プログラミング及び実習2

第 5 回 配列

佐野 彰

本日の内容

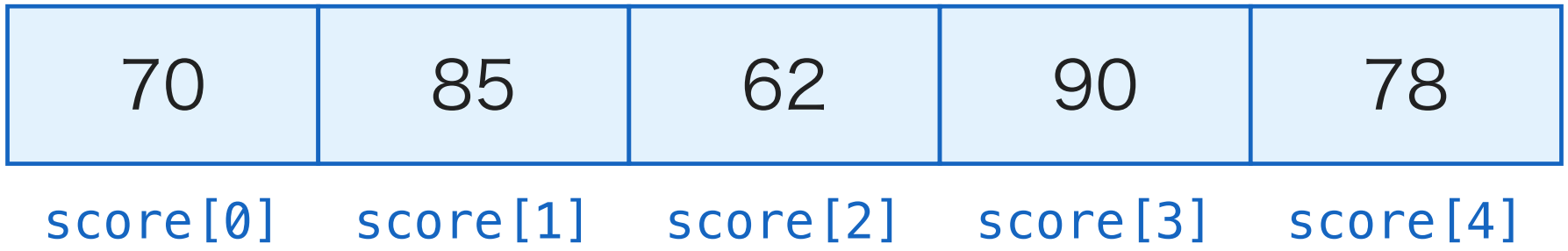
1. 配列変数の基本（宣言・添字・参照・代入・初期化）
2. 配列と繰り返し処理（利用例）
3. 配列のよくある間違い

1. 配列変数の基本

配列変数とは

5 人分の点数を扱うとき、変数を 5 個宣言する代わりに、**番号付きの変数をまとめて用意**できます。

```
int score[5];   // score[0], score[1], ..., score[4] の 5 個の int 型変数を用意する
```



- `score[0]` や `score[3]` が、それぞれ 1 つの `int` 型変数です。1 つ 1 つを**要素**と呼びます
- `[ ]` の中の何番目かを表す 0 以上の整数を**添字**（そえじ）と呼びます

配列変数の宣言

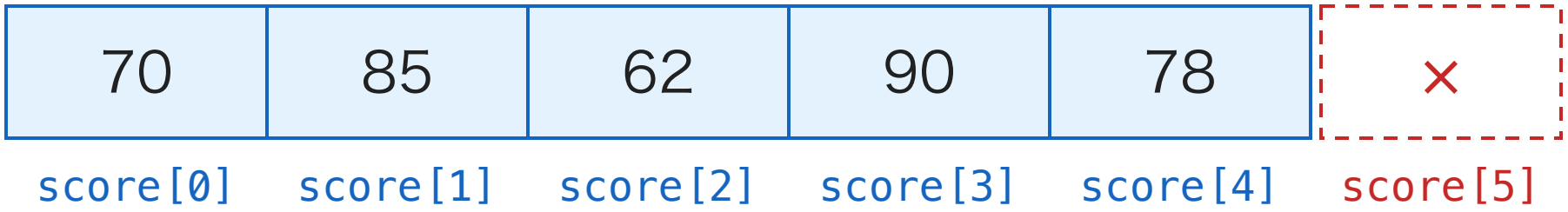
```
型 配列名[要素数];
```

```
int    score[5];    // int 型の要素が 5 個
double temp[7];     // double 型の要素が 7 個（1 週間分の気温など）
char   name[20];    // char 型の要素が 20 個
```

- 1 つの配列の要素は、すべて**同じ型**です（`int` と `double` を混ぜることはできません）
- 要素数は 1 以上の整数です
- 宣言した時点の要素の値は決まっていません（後の「初期化」で値を入れます）

添字は 0 から始まる

`int score[5];` で使える要素は `score[0]` から `score[4]` までです。**`score[5]` はありません。**



要素数 n の配列で使える添字: 0, 1, 2, ..., n−1 （全部で n 個）

- 1 番目の要素が `score[0]`、5 番目（最後）の要素が `score[4]` です
- 「何番目か」から 1 を引いた値が添字になる、と覚えてください

配列変数の何がうれしいのか

5 人の点数を読み込んで合計する処理を比べます。

配列を使わない場合

```
int s0, s1, s2, s3, s4;
scanf("%d", &s0);
scanf("%d", &s1);
scanf("%d", &s2);
scanf("%d", &s3);
scanf("%d", &s4);
sum = s0 + s1 + s2 + s3 + s4;
```

配列を使う場合

```
int s[5];
for (i = 0; i < 5; i++) {
    scanf("%d", &s[i]);
}
sum = 0;
for (i = 0; i < 5; i++) {
    sum += s[i];
}
```

添字を変数 `i` で表せるので、繰り返し処理で全要素を順に扱えます。1000 人になっても、`5` を `1000` に変えるだけです。

配列変数の参照・代入

添字が付くこと以外は、通常の変数と同じように使えます。

```c
int a[3];
a[0] = 5;               // a[0] に 5 を代入する
a[1] = 7;               // a[1] に 7 を代入する
a[2] = a[0] + a[1];     // a[0] と a[1] の値を参照して足し、a[2] に代入する（12）
a[1]++;                 // a[1] を 1 増やす（8）
printf("%d %d %d\n", a[0], a[1], a[2]);   // 5 8 12 と出力される
```

- `=` の左にあれば代入、式の中にあれば値の参照です。ふつうの変数と同じです
- `scanf()` で読み込むときも、ふつうの変数と同じく `&` を付けます: `scanf("%d", &a[0]);`

添字に変数や式を使う

添字には、整数の変数や式も書けます。添字の値を計算してから、その要素を使います。

```
double x[10];
int i = 0;
x[i] = 0.1;                 // i は 0 なので x[0] = 0.1
i++;                        // i は 1 になる
x[i] = 0.2;                 // x[1] = 0.2
x[i*3] = x[i] * x[0];       // i*3 は 3 なので x[3] = x[1] * x[0] (0.02)
x[i+1] = x[i-1] + 1.0;      // x[2] = x[0] + 1.0 (1.1)
```

添字の式の値が 0 から（要素数 − 1）の範囲に入るかどうかは、プログラムを書く人が確かめます。C 言語は自動では確かめません。

配列変数の初期化

宣言と同時に、`{ }` の中に値を並べて初期値を与えられます。

```
int a[5] = {10, 20, 30, 40, 50};   // a[0] = 10, a[1] = 20, ..., a[4] = 50
```

書き方	結果
`int a[5] = {1, 2};`	足りない分は 0 になる: `1 2 0 0 0`
`int z[5] = {0};`	**すべての要素を 0 にする**定番の書き方: `0 0 0 0 0`
`int c[] = {3, 1, 4, 1, 5};`	要素数を省略すると、値の個数（5）が要素数になる
`int d[3] = {1, 2, 3, 4};`	多すぎる。警告が出る（誤り）

`int d[3] = {1, 2, 3, 4};` をコンパイルしたときの gcc の警告:

```
warning: excess elements in array initializer
```

初期化しない配列の中身

`main()` の中で宣言しただけの配列の要素には、**何が入っているか分かりません。**

```
int sum[3];
sum[0] += 5;       // ✗ sum[0] の元の値が不明なので、結果も不明
```

使う前に、初期化するか、代入しておきます。

```
int sum[3] = {0};           // ○ 宣言と同時に 0 で初期化する

int cnt[3];
for (i = 0; i < 3; i++) {   // ○ 繰り返し処理で 0 を代入する
    cnt[i] = 0;
}
```

要素数に変数を使う（C99 規格）

要素数に変数を書くと、実行時に配列の大きさを決められます。

```c
int n;
scanf("%d", &n);
if (n <= 0) {
    return 0;      // 要素数として使えない値なら、何もせずに終了する
}
int a[n];          // 入力された n 個の要素を持つ配列
```

- `int a[n];` の前で、`n` が 1 以上であることを確かめます
- 要素数が変数の配列は、`{ }` で初期化**できません**。繰り返し処理で値を代入します
  - 例: `int a[n] = {0};` は gcc で `error: variable-sized object may not be initialized ...`

2. 配列と繰り返し処理

全要素を順に扱う定型

要素数を `#define` で名前にしておき、`for` 文で添字を 0 から順に動かします。

```
#define N 5

int score[N];
int i;
for (i = 0; i < N; i++) {      // i は 0, 1, 2, 3, 4 と変わる
    // score[i] を使う処理
}
```

- 条件は **`i < N`** です。`i <= N` と書くと、存在しない `score[N]` まで使ってしまいます
- 要素数を変えるときは、`#define N` の 1 か所を直すだけで済みます

例 1: 点数を読み込み、合計と平均を求める

```c
#include <stdio.h>
#define N 5

int main() {
    int score[N];
    int i, sum = 0;
    for (i = 0; i < N; i++) {
        scanf("%d", &score[i]);
    }
    for (i = 0; i < N; i++) {
        sum += score[i];
    }
    printf("合計 %d, 平均 %.1f\n",
           sum, (double)sum / N);
    return 0;
}
```

実行例

```
$ ./avg
70 85 62 90 78
合計 385, 平均 77.0
```

- 1 つ目の `for` で、入力を `score[0]` から順に読み込みます
- 2 つ目の `for` で、全要素を `sum` に足します
- 平均は `(double)` で実数にしてから割ります（整数どうしの割り算は小数部が切り捨て）

例 2: 平均との差を出す（配列が必要になる例）

平均は全員分を読み込むまで分かりません。**入力を配列に覚えておけば**、平均を求めた後で 1 人ずつ比べられます。

```c
int score[N];
int i, sum = 0;
double avg;
for (i = 0; i < N; i++) {
    scanf("%d", &score[i]);
    sum += score[i];
}
avg = (double)sum / N;
printf("平均 %.1f\n", avg);
for (i = 0; i < N; i++) {
    printf("%d 番: %d 点（平均との差 %+.1f）\n",
           i, score[i], score[i] - avg);
}
```

実行例

```
$ ./avg2
70 85 62 90 78
平均 77.0
0 番: 70 点（平均との差 -7.0）
1 番: 85 点（平均との差 +8.0）
2 番: 62 点（平均との差 -15.0）
3 番: 90 点（平均との差 +13.0）
4 番: 78 点（平均との差 +1.0）
```

`%+.1f` は、正の値にも `+` を付けて表示します。

例 3: 対応表として使う（月の日数）

添字を「番号から値を引く表」として使えます。1 月の日数を `days[0]` に入れるので、`m` 月の日数は `days[m - 1]` です。

```c
int days[12] = {31, 28, 31, 30, 31, 30,
                31, 31, 30, 31, 30, 31};
int m;
scanf("%d", &m);
if (m < 1 || m > 12) {
    printf("1 から 12 を入力してください\n");
    return 0;
}
printf("%d 月は %d 日まであります\n",
       m, days[m - 1]);
```

実行例

```
$ ./days
2
2 月は 28 日まであります
$ ./days
13
1 から 12 を入力してください
```

- `if` で範囲を確かめてから `days[m - 1]` を使います
- `if` がないと、13 を入力したときに存在しない `days[12]` を使ってしまいます

例 4: 隣の要素と比べる（前日との気温差）

1 週間分の気温を読み込み、2 日目以降について前日との差を出します。
`t[i]` の日の前日の気温は、`t[i - 1]` に入っています。

```c
#define N 7

double t[N];
int i;
for (i = 0; i < N; i++) {
    scanf("%lf", &t[i]);
}
for (i = 1; i < N; i++) {   // 1 から始める
    printf("%d 日目: %+.1f\n",
           i + 1, t[i] - t[i - 1]);
}
```

実行例

```
$ ./diff
18.2 19.5 17.0 17.0 20.3 21.1 19.8
2 日目: +1.3
3 日目: -2.5
4 日目: +0.0
5 日目: +3.3
6 日目: +0.8
7 日目: -1.3
```

2 つ目の `for` を `i = 0` から始めると、
存在しない `t[-1]` を使ってしまいます。

3. 配列のよくある間違い

ダメな例: 範囲外の添字

```c
int a[3] = {10, 20, 30};
for (i = 0; i <= 3; i++) {    // ✗ i が 3 のとき、存在しない a[3] を使う
    printf("a[%d] = %d\n", i, a[i]);
}
```

gcc での実行例

```
a[0] = 10
a[1] = 20
a[2] = 30
a[3] = 3
```

- 多くの場合、**エラーにならず**に動きます
- `a[3]` は配列の外にたまたまあった値で、環境によって変わります
- 範囲外に代入すると、ほかの変数を壊したり、異常終了したりします

◯ **正しくは** `i < 3` です。条件の `<` と `<=` に気を付けてください。

ダメな例: 配列をまとめて代入する

```
int a[5] = {1, 2, 3, 4, 5};
int b[5];
b = a;          // ✗ 配列どうしは = で代入できない
```

gcc のエラー: `error: assignment to expression with array type`

○ **正しくは**、繰り返し処理で 1 要素ずつ代入します。代入しながら値を変えることもできます。

```
for (i = 0; i < 5; i++) {
    b[i] = a[i];          // そのまま写す
}
for (i = 0; i < 5; i++) {
    b[i] = a[i] * 2;      // 2 倍して写す → b は 2 4 6 8 10
}
```

ダメな例: 配列をまとめて入出力する

```
int a[3];
scanf("%d", a[i]);      // ✗ & がない
printf("%d\n", a);      // ✗ 配列全体は %d 1 つでは出力できない
```

gcc の警告（`-Wall` 付きでコンパイルした場合）:

```
warning: format '%d' expects argument of type 'int *', but argument 2 has type 'int'
warning: format '%d' expects argument of type 'int', but argument 2 has type 'int *'
```

○ **正しくは**、1 要素ずつ `&a[i]` で読み込み、1 要素ずつ `a[i]` を出力します。

```
for (i = 0; i < 3; i++) {
    scanf("%d", &a[i]);
}
for (i = 0; i < 3; i++) {
    printf("%d\n", a[i]);
}
```

まとめ

- `型 配列名[要素数];` で、同じ型の変数を番号付きでまとめて用意できる
- 要素数 n の配列の添字は **0 から n-1** まで。`a[n]` は存在しない
- 添字には変数や式を書ける。`for (i = 0; i < N; i++)` で全要素を順に扱うのが定型
- 初期化は `{ }` で行う。すべて 0 にするなら `= {0}`。初期化しない要素の値は不明
- 配列どうしの代入や、配列全体の入出力はできない。1 要素ずつ繰り返し処理で扱う
- 入力を配列に覚えておくと、後から何度でも使える（平均との差、前日との差など）

次回予告と今日やること

次回（10/29）：二重ループ、配列の応用（統計量・ソート・数え上げ）

今日の 5 講時（実習）

1. 第 4 回課題の解答例を演習ページで確認する（実数の整数部・小数部、マクローリン展開など）
2. 配列変数の宣言・初期化を確認する
3. 演習課題 ex05 に取り組み、100 点以上を取る

質問は **Teams「★ 質問用チャネル」** か、教室の TA へ。