2. Just as with pointers, every **reference type** is a literal type irrespective of whether the type to which it refers is itself a literal type.

`int&`	`int&` is a *literal type*
`T&`	`T&` is a *literal type* (for any `T`).
`T&&`	`T&&` is a *literal type* (for any `T`).

3. A **class**, **struct**, or **union** is a literal type if it meets each of these four requirements.
   (a) It has a **trivial destructor**.[9]
   (b) Each nonstatic data member is a non**volatile** literal type.[10]
   (c) Each base class is a literal type.
   (d) There is some way to initialize an object of the type during constant evaluation; either it is an **aggregate type**, thereby affording aggregate initialization, or it has at least one **constexpr** constructor (possibly a template) that is not a *copy* or *move* constructor:

```
#include <string>  // std::string
struct LiteralUDT
{
    static std::string s_cache;
        // OK, static data member can have a nonliteral type.

    int d_datum;
        // OK, nonstatic data member of nonvolatile literal type

    constexpr LiteralUDT(int datum) : d_datum(datum) { }
        // OK, has at least one constexpr constructor

    LiteralUDT() : d_datum(-1) { }
        // OK, can have nonconstexpr constructors

    // constexpr ~LiteralUDT() { }   // not permitted until C++20
        // No need to define: implicitly generated destructor is trivial.
};

struct LiteralAggregate
{
    int d_value1;
    int d_value2;
};
```

[9] As of C++20, a destructor can be declared **constexpr** and even both **virtual** and **constexpr**.
[10] In C++17, this restriction is relaxed: For a **union** to be a literal type, only one, rather than all, of its nonstatic data members needs to be of a non**volatile** literal type.