変数領域の動的確保

(C 言語のプログラムでは、通常)

必要な変数や配列は全てその冒頭で宣言し、

その変数のためのメモリ領域を確保する。

しかし、

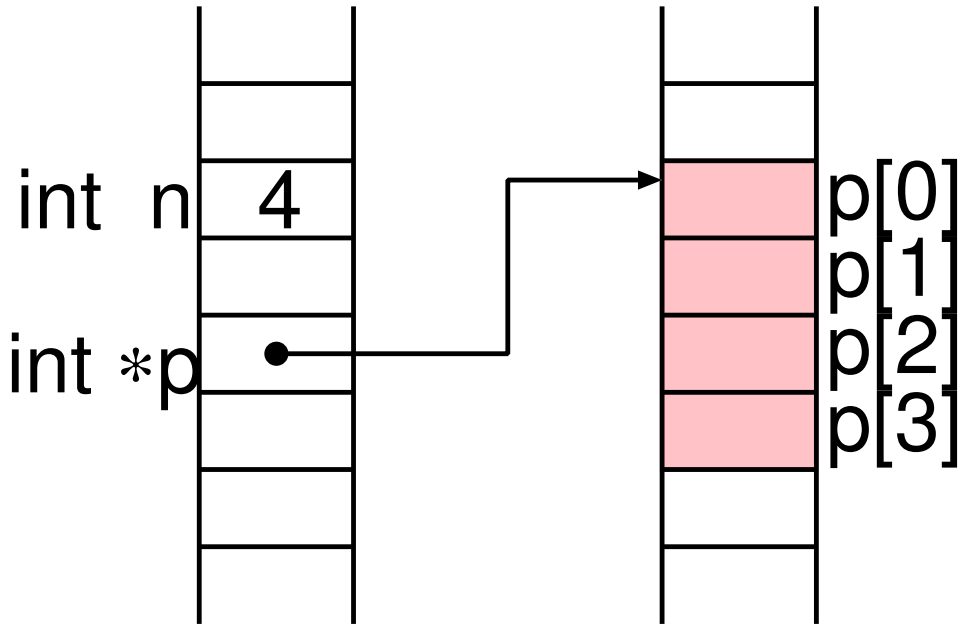
必要な大きさが決まってから
必要なだけ領域確保したいこともある

$\longrightarrow$ **メモリの動的確保**
(dynamic memory allocation)

メモリの動的確保:

```
int n;
int *p;

p = (int *)malloc(sizeof(int) * n);
```



メモリの動的確保: 関数 malloc() を用いる

(1) その型へのポインタ変数を予め宣言

```
type *p;
```

(2) 関数 malloc() でメモリ領域を確保し、
その先頭アドレスをポインタ変数に代入

```
p=(type *)malloc(sizeof(type)*n);
```

(3) そのポインタ変数を用いて、
確保した領域を間接参照

`p=(`*type* `*)malloc(sizeof(`*type*`)*n);`

(a) その変数のサイズ (メモリ量) を求める

(b) 必要な個数を掛けて、
確保するメモリ領域量を決定

(c) 所定の大きさのメモリ領域を確保して、
その先頭アドレスを返値として返す

(d) 適切な型へのポインタにキャスト
(明示的な型変換)

(e) 確保した領域の先頭アドレスを覚えておく

課題 1:

`vector0.c` **を修正して、**

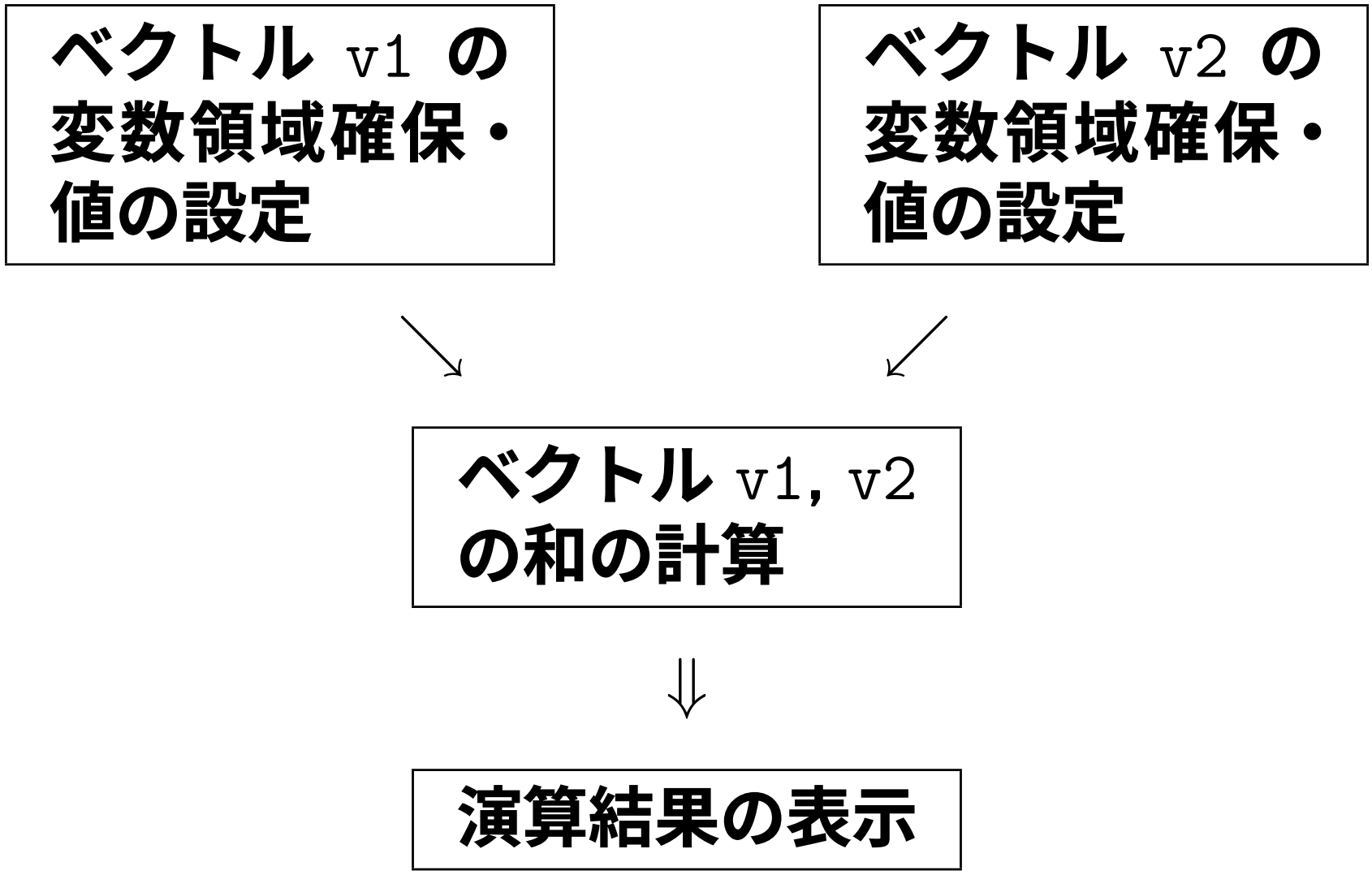
ベクトルの成分の個数を入力してから、

その分だけのメモリ領域を確保することにより、

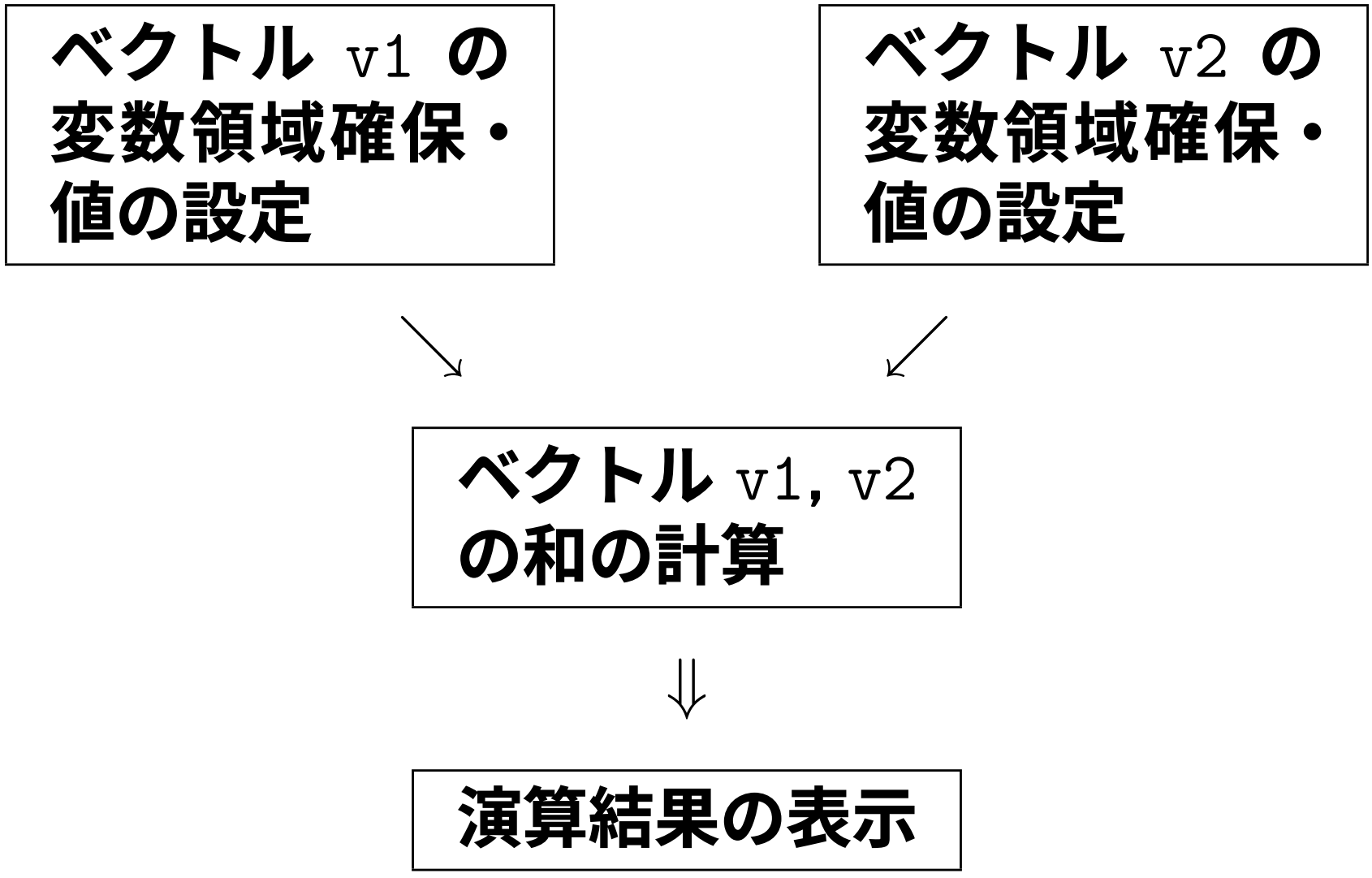
無駄なくメモリを利用するようにした

プログラム `vector1.c` **を作成せよ。**

(extra feature 歓迎!!)



⟶ それぞれを関数化すると簡明・明快



⟶ **それぞれを関数化すると簡明・明快**

変数領域の動的確保により、

入力に応じて異なる大きさのデータに

対応することが出来たが、

計算中に大きさが随時変わるような

データには対応できていない。

(例: 多項式)

このような
不定長・可変長のデータを扱うには、

更に柔軟性を持った設計が必要である。

例えば、
多項式 (1 変数で変数名 X)
を扱うには、

どのようなデータ構造を設計すれば
良いだろうか。

多項式 :: $\emptyset$ || **項＋多項式**

多項式を実現する「項」のデータ構造:

- **項自身の内容**
- **その次の項はどれか**

「値」と「その次を指すもの」との組
を基本単位としたデータ構造

$\longrightarrow$ **リスト構造**

多項式 $::$ $\emptyset \mid\mid$ **項＋多項式**

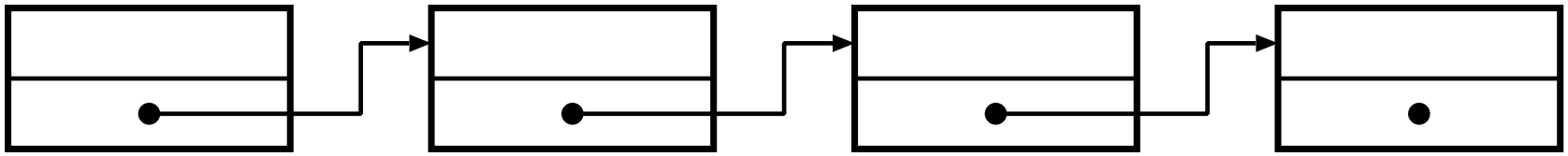
多項式を実現する「項」のデータ構造:

- **項自身の内容**
- **その次の項はどれか**

「値」と「その次を指すもの」との組
を基本単位としたデータ構造

$\longrightarrow$ **リスト構造**

リスト構造 (linked list)



```
typedef struct node {
    int val;
    struct node *next;
} Node;
```