

```

/* 基数排序 - 主位优先 */

/* 假设元素最多有MaxDigit个关键字，基数全是同样的Radix */

#define MaxDigit 4
#define Radix 10

/* 桶元素结点 */
typedef struct Node *PtrToNode;
struct Node{
    int key;
    PtrToNode next;
};

/* 桶头结点 */
struct HeadNode {
    PtrToNode head, tail;
};
typedef struct HeadNode Bucket[Radix];

int GetDigit ( int X, int D )
{ /* 默认次位D=1, 主位D<=MaxDigit */
    int d, i;

    for (i=1; i<=D; i++) {
        d = X%Radix;
        X /= Radix;
    }
    return d;
}

void MSD( ElementType A[], int L, int R, int D )
{ /* 核心递归函数：对A[L]...A[R]的第D位数进行排序 */
    int Di, i, j;
    Bucket B;
    PtrToNode tmp, p, List = NULL;
    if (D==0) return; /* 递归终止条件 */

    for (i=0; i<Radix; i++) /* 初始化每个桶为空链表 */
        B[i].head = B[i].tail = NULL;
    for (i=L; i<=R; i++) { /* 将原始序列逆序存入初始链表List */
        tmp = (PtrToNode)malloc(sizeof(struct Node));
        tmp->key = A[i];
        tmp->next = List;
        List = tmp;
    }
    /* 下面是分配的过程 */
    p = List;
    while (p) {
        Di = GetDigit(p->key, D); /* 获得当前元素的当前位数字 */
        /* 从List中摘除 */
        tmp = p; p = p->next;
        /* 插入B[Di]号桶 */
        if (B[Di].head == NULL) B[Di].tail = tmp;
        tmp->next = B[Di].head;
        B[Di].head = tmp;
    }
    /* 下面是收集的过程 */
    i = j = L; /* i, j记录当前要处理的A[]的左右端下标 */
    for (Di=0; Di<Radix; Di++) { /* 对于每个桶 */
        if (B[Di].head) { /* 将非空的桶整桶倒入A[], 递归排序 */
            p = B[Di].head;
            while (p) {
                tmp = p;
                p = p->next;
                A[j++] = tmp->key;
                free(tmp);
            }
            /* 递归对该桶数据排序，位数减1 */
            MSD(A, i, j-1, D-1);
            i = j; /* 为下一个桶对应的A[]左端 */
        }
    }
}

void MSDRadixSort( ElementType A[], int N )
{ /* 统一接口 */
    MSD(A, 0, N-1, MaxDigit);
}

```