```
#include <cstring>  // std::memcpy

int returnOne()  // invalid attempt to return (int) 1
{
    unsigned int x = 1;  // object of trivially copyable type
    alignas(int) unsigned char a[sizeof x];  // aligned array of bytes
    std::memcpy(a, &x, sizeof x);            // copy object representation
    int* x2 = reinterpret_cast<int*>(a);     // OK, but not OK to use
    return *x2;                              // Bug (UB), no int at x2
}
```

As previously mentioned, the **`reinterpret_cast`** itself is valid, but the resulting pointer (`x2`, above) does not refer to a valid object of type **`int`**; hence, dereferencing `x2` has undefined behavior.[43]

6. **Accessing base or member subobjects via `reinterpret_cast`** — The C++ Standard guarantees that objects of standard-layout class type share the same address as certain subobjects when they exist, namely, the first *non*static data member (including within any base class) and all base-class subobjects. This guarantee does *not*, however, hold for any *other* subobjects, nor for *any* subobjects of a *non*-standard-layout type. Note that the requirement that all base class subobjects of a standard-layout class type have the same address as the most-derived class object was not in the C++11 or C++14 Standards as published and was not codified until June 2018 with the resolution of CWG 2254 as a defect report[44]; not all compilers have yet implemented the resolution described in CWG 2254, however, resulting in unreliable placement of the second and subsequent base-class subobjects.

   For example, suppose we have a standard-layout class type, `D`, that has two base classes, `B1` and `B2`:

```
struct B1 { int i; int j; };  // first  base class (standard layout)
struct B2 { int f(); };       // second base class      "       "
struct D : B1, B2 { };        // multiple inheritance   "       "
```

   A subobject of the first base class, `B1`, and the first **nonstatic data member**, `i`, will reside at the same address (e.g., `b1p` and `i1p`, respectively) as that of the derived class object (e.g., `&d`). Moreover, in compilers that conform to the changes in the resolution of CWG issue 2254, a subobject of the second base class, `B2`, will also reside at that address (e.g., `b2p`):

[43] As of C++20, we use `std::bit_cast<T>(x)` to implicitly create an object of the destination type, `T`, from the value representation of the source object, `x`, where **`sizeof`**`(T) == `**`sizeof`**`(x)` and both are trivially copyable. Future versions of C++ might someday render returning `*x2` from the `returnOne` function valid, thereby obviating our writing `std::bit_cast<`**`unsigned int`**`>(*x2)`; see **smith20**.

[44] The resolution of CWG 2254 (**smith16a**) requires that all of the base class subobjects of an object of standard-layout type must have the same address as the object. Although not all compilers conform to this change and notably MSVC 19.29 (c. 2021) does not, all implementations at least ensure that the *first* base class subobject shares the address of the most-derived class object.