Don’t add to the signed/unsigned mess

Bjarne Stroustrup

Abstract
There are many ways of writing a simple loop. Too many, and we are proposing to add more. My suggestion: don’t.

The root problem is that in C and C++ signed and unsigned integers don’t mix well. We should begin the process of minimizing that problem by not adding more opportunities for such mixing.

Please note that this is not a suggestion to change the WP. It is an argument for keeping status quo until we are certain we have something better.

Loops
As an example, I will use a very simple loop that simply zeros out the elements of a vector.

Use a range for:

```cpp
for (auto& x : v) x=0;
```

That’s the simplest and often the best, but people – many people – like to play with loop variables and occasionally they actually need to:

```cpp
for (int i = 0; i<v.size(); ++i) v[i]=0;       // naive and natural
for (unsigned i = 0; i<v.size(); ++i) v[i]=0; // hard to optimize (e.g., [Carruth,2016])
for (auto i = 0; i<v.size(); ++i) v[i]=0;
for (auto i = 0u; i<v.size(); ++i) v[i]=0;
for (vector<double>::size_type i = 0; i<v.size(); ++i) v[i]=0;  // verbose and error-prone
for (decltype(v)::size_type i = 0; i<v.size(); ++i) v[i]=0;
for (size_t i = 0; i<v.size(); ++i) v[i]=0;
for (ptrdiff_t i = 0; i<v.size(); ++i) v[i]=0;
```

And more variations. This is too much and offers many opportunities for confusion; we should not add to that.

Now people are proposing [p0330r4] [p1227R1]:

```cpp
for (auto& x : v) x=0;
```
for (auto i = 0; i< v.size(); ++i) v[i]=0;
for (auto i = 0z; i< v.size(); ++i) v[i]=0;
for (auto i = 0uz; i< v.size(); ++i) v[i]=0;

As ever, people will get confused and - in addition to my examples - use some of the many more variants that I haven’t mentioned.

Workarounds and alternatives
Of course, some (but not all) compilers warn about common cases:

for (int i = 0; i< v.size(); ++i) v[i]=0;  // warning

That is most annoying because most vectors have far fewer than 2 billion elements. In fact, the standard limits the number of elements of a vector to the largest positive value of its difference type (General Container Requirements, table 64). This leads people to complain bitterly about C++, especially novices and people coming to C++ from other languages. New people come to C++ faster than we can teach them to do such basic things differently from what they were used to.

So, people and organizations ignore those warnings or suppress them, setting a dangerous example for other warnings and causing trouble when you do get a 2B+ vector. False positives do harm.

People also look for alternatives:

for (int i = 0; i < (int)v.size(); ++i) v[i]=0;  // use a cast

That’s unnecessarily verbose, dangerous in the sense that it could be wrong (here, narrowing on some machines), and teaches people to use the terse, general, and error-prone C-style cast.

Here is a variant that does not use a cast:

for (int i = 0, n = v.size(); i < n; ++i ) v[i]=0;   // verbose

Such workarounds avoid warnings (but narrows and converts unsigned to signed). They also make people wonder about the sanity of C++.

In places, people use a helper function. For example:

for (int i = 0; i < elem_count(v); ++i) v[i]=0;

where elem_count() is a function that takes a container or a range and returns a signed value (and hides the cast).

For many examples, there are alternatives to C-style for-loops. I mentioned the range-for up front, but algorithms often offer alternatives

std::fill(v.begin(),v.end(),0);
std::fill(v,0);    // when we get ranges
std::for_each(v.begin(),v.end(),[](int& x){ x=0; });
Again, try to explain that to a C++ novice. Better still, try to get the point across to a novice for whom you are not formally a teacher or a Mentor. Or for someone you will never meet. Unless somehow advised otherwise, such people often (typically?) start with

```c
for (int i = 0; i<v.size(); ++i) v[i]=0;  // annoying, incomprehensible warning
```

For almost all uses, that warning is a false positive; that is, irrelevant.

**Actual proposals**

[p0330r4] proposes to add to the – already confusing – set of suffixes by adding `uz` and `z` (or maybe some other letters). We’d have `u`, `U`, `l`, `Z`, `f`, `F`, and `p` (has `p` been formally proposed?) plus combinations in addition to user-defined suffixes. There are also decimal floating point and soon short floats.

[p1227R1] proposes to change `size()` in the ranges TS and for `span` to unsigned (making them bug compatible with the STL) and adding `ssize()` to all containers and range accessors

- embeds a type in a function name (and it makes me think of Parseltongue 😊)
- leaves the wrong solution (IMO) with the better, more established, and simpler name
- adds a few more cases to the wrong (IMO) solution [P1428R0]

What other types deserves suffixes? What other types could “benefit” from similar addition of signed-type alternatives to current unsigned ones? Are there types for which such additions would offer more help to programmers that the current proposals for (just) signed and unsigned? I suspect so. The quest for patches would be open-ended.

For C++, signed sizes and subscripts are the best solution: make all `size()`s signed! That is not perfect, and I don't propose that for C++20, but it is the solution with the least problems and the best opportunities to catch problems (e.g., contracts and run-time checks) [P1428R0]. We should aim for that and start gathering facts/data, rather than adding to the problem (e.g., by changing `span<T>::size()` to be unsigned).

**Acknowledgements**

Thanks to Peter Dimov, Tony Van Eerd, Peter Sommerlad, Sergey Zubkov, Herb Sutter, and others for constructive comments of a draft of this.

**References**

- [P0330R4] JeanHeyd Meneide and Rein Halbersma: [Literal Suffixes for `ptrdiff_t` and `size_t`](#).
- [p1227R1] Jorg Brown: [Signed `ssize()` functions, unsigned `size()` functions](#).
- [P1428R0] Bjarne Stroustrup: [Subscripts and sizes should be signed](#).