Variant - Visitation

Søren Hansen<sha@ase.au.dk>
V1.1

Introduction

In this exercise we are going to work with `boost::variant` where we are going to handle a predefined number of types.

The exercise comprises of two exercises, the first being simple usage and the second extending this by employing *visitation*, which `boost::variant` supports.

Exercise 1 Defining and using a `boost::variant`

Define a `boost::variant` with a number of different types. The only requirement is that one of them must be user defined.

The task is to write the contents of the `boost::variant` variable to `std::cout` regardlessly of which type is the current one in use. Implementation wise `boost::variant` is *streamable*, if and only if all types it has been parameterized with are *streamables*.

Exercise 1.1 Type not streamable...

To start with our *user defined type* is *not* streamable. In other words, writing something like `std::cout << v << std::endl;` will not work, assuming that `v` is a `boost::variant`.

Therefore inspect the `boost::variant` API and determine which methods to use in order to handle the above.

Exercise 1.2 Fix the streamable issue

To complete this exercise make your type *streamable* by implementing `std::ostream& operator <<(std::ostream&, const <your type>&)`.

Now test the code `std::cout << v << std::endl;` that did not work before.

Exercise 2 Using visitation

As seen in the previous exercise we need to determine which type a given `boost::variant` actually contains before performing some desired deed, which obviously is not surprising. However it means that I/we as developers would have to write the same test code every time.

Fortunately `boost::variant` supports *visitation* that alleviates us from performing this task.

Stuff to do:

- Explain *visitation*
- Write a *functor* that has an function operator overloaded for each type in your `boost::variant`. Remember to inherit properly from `boost::static_visitor`.
- In each of these overloads write out the variable's contents as well as some text such that it is clear from where it came.
- Try to compile your code before all overloads have been completed - What happens and why?

Søren Hansen<sha@ase.au.dk>

Exercise 3 Discuss

Consider where such a type could be useful and present at least one example of such use. This coupled with a explanation as why `boost::variant` in this particular scenario makes for a good solution.