7-13

#define 0 NULL

#define M 100

typedef struct{

keytype key;

datatype other;

} HashTable;

HashTable HT[M];

int LinSearch(HashTable HT[], keytype k)

{

int d = H(k); //待查找的关键字k的散列地址

int i = 0; //冲突后的地址增量

while((i < M) && (HT[d].key != k) && (HT[d].key != 0))

{

i++;

d = (d + 1) % M;

}

return d;

}

void LinInsert(HashTable HT[], HashTable R)

{

int d;

d = LinSearch(HT[], R.key);

if(HT[d].key == 0)

HT[d] = R.key;

else

printf("ERROR");

}

7-14

typedef struct nodetype

{

keytype key;

datatype other;

struct nodetype \* next;

}ChainHash;

ChainHash \* HTC[M];

ChainHash \* ChnSearch(ChainHash \* HTC[], keytype k)

{

ChainHash \* p;

p = HTC[H(k)];

while(p && (p -> key != k))

p = p -> next;

return p;

}

void ChnInsert(ChainHash \* HTC[], ChainHash \* s)

{

int d;

ChainHash \* p = ChnSearch(HTC, s -> key);

if(p)

printf("ERROR");

else

{

d = H(s -> key);

s -> next = HTC[d];

HTC[d] = s;

}

}

void ChnDelete(ChainHash \* HTC[], ChainHash \* s)

{

ChainHash \* pre = HTC[H(s -> key)];

ChainHash \* p = ChnSearch(HTC, s -> key);

if(!p)

printf("ERROR");

else

{

if(pre == p)

{

HTC[H(s -> key)] = p -> next;

free(p);

}

else

{

while(pre -> next != p)

pre = pre -> next;

pre -> next = p -> next;

free(p);

}

}

}