Chapter Summary

Functions are named units of computation and are essential to structuring even modest programs. Every function has a return type, a name, a (possibly empty) list of parameters, and a function body. The function body is a block that is executed when the function is called. When a function is called, the arguments passed to the function must be compatible with the types of the corresponding parameters.

In C++, functions may be overloaded: The same name may be used to define different functions as long as the number or types of the parameters in the functions differ. The compiler automatically figures out which function to call based on the arguments in a call. The process of selecting the right function from a set of overloaded functions is referred to as function matching.

Defined Terms

ambiguous call Compile-time error that results during function matching when two or more functions provide an equally good match for a call.

arguments Values supplied in a function call that are used to initialize the function's parameters.

assert Preprocessor macro that takes a single expression, which it uses as a condition. When the preprocessor variable `NDEBUG` is not defined, `assert` evaluates the condition and, if the condition is false, writes a message and terminates the program.

automatic objects Objects that exist only during the execution of a function. They are created when control passes through their definition and are destroyed at the end of the block in which they are defined.

best match Function selected from a set of overloaded functions for a call. If a best match exists, the selected function is a better match than all the other viable candidates for at least one argument in the call and is no worse on the rest of the arguments.

call by reference See pass by reference.

call by value See pass by value.

candidate functions Set of functions that are considered when resolving a function call. The candidate functions are all the functions with the name used in the call for which a declaration is in scope at the time of the call.

constexpr Function that may return a constant expression. A `constexpr` function is implicitly `inline`.

default argument Value specified to be used when an argument is omitted in a call to the function.

executable file File, which the operating system executes, that contains code corresponding to our program.

function Callable unit of computation.

function body Block that defines the actions of a function.

function matching Compiler process by which a call to an overloaded function is resolved. Arguments used in the call are compared to the parameter list of each overloaded function.

function prototype Function declaration, consisting of the name, return type, and parameter types of a function. To call a function, its prototype must have been declared before the point of call.

hidden names Names declared inside a scope hide previously declared entities with the same names declared outside that scope.

initializer_list Library class that represents a comma-separated list of objects of a single type enclosed inside curly braces.

inline function Request to the compiler to expand a function at the point of call, if possible. Inline functions avoid the normal function-calling overhead.

link Compilation step in which multiple object files are put together to form an executable program.

local static objects Local objects whose value persists across calls to the function. Local `static` objects that are created and initialized before control reaches their use and are destroyed when the program ends.

local variables Variables defined inside a block.

no match Compile-time error that results during function matching when there is no function with parameters that match the arguments in a given call.

object code Format into which the compiler transforms our source code.

object file File holding object code generated by the compiler from a given source file. An executable file is generated from one or more object files after the files are linked together.

object lifetime Every object has an associated lifetime. Nonstatic objects that are defined inside a block exist from when their definition is encountered until the end of the block in which they are defined. Global objects are created during program startup. Local `static` objects are created before the first time execution passes through the object's definition. Global objects and local `static` objects are destroyed when the `main` function ends.

overload resolution See function matching.

overloaded function Function that has the same name as at least one other function. Overloaded functions must differ in the number or type of their parameters.

parameters Local variables declared inside the function parameter list. Parameters are initialized by the arguments provided in each function call.

pass by reference Description of how arguments are passed to parameters of reference type. Reference parameters work the same way as any other use of references; the parameter is bound to its corresponding argument.

pass by value How arguments are passed to parameters of a nonreference type. A nonreference parameter is a copy of the value of its corresponding argument.

preprocessor macro Preprocessor facility that behaves like an inline function. Aside from `assert`, modern C++ programs make very little use of preprocessor macros.

recursion loop Description of a recursive function that omits a stopping condition and which calls itself until exhasuting the program stack.

recursive function Function that calls itself directly or indirectly.

return type Part of a function declaration that specifies the type of the value that the function returns.

separate compilation Ability to split a program into multiple separate source files.

trailing return type Return type specified after the parameter list.

viable functions Subset of the candidate functions that could match a given call. Viable functions have the same number of parameters as arguments to the call, and each argument type can be converted to the corresponding parameter type.

`()` operator Call operator. Executes a function. The name of a function or a function pointer precedes the parentheses, which enclose a (possibly empty) comma-separated list of arguments.