Using an uninitialized pointer variable as a string is a serious error. Consider the following example, which attempts to build the string `"abc"`:

```
char *p;

p[0] = 'a';     /*** WRONG ***/
p[1] = 'b';     /*** WRONG ***/
p[2] = 'c';     /*** WRONG ***/
p[3] = '\0';    /*** WRONG ***/
```

Since `p` hasn't been initialized, we don't know where it's pointing. Using the pointer to write the characters `a`, `b`, `c`, and `\0` into memory causes undefined behavior.

13.3 Reading and Writing Strings

Writing a string is easy using either the `printf` or `puts` functions. Reading a string is a bit harder, primarily because of the possibility that the input string may be longer than the string variable into which it's being stored. To read a string in a single step, we can use either `scanf` or `gets`. As an alternative, we can read strings one character at a time.

Writing Strings Using `printf` and `puts`

The `%s` conversion specification allows `printf` to write a string. Consider the following example:

```
char str[] = "Are we having fun yet?";

printf("%s\n", str);
```

The output will be

```
Are we having fun yet?
```

`printf` writes the characters in a string one by one until it encounters a null character. (If the null character is missing, `printf` continues past the end of the string until—eventually—it finds a null character somewhere in memory.)

To print just part of a string, we can use the conversion specification `%.`*p*`s`, where *p* is the number of characters to be displayed. The statement

```
printf("%.6s\n", str);
```

will print

```
Are we
```