```
i = i & ~0x0070 | 0x0050;   /* stores 101 in bits 4-6 */
```

The & operator clears bits 4–6 of `i`; the | operator then sets bits 6 and 4. Notice that `i |= 0x0050` by itself wouldn't always work: it would set bits 6 and 4 but not change bit 5. To generalize the example a little, let's assume that the variable `j` contains the value to be stored in bits 4–6 of `i`. We'll need to shift `j` into position before performing the bitwise *or*:

```
i = (i & ~0x0070) | (j << 4);   /* stores j in bits 4-6 */
```

The | operator has lower precedence than & and <<, so we can drop the parentheses if we wish:

```
i = i & ~0x0070 | j << 4;
```

- ***Retrieving a bit-field.*** When the bit-field is at the right end of a number (in the least significant bits), fetching its value is easy. For example, the following statement retrieves bits 0–2 in the variable `i`:

```
j = i & 0x0007;             /* retrieves bits 0-2 */
```

If the bit-field isn't at the right end of of `i`, then we can first shift the bit-field to the end before extracting the field using the & operator. To extract bits 4–6 of `i`, for example, we could use the following statement:

```
j = (i >> 4) & 0x0007;   /* retrieves bits 4-6 */
```

PROGRAM XOR Encryption

One of the simplest ways to encrypt data is to exclusive-*or* (XOR) each character with a secret key. Suppose that the key is the & character. If we XOR this key with the character z, we'll get the \ character (assuming that we're using the ASCII character set):

ASCII character set ➤*Appendix E*

```
   00100110  (ASCII code for &)
XOR 01111010  (ASCII code for z)
   01011100  (ASCII code for \)
```

To decrypt a message, we just apply the same algorithm. In other words, by encrypting an already-encrypted message, we'll recover the original message. If we XOR the & character with the \ character, for example, we'll get the original character, z:

```
   00100110  (ASCII code for &)
XOR 01011100  (ASCII code for \)
   01111010  (ASCII code for z)
```

The following program, `xor.c`, encrypts a message by XORing each character with the & character. The original message can be entered by the user or read from a file using input redirection; the encrypted message can be viewed on the screen or saved in a file using output redirection. For example, suppose that the file

input and output redirection ➤*22.1*