ShowArray(things, 6);

The identifier things is the name of an array of int, so it matches the template

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

with T taken to be type int.

Next, consider this function call:

ShowArray(pd, 3);

Here, pd is the name of an array of double *. This could be matched by template A:

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

Here, T would be taken to be type double *. In this case, the template function would
display the contents of the pd array, namely, three addresses. The function call also could
be matched by template B:

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

In this case, T would be type double, and the function displays the dereferenced elements
*arr[i], that is, the double values pointed to by the array contents. Of the two templates,
template B is the more specialized because it makes the specific assumption that the array
contents are pointers, so it is the template that gets used.

Here’s the output:

Listing Mr. E’s counts of things:
template A
13 31 103 301 310 130
Listing Mr. E's debts:
template B
2400 1300 1800