双方向リスト（ヘッダセルを持った巡回リスト）による LIST の実現

```
typedef  struct {
                D_CELL↑         previous ;
                elementtype     element ;
                D_CELL↑         next ;
                        /*  previous, next  は  D_CELL  型データへのポインタ  */
        } D_CELL ;

typedef     D_CELL↑     LIST ;      /*  LIST  は  D_CELL  型データへのポインタ  */

typedef     D_CELL↑     position ;  /*  position  は  D_CELL  型データへのポインタ  */
```

連結リストの場合と違い, 以下のプログラムにおけるパラメタのリスト *L* は D_CELL へのポインタであるので, 値呼びと書かれている場合には, C 言語でも単に *L* を渡すだけでよい. 従って, *Tail* は

```
position  Tail(  LIST  L  )
{
        return  (＊L).previous ;
}
```

となる. しかし, 名前呼びの場合には, ポインタを渡さねばならない. 従って,

```
position   Makenull(  LIST＊  ptr_L  )
{
        ＊ptr_L  :=  (D_CELL＊) malloc( sizeof( D_CELL ) ) ;
        (＊(＊ptr_L)).previous  :=  ＊ptr_L ;
        (＊(＊ptr_L)).next  :=  ＊ptr_L ;
        return  ＊ptr_L ;
} /*  Makenull  */
```

とし, これを呼び出す側では,

```
p :=  Makenull(  &L  ) ;
```

としなければならない.

```
position  Tail(  L  )
        入力 :  L  :     LIST (値呼び)
        出力 :  Tail  :  position
{
        return  (↑L).previous ;
} /*  Tail  */
```

```
position  Makenull( L )
        入力 ： L  :     LIST（名前呼び）
        出力 ： Tail  :  position（整数）
{
        L := new( D_CELL ) ;
        (↑L).previous := L ;
        (↑L).next := L ;
        return  L ;
} /*  Makenull  */
```

```
void  Insert( x, p, L )
        入力 ： x  :     elementtype
                p  :     position（値呼び）
                L  :     LIST（値呼び）
        出力 ： L から辿ることができる双方向リスト
position  q ;                   /*  ポインタデータの退避場所  */
{
        if ( p ≠ NULL ) {
                q := (↑p).next ;
                (↑p).next := new( D_CELL ) ;
                (↑(↑p).next).previous := p ;
                (↑(↑p).next).element := x ;
                (↑(↑p).next).next := q ;
                (↑q).previous := (↑p).next ;
        }
        else   エラーを出力する ;
} /*  Insert  */
```

```
void  Delete( p, L )
        入力 ： p  :     position（値呼び）
                L  :     LIST（値呼び）
        出力 ： L から辿ることができる双方向リスト
position  q ;                   /*  ポインタデータの退避場所  */
{
        if ( p ≠ NULL ) {
                if ( (↑p).next ≠ NULL and (↑p).next ≠ L ) {
                        q := (↑p).next ;
                        (↑p).next := (↑q).next ;
                        (↑(↑q).next).previous := p ;
                        free( q ) ;
                        return ;
        }       }
        エラーを出力する ;
} /*  Delete  */
```

```
position  First( L )
        入力 :  L  :     LIST (値呼び)
        出力 :  First  :  position
{
        return  L ;
} /*  First  */
```

```
position  Next( p, L )
        入力 :  p  :     position (値呼び)
                L  :     LIST (値呼び)
        出力 :  Next  :  position
{
        if ( p ≠ NULL ) {
                if ( (↑p).next ≠ NULL  and  (↑p).next ≠ L ) {
                        return  (↑p).next ;
        }       }
        エラーを出力する ;
} /*  Next  */
```

```
position  Previous( p, L )
        入力 :  p  :     position (値呼び)
                L  :     LIST (値呼び)
        出力 :  Previous  :  position
{
        if ( p ≠ NULL  and  p ≠ L )    return  (↑p).previous ;
        else    エラーを出力する ;
} /*  Previous  */
```

```
position  Locate( x, L )
        入力 :  x  :     elementtype
                L  :     LIST (値呼び)
        出力 :  Locate  :  position
/*  ここでは, elementtype の二つの要素 x, y が同一か否かを判定する関数 Same( x, y )  */
/*  が存在するものとし, それを用いている  */
position  q ;                   /*  要素を順に調べるためのポインタ. 局所的変数  */
{
        q  :=  L ;
        while (  (↑q).next ≠ L  ) {
                if (  Same( x, Retrieve( q, L ) ) = true  )      return  q ;
                q  :=  (↑q).next ;
        }
        return  (↑L).previous ;
} /*  Locate  */
```

```
elementtype  Retrieve( p, L )
        入力 :  p  :     position (値呼び)
                L  :     LIST (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 ( (↑p).next ≠ L )     return (↑(↑p).next).element ;
        else      エラーを出力する ;
} /* Retrieve */
```

```
void  PrintList( L )
        入力 :  L  :     LIST (値呼び)
        出力 :  L の要素を順に印刷したもの
/* ここでは, elementtype の要素 x を出力する関数 PrintElement( x ) が存在する */
/* ものとし, それを用いている */
position  q ;                   /* 要素を順に調べるためのポインタ. 局所的変数 */
{
        q := L ;
        while ( (↑q).next ≠ L ) {
                PrintElement ( Retrieve( q, L ) ) ;
                q := (↑q).next ;
        }
} /* PrintList */
```