template will be used to generate a specialization with $T$ taken as type $\text{char}$. That leaves five viable functions, each of which could be used if it were the only function declared.

Next, the compiler has to determine which of the viable functions is best. It looks at the conversion required to make the function call argument match the viable candidate's argument. In general, the ranking from best to worst is this:

1. Exact match, with regular functions outranking templates.
2. Conversion by promotion (the automatic conversions of $\text{char}$ and $\text{short}$ to $\text{int}$ and of $\text{float}$ to $\text{double}$, for example)
3. Conversion by standard conversion (converting $\text{int}$ to $\text{char}$ or $\text{long}$ to $\text{double}$, for example)
4. User-defined conversions, such as those defined in class declarations

For example, function #1 is better than function #2 because $\text{char}$-to-$\text{int}$ is a promotion (Chapter 3, "Dealing with Data"), whereas $\text{char}$-to-$\text{float}$ is a standard conversion (Chapter 3). Functions #3, #5, and #6 are better than either #1 or #2, because they are exact matches. Both #3 and #5 are better than #6 because #6 is a template. This analysis raises a couple of questions. What is an exact match, and what happens if you get two of them?

**Exact Matches and Best Matches**

C++ allows some "trivial conversions" when making an exact match. Table 8.1 lists them, with $\text{Type}$ standing for some arbitrary type. For example, an $\text{int}$ actual argument is an exact match to an $\text{int \&}$ formal parameter. Note that $\text{Type}$ can be something like $\text{char \&}$, so these rules include converting $\text{char \&}$ to $\text{const char \&}$. The $\text{Type (argument-list)}$ entry means that a function name as an actual argument matches a function pointer as a formal parameter as long as both have the same return type and argument list. (Remember function pointers from Chapter 7, "Functions?C++’s Programming Modules"," and how you can pass the name of a function as an argument to a function expecting a pointer to a function.) We discuss the $\text{volatile}$ keyword later in this chapter.