TABLE 6.1: *Constant tags*

Tag	Type	Format	Interpretation
1	`UTF8`	2+*n* bytes	The first two bytes are an unsigned integer *n*; the remaining *n* bytes are the text of the constant.
2	*not defined*		
3	`Integer`	4 bytes	Signed integer
4	`Float`	4 bytes	IEEE 754 floating-point number
5	`Long`	8 bytes	Long signed integer
6	`Double`	8 bytes	IEEE 754 double-precision number
7	`Class`	2 bytes	Reference to a `UTF8` constant that is the name of a class
8	`String`	2 bytes	Reference to a `UTF8` constant that is the value of the `String`
9	`Fieldref`	4 bytes	The first two bytes are a reference to a `Class`; the second two point to a `NameAndType`.
10	`Methodref`	4 bytes	Same as `Fieldref`
11	`InterfaceMethodref`	4 bytes	Same as `Fieldref`
12	`NameAndType`	4 bytes	The first two bytes point to a `UTF8` that is the name of the field or method; the second two point to a `UTF8`, which is its descriptor.

Most sections begin with a count, which is a two-byte unsigned integer, followed by that many instances of some pattern of bytes. For example, following the major version number is the count of the number of constants. Each constant begins with a tag describing what sort of constant it is, which in turn tells how many bytes make up the constant. The set of constant tags is defined by the virtual machine specification. If any constant tag is invalid, or if the file ends before the correct number of constants are found, then the file is rejected. The valid constant tags are given in Table 6.1.

Similar rules apply to the other sections. If the file ends before all of the parts are found, or if there are extra bytes at the end, then the file is rejected. For more about the details of the inner workings of the `class` file, see chapter 9.

6.3 Are All Constant References Correct?

After asking whether or not the file looks like a properly formatted `class` file, the verification algorithm knows where the constant pool is to be found and how