問2　アプリケーションで使用するデータ構造とアルゴリズムに関する次の記述を読んで，設問1～4に答えよ。

PCのデスクトップ上の好きな位置に付箋(せん)を配置できるアプリケーションの実行イメージを図1に，付箋1枚のデータイメージを図2に示す。

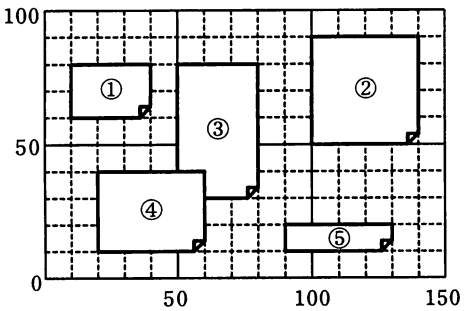

図1　デスクトップ上の付箋のイメージ

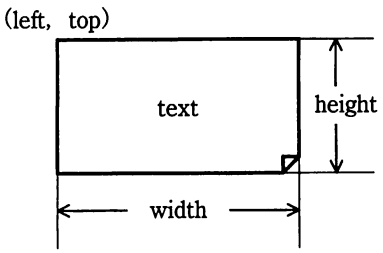

図2　付箋のデータイメージ

複数の付箋データを管理する方法として，配列と双方向リスト（以下，リストという）のいずれがよいかを検討することにした。そこで，図1の付箋③のようにほかの付箋の背後にある付箋を一番手前に移動するアルゴリズムを，配列とリストそれぞれで実装して比較検討する。使用する構造体，配列，定数，変数及び関数を表に示す。また，図1の付箋①～⑤を順番に，配列及びリストにそれぞれ格納した際のイメージを図3，4に示す。

なお，配列及びリストの末尾に近い付箋データほど，デスクトップ上の手前に表示される。

表　使用する構造体，配列，定数，変数及び関数

名称	種類	内容
Memo	構造体	一つの付箋のデータ構造。次の値を管理する構造体である。 id…付箋の一意な ID，text…付箋のメモ内容 left，top…付箋の左端，上端の位置 width，height…付箋の幅，高さ
MEMO_MAX_SIZE	定数	デスクトップに配置できる付箋の最大数。
MemoArray	配列	構造体 Memo を要素（付箋データ）とする，要素数が MEMO_MAX_SIZE の配列。配列の各要素は，MemoArray[i]と表記する（ i は配列の添字）。配列の添字は 0 から始めるものとする。
memoArrayCount	変数	配列 MemoArray に格納されている付箋データの個数。
moveForeArray(id)	関数	付箋 ID が id である付箋データを配列 MemoArray の末尾へ移動する。
findArrayIndex(id)	関数	配列 MemoArray 中で付箋 ID が id である付箋データの添字を返す。
MemoListNode	構造体	付箋データを表すノードのデータ構造。これがリストを構成する。次の値を管理する構造体である。 data…付箋データ（構造体 Memo） prevNode，nextNode…前，次ノードへの参照。リストの先頭ノードの prevNode と末尾ノードの nextNode は null である。
headNode	変数	リストの先頭ノードへの参照。初期値は null である。
tailNode	変数	リストの末尾ノードへの参照。初期値は null である。
moveForeList(id)	関数	付箋 ID が id である付箋データのノードをリストの末尾へ移動する。
findListNode(id)	関数	付箋 ID が id である付箋データが格納されているリスト中のノードへの参照を返す。

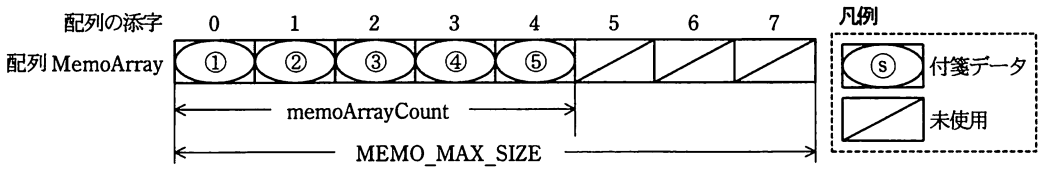

図３　配列の場合のデータ格納例

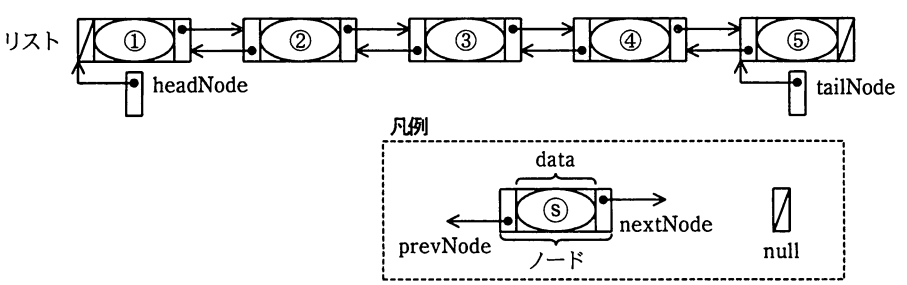

図４　リストの場合のデータ格納例

構造体の要素は“.”を使った表記で表す。“.”の左には，構造体を表す変数又は構造体を参照する変数を書く。“.”の右には，要素の名前を書く。配列の場合，図 3 の付箋⑤のメモ内容は MemoArray[4].text，また，リストの場合，図 4 の付箋②の ID は headNode.nextNode.data.id と表記できる。

〔関数 moveForeArray〕

関数 moveForeArray の処理手順を次の(1)～(4)に，そのプログラムを図 5 に示す。

(1) 配列中の付箋 ID が id である付箋データの添字を取得する。

(2) 配列中の(1)で取得した位置の付箋データを一時変数へ退避する。

(3) 配列中の(1)で取得した位置の次から配列の最後の付箋データがある位置までの付箋データを一つずつ前へずらす。

(4) 配列中の最後の付箋データがあった位置へ(2)で退避した付箋データを代入する。

```
function moveForeArray(id)
  index ← findArrayIndex(id)
  tempMemo ← [ ア ]
  for(i を index+1 から [ イ ] まで 1 ずつ増やす)
    MemoArray[i−1] ← MemoArray[i]
  endfor
  MemoArray[ [ イ ] ] ← tempMemo
endfunction
```

図 5　関数 moveForeArray のプログラム

〔関数 moveForeList〕

関数 moveForeList の処理手順を次の(1)～(4)に，処理手順中の(3)(ii)及び(4)の操作を図 6 に示す。また，関数 moveForeList のプログラムを図 7 に示す。

(1) リストから，付箋 ID が id である付箋データをもつノードへの参照を取得する。

(2) (1)で取得したノード（ノード k）が末尾ノードの場合，処理を終了する。

(3) ノード k が先頭ノードの場合は(i)を，そうでない場合は(ii)を実行する。ここで，ノード k の次ノードをノード k+1，前ノードをノード k−1 と呼ぶ。

(i) リストの先頭ノードへの参照をノード k+1 への参照に変更し，ノード k+1 中の前ノードへの参照を null に変更する。

(ii) ノード k−1 中の次ノードへの参照をノード k+1 への参照に変更し，ノード k+1 中の前ノードへの参照をノード k−1 への参照に変更する。

(4) リストの末尾ノード（ノード n）中の次ノードへの参照をノード k への参照に，ノード k 中の前ノードへの参照をノード n への参照に変更する。ノード k 中の次ノードへの参照を null に，リストの末尾ノードへの参照（tailNode）をノード k への参照に変更する。

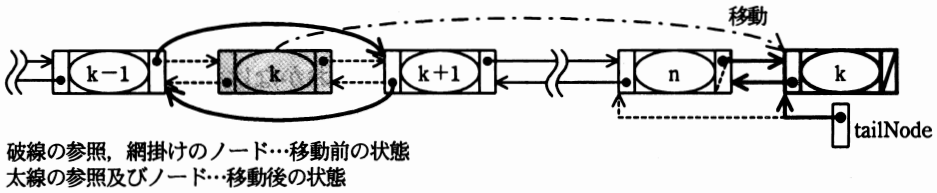

図 6　ノード k をリストの末尾へ移動する操作

```
function moveForeList(id)
  node ← findListNode(id)
  if (node.nextNode と null が等しい)
    // 末尾ノードの場合
    return
  endif
  if (node.prevNode と null が等しい)
    // 先頭ノードの場合
    headNode ← node.nextNode
    node.nextNode.prevNode ← null
  else
    // 先頭ノード以外の場合
    node.prevNode.nextNode ← node.nextNode
    [ ウ ]
  endif
  tailNode.nextNode ← node
  [ エ ]
  node.nextNode ← null
  tailNode ← node
endfunction
```

図 7　関数 moveForeList のプログラム

〔二つのアルゴリズムに関する考察〕

まず，時間計算量について考える。配列の場合，末尾へ移動する要素より後のすべての要素をずらす必要が生じる。この処理の計算量は［　オ　］である。リストの場合，末尾へ移動する付箋データの位置にかかわらず，少数の参照の変更だけでデータ同士の相対的な位置関係を簡単に変えられる。この処理の計算量は［　カ　］である。

次に，必要な領域の大きさについて考える。付箋データ 1 個当たりの領域の必要量は，配列の方が小さい。リストは参照を入れる場所を余分に必要とする。しかし，全体で必要とする領域は，配列の場合，［　キ　］しておかなければならない。リストの場合，配置されている付箋データの個数分だけ領域を確保すればよい。

設問 1　図 1 の付箋①～⑤を格納した図 3 の配列及び図 4 のリストについて，(1)，(2)に答えよ。

(1)　配列に格納されている，付箋①の高さ 20 を求める適切な式を答えよ。

(2)　tailNode.prevNode.data.height の値を答えよ。

設問 2　図 5 中の［　ア　］，［　イ　］に入れる適切な字句を答えよ。

設問 3　図 7 中の［　ウ　］，［　エ　］に入れる適切な字句を答えよ。

設問 4　〔二つのアルゴリズムに関する考察〕について，(1)，(2)に答えよ。

(1)　本文中の［　オ　］，［　カ　］に入れる適切な字句を O 記法で答えよ。なお，配列及びリスト中の付箋データの個数を n とし，関数 findArrayIndex 及び関数 findListNode の計算量は無視する。

(2)　リストの場合，配置されている付箋データの個数分だけ領域を確保すればよいのに対し，配列の場合はどうしなければならないのか。［　キ　］に入れる適切な字句を 30 字以内で答えよ。