C 言語の学習
変数とデータ型・配列と文字列

山本昌志*

2004 年 4 月 25 日

概 要

変数と呼ばれる単純型のデータ構造と配列の学習を行う．プログラムでの変数の意味と型宣言が必要な理由を最初に述べている．その後，変数と配列の使用方法を説明している．

1 ここで理解すべきこと

教科書 [1] の 3-4 章を学習する．ここで，学習すべきことは，変数や配列の使いかたである．変数や配列はデータを格納するものであり，宣言の仕方から，データの出し入れの方法を理解しなくてはならない．各章で，具体的に以下のの内容を理解し，アルゴリズムに利用できるようになるまでを求める．

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 変数とデータ型 (教科書の 3 章)

2.1 プログラムとはなにか

コンピューターは，非常に高速にデータの処理を行う機械である．入力データをもとにさまざまな計算を行い，目的のデータに加工する．たとえば，二次関数の値をコンピューターで計算させる場合，その係数と範囲が入力データで，計算された値が処理された出力データとなる．

データと命令を書いたものをプログラムという．データ処理のためには，この 2 つの要素—データと命令—は必要不可欠である．プログラム中の命令は，`printf` とか，それっぽい英語が書かれているので，ひ

*独立行政法人 秋田工業高等専門学校 電気工学科

とめでわかる．データの方は分かりにくいが，プログラム中に変数として書いてある．今はわからなくてもよいが，本日の学習を進めればすぐにわかるだろう．

2.2 プログラム中のデータ

次のプログラムを見よ．この中の，hoge が変数で，データを表している．そして，printf と return が命令である．このプログラムの動作は単純で，変数 hoge に整数の値 2468 を代入し，その値を表示させている．

```
#include <stdio.h>

int main(void){
  int hoge;

  hoge=246;
  printf("ホゲの値は %d です\n",hoge);

  return 0;
}
```

この例から分かるように，プログラム中の変数には，数値 (データ) を代入することができる．コンピュータープログラムでは，データは変数の中に入れて，その変数を処理するように記述する．プログラム中でデータが丸裸で表れることは良くないとされ，変数の中に隠すのが一般的である．こうすることにより，変数の値を変化させるだけで，プログラムを変更することなく，異なったデータの処理ができる．

データを格納する方法をデータ構造という．ここでしめした例は，最も単純な方法で，ひとつの変数にひとつのデータを格納している．本日は，この単純型と複数のデータを効率良く格納する配列型のデータ構造を学習する．本講義では配列までしか使わないが，問題に応じて，いろいろなデータ構造がある．困難な問題にあたったときには，さらに複雑なデータ構造を学習しなくてはならない．

2.3 変数の型と宣言

単純なイメージ　変数とは数値を入れる箱のようなもので，図 1 にそのイメージを示す．図の左端に書かれている char と int，double は，その変数の型である．それぞれ文字型，整数型，倍精度実数型というもので，箱に格納できるデータの種類を表している．箱の下の文字列は変数名である．文字型の変数名 (c,h,moji) には 1 文字を，整数型の変数 (i,j,seisu) には 1 つの整数を，倍精度実数型の変数名 (x,y,jisu) には 1 つの実数を入れることができる．

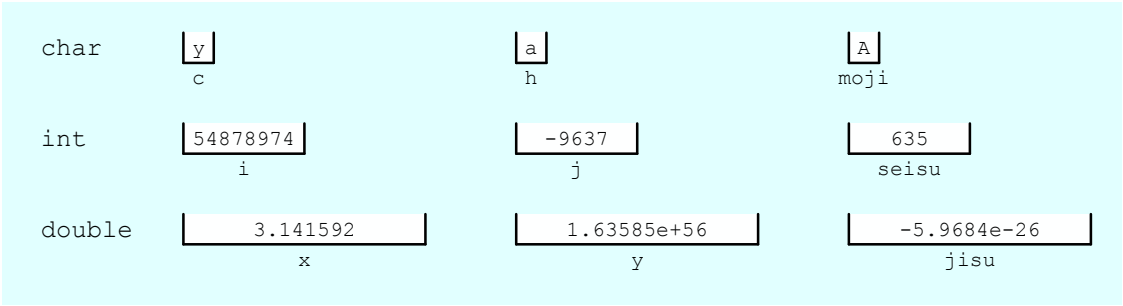

図 1: 変数のイメージ．変数とはデータを入れる箱のようなもの．

型が必要な理由　数学の関数でも変数というものがあり，値を入れることができる．このことは，C 言語の変数と似ている．しかし，数学の変数と決定的に異なることがある．C 言語の変数を使用するとき型を指定しなくてはならない．数学の変数 x であれば，整数でも，実数でも，複素数でも，ベクトルでも $\cdots$ なんでも代入できる．C 言語の場合は，整数型の変数には整数しか代入できない．異なる型のデータを代入すると，コンパイラーが警告を出すか，エラーを出すか，あるいは間違った計算をする可能性がある．

良く考えるとプログラムの動作にはこの型の指定は無意味のように思える．あとで，型を使った操作をしていないからである．また，数学のように，どんなデータでも代入できれば便利である．事実，そういう言語もある．この場合は，計算速度やメモリーの使用効率を犠牲にしている[1]．

コンピューターの都合から，プログラミング言語では型が必要である．このことは，メモリーを考えないと理解できない．先に述べたように，データを格納するものが変数で，実際にはコンピューターのメモリーがその役割を担う．図 1 に示したように，データは型によって大きさ—正確には情報量—が異なる．したがって，1 文字を格納する場合と，1 つの倍精度実数を格納する場合では，必要なメモリーの量が異なる．表 1 に，代表的な型の大きさ—バイト数[2]—をしめす．データを格納するために，メモリーを確保する場合，必要な量をコンピューターに知らせるために，型というものがある．

たいていのプログラミング言語では，変数の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．

表 1: 変数の型と情報料

型名	データ型	バイト数	ビット数
文字型	char	1	8
整数型	int	4	32
倍精度実数	double	8	64

[1] この辺の話は理解でき名手も良い

[2] 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．

メモリーと変数の関係　C 言語の変数には名前と型があり，データを格納している．変数の実体はメモリーで，メモリーにはアドレスと記憶内容がある．それらの関係は，図 2 のようになっている．

- 型に応じて，データが占有するメモリーの大きさが異なる．1 バイトの文字型は，アドレス 1 個分のメモリーを使っている．8 バイトの倍精度実数型の場合，アドレス 8 個分使用している．
- データはメモリーの記憶領域に格納されている．
- 変数名は確保した記憶領域の先頭アドレスを表している．C 言語をコンパイルした機械語では，変数はアドレスに変換されるのである．

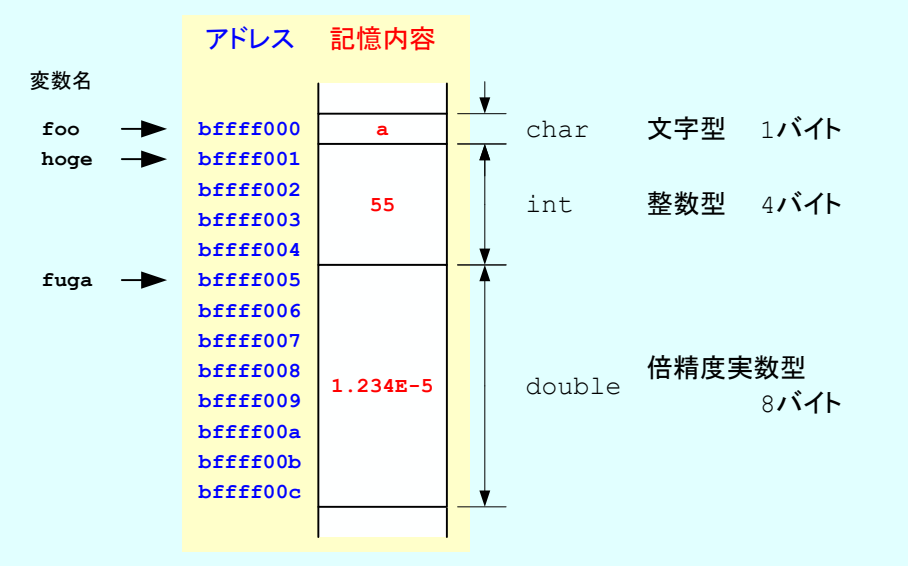

図 2: メモリー中に格納されたデータの例

C 言語の型　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[3]．これは一つの変数の情報量の差から生じる．C 言語の型によって表現できるデータの範囲は，教科書の表 3-1(p.34) の通りである．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．諸君がこの講義で作成するプログラムでは，これで十分である．

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．

[3]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．

変数宣言　C 言語のプログラムで，変数を使う場合，その宣言を最初にしなくてはならない．プログラムを実行する前に，変数のメモリー領域を確保する必要がある．メモリー領域を確保した後，実際の処理が行われるのである．

変数宣言により，変数が使うメモリー領域を確保する．変数宣言の方法は，以下のように型名と変数名を記述するだけである．

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

変数宣言ができる場所は，以下の通りである (教科書の p.54).

1. 関数の外で，宣言を行う (図 3)．これをグローバル変数と言い，どの関数からでもこの変数を使うことができる．
2. 関数の先頭で宣言を行う (図 4)．これはローカル変数と呼ばれ，宣言を行った関数のみで使える．
3. ブロック—{ } で囲まれた部分—の先頭で宣言を行う．変数は，ブロック内のみで有効である．

3 番目は，滅多に使うことがない．1 番目のグローバル変数は，プログラム作法上，好ましくないとされている．諸君は，できるだけ 2 番目のローカル変数を使うことに心がけよ．これら変数宣言とおまじないをあわせると，プログラムの構造は，図 3 や図 4 のようになる．

図 3: おまじないとグローバル変数宣言

図 4: おまじないとグローバル変数宣言

2.4 プログラム例

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リスト 1 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[4]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)．

リスト 1: 変数の学習プログラム

```
1  #include <stdio.h>
2
3  int main(void){
4    char c, h;                    // 文字型変数の宣言
5    int i, j;                     // 整数型変数の宣言
6    double x, y;                  // 倍精度実数型変数の宣言
7
8    c = 'a';
9    h = 'A';
10
11   i = 123;                      // 変数に値の代入
12   j = -987654321;
13
14   x = -1.23456;
15   y = 9.87654321e-12;
16
17   printf("c = %c \t h = %c\n", c, h);
18   printf("i = %d \t j = %d\n", i, j);
19   printf("x = %e \t y = %e\n", x, y);
20
21   return 0;
22 }
```

[**練習 1**] リスト 1 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．

[**練習 2**] 変数 pi に 3.1415 を，変数 e に 2.718281828 を代入し，以下の変数を計算する．

- 変数 wa は，pi+e とする．

[4]記号 ’ をシングルクォーテーションと言う．

– 変数 sa は，pi-e とする．
– 変数 seki は，pi*e とする．
– 変数 shou は，pi/e とする．

これらの 4 つの値を，以下の例のように表示するプログラムを作成せよ．

```
pi+e = 5.859782
pi-e = 0.423218
pi*e = 8.539482
pi/e = 1.155693
```

3 配列と文字列 (教科書の 4 章)

3.1 配列とは

2 節で示した変数[5]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．100 万個のデータを扱う単純型の変数は不可能である．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[6]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 5 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 */
```

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

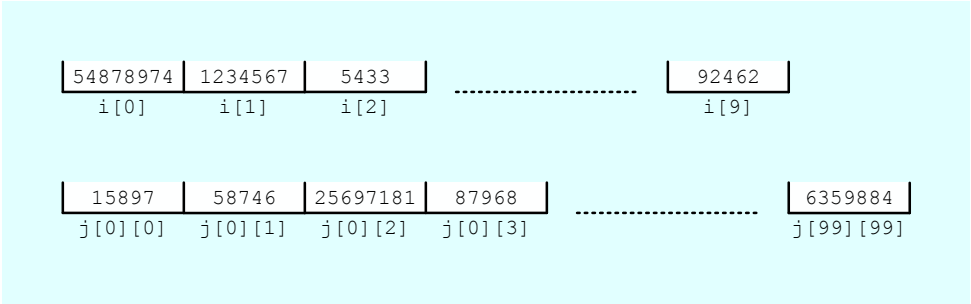

図 5: 配列のイメージ．データを入れる箱がいっぱいある．ただし箱の大きさは全て同じ．

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．

3.2 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 2: 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 3: 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]`
$\vdots$	$\vdots$

3.3 配列を使った例

3.3.1 行列とベクトルの乗算

繰り返し文を使わない例 配列を使って行列とベクトルのかけ算を行うソースをリスト 2 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リスト 2 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.

リスト 2: 行列とベクトルのかけ算

```
1  #include <stdio.h>
2
3  int main(void){
4    double a[3][3],b[3],c[3];
5
6    a[1][1] = 1.5;                          /* 行列にデータを代入 */
7    a[1][2] = 2.6;
8    a[2][1] = -6.3;
9    a[2][2] = -0.58;
10
11   b[1] = 28.5;                            /* ベクトルにデータを代入 */
12   b[2] = -19.1;
13
14   c[1] = a[1][1]*b[1]+a[1][2]*b[2];       /* 行列とベクトルの乗算 */
15   c[2] = a[2][1]*b[1]+a[2][2]*b[2];
16
17   printf("c[1] = %e\n", c[1]);            /* 結果表示 */
18   printf("c[2] = %e\n", c[2]);
19
20   return 0;
21 }
```

繰り返し文を使う例 リスト 2 の方法だと，次元が大きくなると，プログラムを書くことができななくなる．100 次元の行列を考えると，不可能であることがわかる．行列の計算のように，同じような計算を繰り返す場合，ループ文をつかう．ループ文を使うと，リスト 2 は，リスト 3 のように書き換えることができる．15 行目〜19 行目のループ文の内容は，以下の通りである．

15 行 `for` は繰り返しを行え—という命令．繰り返しの回数は，( ) 内に記述する．繰り返しの制御には，変数 `i` を使ってる．

- `i=1` で，`i` の初期値を 1 にしている．
- `i<=2` で，`i` の値が 2 以下ならば，{ } 内を実行する．
- `i++`で，`i` の値を 1，増やしている．`i=i+1` と同じで，`i` の値を 1 増加させたものを，新たな `i` とする．{ } の内部が実行されるたび，この `i++`が実行される．

この最初の繰り返し文で，ベクトル c_1, c_2 を計算することを示している．

16 行 変数 `j` を使って，繰り返しを行っている．ここでは，$c_i = a_{i1}b_1 + c_{i2}b_2$ の演算を行っている．

17 行 行列の計算．`i` と `j` の 2 重ループの内部で，これらの変数が変化しながら計算を行う．演算子`+=`は，累算を示している．例えば，`a+=b` は `a=a+b` とまったく同じでである．これは，右辺の `a+b` を計算して，その結果を左辺の変数 a に代入せよ—と言う意味である．コンピュータ言語で`=`は等しいという

意味はない．右辺の値を計算して，その結果を左辺の変数に代入せよという意味である．非常に紛らわしいが，こうなっている．

18 行 繰り返し制御変数 j を使ったループの終端.

19 行 繰り返し制御変数 i を使ったループの終端.

リスト 3: 繰り返し文を用いた行列とベクトルのかけ算

```
1  #include <stdio.h>
2
3  int main(void){
4    double a[3][3],b[3],c[3];          // 倍精度実数型の配列の宣言
5    int i,j;                           // 整数型変数の宣言
6
7    a[1][1] = 1.5;                     // 行列にデータを代入
8    a[1][2] = 2.6;
9    a[2][1] = -6.3;
10   a[2][2] = -0.58;
11
12   b[1] = 28.5;                       // ベクトルにデータを代入
13   b[2] = -19.1;
14
15   for(i=1; i<=2; i++){
16     for(j=1; j<=2; j++){
17       c[i] += a[i][j]*b[j];          // 行列とベクトルの乗算
18     }
19   }
20
21   printf("c[1] = %e\n", c[1]);       // 結果表示
22   printf("c[2] = %e\n", c[2]);
23
24   return 0;
25 }
```

[**練習 1**] リスト 2 を参考にして，$\boldsymbol{Mx}$ を計算するプログラムを作成せよ.

[**練習 2**] リスト 3 を参考にして，$\boldsymbol{Mx}$ を計算するプログラムを作成せよ.

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

3.3.2 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リスト 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．

リスト 4: フィボナッチ数列

```
1  #include <stdio.h>
2
3  int main(void){
4    int usagi[100];          // 月ごとの兎の数
5    int tuki;                // 月を表す．ループ制御変数
6
7    usagi[0]=1;
8    usagi[1]=2;
9
10   for(tuki=2; tuki<37; tuki++){
11     usagi[tuki] = usagi[tuki-1] + usagi[tuki-2];
12   }
13
14   printf("after 1 year  : %d\n", usagi[12]);
15   printf("after 2 years : %d\n", usagi[24]);
16   printf("after 3 years : %d\n", usagi[36]);
17
18   return 0;
19 }
```

[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%くらいである．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．

消費者金融の利子は複利である．可愛いお姉さんが宣伝しているが，かなりの利子がつくことが分かるであろう．

[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 [2] 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．

3.4 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] 林春比古. 新訂 C 言語入門 シニア編. ソフトバンク パブリッシング, 2004.

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