```
    case 'q': terminate program;
    default:  print error message;
  }
}
```

It will be convenient to have separate functions perform the insert, search, update, and print operations. Since these functions will all need access to `inventory` and `num_parts`, we might want to make these variables external. As an alternative, we could declare the variables inside `main`, and then pass them to the functions as arguments. From a design standpoint, it's usually better to make variables local to a function rather than making them external (see Section 10.2 if you've forgotten why). In this program, however, putting `inventory` and `num_parts` inside `main` would merely complicate matters.

For reasons that I'll explain later, I've decided to split the program into three files: `inventory.c`, which contains the bulk of the program; `readline.h`, which contains the prototype for the `read_line` function; and `readline.c`, which contains the definition of `read_line`. We'll discuss the latter two files later in this section. For now, let's concentrate on `inventory.c`.

inventory.c

```
/* Maintains a parts database (array version) */

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

#define NAME_LEN 25
#define MAX_PARTS 100

struct part {
  int number;
  char name[NAME_LEN+1];
  int on_hand;
} inventory[MAX_PARTS];

int num_parts = 0;   /* number of parts currently stored */

int find_part(int number);
void insert(void);
void search(void);
void update(void);
void print(void);

/*****************************************************
 * main: Prompts the user to enter an operation code,     *
 *       then calls a function to perform the requested   *
 *       action. Repeats until the user enters the        *
 *       command 'q'. Prints an error message if the user *
 *       enters an illegal code.                          *
 *****************************************************/
int main(void)
{
  char code;
```