線形リスト序論

アルゴリズムとデータ構造B

第11回

本日の内容

課題４確認

線形リスト序論

- 線形リスト実現のための準備
- 線形リストを実現するための構造体
  - ✓ 自己参照構造体

課題 4 確認

ex09.c では，グローバル変数を用いて一つのキュー，
それに付随するアルゴリズム（Enque や Deque 等）を実装

- ⇨ 変数の役割を考えれば構造体を用いる方が妥当
- ⇨ 別のキューを使用するとき，`int que[MAX];` 等はもちろん，
  `int Enque(int x);` 等も追加で用意する必要がある．
  `Enque, Deque, Initialize, Display` は，
  どのキューに対しても使用できる汎用的な関数であるべきなため，
  キューを引数として受け取るように変更

課題 4 確認


```
struct Que {
    int que[MAX];
    int front, rear, num;
};

int main() {
    struct Que que1, que2; // 二つのキュー
    Initialize(&que1); // que1 を初期化
    ...
```

```
void Initialize(struct Que *q) {
  q->front = q->rear = q->num = 0;
  for (int i=0; i<MAX; i++)
    q->que[i] = 0;
}
```

※以下ではうまくいかない

```
Initialize(que1);
void Initialize(struct Que q) {
  q.front = q.rear = q.num = 0;
  //配列の 0 での初期化は省略
}
```

構造体変数を値渡ししている
⇨ 構造体変数のメンバの値を変更する場合は
参照渡しをする必要がある

課題 4 確認


```
int x, count;
while(1) {
  scanf("%d", &x);
  if (x が 0 だったら)
    ループ抜ける;
  count += 1;
  //カウントを 2 で割った余りが 0 か 1 かによって
  //x を que1 に Enque するか que2 に Enque する
  //count 増やす（タイミングは任意）
  //que1, que2 を Display する
}
int total = 0;
//que1 と que2 の両方に対して実行する
while (num が 0 じゃない間) {
  デキューする;
  //total に足していく
  //最後に -1 が返ってくることに注意
}
```

```
❯ ./kadai4
?Enque x = 10
que1
 front->     0:      10
             1:       0  <-rear
             2:       0
             3:       0
             4:       0
que2
 front->     0:       0  <-rear
             1:       0
             2:       0
             3:       0
             4:       0
?Enque x = 20
que1
 front->     0:      10
             1:       0  <-rear
             2:       0
             3:       0
             4:       0
que2
 front->     0:      20
             1:       0  <-rear
             2:       0
             3:       0
             4:       0
?Enque x = 30
```

```
?Enque x = 0
10
30
50
20
40
Total: 150
que1
             0:      10
             1:      30
             2:      50
 front->     3:       0  <-rear
             4:       0
que2
             0:      20
             1:      40
 front->     2:       0  <-rear
             3:       0
             4:       0
```

スタック，キューのプログラミング（要点）

- スタックの ptr, キューの front や rear について，
  データを **入れて/出して** から **ずらす** のか，
  ずらして から **入れる/出す** のか
- 確認レポート11 裏面の内容を確認

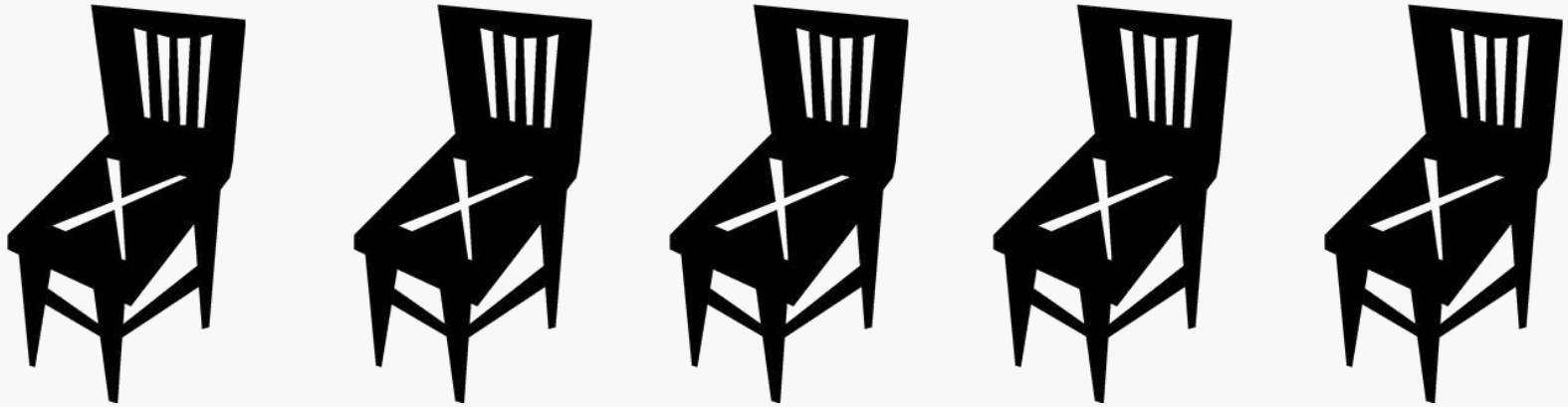
線形リスト序論

本日の内容はここから

線形リスト

- 1次元のデータ構造
- ⇧ の意味では1次元配列と同等だが，**各要素が要素番号をもっていない**

配列のイメージ（番号付き座席）

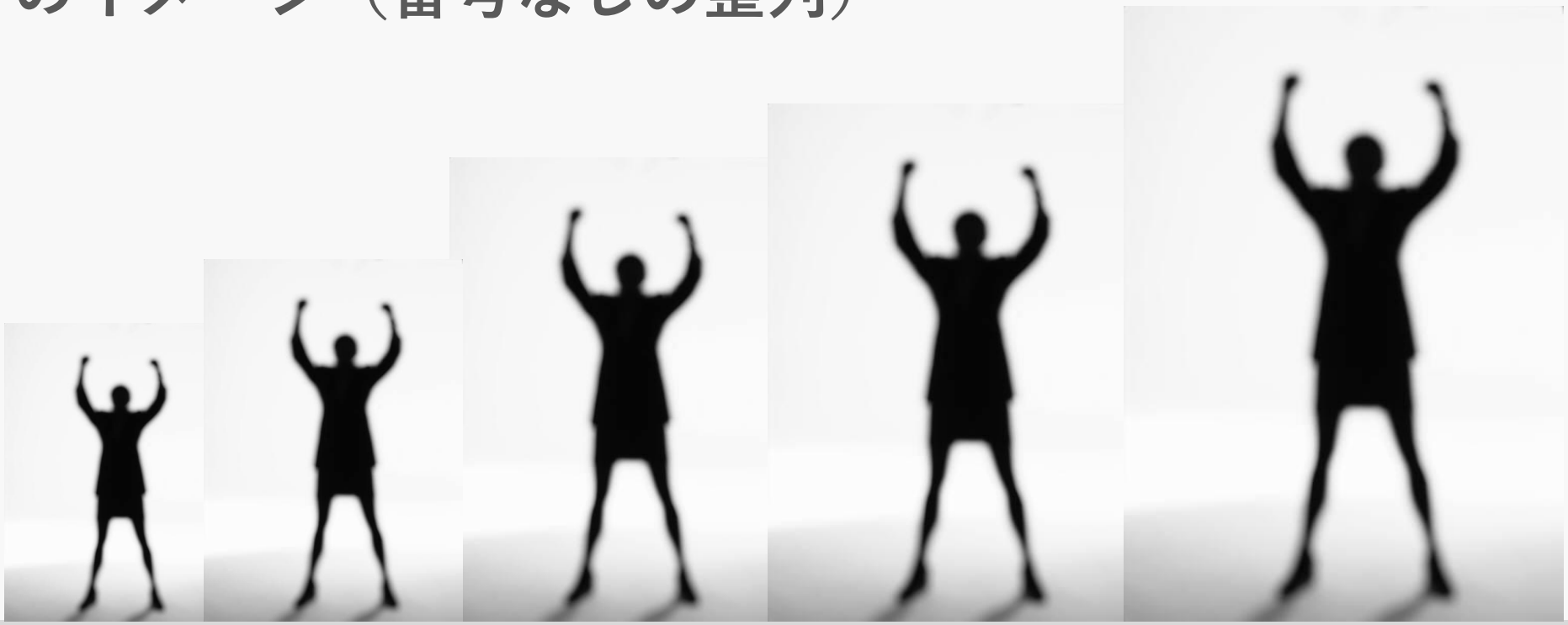


1　　2　　3　　4　　5

線形リスト

- 1次元のデータ構造
- ⇧ の意味では1次元配列と同等だが，**各要素が要素番号をもっていない**

リストのイメージ（番号なしの整列）



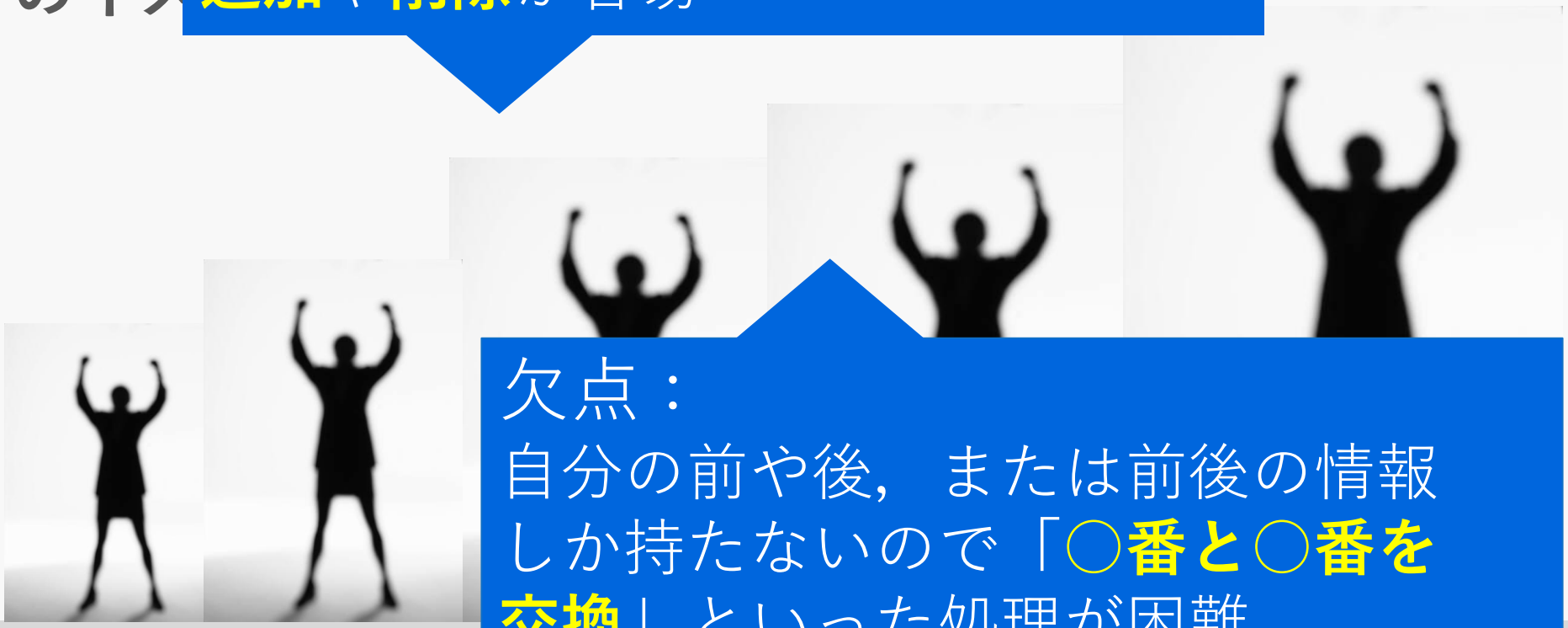
線形リスト

- 1次元のデー
- ⇧の意味では

もっていない

リストのイメ

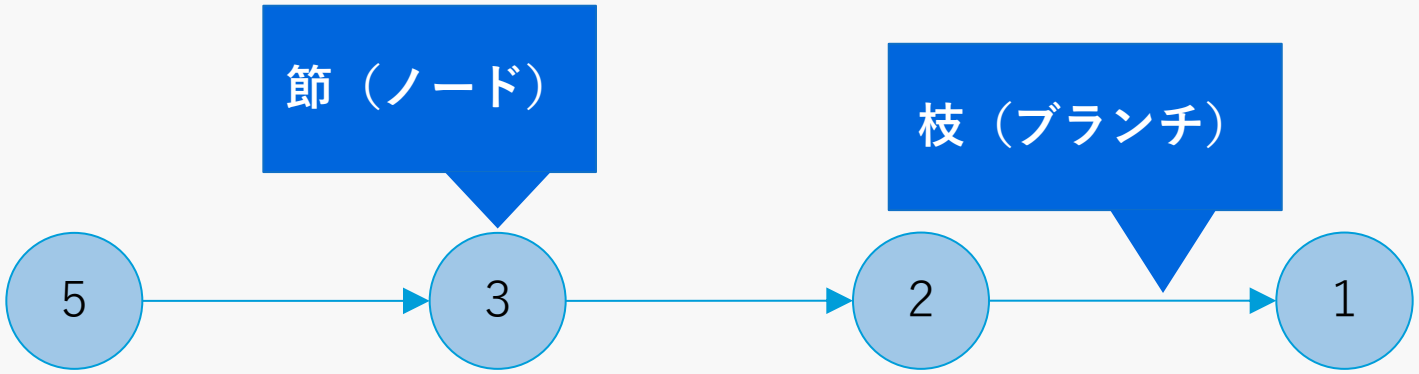
利点：
自分の前や後，または前後の情報のみ保持していれば，列ができる．
追加や**削除**が容易



欠点：
自分の前や後，または前後の情報しか持たないので「**○番と○番を交換**」といった処理が困難

線形リストの概念図

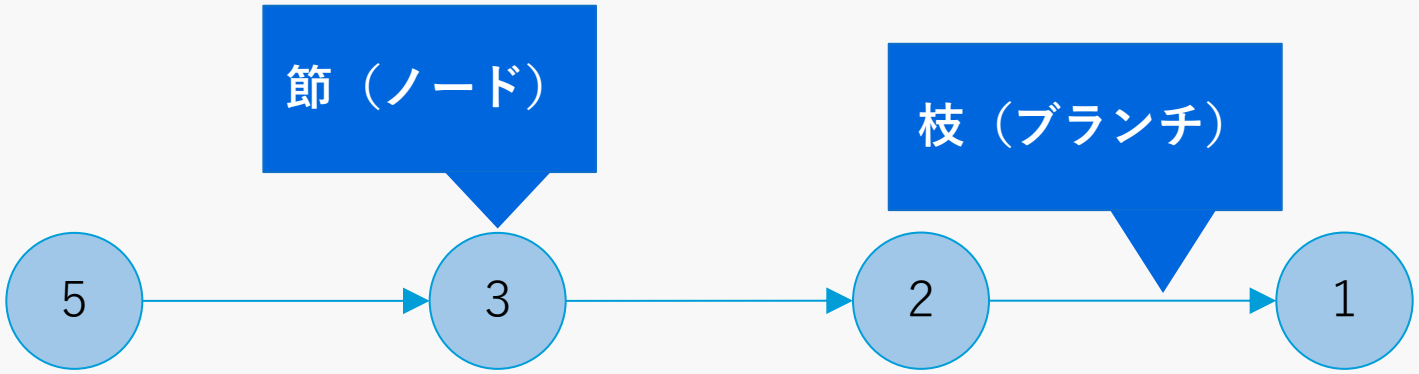
整数値を並べた線形リスト



- 節（ノード）：リストを構成する要素．ブランチで連結される．構造体で表現する．
  ノードにはデータ部とポインタ部が必要であり，
  ポインタ部は次のノード（を表す構造体）を指す
- 枝（ブランチ）：ここでは，構造体のメンバとして定義した
  「次のノードを指すためのポインタ部」が相当する

線形リストの概念図

整数値を並べた線形リスト



- 各ノード&ブランチを**構造体**で表現する
- 次のノードを指すポインタ（スライド10でのブランチ = ポインタ部）が含まれる構造体を特に**自己参照構造体**と呼ぶ

復習：構造体

構造体：複数のデータをまとめて扱うためのデータ型

- 構造体の定義

タグ名

```
struct Seiseki {
    char name[32];
    int kokugo, shakai, suugaku, rika, eigo;
};
```

メンバ

名前	国語	社会	数学	理科	英語
高専太郎	100	80	60	75	90
…	…	…	…	…	…

- 構造体変数の宣言

変数名

```
struct Seiseki ssk_a;
```

- メンバの参照

```
ssk_a.kokugo = 100;
```

復習：構造体ポインタ

構造体ポインタ：構造体の格納されているアドレスを保持するポインタ変数

- 構造体変数のアドレスを代入して利用

```
struct Seiseki *p_ssk = &ssk_a; //宣言＆初期化
```

- メンバの参照（二通り）

```
printf(“%d”, (*p_ssk).kokugo); //ドット演算子
printf(“%d”, p_ssk->kokugo); //アロー演算子
```

線形リストのデータ構造

自己参照構造体

- 自分自身と同じ型のポインタをメンバにもつ

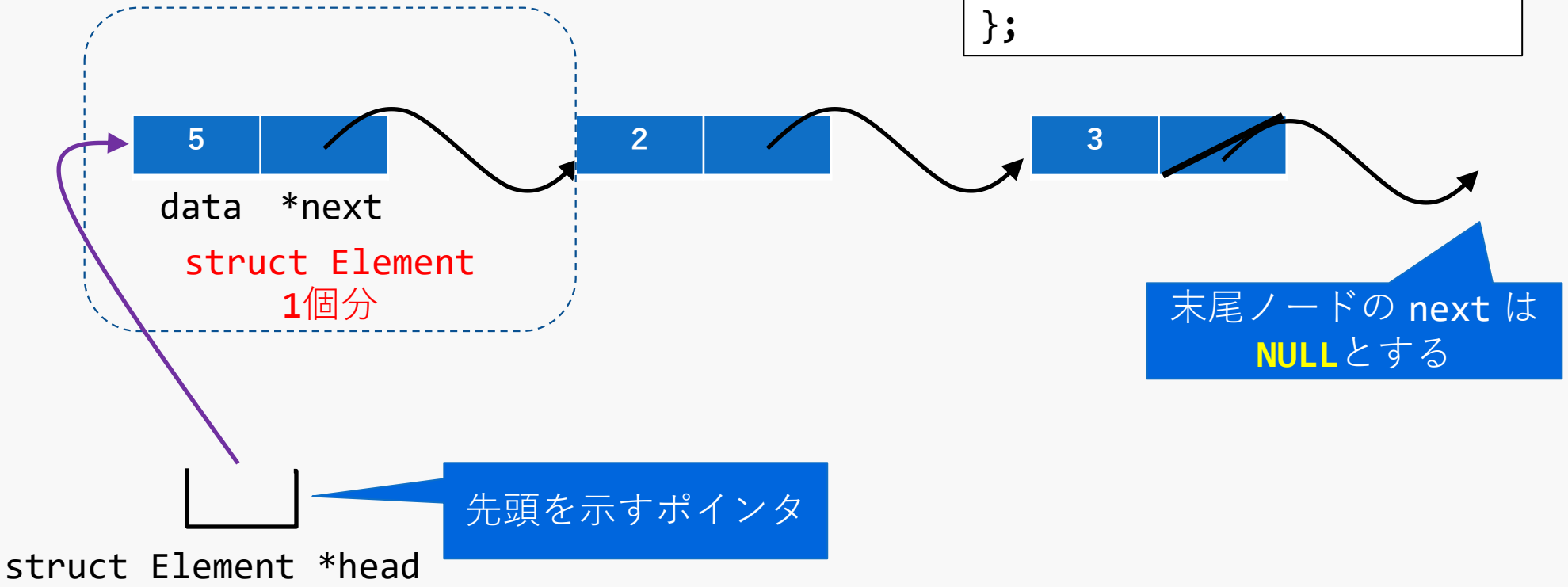
- 整数値を並べた線形リストのための構造体

```
struct Element {
  int data; // データ部（線形リスト内の値）
  struct Element *next; // ポインタ部（次のノードを指すポインタ）
};
```

線形リストの概念図

構造体のメンバが data と next

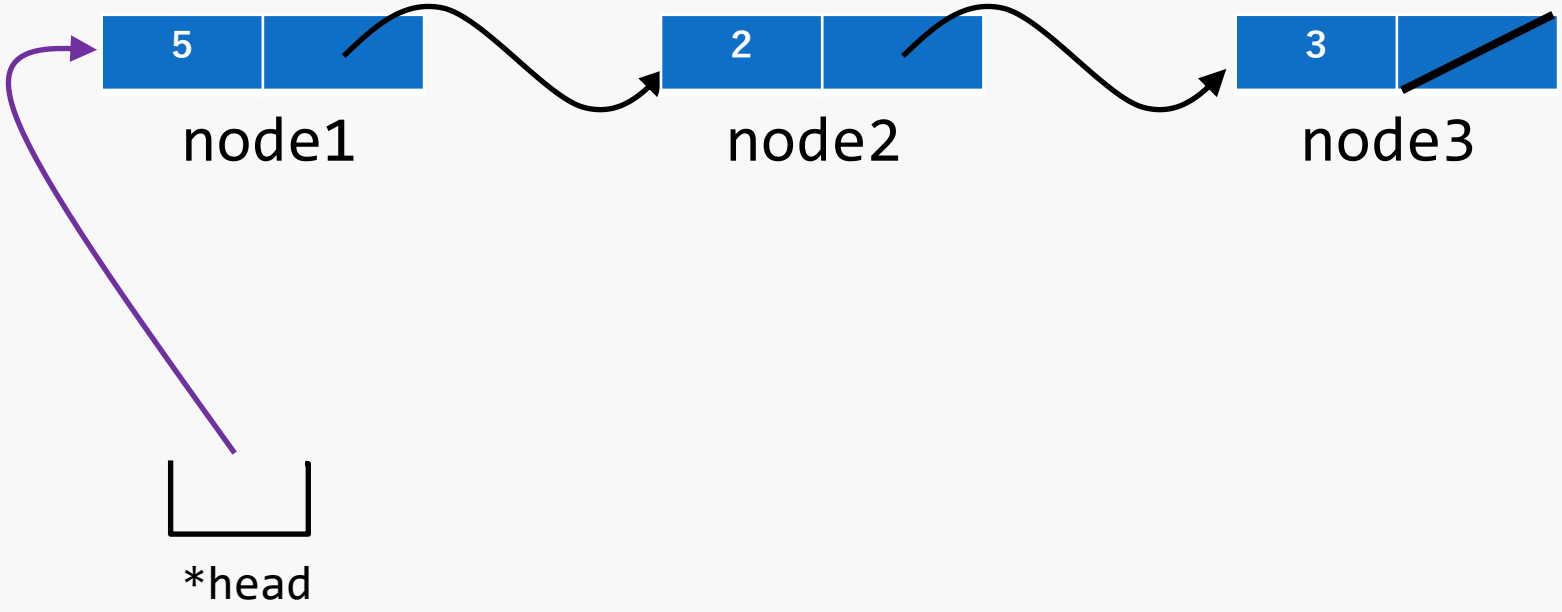
```
struct Element {
  int data;
  struct Element *next;
};
```



ノードの連結（練習）

図のように**静的**に領域を確保したノードを手動で連結してみる

- `struct Element node1, node2, node3; //ノード三つ分`
  `struct Element *head; //線形リストの先頭を示すポインタ`



解答例：

```
node1.data = 5;
node1.next = &node2;

node2.data = 2;
node2.next = &node3;

node3.data = 3;
node3.next = NULL;

head = &node1;
```

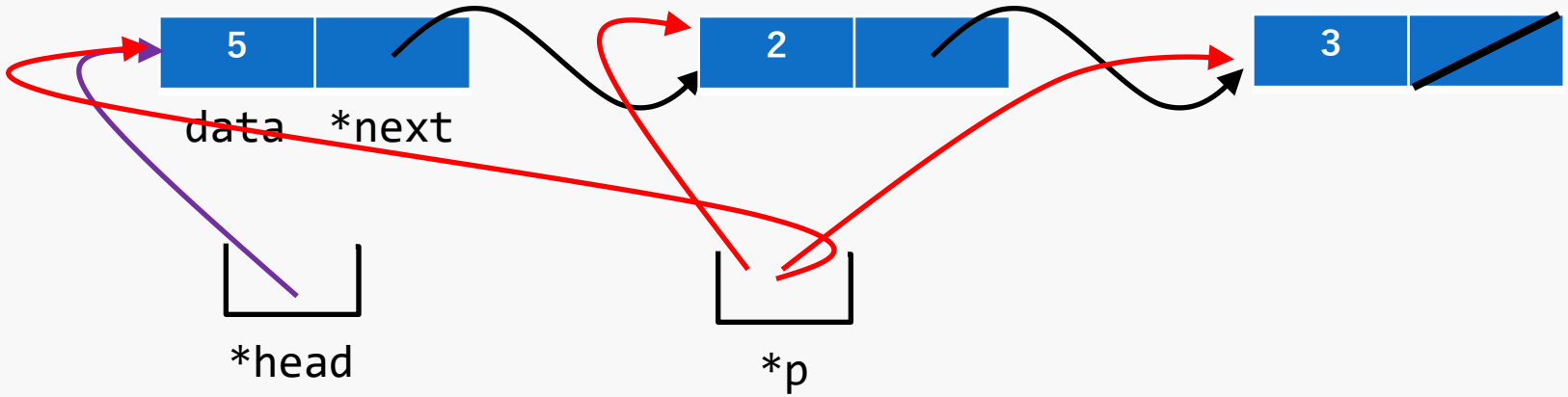
線形リストの走査

リストを先頭から末尾まで順にたどっていく（走査する）には？

アロー演算子 -> で構造体のメンバを参照
（ここでは next により次のノードを参照）

```
struct Element *p; //構造体ポインタ

for (p = head; p != NULL; p = p->next) {
   ...
}
```



本日の演習　ex11.c

ノード用の構造体配列 node（要素数 10）を宣言し，
配列 0 から 9 番目をノードとして順に連結する．
for 文によりリストの先頭から情報を出力する

- node[i] の data には i*i*i を代入する
- node[i] (i=0,…,9) の next には
  node の i+1 番目の要素のアドレスを割り当てる
  - ✓ 0番目の要素が先頭，9番目の要素が末尾となるようにする
- スライド「線形リストの走査」を参考に
  各ノードのアドレス，data, next（次のノードのアドレス）を出力する
  - ✓ 特に，next と次のループ時の自分のノードのアドレスが一致することを確認する