Document number: P1990R0
Project: Programming Language C++
Audience: LEWGI, LEWG, LWG

Daniil Goncharov neargye@gmail.com

Antony Polukhin antoshkka@gmail.com

Date: 2019-12-16

Add operator[] to std::initializer_list

I. Introduction and Motivation

`std::initializer_list` s a lightweight proxy object that provides access to an array of objects of type `const T`, but access to objects by index in the current version is difficult. Therefore, it is proposed to add an `operator[]` to `std::initializer_list`.

Consider the simple example:

Before	**After**

`struct Vector3 {`
`  int x, y, z;`
 	
`  Vector3(std::initializer_list<int> il) {`
`  std::size_t idx = 0;`
`  for (auto i : il) {`
`  if (idx == 0) {`
`  x = i;`
`  } else if (idx == 1) {`
`  y = i;`
`  } else if (idx == 2) {`
`  z = i;`
`  }`
`  ++idx;`
`  }`
`  }`
`  ...`
`};`
	`struct Vector3 {`
	`  int x, y, z;`
	
	`  Vector3(std::initializer_list<int> il) {`
	`  x = il[0];`
	`  y = il[1];`
	`  z = il[2];`
	`  }`
	`  ...`
	`};`

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

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. 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;
};

// 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.

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

#define __cpp_lib_initializer_list DATE OF ADOPTION

IV. Revision History

Revision 0:

- Initial proposal

V. References:

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