担当：長尾 智晴 (nagao@ynu.ac.jp)
濱上 知樹 (hamagami@ynu.ac.jp)

「アルゴリズムとデータ構造」講義日程

~~1. 基本的データ型~~
~~2. 基本的制御構造~~
~~3. 変数のスコープルール．関数~~
~~4. 配列を扱うアルゴリズムの基礎(1)．最大値，最小値~~
~~5. 配列を扱うアルゴリズムの基礎(2)．重複除去，集合演算，ポインタ~~
~~6. ファイルの扱い~~
~~7. 整列(1)．単純挿入整列・単純選択整列・単純交換整列~~
~~8. 整列(2)．マージ整列・クイック整列~~
~~9. 再帰的アルゴリズムの基礎．再帰におけるスコープ．ハノイの塔など．~~
~~10. バックトラックアルゴリズム．8 王妃問題など．~~
~~11. 線形リストを扱うアルゴリズム~~
12. 木構造を扱うアルゴリズム(1) 基礎 ← 本日の内容
13. 木構造を扱うアルゴリズム(2) 挿入，削除，バランスなど．
14. ハッシング
15. その他のアルゴリズム

第 12 回「動的データ構造 ～木構造 1～ 」

☆ 線形リストより複雑な構造

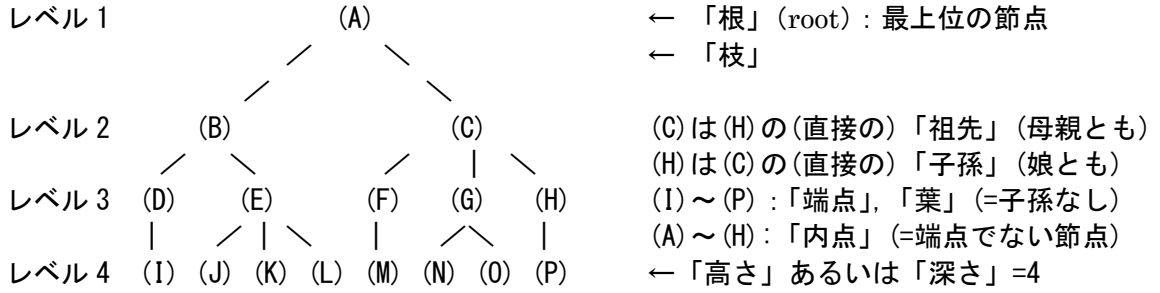
・算術式の構造
・さまざまな項目の分類（の構造）
　（図書の分類，生物学上の分類，計算機のディレクトリ構造）

☆ 木構造（tree）

◎ 定義

・空構造（「空木」）は木構造である．
・一つの項目=節点と有限個の木構造（部分木という）からなる構造も木構造である．
　※ リストは，「各々の節点が高々一つの部分木しか持たない木構造」＝「縮退した木」

◎ グラフ表現



← 「根」(root)：最上位の節点
← 「枝」

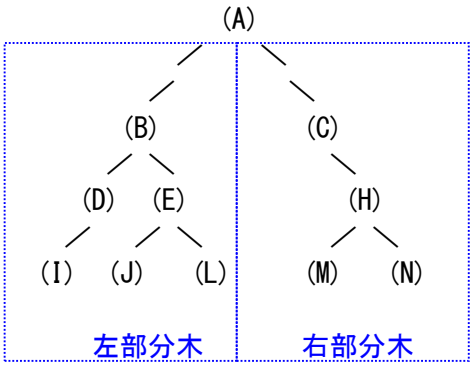
(C)は(H)の(直接の)「祖先」(母親とも)
(H)は(C)の(直接の)「子孫」(娘とも)
(I)～(P)：「端点」，「葉」(=子孫なし)
(A)～(H)：「内点」(=端点でない節点)
←「高さ」あるいは「深さ」=4

「位数」　：内点の（直接の）子孫の数
「木の位数」　：位数の最大値
「順序木」　：各節点から出る枝に順序がついている木

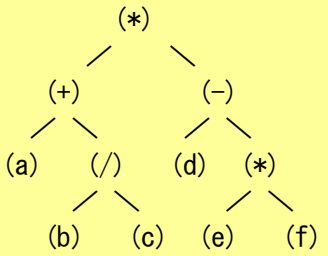
※ グラフ：節点（node）と呼ばれる点の有限集合と，枝（branch）と呼ばれる二つの節点を結ぶ線の有限集合，並びにその結び方によって定義される構造

◎ 2 分木 (binary tree)

・位数 2 の順序木



例) (a+b/c)*(d-e*f)の木表現



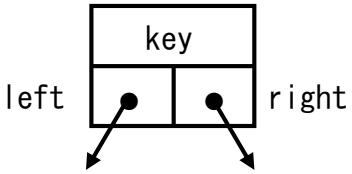
☆ 木のデータ構造

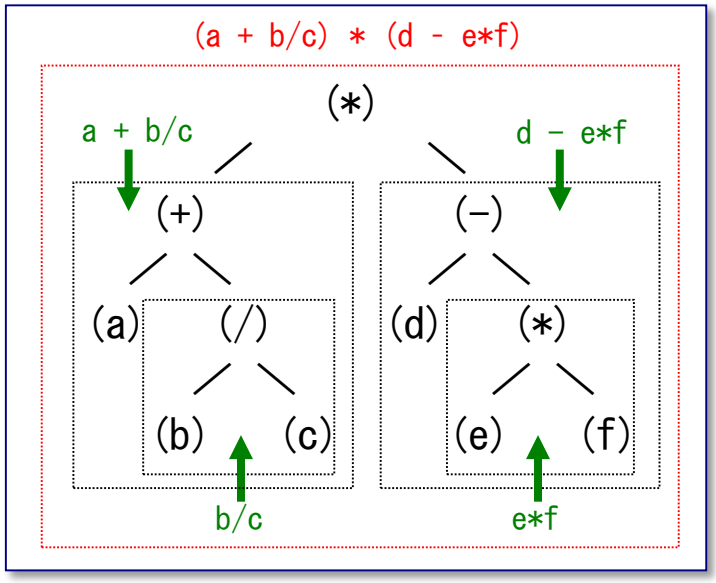
◎ データ構造

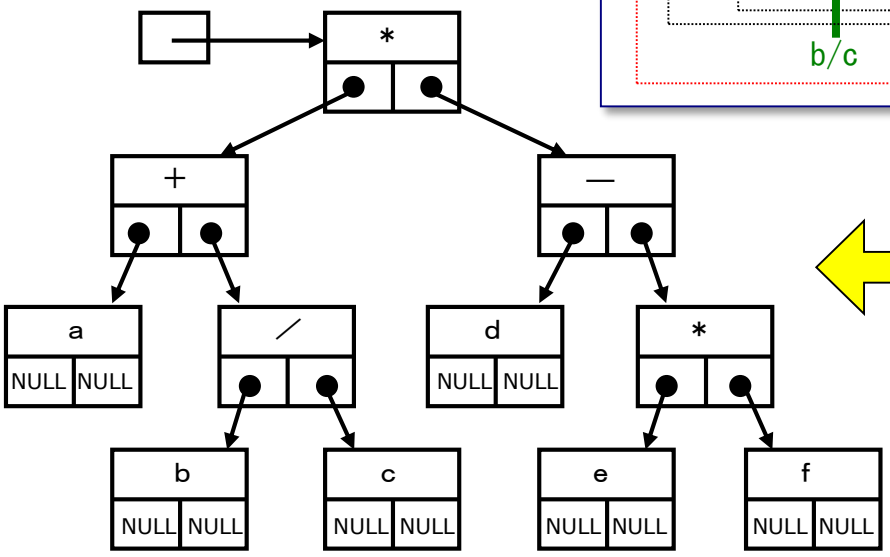
```
struct tree {
    int key;
        …
    struct tree *left;
    struct tree *right;
}
```

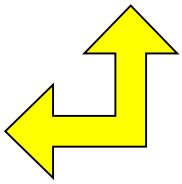
← 識別キー（なくてもよい）
← 一つの節点が保持すべきいくつかのデータ
← 左部分木の根へのポインタ
← 右部分木の根へのポインタ

◎ 図による表記









※ NULL を「空木」とする.

☆ 木の生成（完全バランス木）

◎ 完全にバランス木

・「完全にバランス」
　各々の節点において，左部分木中の節点数と右部分木中の節点数が**高々1 つしか違わない**

・節点が n 個の時
　左部分木の節点数 nl = n/2 (小数以下切捨て)
　右部分木の節点数 nr = n - nl - 1

◎ サンプルプログラム

struct tree *tree(int n)：n 個の節点を持つ完全バランス木を生成

☆ 木の走査（先順，中順，後順）

◎ 木の走査（tree traversal）

・与えられた操作を木の各節点に施す.

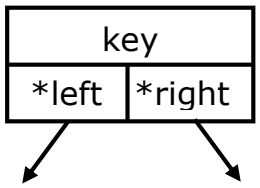
◎ 木構造から自然に生まれる 3 つの基本順序

・**先順**（preorder）　：根（操作）　→ 左部分木　→ 右部分木
・**中順**（inorder）　：左部分木　→ 根（操作）　→ 右部分木
・**後順**（postorder）　：左部分木　→ 右部分木　→ 根（操作）

◎ プログラム

void preorder(struct tree *t)　：ポインタ t で指し示される木を**先順**で走査
void inorder(struct tree *t)　：ポインタ t で指し示される木を**中順**で走査
void postorder(struct tree *t)　：ポインタ t で指し示される木を**後順**で走査

```
1   /*************************************************************
2        アルゴリズムとデータ構造
3        サンプルプログラム treeop1.c
4        <<動的データ構造の例: 完全バランス木の構成ならびに木の走査>>
5        copyright (c)   T.Mori <mori@forest.dnj.ynu.ac.jp>
6   *************************************************************/
7   #include <stdio.h>
8   #include <stdlib.h>
9   /*  木構造のために再帰的データ型の定義  */
10  struct tree {
11       int key;
12       struct tree *left, *right;
13  };
14  struct tree *tree(int n);
15  void print_tree(struct tree *t, int h);
16  void preorder(struct tree *t);
17  void inorder(struct tree *t);
18  void postorder(struct tree *t);
19  void process_node(struct tree *t);
20  #define EOD -1
21  /*  初期データリスト  */
22  int a[ ] =
23  { 8, 9, 11, 15, 19, 20, 21, 7, 3, 2, 1, 5, 6, 4, 13, 14, 10, 12, 17, 16, 18, EOD};
```



関数のプロトタイプ宣言

```
24  int get_data(void);
25  int count_data(void);
26
27  int main(void)
28  {
29      struct tree *root;
30
31      /* 完全バランス木の生成 */
32      root = tree( count_data( ) );
33      /* 木構造の表示 */
34      printf( "木構造:¥n" );
35      print_tree( root, 0 );
36      /* 先順（preorder）の処理の例 */
37      printf( "¥n 先順でのノード処理:¥n" );
38      preorder( root );
39      /* 中順（inorder）の処理の例 */
40      printf("¥n 中順でのノード処理:¥n" );
41      inorder( root );
42      /* 後順（postorder）の処理の例 */
43      printf( "¥n 後順でのノード処理:¥n" );
44      postorder( root );
45      printf( "¥n" );
46      return 0;
47  }
48
49  /* n 個のノードを持つ完全バランス木を生成する
50   * 各ノード内のデータは get_data( )により与えられるものとする */
51  struct tree * tree( int n )
52  {
53      struct tree *newtree;
54      int x, nl, nr;
55
56      if ( n==0 )
57          return( NULL );
58      else {
59          nl = n/2;   nr = n-nl-1;
60          /* 完全バランス木では左右の部分木の節点数の差は高々1 */
61          newtree =(struct tree *)malloc( sizeof( struct tree ) );
62          newtree->key = get_data( );
63          newtree->left = tree( nl );    /* 再帰的に左部分木を生成 */
64          newtree->right = tree( nr );   /* 再帰的に右部分木を生成 */
65          return( newtree );
66      }
67  }
68
69  /*   木を印刷する．h は根（root）から見た木の深さを表す．印刷する順番は，木構
70   * 造について，「右部分木」，「根（そのノード）」，「左部分木」．（中順と異なる）*/
71  void print_tree( struct tree *t, int h )
```

関数のプロトタイプ宣言

データ数(=count_data( ))の節をもつ木を作る

木の表示（枝は表示されない．節だけ．）

処理するデータ数が 0 になったら NULL(=0)を返す．

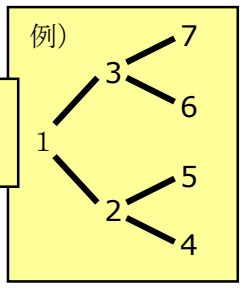
切り捨て割り算

左に nl 個，右に nr 個の木を生成するための再帰呼び出し．

```
72  {
73      int i;
74
75      if ( t != NULL ) {
76          print_tree( t->right, h+1 );   /* 右部分木を印刷 */
77          for( i=0; i<h; i++ )   /* 木の深さの分だけ TAB(8 文字字下げ)を印刷 */
78          printf( "¥t" );
79          printf( "%d¥n", t->key );      /* そのノードのキーを印刷 */
80          print_tree( t->left, h+1 );   /* 左部分木を印刷 */
81      }
82  }
83
84  /* 先順の処理例 */
85  void preorder( struct tree *t )
86  {
87      if ( t != NULL ) {
88          process_node( t );
89          preorder( t->left );
90          preorder( t->right );
91      }
92  }
93
94  /* 中順の処理例 */
95  void inorder( struct tree *t )
96  {
97      if ( t != NULL ) {
98          inorder( t->left );
99          process_node( t );
100         inorder( t->right );
101     }
102 }
103
104 /* 後順の処理例 */
105 void postorder( struct tree *t )
106 {
107     if ( t != NULL ) {
108         postorder( t->left );
109         postorder( t->right );
110         process_node( t );
111     }
112 }
113
114 /* ノードに対する処理．何でも良い．ここでは，キーの値を印刷している．*/
115 void process_node( struct tree *t )
116 {
117     printf( "<%d> ", t->key );
118 }
119
120 /* データ数カウント */
```

再帰呼び出し

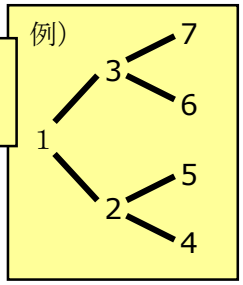
preorder：
1→2→4→5→3→6→7



もし t が NULL(=0)でなければ

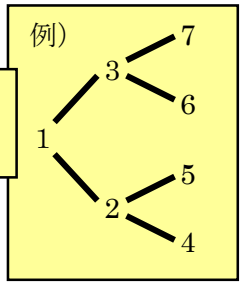
根(操作)→ 左部分木 → 右部分木の順.

inorder：
4→2→5→1→6→3→7



左部分木 → 根(操作)→ 右部分木の順.

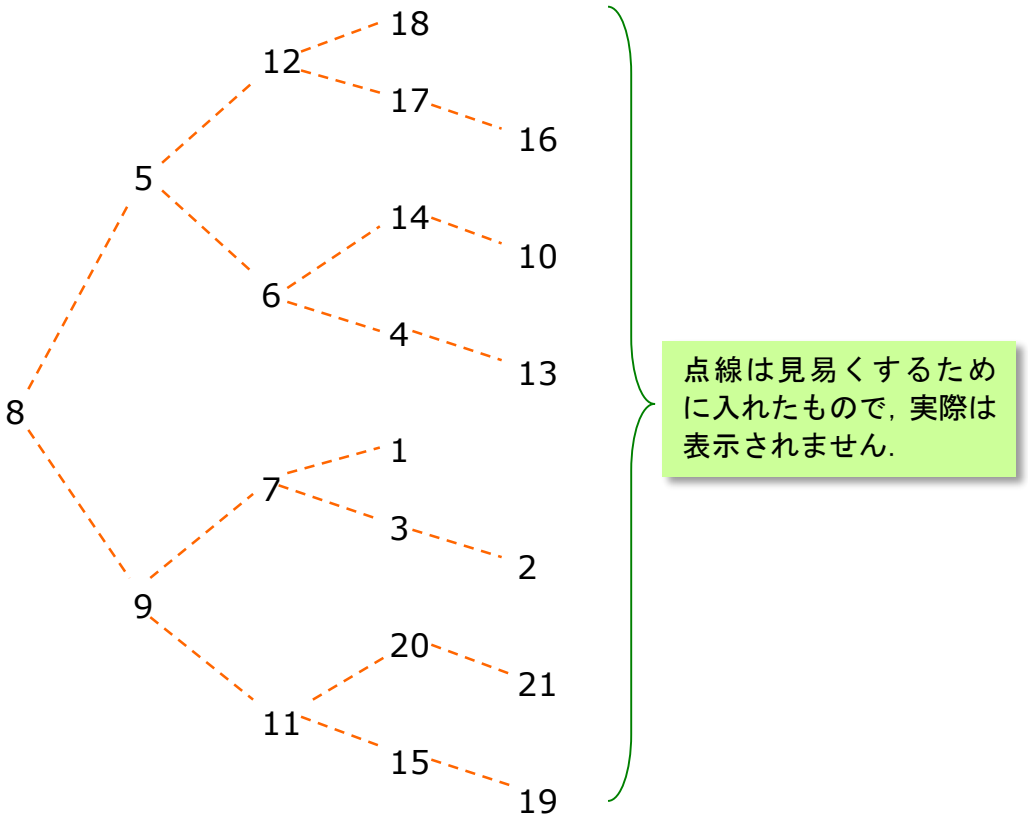
postorder：
4→5→2→6→7→3→1



左部分木 → 右部分木 → 根(操作)の順.

< key の値 > と表示される.

```
121  int count_data( void )
122  {
123      int i = 0;
124
125      while( a[i] != EOD )
126          i++;
127      return i;
128  }
129
130  /* データ取得 */
131  int get_data( void )
132  {
133      static int i=0;
134      return a[i++];
135  }
136
137  【実行結果】(木構造の印字は，時計方向に 90 度回転させて見る)
138  木構造:
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161  先順でのノード処理:
162  <8> <9> <11> <15> <19> <20> <21> <7> <3> <2> <1> <5> <6>
163  <4> <13> <14> <10> <12> <17> <16> <18>
164  中順でのノード処理:
165  <19> <15> <11> <21> <20> <9> <2> <3> <7> <1> <8> <13> <4>
166  <6> <10> <14> <5> <16> <17> <12> <18>
167  後順でのノード処理:
168  <19> <15> <21> <20> <11> <2> <3> <1> <7> <9> <13> <4> <10>
169  <14> <6> <16> <17> <18> <12> <5> <8>
```



Q & A コーナー

Q. 「アルゴリズムとデータ構造」と「プログラミング」はどう違うのでしょうか？

A. **「アルゴリズムとデータ構造」**は，コンピュータを使って物事を処理・解決するための**定式化（データの表現方法）・処理手順・解法**などを表し，**「プログラミング」**とは，それを特定の計算機の言語処理系で実現するための**プログラムを作る作業**を指します．前者はそれを実現する計算機・言語処理系などには依存しません．また後者には C 言語だけでなく Fortran，Pascal，Lisp，Prolog など様々なプログラミング言語があります．

Q. C 言語のプログラミングが上達するためにはどうしたらよいのでしょう？

A. これは皆さんから頻繁に受ける質問です．次の手順に従って学習を進めるとよいでしょう．ただし，括弧内は本学科が関係している科目名です．

1. **基礎的な文法を理解する**（「情報リテラシー」，「プログラミング入門」など）．
2. **基礎的なアルゴリズムを理解する**（「アルゴリズムとデータ構造」など）．
3. **より大きな（実用的な）プログラムを自作してみる**（「プログラミング演習」など）

■ 上達のコツを格言で言うと，**「習うより慣れよ」「好きこそものの上手なれ」**です．参考書を眺めているだけでは上達しません．**とにかく自分でプログラミングをしてみましょう**．また，できるだけ**『自分が本当に作りたいと思うプログラムを作りましょう』**．退屈な例題のプログラムを作るだけでなく，自分が本当に興味を持てるもの，例えばコンピュータグラフィックスの画像生成，画像処理・認識，画像の符号化，音声認識，テキストの暗号化，株価変動予測，ゲーム，情報検索などの面白いテーマについて自分でプログラミングしてみると良いでしょう．

■ 自分で何か作ってみたいものはあるが市販されている本などには参考になるような例題がなく，具体的にどのようにしたら良いか分からない場合は**当学科の情報系の先生方に直接尋ねてみると良い**でしょう．多かれ少なかれ，何らかの助言をもらえることと思います．

Q. 講義を受けてもあまりモチベーションが上がりません．どうしたら良いのでしょう？

A. これも良く受ける質問です．いま学んでいることが将来，どのような研究や仕事に役立つのかを知るためには，**研究室見学**をお薦めします．電子情報工学科の研究室であればどの研究室でも大丈夫ですから，教員に直接メールを出して研究室見学を希望して下さい．きっと対応してくれるはずです．夏休み期間はゆっくりと見学できる絶好の機会です．皆さん，お誘い合わせの上，研究室見学へ Go!

「アルゴリズムとデータ構造」の座学による講義はあと 2 回だけですが，期末試験に出席することをお忘れなく．期末試験までに配布資料や演習を復習しておきましょう．ではまた来週！

MEMO