Add(*name*, *Q*), *Deleted*(*Q*), および *Count*(*name*, *Q*) の操作

本の図 4-5 に示されたデータ構造を定義しよう.

まず, 顧客の名前の型 NAME は, 例えば第 1 章で示したように, 次のようにしてみよう.

```
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, 顧客の名前を入れる欄 *customer* の型は NAME 型とするより, NAME 型へのポインタとする方が効率的である. そこで, 次のように定義する.

```
typedef struct {
        NAME↑     customer ;
        int       position ;
        CELL44↑   next ;
} CELL44 ;

typedef struct {
        int       front ;
        int       rear ;
        CELL44↑   queue[1..maxlength] ;
        CELL44↑   hash_table[0..B–1] ;
} HASH_and_QUEUE ;


HASH_and_QUEUE      Q ;
```

ここで, *maxlength* 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.

次に, *name* に対するハッシュ関数 *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) を用意しなければならない. これらの詳細は省略するが, 本の 73 ページにあるハッシュ関数の例や 27 ページの同名の関数 *Same*(*x*, *y*) から, 容易に実現できるであろう.

```
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 */
```

また, 循環配列を用いたキューを用いるので, 与えられた整数 *i* の次を計算する関数 *Plus1*(*i*) (本の 41 ページ) も必要となる.

問題の関数 *Add*, *Deleted*, および *Count* のプログラムを作るため, これらの関数の操作の概要 (本の 80 ページの記述) を検討してみると, *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操作を, 上で定義したデータ構造のオープンハッシュ *hash_table*[ ] を用いて実現しなければならないことが分かる. そこで, 次の関数 *FindCell*(*name*, *Q*) を用意することにしよう.

この関数は, 集合 *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 (CELL44↑) を返し, 存在しないならば, **NULL** を返すものとする. こうしておくと, *Add*, *Deleted*, および *Count* のプログラムの作成が簡潔になることが期待できる.

なお, この関数の操作は, 集合をオープンハッシュで実現したときの *Member*(*x*, *A*) と同じである.

```
CELL44↑  FindCell( name, Q )
        入力: name  :    NAME 型のデータへのポインタ (値呼び)
              Q  :       HASH_and_QUEUE 型 (名前呼び)
        出力: FindCell  : CELL44↑ 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 (CELL44↑) 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 */
} /* FindCell */
```

これらを用いれば, *Add*(*name*, *Q*) の操作は次のように書ける.

```
boolean  Add( name, Q )
        入力: name  :    NAME 型のデータへのポインタ (値呼び)
              Q  :       HASH_and_QUEUE 型 (名前呼び)
        出力: Add  :     Boolean
              Q  :       HASH_and_QUEUE 型 (名前呼び)
        /* name で指定される顧客が集合 Q に無いならば, その顧客を Q の */
        /* 一番最後に入れ, true を返す. 在るならば false を返す */
{
        if ( FindCell( name, Q )≠NULL )    return   false ;
        else {                              /* name で指定される顧客は Q に存在しない */
              Q.rear := Plus1( Q.rear ) ;
              Q.queue[ Q.rear ] := new( CELL44 ) ;
              (↑Q.queue[ Q.rear ]).customer := name ;
              (↑Q.queue[ Q.rear ]).position := Q.rear ;
              (↑Q.queue[ Q.rear ]).next := Q.hash_table[ h( name ) ] ;
              Q.hash_table[ h( name ) ] := Q.queue[ Q.rear ] ;
              return   true ;
        }
} /* Add */
```

関数 *Delete* では, 顧客の名前のデータへのポインタ (NAME↑) を返すことにする. そうすると, *Deleted*(*Q*) の操作は次のように書ける.

```
NAME↑  Delete( Q )
        入力:    Q  :    HASH_and_QUEUE 型 (名前呼び)
        出力:  Delete  :  NAME 型のデータへのポインタ
                 Q  :    HASH_and_QUEUE 型 (名前呼び)
        /* Q の先頭に入っている顧客の名前を返し, Q からその顧客を取り除く */
        /* Q が空であれば何もしない */
{
        NAME↑    name ;
        CELL44↑              p ;
        if ( Plus1( Q.rear ) = Q.front )          /* Q は空なので, 何もしない */
                return  NULL ;
        else {                                    /* Q の先頭の顧客を取り除く */
                name := (↑Q.queue[Q.front ]).customer ;
                p := Q.hash_table[ h( name ) ] ;
                if ( Q.queue[Q.front ] = p )
                        Q.hash_table[ h( name ) ] := (↑p).next ;
                else {
                        while ( (↑p).next≠Q.queue[Q.front ] )  p := (↑p).next ;
                        (↑p).next := (↑Q.queue[Q.front ]).next ;
                }
                free( Q.queue[Q.front ] ) ;
                Q.front := Plus1(Q.front ) ;
                return name ;
        }
} /* Delete */
```

Count(*name*, *Q*) の操作は次のように書ける.

```
int   Count( name, Q )
        入力: name :    NAME 型のデータへのポインタ (値呼び)
              Q :       HASH_and_QUEUE 型 (名前呼び)
        出力: Count :   整数
        /* name で指定される顧客が待ち行列 Q において, 前から何番目かを調べる */
        /* Q に無いならば 0 を返す. 在るならば前から何番目かを示す数を返す */
{
        int   k ;
        if ( FindCell( name, Q ) = NULL )    return   0 ;
        else {                               /* name で指定される顧客は Q に含まれる */
              k := (↑FindCell( name, Q )).position ;
              if ( Q.front≦k )               return   k – Q.front + 1 ;
              else                           return   maxlength – Q.front + k + 1 ;
        }
} /* Count */
```

この解答では, *name* の型を顧客の名前 (NAME 型) へのポインタとしたが, 顧客データには名前の他に種々のデータが必要であろうから, 顧客データの入った構造体 (PERSON) を用意し, *name* の型をこのような構造体 (PERSON) へのポインタとすることもできる. その場合, 構造体 (PERSON) の中に名前を入れた欄 *c_name* を設け, この値をキーとしてハッシュ関数を計算することになるが, ハッシュ関数 *h*(*name*) の変更などは容易であろう.