

シラバス

15,16. 関数と配列

配列を引数に持つ関数による同型多数データを扱う処理のモジュール化方法の理解

事前学修：授業スライド「第8回 関数と配列」と教科書p.118～p.128の熟読。（60分）

事後学修：授業スライドの用語とその意味の理解。C言語における配列の1つの要素を関数に渡す方法，配列全体を関数に渡す方法，関数から配列の1つの要素を返す方法，配列全体を返す方法を説明できること。（60分）

演習課題のプログラムの完成と提出。（180分）

プログラミングの基礎及び演習

日本大学 工学部 情報工学科

プログラミングの基礎 第8回

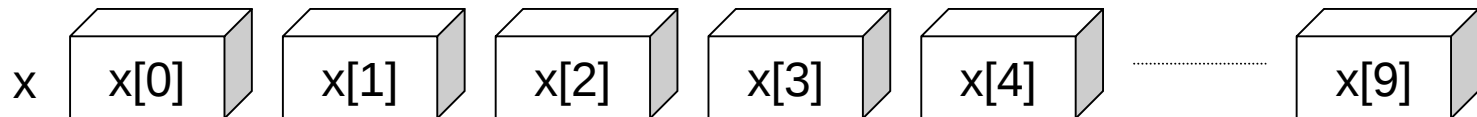
■ 配列 (2)

- 関数に配列を渡すには？
- 関数から配列を受け取るには？

- 教科書： p.118～p.128

配列とは

- 番号付きで並べられた変数
 - それぞれの要素には同じ型の値が格納される
 - 同じ種類のデータをまとめて扱うのに便利
- 配列全体を表すための名前（配列名）を設定



- 個々の要素は番号（添字）を使って参照
 - 配列名[添字] で表わす
 - 個々の要素は変数と同様に扱える
 $x[5] = 80$; など
 - C言語では添字は0から始まる

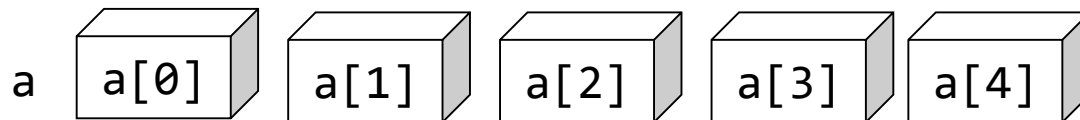
重要： 配列の添字

■ 配列の添字は0から始まる

□ 1 ではないことに注意！

□ 例： `int a[5];` という宣言をした場合：

- `a[0]`, `a[1]`, `a[2]`, `a[3]`, `a[4]` の5つが使える
- `a[1]`, `a[2]`, `a[3]`, `a[4]`, `a[5]` ではない！



「添字」は、
「要素番号」「インデックス」と呼ばれることもある

配列

- いくつかの点数score をプログラムで扱うのに, score0, score1, score2とすると, たくさんの変数が必要になって不便
- score という配列にすれば,
 - score[0], score[1], score[2]だけでなく, 変数を使って
 - score[i] として, iの値を0, 1, 2と変えて使うことができ便利



例題9.1: 配列の値の合計の計算

```
#include <stdio.h>
```

```
int main(void) {
```

```
    int i;
```

```
    int score[4];
```

} 要素数 4 の配列scoreの宣言

```
    int total = 0;
```

} 合計を格納する変数totalの宣言と初期化

```
    score[0] = 3;
```

```
    score[1] = 2;
```

```
    score[2] = 4;
```

```
    score[3] = 5;
```

} scoreの各要素への値の代入.
要素数 4 なので, [0], [1], [2], [3]である.
score[4]は使えない.

```
    for( i = 0; i <= 3; i++ )
```

```
        total += score[i];
```

```
    printf("Totalscore: %d\n", total);
```

```
    return 0;
```

```
}
```

合計の計算

$i \leq 4$ ではない

$i < 4$ ならOK

プログラミングの基礎 第8回

■ 配列 (2)

- 関数に配列を渡すには？

- 関数から配列を受け取るには？

- 教科書： p.118～p.128

関数に配列を渡す

- 「関数に配列を渡す」には2つの意味がある
 - 配列の要素の1つを渡す
 - 配列全体を渡す

関数main

```
int array[10];
```

配列の要素の1つarray[3]を
関数func1に渡す

配列array全体を
関数func2に渡す

関数func1

配列の1つの要素array[3]のみを受け取る

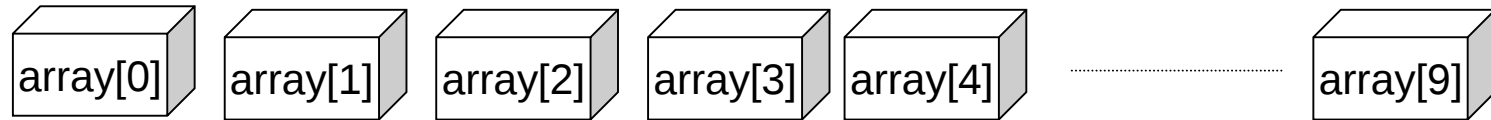
他の要素は受け取っていないため、
関数func1内ではarray[3]の値だけを使える

関数func2

配列array全体を受け取る

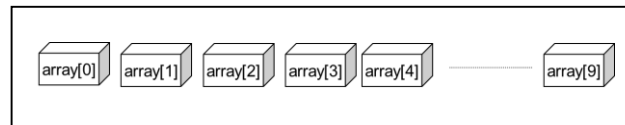
関数func2内では、配列arrayの
10個の要素全てを使える

関数に配列を渡す：両者の違い



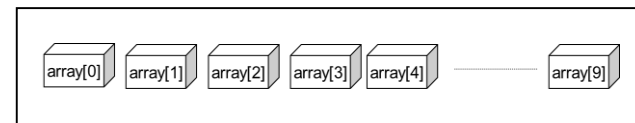
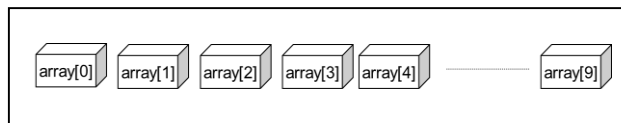
■ 配列の1つの要素を渡す・受け取る

- 受け取った関数で使えるのは、array[3]の値のみ.



■ 配列全体を渡す・受け取る

- 受け取った関数では、array[0]からarray[9]まで10個の要素を全て使える





関数に配列の要素の1つを渡す

■ 通常の変数と同じ方法で可能

実行結果：

添字[0]の要素は3です
添字[1]の要素は5です
添字[2]の要素は8です

```
int main(void){  
    int i;  
    int array[3] = { 3, 5, 8 };  
  
    for( i = 0; i < 3; i++ ) {  
        output( i, array[i] );  
    }  
    return 0;  
}
```

配列array[]のi番要素のみを関数に渡す
[i]は添字（要素の番号）である
他の要素は渡さない

受け取り側は、配列ではなく、
通常の変数として受け取る

```
void output( int index, int element ){  
    printf( "添字[%d] の要素は%dです¥n", index, element );  
}
```

関数に配列全体を渡す

例題9.2

```
#define N 3
int keisan( int a[] );

int main( void ){
    int i, a[N], sum;
    for( i = 0; i < N; i++ ) {
        printf( "a[%d]=", i );
        scanf( "%d", &a[i] );
    }
    sum = keisan( a );
    printf( "sum = %d\n", sum );
    return 0;
}
```

配列を渡す側の書き方

- int型の配列a[]の全ての要素を渡す
- 添字を書かないことに注意

配列を受け取る側の書き方

- int型の配列a[]の全ての要素を受け取る

```
int keisan( int a[] ) {
    int i, sum = 0;
    for( i = 0; i < N; i++ ) {
        sum += a[i];
    }
    return sum;
}
```

受け取った関数でも
配列として使用可能

プロトタイプ宣言

■ 配列全体を受け取る関数のプロトタイプ宣言

```
int keisan(int a[]);
```

- `int a[]` で配列であることを明示
- `int a[N]` のように、配列の要素数を記述しても `N`は無視される
- 要素の大きさ `N` を書いても書かなくても同じ
 - ・ 受け取った関数では配列の要素数は分からないことに注意

■ 配列の1つの要素を受け取る関数のプロトタイプ宣言

```
void output(int index, int element);
```

- `int element` のように、通常の変数と同じように書く
-



関数に渡された配列と変数の違い (1)

- 配列の1つの要素を受け取る関数
 - ・ ・ ・ 通常の変数として扱われる
 - 値のコピーが渡される
 - 関数で値を変更しても、元の配列の値は変わらない
 - ・ 局所変数だから
- 配列全体を受け取る関数
 - ・ ・ ・ 配列として扱われる
 - 配列のアドレスが渡される
 - ・ アドレス： メモリ上の番地
 - ・ 詳しいことは、2年前期の「データ構造入門」で
 - 関数で配列の値を変更すると、元の配列の値も変わる



関数に渡された配列と変数の違い(2)

■ 配列の1つの要素を受け取る関数

p.96 例題7.6改

```
int main(void) {  
    int c[4] = {2, 3, 4, 6};  
    printf( "[1] c[0] = %d, c[1] = %d\n", c[0], c[1] );  
    koukan( c[0], c[1] );  
    printf( "[2] c[0] = %d, c[1] = %d\n", c[0], c[1] );  
    return 0;  
}  
  
void koukan(int a, int b) {  
    int copy;  
    copy = a; a = b; b = copy;  
}
```

実行結果 :

```
[1] c[0]=2, c[1]=3  
[2] c[0]=2, c[1]=3
```

実行結果において

[2]のc[0],c[1]の値が[1]と同じことに注意



通常の変数として渡される場合、関数側で値を変更しても元の値は変わらない

関数に渡された配列と変数の違い(3)

■ 配列全体を受け取る関数

```
int main(void){
    int i, a[N];
    for( i = 0; i < N; i++ ) {
        printf( "a[%d] = ", i );
        scanf( "%d", &a[i] );
    }
    multi2( a );
    for( i = 0; i < N; i++ ) {
        printf( "a[%d] = %d\n", i, a[i] );
    }
    return 0;
}

void multi2(int b[]) {
    int i;
    for( i = 0; i < N; i++ ) {
        b[i] *= 2;
    }
}
```

実行結果 :

a[0] = 1	Return
a[1] = 3	Return
a[2] = 5	Return

a[0] = 2
a[1] = 6
a[2] = 10

元の値も2倍になっている



受け取った関数で、配列の値を2倍にする

プログラミングの基礎 第8回

■ 配列 (2)

- 関数に配列を渡すには？
- 関数から配列を受け取るには？
- 教科書： p.118～p.128

関数から配列を返す

■ 関数からの戻り値

□ 通常の変数の場合

- `return`文を使って呼び出し元に返す
- `return x;` と書けば、変数`x`の値が返される

□ 配列の1つの要素だけを返す場合

- 通常の変数と同じ
- `return a[3];` と書けば、配列`a`の`[3]`要素が返される

□ 配列全体を返す場合

- `return`文では返せない

例) 関数での計算結果を配列に格納して、配列全体を返したい

配列全体を返すには？

- 通常のreturn文では一つの値しか返せない.
では, どうするか?
 - 「配列全体を関数に渡す」ときの性質を利用する
 - 「配列全体を関数に渡した」場合,
受け取った関数で値を変更すると,
元の配列の値も変更される
- ⇒戻り値に相当する配列も「引数」として渡せば良い
- 例) 関数での計算結果を入れる配列を引数として
関数に渡しておく



例題：関数から配列を返す

```
int main(void) {  
    int i, a[N], b[N], c[N];  
  
    for (i = 0; i < N; i++) {  
        printf("a[%d] = ", i);  
        scanf("%d", &a[i]);  
    }  
    for (i = 0; i < N; i++) {  
        printf("b[%d] = ", i);  
        scanf("%d", &b[i]);  
    }  
  
    plus(a, b, c);  
  
    for(i = 0; i < N; i++) {  
        printf("c[%d] = %d¥n", i, c[i]);  
    }  
    return 0;  
}
```

配列aとbは加算のための引数
配列cは「値を返却するための引数」

```
void plus(int a[], int b[], int c[])  
{  
    int i;  
    for (i = 0; i < N; i++) {  
        c[i] = a[i] + b[i];  
    }  
}
```

return文ではなく、「引数」として
配列cを受け渡している

配列aと配列bの各要素を加算して、
結果を配列cに格納する
⇒ 関数plusから配列cを返す

const修飾子

- constant : 定数 の意味
- 変数の値の変更を禁止する修飾子
 - 定数として扱われる
 - プログラムの不注意な書き換えによる不具合を防ぐことが可能
- 使い方 : 変数宣言の前にconstを付けて宣言する
- 代入は不可能だが, 宣言時の初期化は可能
例 : `const int total = 10;`
`const int score[] = {10, 100, 20, 0};`



const修飾子

例: 関数plus(スライド15ページ)の仮引数に使用

```
int main(void) {  
    int i, a[N], b[N], c[N];  
    for (i = 0; i < N; i++) {  
        printf("a[%d] = ", i);  
        scanf("%d", &a[i]);  
    }  
    for (i = 0; i < N; i++) {  
        printf("b[%d] = ", i);  
        scanf("%d", &b[i]);  
    }  
}
```

```
    plus(a, b, c);  
    for(i = 0; i < N; i++) {  
        printf("c[%d] = %d\n", i, c[i]);  
    }  
    return 0;  
}
```

```
void plus(const int a[], const int b[], int c[]) {  
    int i;  
    for (i = 0; i < N; i++) {  
        c[i] = a[i] + b[i];  
    }  
}
```

(A)

関数内でa[], b[]の内容を変更できない。
(A)を間違えて,
a[i] = a[i] + b[i];
としてもコンパイル時にエラーが発生する
→ 不注意による不具合を防ぐことができる

const修飾子

- const修飾子を付けると読み込み専用で定数扱いになる
- 値を変更しないいつもの変数に代入する文などがあった場合、コンパイル時に発見できる（コンパイルエラーになるので）
 - const を付けた変数にキャストをつけるときは、const もつけること。そうでないと、読み込み専用にしたのが無効になってしまう。
 - (例) `z = (const double) x/y;`

配列を使ったプログラミング

- 同様のものが複数あるデータは配列を使うと便利ことが多い
 - 複数データの和, 平均, 統計など
 - ベクトル, 行列
 - 複数のロボットやセンサーのデータ

練習問題

- 2つのベクトルの差 $x - y$ を求める
プログラムを作成せよ. 2つのベクトルを
受け取り, 差を計算する関数
`subtract_vector`を作成して用いること
- ベクトルを配列で表す
- 講義資料の中の`plus`を参考にして考える

ヒント

- ベクトルを配列によって表現する
 - 3次元のベクトル $\Rightarrow$ 要素数 3 の配列
- ベクトルの差の計算
 - 各要素の差を計算すれば良い



解答

```
#include <stdio.h>
#define DIM 3
```

配列xとyを引数として渡す
配列zは、計算結果を返却するための引数

```
void subtract_vector(int x[], int y[], int z[]);
```

```
int main(void) {
    int i;
    int x[DIM] = {1, -2, 1};
    int y[DIM] = {2, 0, -2};
    int z[DIM];

    subtract_vector(x, y, z);
```

```
    for(i = 0; i < DIM; i++) {
        printf("z[%d]=%d\n", i, z[i]);
    }
    return 0;
}
```

```
void subtract_vector(int x[], int y[],
                    int z[]) {
    int i;
    for (i = 0; i < DIM; i++) {
        z[i] = x[i] - y[i];
    }
}
```

【演習課題】

- 演習課題提出システムの
「第8回」演習課題を実施
- 演習課題8-1, 8-2, 8-3, 8-4
 - 演習時間に，設計後に演習環境に
リモートログインしプログラムを作成
 - 演習環境でコンパイル，テストし，完成
 - ソースファイル（8-x.c）をWinSCP等でPCへ転送
 - 課題提出システムへSubmitし
各課題100点をを目指す
 - 時間内に終了しない場合 ⇒ 宿題
- 授業翌週水曜日の午後5時までに必ず提出
- レポート，今回はなし