#2 already explicitly said the function argument was pointer-to-`Type` so `Type` could be directly identified with `blot`. Template #1, however, had `Type` as the function argument, so `Type` had to be interpreted as pointer-to-`blot`. That is, in template #2, `Type` already was specialized as a pointer, hence the term "more specialized."

The rules for finding the most specialized template are called the **partial ordering rules** for function templates. Like explicit instantiations, they are new additions to the language.

**A Partial Ordering Rules Example**

Let's examine a complete program that uses the partial ordering rules for identifying which template definition to use. **Listing 8.12** has two template definitions for displaying the contents of an array. The first definition (template A) assumes that the array passed as an argument contains the data to be displayed. The second definition (template B) assumes that the array argument contains pointers to the data to be displayed.

**Listing 8.12 temptempover.cpp**

```cpp
// temptempover.cpp --- template overloading
#include <iostream>
using namespace std;
template <typename T> // template A
void ShowArray(T arr[], int n);

template <typename T> // template B
void ShowArray(T * arr[], int n);

struct debts
{
    char name[50];
    double amount;
};

int main(void)
{
    int things[6] = { 13, 31, 103, 301, 310, 130};
```