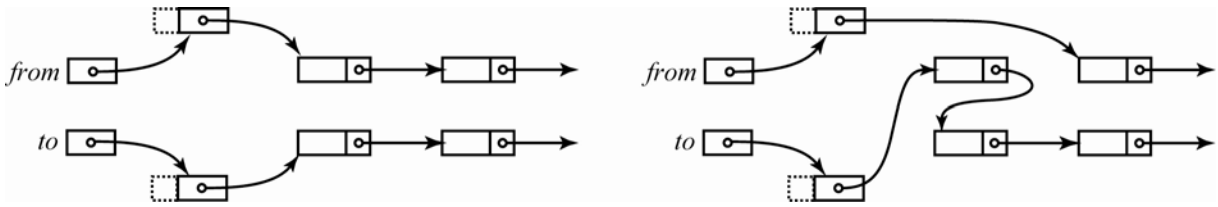
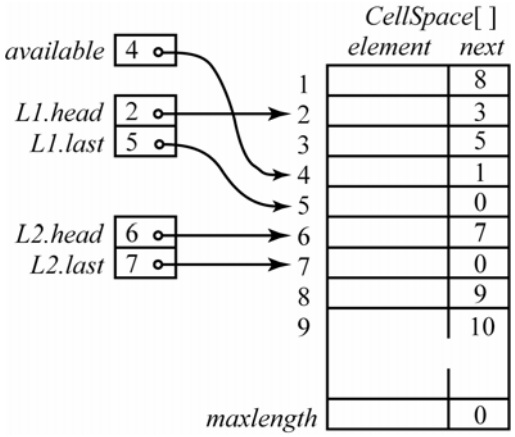
アルゴリズムとデータ構造1　　演習問題 (5)　　回答

カーソルを用いて連結リストを実現するための CCELL 型の配列 *CellSpace*[1..*maxlength*] が, 次のように定義されている.

```
typedef     int             cursor ;            /*  カーソルはポインタの役目をする整数  */
typedef  struct {
            elementtype     element ;           /*  elementtype 型の要素を入れる欄  */
            cursor          next ;              /*  次のセルを指すカーソル  */
        } CCELL ;
typedef  struct {
            cursor          head ;              /*  連結リストの先頭のセルを指すカーソル  */
            cursor          last ;              /*  連結リストの最後のセルを指すカーソル  */
        } CLIST ;

CCELL       CellSpace[1..maxlength] ;   /*  連結リストを蓄えておくための配列  */
cursor      available ;                 /*  使用可能なセルのリストの先頭のセルを指すカーソル  */

/*  CellSpace[・] 内の現在使用されていないセルは, next 欄を用いて連結リストのように覚えられている  */
/*  その連結リストの先頭のセルをカーソル available が示す  */
/*  available が示す連結リストの最後のセルの next 欄には, NULL を意味する 0 が入っている  */
/*  CellSpace および available は大域的変数とする  */
/*  CellSpace[・] 内に作られる CLIST 型の連結リストはヘッダセルを持つものとする  */
/*  連結リスト L の欄 L.last は, 最後のセルを指し, そのセルの next 欄には 0 が入っている  */
/*  連結リスト L が要素を持たないとき, L.head および L.last は共にヘッダセルを指す  */
```




```
void   Move(  cursor↑ from,   cursor↑ to   )
       /*  from が指すカーソルの指すセルを, to が指すカーソルの指すセルの前に移動する  */
       /*  from が指すカーソルが 0 のとき, すなわち移動するセルが無いときは, エラーを出力する  */

{
    cursor    tmp ;                                     /*   tmp はカーソルで局所変数  */

    tmp  :=  ↑from ;
    if (   tmp = 0   ) {
          "移動するセルが無いと出力する"   ;
    }
    else {                                              /*  カーソルを変更する順番に注意  */
          ↑from  :=  CellSpace[tmp].next ;
          CellSpace[tmp].next  :=  ↑to ;
          ↑to  :=  tmp ;
    }
} /*   Move   */
```




```
cursor   MakeNull(   CLIST↑   ptr_L   )
    /*   CellSpace[・] 内に連結リスト L を作成し，L のヘッダセルへのカーソルを返す関数   */
    /*   L.head および L.last は共にヘッダセルを指し, ヘッダセルの next 欄には 0 が入る   */
{
      (↑ptr_L).head  :=  0 ;                    /*  カーソル (↑ptr_L).head が指すセルが無い状態にする  */
      Move(   &available,   &(↑ptr_L).head   ) ;
                                                /*   available の指すセルを (↑ptr_L).head が指すように移動する  */
      (↑ptr_L).last  :=  (↑ptr_L).head ;        /*  (↑ptr_L).last もヘッダセルを指すようにする  */
      return   (↑ptr_L).head ;                  /*  ヘッダセルへのカーソルを返す  */
} /*  MakeNull  */
```

```
void  Concatenate(  CLIST↑   ptr_L1,   CLIST↑   ptr_L2   )
    /*   CellSpace[・] 内の連結リスト L2 を L1 の後ろに繋げる（連接：concatenate：する）．すなわち，    */
    /*   L1 の最後のセルが L2 の先頭の要素が入ったセルを指すようにし，    */
    /*   L1.last が L2 の最後のセルを指すようにする   */
    /*   L2 は無くなるので, L2 のヘッダセルは available のリストに戻す   */
{
      cursor        q ;                         /*  局所変数のカーソル  */

      q  :=  CellSpace[ (↑ptr_L2).head].next ;  /*  q は L2 の先頭の要素が入ったセルを指す  */
      if (   q ≠ 0   ) {                        /*  L2 は空リストではない  */
          CellSpace[(↑ptr_L1).last].next  :=  q ;
          (↑ptr_L1).last  :=  (↑ptr_L2).last ;
      }
      Move(   &(↑ptr_L2).head,   &available   ) ;   /*  L2 のヘッダセルを available のリストに戻す  */
} /*  Concatenate  */
```