Chapter 2: File Basics, Comments, and Program Headings

To program some of the very early computers, programmers had to rewire the machine. The programers got a large circuit board called a plug board, which was filled with little holes where they plugged in wires to create the program. Once they had programmed the board, they slid it into the computer and ran the program.

Computers soon evolved to the point where programmers could program them in text. They typed their program on a machine that output punched cards, dropped the thick deck of cards into the card reader, and the computer did the rest. Editing the program was as simple as replacing cards, but woe be to the programmer who dropped the program and scattered the cards.

Today we use text editors, which are certainly an improvement over punched cards and plug boards, but they do have their limitations. Knowing these limitations can help you to write code that will always be readable.

File Basics

C can accept files of almost any size, but there are some practical limitations. The longer a file, the more time and effort it takes to edit and print it. Most editors tend to get a bit slow when the file size gets to be more than about 3,000 lines. Keep yours within this limit.

Rule 2-1:

Keep programs files to no longer than about 2,000 to 3,000 lines.

Not only are there length limitations, but width limits as well. The old punch cards contained 80 columns. Because of that, most terminals and printers at the time were limited to 80 columns. This limitation has carried over to present-day computers. For example, the PC screen is only 80 columns wide.

Long lines can be a problem in programming. Many printers truncate lines that are too long, resulting in listings that look like this:

```
result = draw_line(last_x, last_y, next_x, next_y, line_style, end_style,
```

The code that fell off the right side is referred to as mystery code. So you need to limit the width of your program files to 80 characters. Actually, you need a slightly smaller limit. Program printers such as *cpr* print line numbered listings. Line numbers take up space, so a better limit, and one with enough history to be taken seriously, is 72 characters.

Rule 2-2:

Keep all lines in your program files down to 72 characters or fewer.

Early terminals had fixed tabs. Every eight characters you had a tab stop whether you liked it or not. You couldn't change them since they were built into the machine. This fixed tab size became an industry standard that is still in force today. If you type a file containing tabs under UNIX or DOS, the tabs come out every eight characters.

Many editors allow you to set your own tab stops. If you are programming in C with an indention size of 4, it is convenient to set the tab stop in your editor to 4. That way, to indent all you have to do is hit the Tab key. The problem with is that your tab setting is non-standard. If someone else edits your program, they won't know about your tabs and will assume that your code is indented strangely. Also, many printing programs and older programs default to a tab size of 8. Some, like DOS, can't be changed.

Note that tab size and indentation level are two different things. It is perfectly acceptable to use a tab size of 8 and an indentation level of 4. You would then use four spaces to reach the first level of indentation, a tab to reach the second, and so on.

Rule 2-3:

Use 8-character tab stops.

Finally, there is the character set. There are 95 printing characters in the standard ASCII set. The PC extended this set to include foreign characters and a line drawing set. It is possible to use the PC character set to draw interesting shapes,

like the following example:

```
/***************************************************
 * Boxes look like                                 *
 *     ┌──────────────────────────────────────────┐*
 *     │                                          │*
 *     │                                          │*
 *     │                                          │*
 ******└──────────────────────────────────────────┘*/
```

This makes for a nice picture on the screen, but what happens when you try to print it out? Most printers don't understand the PC character set, so you can easily get something that looks like this:

```
/***************************************************
 * Boxes look like                                 *
 *   LQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQL  *
 *   M                                         M  *
 *   M                                         M  *
 *   LQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQL  *
 **************************************************/
```

Worse, if this program is ported to another machine, such as a UNIX system, no one will understand the funny characters.

Rule 2-4:

> *Use only the 95 standard ASCII characters in your programs. Avoid exotic characters. (Foreign characters may be used if you are writing comments in a foreign language.)*

The Comment

Well-written code can help a programmer understand what is going on, but the best form of communication is the comment. Comments are used to explain everything. Without comments, a programmer has to go through the long and painful process of decrypting the program to figure out what the heck it does.

The comment can convey a variety of information, including program design, coding details, tricks, traps, and sometimes even jokes. There are many types of comments, from important ones that you want make sure the programmer doesn't miss to explanations of the tiniest details.

The author of a book has the advantage of typesetting devices. Important information can be set in **BIG BOLD LETTERS,** or words can be *emphasized* with italics.

The programmer, on the other hand, is limited to a single size, single face, monospaced font and personal imagination. Over the years a lot of imagination has been used to make up for the limitations of the medium.

Rule 2-5:

> *Include a heading comment at the beginning of each file that explains the file.*

The following program illustrates a variety of commenting styles collected over the years from many different sources.

```
/********************************************************
 * This is a boxed comment.  the box draws attention    *
 * to it.  This type of comment is used for programs,   *
 * module and function headings.                        *
 ********************************************************/

/* >>>>>>>>>>>>>>> Major Section Marker <<<<<<<<<<<<<<< */

/* --------------- Minor Section Marker --------------- */

static int count = 0; /* A simple end of line comment */

/* This was an end of line comment that grew too long */
static int total = 0;

/********************************************************
 ********************************************************
 ********** Warning: This is a very important  **********
 ********** message. If the programmer misses  **********
 ********** it, the program might crash and    **********
 ********** burn. (Gets your attention,        **********
 ********** doesn't it?)                       **********
 ********************************************************
 ********************************************************/
int main(void)
{
    /* This is an in-line comment */

    ++total;

    /*
     * This is a longer in-line comment.
     * Because it is so verbose it is split into two lines.
     */

    return (0);
}
```

Other types of comments include:

```
    /*-------------------*\
     * Another box style  *
    \*-------------------*/

    /*
     *      *================*
     *      * Section Header *
     *      *================*
     *
     * This is a sentence with **one** word emphasized.
     */
```

Graphics

Computers are becoming more and more graphically oriented. Screen layouts, windowing systems, and games all require sophisticated graphics. It's not possible to put graphic comments into a program yet, but you can make a good attempt by using ASCII line art.

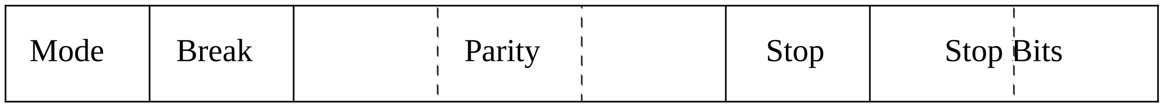
A typical screen layout comment might look like this:

```
    /***************************************************
     * Format of address menu                          *
     *                                                 *
     *  <----- MENU_WIDTH ------>        MENU_HEIGHT   *
     *  +-----------------------+                 ^    *
     *  | Name: _______________ | <-- NAME_LINE    |   *
     *  | Address: ____________ | <-- ADDRESS_LINE |   *
     *  | City: _______________ | <-- CITY_LINE    |   *
     *  | State: __ Zip: ______ | <-- STATE_LINE   |   *
     *  +-----------------------+                 v    *
     *    ^         ^                                  *
     *    |         |                                  *
     *    |         +--- ZIP_X                         *
     *    +--- BLANK_X                                 *
     ***************************************************/
```

Even with the crude drawing tools in the 95 standard ASCII characters, you can produce a recognizable picture. This type of comment actually conveys graphically the relationships among the constants shown.

Packing bits

If you do 1/0 programming, you know that hardware designers like to pack a lot of functions into a single byte. For example, the serial 1/0 chip that controls the COM ports on a PC contains a variety of packed registers.



```
Parity:
        000 - No parity
        001 - Odd Parity / No Check
        010 - Even Parity / No Check
        011 - High Bit always set
        100 - Odd Party / Check Incoming characters
        101 - Even Parity / Check Incoming characters
        110 - Undefined
        111 - Parity Clear

Stop Bits:

        00 - 1 Stop Bit
        01 - 1.5 Stop Bits
        10 - 2 Stop Bits
        11 - Undefined
```

These registers are documented in the chip's data sheet. Unfortunately, most programmers don't carry around a complete set of data sheets, so it is very useful to copy the register specification into the

This can be turned into a nice comment and a few defines. (How to write the **#define** statements is discussed in Chapter 6.)

```
/************************************************
 * Line Control Register                        *
 *   for the PC's COM ports                     *
 *                                              *
 *      76543210                                *
 *      XXXXXXXX                                *
 *      ^^^^^^++---- Number of stop bits        *
 *      |||||+------ Enable Transmitter         *
 *      ||+++------- Parity Generation          *
 *      |+---------- Send Break                 *
 *      +----------- Mode control               *
 ************************************************/

/*
 * Define the number of stop bits
 */
#define STOP_1_BIT        (0 << 0)
#define STOP_15_BIT       (1 << 0)
#define STOP_2_BIT        (2 << 0)

#define TRANSMIT_ENABLE   (1 << 2)
/*
 * Parity Mode
 */
#define PARITY_NONE       (0 << 3)
#define PARITY_ODD        (1 << 3)
#define PARITY_EVEN       (2 << 3)
#define PARITY_HIGH       (3 << 3)
#define PARITY_ODD_CHECK  (4 << 3)
#define PARITY_EVEN_CHECK (5 << 3)
#define PARITY_CLEAR      (7 << 3)

#define BREAK             (1 << 6)

#define MODE_ASYNC        (0 << 7)
#define MODE_SYNC         (1 << 7)
```

Letting the Editor Help You

Most editors have a macro of abbreviation features that make it quick and easy to create boxed comments.

If you use the UNIX editor vi, you can put the following in your. *exrc* file to define two abbreviations:

```
:ab #b /************************************************
:ab #e ************************************************/
```

When you type `#b`, the editor changes it to a beginning box, while typing `#e` creates an ending comment.

On the PC, there is Borland's C++ compiler, which comes with a macro file named *CMACROS.TEM*. These macros must be installed using the *TEMC* command. Type:

```
TEMC cmacros.tem tcconfig.tc
```

These macros are a bit limited, however, and you might want to edit them before using them in production.

Beginning Comment Block

The first two questions a programmer should ask when confronting a strange program are “What is it?” and “What does it do?” Heading comments should answer both questions.

The top of a program serves as a sort of title page and abstract. It briefly describes the program and provides vital information about it.

Here, the heading comments are boxed. This not only makes them stand out, but it easily identifies them as containing important and generally useful information. The first line of the heading block contains the name of the program and a short description of what it does.

The sections of a heading

The following is a list of heading sections, but not all sections apply to all programs. Use only those that are useful to your program.

- **Purpose**

  Why was this program written? What does it do?

- **Author**

  it took you a great deal of time and trouble to create this program. Sign your work. Also, when someone else has to modify this program, they can come to you and ask you to explain what you did.

- **Copyright or License**

  Most commercial programs are protected by copyright or trade secret laws. Generally, this is some boilerplate text devised by lawyers. You don't have to understand it, but you should put it in.

- **Warning**

  Sometimes a programmer discovers the hard way that his program contains traps or pitfalls. This section should warn about potential problems. For example: “Don't compile with stack checking on. This is a clever program, and it does strange things with the stack.”

- **Usage**

  Briefly explain how to use the program. Oualline's law of documental states: 90 percent of the time, the documentation is lost. Of the remaining 10 percent, 9 percent of the time the documentation is for a different version of the software and is completely useless. The 1 percent of the time you have the correct documentation, it is written in Chinese.

  A simple way to prevent the program and the documentation from being separated is to put the documentation in the program. You don't need a complete tutorial, just enough for a quick reference.

- **Restrictions**

  This section lists any restrictions that the program might have, such as “This program is designed to process the output of PLOT5 program. It does not do extensive error checking and may behave strangely if given bad input.”

- **Algorithms**

  If this program uses any special techniques or algorithms, list them here.

- **References**

  Often a programmer will find it useful to copy or adapt an algorithm from a book or other source (as long as copyright laws are not violated). But give credit where credit is due. Listing the source of any special algorithms in this section gives the people who follow you a chance to check the original work.

- **File Formats**

  This section briefly describes the format of any data files used by the program. This section may also be duplicated in the module that reads or writes the file.

- **Revision History**

  it's not unusual for a number of people to work on a single program over the years. This section lists those who worked on the program, gives a short description of what they did, and tells when the work was done. Revision control software such as *RCS* and *SCCS* will automatically generate this information.

- **Notes**

  This is a catch-all for any other information you may want future programmers to have.

```
/********************************************************
 * Analyze -- analyze the complexity of a program       *
 *                                                      *
 * Purpose:                                             *
 *      This program produces a set of statistics that  *
 *      are supposed to tell you how complex a program  *
 *      is.                                             *
 *                                                      *
 * Author: John Jones.                                  *
 *                                                      *
 * Copyright 1999 by John Jones.                        *
 *                                                      *
 * Warning: Compiling with optimization causes          *
 *      incorrect code to be generated.                 *
 *                                                      *
 * Restrictions: Works only on error-free C files.      *
 *      Does not know about pre-processor directives.   *
 *                                                      *
 * Algorithms: Uses a classic unbalanced binary tree    *
 *      for a symbol table.                             *
 *                                                      *
 * References: "Software complexity measurements",      *
 *      Flying Fingers Newsletter, May 5, 1995.         *
 *                                                      *
 * Output file format for raw data file:                *
 *              <magic number> (AC_DATA_MAGIC)          *
 *              <# statistics that follow>              *
 *                 <stat table index>                   *
 *                 <value>                              *
 *                 (Repeat for each stat.)              *
 *                                                      *
 * Revision history:                                    *
 *      1.0 July 5, 1995        Ralph Smith             *
 *              Initial Version.                        *
 *                                                      *
 *      1.5 Nov 5, 1995         Bill Green              *
 *              Add comment / code ratio                *
 *                                                      *
 *      2.0 Jan 8, 1996         Bill Green              *
 *              Extensive rework of the report gen.     *
 *                                                      *
 *      2.1 Jan 30, 1997        Bill Green               *
 *              Ported to Windows-95.                    *
 *                                                       *
 * Note: This program generates a lot of numbers         *
 *      about the target program.  Not all are useful.  *
 ********************************************************/
```

This particular example is a bit long. It was created to show practical uses of every section. But it illustrates a problem with style guidelines: there is a strong temptation to overdo it. All too often, a programming team will form a style committee, toss around a bunch of ideas for documenting the code, and end up throwing them all into the header. This is almost guaranteed to produce confusing headers that are themselves a maintenance headache.

Heading comments should be as simple as possible, but no simpler.

Too much information is a burden on the programmer. It takes time to type it in and to maintain it. Comments that take a lot of time to create and maintain tempt the programmer to take shortcuts. The result is incorrect or misleading comments, and a *wrong comment is worse than no comment at all.*

Real programs have shorter headers. Here is a header taken from a real program:

```
/********************************************************
 * lab -- handle the labeling of diskettes.             *
 *                                                      *
 * Usage:                                               *
 *      lab -w <drive>:<name>   Write label to disk.    *
 *      lab -r <drive>:         Read label.             *
 *      lab -c <drive: drive:   Copy label.             *
 *                                                      *
 * Copyright 1992, Steve Oualline                       *
 ********************************************************/
#include <stdio.h>
```

Other sections

We've listed a general set of heading sections. You may need additional sections, depending on your environment. For example, a student may be required to put in an assignment number, social security number, and teacher's name. Professional programs may require a part number. Shareware must include a paragraph that asks the user to pay a license fee, along with an address to which users can send money.

Module Headings

Modules are similar to program files, except that there is no `main` function. Their heading comments are also structured similarly, except that there is no “Usage” section.

```
/********************************************************
 * symbol.c -- Symbol table routines                    *
 *                                                      *
 * Author: Steve Oualline                               *
 *                                                      *
 * Copyright 1992 Steve Oualline                        *
 *                                                      *
 * Warning: Running out of memory kills the program.    *
 *                                                      *
 * Algorithm:                                           *
 *      The symbol table is kept as a balanced binary   *
 *      tree.                                           *
 ********************************************************/
```

Some programmers put a list of the public functions in the heading comments. This is not recommended. First, all the public functions are already listed in the header file for this module. Second, keeping this list up to date requires work, and frequently a programmer will forget to make the updates.

Function Headings

C functions serve much the same purpose as sections of a chapter in a book. They deal with a single subject or operation that the reader can easily absorb.

In this book, each section starts with a section heading in bold letters. This allows the user to scan down a page to locate a section.

A function needs a similar heading. The comment box for a function should contain the following sections:

- **Name**

  The name of the function and a brief comment describing it.

- **Parameters**

  A list of parameters (one per line) and a brief description of each of them. Sometimes the words (returned) or (updated) are added.

- **Return value**

  Describes what value the function returns. In addition to these standard sections, any other useful information about the function can be added. Here's an example:

```
/********************************************************
 * find_lowest -- find the lowest number in an array    *
 *                                                      *
 * Parameters                                           *
 *      array -- the array of integers to search.       *
 *      count -- the number of items in the array.      *
 *                                                      *
 * Returns                                              *
 *      The index of the lowest number in the array     *
 *      (in case of a tie, the first instance of the    *
 *      number.)                                        *
 ********************************************************/
```

Some people include another section: Globals Used. This is very useful information, but it is difficult to get and maintain. It takes a lot of work to keep this section current, and frequently a programmer will get lazy and ignore it. It is better not to have a Globals Used section than to have one that is wrong.

Rule 2-6:

Leave out unnecessary comments if they require maintenance and if you are unlikely to maintain them.

When to Write Comments

It is best to put your comments in the program as you are writing it. If you start your program with a set of heading comments, then you should have a pretty good idea what you are planning to do. It helps focus your thoughts.

Avoid the two-step process of coding and later going back and adding comments. This method has several problems. First, you are likely to forget what you did. What may be obvious when you write it may not be so obvious when you re-read it.

Another good reason to write comments while you're writing the code is psychological. When the code is done, you're probably going to feel that the program is done. Adding comments then becomes a chore to be completed as quickly as possible. Generally, this means you'll put in too few comments.

It is especially helpful to do things like screen layouts in comments before you start coding. That way you have a model to work from.

Rule 2-7:

Comment your code as you write it.

Some Comments on Comments

The heading comments always seem a bit long to the person creating the program. To the person trying to maintain it, they always seem far too short. Balance is the key to good commenting. Make your comments short enough so they aren't bothersome to put in, yet long enough to give other programmers a good idea of what's going on.

Overly commented programs are rare. Usually they turn up in the work of eager first-year programming students.

Under-commented programs are far too frequent. Too many programmers think that their code is obvious. It is not.

There is a reason it is called “code.”