Example: String Split

Here, we'll determine whether string_view can be used for splitting a string into several view objects.

`string_view` might be a potential optimization for string splitting. If you own a large persistent string, you might want to create a list of string_view objects that maps words of that larger string.

> *Note:* The code is inspired by the article by Marco Arena - string_view odi et amo1.

```
#include <iostream>
#include <string>
#include <vector>
#include<algorithm>
using namespace std;

std::vector<std::string_view>
splitSV(std::string_view strv, std::string_view delims = " ")
{
  std::vector<std::string_view> output; auto first = strv.begin();
  while (first != strv.end()) {
    const auto second = std::find_first_of( first, std::cend(strv),
                                            std::cbegin(delims), std::cend(delims));
    if (first != second) {
      output.emplace_back(strv.substr(std::distance(strv.begin(), first), std::distance(first
    }

    if (second == strv.end())
      break;
    first = std::next(second);
  }
  return output;
}


int main() {
  const std::string str {
    "Hello Extra,,, Super, Amazing World"
  };
  for (const auto& word : splitSV(str, " ,"))
    std::cout << word << '\n';
}
```

The algorithm iterates over the input `string_view` and finds breaks - characters that match the delimiter list. Then the code extracts part of that sequence - between the last and the new break. The sub-view is stored in the output vector.

Some notes regarding the implementation:

- The `string_view` version of the algorithm assumes the input string is persistent and not a temporary object. Be careful with the returned `vector` of `string_view` as it also points to the input string.
- The instruction `if (first != second)` - protects from adding empty "words", in a case where there are multiple delimiters next to each other (like double spaces).
- The algorithm uses `std::find_first_of` but it's also possible to use `string_view::find_first_of`. The member method doesn't return an iterator, but the position in the string.
- The member method of `string_view` appeared to be slower than the `std::find_first_of` version in some tests when the number of delimiters is small.

If you want to see some experiments regarding the code in this section have a look at: Performance of std::string_view vs std::string from C++17 and Speeding Up string_view String Split Implementation. Those two blog posts describe the benchmark results and add some more possible improvements to the code.

That's it for `std::string_view`. By now, you must have a better understanding of why this utility is a useful addition to C++. The summary and quiz ahead will help reinforce all the concepts we have learned here.

See you in the next section, where we will tackle `std::search`.