Refactorization with auto

Let's see if we can refactor existing code to work with auto instead of explicit type declarations.

WE'LL COVER THE FOLLOWING

Problems with refactoring data types

`auto` supports the refactorization of our code very well. First, it's very easy to restructure our code if there is no type information.

Second, the compiler automatically takes care of identifying the right types. What does that mean? We will see the answer in the form of a code snippet. To start, here's the code without `auto`:

```
int a = 5;
int b = 10;
int sum =  a * b * 3;
int res = sum + 10;
```

When we replace the variable `b` with type `int` by a double, `10.5`, we have to adjust all the dependent types. That is laborious and dangerous. We have to use the right types and take care of narrowing and other *intelligent phenomenons in C++*.

```
int a2 = 5;
double b2 = 10.5;
double sum2 = a2 * b2 * 3;
double res2 = sum2 * 10.5;
```

Using `auto`

The danger seen above is not present when using `auto`. Everything happens

The danger seen above is not present when using `auto`. Everything happens `auto`-matically. Let us see an example of this:

```
#include <typeinfo>
#include <iostream>

int main(){

  std::cout << std::endl;

  auto a = 5;
  auto b = 10;
  auto sum =  a * b * 3;
  auto res = sum + 10;
  std::cout << "typeid(res).name(): " << typeid(res).name() << std::endl;

  auto a2 = 5;
  auto b2 = 10.5;
  auto sum2 = a2 * b2 * 3;
  auto res2 = sum2 * 10;
  std::cout << "typeid(res2).name(): " << typeid(res2).name() << std::endl;

  auto a3 = 5;
  auto b3 = 10;
  auto sum3 = a3 * b3 * 3.1f;
  auto res3 = sum3 * 10;
  std::cout << "typeid(res3).name(): " << typeid(res3).name() << std::endl;

  std::cout << std::endl;

}
```

The small variations in the code snippet always determine the right type of `res`, `res2`, or `res3`. That's the job of the compiler. The variable `b2` in line 15 is a double and, therefore, `res2` is also a double.

The variable `sum3` in line 22 becomes a floating point number because it's being multiplied by the float literal, `3.1f`. Therefore the final result, `res3`, is also a float point number. To get the data type from the compiler, we have used the `typeid` operator that is defined in the header `typeinfo`.

By now, we should understand how `auto` combines the dynamic behavior of an interpreter with the static behavior of a compiler.

It also protects the programmer, as the right type is automatically given.

Next, we'll look at a couple of examples pertaining to the use of `auto`

Next, we'll look at a couple of examples pertaining to the use of `auto`.