参加登録者データベース

この問題において取り扱うべきデータは, 参加者の集合 $Q$ で, $Q$ の各要素 (参加者) には,

名前　*name*　(人名. 同姓同名はない)　および

参加費　$i$　($1 \leqq i \leqq k$ なる整数)

なる情報が付随している.

これらのデータに対して, 実行すべき処理は,

*Makenull*( $Q$ )　参加者の集合 $Q$ を空にする手続き.

*Add*( *name*, $i$, $Q$ )　集合 $Q$ に, 名前が *name* で, 支払った参加費種別が $i$ の人のデータを入れる関数. $Q$ に *name* の人が居ない場合には, データを実際に入れ, 真を返す. 既に入っている場合には, 参加費種別を $i$ と上書きし, 偽を返す.

*Paid*( *name*, $Q$ )　名前が *name* の人が支払った参加費の種別を返す関数. $Q$ に *name* の人が居ない場合には, 0 を返す.

*ListName*( $i$, $Q$ )　集合 $Q$ に入っている人で, 参加費種別 $i$ ($1 \leqq i \leqq k$) の参加費を支払った人全員の名前を印字する手続き.

である. ここでは, *ListName*( $i$, $Q$ ) のパラメタを本に書いた *ListName*( $i$ ) から上記に変更している. $Q$ も入力として必要であるからである.

これらの操作を検討すると, *ListName*( $i$ ) の操作が無ければ, その他の操作は写像の操作と同じであることが分かる. 従って, 写像を覚えるのに適切なハッシュ表に, *ListName*( $i$ ) の操作を効率的に実現できるデータ構造を組合わせることが考えられる. *ListName*( $i$ ) の操作は, 会費種別 $i$ の人を連結リスト構造で覚えておけば, そのリストを辿るだけで良くなる.

そこで, 本の図 4.5 (ページ 80) のデータ構造を参考にし, オープンハッシュに連結リストを組み込むことにしよう. その概念図を次ページに示す.

顧客の名前の型 NAME は, 問 4.4 と同様, 次のようにしておく.

```
typedef struct {
        char   first[0..namelength–1] ;
        char   last[0..namelength–1] ;
} NAME ;
```

そうすると, NAME は 2 つの配列であるから, 集合 $Q$ の要素を覚えるためのセルにおける名前

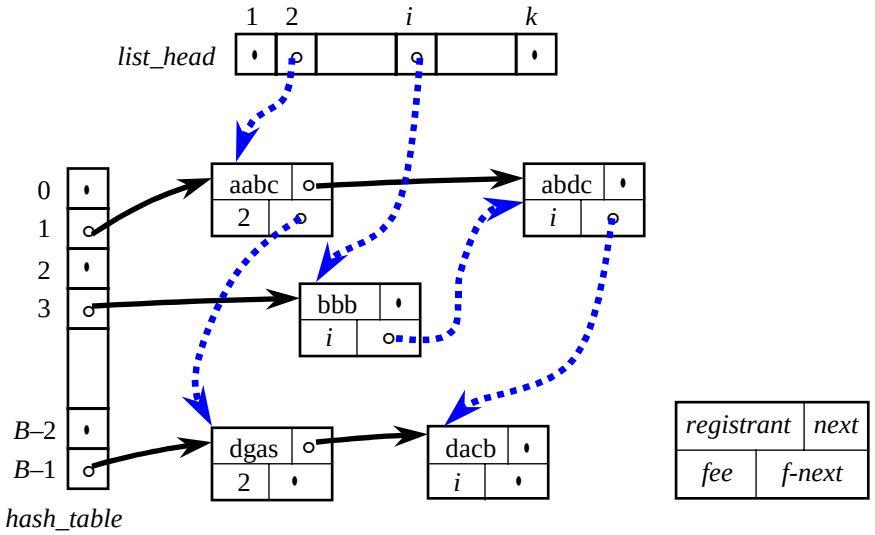
の欄（以下では *registrant* とする）は NAME 型へのポインタとする方が効率的である．
そこで，オープンハッシュの各バケツに入れる $Q$ の要素を覚えるためのセル（CELL41）を次のように定義する．ここで，欄 *next* は，オープンハッシュの一つのバケツの中に入っているセルを繋ぐためのポインタであり，*f_next* は，同じ費用を支払った人を繋ぐ連結リスト用のポインタである．

```
typedef struct {
        NAME↑    registrant  ;          /*  登録者の名前へのポインタ  */
        int      fee  ;                 /*  登録者が支払った費用の種類  */
        CELL41↑  next  ;                /*  オープンハッシュのポインタ  */
        CELL41↑  f_next  ;              /*  連結リストのポインタ  */
} CELL41  ;

typedef struct {
        CELL41↑  list_head[1..k]  ;     /*  連結リストの先頭へのポインタ  */
        CELL41↑  hash_table[0..B–1]  ;  /*  オープンハッシュのハッシュ表  */
} HASH_and_LIST  ;


HASH_and_LIST    Q  ;
```

ここで，$k$ および $B$ は定数である．



この定義に伴って, 顧客の名前を表す変数 *name* も NAME 型へのポインタ(NAME↑)とする.

指定された関数を実現するために, 問 4.4 の場合と同様, *name* で指定される名前を持つ顧客が *Q* の中に存在するか否かを判定する関数 *FindCell41*( *name*, *Q* ) を用意する. この関数は, *Q* の中に *name* で指定される顧客が存在するならば, *name* を持つセルへのポインタ(CELL41↑)を返し, 存在しないならば, **NULL** を返すものとする. この関数では, ハッシュ関数 *h*( *name* ) および 2 つの名前 *name*1, *name*2 が同じか否かを判定する関数 *Same*( *name*1, *name*2 ) が必要となるが, それらはできているものとする.

```
CELL41↑  FindCell41( name, Q )
        入力: name :    NAME 型のデータへのポインタ(値呼び)
              Q :       HASH_and_LIST 型(名前呼び)
        出力: FindCell41 :      CELL41↑ 型(name を持つセルへのポインタ)
    /* name で指定される顧客が集合 Q の中に存在するか否かを判定する */
    /* あれば name を持つセルへのポインタ(CELL41↑)を, なければ NULL を返す */
{
        p := Q.hash_table[ h( name ) ] ;                /* p はセルへのポインタ */
        while ( p≠NULL ) {
              if ( Same( name, (↑p).customer ) )        return  p ;
              p := (↑p).next ;
        }
        return  NULL ;                                  /* name は存在しない */
} /* FindCell41 */
```

```
int  h( name )
    /* name が指す NAME 型の名前に対して, 0≦h(name)≦B–1 なる整数を返す関数 */
    /* ここで B は定数である */
{     } /* h */
```

```
boolean  Same( name1, name2 )
    /* name1 および name2 が指す NAME 型の名前が同じであるか否かを判定する */
    /* 同じであれば true を, さもなくば false を返す */
{     } /* Same */
```

このようにすると, 操作は次のように実現できる.

*Makenull*( *Q* )　　参加者の集合 *Q* を空にする手続き.

*Q.hash_table*[ *j* ] (0≦*j*≦*B*–1) および *Q.list_head*[ *i* ] (1≦*i*≦*k*) に **NULL** を入れる.

*Add*( *name, i, Q* )　集合 *Q* に, 名前が *name* で, 支払った参加費種別が *i* の人のデータを入れる関数. *Q* に *name* の人が居ない場合には, データを実際に入れ, 真を返す. 既に入っている場合には, 参加費種別を *i* と上書きし, 偽を返す.

*FindCell41*( *name, Q* ) を用いて, *Q* に *name* の人が居るか否かを調べる. 居たならば, *FindCell41*( *name, Q* ) が指定するセルの欄 *fee* を *i* に書き換え, false を返す. 居ないならば, 新たなセルを生成し, オープンハッシュのバケツのリストの先頭で, かつ参加費種別が *i* の連結リストの先頭にこのセルを入れる. また, 名前 *name* および参加費種別 *i* をこのセルの対応する欄に入れ, true を返す.

*Paid*( *name, Q* )　名前が *name* の人が支払った参加費の種別を返す関数. *Q* に *name* の人が居ない場合には, 0 を返す.

*FindCell41*( *name, Q* ) を用いて, *Q* に *name* の人が居るか否かを調べる. 居たならば, *FindCell41*( *name, Q* ) が指定するセルの欄 *fee* の値を返す. 居ないならば, 0 を返す.

*ListName*( *i, Q* )　集合 *Q* に入っている人で, 参加費種別 *i* (1≦*i*≦*k*) の参加費を支払った人全員の名前を印字する手続き.

*Q.list_head*[ *i* ] と欄 *f_next* を用いて, 連結リスト内の参加者名を印字する.

詳細は以下のようである.

```
void   Makenull( Q )
       入力: Q : HASH_and_LIST 型 (名前呼び)
       出力: Q : HASH_and_LIST 型 (名前呼び)
       /* Q を空にする */
{
       for ( 0≦j≦B–1 なる各 j )      Q.hash_table[ j ] := NULL ;
       for ( 1≦j≦k なる各 j )        Q.list_head[ j ] := NULL ;
} /* Makenull */
```

```
boolean   Add( name, i, Q )
        入力: name  :    NAME 型のデータへのポインタ(値呼び)
              i  :       1≦i≦k なる整数(値呼び)
              Q  :       HASH_and_LIST 型(名前呼び)
        出力: Add  :     Boolean
              Q  :       HASH_and_LIST 型(名前呼び)
        /* Q に, 名前が name で, 支払った参加費種別が i の人のデータを入れる */
        /* 既に入っている場合には, 参加費種別を i と上書きし, false を返す */
{
        CELL41↑   p ;
        if ( FindCell41( name, Q )≠NULL ) {          /* name は Q に含まれる */
              (↑FindCell41( name, Q )).fee := i ;
              return   false ;
        }
        else {                                  /* name で指定される顧客は Q に含まれない */
              p := new( CELL41 ) ;        /* p は新たに生成したセルを指す */
              (↑p).registrant := name ;
              (↑p).fee := i ;
              (↑p).next := Q.hash_table[ h( name ) ] ;
              (↑p).f_next := Q.list_head[ i ] ;
              Q.hash_table[ h( name ) ] := p ;
              Q.list_head[ i ] := p ;
              return   true ;
        }
} /* Delete */
```

```
int   Paid( name, Q )
        入力: name  :    NAME 型のデータへのポインタ(値呼び)
              Q  :       HASH_and_LIST 型(名前呼び)
        出力: Paid  :    0≦paid≦k なる整数
{
        if ( FindCell( name, Q ) = NULL )    return   0 ;
        else                                    /* name で指定される顧客は Q に含まれる */
              return   (↑FindCell41( name, Q )).fee ;
} /* Paid */
```

```
#include    <stdio.h>
void   ListName(  i,  Q  )
        入力:   i  :        1≦i≦k なる整数 (値呼び)
                Q  :        HASH_and_LIST 型 (名前呼び)
        出力:   名前 (NAME 型) の系列
    /*  参加費 i を支払った人全員の名前を印字する  */
{
        p  :=  Q.list_head[ i ]  ;                          /*  p はセルへのポインタ  */
        while (  p≠NULL  ) {
                (↑p).registrant  が指す  NAME  型のデータを印字  ;
                p  :=  (↑p).f_next  ;
        }
        printf( "end of list" )  ;                          /*  リストの終わりを印字  */
} /*  ListName  */
```