int m = 0;
while ( m <= n && str[m] != '\0')
    m++;
char * p = new char[m+1];
// use m instead of n in rest of code

Function Polymorphism (Function Overloading)

Function polymorphism is a neat C++ addition to C's capabilities. While default arguments let you call the same function using varying numbers of arguments, *function polymorphism*, also called *function overloading*, lets you use multiple functions sharing the same name. The word "polymorphism" means having many forms, so function polymorphism lets a function have many forms. Similarly, the expression "function overloading" means you can attach more than one function to the same name, thus overloading the name. Both expressions boil down to the same thing, but we'll usually use the expression function overloading—it sounds harder-working. You can use function overloading to design a family of functions that do essentially the same thing, but using different argument lists.

Overloaded functions are analogous to verbs having more than one meaning. For example, Miss Piggy can root at the ball park for the home team, and or she can root in the soil for truffles. The context (one hopes) tells you which meaning of root is intended in each case. Similarly, C++ uses the context to decide which version of an overloaded function is intended.

The key to function overloading is a function's argument list, also called the *function signature*. If two functions use the same number and types of arguments in the same order, they have the same signature; the variable names don't matter. C++ enables you to define two functions by the same name provided that the functions have different signatures. The signature can differ in the number of arguments or in the type of arguments, or both. For example, you can define a set of print() functions with the following prototypes:

void print(const char * str, int width);  // #1
void print(double d, int width);          // #2
void print(long l, int width);            // #3
void print(int i, int width);             // #4