Inline Variables

Let's take a look at how C++ deals with inline variable declaration.

With Non-Static Data Member Initialization introduced in C++11, you can now declare and initialize member variables in one place:

```
class User
{
  int _age {0};
  std::string _name {"unknown"};
};
```

Still, with **static** variables (or const static), ou need a declaration and then a definition in the implementation file.

C++11 with `constexpr` keyword allows you to declare and define static variables in one place, but it's limited to constant expressions only.

Previously, only methods/functions could be specified as `inline`, but now you can do the same with variables, inside a header file.

From the proposal P0386R2:

> A variable declared inline has the same semantics as a function declared inline: it can be defined, identically, in multiple translation units, must be defined in every translation unit in which it is used, and the behaviour of the program is as if there was exactly one variable.

For example:

```
// inside a header file:
struct MyClass
{
  static const int sValue;
};
```

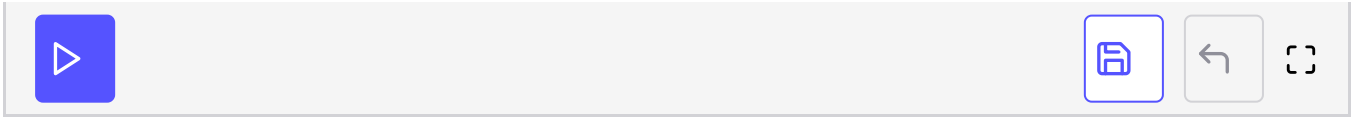
```
// later in the same header file:
inline int const MyClass::sValue = 777;
```

Or even declaration and definition in one place:

```
#include <iostream>
using namespace std;

struct MyClass
{
  inline static const int sValue = 777;
};

int main(){
  MyClass x;
  cout << x.sValue << endl;
}
```



Also, note that `constexpr` variables are `inline` implicitly, so there's no need to use `constexpr inline myVar = 10;`.

An `inline` variable is also more flexible than a `constexpr` variable as it doesn't have to be initialised with a constant expression.

For example, you can initialise an `inline` variable with `rand()`, but it's not possible to do the same with `constexpr` variable.

We'll see an example of why inline variables are useful in making code simpler.