The New C Standard (Excerpted material)

An Economic and Cultural Commentary

Derek M. Jones
derek@knosof.co.uk

5.2.2 Character display semantics{ref character display semantics}

Commentary

There is no guarantee that a character display will exist on any hosted implementation. If such a device is supported by an implementation, this clause specifies its attributes.

C++

Clause 18 mentions "display as a wstring" in *Notes:*. But, there is no other mention of display semantics anywhere in the standard.

Common Implementations

termcap database

Most Unix-based environments contain a database of terminal capabilities, the so-called *termcap* database.[1] This database provides information to the host on a large number of terminal capabilities and characteristics. Knowing the display device currently being used (this usually relies on the user setting an environment variable) enables the database to be queried for device attribute information. This information can then be used by an application to handle its output to display devices. There is a similar database of information on printer characteristics.

The *active position* is that location on a display device where the next character output by the **fputc** function would appear. 252

Commentary

This defines the term *active position*; however, the term *current cursor position* is more commonly used by developers.

The wide character output functions act as if `fputc` is called.

C++

C++ has no concept of active position. The `fputc` function appears in "Table 94" as one of the functions supported by C++.

Other Languages

Most languages don't get involved in such low-level I/O details.

The intent of writing a printing character (as defined by the **isprint** function) to a display device is to display a graphic representation of that character at the active position and then advance the active position to the next position on the current line. 253

Commentary

The standard specifies an intent, not a requirement. Some devices produce output that cannot be erased later (e.g., printing to paper) while other devices always display the last character output at a given position (e.g., VDUs). The ability of printers to display two or more characters at the same position is sometimes required. For instance, programs wanting to display the **ô** character on a wide variety of printers might generate the sequence **o**, backspace, **^** (all of these characters are contained in the invariant subset of ISO 646).

The intended behavior describes the movement of the active position, not the width of the character displayed. There is nothing in this definition to prevent the writing of one character affecting previously written characters (which can occur in Arabic). This specification implies that the positions are a fixed width apart.

glyph

The graphic representation of a character is known as a *glyph*.

C++

The C++ Standard does not discuss character display semantics.

Common Implementations

In some oriental languages, character glyphs can usually be organized into two groups, one being twice the width as the other. Implementations in these environments often use a fixed width for each glyph, creating empty spaces between some glyph pairs.

Some orthographies, which use an alphabetic representation, contain single characters that use what appears to be two characters in their visual representation. For instance, the character denoted by the Unicode value U00C6 is *Æ*, and the character denoted by the Unicode value U01C9 is *lj*. Both representations are considered to be a single character (the former is also a single letter, while the latter is two letters).

Coding Guidelines

The concept of active position is useful for describing the basic set of operations supported by the C Standard. The applications' requirements for displaying characters may, or may not, be feasible within the functionality provided by the standard; this is a top-level application design issue. How characters appear on a display device is an application user interface issue that is outside the scope of this book.

---

254 The direction of writing is locale-specific.

writing direction locale-specific

Commentary

Although left-to-right is used by many languages, this direction is not the only one used. Arabic uses right-to-left (also Hebrew, Urdu, and Berber). In Japanese it is possible for the direction to be from top to bottom with the lines going right-to-left (mainland Chinese has the columns going from left-to-right, in Taiwan it goes right-to-left), or left-to-right with the lines going top to bottom (the same directional conventions as English)

There is no requirement that the direction of writing always be the same direction, for instance, braille alternates in direction between adjacent lines (known as *boustrophedron*), as do Egyptian hieroglyphs, Mayan, and Hittite. Some Egyptian hieroglyphic characters can face either to the left or right (e.g.,  or ), information that readers can use to deduce the direction in which a line should be read.

Some applications need to simultaneously handle locales where the direction of writing is different, for instance, a word processor that supports the use of Hebrew and English in the same document. This level of support is outside the scope of the C Standard.

C++

The C++ Standard does not discuss character display semantics.

Coding Guidelines

The direction of writing is an application issue. Any developer who is concerned with the direction of writing will, of necessity, require a deeper involvement with this topic than the material covered by the C Standard or these coding guidelines.

Example

The direction of writing can change during program execution. For instance, in a word processor that handles both English and Arabic or Hebrew, the character sequence *ABCdefGHJ* (using lowercase to represent English and uppercase to represent Arabic/Hebrew) might appear on the display as **JHGdefCBA**.

---

255 If the active position is at the final position of a line (if there is one), the behavior of the display device is unspecified.

Commentary

The Committee recognized that there is no commonality of behavior exhibited by existing display devices when the final position on a line is reached.

C++

The C++ Standard does not discuss character display semantics.

Common Implementations

Some display devices wrap onto the next line, effectively generating an extra new-line character. Other devices write all subsequent characters, up to the next new-line character, at the final position. On some displays, writing to the bottom right corner of a display has an effect other than displaying the character

output, for instance, clearing the screen or causing it to scroll. The `termcap` and `ncurses` both provide configuration options that specify whether writing to this display location has the desired effect.

Coding Guidelines

Organizing the characters on a display device is an application domain issue. The fact that the C Standard does not provide a defined method of handling the situation described here needs to be dealt with, if applicable, during the design process. This is outside the scope of these coding guidelines.

Alphabetic escape sequences representing nongraphic characters in the execution character set are intended to produce actions on display devices as follows: 256

Commentary

This is the behavior of Ascii terminals enshrined in the C Standard.

Rationale To avoid the issue of whether an implementation conforms if it cannot properly effect vertical tabs (for instance), the Standard emphasizes that the semantics merely describe *intent*.

These escape sequences can also be output to files. The data values written to a file may depend on whether the stream was opened in text or binary mode.

C++

The C++ Standard does not discuss character display semantics.

Other Languages

Java provides a similar set of functionality to that described here.

Common Implementations

Most display devices are capable of handling most of the functions described here.

Coding Guidelines

A program cannot assume that any of the functionality described will occur when the escape sequence is sent to a display device. The root cause for the variability in support for the intended behaviors is the variability of the display devices. In most cases an implementation's action is to send the binary representation of the escape sequence to the device. The manufacturers of display devices are aware of their customers expectations of behavior when these kinds of values are received.

There is little that coding guidelines can recommend to help reduce the dependency on display devices. The design guidelines of creating individual functions to perform specific operations on display devices and isolating variable implementation behaviors in one place are outside the scope of these coding guidelines.

\a (*alert*) Produces an audible or visible alert without changing the active position. 257

Commentary

The intent of an alert is to draw attention to some important event, such as a warning message that the host is to be shut down, or that some unexpected situation has occurred. A program running as a background process (a concept that is not defined by the C Standard) may not have a display device attached (does a tree falling in a forest with nobody to hear it make a noise?).

C++

Alert appears in Table 5, 2.13.2p3. There is no other description of this escape sequence, although the C behavior might be implied from the following wording:

17.4.1.2p3 *The facilities of the Standard C Library are provided in 18 additional headers, as shown in Table 12:*

Common Implementations

Most implementations provide an audible alert. On display devices that don't have a mechanism for producing a sound, a visible alert might be to temporarily blank the screen or to temporarily increase the brightness of the screen.

Coding Guidelines

Programs that produce too many alerts run the risk of having them ignored. The human factor involved in producing alerts are outside of the scope of these coding guidelines. Issues such as a display device not being able to produce an audible alert because its speaker is broken, is also outside the scope of these coding guidelines.

---

258 **\b** (*backspace*) Moves the active position to the previous position on the current line.

backspace escape sequence

Commentary

The standard specifies that the active position is moved. It says nothing about what might happen to any character displayed prior to the backspace at the new current active position.

Common Implementations

Some devices erase any character displayed at the previous position.

C++

Backspace appears in Table 5, 2.13.2p3. There is no other description of this escape sequence, although the C behavior might be implied from the following wording:

The facilities of the Standard C Library are provided in 18 additional headers, as shown in Table 12: 17.4.1.2p3

Example

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5  printf("h\bHello \b World\n");
6  }
```

---

259 If the active position is at the initial position of a line, the behavior of the display device is unspecified.

Commentary

Some terminals have input locking states. In such cases an unspecified behavior put the display device into a state where it no longer displays characters written to it.

C90

If the active position is at the initial position of a line, the behavior is unspecified.

This wording differs from C99 in that it renders the behavior of the program as unspecified. The program simply writes the character; how the device handles the character is beyond its control.

C++

The C++ Standard does not discuss character display semantics.

Common Implementations

The most common implementation behavior is to ignore the request leaving the active position unchanged. Some VDUs have the ability to wrap back to the final position on the preceding line.

Coding Guidelines

While it may be technically correct to specify that the behavior of the display device as unspecified, it does indirectly affect the output behavior of a program in that subsequent output may not appear on that display device.

\f (*form feed*) Moves the active position to the initial position at the start of the next logical page. 260

Commentary

page logical

Whatever a page, logical or otherwise, is. This concept is primarily applied to printers. The functionality to move to the start of the next page, from anywhere on the current page, is generally provided by printer vendors. Programs might use this functionality since it frees them from needing to know the number of lines on a page (provided the minimum needed to support the generated output is available).

C++

Form feed appears in Table 5, 2.13.2p3. There is no other description of this escape sequence, although the C behavior might be implied from the following wording:

17.4.1.2p3 *The facilities of the Standard C Library are provided in 18 additional headers, as shown in Table 12:*

Coding Guidelines

Use of this escape sequence could remove the need for a program to be aware of the number of lines on the page of the display device being written. However, it does place a dependency on the characteristics of the display device being known to the host executing the program, or on the device itself, to respond to the data sent to it.

termcap database

---

new-line escape sequence

\n (*new line*) Moves the active position to the initial position of the next line. 261

Commentary

What happens to the preceding lines is not specified. For instance, whether the display device scrolls lines or wraps back to the top of any screen. The standard is silent on the issue of display devices that only support one line. For instance, do the contents of the previous line disappear?

C++

New line appears in Table 5, 2.13.2p3. There is no other description of this escape sequence, although the C behavior might be implied from the following wording:

17.4.1.2p3 *The facilities of the Standard C Library are provided in 18 additional headers, as shown in Table 12:*

Other Languages

Some languages provide a library function that produces the same effect.

Common Implementations

On some hosts the new-line character causes more than one character to be sent to the display device (e.g., carriage return, line feed).

A printing device may simply move the media being printed on. A VDU may display characters on some previous line (wrapping to the start of the screen). On some display devices (usually memory-mapped ones), the start of a new line is usually indicated by an end-of-line character appearing at the end of the previous line. On other display devices, a fixed amount of storage is allocated for the characters that may occur on each line. In this case the end of line is not stored as a character in the display device.

end-of-line representation

Coding Guidelines

Issues, such as handling lines that are lost when a new line is written or display devices that contain a single line, are outside the scope of these coding guidelines.

---

262 `\r` (*carriage return*) Moves the active position to the initial position of the current line.

carriage return escape sequence

Commentary

The behavior might be viewed as having the same effect as writing the appropriate number of backspace characters. However, the effect of writing a backspace character might be to erase the previous character, while a carriage return does not cause the contents of a line to be erased. Like backspace, the standard says nothing about the effect of writing characters at the position on a line that has previously been written to.

258 backspace escape sequence

C++

Carriage return appears in Table 5, 2.13.2p3. There is no other description of this escape sequence, although the C behavior might be implied from the following wording:

> *The facilities of the Standard C Library are provided in 18 additional headers, as shown in Table 12:* 17.4.1.2p3

---

263 `\t` (*horizontal tab*) Moves the active position to the next horizontal tabulation position on the current line.

horizontal tab escape sequence

Commentary

Horizontal tabulation positions are provided by vendors of display devices as a convenient method of aligning data, on different lines, into columns. In some cases they can remove the need for a program to count the number of characters that have been written. The C Standard does not provide a method for controlling the location of horizontal tabulation positions. Neither does a program have any method of finding out which positions they occupy.

C++

Horizontal tab appears in Table 5, 2.13.2p3. There is no other description of this escape sequence, although the C behavior might be implied from the following wording:

> *The facilities of the Standard C Library are provided in 18 additional headers, as shown in Table 12:* 17.4.1.2p3

Common Implementations

The location of tabulation positions on a line are usually controlled by the display device. There may be a limited number that can be configured on a line. Configuring a horizontal tab position every eight active positions from the start of the line is a common default. Many hosts allow the default setting to be changed, and some users actively make use of this configuration option.

Coding Guidelines

A commonly seen application problem is the assumption, by the developer, of where the horizontal tabulation positions occur on a display device. However, the handling display devices are outside the scope of these coding guidelines.

---

264 If the active position is at or past the last defined horizontal tabulation position, the behavior of the display device is unspecified.

Commentary

The standard does not specify how many horizontal tabulation positions must be supported by an implementation, if any.

C90

If the active position is at or past the last defined horizontal tabulation position, the behavior is unspecified.

Common Implementations

Some implementations do not move the active position when the last defined horizontal tabulation position has been reached; others treat writing such a character as being equivalent to writing a single white-space character at this position. In some cases the behavior is to move the active position to the first horizontal tabulation position on the next line.

---

vertical tab escape sequence

\v (*vertical tab*) Moves the active position to the initial position of the next vertical tabulation position. 265

Commentary

Although the standard recognizes that the direction of writing is locale-specific, it says nothing about the order in which lines are organized. The vertical tab (and new line) escape sequence move the active position in the same line direction. There is no escape sequence for moving the active position in the opposite direction, similar to backspace for movement within a line.

page 260 logical

The concept of vertical tabulation implicitly invokes the concept of *current page*. This concept is primarily applied to printers, while the dimensions of a page might be less variable than a terminal. Before laser printers were invented, it was very important to ensure that output occurred in a controlled, top-down fashion.

C++

Vertical tab appears in Table 5, 2.13.2p3. There is no other description of this escape sequence, although the C behavior might be implied from the following wording:

17.4.1.2p3 *The facilities of the Standard C Library are provided in 18 additional headers, as shown in Table 12:*

Common Implementations

In most implementations a vertical tab moves the active position to the next line, with the relative position within the line staying the same.

If the active position is at or past the last defined vertical tabulation position, the behavior of the display device is unspecified. 266

Commentary

The intended behavior is likely to vary between terminals and printers.

C90

If the active position is at or past the last defined vertical tabulation position, the behavior is unspecified.

Common Implementations

Many display devices do not define vertical tabulation positions; this escape sequence simply causes the active position to move to the next line. The behavior is the same as when a new line escape sequence is written at the end of a page, or screen.

---

escape sequence fit in char object

Each of these escape sequences shall produce a unique implementation-defined value which can be stored in a single **char** object. 267

Commentary

These escape sequences are defined to be members of the basic execution character set and also to fit in a byte.

basic execution character set
basic character set fit in a byte

The mapping to this implementation-defined value occurs at translation time. The execution time value actually received by the display device is outside the scope of the standard. The library function `fputc` could map the value represented by these single **`char`** object into any sequence of bytes necessary.

C++

This requirement can be deduced from 2.2p3.

Other Languages

Java explicitly defines the values of the escape sequences it specifies.

Common Implementations

The specified escape sequences are available in the Ascii character set (and thus also in ISO 10646).

ISO 10646

---

268 The external representations in a text file need not be identical to the internal representations, and are outside the scope of this International Standard.

Commentary

The Committee recognizes that host file systems may use a representation for text files that is different from that used for binary files. The output functions will know the mode with which a stream was opened and can process the bytes written appropriately. There is a guarantee for binary files, which does not hold for text files, that the bytes written out shall compare equal to the same bytes read back in again.

C++

The C++ Standard does not get involved in such details.

Common Implementations

The external representation of a text file is usually the same as that used to hold a C source file. The representation of the new line escape sequence is usually the same as that for end-of-line, which is not always a single character.

end-of-line representation

From an executing program's point of view, on hosts that support output redirection, there may be no distinction made between a display device and a text file. However, the driver for a display device may respond differently for some characters.

---

269 **Forward references:** the **`isprint`** function (7.4.1.8), the **`fputc`** function (7.19.7.3).

References

1. J. Strang, T. O'Reilly, and L. Mui. *Termcap and Terminfo*. O'Reilly & Associates, Inc, 1989.