32 PAGES

ガイダンス，
ポインタの復習

アルゴリズムとデータ構造B

第1回

本日の予定

ガイダンス

- シラバスを用いた授業説明
- 授業で学ぶデータ構造の紹介

ポインタの復習，ポインタと配列

- ポインタの利用法の整理
  - ✓ **表や線形リスト等のデータ構造でポインタを利用するため**

科目概要

プログラムを設計するために重要なものは、アルゴリズムとデータ構造である。本科目では、C言語の文法、探索や整列のアルゴリズムを踏まえて、基本的なデータ構造として**表**、**スタック**、**キュー**、**リスト**、**木構造**を学ぶ。そして、アルゴリズムやデータ構造の理解を深めるために、実際にC言語のプログラムを作成する。さらに、データ構造を用いた実用的なアルゴリズムとC＋＋による動的結果出力方法により、アルゴリズムやデータ構造への理解を深める。本科目は、講義と各自のノートパソコンを用いた演習を交互に実施、プログラミング能力を身につけるものである。

到達目標

(ア) C言語の文法とC言語によるプログラミングの基礎から上級までを理解し、
プログラム作成に利用できる

(イ) アルゴリズムとデータ構造がプログラミングの要であることを理解する。

(ウ) コンピュータ内部でデータを表現する方法（データ構造）には
バリエーションがあることを説明できる。

(エ) 同一の問題に対し、選択したデータ構造によってアルゴリズムが変化しうることを
説明できる。

(オ) リスト構造、スタックやキュー、木構造などの基本的なデータ構造の概念と操作を理解し、
プログラムで実装できる。

(カ) C++ による動的結果出力によって、アルゴリズムやデータ構造の理解を深める。

ルーブリック

	最低限の到達レベルの目安（優）	最低限の到達レベルの目安（良）	最低限の到達レベルの目安（不可）
評価項目（ア）	C言語の文法とC言語によるプログラミングの基礎から上級までを理解し、プログラム作成に利用できる。	C言語の文法とC言語によるプログラミングの基礎から上級までを理解する。	C言語の文法とC言語によるプログラミングの基礎から上級までを理解できない。
評価項目（イ）	配列や構造体、および、スタックやキュー、リストなどの基本的なデータ構造を理解し、プログラムで実現でき、さらに，様々なデータ管理に利用できる。	配列や構造体、および、スタックやキュー、リストなどの基本的なデータ構造を理解する。	配列や構造体、および、スタックやキュー、リストなどの基本的なデータ構造を理解てきない。
評価項目（ウ）	同一の問題に対し、選択したデータ構造によってアルゴリズムが変化しうることを理解し、問題を解く過程を説明できる。	同一の問題に対し、選択したデータ構造によってアルゴリズムが変化しうることを理解している。	同一の問題に対し、選択したデータ構造によってアルゴリズムが変化しうることを理解できない。

評価

- 中間試験：25%
  - ✓ データ構造を中心に
- 定期試験：50%
- 課題：25%
  - ✓ データ構造に関する課題
    - 課題点が入る演習問題は都度アナウンス
  - ✓ C++ を使用した動的結果出力に関する課題

授業計画

授業計画				
		週	授業内容	週ごとの到達目標
後期	3rdQ	1週	シラバスを用いた授業内容の説明、ポインタ・構造体復習	C言語の文法とC言語によるプログラミングの基礎を理解している。
		2週	配列の動的生成	配列を動的に生成する方法について理解している。
		3週	基本的なデータ構造（１）：表	基本的なデータ構造である表を理解している。
		4週	基本的なデータ構造（２）：スタック	基本的なデータ構造であるスタックを理解している。
		5週	基本的なデータ構造（３）：キュー	基本的なデータ構造であるキューを理解している。
		6週	基本的なデータ構造（４）：線形リスト	基本的なデータ構造である線形リストの概要を理解している。
		7週	基本的なデータ構造（４）：線形リスト（基本操作）	線形リストの基本操作を理解している。
		8週	基本的なデータ構造（４）：線形リスト（応用・発展操作）	線形リストの応用・発展操作を理解している。
	4thQ	9週	中間試験、基本的なデータ構造（５）：木構造	基本的なデータ構造を理解し、プログラムで実現でき、さらに，様々なデータ管理に利用できる。また、基本的なデータ構造である木構造を理解している。
		10週	基本的なデータ構造（５）：木構造（二分木）	木構造の一つである二分木を理解している。
		11週	その他のデータ構造、逆ポーランド記法、自己再編成探索	その他のデータ構造の概要、逆ポーランド記法、自己再編成探索について理解している。
		12週	データ構造・アルゴリズム可視化のための環境構築、配列構造の表現	データ構造・アルゴリズム可視化に用いる演習環境の構築ができる。また、配列の可視化を実現できる。
		13週	スタック・キューの表現	スタック・キューを図的に表現し、操作アルゴリズムによる動作を可視化できる。
		14週	スタック・キューの表現、線形リストの表現	スタック・キューを図的に表現し、操作アルゴリズムによる動作を可視化できる。
		15週	線形リストの表現	線形リストを図的に表現し、操作アルゴリズムによる動作を可視化できる。

ガイダンス：データ構造とは

アルゴリズムとデータ構造A

- アルゴリズム（探索，整列）を中心に学習

アルゴリズムとデータ構造B

- 基本的なデータ構造を学習
  - ✓ さらに発展的な内容を「オブジェクト指向プログラミング」で，実用的なデータ構造の理論を「離散数学」で学ぶ

データ構造とは

- データ単位とデータ自身とのあいだの物理的又は論理的な関係（JIS X0015 03.01）　教科書 p. 42
- コンピュータ言語が持つデータ型だけでは，大量のデータや複雑なデータを効率よく操作することはできない
- そこでデータ群を都合よく組織化するための抽象的なデータ型をデータ構造と呼ぶ　教科書 p. 214
  - ✓ データ型：short, int, long, float, double, char, ...
  - ✓ データ構造：配列，構造体，...

データ構造とアルゴリズムは密接な関係にあり、
よいデータ構造の選択がよいプログラムの作成に繋がる

アルゴリズムとデータ構造Bで学ぶデータ構造

表（テーブル）

- 二次元配列で実装，領域の動的割り当て

スタックとキュー

- 先入れ先出し，先入れ後出し
- 配列での実装，リストでの実装

線形リスト

- 自己参照構造体で実現

木構造

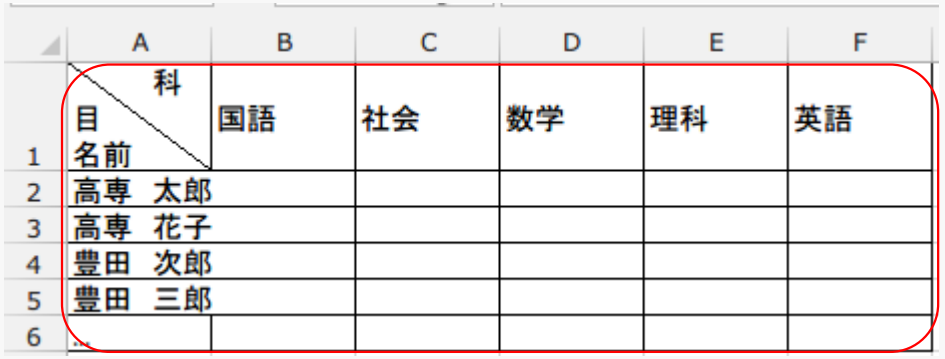
- データ構造を複数の方法で実現

グラフ

- 詳しくは離散数学で学習

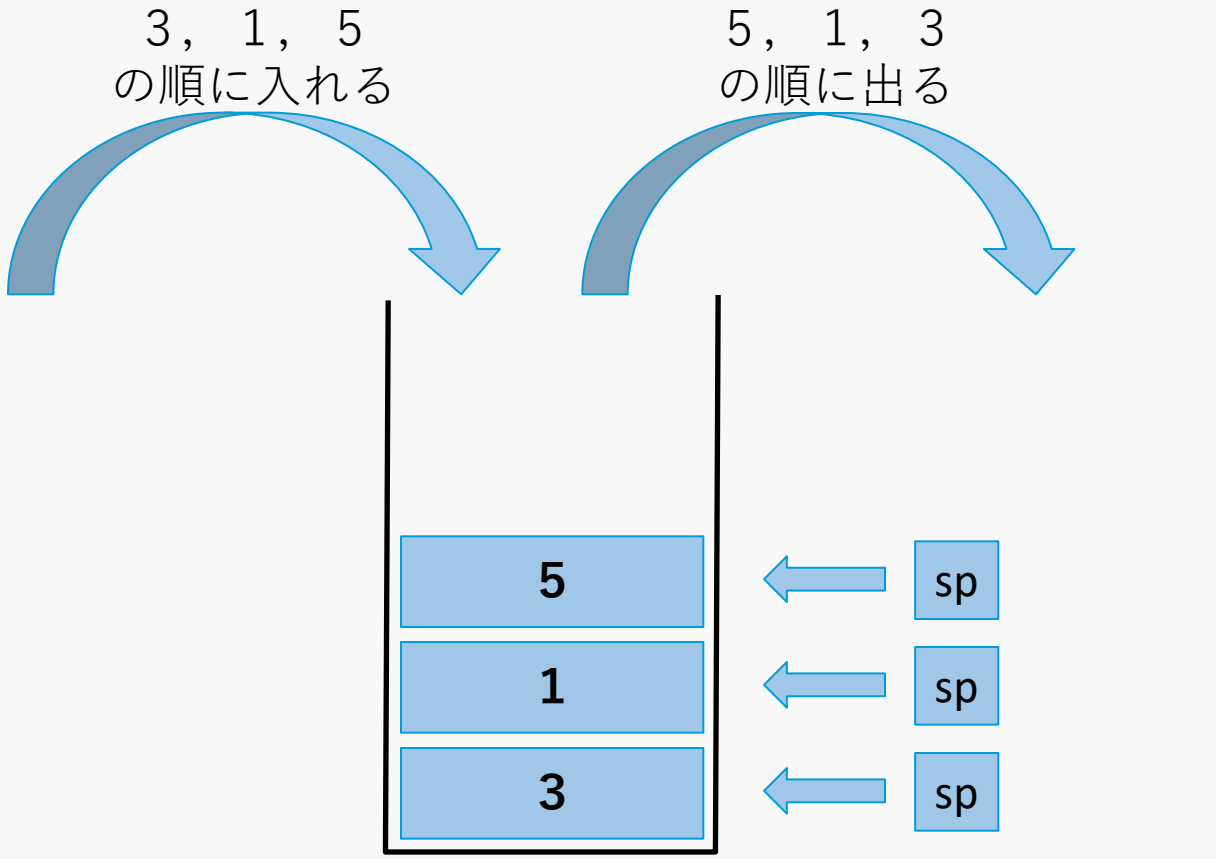
表（テーブル）の概念図

- 基本：2次元配列で実現（Excelのシート等のイメージ）
  - ✓ 例）成績表，行列



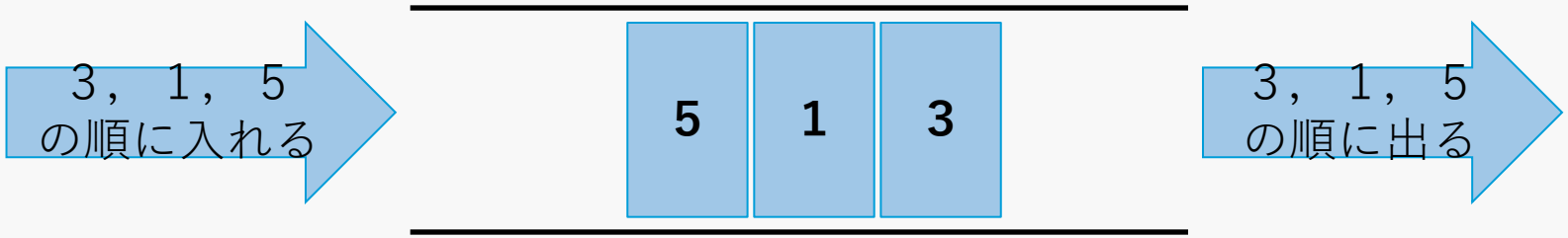
	A	B	C	D	E	F
1	科目 名前	国語	社会	数学	理科	英語
2	高専　太郎					
3	高専　花子					
4	豊田　次郎					
5	豊田　三郎					
6	…					

スタックの概念図



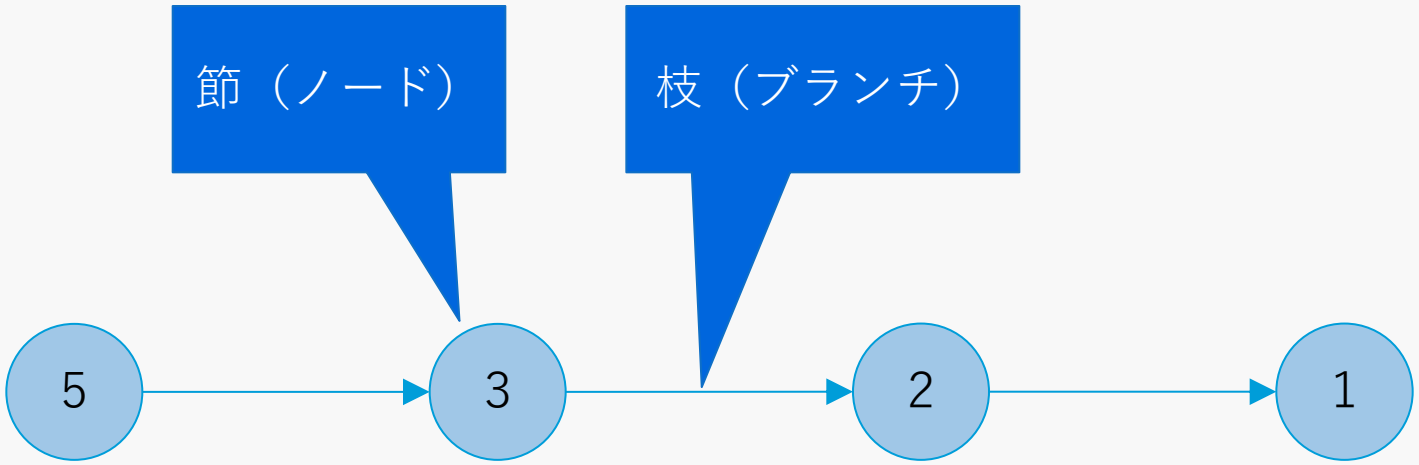
先入れ後出し方式
FILO（First In Last Out）

キューの概念図



先入れ先出し方式
FIFO（First In First Out）

リストの概念図（数学的）

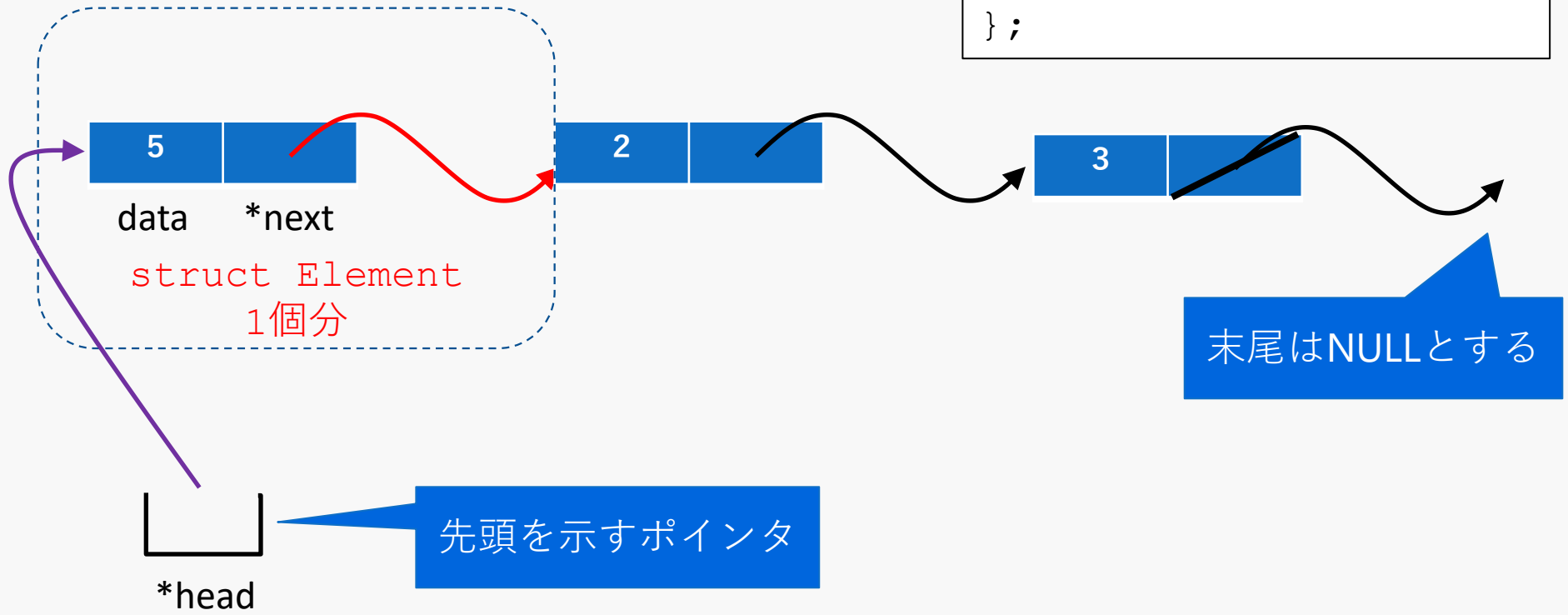


- 各ノードを構造体で表現する
- 各構造体には、次のノードを指すポインタが含まれる
  （**自己参照構造体**）

リストの概念図

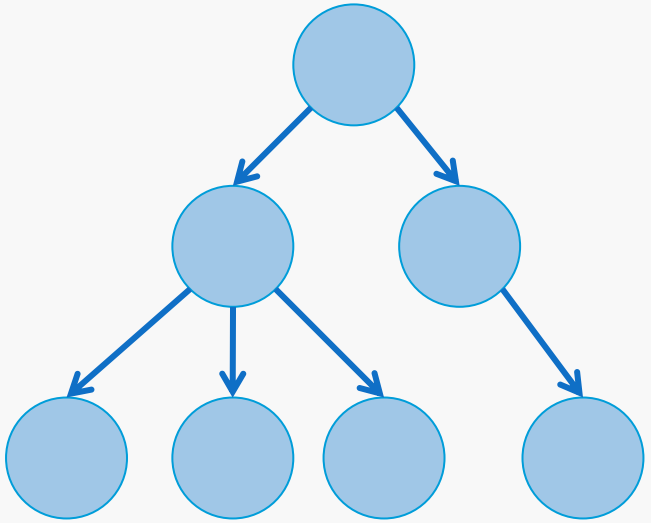
- 構造体のメンバが data と next

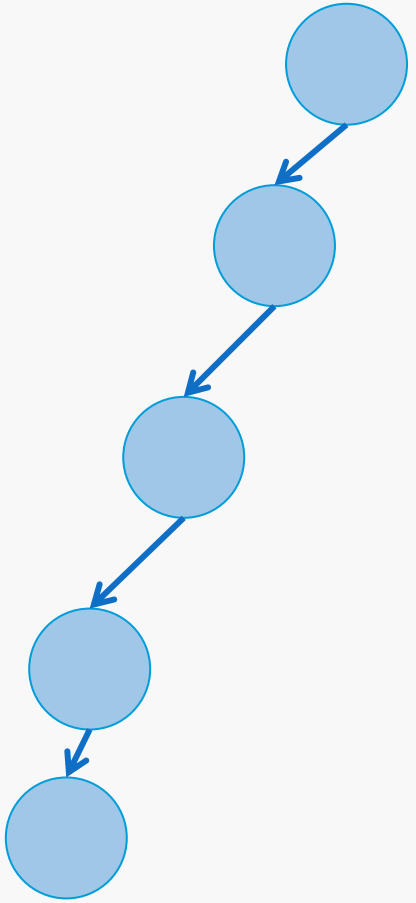
```
struct Element {
  int data;
  struct Element *next;
};
```



木構造の概念図

木構造の例





- ※線形リストは木に含まれる　**線形リスト⊆木構造**
- 木構造の代表例：**二分木**⇒子の数が2個以下

ポインタの復習

C言語のポインタ

書式について

- 同じ記号「*」がよく似た別の意味を表現
- 同じ意味の式を複数の表現で記述可能

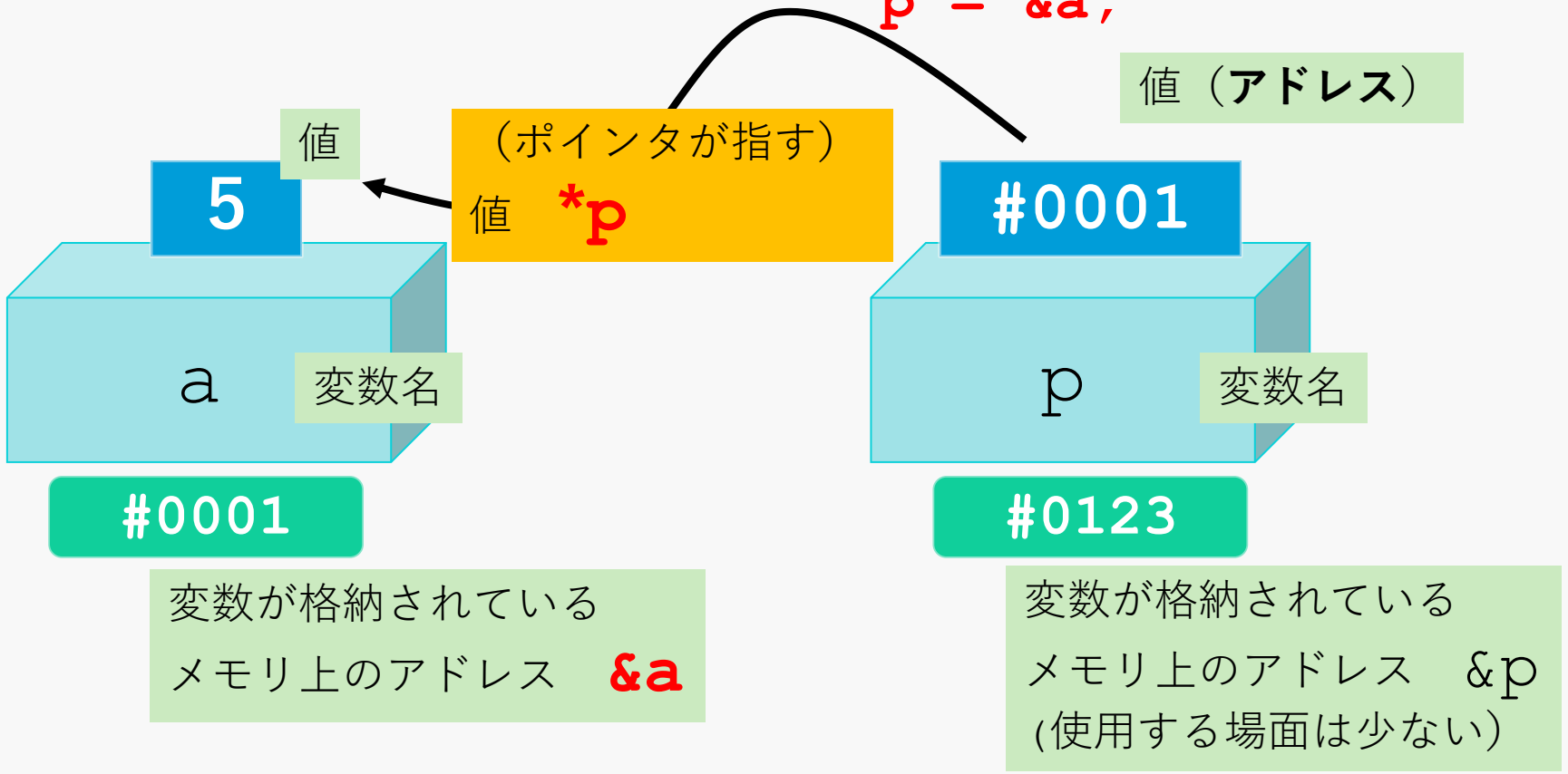
機能について

- 他の高級言語では許されない機能まで許される
- 不用意なメモリ利用もプログラマの責任で許される

変数：値とアドレス

通常の変数

ポインタ変数



ポインタの宣言、初期化、利用

宣言：**型 *ポインタ名;**

- 例）`int *p; // この * はポインタ変数であることを示す記号`

初期化：**ポインタ名 = アドレス;**

- 例1）`int a;` のアドレスを割り当てる場合

  `p = &a;`

- 例2）`int a[N];` の（先頭）アドレスを割り当てる場合

  `p = a;` または `p = &a[0];`

※宣言＋初期化をまとめた記述

- 例）`int *p = &a;`

宣言　　初期化

ポインタの宣言、初期化、利用

利用：***ポインタ名**

※ここでまた「*」が出てくるので注意

- ポインタ変数が指している変数の値の利用

  `*p = 5;`（ポインタが指す変数に値を代入）等

- ここまでで、押さえておくとよいこと

  - **ポインタは、利用する前に、宣言、初期化が必要**
  - **「*」は宣言時と利用時で意味が異なる**
  - **ポインタ変数のアドレス &p は通常利用しない**
  - **通常の変数のポインタは、引数の参照渡しで利用**

今日の内容のまとめ方

- 変数やポインタの「値」や「アドレス」の表現方法

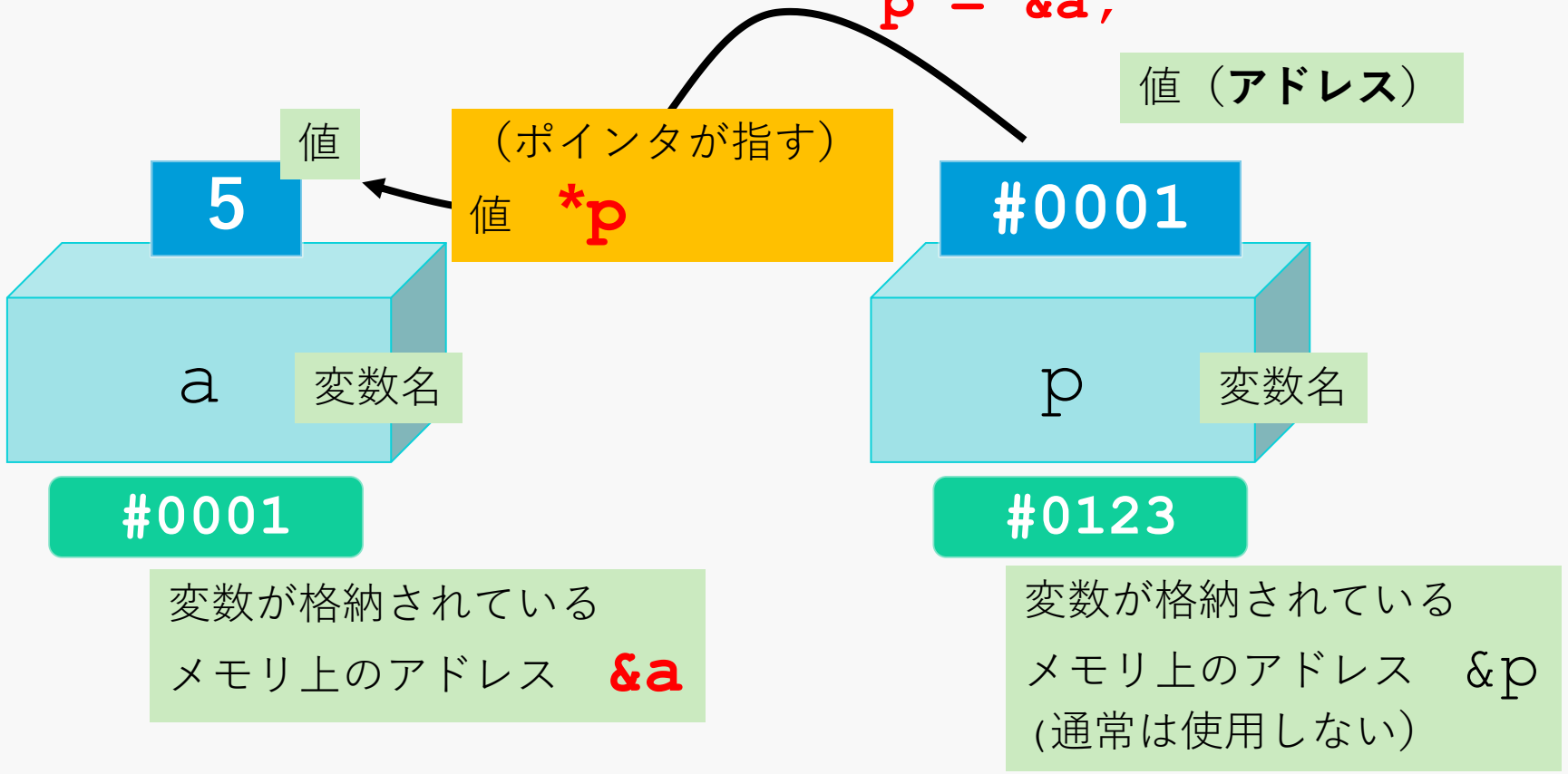
確認レポート空欄：通常の変数

変数のタイプ	宣言	初期化	値（の利用）	アドレス（の利用）
通常の変数	`int a;`	`a = 0;` 等 しなくても セグメントエラー にはならない	`a`	`&a`
ポインタ変数	`int *p;`	`p = &a;` 等 しないと セグメントエラー	`*p`	`p` ※`&p`は通常 利用しない

変数：値とアドレス

通常の変数

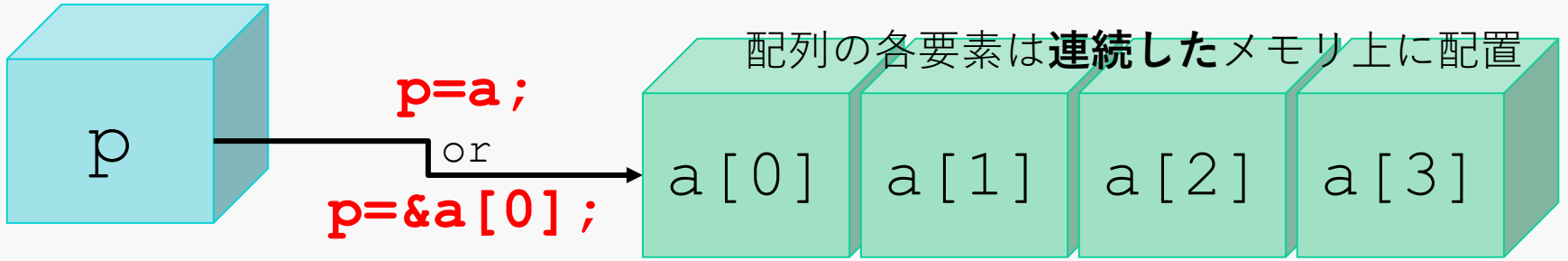
ポインタ変数



ポインタと配列

ポインタと配列はほぼ同一のもののように扱える（C言語では）

- ポインタ変数に配列の（先頭）アドレスを割り当てる
  - ✓ ポインタ変数を配列とみなしてプログラムを記述できる
  - ✓ 配列変数をポインタとみなしてプログラムを記述できる



要素の値	a[0]	a[1]	a[2]	a[3]
要素のアドレス	&a[0]	&a[1]	&a[2]	&a[3]
要素の値	*p	*(p+1)	*(p+2)	*(p+3)
要素のアドレス	p	p+1	p+2	p+3

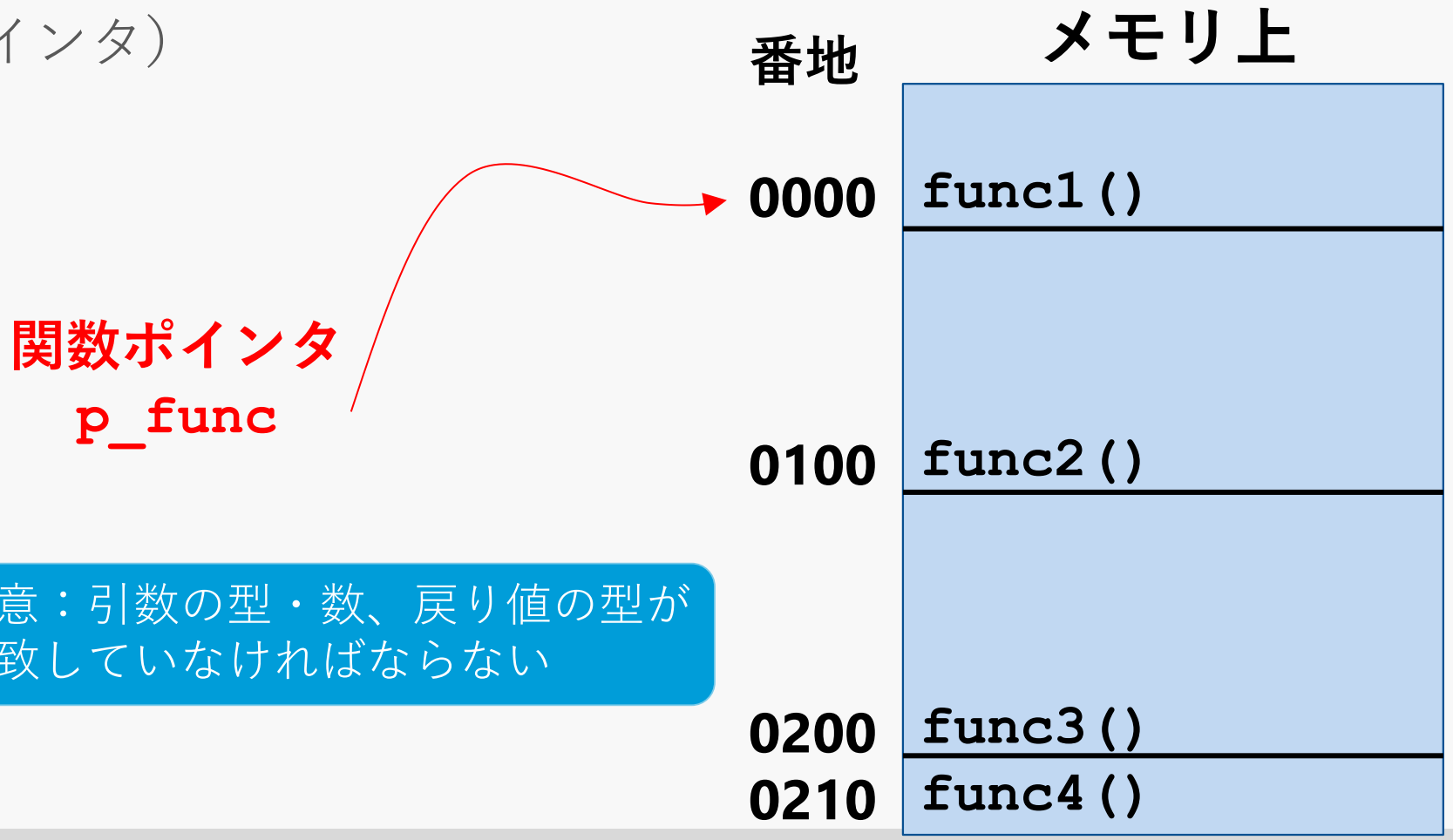
確認レポート空欄：配列とポインタ

変数のタイプ	宣言	初期化	値（の利用）	アドレス（の利用）
配列（の先頭）	`int a[N];`		`a[0]`	`a` または `&a[0]`
ポインタを配列として扱うとき	`int *p;`	`p=a;` または `p=&a[0];`	`*p`	`p`
配列の i 番目の要素について			`a[i]` **`*(a+i)`** も可	`&a[i]` **`a+i`** も可
i 番目の要素について（ポインタ風）			`*(p+i)` または `p[i]`	`p+I` または `&p[i]`

- **配列風とポインタ風の二通りの記述が可能**
- **配列を引数で渡す場合は、参照渡し＝ポインタを利用**

関数ポインタ・関数ポインタ配列

- 関数もポインタで指すことができる（関数ポインタ）



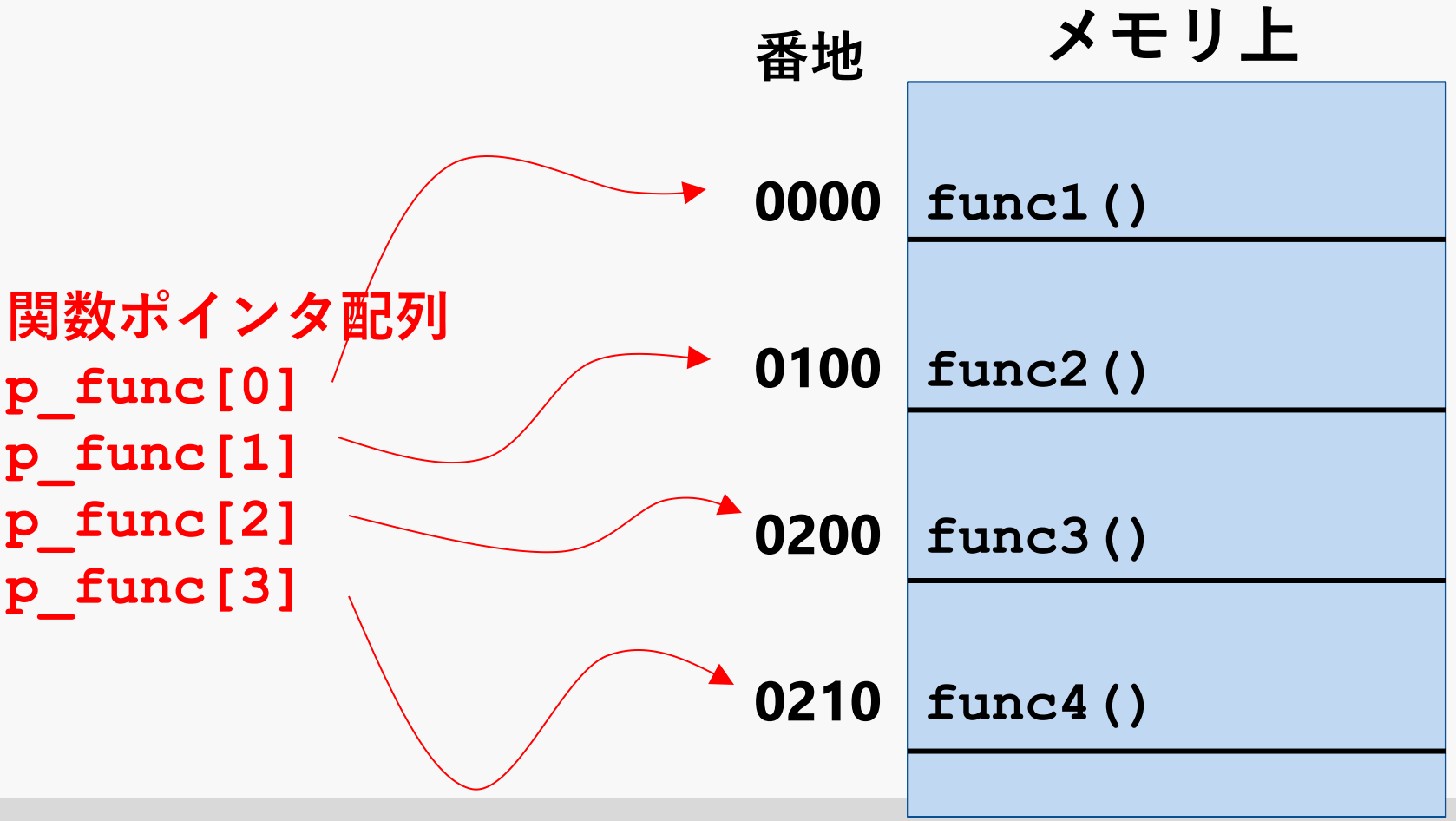
関数ポインタ・関数ポインタ配列

- 宣言：型 (*関数ポインタ名)(引数);
  - ✓ 例） `int func(int);`が定義されているとして、
    `int (*p_func)(int);`
- 初期化 ※ 以下、例のみ
  - ✓ **`p_func = func;`** **// ()をつけない！**
    - `p_func = &func;` でもOK
- 宣言＋初期化の記述
  - ✓ **`int (*p_func)(int) = func;`**
- 利用
  - ✓ **`p_func(5);`** 好き好きで **`(*p_func)(5);`**

関数ポインタ・関数ポインタ配列

すべての関数の引数の型・数、
戻り値の型が一致しているとする

- 関数を指すポインタが配列



関数ポインタ・関数ポインタ配列

- 宣言
  - ✓ 型 (*関数ポインタ名**[要素数]**)(引数);
  - ✓ 例） `int func0(int);`～ `int func4(int);`が定義されているとして、

    ```
    int (*p_func[5])(int);
    ```
- 初期化 ※ 以下、例のみ
  - ✓ `p_func[0] = func0;` // ()をつけない！
- 利用
  - ✓ `p_func[0](5);`

講義内演習の進め方

- Teams の「第〇回講義演習」から ex〇.c をダウンロード
- 指示に従って編集，コンパイル，実行
  - ✓ 環境はお任せします（アルゴリズムとデータ構造Aと同じで大丈夫です）
- 出力例と比較
- 完成したら，出力結果をソースコードに直接貼り付け，
  編集済みの ex〇.c を提出
  - ✓ 特にフィードバックの予定はなし

- 講義内の演習は**課題点に関係しません**

演習：プログラムで確認

ex1.c：ポインタのまとめ

- ポインタが指すアドレス，ポインタが指すアドレスの値を，
  通常の変数，配列，関数の場合に正しく出力する
  - ✓ printf()で，ポインタが指すアドレスを表示するためには
    %p を使ってください
- 配列中の二つの値を入れ替える swap 関数を作成して，正しく呼び出す
  - ✓ 参照渡しに注意