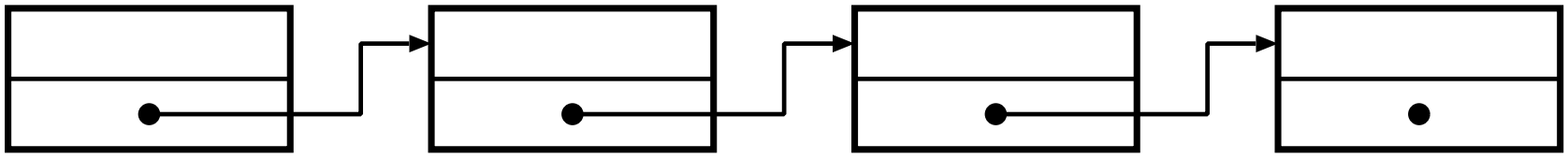
リスト構造 (linked list)

“「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



節点 (node):

- その節点自身の内容
- 次の節点を指すポインタ

$\longrightarrow$ 自身と同じ型へのポインタを
要素として持つ構造体として実装

int 型の値を持つ節点の実装

- **節点を基本の型とする**

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

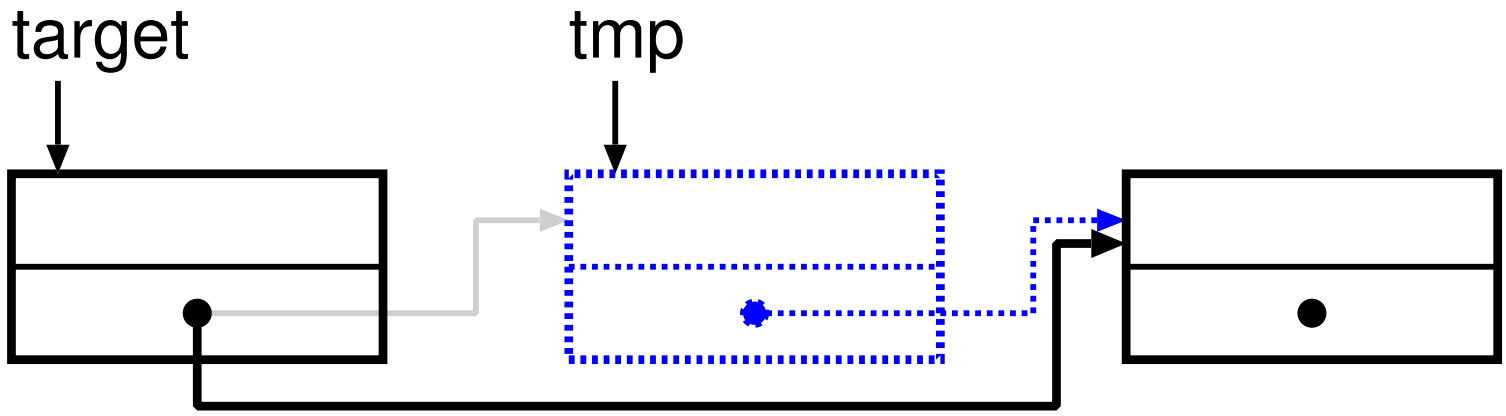
- **節点へのポインタを基本の型とする**

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

リスト構造の基本操作

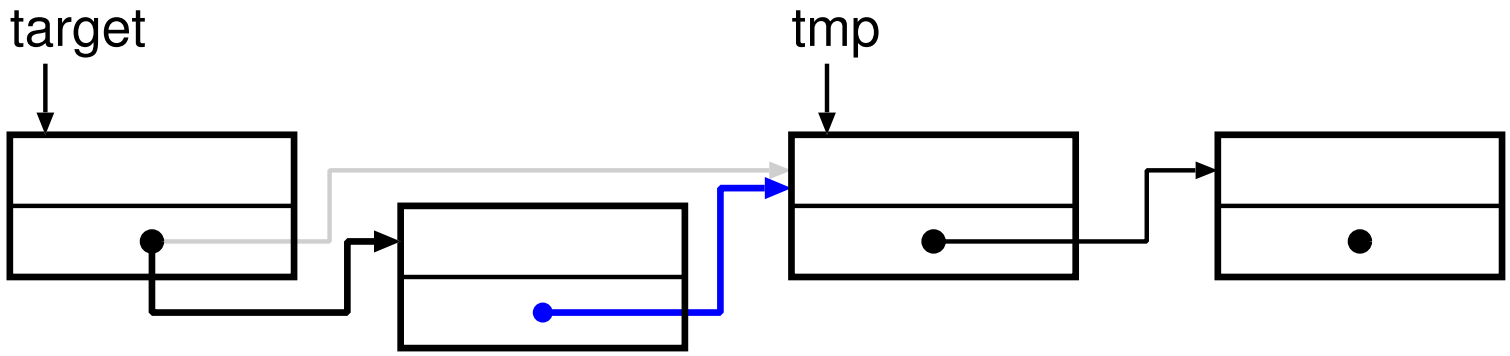
- 新規作成
- 走査
- 削除
- 挿入

リストからの節点の削除



(1) 削除したい節点の**手前の節点**に、追跡用ポインタを向ける
(2) 削除したい節点を憶えておく
(3) 削除したい節点の次の節点に、追跡用ポインタの指す節点の次へのリンクを向ける
(4) 削除したい節点の領域解放

リストへの節点の挿入



(1) 挿入したい場所の**手前の節点**に、追跡用ポインタを向ける
(2) 挿入したい場所の次の節点を憶えておく
(3) 挿入する節点を領域確保し、追跡用ポインタの指す節点の次へのリンクを新節点に向ける
(4) 憶えておいた節点に、新節点の次へのリンクを向ける

課題 3:(〆切 11/3(月))

リストを用いて、

次々と入力した整数値を

大きさの順に並べ替えて表示するプログラム

`list2.c` **を作成せよ。**

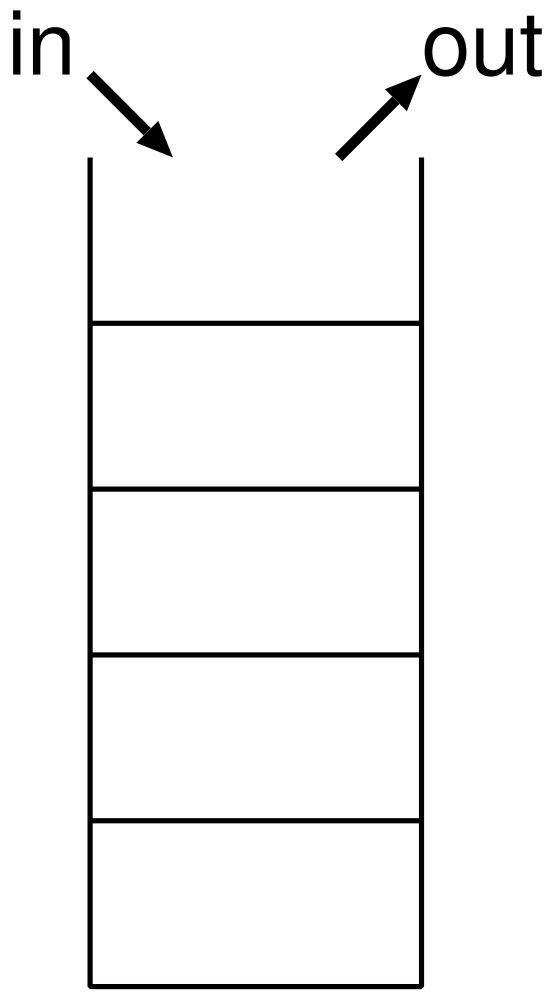
- **入力した整数を値とする新節点を、リストの適切な場所に挿入し、作成したリストの各節点の値が大きさの順になるようにする**
- `0` **が入力されたら終了**
- **先頭からリストを辿って全ての値を順に表示**

リストで表せるデータ構造の中で、
それに対して限られた操作しか行なわない
特殊な構造がある。

- スタック:
  先頭での挿入・削除のみ

- キュー:
  末尾への追加・先頭からの削除のみ

プッシュダウンスタック (pushdown stack)



- 新しい要素を先頭に挿入
  (プッシュ (push))
- 先頭の要素を取り出して削除
  (ポップ (pop))

Last-In First-Out (LIFO)

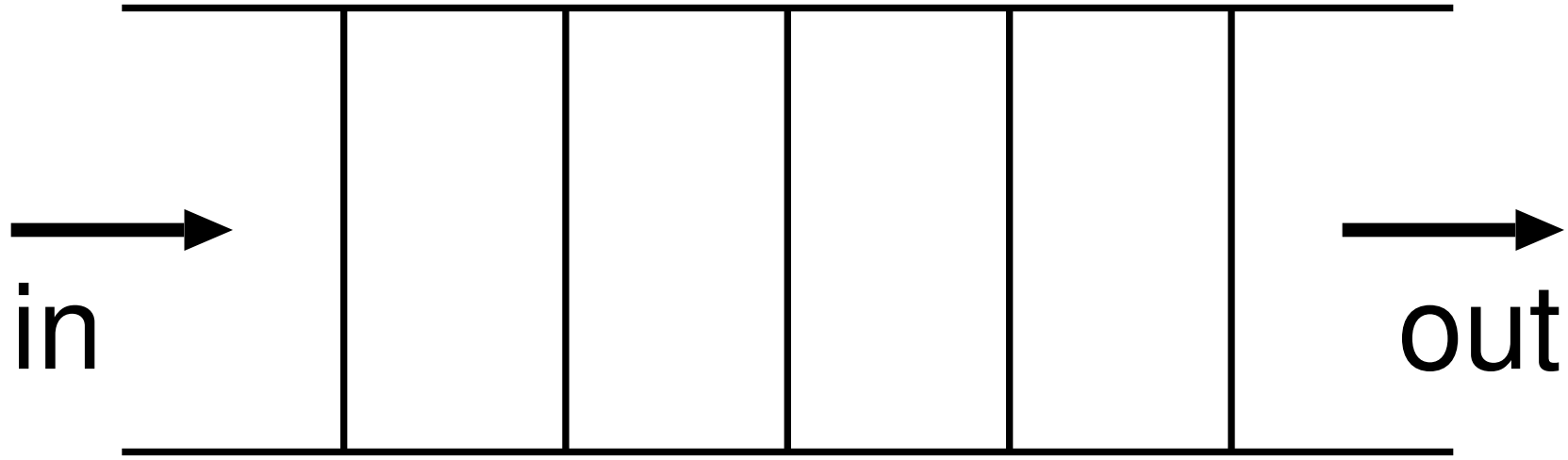
プッシュダウンスタック (pushdown stack)

- **型定義**

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

- **プッシュ:** `void push( Link, int );`
- **ポップ:** `int pop( Link );`
- **空判定:** `int isstackempty( Link );`

FIFO キュー (queue)



- 新しい要素を末尾に挿入
- 先頭の要素を取り出して削除

First-In First-Out (FIFO)

演習 5

配列とリストとについて、

それぞれの長所・短所をまとめ、

- **どのような場合に**
- **どちらを使うのが**
- **どんな意味で**

効率的か、例を挙げて考察せよ。

これでリストの基本は一段落。

今度はもっと豊富な構造として、

「次」が複数に枝分かれする

構造を考えよう。

⟶ 木 (tree) 構造

これでリストの基本は一段落。

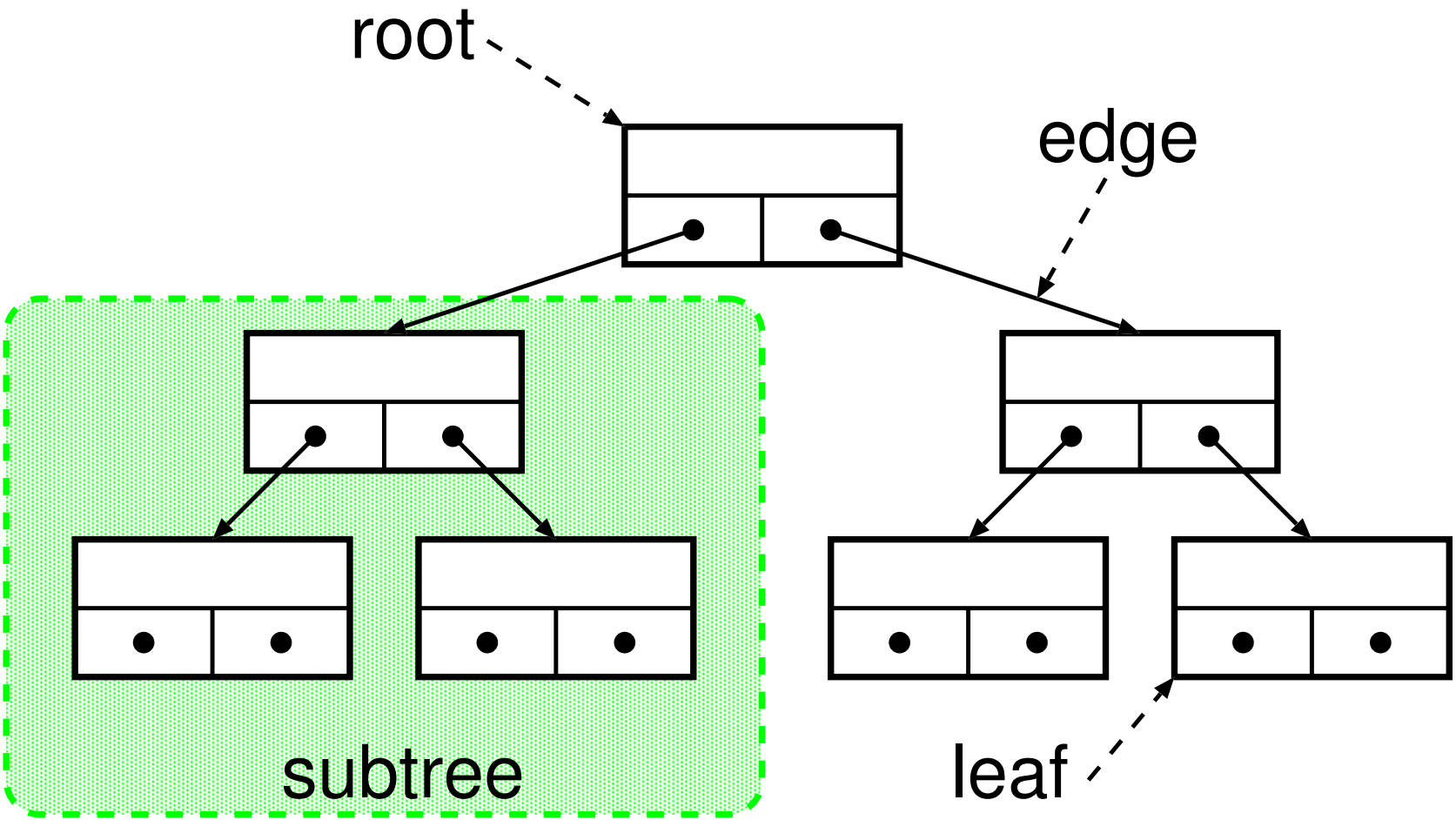
今度はもっと豊富な構造として、

「次」が複数に枝分かれする

構造を考えよう。

$\longrightarrow$ 木 (tree) 構造

木 (tree)・二分木 (binary tree)



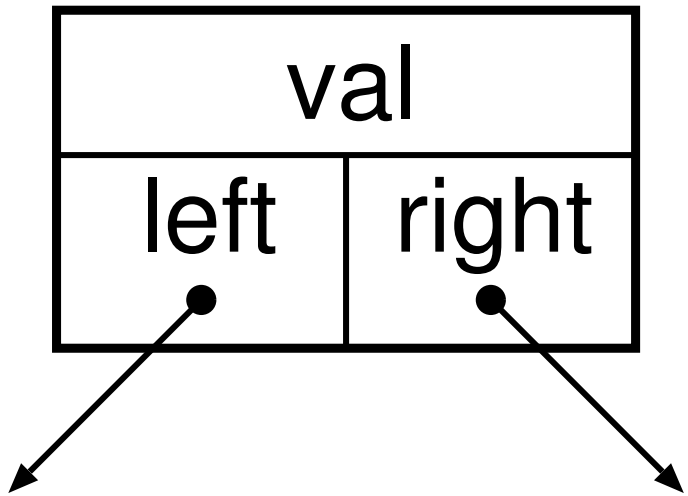
次回までの宿題 (考えてくること)

(1) (二分木と限らず)
木構造を持つと見るべきデータの具体例を、
身の回りで探せ。

(2) 例えば、3+4*2 とか 3*4+2 とかのように、
二項演算子で書かれた式は、
内在的に二分木の構造を持っている。
それを図に表せ。
また、その式の値を計算するには
どうしたらよいか。

二分木の型定義

```
typedef struct node {
    int val;
    struct node *left;
    struct node *right;
} *Link;
```



課題 4:(〆切 11/10(月))

二分木を用いて、

次々と入力した整数値を

大きさの順に並べ替えて表示するプログラム

`tree.c` **を作成せよ。**

課題 4:(〆切 11/10(月))

- 木のどの節点についても次の条件が成り立つように、入力した整数を値とする新節点を、木の適切な先端に追加する。
  - ⋆ その左にある部分木のどの要素の値も、その節点の要素の値より小さい
  - ⋆ その右にある部分木のどの要素の値も、その節点の要素の値より大きい
- 0 が入力されたら終了
- 先頭から木を辿って全ての値を順に表示

実習 4.1

コーディングする前に、

適当なデータを例に、

「箱と矢印の図式」を自分で書いてみて、

木の「成長」の仕組みを理解せよ。

例: $123, 45, 678, 90, 12, 345, 6789, 0$

木の先端への節点の追加

木の先端まで左右のどちらを辿るかを
適切に選択する他は、

先端まで辿る要領はリストと同様

次を行なう関数 `new_Node()` を作成せよ

(1) 動的に領域確保
(2) 要素の値を設定
(3) 左右のリンクを `NULL` に初期化
(4) 節点へのポインタを返す

木の走査・表示

リストと異なり枝分かれがあるので、

全ての節点を走査して表示することが

既に自明な問題ではない !!

$\longrightarrow$ 状況に応じて色々な辿り方がある !!

木の走査・表示

多くの場合、次を適切な順番で行なえば良い

- **左の部分木を表示**
- **右の部分木を表示**
- **自分自身の要素を表示**

今の問題では、
どうすれば大きさ順の表示になるか ？

木の走査・表示

多くの場合、次を適切な順番で行なえば良い

- 左の部分木を表示
- 右の部分木を表示
- 自分自身の要素を表示

今の問題では、
どうすれば大きさ順の表示になるか ？

木の走査・表示

(1) 左の部分木を表示
(2) 自分自身の要素を表示
(3) 右の部分木を表示

「部分木の表示」には
自分 (木を表示する関数) 自身を呼び出す

$\longrightarrow$ 関数の再帰呼出
(recursive function call)

木の走査・表示

(1) 左の部分木を表示
(2) 自分自身の要素を表示
(3) 右の部分木を表示

「部分木の表示」には
自分 (木を表示する関数) 自身を呼び出す

$\longrightarrow$ 関数の再帰呼出
(recursive function call)

木の走査・表示

(1) 左の部分木を表示
(2) 自分自身の要素を表示
(3) 右の部分木を表示

「部分木の表示」には
自分 (木を表示する関数) 自身を呼び出す

$\longrightarrow$ **関数の再帰呼出**
(recursive function call)