Pairs

The idea of a pair of values often comes handy in programming. C++ allows us to make these pairs.

WE'LL COVER THE FOLLOWING

With `std::pair`, you can build pairs of arbitrary types. The class template `std::pair` needs the header `<utility>`. `std::pair` has a default, copy and move constructor. Pair objects can be swapped: `std::swap(pair1, pair2)`.

Pairs will often be used in the C++ library. For example, the function std::minmax returns its result as a pair, the associative container `std::map`, `std::unordered_map`, `std::multimap` and `std::unordered_multimap` manage their key/value association in pairs.

To get the elements of a pair `p`, you can either access it directly or via an index. So, with `p.first` or `std::get<0>(p)` you get the first, with `p.second` or `std::get<1>§ you get the second element of the pair.

Pairs support the comparison operators `==`, `!=`, `<`, `>`, `<=` and `>=`. If you compare two pairs for identity, at first the members `pair1.first` and `pair2.first` will be compared and then `pair1.second` and `pair2.second`. The same strategy holds for the other comparison operators.

std::make_pair

C++ has the practical help function `std::make_pair` to generate pairs, without specifying their types. `std::make_pair` automatically deduces their types.

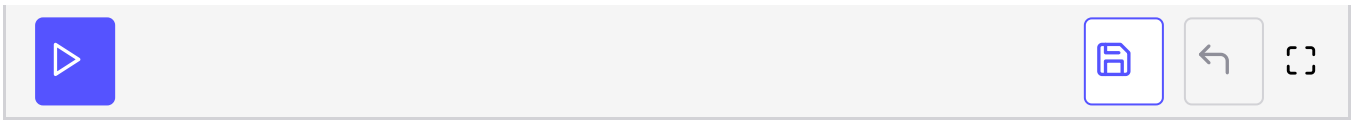
```
// pair.cpp
#include <iostream>
#include <utility>
using namespace std;

int main(){
```

```
int main(){
  pair<const char*, double> charDoub("str", 3.14);
  pair<const char*, double> charDoub2 = make_pair("str", 3.14);
  auto charDoub3 = make_pair("str", 3.14);

  cout << charDoub.first << ", " << charDoub.second << "\n";    // str, 3.14
  charDoub.first = "Str";
  get<1>(charDoub) = 4.14;
  cout << charDoub.first << ", " << charDoub.second << "\n";    // Str, 4.14

  return 0;
}
```



The helper function `std::make_pair

In the next lesson, we will talk about tuples in C++.