

情報処理演習 (7)関数その2 配列を引数に取る関数

知能システム学 准教授
万 偉偉(ワン ウェイウェイ)

良いプログラムの第一要素

習慣に従って、他人に理解しやすい

注意事項

- 変数の定義
 - 大文字と小文字の混雑の使用
 - int m, Mの混雑の使用の汎用性が悪い, 特にWindowsシステム用のコンパイラは認識できない
 - 習慣的に、#defineされたマクロは大文字
 - 小文字と_を活用すること
 - forループの完備化と簡明化
 - for(i=1, j=2; i<5, j<10; i++, j++)などの使い方は理解し難い, 習慣的にfor(i=1; i<5; i++) と if(j>=10) break; の形で実装されること
 - int q=0; for(; q<10; q++)よりfor(q=0; q<10; q++)

配列を受け取る関数であることを宣言する

配列を引数に渡す

```
int minimum(int data[], int num);
```

```
int main(void) {  
    int seiseki[10] = {70, 85, ... };  
    int min;  
  
    min = minimum(seiseki, 10);  
    printf("minimum is %d\n", min);  
}
```

呼び出し側では、配列の名前のみ指定すればよい

配列の大きさは別の変数で関数に伝える

```
int minimum(int data[], int num) {  
    ....  
}
```

受け取り側では、配列の要素数を指定する必要はない

配列の中身の変更

```
int change(int data);
```

```
int main(void) {  
    int d = 70;  
  
    change(d);  
    printf("d is %d\n", d);  
}
```

```
int change(int data) {  
    data = 0;  
}
```

dの値は変わらない

```
int change(int data[], int num);
```

```
int main(void) {  
    int d [10] = {70, 85, ... };  
  
    change(d, 10);  
    printf("d[0] is %d\n", d[0]);  
}
```

```
int change(int data[], int num) {  
    data[0] = 0;  
}
```

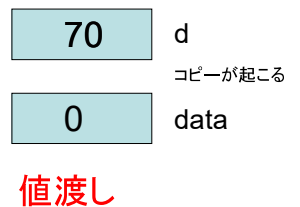
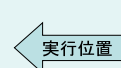
d[0]値が変わる

引数を関数に渡す仕組み(1)

```
int change(int data);
```

```
int main(void) {  
    int d = 70;  
  
    change(d);  
    printf("d is %d\n", d);  
}
```

```
int change(int data) {  
    data = 0;  
}
```



引数を関数に渡す仕組み(2)

```
int change(int data[], int num);
```

データが格納される領域は共通

```
int main(void) {  
    int d [7] = {70, 85, ... };  
  
    change(d, 7);  
    printf("d[0] is %d\n", d[0]);  
}
```

```
int change(int data[], int num) {  
    data[0] = 0;  
}
```

d	0	0
	1	85
	2	...
data	3	
	4	
	5	
	6	

参照渡し

const 修飾子

- const は値の変更を禁じる
- 変数を const にした場合: 値の変更が不可能
`const int a = 10;`
`a = 20;` ← エラー
- 引数を const にした場合: 値が変更されない
`int func(const int data[], int n);`
とすると, func 内で data に値を代入できない
`int func(const int data[], int n) {`
 `data[0] = 10;` ← エラー
`}`

二次元配列の引数渡し

- 二次元配列の場合は、引数の個数を指定する必要がある
 - `int func(int data[]);` ← エラー
 - `int func(int data[2][3]);` ← OK
 - `int func(int data[][3]);` ← OK
- 二次元配列の場合は、**少なくとも二番目の**引数の個数を指定すべき
- 指定しないと、一つの単元のサイズを決められない

まとめ

- 値渡し
- 参照渡し
- `const`修飾子
- 二次元配列の場合

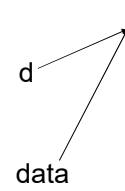
二次元配列

```
int change(int data[][3]);
```

```
int main(void) {  
    int d [2][3] = {{1, 2, 3},  
                    {4, 5, 6}};
```

```
    change (d);  
    printf("d[1][2] is %d\n",  
           d[1][2]);  
}
```

```
int change(int data[][3]) {  
    data[1][2] = 0;  
}
```



<code>[0][0]</code>	1
<code>[0][1]</code>	2
<code>[0][2]</code>	3
<code>[1][0]</code>	4
<code>[1][1]</code>	5
<code>[1][2]</code>	0

`data[a][b]` は、 $a * 3 + b$ 番の要素
(この計算を実行時に行うために3が要る)