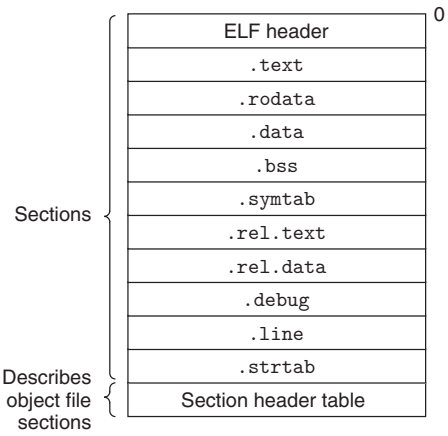


**Figure 7.3**
**Typical ELF relocatable object file.**

## 7.4 Relocatable Object Files

Figure 7.3 shows the format of a typical ELF relocatable object file. The *ELF header* begins with a 16-byte sequence that describes the word size and byte ordering of the system that generated the file. The rest of the ELF header contains information that allows a linker to parse and interpret the object file. This includes the size of the ELF header, the object file type (e.g., relocatable, executable, or shared), the machine type (e.g., x86-64), the file offset of the section header table, and the size and number of entries in the section header table. The locations and sizes of the various sections are described by the *section header table*, which contains a fixed-size entry for each section in the object file.

Sandwiched between the ELF header and the section header table are the sections themselves. A typical ELF relocatable object file contains the following sections:

`.text` The machine code of the compiled program.

`.rodata` Read-only data such as the format strings in `printf` statements, and jump tables for switch statements.

`.data` *Initialized* global and static C variables. Local C variables are maintained at run time on the stack and do *not* appear in either the `.data` or `.bss` sections.

`.bss` *Uninitialized* global and static C variables, along with any global or static variables that are initialized to zero. This section occupies no actual space in the object file; it is merely a placeholder. Object file formats distinguish between initialized and uninitialized variables for space efficiency: uninitialized variables do not have to occupy any actual disk space in the object file. At run time, these variables are allocated in memory with an initial value of zero.

**Aside** Why is uninitialized data called `.bss`?

The use of the term `.bss` to denote uninitialized data is universal. It was originally an acronym for the "block started by symbol" directive from the IBM 704 assembly language (circa 1957) and the acronym has stuck. A simple way to remember the difference between the `.data` and `.bss` sections is to think of "bss" as an abbreviation for "Better Save Space!"

`.symtab` A *symbol table* with information about functions and global variables that are defined and referenced in the program. Some programmers mistakenly believe that a program must be compiled with the `-g` option to get symbol table information. In fact, every relocatable object file has a symbol table in `.symtab` (unless the programmer has specifically removed it with the STRIP command). However, unlike the symbol table inside a compiler, the `.symtab` symbol table does not contain entries for local variables.

`.rel.text` A list of locations in the `.text` section that will need to be modified when the linker combines this object file with others. In general, any instruction that calls an external function or references a global variable will need to be modified. On the other hand, instructions that call local functions do not need to be modified. Note that relocation information is not needed in executable object files, and is usually omitted unless the user explicitly instructs the linker to include it.

`.rel.data` Relocation information for any global variables that are referenced or defined by the module. In general, any initialized global variable whose initial value is the address of a global variable or externally defined function will need to be modified.

`.debug` A debugging symbol table with entries for local variables and typedefs defined in the program, global variables defined and referenced in the program, and the original C source file. It is only present if the compiler driver is invoked with the `-g` option.

`.line` A mapping between line numbers in the original C source program and machine code instructions in the `.text` section. It is only present if the compiler driver is invoked with the `-g` option.

`.strtab` A string table for the symbol tables in the `.symtab` and `.debug` sections and for the section names in the section headers. A string table is a sequence of null-terminated character strings.

## 7.5 Symbols and Symbol Tables

Each relocatable object module, $m$, has a symbol table that contains information about the symbols that are defined and referenced by $m$. In the context of a linker, there are three different kinds of symbols: