LIST の配列による実現.

```
typedef  struct {
                elementtype     elements[1..maxlength] ;
                int             last ;
        } LIST ;
```

以下のプログラムにおいて, パラメタのリスト *L* は, C 言語では *L* へのポインタを渡すことになる. 例えば, *Tail* は C 言語では

```
position  Tail(  LIST＊  ptr_L  )
{
        return  (＊ptr_L).last + 1 ;
}
```

と書き, これを呼び出す側では,

```
i := Tail(  &L  ) ;
```

とする. このことは 1.4 節にも書いたので, 今後, この説明は省略する.

```
position  Tail(  L  )
        入力 ： L  :    LIST (名前呼び)
        出力 ： Tail  :  position (整数)
{
        return  L.last＋1 ;
} /*  Tail  */
```

```
position  Makenull(  L  )
        入力 ： L  :    LIST (名前呼び)
        出力 ： Tail  :  position (整数)
{
        L.last  :=  0 ;
        return  1 ;
} /*  Makenull  */
```

```
void  Insert(  x, p, L  )
        入力 ： x  :     elementtype
                p  :     position (値呼び)
                L  :     LIST (名前呼び)
        出力 ： L  :     LIST
position  q ;                /*  要素を移動させるためのカウンタ(整数). 局所的変数  */
{
        if (  1 ≦ p ≦ L.last  ＋1  ) {
                L.last  :=  L.last  ＋1 ;
                q  :=  L.last ;
                while (  p < q  ) {
                        L.elements[q]  :=  L.elements[q－1] ;
                        q  :=  q－1 ;
                }
                L.elements[p]  :=  x ;
        }
        else     エラーを出力する ;
} /*  Insert  */
```

```
void  Delete(  p, L  )
        入力 ： p  :     position (値呼び)
                L  :     LIST (名前呼び)
        出力 ： L  :     LIST
position  q ;                /*  要素を移動させるためのカウンタ(整数). 局所的変数  */
{
        if (  1 ≦ p ≦ L.last  ) {
                q  :=  p ;
                while (  q < L.last  ) {
                        L.elements[q]  :=  L.elements[q＋1] ;
                        q  :=  q＋1 ;
                }
                L.last  :=  L.last－1 ;
        }
        else     エラーを出力する ;
} /*  Delete  */
```

```
position  First(  L  )
        入力 ： L  :     LIST (名前呼び)
        出力 ： First  :  position (整数)
{
        return  1 ;
} /*  First  */
```

```
position  Next( p, L )
        入力 ： p  :     position (値呼び)
                L  :     LIST (名前呼び)
        出力 ： Next  :  position (整数)
{
        if ( 1 ≦ p ≦ L.last )     return p＋1 ;
        else      エラーを出力する ;
} /*  Next  */
```

```
position  Previous( p, L )
        入力 ： p  :     position (値呼び)
                L  :     LIST (名前呼び)
        出力 ： Previous  :  position (整数)
{
        if ( 2 ≦ p ≦ L.last＋1 )     return p－1 ;
        else      エラーを出力する ;
} /*  Previous  */
```

```
position  Locate( x, L )
        入力 ： x  :     elementtype
                L  :     LIST (名前呼び)
        出力 ： Locate  :  position (整数)
/*  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カウンタ (整数) . 局所的変数  */
{
        q  :=  1 ;
        while ( q ≦ L.last ) {
                if ( Same( x, Retrieve( q, L ) ) = true )     return q ;
                q  :=  q＋1 ;
        }
        return  q ;
} /*  Locate  */
```

```
elementtype  Retrieve( p, L )
        入力 ： p  :     position
                L  :     LIST (名前呼び)
        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 ≦ p ≦ L.last )     return L.elements[p] ;
        else      エラーを出力する ;
} /*  Locate  */
```

```
void  PrintList( L )
        入力 ： L  ：      LIST (名前呼び)
        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カウンタ (整数). 局所的変数 */
{
        q := 1 ;
        while ( q ≦ L.last ) {
                PrintElement ( Retrieve( q, L ) ) ;
                q := q+1 ;
        }
} /* PrintList */
```