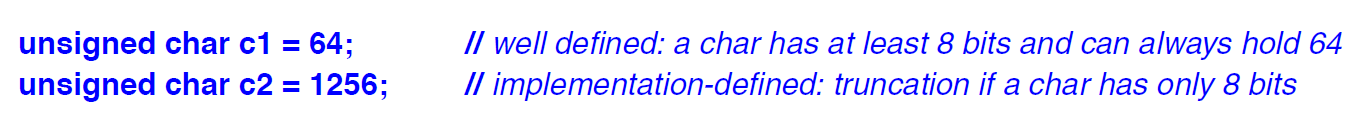
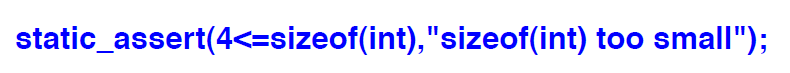
* Many important things are deemed as implementation-defined by the standard. For example,

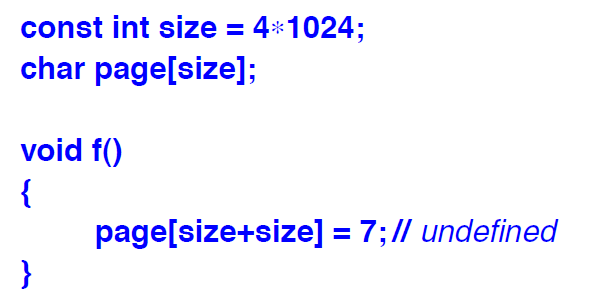


If the char has only 8 bits, then 1256 will be converted to 232. (No idea how?)

* Many assumptions about implementation-defined features can be checked by stating them as static assertions. For example,

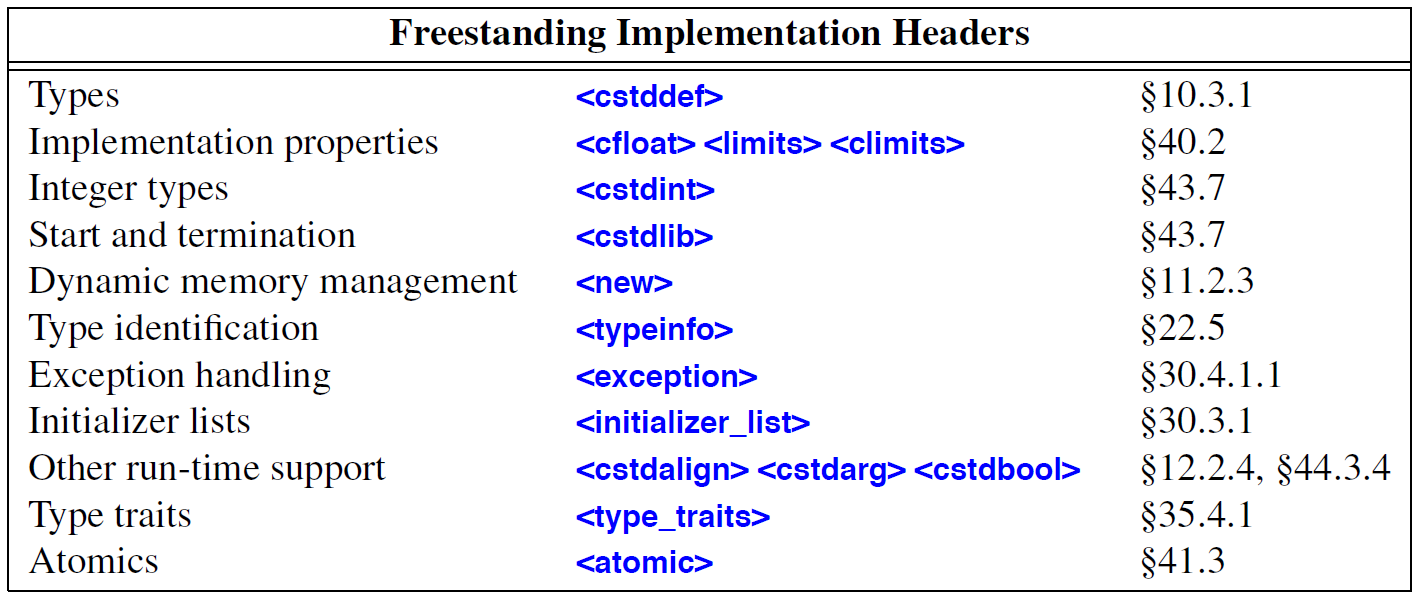


* A construct is deemed undefined by the standard if no reasonable behaviour is required by the implementation. For example,

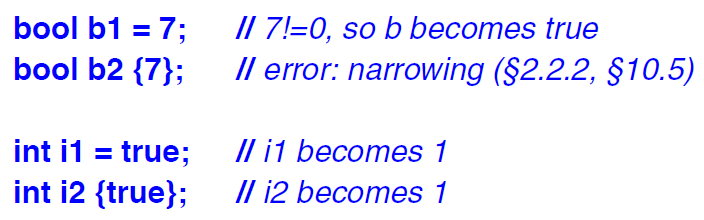


Plausible outcomes of this code fragment include overwriting unrelated data and triggering a hardware error/exception.

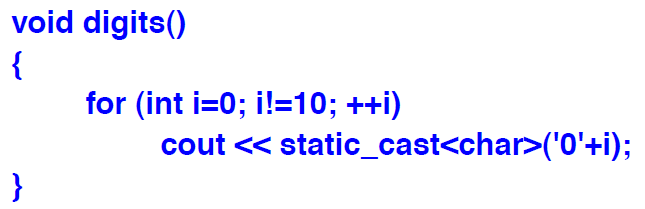
An implementation is not required to choose among plausible outcomes.



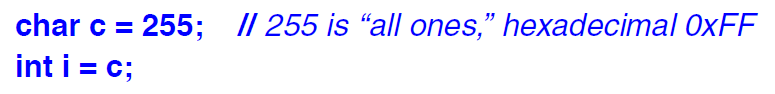
* C++ has a set of fundamental types –
* Boolean type (bool)
* Character types (char, wchar\_t )
* Integer types (int, long long)
* Floating point types (double, long double)
* Type to signify absence of information (void)
* Other types, constructed from the fundamental types, using declarator operators –
* Pointer types (int\*)
* Array types (char[])
* Reference types (double&, vector<int>&&)
* User defined additional types –
* Data structures and classes
* Enumeration types for representing specific sets of values (enum, enum class)
* **Integral types:** Boolean, character, integer types.
* **Arithmetic types:** Integral and floating-point types.
* **User-defined types:** Must be defined by users rather than being available for use without previous declaration, e.g. enums and classes.
* **Built-in types:** Fundamental types, pointers and references.
* **Booleans:** By definition, true has the value 1 when converted to integer, false has the value 0. Conversely, integers can be converted to bool values: non-zero integers convert to true and zero integers convert to false.



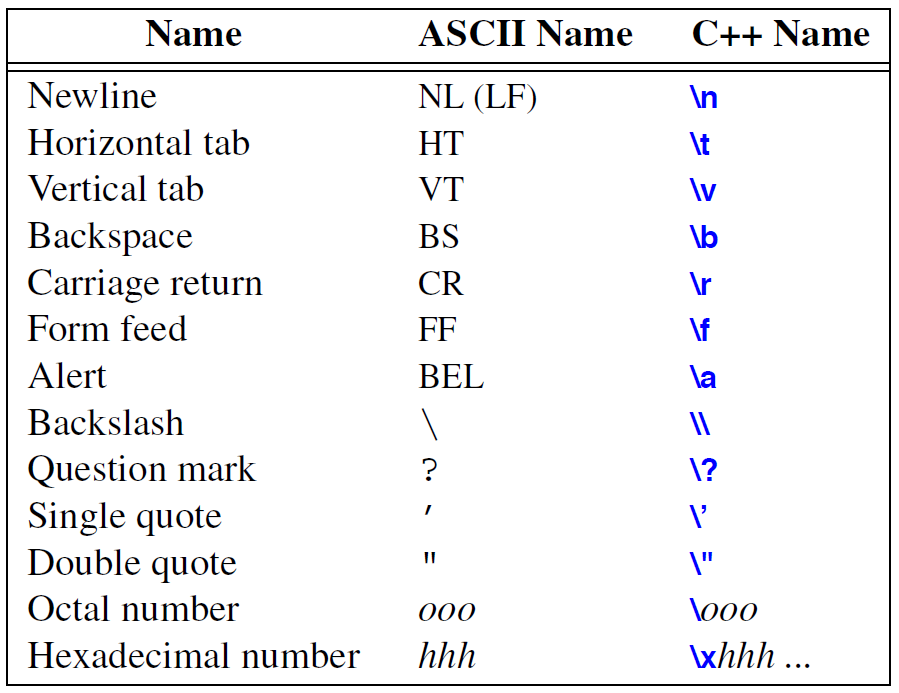
* **Character types:** C++ offers a variety of character types –
* *char*: Default character type. Usually 8 bits.
* *signed* *char*: Capable of holding both positive and negative values.
* *unsigned* *char*: A char that is guaranteed to be unsigned.
* *wchar\_t:* Can hold characters of a larger character set such as Unicode. Its size is implementation dependent.
* *char16\_t*: Holds 16-bit character sets, such as UTF-16.
* *char32\_t*: Holds 32-bit character sets, such as UTF-32.
* The following is an example of the use of *static\_cast –*



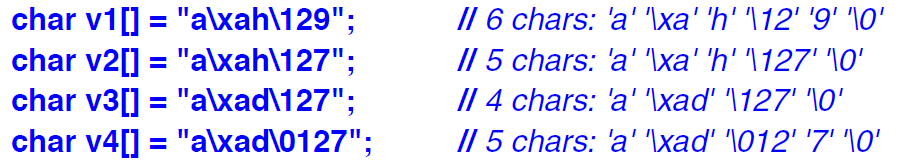
* The above is a program to print the 10 integer numbers, i.e. 0, 1, 2 and so on.
* By leaving out the *static\_cast*, the output will be 48, 49, 50 and so on.
* **Signed and unsigned chars:** It is implementation dependent whether a plain char is considered as signed or unsigned. This leads to confusions, e.g. –



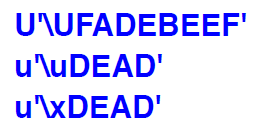
* On a machine where a char is unsigned, the answer is 255.
* On a machine where a char is signed, the answer is -1.
* Pointers of the types char, unsigned char and signed char cannot be mixed.
* Variables of the three char types can be freely assigned to one another. However, assigning too large value to a signed char is undefined.
* None of the potential problems and confusions arise if plain char is used throughout and negative values are not used.
* **Character literals:** A single character enclosed in single quotes, e.g. ‘a’, ‘B’, ‘4’. The type of a character literal is char.
* Use of character literals rather than decimal notations make programs more portable.
* The following are special characters. Despite their appearance, they are considered as single characters.



* A sequence of octal or hexadecimal digits is terminated by the first character that is not an octal or hexadecimal digit respectively.



* For octal constants, always use 3 digits to represent a number.
* For hexadecimal constants, always use 2 digits to represent a number.
* **Wide character literals** are of the form L’ab’ and are of type wchar\_t. The number of characters between the quotes and their meanings are implementation-defined.
* Literals of larger character sets, such as **Unicode,** are presented as sequences of 4 or 8 hexadecimal digits preceded by a U or a u, e.g.



* A number of hexadecimal digits different from 4 or 8 is a lexical error.
* **Universal character names:** The values of hexadecimal number, defined by the ISO/IEC 10646 standard.
* **Integer types:** Integers come in 3 forms –
* plain int, referred to as ‘int’
* signed int, referred to as ‘signed’
* unsigned int, referred to as 'unsigned’
* **Integer sizes:** Integers come in 4 sizes –
* short int, referred to as ‘short’
* plain int, referred to as ‘int’
* long int, referred to as ‘long’
* long long int, referred to as 'long long’
* Plain ints are always signed.
* To get more detailed control over integer sizes, the following aliases from <cstdint> can be used –
* *int64\_t*: A signed integer with exactly 64 bits.
* *uint\_fast16\_t*: An unsigned integer with exactly 16 bits, supposedly the fastest such integer.
* *int\_least32\_t*: A signed integer with at least 32 bits, just like plain int.
* *extended integer types*: Behave exactly like integers but usually have greater range and occupy more space.