In the original program, find_part returns an index into the inventory array. In the new program, find_part will return a pointer to the node that contains the desired part number. If it doesn't find the part number, find_part will return a null pointer. Since the inventory list is sorted by part number, the new version of find_part can save time by stopping its search when it finds a node containing a part number that's greater than or equal to the desired part number. find_part's search loop will have the form

```
for (p = inventory;
     p != NULL && number > p->number;
     p = p->next)
  ;
```

The loop will terminate when p becomes NULL (indicating that the part number wasn't found) or when number > p->number is false (indicating that the part number we're looking for is less than or equal to a number already stored in a node). In the latter case, we still don't know whether or not the desired number is actually in the list, so we'll need another test:

```
if (p != NULL && number == p->number)
  return p;
```

The original version of insert stores a new part in the next available array element. The new version must determine where the new part belongs in the list and insert it there. We'll also have insert check whether the part number is already present in the list. insert can accomplish both tasks by using a loop similar to the one in find_part:

```
for (cur = inventory, prev = NULL;
     cur != NULL && new_node->number > cur->number;
     prev = cur, cur = cur->next)
  ;
```

This loop relies on two pointers: cur, which points to the current node, and prev, which points to the previous node. Once the loop terminates, insert will check whether cur isn't NULL and new_node->number equals cur->number; if so, the part number is already in the list. Otherwise insert will insert a new node between the nodes pointed to by prev and cur, using a strategy similar to the one we employed for deleting a node. (This strategy works even if the new part number is larger than any in the list; in that case, cur will be NULL but prev will point to the last node in the list.)

Here's the new program. Like the original program, this version requires the read_line function described in Section 16.3; I assume that readline.h contains a prototype for this function.

inventory2.c   /* Maintains a parts database (linked list version) */

```
#include <stdio.h>
#include <stdlib.h>
#include "readline.h"
```