Chapter 2: Comments

Comments are used to indicate something to the person reading the code. Comments are treated like a blank by the compiler and do not change anything in the code's actual meaning. There are two syntaxes used for comments in C, the original `/* */` and the slightly newer `//`. Some documentation systems use specially formatted comments to help produce the documentation for code.

Section 2.1: Commenting using the preprocessor

Large chunks of code can also be "commented out" using the preprocessor directives `#if 0` and `#endif`. This is useful when the code contains multi-line comments that otherwise would not nest.

```
#if 0 /* Starts the "comment", anything from here on is removed by preprocessor */

/* A large amount of code with multi-line comments */
int foo()
{
    /* lots of code */
    ...

    /* ... some comment describing the if statement ... */
    if (someTest) {
        /* some more comments */
        return 1;
    }

    return 0;
}

#endif /* 0 */

/* code from here on is "uncommented" (included in compiled executable) */
...
```

Section 2.2: /* */ delimited comments

A comment starts with a forward slash followed immediately by an asterisk (`/*`), and ends as soon as an asterisk immediately followed by a forward slash (`*/`) is encountered. Everything in between these character combinations is a comment and is treated as a blank (basically ignored) by the compiler.

```
/* this is a comment */
```

The comment above is a single line comment. Comments of this `/*` type can span multiple lines, like so:

```
/* this is a
multi-line
comment */
```

Though it is not strictly necessary, a common style convention with multi-line comments is to put leading spaces and asterisks on the lines subsequent to the first, and the `/*` and `*/` on new lines, such that they all line up:

```
/*
 * this is a
 * multi-line
 * comment
```

```
 */
```

The extra asterisks do not have any functional effect on the comment as none of them have a related forward slash.

These `/*` type of comments can be used on their own line, at the end of a code line, or even within lines of code:

```
/* this comment is on its own line */
if (x && y) { /*this comment is at the end of a line */
    if ((complexCondition1) /* this comment is within a line of code */
            && (complexCondition2)) {
        /* this comment is within an if, on its own line */
    }
}
```

Comments cannot be nested. This is because any subsequent `/*` will be ignored (as part of the comment) and the first `*/` reached will be treated as ending the comment. The comment in the following example *will not work*:

```
/* outer comment, means this is ignored => /* attempted inner comment */ <= ends the comment, not
this one => */
```

To comment blocks of code that contain comments of this type, that would otherwise be nested, see the Commenting using the preprocessor example below

Section 2.3: // delimited comments

`Version ≥ C99`

C99 introduced the use of C++-style single-line comments. This type of comment starts with two forward slashes and runs to the end of a line:

```
// this is a comment
```

This type of comment does not allow multi-line comments, though it is possible to make a comment block by adding several single line comments one after the other:

```
// each of these lines are a single-line comment
// note how each must start with
// the double forward-slash
```

This type of comment may be used on its own line or at the end of a code line. However, because they run *to the end of the line*, they may *not* be used within a code line

```
// this comment is on its own line
if (x && y) { // this comment is at the end of a line
    // this comment is within an if, on its own line
}
```

Section 2.4: Possible pitfall due to trigraphs

`Version ≥ C99`

While writing `//` delimited comments, it is possible to make a typographical error that affects their expected operation. If one types:

```
int x = 20;  // Why did I do this??/
```

The `/` at the end was a typo but now will get interpreted into `\`. This is because the `??/` forms a trigraph.

The `??/` trigraph is actually a longhand notation for `\`, which is the line continuation symbol. This means that the compiler thinks the next line is a continuation of the current line, that is, a continuation of the comment, which may not be what is intended.

```
int foo = 20; // Start at 20 ??/
int bar = 0;

// The following will cause a compilation error (undeclared variable 'bar')
// because 'int bar = 0;' is part of the comment on the preceding line
bar += foo;
```