```
    int x, *xp = &x;  // integer and pointer-to-integer variables

    // Convert pointer to various integral types.
    short int  si = reinterpret_cast<short int>(&x);  // Error, too small
    int         i = reinterpret_cast<int>      (&x);  // OK, if 32-bit void*
    long       li = reinterpret_cast<long>     (&x);  // OK, typically
    long      qli =      static_cast<long>     (&x);  // Error, static_cast
    long long lli = reinterpret_cast<long long>(&x);  // OK, typically
    __int128 llli = reinterpret_cast<__int128> (&x);  // OK, e.g., on GCC

    // Convert to optionally supported standard integral types.
    std::uintptr_t uipt = reinterpret_cast<std::uintptr_t>(&x);
                                                      // OK, if supported
    std::intptr_t   ipt = reinterpret_cast<std::intptr_t> (&x);
                                                      // OK, if supported

    // Convert integral values back to pointers:
    int*   lp = reinterpret_cast<int*>(  li);         assert(&x == lp);
    int*  llp = reinterpret_cast<int*>( lli);         assert(&x == llp);
    int* lllp = reinterpret_cast<int*>(llli);         assert(&x == lllp);

    int* p1   = reinterpret_cast<int*>(uipt);         assert(&x == p1);
    int* p2   = reinterpret_cast<int*>( ipt);         assert(&x == p2);
}
```

As the example code above shows, we can safely **`reinterpret_cast`** a pointer (e.g., `&x`) to an integral type of sufficient size *by value* and then back to the original pointer type. Note that **`static_cast`** can be used only for types that are implicitly convertible in at least one direction; hence, **`static_cast`** cannot be used to convert between pointers and integral values. Although optionally supported, most modern standard libraries provide standard types `std::intptr_t` and `std::uintptr_t` (in header `<cstdint>`), which are aliases for the platform-dependent signed and unsigned **integer types**, respectively, that are large enough to hold a pointer value:

```
#include <cstdint>  // std::uintptr_t, std::intptr_t, if supported

static_assert(sizeof(std::intptr_t) >=sizeof(void*),"");  // OK, if supported
static_assert(sizeof(std::uintptr_t)>=sizeof(void*),"");  // OK, if supported
```

Although modifying virtual memory addresses is not portable, we can often take advantage of "spare bits" in a pointer holding the address of a type `T` for which **`alignof`**`(T)` is 2 or or more — typically any scalar type larger than **`char`** and any class type having a nonstatic member of such a scalar type. In most implementations, casting a valid `T*` to an integral quantity always yields an even value: