

# 関数と配列

三池 克明

配列を引数にする関数の性質について解説します。  
ただの変数を引数にした場合とは少し性質が異なりますので、その点に注意  
しましょう。

## —目 次—

1.	関数と配列.....	1
1.1.	引数の内容を書き換える（配列の場合）.....	1
2.	関数を使ったプログラム.....	4
2.1.	int 型配列の最大値を求める.....	4
2.2.	double 型配列の合計を求める.....	9
2.3.	文字列の長さを得る.....	13
3.	演習問題.....	17

## 1. 関数と配列

関数の引数は変数そのものではなく、その値を渡すことは前回解説したとおりです。では引数が配列の場合もそうなのでしょうか？

### 1.1. 引数の内容を書き換える（配列の場合）

ここで再度 `twice1.cpp` を確認しましょう。

`twice1.cpp`

```
1: #include <stdio.h>
2:
3: void twice(double);
4:
5: main()
6: {
7:     double a;
8:
9:     printf("a = ");
10:    scanf("%lf", &a);
11:
12:    twice(a);
13:
14:    printf("a = %f\n", a);
15: }
16:
17: void twice(double a)
18: {
19:     a *= 2;
20: }
```

実行例

```
a = 0.8
a = 0.800000
```

このように関数 `main()` の変数 `a` は2倍にならないのを確認したと思います。では、配列を引数にした場合も同じ結果になるかどうかを確かめてみましょう。

twice() は1つの変数が対象でしたが、今度は配列変数の各値を二倍する関数 twicearray() を作ってみましょう。

仕様は以下のとおりです。

プロトタイプ	void twicearray(double array[], int n)
引数	double array[]   二倍にしたい配列
	int n           array[]の大きさ
返値	void

ソースコードは以下のとおりです。入力しコンパイル・実行してみましょう。

twicearray.cpp

```
1: #include <stdio.h>
2:
3: #define SIZE    5
4:
5: void twicearray(double [], int);
6:
7: main()
8: {
9:     double  a[SIZE];
10:    int      i;
11:
12:    for(i = 0; i < SIZE; i++) {
13:        printf("a[%d] = ", i);
14:        scanf("%lf", &a[i]);
15:    }
16:
17:    twicearray(a, SIZE);
18:
19:    for(i = 0; i < SIZE; i++) {
20:        printf("a[%d] = %f¥n", i, a[i]);
21:    }
22: }
23:
24: void twicearray(double array[], int      n)
25: {
```

```
26:         int    i;
27:
28:         for(i = 0; i < n; i++){
29:             array[i] *= 2;
30:         }
31:     }
```

## 実行例

```
a[0] = 0.9
a[1] = 1
a[2] = 1.1
a[3] = 1.2
a[4] = 1.3
a[0] = 1.800000
a[1] = 2.000000
a[2] = 2.200000
a[3] = 2.400000
a[4] = 2.600000
```

ソースコードを見ると `twice1.cpp` と同じ記述であることがわかります。  
であるならばこちらも失敗するはずですが、成功していますね。  
この理由については後述しますので、とりあえず今回は、

- 変数を引数にすると、その値が関数に渡される
- しかし配列変数を引数にすると、値ではなく配列変数そのものが関数に渡される

と覚えて下さい。

## 2. 関数を使ったプログラム

配列を引数とする関数のプログラム例です。

### 2.1. int 型配列の最大値を求める

以下の関数 `intsaidai()` を作り、動作を確かめてみましょう。

プロトタイプ	<code>int intsaidai(int array[], int n)</code>	
引数	<code>int array[]</code>	最大値を求める配列
	<code>int n</code>	<code>array[]</code> の大きさ
返値	<code>int</code>	<code>array[]</code> の最大値
<b>【解説】</b> 配列の最大値を求める。		

この仕様より、プロトタイプ宣言は、

```
int intsaidai(int [], int);
```

とすればよいことがわかります。また、関数を記述するときは、

```
int intsaidai(int array[], int n)
{
    (ここに処理を記述)
}
```

とすればよいことがわかります

そして最終的な `intsaidai()` の仕様は以下のようになります。

<関数 intsaidai() の仕様>

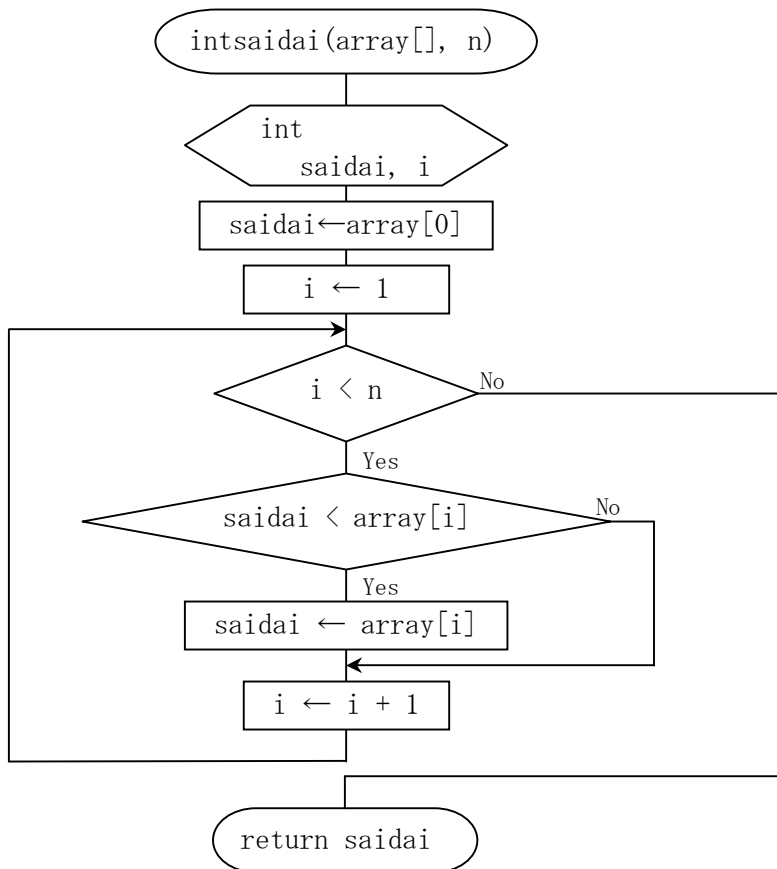
1. 外部向けの仕様

<b>プロトタイプ</b>	<code>int intsaidai(int array[], int n)</code>	
<b>引数</b>	<code>int array[]</code>	最大値を求める配列
	<code>int n</code>	array[] の大きさ
<b>返値</b>	<code>int</code>	array[] の最大値
<p><b>【解説】</b> 配列の最大値を求める。</p> <p><b>【例】</b></p> <pre style="text-align: center;">max = intsaidai(a, 100);</pre> <p>配列 a[] の最大値を求め、max に代入します。</p>		

2. 関数内で使用する変数

変数名	型	概要
saidai	int	最大値
i	int	カウンタ

### 3. フローチャート



### 4. ソースコード

```
int intsaidai(int array[], int n)
{
    int    i, saidai;

    saidai = array[0];
    for(i = 1; i < n; i++){
        if(saidai < array[i]){
            saidai = array[i];
        }
    }
}
```

```

    }

    return saidai;
}

```

### <以上>

それでは intsaidai() の動作確認プログラムを作りましょう。  
以下のプログラムを入力し、コンパイル・実行します。

intsaidai.cpp

```

1: #include <stdio.h>
2:
3: #define SIZE    5
4:
5: int intsaidai(int [], int);
6:
7: main()
8: {
9:     int    a[SIZE];
10:    int    i;
11:
12:    for(i = 0; i < SIZE; i++){
13:        printf("a[%d] = ", i);
14:        scanf("%d", &a[i]);
15:    }
16:
17:    printf("最大値は%dです。¥n", intsaidai(a, SIZE));
18: }
19:
20: int intsaidai(int array[], int n)
21: {
22:     int    i, saidai;
23:
24:     saidai = array[0];
25:     for(i = 1; i < n; i++){
26:         if(saidai < array[i]){
27:             saidai = array[i];

```

```
28:         }  
29:     }  
30:  
31:     return saidai;  
32: }
```

#### 実行例

a[0] = -5

a[1] = -8

a[2] = 20

a[3] = 45

a[4] = 9

最大値は 45 です。

どうやら問題なさそうですね。

## 2.2. double 型配列の合計を求める

過去に配列の値の合計を求めるプログラムをつくりましたが、今回はそれを関数化してみましょう。

ここでは double 型配列の合計を求める関数 `goukei()` とします  
また `goukei()` の仕様は以下のとおりです。

<関数 `goukei()` の仕様>

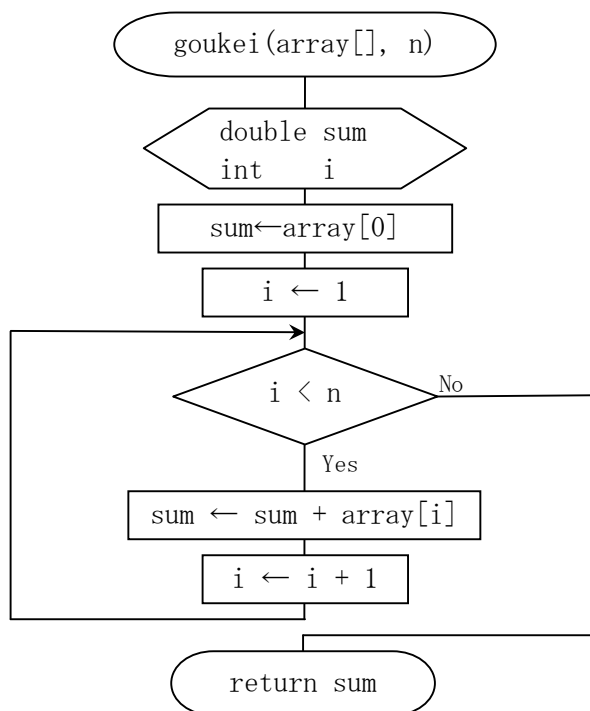
### 1. 外部向けの仕様

プロトタイプ	<code>double goukei(double array[], int n)</code>	
引数	<code>double array[]</code>	合計を求める配列
	<code>int n</code>	<code>array[]</code> の大きさ
返値	<code>double</code>	<code>array[]</code> の合計
<b>【解説】</b> 配列の合計を求める。		
<b>【例】</b> <pre>sum = goukei(a, SIZE);</pre> 配列 <code>a[]</code> の合計求め、 <code>sum</code> に代入します。		

### 2. 関数内で使用する変数

変数名	型	概要
<code>sum</code>	<code>double</code>	合計
<code>i</code>	<code>int</code>	カウンタ

### 3. フローチャート



### 4. ソースコード

```
double goukei(double array[], int n)
{
    double sum;
    int i;

    sum = array[0];
    for(i = 1; i < n; i++){
        sum += array[i];
    }

    return sum;
}
```

<以上>

それでは goukei () の動作確認プログラムを作りましょう。  
以下のプログラムを入力し、コンパイル・実行します。

goukei.cpp

```
1: #include <stdio.h>
2:
3: #define SIZE    5
4:
5: double goukei(double [], int);
6:
7: main()
8: {
9:     double  data[SIZE];
10:    int      i;
11:
12:    for(i = 0; i < SIZE; i++){
13:        printf("data[%d] = ", i);
14:        scanf("%lf", &data[i]);
15:    }
16:
17:    printf("合計 = %g\n", goukei(data, SIZE));
18: }
19:
20: double goukei(double array[], int n)
21: {
22:     double  sum;
23:     int      i;
24:
25:     sum = array[0];
26:     for(i = 1; i < n; i++){
27:         sum += array[i];
28:     }
29:
30:     return  sum;
31: }
```

## 実行例

```
data[0] = 10.5  
data[1] = -222  
data[2] = 55.235  
data[3] = 33.3  
data[4] = 83.2  
合計 = -39.765
```

どうやら問題なさそうですね。

---

## 2.3. 文字列の長さを得る

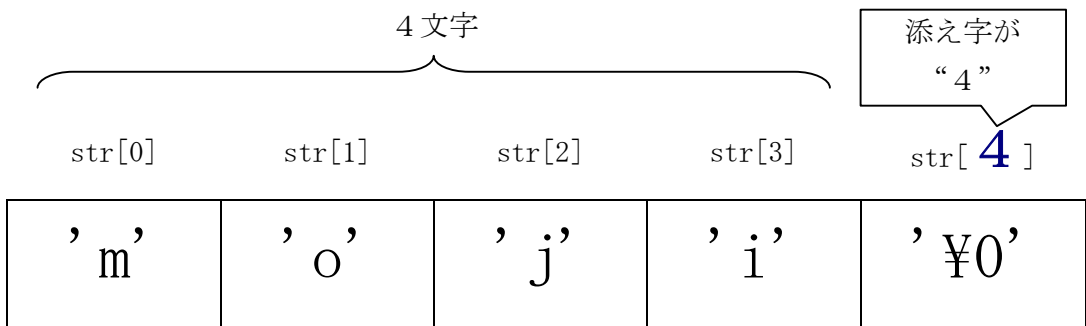
---

関数 `strlen()` は文字列の長さを得る関数ですが、同じ処理をする関数をつくってみましょう。

考え方としては、まず文字列の最後には必ずヌル文字「`¥0`」が付くというルールがありますので、

- 先頭から一文字ずつチェックする
- その文字が`¥0` だったら、その文字配列の添え字（`[]`の中の数字）が文字列の長さになる（下図参照）。

`char str[] = "moji";` の場合



としましょう。

また関数名は `mojinagasa()` とし、細かい仕様は以下のとおりとします。

<関数 mojinagasa() の仕様>

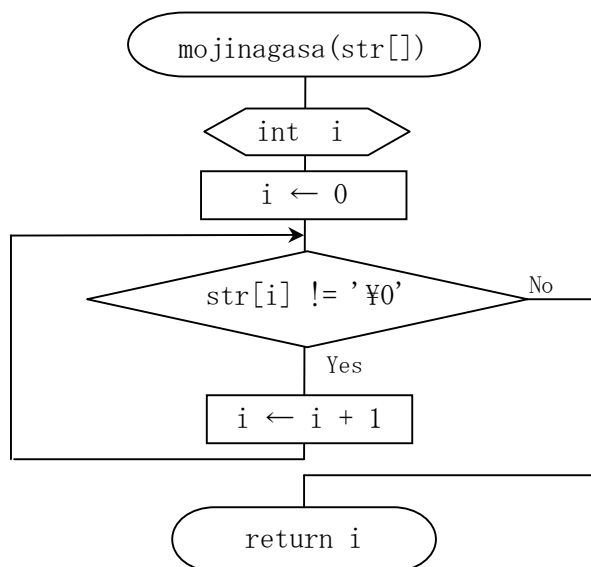
1. 外部向けの仕様

プロトタイプ	<code>int mojinagasa(char str[])</code>	
引数	<code>char str[]</code>	長さを求める文字列
返値	<code>int</code>	<code>str</code> の長さ
<b>【解説】</b> 文字列の長さ（¥0 は含まない）を求めます。		
<b>【例】</b> <pre>len = mojinagasa("mojiretsu");</pre> "mojiretsu" の長さを求め、len に代入します。		

2. 関数内で使用する変数

変数名	型	概要
i	int	カウンタ

### 3. フローチャート



### 4. ソースコード

```
int mojinagasa(char str[])  
{  
    int    i;  
  
    for(i = 0; str[i] != '¥0'; i++){  
    }  
  
    return i;  
}
```

<以上>

それでは mojinagasa() の動作確認プログラムを作りましょう。  
以下のプログラムを入力し、コンパイル・実行します。

mojinagasa.cpp

```
1: #include <stdio.h>
2:
3: #define SIZE    128
4:
5: int mojinagasa(char []);
6:
7: main()
8: {
9:     char    s[SIZE];
10:
11:     printf("文字列を入力して下さい\n");
12:     gets(s);
13:
14:     printf("入力した文字列は%d文字です。 \n", mojinagasa(s));
15: }
16:
17: int mojinagasa(char str[])
18: {
19:     int    i;
20:
21:     for(i = 0; str[i] != '\0'; i++){
22:     }
23:
24:     return i;
25: }
```

実行例

文字列を入力して下さい

moji

入力した文字列は 4 文字です。

どうやら問題なさそうですね。

### 3. 演習問題

11-1. intsaidai.cpp を参考に以下の仕様を満たす関数 intsaishyo() を完成させ、その動作を確認するプログラム「ensyu11-1.cpp」を作りなさい。

プロトタイプ	int intsaishyo(int array[], int n)	
引数	int array[]	最小値を求める配列
	int n	array[] の大きさ
返値	int	array[] の最小値

**【解説】**  
配列の最小値を求める。

**【例】**

```
min = intsaishyo(a, 100);
```

配列 a[] の最小値を求め、min に代入します。

11-2. goukei.cpp を参考に以下の仕様を満たす関数 heikin() を完成させ、その動作を確認するプログラム「ensyu11-2.cpp」を作りなさい。

プロトタイプ	<code>double heikin(double array[], int n)</code>	
引数	<code>double array[]</code>	平均を求める配列
	<code>int n</code>	array[]の大きさ
返値	<code>double</code>	array[]の平均
<b>【解説】</b> 配列の平均を求める。		
<b>【例】</b> <pre>average = heikin(a, SIZE);</pre>		
配列 a[] の平均を求め、average に代入します。		