

配列

三池 克明

多数のデータを扱うには配列が便利です。
配列とは番号が振られた変数だと思って下さい。

目 次

1.	配列変数とは？	1
1.1.	配列を使う、その1	1
1.2.	配列を使う、その2	3
2.	文字型の配列 文字列	5
2.1.	ヌル文字「¥0」	5
2.2.	" (ダブルクォーテーション) で囲んだ文字列の扱い	7
3.	配列の活用	9
3.1.	一括で同じ演算を行う	9
3.2.	#define を使って改良	11
3.3.	文字列処理	16
4.	演習問題	18

1. 配列変数とは？

配列変数とは番号が振られた変数です。
まずは実際に使ってみましょう。

1.1. 配列を使う、その1

まずは以下のソースプログラムを入力し、コンパイル、実行してみましょう。

array1.cpp

```
1: #include <stdio.h>
2:
3: main()
4: {
5:     double  data[5];
6:     int     i;
7:
8:     printf("5つの実数を入力して下さい。¥n");
9:     for(i = 0; i < 5; i++){
10:         printf("data[%d] = ", i);
11:         scanf("%lf", &data[i]);
12:     }
13:
14:     printf("入力した実数は以下のとおりです。¥n");
15:     for(i = 0; i < 5; i++){
16:         printf("data[%d] = %f¥n", i, data[i]);
17:     }
18: }
```

実行例

5つの実数を入力して下さい。

data[0] = 1

data[1] = 3.14

data[2] = -2.71828

data[3] = 50

```
data[4] = 62
```

入力した実数は以下のとおりです。

```
data[0] = 1.000000
```

```
data[1] = 3.140000
```

```
data[2] = -2.718280
```

```
data[3] = 50.000000
```

```
data[4] = 62.000000
```

変数に番号を振ることで「i 番目の値」といった記述が可能になります。

よって for 文などの繰り返し処理と併用することで大量のデータ処理を短いコードで記述することも可能です。

以下は配列を宣言するときの構文です。

構文：

```
型 変数名[要素数];
```

例) `double data[5];`

`double` 型変数 `data[0] ~ data[4]` の 5 つの配列変数が宣言される。

また、添字 (`data[1]` なら添字は 1) の範囲は `[0] ~ [要素数 - 1]` であることに注意する。

1.2. 配列を使う、その2

続いて配列を宣言と同時に値を代入する場合の例です。

array2.cpp

```
1: #include <stdio.h>
2:
3: main()
4: {
5:     int    score[] = {82, 65, 91, 79, 63};
6:     int    no;
7:
8:     printf("番号を入力して下さい:");
9:     scanf("%d", &no);
10:
11:     printf("%d 番の点数は、%d 点です。¥n", no, score[no - 1]);
12: }
```

実行例 1

```
番号を入力して下さい:1
1 番の点数は、82 点です。
```

実行例 2

```
番号を入力して下さい:5
5 番の点数は、63 点です。
```

実行例 3

```
番号を入力して下さい:12
12 番の点数は、7 点です。
```

このように、中カッコ{ }で囲むことで宣言と同時に値を代入することも可能です。また、この場合は要素数を省略することも可能です。

ここで実行例 3 をもう一度見てみましょう。

実行例 3

番号を入力して下さい:12

12 番の点数は、7 点です。

配列 `score[]` の要素数は 5 なので `score[0]` ~ `score[4]` までしか存在しないはずで
す。しかし C 言語では添字が範囲の中にあるか外にあるかをチェックしません。

よってこの実行例 3 のようなことも起こりえます。

しかしこれは危険なことです。配列を扱うときには十分注意しましょう。

2. 文字型の配列 文字列

C 言語では char 型配列を「文字列」と呼びます。
また文字列の定数は”（ダブルクォーテーション）で囲みます。
つまり printf()などで多用していたのは char 型配列の定数です。

2.1. ヌル文字「¥0」

まずは char 型配列変数を使ったプログラムの例です。

array3.cpp

```
1: #include <stdio.h>
2:
3: main()
4: {
5:     char    s[] = { 'H', 'e', 'l', 'l', 'o', '\n', '¥0' };
6:
7:     printf(s);
8: }
```

実行結果

Hello

それではソースコードをたどってみましょう。
まずは5行目です。

```
4: {
5:     char    s[] = { 'H', 'e', 'l', 'l', 'o', '\n', '¥0' };
6:
```

s[]の仕様を宣言し、s[0]から順に'H'、'e'、'l'、'l'、'o'、'\n'、'¥0'と値（ここでは文字型定数）を代入しているのが分かります。

ところで最後の'¥0'とは何なのでしょう？

これは「ヌル文字」というもので文字列の最後をあらわします。

なぜ、そのようなものが必要なのでしょう？

それは7行目の処理を解説しながら説明しましょう。

```
6:
7:     printf(s);
8: }
```

これは関数 `printf()` を使って文字列変数 `s[]` の内容をそのまま表示させています（`[]` が付いてないことに注意）。

ただし、これだけでは何文字表示すればよいのかが分かりません。

そこで `printf()` は `s[0]`、`s[1]`、`...` と '¥0' になるまで一文字ずつ表示していきま

す。

よって文字列を扱うには、

- 文字数+1 の配列を確保する
- 一番最後には必ず '¥0' を代入する

に注意する必要があります。

2.2. " (ダブルクォーテーション) で囲んだ文字列の扱い

続いてこちらのソースプログラムも入力し、コンパイル、実行してみましょう。

array4.cpp

```
1: #include <stdio.h>
2:
3: main()
4: {
5:     char    s[] = "Hello\n";
6:
7:     printf("%s", s);
8: }
```

実行結果

Hello

array3.cpp と同じ実行結果になりました。
それではソースコードをたどってみましょう。
まずは5行目です。

```
4: {
5:     char    s[] = "Hello\n";
6:
```

char 型配列変数を宣言と同時に代入する場合、

```
char    配列名[] = "文字列";
```

と文字列をダブルクォーテーションで囲むことで簡潔に記述することが可能です。なおヌル文字は自動で追加されます。

ただしこれは宣言と同時に代入する場合に限ります。

次に7行目です。

```
6:  
7:     printf("%s", s);  
8: }
```

関数 `printf()` で文字列変数を表示するには

- `%s` を使う
- `[]` は不要

としなければなりません。詳しい解説は別の機会に譲ります。

3. 配列の活用

3.1. 一括で同じ演算を行う

以下は五つの値を配列に代入し、それぞれを 10 倍した値を別の配列に代入するプログラムです。

array5.cpp

```
1: #include <stdio.h>
2:
3: main()
4: {
5:     double  a[5], b[5];
6:     int      i;
7:
8:     printf("5個の実数を入力して下さい。¥n");
9:     for(i = 0; i < 5; i++){
10:         printf("a[%d] = ", i);
11:         scanf("%lf", &a[i]);
12:     }
13:
14:     for(i = 0; i < 5; i++){
15:         b[i] = a[i] * 10.0;
16:     }
17:
18:     printf("aを10倍したbは以下のとおりです。¥n");
19:     for(i = 0; i < 5; i++){
20:         printf("b[%d] = %f¥n", i, b[i]);
21:     }
22: }
```

実行例

5個の実数を入力して下さい。

a[0] = 0.1

a[1] = 0.2

a[2] = 0.3

a[3] = 0.4

a[4] = 0.5

a を 10 倍した b は以下のとおりです。

b[0] = 1.000000

b[1] = 2.000000

b[2] = 3.000000

b[3] = 4.000000

b[4] = 5.000000

入力した値が確かに 10 倍になっていますね。

3.2. #define を使って改良

#define を使ってソースコードを読みやすくします。
またそれだけでなく、コードの書き換えも容易になります。

array5.cpp を一部修正したのが以下の array6.cpp です。
以下のソースプログラムを入力し、コンパイル、実行してみましょう。

array6.cpp

```
1: #include <stdio.h>
2:
3: #define SIZE    5
4:
5: main()
6: {
7:     double  a[SIZE], b[SIZE];
8:     int     i;
9:
10:    printf("%d 個の実数を入力して下さい。 %n", SIZE);
11:    for(i = 0; i < SIZE; i++){
12:        printf("a[%d] = ", i);
13:        scanf("%lf", &a[i]);
14:    }
15:
16:    for(i = 0; i < SIZE; i++){
17:        b[i] = a[i] * 10;
18:    }
19:
20:    printf("a を 10 倍した b は以下のとおりです。 %n");
21:    for(i = 0; i < SIZE; i++){
22:        printf("b[%d] = %f %n", i, b[i]);
23:    }
24: }
```

5 個の実数を入力して下さい。

a[0] = 0.1

a[1] = 0.2

a[2] = 0.3

a[3] = 0.4

a[4] = 0.5

a を 10 倍した b は以下のとおりです。

b[0] = 1.000000

b[1] = 2.000000

b[2] = 3.000000

b[3] = 4.000000

b[4] = 5.000000

array5.cpp と同じ動作をしているようです。

それでは#define について説明しましょう。

構文は以下の通りです。

構文：

```
#define 文字列A 文字列B
```

この構文以後にて文字列 A を文字列 B に置換えてコンパイルする。

これをマクロ定義と呼ぶ。

なお、文字列 A はアルファベット大文字を使うのが通例である。

それではソースコードをたどってみましょう。

まずは3行目です。

```
2:
3: #define SIZE    5
4:
```

「#define SIZE 5」となっています。
よって以後「SIZE」という文字列はすべて「5」と置換えられます。

次に7行目です。

```
6: {
7:     double  a[SIZE], b[SIZE];
8:     int     i;
```

「SIZE」がありますね。この部分は3行目で定義してあるとおり。

```
double a[ 5 ], b[ 5 ];
```

と置換えられます。
これ以降も同じなので特に説明する必要はありませんね。

この array6.cpp は5個の配列を扱っていますが、急遽「配列の大きさを10に変更してほしい」と仕様変更を指示されたらどうなるでしょうか。

配列の宣言やメッセージの文章、そして for 文の条件式をすべて書き換えなければなりませんね。これは大変な作業です。

しかし、#define であらかじめマクロ定義を済ませていればわずかな変更で対応できます。

それが以下の array7.cpp です。

```
1: #include <stdio.h>
2:
3: #define SIZE    10
4:
5: main()
6: {
7:     double  a[SIZE], b[SIZE];
8:     int     i;
9:
10:    printf("%d 個の実数を入力して下さい。¥n", SIZE);
11:    for(i = 0; i < SIZE; i++){
12:        printf("a[%d] = ", i);
13:        scanf("%lf", &a[i]);
14:    }
15:
16:    for(i = 0; i < SIZE; i++){
17:        b[i] = a[i] * 10;
18:    }
19:
20:    printf("a を 10 倍した b は以下のとおりです。¥n");
21:    for(i = 0; i < SIZE; i++){
22:        printf("b[%d] = %f¥n", i, b[i]);
23:    }
24: }
```

実行例

10 個の実数を入力して下さい。

```
a[0] = 1
a[1] = 2
a[2] = 3
a[3] = 4
a[4] = 5
a[5] = 6
a[6] = 7
a[7] = 8
a[8] = 9
```

```
a[9] = 0
```

a を 10 倍した b は以下のとおりです。

```
b[0] = 10.000000
```

```
b[1] = 20.000000
```

```
b[2] = 30.000000
```

```
b[3] = 40.000000
```

```
b[4] = 50.000000
```

```
b[5] = 60.000000
```

```
b[6] = 70.000000
```

```
b[7] = 80.000000
```

```
b[8] = 90.000000
```

```
b[9] = 0.000000
```

3 行目の「5」を「10」に変更しただけで対応できました。

このように#define でマクロ定義をすることにより、

- ソースコード上で使う定数の意味が分かりやすくなる
- ソースコードの変更が容易になる

と、プログラミングが容易になります。

プログラムの変更

顧客の要求、思わぬバグの発見などによりプログラムの変更はよくあります。
(それが完成期限ぎりぎりであっても)

ですからプログラミングの世界において「バグが無いプログラミング」と同様に「変更が容易なソースコード」を記述できるテクニックも大切です。

3.3. 文字列処理

今度は入力した文字列の内容と文字数を確認するプログラムを作ってみましょう。

array8.cpp

```
1: /*文字数を調べる*/
2:
3: #include <stdio.h>
4:
5: #define SIZE    128
6:
7: main()
8: {
9:     char    a[SIZE];
10:    int     len;
11:
12:    printf("文字列 (最大%d 文字) を入力して下さい。:", SIZE - 1);
13:    scanf("%s", a);
14:
15:    for(len = 0; a[len] != '\0'; len++){
16:    }
17:
18:    printf("入力した文字列:%s、文字数:%d\n", a, len);
19: }
```

実行例

```
文字列 (最大 127 文字) を入力して下さい。:qwertyuiop@[
入力した文字列:qwertyuiop@[、文字数:12
```

どうやら問題なさそうです。

なお関数 `scanf()` で文字列を入力させる場合は、

- 格納する変数は文字列 (char 型配列) 変数にする
- `%s` を使う
- 変数名に `&` は不要
- 配列変数であっても `[]` は不要

としなければなりません。解説は別の機会に譲ります。

4. 演習問題

8-1. int 型配列 data1 を以下の仕様に従って変換し int 型配列 data2 に書き込むプログラム「ensyu8 -1.cpp」を作りなさい。なお、以下の仕様と実行例に従うこと。

<仕様>

1. data1 および data2 の要素数は5とする。
2. data1 の各値はキーボードで入力させる。
3. data1 から data2 に値を移すとき、(a)奇数なら二倍に、(b)偶数なら半分にする

実行例

5 個の整数を入力して下さい。

```
data1[0] = 1
```

```
data1[1] = 2
```

```
data1[2] = 3
```

```
data1[3] = 4
```

```
data1[4] = 5
```

変換後の値は以下のとおりです。

```
data2[0] = 2
```

```
data2[1] = 1
```

```
data2[2] = 6
```

```
data2[3] = 2
```

```
data2[4] = 10
```

ヒント：

2 で割ったときに、余りがあれば奇数、無ければ偶数ですよね。

8-2. 前問で作成したプログラムの仕様を以下のものに変更したプログラムラム「ensyu8 -2.cpp」を作りなさい。

<仕様（変更部分、他は前問と同じ）>

1. data1 および data2 の要素数は8とする。