第4回　配列

1

今回の目標

- １次元配列を理解する。
- ２次元配列を理解する。
- 文字列の入出力を理解する。

☆２×２の行列の行列式を計算する
プログラムを作成する

2

配列宣言

配列：同じ型の変数（配列の要素）の集まり。

配列宣言によって、変数を一度にたくさん作ることができる。

配列宣言の書式

要素のデータ型　配列名[要素数];

配列宣言の例

```
double   d[ 5 ];
```

配列名　要素数

注意：
1. 変数は「要素数」の個数だけ作られる。
2. 要素数は定数（変数で指定できない）

double 型の変数が5個作られる。

3

配列要素の使い方

配列：同じ型の変数（配列の要素）の集まり。

配列名に添え字を与えることで、配列の要素を指定できる。
配列要素は式の中で通常の変数と同様に使うことができる。

配列要素の書式

配列名[添え字]

配列要素の使用例

```
d[3] = 12.3;
```

配列名　添え字

注意：
1. 添え字は整数型の式。
2. 添え字の値は 0 ～ (要素数-1)。

dはdouble 型の要素（5個）を持つ配列。

4

プログラム例1（配列利用の練習）

```
/* 配列利用実験  days.c  コメント省略        */

#include <stdio.h>

/* 1年の月数、配列daysの要素数 */
#define  MONTHS  12

int  main()
{
      double  days[MONTHS]; /* 各月の日数 */
                                /* days[i] : (i+1)月の日数 */

      int  month ; /* 入力された月の番号（1～12）*/
                  /* (month-1) が配列 days の添え字になる */

/* 続く */
```

配列の要素数は、通常、マクロ定義した定数で与える。

5

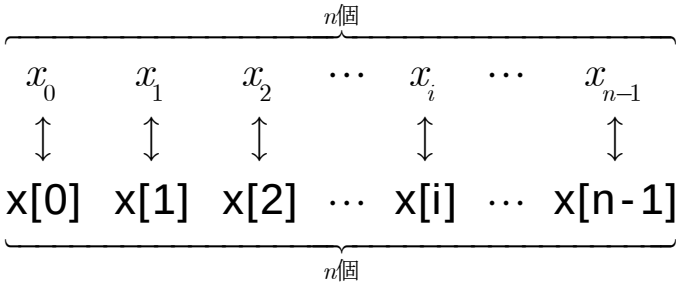
```
/* 続き */

      days[0] = 31; /* 1月の日数 */
      days[1] = 28; /* 2月の日数 */
      days[2] = 31; /* 3月の日数 */
      days[3] = 30; /* 4月の日数 */
      days[4] = 31; /* 5月の日数 */
      days[5] = 30; /* 6月の日数 */
      days[6] = 31; /* 7月の日数 */
      days[7] = 31; /* 8月の日数 */
      days[8] = 30; /* 9月の日数 */
      days[9] = 31; /* 10月の日数 */
      days[10] = 30; /* 11月の日数 */
      days[11] = 31; /* 12月の日数 */

      printf("何月の日数？(1～12):¥n");
      scanf("%d", &month);

      printf("%d月の日数は、", month);
      printf("%d日です。¥n", days[month-1] );

      return  0;
}
```

月の番号と、配列の添え字が、1ずれていることに注意。

配列の添え字には、変数を使った式を与えることもできる。

6

数学の添え字付き変数とC言語の配列



7

2次元配列

宣言:

```
要素のデータ型　配列名[行の要素数][列の要素数];
```

例

```
#define     TATE   5
#define     YOKO   3

double      m[TATE][YOKO];
```

使いかた:

```
配列名[添字1][添字2]
```

で普通の変数のように使える。
また、添字には（整数型の）
変数や式も使える。

8

2次元配列のイメージ

2次元配列の宣言:
double　m[TATE][YOKO];

*j*列

*i*行						
	m[0][0]	m[0][1]	・・・	m[0][*j*]	・・・	
	m[1][0]	m[1][1]	・・・	m[1][*j*]	・・・	
	・ ・ ・	・ ・ ・		・ ・ ・		
	m[*i*][0]	m[*i*][1]	・・・	m[*i*][*j*]	・・・	
	・ ・ ・	・ ・ ・		・ ・ ・		
						m[TATE-1][YOKO-1]

9

プログラム例2（2次元配列利用の練習）

```
/*  tenchi.c    2次元配列実験(転置行列)
        コメント省略
*/
#include <stdio.h>
#define  GYO  2  /* 入力される行列の行の数 */
#define  RETU 2  /* 入力される行列の列の数 */

int  main()
{
        /* 配列の宣言 */
        doulbe x[GYO][RETU];        /*入力された行列*/
        double tx[RETU][GYO];       /*転置行列*/

/* 続く */
```

10

```
/* 続き */

        /*  行列の要素の入力 */
        printf("x[0][0]?");
        scanf("%lf", &x[0][0]);
        printf("x[0][1]?");
        scanf("%lf", &x[0][1]);
        printf("x[1][0]?");
        scanf("%lf", &x[1][0]);
        printf("x[1][1]?");
        scanf("%lf", &x[1][1]);

        /* 転置行列の計算 */
        tx[0][0] = x[0][0];
        tx[0][1] = x[1][0];
        tx[1][0] = x[0][1];
        tx[1][1] = x[1][1];

/* 続く */
```

11

```
/* 続き */

        /* 入力された行列と、その転置行列の表示*/
        printf("x¥n");
        printf("%6.2f %6.2f ¥n", x[0][0], x[0][1]);
        printf("%6.2f %6.2f ¥n", x[1][0], x[1][1]);

        printf("xの転置行列¥n");
        printf("%6.2f %6.2f ¥n", tx[0][0], tx[0][1]);
        printf("%6.2f %6.2f ¥n", tx[1][0], tx[1][1]);

        return 0;
}
```

12

多次元配列

宣言:

データ型 配列名[次元1の要素数][次元2の要素数][次元3の要素数]･･･;

例

```
#define     TATE    5
#define     YOKO    4
#define     OKU     3

double      cube[TATE][YOKO][OKU];
```

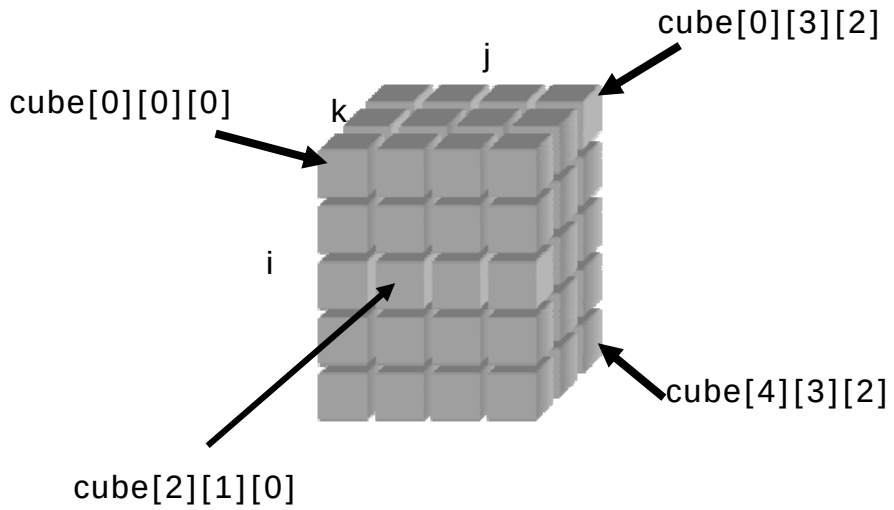
使いかた:

配列名[添字1][添字2][添字3]･･･

で普通の変数のようにつかえる。
また、添え字には(整数型の)
変数や式も使える。

13

3次元配列のイメージ



14

定数の分類とデータ型

定数:プログラム中で、常に特定の値を持つ式

定数の種類	表現の対象	データ型	定数の記述例
数値定数	整数	int	123
			0
	浮動小数点数（実数）	double	12.34
			1.0e-5
			0.0
文字定数	1文字	char	'a'
			'¥n'
	文字列	char*	"abc"
			"Hello¥n"

15

プログラム（ソースコード）内での文字定数の表現

（1文字 : char 型）

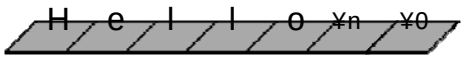
シングルクォート「'」で囲まれた、1バイト文字またはエスケープ文字

'A'

（文字列 : char* 型）

ダブルクォート「"」で囲まれた、文字列（エスケープ文字を含んでもよい）

"Hello¥n"



char* 型は、正しくは「char型変数のアドレス」型。
詳しくは、第13回（ポインタ）で学習する。

16

エスケープ文字

ソースコード内で¥（バックスラッシュ）で始まる文字列（2文字）は、
コンピュータの内部では一つの記号を表す。
このような文字をエスケープ文字という。

エスケープ文字集

¥n	改行
¥t	タブ
¥b	バックスペース
¥0	終端文字

¥¥	バックスラッシュ
¥'	シングルクォーテーション
¥"	ダブルクォーテーション

¥（バックスラッシュ）で始まる文字列は特別な意味を持つ。
と考えても良い。

なお、この資料では、「¥」でバックスラッシュを表わす。
UNIX上では、「 ＼ 」がバックスラッシュを表す。

17

printf文による文字型の値の表示

```
printf("こんにちは %s さん。¥n", name);
```

文字列を標準出力（ディスプレイ）に出力するライブラリ関数

変換仕様 printf文の文字列内の「 **% 変換文字**」
後ろの変数に関する出力指示を表わす

- 文字列

%s char*型の文字列を表示
%6s 文字列を表示、少なくとも6文字幅で表示

18

標準入力から文字配列への文字列の読み込み

scanf文を用いると、
標準入力（キーボード）から
char型の配列に文字列を代入できる。

- 「"」と「"」の間に**変換仕様だけを**記述
- カンマの後に「**配列名**」

文字列を読み込む場合のscanfの使い方

```
scanf("%バイト数s", 配列名);
```

「&」が付かないので注意！

19

scanf文による文字型の値の読み込み

```
scanf("%127s", name);
```

標準入力（キーボード）から変数に値を読み込むライブラリ関数

変換仕様 scanf文の文字列内の「 **% 変換文字**」
scanfの「”」と「”」の間には**変換仕様しか書かないこと**

- 文字列

%127s char型の配列（要素数128以上）に、
最大127バイトの文字列を入力

配列に文字列を読み込む場合には、

- 最大何バイト読み込めるか（配列の要素数-1）を指定。
（変換文字中ではマクロ定義を使えないので注意）
- 配列名には「＆」を付けない。

20

プログラム例3（文字列入出力の練習）

```
/* 文字列入出力実験 echo_name.c コメント省略 */
#include <stdio.h>

/* 配列nameの要素数 */
#define NAMESIZE 128

int main()
{
    char name[NAMESIZE]; /* 入力された名前（127バイトまで）を格納 */

    printf("あなたの名前は？¥n");

    scanf("%127s", name);
    printf("こんにちは、%sさん。¥n", name);

    return 0;
}
```

21

プログラム例4の原理：行列と行列式

$$\boldsymbol{X} = \begin{pmatrix} x_{00} & x_{01} \\ x_{10} & x_{11} \end{pmatrix}$$ のとき、

$\boldsymbol{X}$ の行列式 $\left|\boldsymbol{X}\right|$ は次式で定義される。

$$\left|\boldsymbol{X}\right| = \begin{vmatrix} x_{00} & x_{01} \\ x_{10} & x_{11} \end{vmatrix} = x_{00}x_{11} - x_{01}x_{10}$$

22

プログラム例4:行列式の値を計算するプログラム

```
/*
        作成日: yyyy/mm/dd
        作成者:本荘太郎
        学籍番号:B00B0xx
        ソースファイル:determinant.c
        実行ファイル:determinant

        説明:2×2行列の行列式を計算するプログラム。

        入力:   標準入力から行列の4つの成分を入力する。
                行列の成分は全て任意の実数値とする。
                同じ行の要素が連続して入力されるとする。

        出力:標準出力に入力された行列の行列式の値を出力する。
                行列式の値は実数値であり、小数点以下2桁まで出力する。
*/

/* 続く */
```



```
/* 続き */

#include <stdio.h>

/*マクロ定義*/
#define   SIZE   2 /* 行列の次元 */

int  main()
{
        /* 変数、配列の宣言 */
        double  matrix[SIZE][SIZE];
                                        /* 入力された行列 */
        double  det; /* 行列matrixの行列式の値 */

/* 続く */
```

```
/* 続き */

        /* 行列の各成分の入力*/
        printf("matrix[0][0]?");
        scanf("%lf", &matrix[0][0]);
        printf("matrix[0][1]?");
        scanf("%lf", &matrix[0][1]);
        printf("matrix[1][0]?");
        scanf("%lf", &matrix[1][0]);
        printf("matrix[1][1]?");
        scanf("%lf", &matrix[1][1]);

/* 続く */
```

25

```
/* 続き */

        /*行列式の計算*/
        det = matrix[0][0]*matrix[1][1]
                - matrix[0][1]*matrix[1][0];

        printf("行列 matrix :¥n");
        printf("%6.2f %6.2f ¥n",
                matrix[0][0], matrix[0][1]);
        printf("%6.2f %6.2f ¥n",
                matrix[1][0], matrix[1][1]);
        printf("の行列式の値は、¥n");
        printf("%6.2f です。¥n", det);

        return  0;
}
```

26

プログラム例4の実行結果

```
$./determinant
matrix[0][0]?1.0
matrix[0][1]?2.0
matrix[1][0]?3.0
matrix[1][1]?4.0
行列 matrix :
   1.00  2.00
   3.00  4.00
の行列式の値は
 -2.00です。
$
```