```
#include <cctype>  // std::tolower

#include <encoder.h>

void lowercaseEncode(std::string* result, const std::string& input)
{
    auto encodedLength = Encoder::encodedLengthFor(input);

    result->resize(encodedLength);
    Encoder::encode(result->begin(), input);

    while (--encodedLength >= 0)  // (1)
    {
        (*result)[encodedLength] = std::tolower((*result)[encodedLength]);
    }
}
```

The `encodedLength` variable in the example above uses **`auto`** to deduce its type from the return value of `Encoder::encodedLengthFor`. If the maintainers of the `Encoder` library changed the return type of `encodedLengthFor` function to an unsigned type, e.g., `std::size_t`, instead of **`int`**, the `lowercaseEncode` function would become defective due to different behavior of decrementing `0` for unsigned types.

Surprising deduction for list initialization

`auto` type-deduction rules differ from those of function templates if brace-enclosed initializer lists are used. Function template argument deduction will always fail, whereas, according to C++11 rules, `std::initializer_list` will be deduced for **`auto`**.

```
auto example0 = 0; // copy initialization, deduced as int
auto example1(0);  // direct initialization, deduced as int
auto example2{0};  // list initialization, deduced as std::initializer_list<int>

template <typename T> void func(T);

void testFunctionDeduction()
{
    func(0);    // T deduced as int
    func({0});  // Error
}
```

This surprising behavior was, however, widely regarded as a mistake.[4]

[4]This erroneous behavior was formally rectified in C++17 with, e.g., **`auto`** `i{0}` deducing **`int`**. Furthermore, mainstream compilers had applied this deduction-rule change retroactively as early as GCC 5.1 (c. 2015), Clang 3.8 (c. 2016), and MSVC 19.00 (c. 2015), with the revised rule being applied even if `std=c++11` flag is explicitly supplied.