ポインタによる木の実現

木の各点に対して, *element*, *parent*, *lm_child*, *r_sibling* の 4 つの欄を持つセルを定義し, このようなセルへのポインタを NODE 型とし, 木(TREE)も NODE 型とする.

```
typedef  CELL↑   NODE, TREE ;

typedef  struct {
                    elementtype     element ;
                    NODE            parent ;
                    NODE            lm_child ;
                    NODE            r_sibling ;
          } CELL ;
```

こうしておくと, 木の各操作は次のように実現できる.

```
void  Makenull( T )
        入力 : T :    TREE (名前呼び)
        出力 : T :    TREE
   /* ここでは, 木 T には点が一つも無く, 新たに空の木を作る場合を想定している */
   /* 既に点を持つ木 T を空の木にするには, 木を探索しながら各点を free する */
   /* という操作が必要となる. 3.3 節の木の探索を参考にせよ */
{
        T := NULL ;
} /* Makenull */
```

この手続き *Makenull* のパラメタ *T* は名前呼びであるので, C 言語では *T* へのポインタを渡さねばならない. すなわち, 呼び出す側では,

```
        Makenull( &T ) ;
```

とし, *Makenull* の C 言語プログラムでは

```
        void  Makenull( TREE＊  ptr_T )
        {
                ＊ptr_T := NULL ;
        } /* Makenull */
```

としておかねばならない.

単にこの部分だけを見ていると, *Makenull* を手続きにする必要がなく, *Makenull* の呼び出し側で,

```
        T := NULL ;
```

としておけばよいように見える. しかし, このようにしてしまうと, 木のデータ構造を変えたとき, 呼び

出し側のプログラムも変更する必要が生じる.（ただし, C 言語では *Makenull* を用いても, 呼び出し側の手続き変更しなければならない.）

```
NODE  Root( T )
        入力 :  T  :     TREE （値呼び）
        出力 :  Root  :  NODE
{
        return   T ;
} /*  Root  */
```

```
NODE  Parent( v, T )
        入力 :  v  :     NODE （値呼び）
                T  :     TREE （値呼び）
        出力 :  Parent :  NODE
{
        if( v ≠ NULL )     return   (↑v).parent ;
} /*  Parent  */
```

```
NODE  Leftmost_Child( v, T )
        入力 :  v  :     NODE （値呼び）
                T  :     TREE （値呼び）
        出力 :  Leftmost_Child  : NODE
{
        if( v ≠ NULL )     return   (↑v).lm_child ;
} /*  Leftmost_Child  */
```

```
NODE  Right_Sibling( v, T )
        入力 :  v  :     NODE （値呼び）
                T  :     TREE （値呼び）
        出力 :  Leftmost_Child  : NODE
{
        if( v ≠ NULL )     return   (↑v).r_sibling ;
} /*  Right_Sibling  */
```

```
elementtype  Retrieve( v, T )
        入力 :  v  :     NODE (値呼び)
                T  :     TREE (値呼び)
        出力 :  Retrieve  :        elementtype
    /* ここでは, elementtype の要素を関数の戻し値とできるものとしている */
{
        if( v ≠ NULL )     return    (↑v).element ;
} /* Retrieve */
```

```
void  Assign( x, v, T )
        入力 :  x  :     elementtype (値呼び)
                v  :     NODE (値呼び)
                T  :     TREE (値呼び)
        出力 :  v  :     NODE
                T  :     TREE
{
        if( v ≠ NULL )     (↑v).element := x ;
} /* Assign */
```

```
NODE  Create_2( x, T1, T2 )
        入力 :  x  :     elementtype (値呼び)
                T1, T2  :        NODE (値呼び)
        出力 :  Create_2  :      NODE
{
        v := new( CELL ) ;
        (↑v).element := x ;
        (↑v).parent := NULL ;
        (↑v).lm_child := T1 ;
        (↑v).r_sibling := T2 ;
        return  v ;
} /* Create_2 */
```

*Create*_0(*x*) や *Create_k*(*x*, *T*1, *T*2,⋯, *Tk*) は, *Create*_2 を参考にして容易に作成できるであろう.