1-1 リスト構造

問 1-1-1

次の各設問に答えよ。

設問１　0Sが記憶領域の割当てと解放を繰り返すことによって，こま切れの未使用領域が多数できてしまう場合がある。この現象を何というか。

ア　オーバレイ　　イ　スワッピング
ウ　フラグメンテーション　　エ　ページング

設問２　記憶領域の動的な割当て及び解放を繰り返すことによって，どこからも利用できない記憶領域が発生することがある。このような記憶領域を再び利用可能にする機能はどれか。

ア　スタック　　イ　スラッシング
ウ　ヒープ　　エ　メモリコンパクション

設問３　配列と比較した場合の連結リストの特徴に関する記述として，適切なものはどれか。

ア　要素を更新する場合，ポインタを順番にたどるだけなので，処理時間は短い。
イ　要素を削除する場合，削除した要素から後ろにあるすべての要素を前に移動するので，処理時間は長い。
ウ　要素を参照する場合，ランダムにアクセスできるので，処理時間は短い。
エ　要素を挿入する場合，数個のポインタを書き換えるだけなので，処理時間は短い。

設問４　データ構造の一つであるリストは，配列を用いて実現する場合と，ポインタを用いて実現する場合とがある。配列を用いて実現する場合の特徴はどれか。ここで，配列を用いたリストは，配列に要素を連続して格納することによって構成し，ポインタを用いたリストは，要素から次の要素へポインタで連結することによって構成するものとする。

ア　位置を指定して，任意のデータに直接アクセスすることができる。
イ　並んでいるデータの先頭に任意のデータを効率的に挿入することができる。
ウ　任意のデータの参照は効率的ではないが，削除や挿入の操作を効率的に行える。
エ　任意のデータを別の位置に移動する場合，隣接するデータを移動せずにできる。

設問５　双方向のポインタをもつリスト構造のデータを表に示す。この表において新たな社員Ｇを社員Ａと社員Ｋの間に追加する。追加後の表のポインタａ～ｆの中で追加前と比べて値が変わるポインタだけをすべて列記したものはどれか。

表

アドレス	社員名	次ポインタ	前ポインタ
100	社員Ａ	300	0
200	社員Ｔ	0	300
300	社員Ｋ	200	100

追加後の表

アドレス	社員名	次ポインタ	前ポインタ
100	社員Ａ	a	b
200	社員Ｔ	c	d
300	社員Ｋ	e	f
400	社員Ｇ	x	y

ア　a，b，e，f　　イ　a，e，f
ウ　a，f　　エ　b，e

問 1-1-2

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。

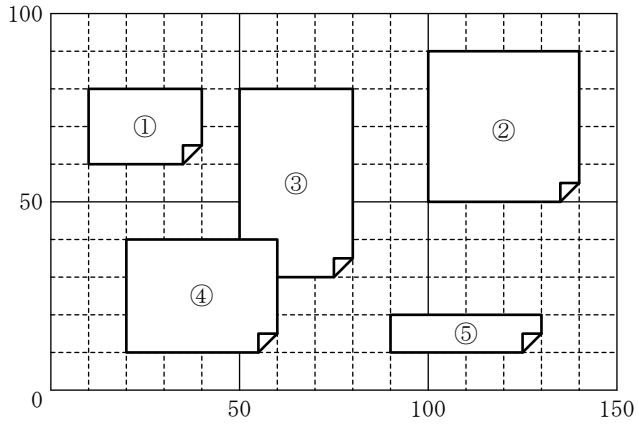


図１　デスクトップ上の付箋のイメージ

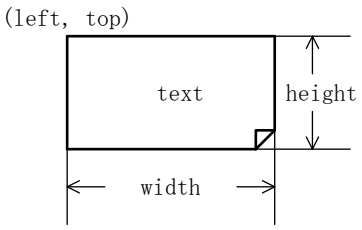


図２　付箋のデータイメージ

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１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１　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。

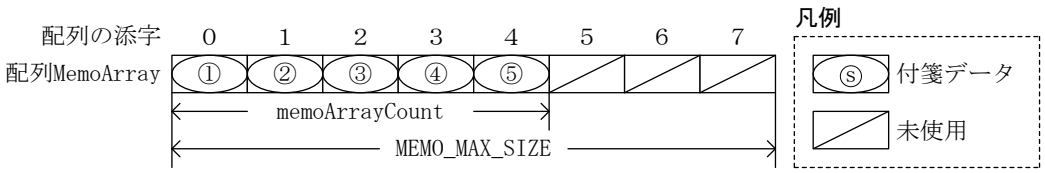


図３　配列の場合のデータ格納例

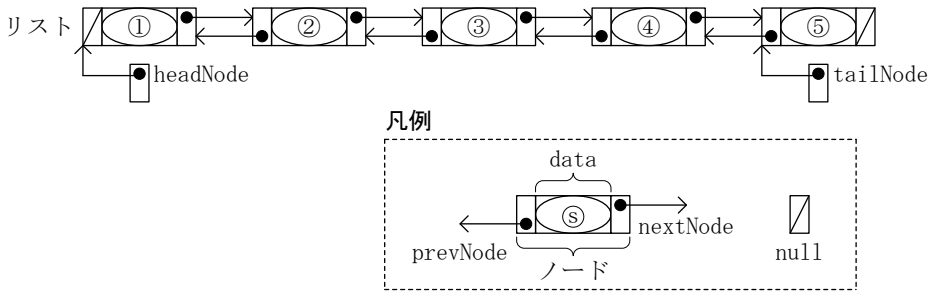


図４　リストの場合のデータ格納例

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。

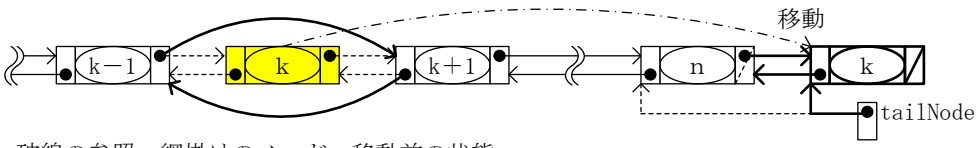
```
function moveForeArray(id)
  index ← findArrayIndex(id)
  tempMemo ← [ a ]
  for(iをindex＋1から [ b ] まで１ずつ増やす)
    MemoArray[i－1] ← MemoArray[i]
  endfor
  MemoArray[ [ b ] ] ← tempMemo
endfunction
```

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。



破線の参照，網掛けのノード…移動前の状態
太線の参照及びノード…移動後の状態

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
    [ c ] ← [ d ]
  endif
  tailNode.nextNode ← node
  [ d ] ← [ e ]
  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［ f ］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［ g ］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，デスクトップに配置できる付箋の最大数分の領域を確保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。

解答欄		
	(1)	
	(2)	

設問２　図５中の [a] ，[b] に入れる適切な字句を答えよ。

解答欄	a		b	

設問３　図７中の [c] ～ [e] に入れる適切な字句を解答群の中から選び，記号で答えよ。

解答群

ア　headNode　　イ　node.nextNode

ウ　node.nextNode.prevNode　　エ　node.prevNode

オ　node.prevNode.nextNode　　カ　tailNode

解答欄	c		d		e		

設問４　〔二つのアルゴリズムに関する考察〕の [f] ，[g] に入れる適切な字句を解答群から選び，記号で答えよ。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。

解答群

ア　$O(1)$　　イ　$O(\log n)$　　ウ　$O(n)$

エ　$O(n^2)$　　オ　$O(n \log n)$　　カ　$O(n!)$

解答欄	f		g	

問 1-1-3 【2013年秋期 応用 問2】

リストによるメモリ管理に関する次の記述を読んで，設問１～３に答えよ。

与えられたメモリ空間（以下，ヒープ領域という）の中に，可変長のメモリブロックを動的に割り当てるためのデータ構造及びアルゴリズムを考える。

ヒープ領域は，一つ以上の連続したメモリブロックで構成する。メモリブロックは，固定長のヘッダ部分と可変長のデータ部分で構成される。ヘッダ部分は構造体で，prev，next，status及びsizeのメンバによって構成される。メモリブロックの構造を図１に，ヘッダ部分のメンバの意味を表１にそれぞれ示す。メモリブロックを指すポインタ変数には，メモリブロックの先頭アドレスをセットする。あるメモリブロックを指すポインタ変数をqとするとき，そのメンバprevの参照は，q->prevと表記する。また，ヘッダ部分のバイト数は，HSIZEとする。

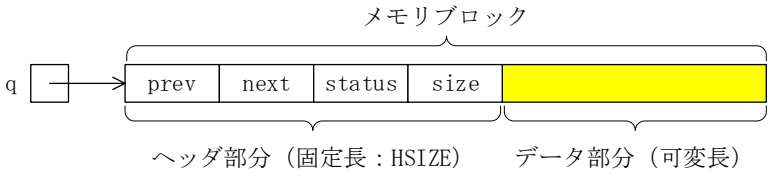


図１ メモリブロックの構造

表１ ヘッダ部分のメンバの意味

メンバ名	メンバの意味
prev	一つ前のメモリブロックの先頭アドレスへのポインタ
next	一つ後のメモリブロックの先頭アドレスへのポインタ
status	'A'：データ部分は割当て済みメモリである。 'F'：データ部分は空きメモリである。
size	データ部分のバイト数

ヘッダ部分と同じ構造体の変数EDGEをヒープ領域の外に定義する。そのメンバprev及びnextには，それぞれヒープ領域の最後尾及び先頭のメモリブロックの先頭アドレスをセットする。ヒープ領域の先頭のメモリブロックのメンバprevと最後尾のメモリブロックのメンバnextには，ともにEDGEの先頭アドレスをセットする。これによって，EDGEを含むメモリブロックが双方向の循環リストを構成する。EDGEにはデータ部分はなく，メンバsizeには０が設定されている。データ構造の全体像を図２に示す。

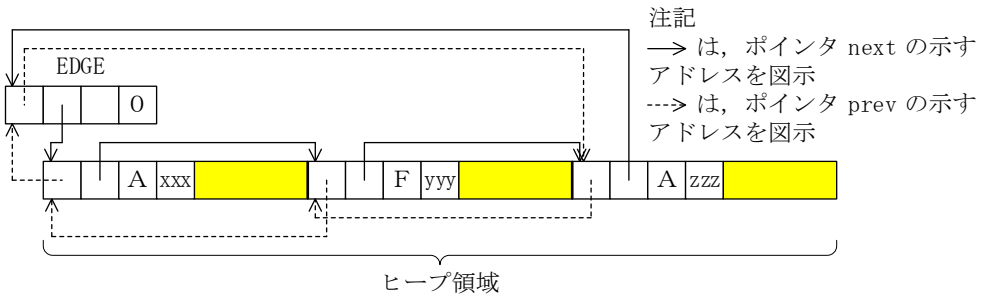


図２　メモリ管理のためのデータ構造

〔メモリ割当ての関数〕

メモリ割当ての関数は，割り当てたいバイト数（msize）を引数とし，そのバイト数以上の大きさのデータ部分をもつメモリブロックを，ヒープ領域から探索する。このアルゴリズムを次のように考えた。

(1) ポインタ変数qを定義し，初期値として変数EDGEのnextの値をセットする。

(2) qが［ア］と等しい場合は，ヒープ領域には十分な空きメモリをもったメモリブロックが無かったことを意味する。関数の戻り値にNULLをセットして終了する。それ以外の場合は，次の(3)～(5)を実行する。

(3) q->［イ］が'A'の場合，又はq->sizeがmsize未満である場合は，qに q->［ウ］をセットして(2)に戻る。

(4) q->sizeがHSIZE＋msize以下の場合は，q->［イ］に'A'をセットし，関数の戻り値にqの値をセットして終了する。

(5) q->sizeがHSIZE＋msizeよりも大きい場合は，そのメモリブロックを割当て済みのメモリブロックと，残りの空きメモリブロックの二つに分割する（図３参照）。ポインタ変数rを定義し，初期値としてq＋HSIZE＋msizeをセットする。q->［イ］に'A'をセットし，r->［イ］に'F'をセットする。r->sizeに q->size-HSIZE-msizeをセットし，q->sizeにmsizeをセットする。r->prevには［エ］を，r->nextには［オ］を，q->next->prevにはrを，q->nextにはrを順にセットする。関数の戻り値にqの値をセットして終了する。

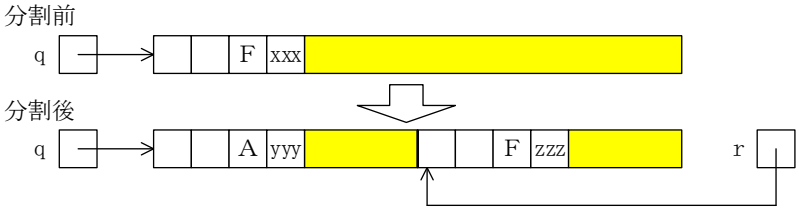


図３　メモリ割当てによるメモリブロックの分割

〔メモリ解放の関数〕

メモリ解放の関数freememは，解放したいメモリブロックの先頭アドレスを引数とし，そのメモリブロックを空きメモリブロックの状態に変更する。このとき，できるだけ大きな連続した空きメモリが後で確保できるよう，その前後のメモリブロックも空きメモリブロックかどうかを確認する。空きメモリブロックが連続する場合には，それらをまとめて一つの空きメモリブロックにする。

関数freememのプログラムを図４に示す。この関数を正しく動作させるためには，変数EDGEのメンバstatusの値は［カ］である必要がある。

```
function freemem( q )  // qは解放したいメモリブロックの先頭アドレス
  p ← q->prev            // qの前のメモリブロック
  r ← q->next            // qの後のメモリブロック
  if ( p->statusが'F'と等しい ) then        // 前が空き
    if ( r->statusが'F'と等しい ) then      // 後も空き
       p->next ← r->next
       p->size ← [キ]
    else                                        // 後が割当て済み
       p->next ← r
       p->size ← p->size + q->size + HSIZE
    endif
    p->next->prev ← [ク]
  else                                          // 前が割当て済み
    if ( r->statusが'F'と等しい ) then      // 後が空き
       q->next ← r->next
       q->size ← [ケ]
       q->next->prev ← q
    endif
    q->status ← 'F'
  endif
endfunction
```

図４　メモリ解放の関数freemem

〔メモリコンパクション〕

メモリの確保や解放の処理を繰り返すと，サイズの小さな空きメモリが分散してしまい，サイズの大きな空きメモリの確保が難しくなることがある。このような現象を［コ］と呼ぶ。このとき，割当て済みのメモリブロックが連続するようにメモリブロックを移動し，移動したメモリブロックの後ろに大きな空きメモリを確保することをメモリコンパクションという（図５参照）。

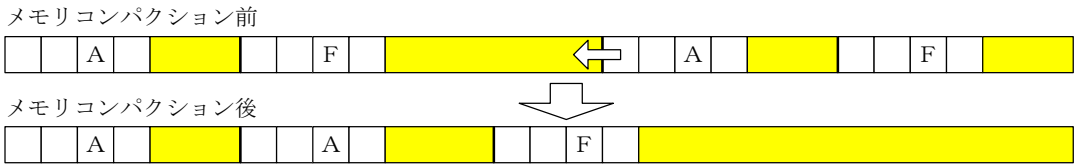


図５　メモリコンパクションのイメージ

ヒープ領域が図６のように左上から右下にかけて連続する構成の場合，メモリコンパクションを実行すると，［サ］バイトの空きメモリができる。

メモリコンパクションを実行すると，①メモリコンパクション前に実行したメモリ割当て関数の戻り値は，メモリ解放の関数の引数としては使えなくなる場合がある。

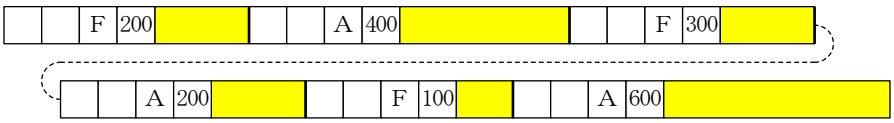


図６　ヒープ領域の状態

設問1　〔メモリ割当ての関数〕について，(1)，(2)に答えよ。

(1) 本文中の［ア］～［ウ］に入れる適切な字句を答えよ。

(2) 本文中の［エ］，［オ］に入れる適切な字句を，ポインタ変数qを用いて答えよ。

解答例	(1)	ア		イ	
		ウ			
	(2)	エ		オ	

設問2　〔メモリ解放の関数〕について，(1)，(2)に答えよ。

(1) 本文中の［カ］に入れる適切な字句を答えよ。

(2) 図4中の［キ］～［ケ］に入れる適切な字句を答えよ。

解答例	(1)	カ	
	(2)	キ	
		ク	
		ケ	

設問3　〔メモリコンパクション〕について，(1)～(3)に答えよ。

(1) 本文中の［コ］に入れる適切な字句をカタカナで答えよ。

(2) 本文中の［サ］に入れる適切な式を答えよ。

(3) 本文中の下線①の理由を25字以内で述べよ。

解答例	(1)	コ	
	(2)	サ	
	(3)		