Document number: *P **[TO-BE-CONFIRMED]** R0*
Date: *2022-09-21*
Audience: C++ Standard Committee
Reply-to: *T. P. K. Healy <tho8ma8spkhe8aly at yah8oo dot co8m> Remove all 8's from address*

I. Table of Contents

II. Introduction

The C++ feature of operator overloading for classes allows for syntactical sugar such as:

```
cout << "hello world" << endl;
```

Smart pointers such as "**std::unique_ptr**" and "**std::shared_ptr**" overload the '**->**' operator to give direct access to the referred-to object. The "**std::optional**" class overloads '**->**' to give direct access to the hosted object.

This proposal is centred specifically on making the '**->**' operator more versatile and adaptable, so that classes such as "**std::variant**" can give direct access to the hosted object, for example:

```
struct Laser {
   bool Init(void)
   {
      return true;
   }
};

struct Attenuator {
   bool Init(void)
   {
      return false;
   }
};

std::variant<Laser, Attenuator> obj;

bool retval = obj->Init();     // This here is the syntactical sugar
```

III. Motivation and Scope

It would be useful if "**std::variant**"' could overload the '**->**' operator to give direct access to the currently hosted object, but this currently isn't possible with C++20, as it would require either:
(Possibility 1) A change to the specification of '**std::variant**' stating that it can achieve something which isn't achievable without special compile support (similar to '**std::has_virtual_destructor**').
(Possibility 2) A change to the C++ core language

This proposal is for *Possibility 2* to change the C++ core language.

IV. Impact On the Standard

There will be full backward-compatibility. Old code will be unaffected.

V. Design Decisions

The C++ programming language already has about a hundred keywords as well as six identifiers with special meaning, so I don't want to add to the list. I propose the following new syntax making use of the '**inline**' keyword:

```
struct Device {

   int Init(int const arg)
   {
      return 5 + arg;
   }
};

struct Morpher {

   Device dev;

   operator->
   {
      return  dev.inline;
   }
};
```

The keyword '**inline**' shall be expanded to whatever appears after the '**->**' operator. For example the following code snippet:

```
int main(void)
{
   Morpher obj;
   obj->Init(4);
}
```

shall behave as though the previous code snippet were written as:

```
struct Morpher {

   Device dev;

   auto operator->(void)
   {
      return  dev.Init(4);
   }
};
```

VI. Technical Specifications

The keyword '**inline**' shall expand to:

```
identifier(arguments)
```

In the previous example where we had an invocation in the form of:

```
Morpher obj;
obj->Init(4);
```

The '**inline**' keyword shall expand to:

```
Init(4)
```

The arguments are evaluated once upon entering the routine, and are not evaluated a second time even if '**inline**' appears multiple times in the body of the routine.

If the definition of "**operator->**" contains any static objects, then there is only one instance of these objects for the entire program. For example:

```
operator->
{
   static unsigned counter = 0u;  // There is only one copy of this
                                  //  variable for the entire program

   ++counter;

   if ( counter & 1u ) dev.SetReply(nullptr);

   return  dev.inline;
}
```

If this proposal were to be accepted to the C++ programming language, it would allow us to amend '**std::variant**' as follows:

```
struct std::variant {

   operator->
   {
       return std::visit( [this]<class T>(T &u) { return u.inline; }, *this );
   }
};
```

Of course this new language feature would not just be limited to '**std::variant**'. Any library developer or programmer would be free to write their own classes using this new feature.

VII. Acknowledgements **This is just as draft**

VIII. References **This is just as draft**