A Consistent Grammar for Sequences

Document #: P3340R0
Date: 2024-06-24
Project: Programming Language C++
Audience: Core
Reply-to: Alisdair Meredith
<ameredith1@bloomberg.net>

Contents

1 Abstract

The C++ grammar defines and names many sequences and does so in a variety of ways with no obvious distinction for why a specific formulation is chosen. This paper proposes consistently adopting the simplest formulation.

2 Revision History

R0 June 2024 (St Louis meeting)

Initial draft of this paper.

3 Introduction

The intent of this paper is essentially editorial, providing a simple and consistent refactoring of the C++ grammar for sequences. This paper further suggests a drafting policy to which the Core Working Group could adhere. Because the nonfunctional changes touch the C++ grammar and because the intent is to form a future drafting policy, that the Core Working Group deem this change desirable and more than just an editorial update is important.

4 Analysis

A sequence consists of an initial term of some kind, followed by an unlimited number of additional terms of the same kind, without any form of list separator to distinguish them. We have predominantly three forms of specifying a sequence in the C++ grammar, and several additional uses of the term go beyond the simple notion of a sequence. Note that sequences are never empty since the leading term is not optional.

4.1 Two terminals

The most common form of sequence specifies a terminal that is a single element and a second terminal that (recursively) comprises a sequence of the same kind followed by a single element; for example:

c-char-sequence :
  c-char
  c-char-sequence c-char

This form is used by the following sequences:

— *c-char-sequence*
— *d-char-sequence*
— *h-char-sequence*
— *n-char-sequence*
— *q-char-sequence*
— *r-char-sequence*
— *s-char-sequence*
— *simple-octal-digit-sequence*
— *simple-hexadecimal-digit-sequence*
— *balanced-token-seq*
— *declaration-seq*
— *label-seq*
— *lambda-specifier-seq*
— *requirement-seq*
— *statement-seq*
— *virt-specifier-seq*
— *elif-groups*
— *pp-tokens*
— *h-pp-tokens*

4.2 Optional trailing sequence

The simplest form is a single terminal, leading with the element type, followed by an (optional) recursive mention of the sequence being specified; for example:

cv-qualifier-seq :
  cv-qualifier cv-qualifier-seq$_{opt}$

This form is used by the following sequences:

— *cv-qualifier-seq*
— *handler-seq*
— *conversion-declarator*
— *member-specification*

4.3 Optional leading sequence

The third form is to place the optional list *before* the terminating element. This form is used by only *attribute-specifier-seq*:

attribute-specifier-seq :
  attribute-specifier-seq$_{opt}$ *attribute-specifier*

4.4 Disguised lists

Several grammar productions define sequences that might optionally be a list, where there is an optional separator; for example:

hexadecimal-digit-sequence :
  hexadecimal-digit
  hexadecimal-digit-sequence `'`$_{opt}$ *hexadecimal-digit*

In this case, the grammar supports using the `'` character as an optional separator between each digit, but a list must be created since a *hexadecimal-digit-sequence* can neither start nor end with a `'`. However, the optional separator can be entirely omitted, producing a token that is essentially still a sequence. These kinds of sequences must preserve the original list-style formulation:

— *digit-sequence*
— *hexadecimal-digit-sequence*

4.5 Multiple sequences

Some sequences in the grammar are just group names for a variety of other sequences and thus should not be touched by this paper; for example:

escape-sequence :
  simple-escape-sequence
  numeric-escape-sequence
  conditional-escape-sequence

Those sequences should not be touched by this paper:

— *decl-specifier-seq*
— *defining-type-specifier-seq*
— *escape-sequence*
— *numeric-escape-sequence*
— *type-specifier-seq*

4.6 Escape sequences

Several grammar terms use the suffix *-sequence* but do not form a repeating sequence; these all happen to be escape sequences; for example:

> *simple-escape-sequence :*
> \ *simple-escape-sequence-char*

Whether the *-sequence* suffix is the most consistent naming is unclear but is also well established and, therefore, should not be touched by this paper:

— *conditional-escape-sequence*
— *octal-escape-sequence*
— *simple-escape-sequence*

4.7 Naming

Most sequence grammars have a *seq* suffix, although the character and digit sequences use the full name *sequence*. A small assortment of miscellaneous terms have no suffix that would suggest they form a sequence.

Some consistency could be applied here, too, such as uniformly adopting either *seq* or *sequence* rather than maintaining both. Such a change would simplify drafting future language proposals by having a clear precedent to follow, but whether that consistent naming offers as much benefit as the consistent formulation for the grammar terms is unclear. Further, changing the names of the terms is a bigger edit to the Standard since every use of a renamed term must also be changed consistently and thus is beyond the reach of this paper.

Alternatively, perhaps an unwritten policy is at play here. Note that *-sequence* seems to denote a sequence that accumulates characters to build a single token, whereas *-seq* denotes a sequence that is a collection of tokens. If the Core working group expects this scheme when naming grammar terms, where is that expectation documented? Neither *-sequence* nor *seq* covers the miscellaneous cases that use neither suffix.

5 Design Principles

The principles underlying the proposed changes are described here.

5.1 Clarity

The overriding principle is clarity; the Standard must be easy to interpret so that it is unambiguous and that all readers easily agree on the same interpretation.

5.2 Consistency

A consistent presentation of ideas is much simpler to understand because it removes the notion of false negatives. If we say the same thing in the same way, readers will not wonder if they might be missing subtle differences in the different forms.

5.3 Simplicity

If there are multiple ways to say the same thing, choose the simplest since that will often be the clearest, and if all things are equal, the simplest way at least removes unnecessary complexity.

6 Proposed Solution

Adopt the optional trailing sequence formulation of the grammar as the simple consistent form to be used everywhere. Adopting the one-terminal form rather than the predominant two-terminal form also helps to distinguish sequences from lists, as lists must always use the two-terminal form in order to accommodate the list separator.

6.1 Policy: Specifying sequences

When specifying a sequence, prefer to use a recursive formulation in a single line, with the optional recursion on the trailing term.

6.2 Policy: Naming sequences

When a sequence is parsed character by character to consume a single token or a partial token, the suffix *-sequence* is used.

When a sequence is parsed token by token, the suffix *-seq* is used.

7 Core Review : St Louis, 2024 June 24

This paper was reviewed, and approved for plenary **at Wrocław** given its late arrival. The only review comment was to remove *module-name-qualifier* from revision, as its form and purpose essentially follow those of *nested-name-specifier*.

8 Wording

Make the following changes to the C++ Working Draft. All wording is relative to [N4981], the latest draft at the time of writing.

5.3 [lex.charset] Character sets

n-char-sequence :
  n-char *n-char-sequence*$_{opt}$
  ~~*n-char-sequence n-char*~~

simple-hexadecimal-digit-sequence :
  hexadecimal-digit *simple-hexadecimal-digit-sequence*$_{opt}$
  ~~*simple-hexadecimal-digit-sequence hexadecimal-digit*~~

5.8 [lex.header] Header names

h-char-sequence :
  h-char *h-char-sequence*$_{opt}$
  ~~*h-char-sequence h-char*~~

q-char-sequence :
  q-char *q-char-sequence*$_{opt}$
  ~~*q-char-sequence q-char*~~

5.13.3 [lex.ccon] Character literals

c-char-sequence :
  c-char *c-char-sequence*$_{opt}$
  ~~*c-char-sequence c-char*~~

simple-octal-digit-sequence :
  octal-digit *simple-octal-digit-sequence*$_{opt}$
  ~~*simple-octal-digit-sequence octal-digit*~~

5.13.5 [lex.string] String literals

s-char-sequence :
  s-char *s-char-sequence*$_{opt}$
  ~~*s-char-sequence s-char*~~

r-char-sequence :
  r-char *r-char-sequence*$_{opt}$
  ~~*r-char-sequence r-char*~~

d-char-sequence :
  d-char *d-char-sequence*$_{opt}$
  ~~*d-char-sequence d-char*~~

7.5.5.1 [expr.prim.lambda.general] General

lambda-specifier-seq :
  lambda-specifier *lambda-specifier-seq*$_{opt}$
  ~~*lambda-specifier-seq lambda-specifier*~~

7.5.7.1 [expr.prim.req.general] General

requirement-seq :
  requirement requirement-seq$_{opt}$
  ~~*requirement-seq requirement*~~

8.4 [stmt.block] Compound statement or block

label-seq :
  label label-seq$_{opt}$
  ~~*label-seq label*~~

statement-seq :
  statement statement-seq$_{opt}$
  ~~*statement-seq statement*~~

9.1 [dcl.pre] Preamble

declaration-seq :
  declaration declaration-seq$_{opt}$
  ~~*declaration-seq declaration*~~

9.12.1 [dcl.attr.grammar] Attribute syntax and semantics

attribute-specifier-seq :
  ~~*attribute-specifier-seq*$_{opt}$ *attribute-specifier*~~
  attribute-specifier attribute-specifier-seq$_{opt}$

balanced-token-seq :
  balanced-token balanced-token-seq$_{opt}$
  ~~*balanced-token-seq balanced-token*~~

11.4.1 [class.mem.general] General

virt-specifier-seq :
  virt-specifier virt-specifier-seq$_{opt}$
  ~~*virt-specifier-seq virt-specifier*~~

15.1 [cpp.pre] Preamble

elif-groups :
  elif-group elif-groups$_{opt}$
  ~~*elif-groups elif-group*~~

pp-tokens :
  preprocessing-token pp-tokens$_{opt}$
  ~~*pp-tokens preprocessing-token*~~

15.2 [cpp.cond] Conditional inclusion

h-pp-tokens :
  h-preprocessing-token h-pp-tokens$_{opt}$
  ~~*h-pp-tokens h-preprocessing-token*~~

9 Acknowledgements

Thanks to Michael Park for the pandoc-based framework used to transform this document's source from Markdown.

Thanks for Richard Smith for early technical feedback.

10 References

[N4981] Thomas Köppe. 2024-04-16. Working Draft, Programming Languages — C++.
https://wg21.link/n4981