Intrinsic for reading uninitialized memory

Document #:	P3530R0
Date:	2024-12-17
Project:	Programming Language C++
Audience:	EWG, Core, LWG

Reply-to:	Boleyn Su <boleyn.su@gmail.com> Gašper Ažman <gasper.azman@gmail.com>

Contents

1 Abstract

With current standard, it is impossible to safely read uninitialized memory. Thus, data structures or algorithms relying on reading uninitialized memory cannot be implemented in standard C++.

We therefore propose to add a new intrinsic to allow reading uninitialized memory.

2 Motivation

The adoption of erroneous behavior [P2795R5] defined the behaviour of reading uninitialized memory to a greater extent than in C++23.

While this is a good thing in the vast majority of cases, it breaks rare legitimate usecases for algorithms that rely on reading uninitialized memory for their performance guarantees (cited below), with no recourse.

We need an intrinsic to mark reading uninitialized memory as intended, not erroneous.

3 Background

There is a well-known data structure to present sparse sets [Sparse] as shown below.

```cpp
template <int n>
class SparseSet {
  // Invariants:
  // - index_of[elements[i]] == i for all 0<=i<size
  // - elements[0..size-1] contains all elements in the set.
  // These invariants guarantee the correctness.
  int elements[n];
  int index_of[n];
  int size;
public:
  SparseSet() : size(0) {}  // we do not initialize elements and index_of
  void clear() { size = 0; }
  bool find(int x) const {
    // assume x in [0, n)
    // There is a chance we read index_of[x] before writing to it.
    int i = index_of[x];
    // The algorithm is correct for an arbitrary value of index_of[x],
    // as long as the read itself is not undefined behavior,
    // because the invariants guarantee x is in the set if and only if
    // the below condition holds.
    return 0 <= i && i < size && elements[i] == x;
  }
  void insert(int x) {
    // assume x in [0, n)
    if (find(x)) { return; }
    // The invariants are maintained.
    index_of[x] = size;
    elements[size] = x;
    size++;
  }
  void remove(int x) {
    // assume x in [0, n)
    if (!find(x)) { return; }
    // The invariants are maintained.
    size--;
    int i = index_of[x];
    elements[i] = elements[size];
    index_of[elements[size]] = i;
  }
};
```

The read `index_of[x]` in `find` above may read uninitialized memory, which C++26 makes erroneous, without recourse.

It is impossible to implement a data structure supporting the same set of operations with a worst-case time complexity of O(1) for each operation without relying on reading uninitialized memory, to the author's best knowledge.

4 Analysis

The LLVM project already supports `freeze` instruction which is the lower level equivalent of our proposal [Freeze].

5 Proposal

5.1 Alternative 1

We propose to add a magic function

```
template <typename T>
T std::read_maybe_uninitialized(const T& v) noexcept;
```

where `T` must be an implicit-lifetime type.

`read_maybe_uninitialized(v)` returns the value of `v` if `v` is initialized.

Otherwise, it returns an unspecified value. (This is how LLVM's `freeze` works.)

5.2 Alternative 2

This approach is modeled after `start_lifetime_as_array`, but also marks the storage as having been initialized to an unspecified but valid value.

```
template <typename T>
T* start_lifetime_as_array_uninitialized(void* v, size_t n) noexcept;
```

Implicitly creates an array with element type `T` and length `n`, and the storage is considered initialized even if no write occured to the storage previously.

The rest as `start_lifetime_as_array`.

A companion function

```
template <typename T>
T* start_lifetime_as_uninitialized(void*);
```

is also provided, with the same semantics.

6 Wording

Will be provided when the alternative is chosen.

7 References

[Freeze] Juneyoung Lee, Yoonseung Kim, YoungJu Song, Chung-Kil Hur, Sanjoy Das, David Majnemer, John Regehr, and Nuno P. Lopes. Taming Undefined Behavior in LLVM.
https://sf.snu.ac.kr/publications/undefllvm.pdf

[P2795R5] Thomas Köppe. 2024-03-22. Erroneous behaviour for uninitialized reads.
https://wg21.link/p2795r5

[Sparse] Preston Briggs and Linda Torczon. An Efficient Representation for Sparse Sets.
https://dl.acm.org/doi/pdf/10.1145/176454.176484