第１２週
木構造と探索（１）

分割コンパイル

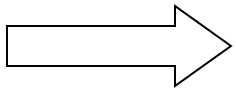
- 複数のファイルをそれぞれコンパイルし、生成されたオブジェクトファイルをまとめる（リンク）すること

  利点

  - コンパイル時間の短縮
  - オブジェクトファイルを利用・提供可能

オブジェクトファイルの生成方法

```
gcc -c filename.c
```



オブジェクトファイル

```
filename.o
```

オプション“-c”を指定する

分割コンパイルの例

- prog.c とfunc.c にプログラムが分かれている場合。
- gcc **-c** prog.c を実行するとprog.o が作成される。
- gcc **-c** func.c を実行するとfunc.o が作成される。
- この２つのオブジェクトに対して
- gcc prog.o func.o -o prog とすると実行ファイルprog を作成することができる。

まとめたいオブジェクト
の数だけ並べる

今回は、print_tree.cと自分で作成したファイル名.cの分割コンパイルを行う

print_tree.cはプロ演２公式サイトから
ダウンロードできます

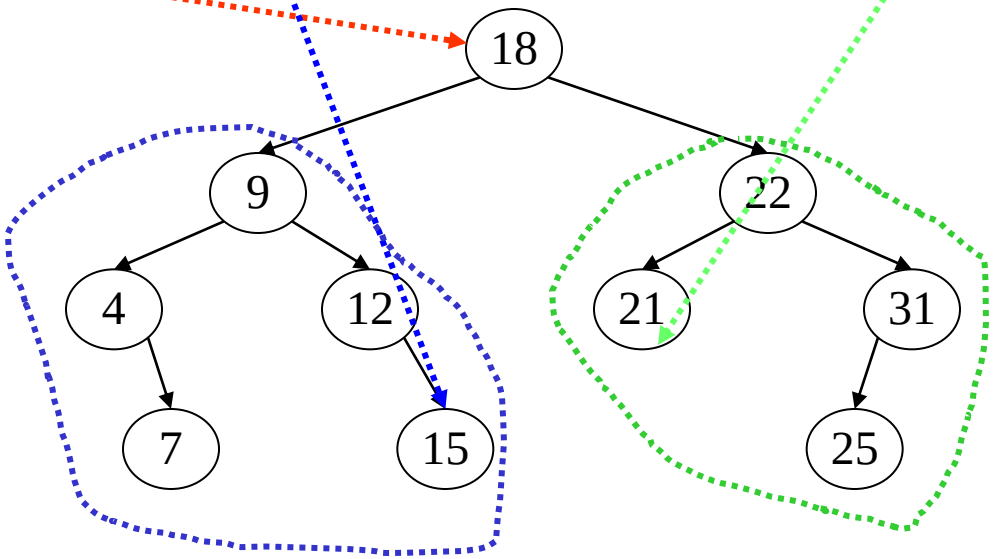
2分探索木

●定義（広い意味での定義）

- 根付き2分木の各点に要素を１つずつ対称順に配置したもの
- 対称順とは
  - 例えば下記のような順序関係を指す

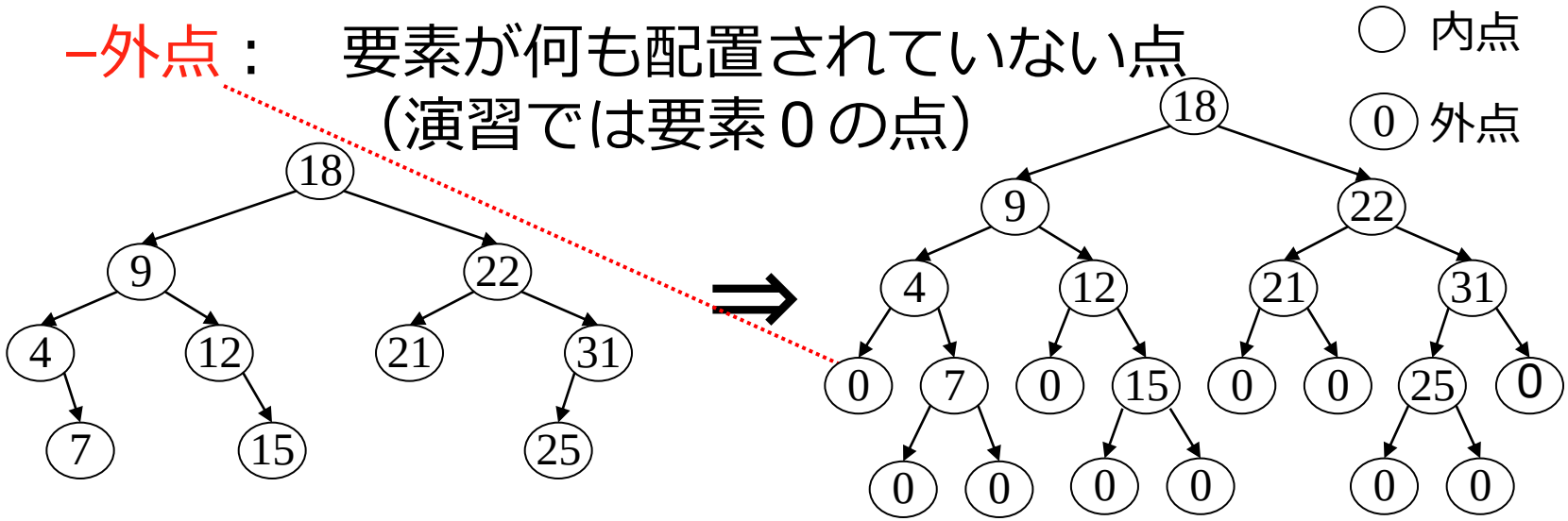
点vの左側の要素の最大値 ＜ 点vの要素 ＜ **点vの右側の要素の最小値**

例：18は、**左側最大（15）**より大きく、**右側最小（21）**より小さい



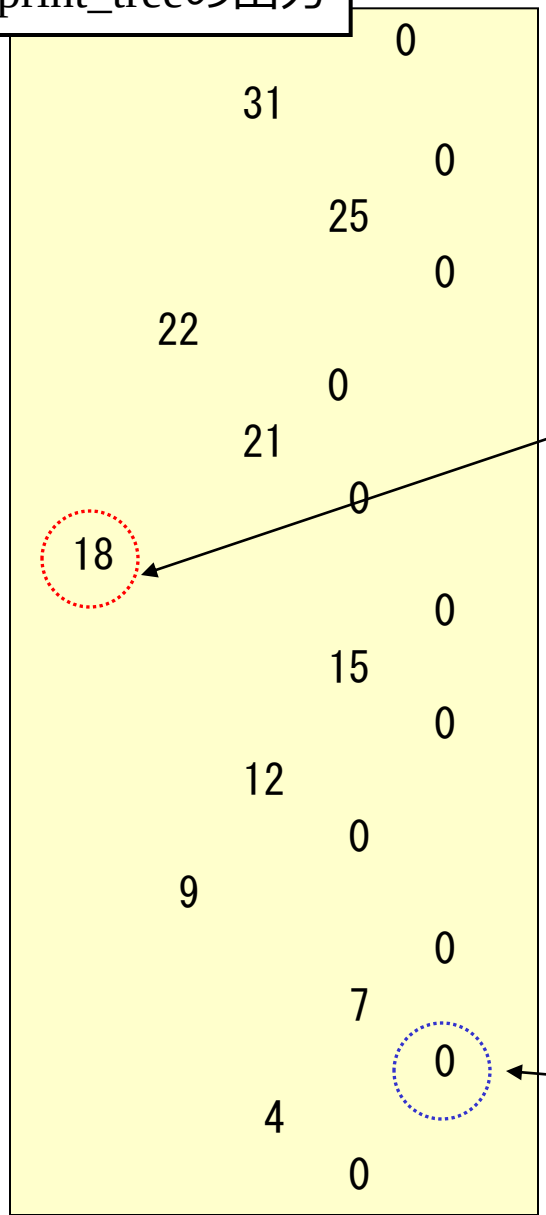
正則2分木

- 定義
  - 各点が左子、右子の両方の子を持つか、または全く子を持たない根付き2分木
    - （左子、右子の一方のみが存在する場合は正則2分木ではない）
- すべての2分探索木は正則2分木として考えられる？
  - 方法：　左子、右子の一方しか存在しない場合は外点を追加して正則2分木に変形
  - 外点：　要素が何も配置されていない点（演習では要素０の点）

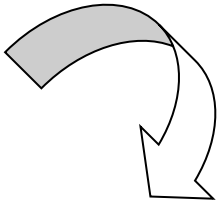


必須課題12-1：print_treeの出力の見方

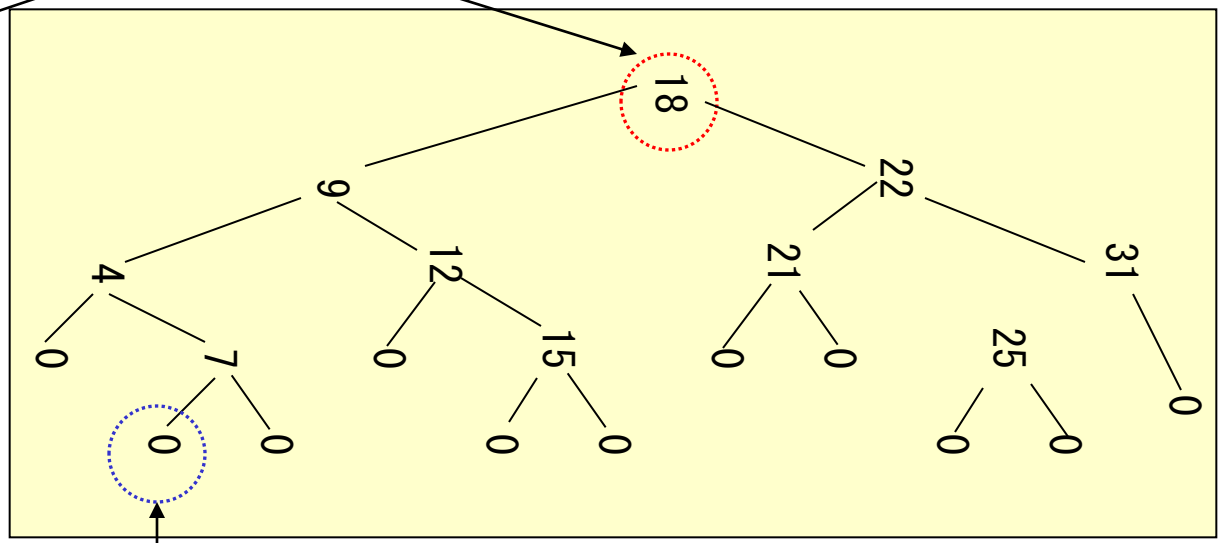
print_treeの出力



90度回転



ルートノード



外点：ノードの値が０

必須課題12-１： new_node

```
struct node *new_node(int key)
```

確保したnode 型のポインタ

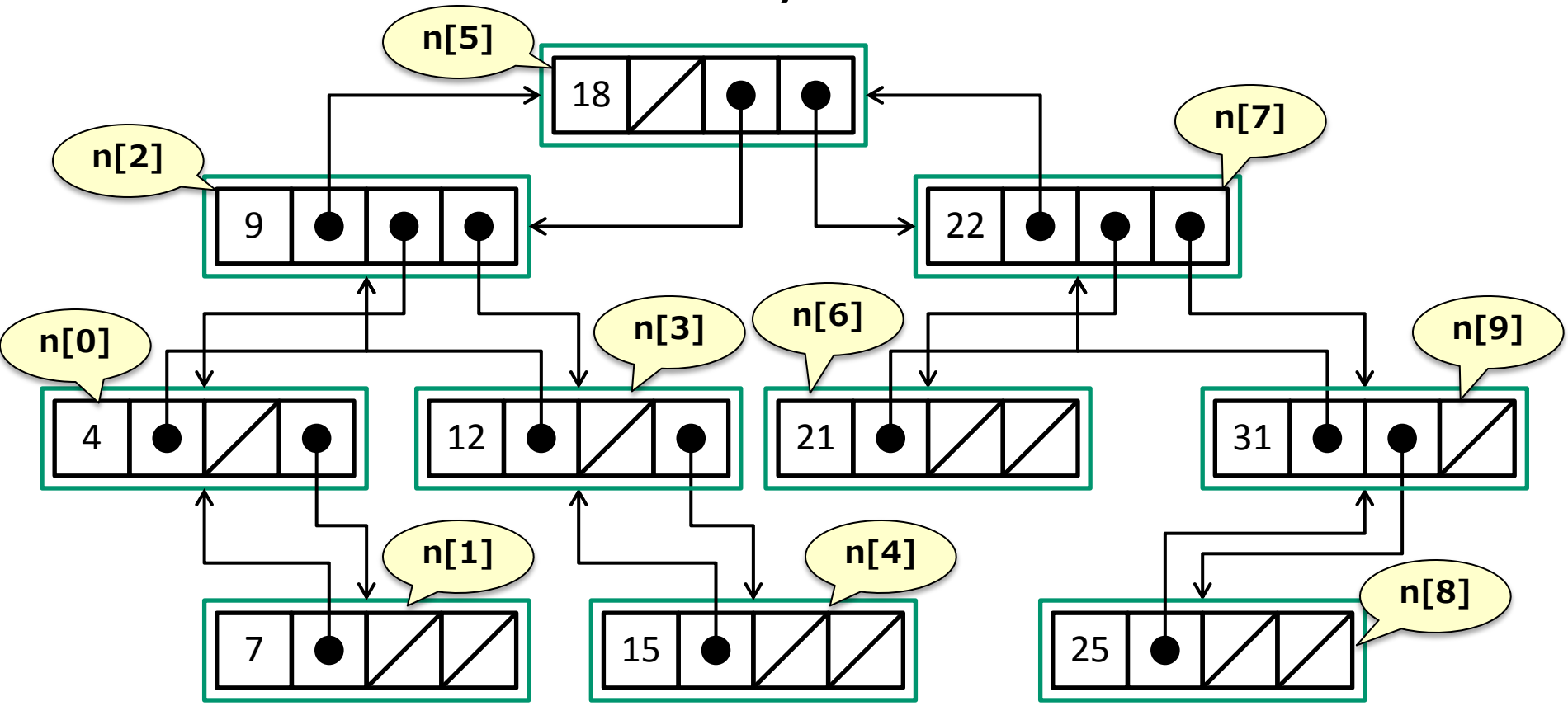
要素の値

```
struct node {
   int key;
   struct node *parent, *left, *right;
};
```

- 機能:
  1. 構造体node 型の領域を**動的メモリ確保**
  2. 要素の値（key）を代入
  3. 親・子を指すポインタはすべて**NULL で初期化**
- 引数key: 要素の値(キー)
- 戻り値: 作成した構造体node 型の要素へのポインタ

インデックスと点の対応

- ２分木を構造体nodeの**ポインタ配列**を使って表現
  - struct node　*n[10];
- 配列のインデックスはkeyの値が小さい順に割り当て



スケルトンコードの説明

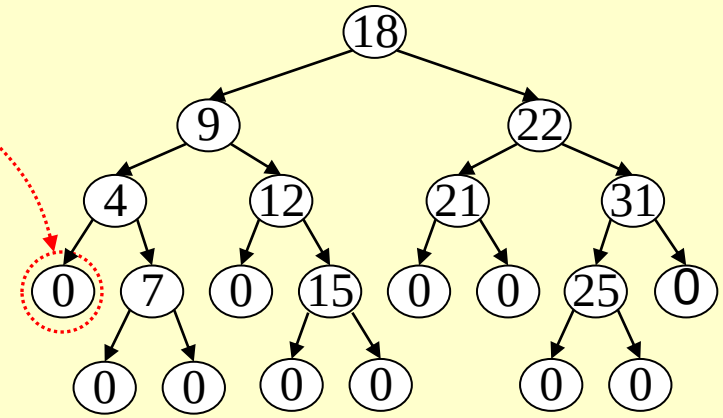
```
main()
{
        struct node *root, *n[10];
        n[0] = new_node(4);
        /* この間に適切なコードを書く*/
        n[5] = new_node(18);
        /* この間に適切なコードを書く*/

        root = n[5];
        n[0]->left = new_node(0);
        n[0]->right = n[1];
        n[0]->parent = n[2];
        /* この間に適切なコードを書く*/
        n[5]->left = n[2];
        n[5]->right = n[7];
        n[5]->parent = NULL;
        /* この間に適切なコードを書く*/
        print_tree(root);
        free_tree(root);
}
```

ノードの値が小さい順に
new_nodeを利用し
ノードを作成し、配列 n に保存

ノードを繋ぎ
レジュメp30の実行例と同じ
2分探索木を作成する

値０は、
外点



全てのノードのメモリを解放する
プログラムは次ページを参照

free_tree: すべてのnodeのメモリを解放する関数

```
void free_tree(struct node *node) {
        if(node != NULL){
                free_tree(node->left);
                free_tree(node->right);
                free(node);
        }
}
```

著者リスト

1. 安積 卓也（情報システム学科）
2. 泉　朋子 （情報コミュニケーション学科）