カーソルによる木の実現

```
typedef     int     NODE, TREE ;

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

CELL    CellSpace[1..maxlength] ;
                    /*  CellSpace は各要素が CELL 型の配列. 大域的変数  */

NODE    available ;         /*  CellSpace の空き領域を示す大域的変数*/
```

```
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  :=  0 ;
} /*  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  :=  0 ;
        } /*  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  :=   0 ;
```

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( 1 ≦ v ≦ maxlength )     return   CellSpace[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( 1 ≦ v ≦ maxlength )     return   CellSpace[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( 1 ≦ v ≦ maxlength )     return   CellSpace[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( 1 ≦ v ≦ maxlength )     return   CellSpace[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( 1 ≦ v ≦ maxlength )     return   CellSpace[v].element := x ;
} /* Assign */
```

```
NODE  Create_0( x )
        入力 ： x  :     elementtype （値呼び）
        出力 ： Create_0  :        NODE
{
        if( available = 0 )       利用可能なセルが無いので, エラー表示を行う ;
        else {
                r := available ;
                available := CellSpace[r].r_sibling ;
                CellSpace[r].element := x ;
                CellSpace[r].parent := 0 ;
                CellSpace[r].lm_child := 0 ;
                CellSpace[r].r_sibling := 0 ;
                return   r ;
} /* Create */
```

*Create*_0(*x*) は, *Create*_1(*x, T*) があれば, 不要かもしれない.