```
// -----
// A collection of functions: does not compile
// Shows differences of interfaces:
//
         functional / procedural
//
//
           recursve / iterative
//
//
// -----
// Interface: recursive + functional
// Single return statement
//
// Call like this: anchor = list remove tail(anchor);
//
// Remove tail node of list
node_t* list_remove_tail(node_t* node){
 node t* result = node;
 if(node != NULL) {
   if (node->next == NULL) {
     // Remove last node
     free(node);
     result = NULL;
   } else {
     // Recursive call
     node->next = list_remove_tail(node->next);
 }
 return result;
```

```
// Interface: recursive + procedural
// Single return statement
//
// Call like this: list_remove_tail(&anchor);
//
// Remove tail node of list
void list_remove_tail(node_t* *pnode){
  node_t* node = *pnode;
  node t* result = node;
  if (\overline{n}ode != NULL) {
    if (node->next == NULL) {
      // Remove last node
      free(node);
      result = NULL;
    } else {
      // Recursive call
      list_remove_tail(&node->next);
    }
  *pnode = result;
```

```
// -----
// Interface: iterative + functional
// Single return statement
// Call like this: anchor = list remove tail(anchor);
//
// Remove tail node of list
node t* list remove tail(node t* node){
 node t *result = node; // init result
 if (node != NULL) {
   node t* last = NULL; // Last node before tail node
   // Go to tail node in list
   while (node->next != NULL) {
     last = node;
     // Advance iteration
     node = node -> next;
   // node points to tail node in list
   // Free that node
   free(node);
   if (last == NULL) {
     // node was the only node in the list
     result = NULL;
   } else {
     // List had at least two elements.
     // Terminate list.
     last->next = NULL;
   }
 }
  return result;
```

```
// Interface: iterative + procedural
// Single return statement
// Call like this: list remove tail(&anchor);
//
// Remove tail node of list
void list_remove_tail(node_t* *pnode){
 node_t = *pnode;
 node t *result = node; // init result
 if (node != NULL) {
    node t* last = NULL; // Last node before tail node
   // Go to tail node in list
   while (node->next != NULL) {
      last = node;
      // Advance iteration
      node = node -> next;
   // node points to tail node in list
    // Free that node
    free(node);
    if (last == NULL) {
      // node was the only node in the list.
      // Update result.
      result = NULL;
    } else {
      // The list had at least two elements.
      // Terminate the list
      last->next = NULL;
   }
  *pnode = result;
```