Chapter Summary

Types are fundamental to all programming in C++.

Each type defines the storage requirements and the operations that may be performed on objects of that type. The language provides a set of fundamental built-in types such as `int` and `char`, which are closely tied to their representation on the machine's hardware. Types can be non`const` or `const`; a `const` object must be initialized and, once initialized, its value may not be changed. In addition, we can define compound types, such as pointers or references. A compound type is one that is defined in terms of another type.

The language lets us define our own types by defining classes. The library uses the class facility to provide a set of higher-level abstractions such as the IO and `string` types.

Defined Terms

address Number by which a byte in memory can be found.

alias declaration Defines a synonym for another type: `using` *name* = *type* declares *name* as a synonym for the type *type*.

arithmetic types Built-in types representing boolean values, characters, integers, and floating-point numbers.

array Data structure that holds a collection of unnamed objects that are accessed by an index. Section 3.5 covers arrays in detail.

auto Type specifier that deduces the type of a variable from its initializer.

base type type specifier, possibly qualified by `const`, that precedes the declarators in a declaration. The base type provides the common type on which the declarators in a declaration can build.

bind Associating a name with a given entity so that uses of the name are uses of the underlying entity. For example, a reference is a name that is bound to an object.

byte Smallest addressable unit of memory. On most machines a byte is 8 bits.

class member Part of a class.

compound type A type that is defined in terms of another type.

const Type qualifier used to define objects that may not be changed. `const` objects must be initialized, because there is no way to give them a value after they are defined.

const pointer Pointer that is `const`.

const reference Colloquial synonym for reference to `const`.

constant expression Expression that can be evaluated at compile time.

constexpr Variable that represents a constant expression. § 6.5.2 (p. 239) covers `constexpr` functions.

conversion Process whereby a value of one type is transformed into a value of another type. The language defines conversions among the built-in types.

data member Data elements that constitute an object. Every object of a given class has its own copies of the class' data members. Data members may be initialized when declared inside the class.

declaration Asserts the existence of a variable, function, or type defined elsewhere. Names may not be used until they are defined or declared.

declarator The part of a declaration that includes the name being defined and an optional type modifier.

decltype Type specifier that deduces the type of a variable or an expression.

default initialization How objects are initialized when no explicit initializer is given. How class type objects are initialized is controlled by the class. Objects of built-in type defined at global scope are initialized to 0; those defined at local scope are uninitialized and have undefined values.

definition Allocates storage for a variable of a specified type and optionally initializes the variable. Names may not be used until they are defined or declared.

escape sequence Alternative mechanism for representing characters, particularly for those without printable representations. An escape sequence is a backslash followed by a character, three or fewer octal digits, or an `x` followed by a hexadecimal number.

global scope The scope that is outside all other scopes.

header guard Preprocessor variable used to prevent a header from being included more than once in a single file.

identifier Sequence of characters that make up a name. Identifiers are case-sensitive.

in-class initializer Initializer provided as part of the declaration of a class data member. In-class initializers must follow an `=` symbol or be enclosed inside curly braces.

in scope Name that is visible from the current scope.

initialized A variable given an initial value when it is defined. Variables usually should be initialized.

inner scope Scope that is nested inside another scope.

integral types See arithmetic type.

list initialization Form of initialization that uses curly braces to enclose one or more initializers.

literal A value such as a number, a character, or a string of characters. The value cannot be changed. Literal characters are enclosed in single quotes, literal strings in double quotes.

local scope Colloquial synonym for block scope.

low-level const A `const` that is not top-level. Such `const`s are integral to the type and are never ignored.

member Part of a class.

nonprintable character A character with no visible representation, such as a control character, a backspace, newline, and so on.

null pointer Pointer whose value is 0. A null pointer is valid but does not point to any object.

nullptr Literal constant that denotes the null pointer.

object A region of memory that has a type. A variable is an object that has a name.

outer scope Scope that encloses another scope.

pointer An object that can hold the address of an object, the address one past the end of an object, or zero.

pointer to const Pointer that can hold the address of a `const` object. A pointer to `const` may not be used to change the value of the object to which it points.

preprocessor Program that runs as part of compilation of a C++ program.

preprocessor variable Variable managed by the preprocessor. The preprocessor replaces each preprocessor variable by its value before our program is compiled.

reference An alias for another object.

reference to `const` A reference that may not change the value of the object to which it refers. A reference to `const` may be bound to a `const` object, a nonconst object, or the result of an expression.

scope The portion of a program in which names have meaning. C++ has several levels of scope:

> **global**—names defined outside any other scope
>
> **class**—names defined inside a class
>
> **namespace**—names defined inside a namespace
>
> **block**—names defined inside a block

Scopes nest. Once a name is declared, it is accessible until the end of the scope in which it was declared.

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

signed Integer type that holds negative or positive values, including zero.

string Library type representing variable-length sequences of characters.

struct Keyword used to define a class.

temporary Unnamed object created by the compiler while evaluating an expression. A temporary exists until the end of the largest expression that encloses the expression for which it was created.

top-level const The `const` that specifies that an object may not be changed.

type alias A name that is a synonym for another type. Defined through either a `typedef` or an alias declaration.

type checking Term used to describe the process by which the compiler verifies that the way objects of a given type are used is consistent with the definition of that type.

type specifier The name of a type.

typedef Defines an alias for another type. When `typedef` appears in the base type of a declaration, the names defined in the declaration are type names.

undefined Usage for which the language does not specify a meaning. Knowingly or unknowingly relying on undefined behavior is a great source of hard-to-track run-time errors, security problems, and portability problems.

uninitialized Variable defined without an initial value. In general, trying to access the value of an uninitialized variable results in undefined behavior.

unsigned Integer type that holds only values greater than or equal to zero.

variable A named object or reference. In C++, variables must be declared before they are used.

void* Pointer type that can point to any nonconst type. Such pointers may not be dereferenced.

void type Special-purpose type that has no operations and no value. It is not possible to define a variable of type `void`.

word The natural unit of integer computation on a given machine. Usually a word is large enough to hold an address. On a 32-bit machine a word is typically 4 bytes.

`&` operator Address-of operator. Yields the address of the object to which it is applied.

`*` operator Dereference operator. Dereferencing a pointer returns the object to which the pointer points. Assigning to the result of a dereference assigns a new value to the underlying object.

#define Preprocessor directive that defines a preprocessor variable.

#endif Preprocessor directive that ends an `#ifdef` or `#ifndef` region.

#ifdef Preprocessor directive that determines whether a given variable is defined.

#ifndef Preprocessor directive that determines whether a given variable is not defined.