Behavior of std::bind and std::function

Let's take a step deeper into the workings of std::bind and std::function.

WE'LL COVER THE FOLLOWING

- std::bind
- std::function

std::bind

Because of `std::bind`, you can create function objects in a variety of ways:

- bind the arguments to an arbitrary position,
- change the order of the arguments,
- introduce placeholders for arguments,
- partially evaluate functions,
- invoke the newly created function objects, use them in the algorithm of the STL or store them in `std::function`.

std::function

`std::function` can store arbitrary callables in variables. It's a kind of polymorphic function wrapper. A callable may be a lambda function, a function object or a function. `std::function` is always necessary and can't be replaced by `auto`, if you have to specify the type of the callable explicitly.

```
// dispatchTable.cpp
#include <iostream>
#include <map>
#include <functional>
using std::make_pair;
using std::map;

int main(){
  map<const char, std::function<double(double, double)>> tab;
  tab.insert(make_pair('+', [](double a, double b){ return a + b; }));
  tab.insert(make_pair('-', [](double a, double b){ return a - b; }));
  tab.insert(make_pair('*', [](double a, double b){ return a * b; }));
  tab.insert(make_pair('/', [](double a, double b){ return a / b; }));


  std::cout << "3.5 + 4.5\t=  " << tab['+'](3.5, 4.5) << "\n";  //3.5 + 4.5     =  8
  std::cout << "3.5 - 4.5\t=  " << tab['-'](3.5, 4.5) << "\n";  //3.5 - 4.5     =  -1
  std::cout << "3.5 * 4.5\t=  " << tab['*'](3.5, 4.5) << "\n";  //3.5 * 4.5     =  15.75
  std::cout << "3.5 / 4.5\t=  " << tab['/'](3.5, 4.5) << "\n";  //3.5 / 4.5     =  0.777778

  return 0;
}
```

A dispatch table with `std::function`

The type parameter of `std::function` defines the type of callables `std::function` will accept.

Function type	Return type	Type of the arguments
`double(double, double)`	`double`	`double`
`int()`	`int`	
`double(int, double)`	`double`	`int`, `double`
`void()`		

Return type and type of the arguments

Now, let's dive into another utility of the C++ Standard Library – pairs and tuples.