基本的なデータ構造（1）

表

アルゴリズムとデータ構造B

第05回

本日の予定

配列の動的生成（記憶域の動的な確保）の復習

- 静的な確保から動的な確保へ
- 利用関数：`malloc()`

基本的なデータ構造（１）：表（テーブル）

- **2次元配列で実現**
- **2次元配列の動的生成への発展**

課題１補足

- 問題２：malloc と scanf の使い方

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

✓ **int** *p で 要素数 n の **int** 型配列を確保するときの書き方.
今回は **struct stock** 型だから…

stock_p = malloc(sizeof(**struct stock**) * n);

scanf でのデータの取得：アドレスを渡すことに注意

例）scanf(“%s”, stock_p[i].title);
scanf("%d", &stock_p[i].year);

課題１補足

- 問題５：メンバ stock × メンバ price の値でソートする
  - ✓ 参考）ex3.c から

```
void bsort(int a[], int n) {  //バブルソート
    int i, j, tmp;
    for (i=0; i<n-1; i++)
        for (j=0; j<n-1-i; j++)
            if (a[j] > a[j+1]) {  //配列aの要素の値でソート
                tmp = a[j];
                a[j] = a[j+1];
                a[j+1] = tmp;
            }
}

int main() {
    …
    set_random(p, n); //pはint型の配列（とみなすもの），nは要素数
```

配列の動的生成

静的な確保：`int a[N];` ※Nは十分大きな値

- コンパイル時にそのサイズが固定
- プログラムの実行開始から終了まで確保される
  - ✓ 無駄が多い（かもしれない）

動的な確保：

- プログラムの実行中に必要な分だけを変数で指定して確保
- 不要になったら記憶域を解放（`free`）

必要なときに必要な分だけ確保するので効率的

動的生成の手順

1. **配列に相当するポインタを用意する**
   - ✓ 1次元配列ならば，１連鎖（通常の）ポインタ
     - `int a[N];` 相当は `int *p;`
   - ✓ **今日の内容**：２次元配列ならば，２連鎖ポインタ
     - `int a[N][N];` 相当は `int **p;`
2. **`malloc()` 関数の戻り値をポインタに代入する**
   - ✓ `p = malloc(sizeof(int)*n);`
   - ✓ ここで `n` は要素数を示す**変数**とする
     - 定数での指定ではあまり意味がない
3. **ポインタ `p` を配列とみなして処理する**
   - ✓ 値：`p[i]`（配列風），`*(p+i)`（ポインタ風）等
4. **不要になったら `free()`で確保した領域を解放する**

malloc() の使い方

- malloc()（マロック，エムアロック）
- #include <stdlib.h>
  - ✓ このヘッダファイルの中に定義が記述されている
- 型 *ポインタ名; に対して，
  ポインタ名 = malloc(sizeof(型)*要素数);
  - ✓ malloc() の引数は、確保する**バイト数**
  - ✓ sizeof(型) でその型１つ分のバイト数を計算
  - ✓ 要素数は変数で示すようにする
  - ✓ 確保できた場合はポインタにその領域の先頭アドレスのポインタが，
    確保できない場合はNULLポインタが返される
- 使い終わったら領域を解放：free(ポインタ名);

ex03.c


```c
int main() {
    // 設問２：一次元配列に相当する一連鎖のポインタ a を宣言する
    int *p;

    int n, i; // n は要素数
    // 設問３：scanfを用いて配列の要素数を取得し n へ代入
    printf("設問3：scanfを用いて要素数を取得し n へ代入\n");
    printf("要素数を入力：");
    scanf("%d", &n);

    // 設問４：取得した要素数を指定して配列を動的に生成する
    p = malloc(sizeof(int) * n);

    // エラー処理
    if (p == NULL) {
        printf("領域が確保できませんでした\n");
        exit(1);
    }

    printf("%dの要素数の領域を確保しました\n", n);
```

ex03.c


```c
    // 設問５：配列をランダムな値で初期化
    set_random(p, n);
    printf("設問６：ソート前の配列の中身をfor文を用いて出力\n");
    // 設問６：ソート前の配列の中身をfor文を用いて出力
    for (i=0; i<n; i++)
        printf("%3d ", p[i]);
    printf("\n");

    // 設問７：バブルソートを行う関数を呼び出して，p をソート
    bsort(p, n);

    printf("設問８：ソート後の配列の中身を出力\n");
    // 設問６と同様に，配列の中身を出力
    for (i=0; i<n; i++)
        printf("%3d ", p[i]);
    printf("\n");

    // 設問９：配列の利用が終わったら，領域を解放する
    free(p);
    printf("確保した領域を解放しました\n");
```

基本的なデータ構造（1）：表（テーブル）

本日の内容はここから

教科書 pp.74-77（多次元配列）

データ構造概要

- 表（テーブル）
- 棚（スタック）
- 待ち行列（キュー）
- 線形リスト（リンクリスト）
- 木構造（ツリー）
- グラフ

データ構造概要（教科書p.41-83）

- **表（テーブル）**
- 棚（スタック）
- 待ち行
- 線形リ
- 木構造
- グラフ

	名前＼科目	国語	社会	数学	理科	英語
1	赤川一郎	75	80	90	73	81
2	岸　伸彦					
3	佐藤健一			75		
4	鈴木太郎					
5	三木良介					90

表5.1　表（table）

表（テーブル）の実現

基本：2次元配列で実現

- 例）成績表，行列
  - ✓ Excelのシートがイメージしやすい
- 2次元配列での実現方法

```
int seiseki_data[4][5]; //（暗黙的に）4人，5科目
seiseki_data[1][3] = 83; //花子の理科

#define N1 4 // 4人
#define N2 5  // 5科目
int seiseki_data[N1][N2];
```

  - ✓ **利点**：プログラムでの取り扱いが容易
  - ✓ **欠点**：各行，各列の意味が分かりにくいかも（？）

	A	B	C	D	E	F
1	名前＼科目	国語	社会	数学	理科	英語
2	高専　太郎	98	92	90	95	67
3	高専　花子	80	81	80	83	85
4	豊田　次郎	88	68	75	69	84
5	豊田　三郎	62	63	100	65	69

改善策

欠点：各行，各列の意味が分かりにくいかも

- 改善策1：各列の要素番号を列挙型で定義する
  - ✓ `enum {KOKUGO, SYAKAI, SUUGAKU, RIKA, EIGO};`
  - ✓ `seiseki_data[0][KOKUGO] = 98; //太郎の国語. [0][0]よりはまだ分かりやすい`

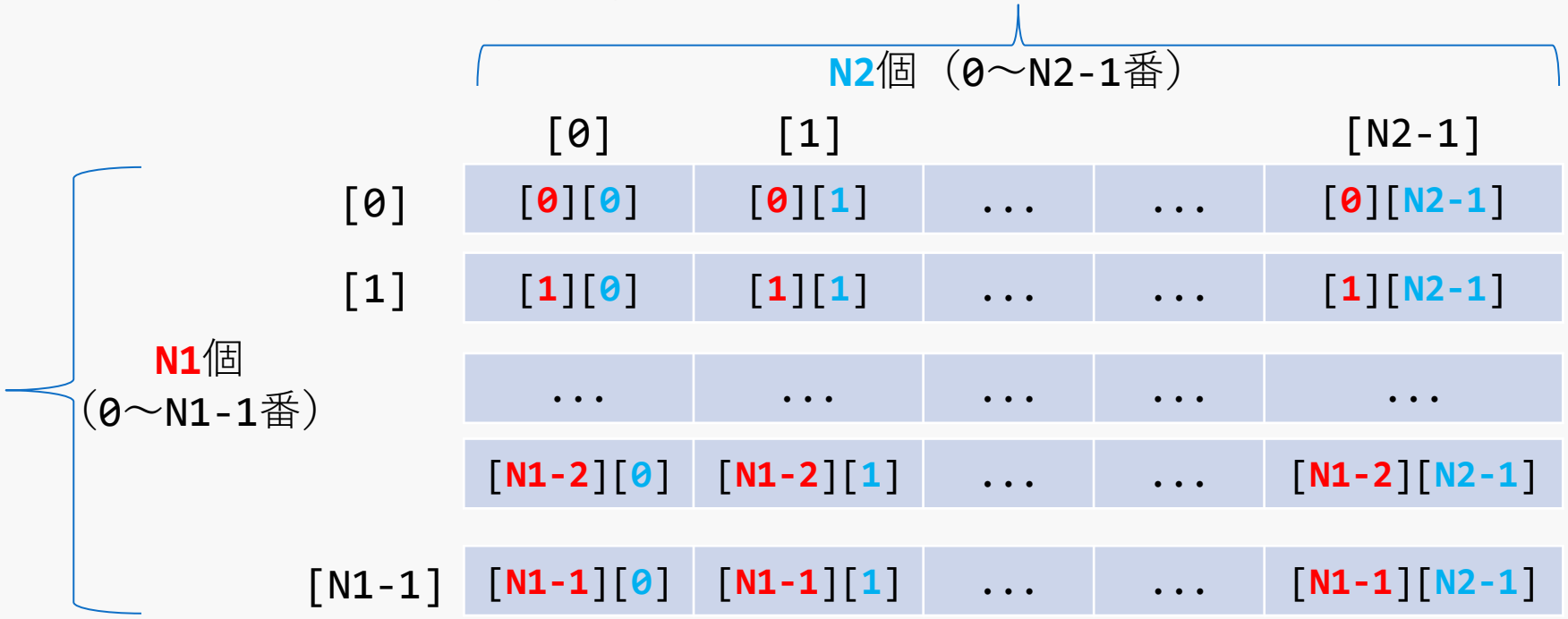
- 改善策2：構造体配列でデータを保持する（第3回講義で学習済み）
  - ✓
```
struct Seiseki {
  char name[20]; //名前も管理
  int kokugo, syakai, suugaku, rika, eigo;
} seiseki_data[N];
```

確認レポート５の問題1はここまでの内容

静的な2次元配列のデータ構造

実メモリ上での割り当て

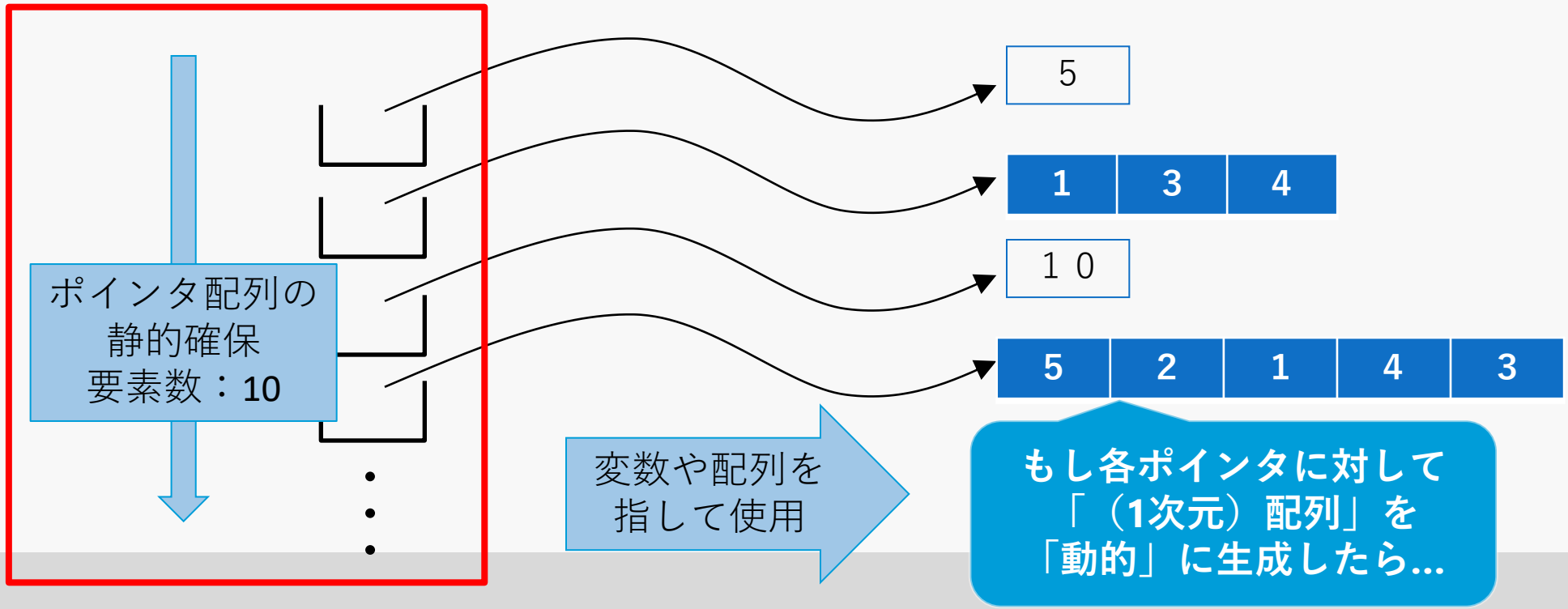
- `int seiseki_data [N1][N2];` の場合
  - ✓ ※実際にはメモリ上に1次元的に確保



ポインタの配列

- 二次元配列の前にまず：**ポインタの配列**

```
int *p[10];  //int 型ポインタ10個の配列
```

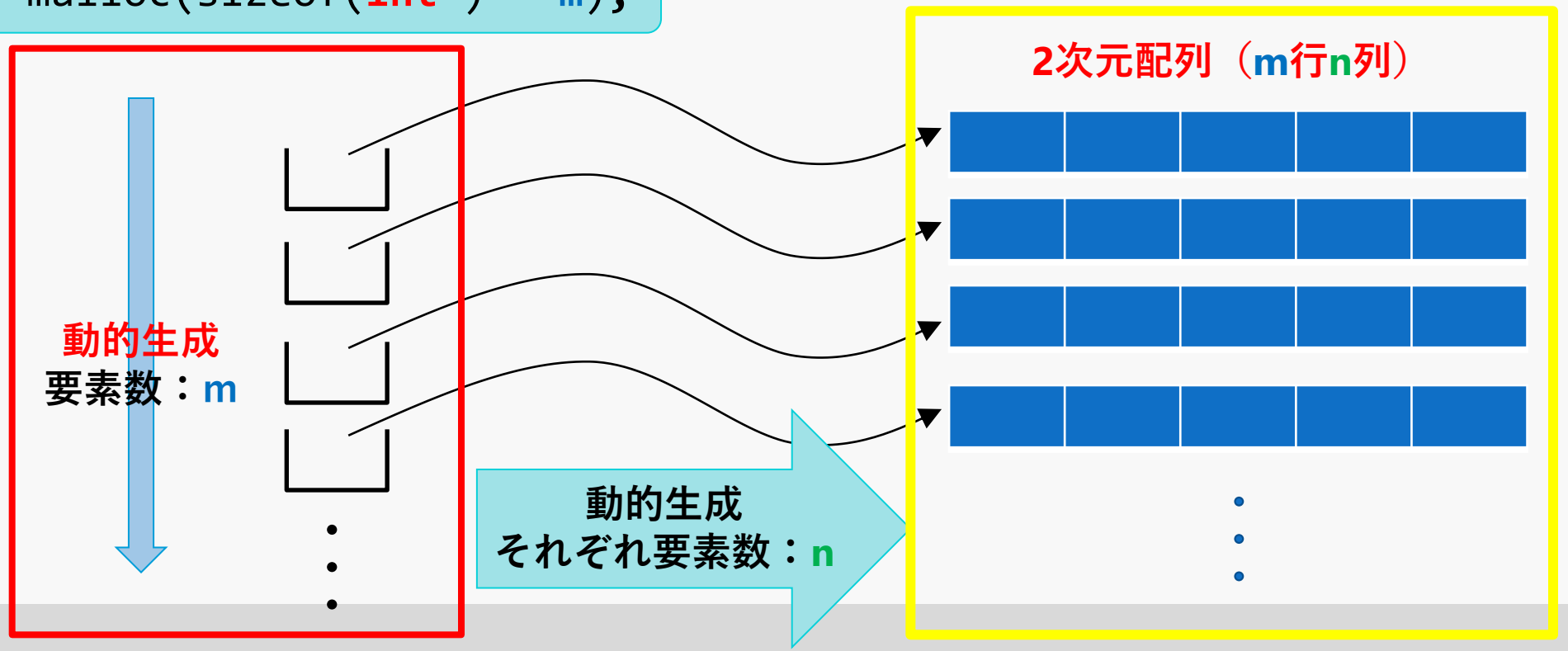


2次元配列の動的生成

- 概念図
  - ✓ int **p; に対して

```
for (i = 0; i < m; i++)
 p[i] = malloc(sizeof(int) * n);
```

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



2次元配列の動的生成（m行n列）

2連鎖のポインタを宣言

- `int **p;`

int *型の配列（要素数 m 個）を動的に生成

- `p = malloc(sizeof(int*) * m);`

生成した配列の 0 番目から m-1 番目の要素 p[i] に対し，
要素数 n の int 型配列を動的に生成して割り当て

- `for (i=0; i<m; i++)`

一次元配列の動的生成と（ほぼ）同じコード

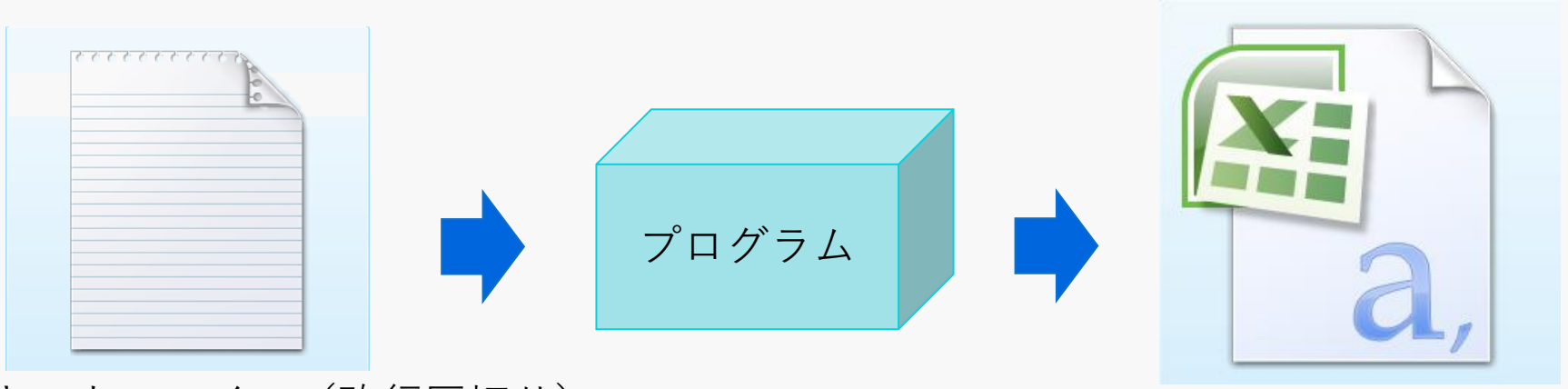
```
p[i] = malloc(sizeof(int) * n);
```

p[i], p の順に free()

演習　ex05.c

動的に生成した int型の2次元配列を表とみなし，
データの入力，合計点・平均点の算出，出力を行う

- 入力：テキストファイル（input.txt）
- 出力：CSVファイル（output.csv）



テキストファイル（改行区切り）
input.txt

CSVファイル（カンマ区切り）
output.csv

演習　ex05.c

ファイルの入出力に必要な手続き

- FILE *fp; //ファイルポインタ
  fp = fopen(“ファイル名”,“r（またはw）”);
  - ✓ r : ファイルを読み込みでオープン
  - ✓ w : ファイルを書き込みでオープン
- データの読み込み：fscanf(fp, “%d”, &a);
  データの書き込み：fprintf(fp, “%d”, a);　← 今回の演習はコレではうまくいかない
- fclose(fp); //ファイルを閉じる

CSV（Comma Separated Values）形式

- カンマ区切りの**テキストファイル**
- Excel 等で利用可能．テキストファイルなので，メモ帳等でも見れる

演習　ex05.c

本日の演習手順

- ex5.c, input.txt をダウンロード
  - ✓ input.txt は ex5.c と同じ場所に置く
    - 分かってる人は必ずしもそうでなくて良いです
- 2次元配列を動的に生成
  - ✓ スライド14を参考に
- input.txt から2次元配列に数値を読み込み
- output.csv に，読み込んだ数値および合計点・平均点を計算して書き込み
- 生成した2次元配列を free