```
sub nice_string {
    join("",
      map { $_ > 255 ?                  # if wide character...
            sprintf("\\x{%04X}", $_) :  # \x{...}
            chr($_) =~ /[[:cntrl:]]/ ?  # else if control character ...
            sprintf("\\x%02X", $_) :    # \x..
            quotemeta(chr($_))          # else quoted or as themselves
      } unpack("U*", $_[0]));           # unpack Unicode characters
}
```

For example,

```
nice_string("foo\x{100}bar\n")
```

returns the string

```
'foo\x{0100}bar\x0A'
```

which is ready to be printed.

54.1.10 Special Cases

- Bit Complement Operator ~ And vec()

  The bit complement operator ~ may produce surprising results if used on strings containing characters with ordinal values above 255. In such a case, the results are consistent with the internal encoding of the characters, but not with much else. So don't do that. Similarly for `vec()`: you will be operating on the internally-encoded bit patterns of the Unicode characters, not on the code point values, which is very probably not what you want.

- Peeking At Perl's Internal Encoding

  Normal users of Perl should never care how Perl encodes any particular Unicode string (because the normal ways to get at the contents of a string with Unicode–via input and output–should always be via explicitly-defined I/O layers). But if you must, there are two ways of looking behind the scenes.

  One way of peeking inside the internal encoding of Unicode characters is to use `unpack("C*", ...` to get the bytes or `unpack("H*", ...)` to display the bytes:

  ```
  # this prints  c4 80  for the UTF-8 bytes 0xc4 0x80
  print join(" ", unpack("H*", pack("U", 0x100))), "\n";
  ```

  Yet another way would be to use the Devel::Peek module:

  ```
  perl -MDevel::Peek -e 'Dump(chr(0x100))'
  ```

  That shows the `UTF8` flag in FLAGS and both the UTF-8 bytes and Unicode characters in `PV`. See also later in this document the discussion about the `utf8::is_utf8()` function.

54.1.11 Advanced Topics

- String Equivalence

  The question of string equivalence turns somewhat complicated in Unicode: what do you mean by "equal"?

  (Is `LATIN CAPITAL LETTER A WITH ACUTE` equal to `LATIN CAPITAL LETTER A`?)

  The short answer is that by default Perl compares equivalence (`eq`, `ne`) based only on code points of the characters. In the above case, the answer is no (because 0x00C1 != 0x0041). But sometimes, any CAPITAL LETTER As should be considered equal, or even As of any case.

  The long answer is that you need to consider character normalization and casing issues: see *Unicode::Normalize*, Unicode Technical Reports #15 and #21, *Unicode Normalization Forms* and *Case Mappings*, http://www.unicode.org/unicode/reports/tr15/ and http://www.unicode.org/unicode/reports/tr21/

  As of Perl 5.8.0, the "Full" case-folding of *Case Mappings/SpecialCasing* is implemented.