4.4 — Signed integers

ALEX AUGUST 5, 2021

An **integer** is an integral type that can represent positive and negative whole numbers, including 0 (e.g. -2, -1, 0, 1, 2). C++ has *4* different fundamental integer types available for use:

Type	Minimum Size	Note
short	16 bits	
int	16 bits	Typically 32 bits on modern architectures
long	32 bits	
long long	64 bits	

The key difference between the various integer types is that they have varying sizes -- the larger integers can hold bigger numbers.

A reminder

C++ only guarantees that integers will have a certain minimum size, not that they will have a specific size. See lesson 4.3 -- Object sizes and the sizeof operator for information on how to determine how large each type is on your machine.

Signed integers

When writing negative numbers in everyday life, we use a negative sign. For example, *-3* means "negative 3". We'd also typically recognize *+3* as "positive 3" (though common convention dictates that we typically omit plus prefixes). This attribute of being positive, negative, or zero is called the number's **sign**.

By default, integers are **signed**, which means the number's sign is stored as part of the number (using a single bit called the **sign bit**). Therefore, a signed integer can hold both positive and negative numbers (and 0).

In this lesson, we'll focus on signed integers. We'll discuss unsigned integers (which can only hold non-negative numbers) in the next lesson.

Related content

We discuss how the sign bit is used when representing numbers in binary in lesson O.4 -- Converting between binary and decimal.

Defining signed integers

Here is the preferred way to define the four types of signed integers:

```
short s;
int i;
long l;
long long ll;
```

All of the integers (except int) can take an optional *int* suffix:

```
short int si;
long int li;
long long int lli;
```

This suffix should not be used. In addition to being more typing, adding the *int* suffix makes the type harder to distinguish from variables of type *int*. This can lead to mistakes if the short or long modifier is inadvertently missed.

The integer types can also take an optional *signed* keyword, which by convention is typically placed before the type name:

```
signed short ss;
signed int si;
signed long sl;
signed long long sll;
```

However, this keyword should not be used, as it is redundant, since integers are signed by default.

Best practice

Prefer the shorthand types that do not use the *int* suffix or signed prefix.

Signed integer ranges

As you learned in the last section, a variable with *n* bits can hold 2^n possible values. But which specific values? We call the set of specific values that a data type can hold its **range**. The range of an integer variable is determined by two factors: its size (in bits), and whether it is signed or not.

By definition, an 8-bit signed integer has a range of -128 to 127. This means a signed integer can store any integer value between -128 and 127 (inclusive) safely.

As an aside...

Math time: an 8-bit integer contains 8 bits. 2^8 is 256, so an 8-bit integer can hold 256 possible values. There are 256 possible values between -128 to 127, inclusive.

Here's a table containing the range of signed integers of different sizes:

Size/Type	Range
8 bit signed	-128 to 127
16 bit signed	-32,768 to 32,767
32 bit signed	-2,147,483,648 to 2,147,483,647
64 bit signed	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

For the math inclined, an n-bit signed variable has a range of $-(2^{n-1})$ to $2^{n-1}-1$.

For the non-math inclined… use the table. :)

Integer overflow

What happens if we try to assign the value *280* to an 8-bit signed integer? This number is outside the range that a 8-bit signed integer can hold. The number 280 requires 9 bits (plus 1 sign bit) to be represented, but we only have 7 bits (plus 1 sign bit) available in a 8-bit signed integer.

Integer overflow (often called *overflow* for short) occurs when we try to store a value that is outside the range of the type. Essentially, the number we are trying to store requires more bits to represent than the object has available. In such a case, data is lost because the object doesn't have enough memory to store everything.

In the case of signed integers, which bits are lost is not well defined, thus signed integer overflow leads to undefined behavior.

> **Warning**
>
> Signed integer overflow will result in undefined behavior.

In general, overflow results in information being lost, which is almost never desirable. If there is *any* suspicion that an object might need to store a value that falls outside its range, use a type with a bigger range!

Integer division

When dividing two integers, C++ works like you'd expect when the quotient is a whole number:

```
#include <iostream>

int main()
{
    std::cout << 20 / 4;
    return 0;
}
```

This produces the expected result:

```
5
```

But let's look at what happens when integer division causes a fractional result:

```
#include <iostream>

int main()
{
    std::cout << 8 / 5;
    return 0;
}
```

This produces a possibly unexpected result:

```
1
```

When doing division with two integers (called**integer division**), C++ always produces an integer result. Since integers can't hold fractional values, any fractional portion is simply dropped (not rounded!).

Taking a closer look at the above example, 8 / 5 produces the value 1.6. The fractional part (0.6) is dropped, and the result of 1 remains.

Similarly, -8 / 5 results in the value -1.

Warning

Be careful when using integer division, as you will lose any fractional parts of the quotient. However, if it's what you want, integer division is safe to use, as the results are predictable.

If fractional results are desired, we show a method to do this in lesson 5.2 -- Arithmetic operators.

Next lesson

4.5 Unsigned integers, and why to avoid them

Back to table of contents

Previous lesson

4.3 Object sizes and the sizeof operator

Leave a comment... Put C++ code between triple-backticks (markdown style):```Your C++ cod

Name*

Email*

Avatars from https://gravatar.com/ are connected to your provided email address.

Notify me about replies: POST COMMENT

DP N N FOUT

Newest

X

X