std::tuple

This lesson talks about std::tuple in detail.

WE'LL COVER THE FOLLOWING

- std::make_tuple
- std::tie and std::ignore
- Further information

Tuples extend the principles of a pair to a broader range of functions. We can create tuples of arbitrary lengths and types with std::tuple. The class template needs the header `<tuple>`. `std::tuple` is a generalization of `std::pair`. We can convert between tuples with two elements and pairs. The tuple has, like his younger brother `std::pair`, a default, a copy, and a move constructor. We can swap tuples with the function `std::swap`.

The i-th element of a tuple, `t`, can be referenced by the function template `std::get`: `std::get<i-1>(t)`. By `std::get<type>(t)`, we can directly refer to the element of the type `type`.

Tuples support the comparison operators `==`, `!=`, `<`, `>`, `<=`, and `>=`. If we compare two tuples, the elements of the tuples will be compared lexicographically. The comparison starts at index 0.

std::make_tuple

The helper function std::make_tuple is quite convenient for the creation of tuples. We don't have to provide the types. The compiler automatically deduces them.

For a better understanding, consider the example below:

```
#include <iostream>
```

```
#include <string>
#include <tuple>


int main(){

  std::cout << std::boolalpha << std::endl;

  // create two tuples
  std::tuple<std::string, int, double> tup1("first", 3, 4.17);
  std::tuple<std::string, int, double> tup2 = std::make_tuple("second", 4, 1.1);
  // auto tup2= std::make_tuple("second", 4, 1.1);

  // read the values
  std::cout << "tup1: "  << std::get<0>(tup1) << ", " << std::get<1>(tup1) << ", " << std::ge
  std::cout << "tup2: "  << std::get<0>(tup2) << ", " << std::get<1>(tup2) << ", " << std::ge

  // compare them
  std::cout << "tup1 < tup2: " << (tup1 < tup2) << std::endl;

  std::cout << std::endl;

  // modify a tuple value
  std::get<0>(tup2) = "Second";

  // read the values
  std::cout << "tup1: "  << std::get<0>(tup1) << ", " << std::get<1>(tup1) << ", " << std::ge
  std::cout << "tup2: "  << std::get<0>(tup2) << ", " << std::get<1>(tup2) << ", " << std::ge

  // compare them
  std::cout << "tup1 < tup2: " << (tup1 < tup2) << std::endl;

  std::cout << std::endl;

}
```

The helper function `std::make_tuple`

std::tie and std::ignore

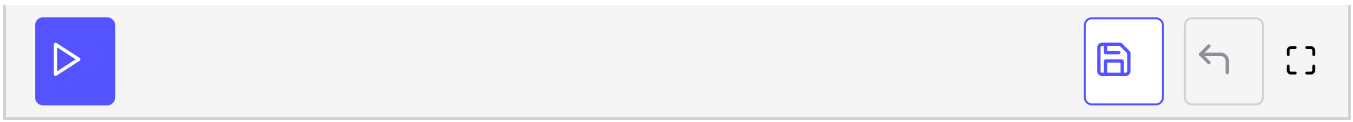
std::tie enables us to create tuples whose elements reference variables. With std::ignore, we can explicitly ignore elements of the tuple.

Let's take a look at an example:

```
#include <functional>
#include <iostream>
#include <tuple>

int main(){

  std::cout << std::endl;

  // make a tuple
```

```
// make a tuple
auto tup1 = std::make_tuple(1, 2, 3);

// print the values
std::cout << "std::tuple tup1: ("<< std::get<0>(tup1) << ", " << std::get<1>(tup1) << ", "

std::cout << std::endl;

int first = 1;
int second = 2;
int third = 3;
int fourth = 4;

// create a tuple with references
auto tup2 = std::make_tuple(std::cref(first), std::ref(second), std::ref(third), fourth);

// print the values
std::cout << "std::tuple tup2: (" << std::get<0>(tup2) << ", " << std::get<1>(tup2) << ", '

std::cout << std::endl;

//change the values
// will not work, because of std::cref(first)
// std::get<0>(tup2)= 1001;
first = 1001;
std::get<1>(tup2) = 1002;
third = 1003;
fourth = 1004;

// print the values
std::cout << "std::tuple tup2: (" << std::get<0>(tup2) << ", " << std::get<1>(tup2) << ", '
std::cout << "global variables: " << first << " " << second << " " << third << " " << fourt

std::cout << std::endl;

first = 1;
second = 2;
third = 3;
fourth = 4;

// create tup3 and set the variables
auto tup3 = std::tie(first, second, third, fourth) = std::make_tuple(1001, 1002, 1003, 1004

// print the values
std::cout << "std::tuple tup3: (" << std::get<0>(tup3) << ", " << std::get<1>(tup3) << ", '
std::cout << "global variables: " << first << " " << second << " " << third << " " << fourt

std::cout << std::endl;

int a;
int b;

// bind the 2th and 4th argument to a and b
std::tie(std::ignore, a, std::ignore, b)= tup3;

// print the values
std::cout << "a: " << a << std::endl;
std::cout << "b: " << b << std::endl;

std::cout << std::endl;

// will also work for std::pair
std::tie(a, b) = std::make_pair(3001, 3002);
```

```
  // print the values
  std::cout << "a: " << a << std::endl;
  std::cout << "b: " << b << std::endl;

  std::cout << std::endl;

}
```



The helper functions `std::tie` and `std::ignore`

Further information

- std::tuple
- std::make_tuple
- std::tie
- std::ignore

In the next lesson, we'll solve an exercise regarding `std::tuple`.