j, seisu;
double x, y, jisu;
```

のように書く。これを変数の宣言と言う。

諸君は方程式を使って問題を解く場合、変数というものを使っている。数学の変数と C 言語の使用方法での決定的な違いは、変数の型の宣言が必要なことである。プログラムの動作にはこの変数宣言は無駄のように思える。これが必要なのは、コンピューターの都合である。たいていのプログラミング言語では、これを宣言することにより、メモリーを確保する。必要なメモリー量はプログラマーが決めなくてはならない。コンピューターは、このプログラムがどの程度のメモリーが必要か全く分からないからである。

図 1 を見て分かるように、箱の大きさが型によって異なる。これは、一つのデータを表現するために必要な情報量が異なるためである。情報量の単位は、ビット (bit) が使われる。2 進数の 1 桁を 1 ビットと言

う。8 ビットで 1 バイトとなり、それがコンピューターで使われる基本単位となる。

同じ int 型でもいろいろあり、表現できる範囲が異なっている[2]。これは一つの変数の情報量の差から生まれる。C 言語で使われる型によって表現できる範囲は、教科書の表 3-1(p.34) に示されている。全ての C 言語は同一になっておらず、諸君が使っているシステムではこの表のようになっている。いろいろな型があるが、ほとんどの場合、char、int、double で十分である。諸君が作るプログラムでは、これらで十分、間に合うが、問題が生じたときのみ他の型を使えば良い。

文字型の変数に代入できるのは、諸君の教科書の表紙の裏の黄色のページの文字コード表に書かれているもののみである。漢字や平仮名を代入したい場合は、文字型の配列を使うことになる。興味のある者は自分で調べよ。

文字型と整数型、実数型の変数を宣言、それに値をに代入、そして画面へ出力するソースをプログラム 2 に示す。各行の内容は以下の通りである。

3 行 文字型の変数を宣言。

4 行 整数型の変数を宣言。

5 行 倍精度実数型の変数を宣言。

6-7 行 シングルクォーテーション[3]で囲まれた文字が代入される。

16 行 文字変数を出力するための変換仕様は、%c を使う (教科書 p.322)。

プログラム 2: 変数の学習プログラム

```
1  #include <stdio.h>
2  int main(){
3    char c, h;
4    int i, j;
5    double x, y;
6
7    c = 'a';
8    h = 'A';
9
10   i = 123;
11   j = -987654321;
12
13   x = -1.23456;
14   y = 9.87654321e-12;
15
16   printf("c = %c \t h = %c\n", c, h);
17   printf("i = %d \t j = %d\n", i, j);
18   printf("x = %e \t y = %e\n", x, y);
19
20   return 0;
21 }
```

[2]一つの文字のみ代入可能なものは文字型の変数である。コンピューター内部では文字は整数として扱うので、文字型変数に整数が代入できるのである。

[3]記号 ' をシングルクォーテーションと言う。

[**練習 1**] プログラム 2 を、以下のように変数を使うように変更せよ。もちろん、変数の代入された結果も表示させること。

- 文字型の変数、hoge と hogehoge を宣言し、それぞれに A と k を代入する。
- 整数型の変数、fuga と fugafuga を宣言し、それぞれに-123 と 321 を代入する。
- 倍精度実数型の変数、foo と bar を宣言し、それぞれに -0.1987 と -96.85×10^{-28} を代入する。

4 配列と文字列 (教科書の 4 章)

4.1 配列とは

3 節で示した変数[4]の場合、一度に確保できるメモリーの領域は 1 個なので、大量のデータを扱うのは不向きである。変数だけを使って、100 万個のデータを扱うことは不可能である。100 万個の変数名を用意するのはナンセンス。そこで、順序づけられた同じ型のデータが複数ある場合、配列というデータ構造が考えられた。

これは、同じ型のデータを任意の個数宣言し、配列名と自然数[5]でアクセスすることができるようにしたものである。配列を使うためには、

```
int i[10], j[100][100];
```

のように宣言をする。こうすると、

- 配列名 i の整数型のデータ領域が 10 個用意される。用意されるデータ領域は、i[0]〜i[9] である。
- 配列名 j の整数型のデータ領域が 10000 個用意される。用意されるデータ領域は、j[0][0]〜j[99][99] である。。

となる。図 2 のように、メモリー領域が確保される。このデータ構造では、配列名と添え字 (インデックス)、たとえば i[3] や j[25][49] を指定することで、その領域から値を入出力できる。

```
i[3]=5;          /* 配列 i[3] に 5 を代入  */
c=j[25][49];     /* 配列 j[25][49] の値を変数 c へ代入 */
```

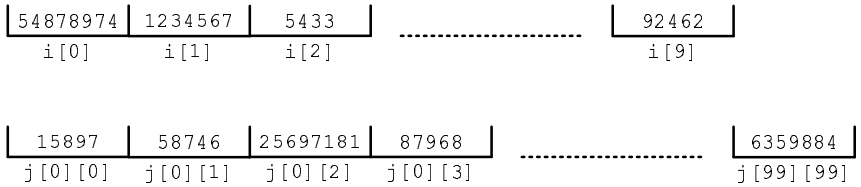

図 2: 配列のイメージ。データを入れる箱がいっぱいある。ただし箱の大きさは全て同じ。

[4] これを単純型のデータ構造と言う
[5] ここでは、0 も自然数に含める。

添え字が 1 つのものを一次元配列と言い、それ以上のものを多次元配列と言う。C 言語では多次元配列を使う場合、

```
int hoge_1[100], hoge_2[100][100], hoge_3[100][100][100];
double huga[10], huge[10][10], hugo[10][10][10];
```

のように宣言を行う。これらも、配列名と複数の添え字で、そこにあるデータにアクセスする事ができる。3 次元以上ももちろん可能である。

4.2 数列、ベクトル、行列を配列で表現

一次元の配列は数学の数列とベクトルと、二次元の配列は行列とよく似ている。実際、数値計算で数列やベクトル、行列に関わる数値演算を行うときには、配列が使われる。これらの数学の表現も、やはり順序づけられた数の集まりにすぎないので、配列と同じである。一方、スカラー量の場合には、通常の変数として扱えばよい。

数列やベクトル、行列の成分を表す場合、下添え字がつく。その添え字と同じように、配列の添え字を使う。実に簡単である。ただし、数学の場合、添え字が 1 から始まることが多いが、C 言語の場合、それは 0 から始まるので注意が必要である。配列の宣言の時、添え字部分に書かれるのは要素数であるので、ベクトルや行列の要素数に 1 を加えた数で領域を確保しなくてはならない。必要数より大きめに確保するのが普通である。

表 1: 数列やベクトルを配列で表現

数学	C 言語
a_1	`a[1]`
a_2	`a[2]`
a_3	`a[3]`
$\vdots$	$\vdots$
a_i	`a[i]`
a_{i+1}	`a[i+1]`
$\vdots$	$\vdots$
a_{2i+1}	`a[2*i+1]`
$\vdots$	$\vdots$

表 2: 行列を配列で表現

数学	C 言語
$a_{1\,1}$	`a[1][1]`
$a_{1\,2}$	`a[1][2]`
$a_{1\,3}$	`a[1][3]`
$\vdots$	$\vdots$
$a_{3\,3}$	`a[3][3]`
$a_{3\,4}$	`a[3][4]`
$\vdots$	$\vdots$
$a_{i\,j}$	`a[i][j]`
$\vdots$	$\vdots$
$a_{i+1\,j+1}$	`a[i+1][j+1]`
$\vdots$	$\vdots$
$a_{2i+1\,3j+2}$	`a[2*i+1][3*j+2]`
$\vdots$	$\vdots$

4.2.1 乗算

配列を使って行列とベクトルのかけ算を行うソースをプログラム 3 にしめす。この例は要素数が少ない場合であるが、多くなると、後で学習する繰り返し処理が必要となる。言うまでもないと思うが、行列とベクトルのかけ算

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{bmatrix} \tag{1}$$

の演算は、

$$c_i = \sum_{\ell=1}^{n} a_{i\ell} b_\ell \tag{2}$$

である。

$n = 2$ の場合の計算を行うプログラム 3 の各行の内容は以下の通りである。

3 行 実数型の配列を宣言。

5-11 行 配列 (行列とベクトル) の要素に値を代入。

13-14 行 行列とベクトルの乗算。

プログラム 3: 行列とベクトルのかけ算

```
1  #include <stdio.h>
2  int main(){
3    double a[3][3],b[3],c[3];
4
5    a[1][1] = 1.5;                          /*行列にデータを代入   */
6    a[1][2] = 2.6;
7    a[2][1] = -6.3;
8    a[2][2] = -0.58;
9
10   b[1] = 28.5;                            /*ベクトルにデータを代入   */
11   b[2] = -19.1;
12
13   c[1] = a[1][1]*b[1]+a[1][2]*b[2];       /*行列とベクトルの乗算   */
14   c[2] = a[2][1]*b[1]+a[2][2]*b[2];
15
16   printf("c[1] = %e\n", c[1]);            /*結果表示   */
17   printf("c[2] = %e\n", c[2]);
18
19   return 0;
20 }
```

[**練習 1**] プログラム 3 を参考にして、$\boldsymbol{Mx}$ を計算するプログラムを作成せよ。

$$\boldsymbol{M} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad \boldsymbol{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

4.2.2 フィボナッチ数列

次のサンプルプログラムは、フィボナッチ (Fibonatti) 数列の問題である。

フィボナッチのウサギ

成熟した 1 つがいのウサギは、1ヶ月後に 1 つがいのウサギを生むとする。そして、生まれたウサギは 1ヶ月かけて成熟して次の月から毎月 1 つがいのウサギを生む。全てのウサギはこの規則に従うとし、死ぬことは無いとする。1 つがいのウサギは、1 年後には何つがいになるか。2、3 年後はどうなっているだろうか?。計算してみると分かるが、恐ろしいことになっている。

この数列は単純で、

$$\begin{cases} F_0 = 1 \\ F_1 = 2 \\ F_k = F_{k-1} + F_{k-2} \end{cases} \tag{3}$$

となっている。この単純な数列が、自然界のいろいろな場所でお目にかかれるらしい。かなり不思議なことのようなので、興味のあるものは調べてみると良い。

フィボナッチ数列 F_k を計算するソースをプログラム 4 にしめす。各行の内容は以下の通りである。

9-11 行 `for` 文 (教科書 p.142) は繰り返しに使われる。ここでは、変数 `tuki` の値を 2〜36 まで、一つずつ増加させている。中括弧 { } 内の文を 1 回実行させるたびに、変数 `tuki` の値を増やしている。

プログラム 4: フィボナッチ数列

```
1  #include <stdio.h>
2  int main(){
3    int usagi[100];
4    int tuki;
5
6    usagi[0]=1;
7    usagi[1]=2;
8
9    for(tuki=2; tuki<37; tuki++){
10     usagi[tuki] = usagi[tuki-1] + usagi[tuki-2];
11   }
12
13   printf("after 1 year  : %d\n", usagi[12]);
14   printf("after 2 years : %d\n", usagi[24]);
15   printf("after 3 years : %d\n", usagi[36]);
16
17   return 0;
18 }
```

[**練習 1**] 消費者金融の利子を見ると恐ろしいものがある。CM などをみると年間 20%前後である。100 万円借りた場合、次の単利と複利の場合の 10 年後の利息を計算せよ。

- 単利の場合、元金の 100 万円のみに利子が付く。
- 複利の場合、元金と利息にも利子が付く。

[**練習 2**] 時間が余った者のみ、チャレンジせよ。この問題は参考文献 [1] から引用した。

Bernadelli はある種類のカブトムシについて考察した。そのカブトムシは、3 年間で成長し、3 年目につぎの世代を生んで死亡する。3 年間のうち第一年目で確率 1/2 で生き残り、さらに第 2 年目で 1/3 が生き残り、第 3 年目でそれぞれの雌が 6 匹の雌を生む。これに対応する行列は、

$$\begin{bmatrix} b_{k\,1} \\ b_{k\,2} \\ b_{k\,3} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 6 \\ \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{3} & 0 \end{bmatrix} \begin{bmatrix} b_{k-1\,1} \\ b_{k-1\,2} \\ b_{k-1\,3} \end{bmatrix}$$

とかける。ここで、$b_{k\,1}$ は k 年のときの 1 年目のカブトムシの数である。
1 年目、2 年目、3 年目の虫がそれぞれ 3000 匹いたとしたときその年以後 6 年間の虫の分布を求めよ。

4.3 文字列

これは、数値計算ではあまり使わないので、興味のあるものは教科書を読んでおくように。

参考文献

[1] Gilbert Strang. 線形代数とその応用. 産業図書株式会社, 1992.