```
element* search(treePointer tree,  int key)
{ /* key 값이 k인 노드에 대한 포인터를 반환함. 그런 노드가 없는 경우에는 NULL
     을 반환 */
   if (!root) return NULL;
   if (k == root→data.key) return &(root→data);
   if (k < root→data.key)
     return search(root→leftChild, k);
   return search(root→rightChild, k);
}
```

프로그램 5.15: 이원 탐색 트리의 순환적 탐색

```
element* iterSearch(treePointer tree,  int k)
{ /* key 값이 k인 노드에 대한 포인터를 반환함. 그런 노드가 없는 경우는 NULL을
     반환 */
   while (tree) {
     if (k == tree→data.key) return &(tree→data);
     if (k < tree→data.key)
       tree = tree→leftChild;
     else
       tree = tree→rightChild;
     }
   return NULL;
}
```

프로그램 5.16: 이원 탐색 트리의 반복적 탐색

```
void insert(treePointer *node, int k, iType theItem)
/* 트리 내 노드가 k를 가리키고 있으면 아무 일도 하지 않음;  그렇지 않은 경우는
   data = (k, theItem)인 새 노드를 첨가 */
   treePointer ptr, temp = modifiedSearch(*node, k);
   if (temp || !(*node)) {
      /* k is not in the tree * /
      MALLOC(ptr, sizeof(*ptr));
      ptr→data.key = k;
      ptr→data.item = theItem;
      ptr→leftChild = ptr→rightChild = NULL;
      if (*node) /* insert as child of temp */
         if (k < temp→data.key) temp→leftChild = ptr;
         else temp→rightChild = ptr;
      else *node = ptr;
   }
}
```

프로그램 5.17: 이원 탐색 트리에 사전 쌍의 삽입