associativity: A property of an operator that determines how operators with the same precedence are evaluated in an expression.

casting: Data type conversion.

compound assignment: The combination of an assignment operator with another operator to form a single operator.

compound statement: A sequence of statements enclosed in curly braces. Synonym for a block.

constant declaration: The declaration and initialization of a value that can be changed during program execution.

declaration: Mentioning the existence of a variable, function, or type that exists somewhere else.

definition: Allocating storage for objects and optionally initializing them.

explicit type conversion: The conversion of a value from one type to another through the cast operator. Contrast with implicit type conversion.

expression: The smallest unit of computation. An expression is made of one or more operands and one or more operators.

expression statement: An expression terminated by a semicolon.

fixed format: A floating-point value shown as an integer part and a fraction part separated by a decimal point, such as (dddd.ddd).

implicit type change: Changing the type of one operand to the type of the other operand in a binary expression to make them both the same type.

implicit type conversion: A conversion automatically generated by a compiler.

implicit type promotion: The promotion of an operand's type to a larger size.

literal: An unnamed constant coded as a part of an expression.

lvalue: An expression that yields an object.

manipulator: An object that is called to manipulate an input or output stream.

minus expression: An expression prefaced with a minus sign.

multiple declaration: The combination of two or more variables in a declaration statement.

multiplicative expression: An expression that contains a multiply, divide, or modulus operator.

name: The identifier for an object in a program.

null statement: An empty statement indicated by a single semicolon.

overflow: The condition that results when an attempt is made to insert data into a list and there is no room.

plus expression: An expression prefaced with a plus sign.

precedence: A property of an operator that defines which operator should be evaluated next in an expression of more than one operator.

primary expression: An expression consisting of only a single value (no operator); the highest-priority expression.

return statement: The statement in a function that returns an object to the calling function or defines that nothing should be returned.

rvalue: An expression that gives a value but not the address.

scientific format: The format in which a number is shown as a value in fixed format multiplied by an exponent (e.g., 10^n).

showpoint: The manipulator used to display a value with a decimal point and a zero fraction.

simple assignment: As assignment statement in a binary expression with two.

single declaration: The declaration of a variable that specifies only its name and type.

sizeof: An operator that returns the size of its operand in bytes.

statement: A part of the program specifying an action to be performed when the program is executed.

type conversion: The implicit or explicit conversion of a value from one type to another.

unary expression: An expression made of an operator applied to a single value.

underflow: An event that occurs when an attempt is made to delete data from an empty data structure.

variable declaration: The declaration and definition of a variable.

An **expression** is an entity with a value and possibly a side effect. An expression can be made of a simple value or multiple values combined with operators.

A **primary expression** contains one value and no operator.

A **unary expression** contains one value and one operator.

A **binary expression** contains one operator and two values.

A **ternary expression** is made of two operators and three values.

Expression evaluation may involve **type conversion**, which can be **implicit** or **explicit**.

To find the order of evaluation of simple expression in a complex expression, we need to think of two properties of operators: precedence and associativity.

Precedence is used when we have an expression of different levels of precedence.

Associativity is used when we have expressions of the same precedence.

Overflow and **underflow** define the rules for determining what happens when a number is either too large or too small for a variable.

Sometimes we need to **format data**. This is done with the help of predefined objects called manipulators. We have manipulators for outputting data and manipulators for inputting data.

To write a program in C++, we need statements. C++ provides many different types of statements, each with a pre-defined task. We discusses five types of statements in this chapter: declaration statement, expression statement, null statement, compound statement, and return statement.

Program design is a process that needs to be carefully followed using at least three steps: understanding the program, developing algorithms, and writing the code.

CP-1. Can a primary expression have a side effect?

Yes. Every expression is an entity with a value that can change the state of memory (side effect)

CP-2. Explain the purpose of a parenthetical expression.

When we have an expression of a lower level of precedence that we want to change to a primary expression, we enclose it in parentheses.

CP-3. Where does the operator come in a unary expression (before or after the operand)?

The operator comes before the operand.

CP-4. What is the unit of the value when we use the "sizeof" operator?

Bytes.

CP-5. How many operators and how many operands do we have in a binary expression?

There are three operators: *multiplication*, *division*, and *remainder*.

There are two operands: *left* and *right*.

CP-6. Where is the operator in a binary expression located relative to the operands?

Operator is located before the operands. The plus operator does not change the value of its operand but the minus does (flips the value).

CP-7. Does an assignment operator have a side effect? Does a compound assignment operator have a side effect?

In an assignment, the value of right expression is stored in the variable (side effect) before the expression value is returned. Both yes.

CP-8. Define the purpose of implicit promotion and mention the reason that C++ compilers do it.

Implicit type promotion is automatically applied to any operand to make it suitable for an arithmetic operation. Not all data types have define arithmetic operator (Boolean and character). The compiler applies five rules:

Table 3.2 Implicit type promotion

Rule	Original Type	Promoted Type
1	bool	int
2	char	int
3	short	int
3	unsigned short	unsigned int
4	float	double

CP-9. What is the difference between implicit and explicit type conversion?

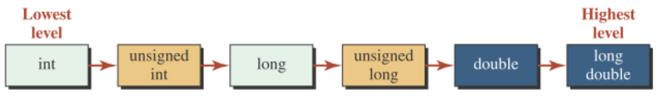

Figure 3.7 Hierarchy of types for implicit conversion

The implicit conversion doesn't require a casting operator. Data type is converted to the higher lever

```
int x = 123;

double y = x;
```

Explicit conversion requires a casting operator. Used to convert data to the lower level.

```
int x = 123;

double y = x;
```

CP-10. What is the use of the "static_cast" type converter?

Used in explicit conversion to convert data type to another level.

```
static_cast <type> (expression)
```

CP-11. In evaluating an expression, which properties do we need to consider first: precedence or associativity?

Precedence first. In the case where the expression has more than one expression at the same level of precedence we need to use the associativity of the simple expression.

CP-12. Which property (precedence or associativity) has several levels?

Precedence has several levels.

CP-13. Which property (precedence or associativity) is based on direction (left-to-right or right-to-left)?

Associativity.

CP-14. What happens if you use a floating-point value less than +min or greater than –min in a program?

All floating-point values are signed. There is no wrapping when overflow and underflow occurs; instead we have sinking. Underflow and Overflow turns into -infinity and +infinity respectively.

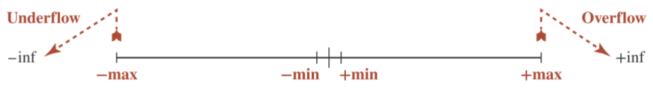

Figure 3.10 Overflow and underflow in floating-point values

CP-15. Which output manipulator can be used only for the Boolean data type? Which can be used for all data types?

Table 3.3 No-argument manipulators for output stream

Manipulator	*Boolean*	*Character*	*Integer*	*Floating point*
endl	√	√	√	√
noboolalpha, boolalpha	√			
dec, oct, hex			√	
noshowbase, showbase			√	
(none), fixed, scientific				√
noshowpoint, showpoint				√
noshowpos, showpos			√	√
nouppercase, uppercase			√	√
left, right, internal	√	√	√	√

CP-16. Can we use "setprecision (n)" without using "setw (n)"?

Table 3.4 One-argument manipulators for output stream

Manipulator	*Boolean*	*Character*	*Integer*	*Floating point*
setprecision (n)				√
setw (n)	√	√	√	√
setfill (ch)	√	√	√	√

setprecision (n) defines the number of digits after the decimal point, hence can be used without setw(n)

CP-17. Name two manipulator sets used for input streams.

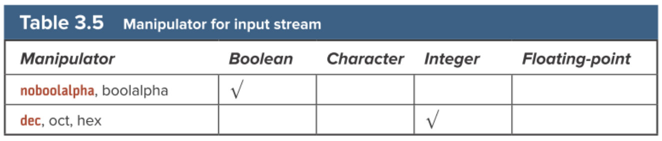

Table 3.5 Manipulator for input stream

Manipulator	*Boolean*	*Character*	*Integer*	*Floating-point*
noboolalpha, boolalpha	√			
dec, oct, hex			√	

CP-18. Does every statement type in C++ need a semicolon at the end?

Some statements need a semicolon at the end as the terminator, but some already have a built-in terminator.

CP-19. Does a compound statement need a semicolon at the end?

We do not need a semicolon at the end of a compound statement; the closing brace serves as the terminator.

CP-20. What will be stored in a local variable if it is not initialized when declared?

If a variable is local (declared inside a function), it is not initialized, but the variable holds some garbage left over from the previous use.

CP-21. What is the purpose of a compound statement?

Used to treat several statements as a single statement. We can combine any number of statements (zero or more) inside a pair of braces. It also improves readability and avoid confusion.

CP-22. Does the main function need a compound statement?

Yes.