

# Data Representations

---

The ANSI C Standard (ANSI document X3.159) requests that each C implementation provide information about various “system specifics,” including how data types are represented. This chapter describes how the High C/C++ compiler implements various data types. It contains the following sections:

§9.1: *Data Types*

§9.2: *Registers*

§9.3: *Unsigned Expressions in Conditional Preprocessing Directives*

---

## 9.1 Data Types

This section describes how data types are implemented. See Table 10.1, *Data Types: Size and Alignment* for more information.

---

### 9.1.1 Characters

Source and execution character sets are both standard ASCII. Each character in the source character set maps into the identical character in the execution character set. Without exception, all character constants map into some value in the execution character set.

A character is stored in a byte, and there are four bytes in an `int`.

In C, a character constant is of type `int`. The ANSI C Standard requires the type to remain `int`, even if truncation occurs.

The type specifier `char`, when not accompanied by an adjective, denotes an unsigned character type. However, you can change this by turning `OFF` the toggle `Char_default_unsigned`. (See Chapter 4: *Using Compiler Toggles*.)

### 9.1.1.1 Multi-Character Constants

High C/C++ permits a character constant that contains more than one character. This is the value of a two-character constant:

$$256 * (\text{value of the first character}) + (\text{value of the second character})$$

For example, the value of `'ab'` is  $256 * 'a' + 'b'$ .

In general, the value of an  $n$ -character constant is recursively defined as:

$$256 * (\text{value of the first } n-1 \text{ characters}) + (\text{value of the last character})$$

---

## 9.1.2 Integers

Integers are represented in twos-complement binary form. Various integer types are restricted to certain ranges of values.

Table 9.1 shows these ranges of values.

**Table 9.1** *Integers: Range of Values*

Type	Minimum Value	Maximum Value
<code>signed char</code>	-128	127
<code>unsigned char</code>	0	255
<code>short</code>	-32,768	32,767
<code>unsigned short</code>	0	65,535
<code>int</code>	-2,147,483,648	2,147,483,647
<code>unsigned int</code>	0	4,294,967,295
<code>long</code>	-2,147,483,648	2,147,483,647
<code>unsigned long</code>	0	4,294,967,295
<code>long long</code>	$-2^{63}$	$2^{63} - 1$
<code>unsigned long long</code>	0	$2^{64} - 1$

---

**Note:** If you are using a system library that does not support `long long` and `unsigned long long`, turn Off toggle `Long_long_8` to map these data types to `long`.

---

<i>Integer conversion</i>	An integer is converted to a shorter signed integer, or <code>int</code> bit field, by truncation of the most significant bytes.
<i>Unsigned integer conversion</i>	An unsigned integer is converted to a signed integer of the same size by transferring the bits from one bit field to the other. All of the bits are retained. If the value of an unsigned integer is larger than the largest positive signed integer value, it is interpreted bit by bit and yields a negative value.
<i>Bitwise operation on a signed integer</i>	The result of a bitwise operation on a signed integer is the same as if the integer were treated as unsigned.
<i>Integer division</i>	The sign of the remainder on integer division is the same as the sign of the dividend.
<i>Right shift of a signed integer</i>	The right shift of a signed integral type is arithmetic; that is, the sign bit is propagated to the right.

### 9.1.3 Floating-Point Numbers

High C/C++ uses the IEEE Standard 754 formats to represent floating-point data. Each floating-point data type has a sign bit, exponent bits, and mantissa bits. Table 9.2 summarizes this information.

**Table 9.2** *Size of Floating-Point Data Types*

Precision	Sign Bit	Exponent Bits	Mantissa Bits	Minimum Value (Absolute Value)	Maximum Value (Absolute Value)
<code>float</code>	1	8	23	$8.43 \times 10^{-37}$	$3.37 \times 10^{+38}$
<code>double</code>	1	11	52	$4.19 \times 10^{-307}$	$1.67 \times 10^{+308}$
<code>long double</code>	1	15	112	$3.362 \times 10^{-4932}$	$1.89 \times 10^{+4932}$

The default rounding mode is “round to nearest.” See §10.1: *Size and Alignment of Data Types* for the length required for each floating-point type.

When a negative floating-point number is truncated to an integral type, the truncation is toward 0 (zero). Thus  $-2.7$  is truncated to  $-2$ , while  $-1.2$  is truncated to  $-1$ .

For details about the IEEE data format, refer to the ANSI/IEEE Standard 754/1985 documents.

---

### 9.1.4 Pointers

The difference of two pointers is of type `ptrdiff_t`, which is defined as type `int` in `stddef.h`.

The type of the value returned by `sizeof` is `int`.

---

### 9.1.5 Structures, Unions, and Bit Fields

Signed and unsigned bit fields are supported.

- A bit field declared as `int` is treated as `unsigned int`.
- A bit field declared as `signed int` is interpreted as `signed`.

For more information about structures, unions, and bit fields, see §10.1.2: *Size and Alignment of struct and union Types* and §10.3.1: *How the Compiler Maps Bit Fields*.

---

## 9.2 Registers

The compiler attempts to map each auto- or register-class variable to a register if its type is appropriate and its address is not taken.

For more information, see §10.6: *How the Compiler Maps Variables to Storage*.

---

## 9.3 Unsigned Expressions in Conditional Preprocessing Directives

A single-character constant in a constant expression controlling conditional inclusion is always non-negative in value, ranging from 0 (zero) to 255.