move a key to the right

```
void MoveRight(Treenode *current, int pos)
{
    int c;
    Treenode *t;
    t = current->branch[pos];
    for (c = t->count; c > 0; c--) {
    /* Shift all keys in the right node one position. */
        t->entry[c + 1] = t->entry[c];
        t->branch[c + 1] = t->branch[c];
    }
    t->branch[1] = t->branch[0];  /* Move key from parent to right node.          */
    t->count++;
    t->entry[1] = current->entry[pos];
    t = current->branch[pos-1];  /* Move last key of left node into parent.      */
    current->entry[pos] = t->entry[t->count];
    current->branch[pos]->branch[0] = t->branch[t->count];
    t->count--;
}
```

/* *MoveLeft: move a key to the left.*

Pre: current *points to a node in a B-tree with entries in the branches* pos *and* pos − 1, *with too few in branch* pos − 1.

Post: *The leftmost entry from branch* pos *has moved into* *current, *which has sent an entry into the branch* pos − 1. */

move a key to the left

```
void MoveLeft(Treenode *current, int pos)
{
    int c;
    Treenode *t;
    t = current->branch[pos-1];  /* Move key from parent into left node.        */
    t->count++;
    t->entry[t->count] = current->entry[pos];
    t->branch[t->count] = current->branch[pos]->branch[0];
    t = current->branch[pos];  /* Move key from right node into parent.         */
    current->entry[pos] = t->entry[1];
    t->branch[0] = t->branch[1];
    t->count--;
    for (c = 1; c <= t->count; c++) {
        /* Shift all keys in right node one position leftward. */
        t->entry[c] = t->entry[c + 1];
        t->branch[c] = t->branch[c + 1];
    }
}
```

```
/* Combine: combine adjacent nodes.
   Pre:  current points to a node in a B-tree with entries in the branches pos and
         pos - 1, with too few to move entries.
   Post: The nodes at branches pos - 1 and pos have been combined into one node,
         which also includes the entry formerly in *current at index pos. */
void Combine(Treenode *current, int pos)
{
   int c;
   Treenode *right;
   Treenode *left;
   right = current->branch[pos];
   left = current->branch[pos-1];  /* Work with the left node.                 */
   left->count++;                  /* Insert the key from the parent.          */
   left->entry[left->count] = current->entry[pos];
   left->branch[left->count] = right->branch[0];
   for (c = 1; c <= right->count; c++) { /* Insert all keys from right node.  */
      left->count++;
      left->entry[left->count] = right->entry[c];
      left->branch[left->count] = right->branch[c];
   }
   for (c = pos; c < current->count; c++) { /* Delete key from parent node.   */
      current->entry[c] = current->entry[c + 1];
      current->branch[c] = current->branch[c + 1];
   }
   current->count--;
   free(right);                    /* Dispose of the empty right node.         */
}
```

combine adjacent nodes

Exercises 10.3

E1. Insert the six remaining letters of the alphabet in the order

z, v, o, q, w, y

into the final B-tree of Figure 10.7 (page 476).

E2. Insert the entries below, in the order stated, into an initially empty B-tree of order **(a)** 3, **(b)** 4, **(c)** 7.

a g f b k d h m j e s i r x c l n t u p

E3. What is the smallest number of entries that, when inserted in an appropriate order, will force a B-tree of order 5 to have height 2 (that is, 3 levels)?

E4. Draw all the B-trees of order 5 (between 2 and 4 keys per node) that can be constructed from the keys 1, 2, 3, 4, 5, 6, 7, and 8.

E5. If a key in a B-tree is not in a leaf, prove that both its immediate predecessor and immediate successor (under the natural order) are in leaves.