Templates & Overloading

Suppose you wanted to print pointers differently than non-pointers, you can add an **overloaded template function** like this:

```
template <typename T>
void print(const T* p)
{
    cout << "Pointer: " << p << " ";
    if (p) cout << *p; else cout << "nullptr";
}
```

Now, what if you want floating-point numbers and Booleans to print differently than other kinds of values? You can add a pair of **explicit, non-template, overloaded functions**, like this:

```
void print(double val, int dec=2)
{
    cout << fixed << setprecsion(dec) << val;
}
void print(bool val)
{
    cout << boolalpha << val;
}
```

Now `print(2.5)` will print `2.50`, while `print(2.5, 4)` will print `2.5000`. Printing a *Boolean* expression will print `true` or `false`, not `0` or `1` like the original template.

When you overload template functions:

- Any call to a template function, where the template argument deduction succeeds, is a **viable member** of the overload **candidate set**.
- If there is a **non-template** function in the viable set, then **it is preferred**.

The **most specialized** template function in the viable set is preferred over the other template functions.