Document number: P1990R1
Project: Programming Language C++
Audience: LEWG, LWG

Daniil Goncharov neargye@gmail.com
Antony Polukhin antoshkka@gmail.com

Date: 2020-05-26

Add operator[] and data() to std::initializer_list

I. Introduction and Motivation

`std::initializer_list` is a lightweight proxy object that provides access to an array of objects of type `const T`, but fully use as array-proxy is difficult. Therefore, it is proposed to add an `operator[]` and `data` to `std::initializer_list`.

II. Impact on the Standard

This proposal is a pure library extension. It proposes changes to existing header `<initializer_list>` such that the changes do not break existing code and do not degrade performance. It does not require any changes in the core language in simple cases of non assembly optimized Standard Library, and it could be implemented in standard C++.

III. Design Decisions

A. Add `operator[]`

Access to objects by index in the current version is difficult, Therefore, it is proposed to add an `operator[]` which can be easily implemented at the library level.

Before	**After**

`struct Vector3 {`
`  int x, y, z;`
`  Vector3(std::initializer_list il) {`
`  x = *(il.begin() + 0);`
`  y = *(il.begin() + 1);`
`  z = *(il.begin() + 2);`
`  }`
`};`
	`struct Vector3 {`
	`  int x, y, z;`
	`  Vector3(std::initializer_list il) {`
	`  x = il[0];`
	`  y = il[1];`
	`  z = il[2];`
	`  }`
	`};`

Before	After

class MultiIndexVector {
using index_t =
std::initializer_list<std::size_t>;
auto operator[](index_t idx) {
return {data1[*(idx.begin() + 0)],
data2[*(idx.begin() + 1)]};
}
};
	class MultiIndexVector {
	using index_t =
	std::initializer_list<std::size_t>;
	auto operator[](index_t idx) {
	return {data1[idx[0]],
	data2[idx[1]]};
	}
	};

B. Add `data()`

`data()` allows to get more access to the array without using iterators.

C. Satisfies concept `std::ranges::contiguous_range`

In C++20 `std::initializer_list` satisfies concept `std::ranges::contiguous_range`, the addition of new methods will not affect code behavior with ranges.

```
template <std::ranges::contiguous_range R>
void foo(R&& r) { /*...*/ }

foo(std::initializer_list<int>{1, 2, 3}); // Ok in C++20.
```

IV. Proposed wording relative to n4835

Modifications to "27.10 Initializer lists" [support.initlist]

All the additions to the Standard are marked with green.

A. Modifications to "17.10.1 Header <initializer_list> synopsis" [initializer.list.syn]

namespace std {

template<class E> class initializer_list {
public:
using value_type = E;
using reference = const E&;
using const_reference = const E&;
using size_type = size_t;
using iterator = const E*;
using const_iterator = const E*;

constexpr initializer_list() noexcept;

```
constexpr size_t size() const noexcept;

constexpr const E* begin() const noexcept;
constexpr const E* end() const noexcept;

constexpr const E& operator[](size_type idx) const;

constexpr const E* data() const noexcept;
};

// 17.10.4, initializer list range access
template<class E> constexpr const E* begin(initializer_list<E> il) noexcept;
template<class E> constexpr const E* end(initializer_list<E> il) noexcept;

}
```

B. Modifications to "17.10.3 Initializer list access" [support.initlist.access]

constexpr const E* begin() const noexcept;
1 Returns: A pointer to the beginning of the array. If size() == 0 the values of begin() and end() are unspecified but they shall be identical.

constexpr const E* end() const noexcept;
2 Returns: begin() + size().

constexpr size_t size() const noexcept;
3 Returns: The number of elements in the array.
4 Complexity: Constant time.

constexpr const E& operator[](size_type idx) const;
5 Preconditions: idx < size() is true.
6 Returns: *(begin() + idx).
7 Throws: Nothing.

constexpr const E* data() const noexcept;
8 Returns: begin().

C. Modify to "17.3.2 Header <version> synopsis" [version.syn]

#define __cpp_lib_initializer_list *DATE OF ADOPTION*

V. Revision History

Revision 0:

- Initial proposal
- Prague voting
    - Add data to this proposal?

- We should promise more committee time to pursuing P1990R0 (std::initializer_list::operator[]), knowing that our time is scarce and this will leave less time for other work.

  NO OBJECTION TO UNANIMOUS CONSENT.

- Forward P1990R0 (std::initializer_list::operator[]) to LEWG for C++23.

  NO OBJECTION TO UNANIMOUS CONSENT.

- Consensus: LEWGI sends P1990R0 (`std::initializer_list::operator[]`), with the guidance below, to LEWG, for C++23.

  - Fix the first example in the comparison table; the left hand side is structurally different from the right (check with Conor Hoekstra for details).
  - Check if these changes will make `std::initializer_list` satisfy `ContiguousView`; if so, consider changing the title of the paper to "Make `std::initialzier_list` satisfy `ContiguousView`".
  - Sanity check this with someone knowledgable in EWG(I) to make sure this change doesn't have any language implications.

Revision 1:

- Add `data()`.
- Fix "Example" and update "Introduction and Motivation".
- Add "Design Decisions".
- Changing the title of the paper to "Add operator[] and data() to std::initializer_list".

VI. References

- [N4835] Working Draft, Standard for Programming Language C++. Available online at https://github.com/cplusplus/draft/raw/master/papers/n4835.pdf