配列とメモリ

プログラマのためのC言語 第13回

概要

- ✔ C言語の配列
- ✔ 配列の定義方法
- ✔ 配列とメモリの関係
- ✔ 配列名は配列データの先頭アドレスを表す
- ✔ 配列の要素へアクセスするということ

C言語の配列

C言語の配列

- ✔ 一言に配列と言っても**固定長配列**と**可変長配列**がある

C言語の配列

- ✔ 一言に配列と言っても**固定長配列**と**可変長配列**がある
- ✔ 固定長配列
  - ・要素数が決まっている、変更不可
- ✔ 可変長配列
  - ・要素数は決まっていない、変更可

C言語の配列

- ✔ 一言に配列と言っても**固定長配列**と**可変長配列**がある
- ✔ 固定長配列
  - ・要素数が決まっている、変更不可
- ✔ 可変長配列
  - ・要素数は決まっていない、変更可
- ✔ C言語の配列は一般的に固定長配列

配列の定義方法

配列とメモリの関係

配列とメモリ

✔ 配列はメモリ上で見ると地続きになっている

値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
番地	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

配列とメモリ

✔ 配列はメモリ上で見ると地続きになっている

値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
番地	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

```
char a[] = { 1, 2, 3, 4, 5 }; // 1byte × 5 = 5byte
```

配列のサイズ = **型のサイズ × 要素数**

配列とメモリ

✔ 配列はメモリ上で見ると地続きになっている

値	0	01	02	03	04	05	0	0	0	0	0	0	0	0	0	0	0	0
番地	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

a

```
char a[] = { 1, 2, 3, 4, 5 }; // 1byte × 5 = 5byte
```

配列とメモリ

✔ 配列はメモリ上で見ると地続きになっている

値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
番地	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

```
short a[] = { 1, 2, 3, 4, 5 }; // 2byte × 5 = 10byte
```

配列とメモリ

✔ 配列はメモリ上で見ると地続きになっている

値	0	01	00	02	00	03	00	04	00	05	00	0	0	0	0	0	0	0
番地	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

a

```
short a[] = { 1, 2, 3, 4, 5 }; // 2byte × 5 = 10byte
```

配列とメモリ

✔ 配列はメモリ上で見ると地続きになっている

値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
番地	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

```
int a[] = { 1, 2, 3, 4, 5 }; // 4byte x 5 = 20byte
```

配列とメモリ

✔ 配列はメモリ上で見ると地続きになっている

値	0	01	00	00	00	02	00	00	00	03	00	00	00	04	00	00	00	05
番地	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

a

入りきらない…

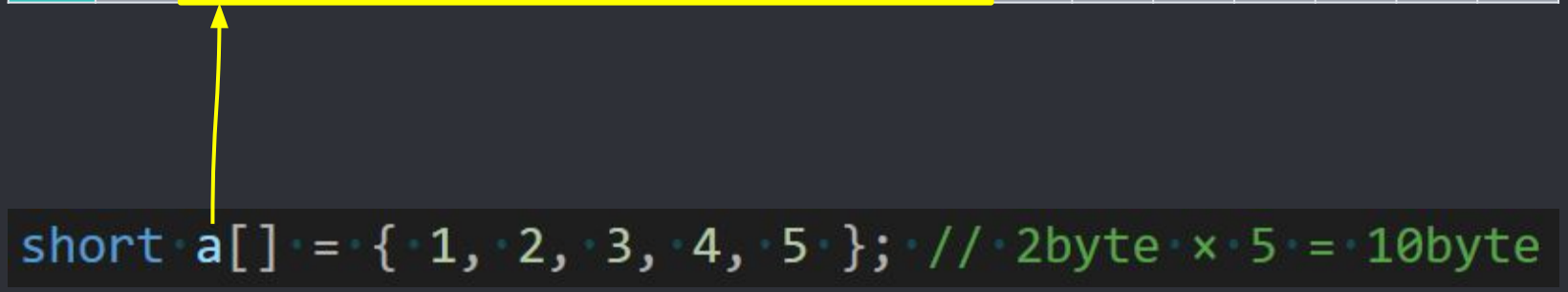
```
int a[] = { 1, 2, 3, 4, 5 }; // 4byte x 5 = 20byte
```

配列名は配列データの先頭アドレスを表す

配列名は配列データの先頭アドレスを表す

✔ 配列名 **a** は 配列データの先頭アドレスを表す

値	0	01	00	02	00	03	00	04	00	05	00	0	0	0	0	0	0	0
番地	10	**11**	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27



配列の要素にアクセスするということ

配列の要素にアクセスする

✔ 配列の要素の位置 = 先頭アドレス + (要素のサイズ × 添字)

```
char a[] = { 1, 2, 3, 4, 5 }; // 1byte × 5 = 5byte
```

値	0	01	02	03	04	05	0	0	0	0	0	0	0	0	0	0	0	0
番地	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

a[0] 👉 11 + (1 × 0) = 11

配列の要素にアクセスする

✔ 配列の要素の位置 = 先頭アドレス + (要素のサイズ × 添字)

```
char a[] = { 1, 2, 3, 4, 5 }; // 1byte × 5 = 5byte
```

値	0	01	02	03	04	05	0	0	0	0	0	0	0	0	0	0	0	0
番地	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

a[3] 👉 11 + (1 × 3) = 14

配列の要素にアクセスする

✔ 配列の要素の位置 = 先頭アドレス + (要素のサイズ × 添字)

```
char a[] = { 1, 2, 3, 4, 5 }; // 1byte × 5 = 5byte
```

値	0	01	02	03	04	05	0	0	0	0	0	0	0	0	0	0	0	0
番地	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

a[10] 👉 11 + (1 × 10) = 21

範囲外アクセスはあかん！

配列の要素にアクセスする

✔ 配列の要素の位置 = 先頭アドレス + (要素のサイズ × 添字)

```
int a[] = { 1, 2, 3, 4, 5 }; // 4byte x 5 = 20byte
```

値	0	01	00	00	00	02	00	00	00	03	00	00	00	04	00	00	00	05
番地	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

a[2] 👉 11 + (4 × 2) = 19

配列の型が変わっても考え方は同じ

配列の要素にアクセスする

✔ 配列名 a は配列の先頭アドレスを表す、つまり

```
int a[] = { 1, 2, 3, 4, 5 }; // 4byte x 5 = 20byte
```

値	0	01	00	00	00	02	00	00	00	03	00	00	00	04	00	00	00	05
番地	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

a

配列の要素にアクセスする

✔ 配列名 a は配列の先頭アドレスを表す、つまり、間接参照したら先頭要素を表す

```
int a[] = { 1, 2, 3, 4, 5 }; // 4byte x 5 = 20byte
```

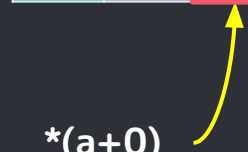
値	0	01	00	00	00	02	00	00	00	03	00	00	00	04	00	00	00	05
番地	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

配列の要素にアクセスする

✔ アドレスに足し算する書き方もできる

```
int a[] = { 1, 2, 3, 4, 5 }; // 4byte x 5 = 20byte
```

値	0	01	00	00	00	02	00	00	00	03	00	00	00	04	00	00	00	05
番地	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27



配列の要素にアクセスする

✔ アドレスに足し算する書き方もできる

```
int a[] = { 1, 2, 3, 4, 5 }; // 4byte x 5 = 20byte
```

値	0	01	00	00	00	02	00	00	00	03	00	00	00	04	00	00	00	05
番地	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

配列の要素にアクセスする

✔ アドレスに足し算する書き方もできる

```
int a[] = { 1, 2, 3, 4, 5 }; // 4byte x 5 = 20byte
```

値	0	01	00	00	00	02	00	00	00	03	00	00	00	04	00	00	00	05
番地	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27

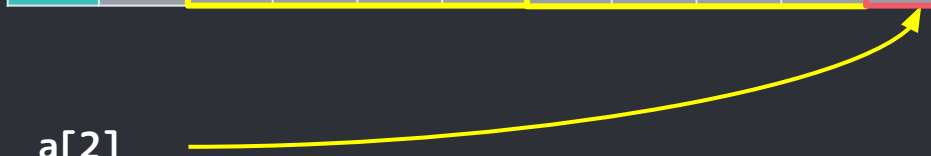
*(a+2)

配列の要素にアクセスする

✔ a[2] と *(a+2) は同等

```
int a[] = { 1, 2, 3, 4, 5 }; // 4byte x 5 = 20byte
```

値	0	01	00	00	00	02	00	00	00	03	00	00	00	04	00	00	00	05
番地	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27



配列の要素にアクセスする

✔ さらに a[2] と 2[a] も同等

```
int a[] = { 1, 2, 3, 4, 5 }; // 4byte x 5 = 20byte
```

値	0	01	00	00	00	02	00	00	00	03	00	00	00	04	00	00	00	05
番地	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27