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。

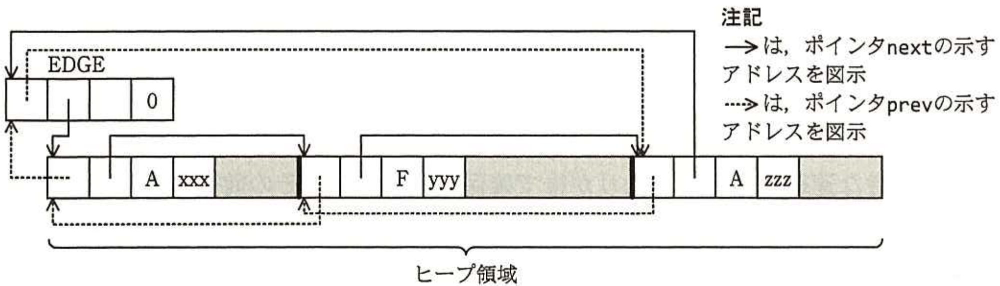

図2　メモリ管理のためのデータ構造

〔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。

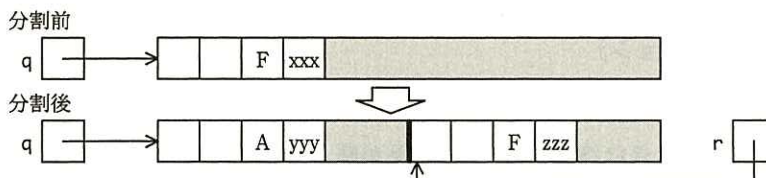

図3　メモリ割当てによるメモリブロックの分割

〔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）。

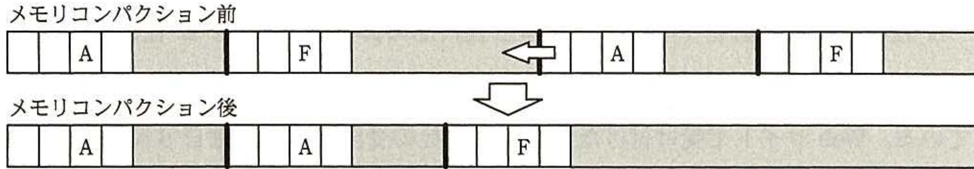

図5　メモリコンパクションのイメージ

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。

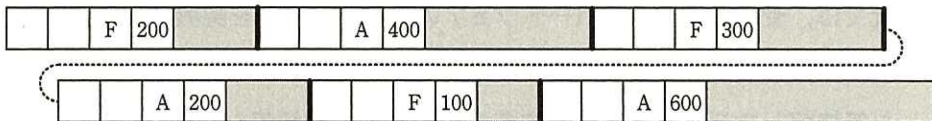

図6　ヒープ領域の状態

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。

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。

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。