13.6 Syntax and Runtime Errors

The most common syntax errors will be lack of matching parentheses, brackets, braces, or quotation marks. When you encounter a syntax error, this is the first thing you should check and double-check. I highly recommend that you use a text editor that does parentheses matching and syntax coloring for R, such as Vim or Emacs.

Be aware that often when you get a message saying there is a syntax error on a certain line, the error may actually be in a much earlier line. This can occur with any language, but R seems especially prone to it.

If it just isn’t obvious to you where your syntax error is, I recommend selectively commenting out some of your code, better enabling you to pinpoint the location of the syntax problem. Generally, it helps to follow a binary search approach: Comment out half of your code (being careful to maintain syntax integrity) and see if the same error arises. If it does, it’s in the remaining half; otherwise, it’s in the half you deleted. Then cut that half in half, and so on.

You may sometimes get messages like the following:

There were 50 or more warnings (use warnings() to see the first 50)

These should be heeded—run warnings() as suggested. The problem could range from nonconvergence of an algorithm to misspecification of a matrix argument to a function. In many cases, the program output may be invalid, though it may well be fine, too, say with this message:

Fitted probabilities numerically 0 or 1 occurred in: glm...

In some cases, you may find it useful to issue this command:

> options(warn=2)

This instructs R to turn warnings into actual errors and makes the locations of the warnings easier to find.

13.7 Running GDB on R Itself

This section may be of interest to you even if you are not trying to fix a bug in R. For example, you may have written some C code to interface to R (covered in Chapter 15) and found it to be buggy. In order to run GDB on that C function, you must first run R itself through GDB.

Or, you may be interested in the internals of R, say to determine how you can write efficient R code, and wish to explore the internals by stepping through the R source code with a debugging tool such as GDB.
Although you can invoke R through GDB from a shell command line (see Section 15.1.4), for our purposes here, I suggest using separate windows for R and GDB. Here’s the procedure:

1. Start R in one window, as usual.
2. In another window, determine the ID number of your R process. In UNIX family systems, for instance, this is obtained by something like `ps -a`.
3. In that second window, submit GDB’s `attach` command with the R process number.
4. Submit the `continue` command to GDB.

You can set breakpoints in the R source code either before continuing or by interrupting GDB later with `CTRL-C`. See Section 15.1.4 for details for debugging C code called from R. If, on the other hand, you wish to use GDB to explore the R source code, note the following.

The R source code is dominated by S expression pointers (SEXP$s), which are pointers to C structs that contain an R variable’s value, type, and so on. You can use the R internal function `Rf_PrintValue(s)` to inspect SEXP values. For example, if the SEXP is named `s`, then in GDB, type this:

```
call Rf_PrintValue(s)
```

This prints the value.