4.x — Chapter 4 summary and quiz

learncpp.com/cpp-tutorial/chapter-4-summary-and-quiz/

Chapter Review

The smallest unit of memory is a **binary digit**, also called a **bit**. The smallest unit amount of memory that can be addressed (accessed) directly is a **byte**. The modern standard is that a byte equals 8 bits.

A **data type** tells the compiler how to interpret the contents of memory in some meaningful way.

C++ comes with support for many fundamental data types, including floating point numbers, integers, boolean, chars, null pointers, and void.

Void is used to indicate no type. It is primarily used to indicate that a function does not return a value.

Different types take different amounts of memory, and the amount of memory used may vary by machine.

Related content

See 4.3 -- Object sizes and the sizeof operator for a table indicating the minimum size for each fundamental type.

The **sizeof** operator can be used to return the size of a type in bytes.

Signed integers are used for holding positive and negative whole numbers, including 0. The set of values that a specific data type can hold is called its **range**. When using integers, keep an eye out for overflow and integer division problems.

Unsigned integers only hold positive numbers (and 0), and should generally be avoided unless you're doing bit-level manipulation.

Fixed-width integers are integers with guaranteed sizes, but they may not exist on all architectures. The fast and least integers are the fastest and smallest integers that are at least some size. `std::int8_t` and `std::uint8_t` should generally be avoided, as they tend to behave like chars instead of integers.

size_t is an unsigned integral type that is used to represent the size or length of objects.

Scientific notation is a shorthand way of writing lengthy numbers. C++ supports scientific notation in conjunction with floating point numbers. The digits in the significand (the part before the e) are called the **significant digits**.

Floating point is a set of types designed to hold real numbers (including those with a fractional component). The **precision** of a number defines how many significant digits it can represent without information loss. A **rounding error** can occur when too many significant digits are stored in a floating point number that can't hold that much precision. Rounding errors happen all the time, even with simple numbers such as 0.1. Because of this, you shouldn't compare floating point numbers directly.

The **Boolean** type is used to store a `true` or `false` value.

If statements allow us to execute one or more lines of code if some condition is true. The conditional expression of an if-statement is interpreted as a boolean value. An **else statement** can be used to execute a statement when a prior if-statement condition evaluates to false.

Char is used to store values that are interpreted as an ASCII character. When using chars, be careful not to mix up ASCII code values and numbers. Printing a char as an integer value requires use of `static_cast.`

Angled brackets are typically used in C++ to represent something that needs a parameterizable type. This is used with `static_cast` to determine what data type the argument should be converted to (e.g. `static_cast<int>(x)` will return the value of `x` as an `int`).

Quiz time

Question #1

Pick the appropriate data type for a variable in each of the following situations. Be as specific as possible. If the answer is an integer, pick int (if size isn't important), or a specific fixed-width integer type (e.g. std::int16_t) based on range.

a) The age of the user (in years) (assume the size of the type isn't important)

Show Solution

b) Whether the user wants the application to check for updates

Show Solution

c) pi (3.14159265)

Show Solution

d) The number of pages in a textbook (assume size is not important)

Show Solution

e) The length of a couch in feet, to 2 decimal places (assume size is important)

Show Solution

f) How many times you’ve blinked since you were born (note: answer is in the millions)

Show Solution

g) A user selecting an option from a menu by letter

Show Solution

h) The year someone was born (assuming size is important)

Show Solution

Question #2

Author’s note

The quizzes get more challenging starting here. These quizzes that ask you to write a program are designed to ensure you can integrate multiple concepts that have been presented throughout the lessons. You should be prepared to spend some time with these problems. If you’re new to programming, you shouldn’t expect to be able to answer these immediately.

Remember, the goal here is to help you pinpoint what you know, and which concepts you may need to spend additional time on. If you find yourself struggling a bit, that’s okay.

Here are some tips:

- Don’t try to write the whole solution at once. Write one function, then test it to make sure it works as expected. Then proceed.
- Use your debugger to help figure out where things are going wrong.
- Go back and review the answers to quizzes from prior lessons in the chapter, as they’ll often contain similar concepts.

If you are truly stuck, feel free to look at the solution, but take the time to make sure you understand what each line does before proceeding. As long as you leave understanding the concepts, it doesn’t matter so much whether you were able to get it yourself, or had to look at the solution before proceeding.

Write the following program: The user is asked to enter 2 floating point numbers (use doubles). The user is then asked to enter one of the following mathematical symbols: +, -, *, or /. The program computes the answer on the two numbers the user entered and prints the results. If the user enters an invalid symbol, the program should print nothing.

Example of program:

```
Enter a double value: 6.2
Enter a double value: 5
Enter +, -, *, or /: *
6.2 * 5 is 31
```

Show Hint

Show Hint

Show Solution

Question #3

Extra credit: This one is a little more challenging.

Write a short program to simulate a ball being dropped off of a tower. To start, the user should be asked for the height of the tower in meters. Assume normal gravity (9.8 m/s^2), and that the ball has no initial velocity (the ball is not moving to start). Have the program output the height of the ball above the ground after 0, 1, 2, 3, 4, and 5 seconds. The ball should not go underneath the ground (height 0).

Use a function to calculate the height of the ball after x seconds. The function can calculate how far the ball has fallen after x seconds using the following formula: distance fallen = gravity_constant * x_seconds^2 / 2

Expected output:

```
Enter the height of the tower in meters: 100
At 0 seconds, the ball is at height: 100 meters
At 1 seconds, the ball is at height: 95.1 meters
At 2 seconds, the ball is at height: 80.4 meters
At 3 seconds, the ball is at height: 55.9 meters
At 4 seconds, the ball is at height: 21.6 meters
At 5 seconds, the ball is on the ground.
```

Note: Depending on the height of the tower, the ball may not reach the ground in 5 seconds - - that's okay. We'll improve this program once we've covered loops.
Note: The `^` symbol isn't an exponent in C++. Implement the formula using multiplication instead of exponentiation.
Note: Remember to use double literals for doubles, e.g. `2.0` rather than `2`.

Show Solution