```
int num_parts;

int main(void)
{
  FILE *fp;
  int i;

  if ((fp = fopen("inventory.dat", "rb+")) == NULL) {
    fprintf(stderr, "Can't open inventory file\n");
    exit(EXIT_FAILURE);
  }

  num_parts = fread(inventory, sizeof(struct part),
                    MAX_PARTS, fp);

  for (i = 0; i < num_parts; i++)
    inventory[i].on_hand = 0;

  rewind(fp);
  fwrite(inventory, sizeof(struct part), num_parts, fp);
  fclose(fp);

  return 0;
}
```

Calling `rewind` is critical, by the way. After the `fread` call, the file position is at the end of the file. If we were to call `fwrite` without calling `rewind` first, `fwrite` would add new data to the end of the file instead of overwriting the old data.

## 22.8 String I/O

The functions described in this section are a bit unusual, since they have nothing to do with streams or files. Instead, they allow us to read and write data using a string as though it were a stream. The `sprintf` and `snprintf` functions write characters into a string in the same way they would be written to a stream; the `sscanf` function reads characters from a string as though it were reading from a stream. These functions, which closely resemble `printf` and `scanf`, are quite useful. `sprintf` and `snprintf` give us access to `printf`'s formatting capabilities without actually having to write data to a stream. Similarly, `sscanf` gives us access to `scanf`'s powerful pattern-matching capabilities. The remainder of this section covers `sprintf`, `snprintf`, and `sscanf` in detail.

Three similar functions (`vsprintf`, `vsnprintf`, and `vsscanf`) also belong to `<stdio.h>`. However, these functions rely on the `va_list` type, which is declared in `<stdarg.h>`. I'll postpone discussing them until Section 26.1, which covers that header.