Subject: Make THISFN simpler and more powerful
Posted by Lance on Tue, 18 Oct 2022 19:08:20 GMT
View Forum Message <> Reply to Message

in Core/Function.h, we have THISFN defined like this

```
template <class Ptr, class Class, class Res, class... ArgTypes>
Function<Res (ArgTypes...)> MemFn(Ptr object, Res (Class::*method)(ArgTypes...))
{
 return [=](ArgTypes... args) { return (object->*method)(args...); };
}

#define THISFN(x)   MemFn(this, &CLASSNAME::x)
```

This can be done in C++20 with the following

```
#define THISFN(memfun) std::bind_front(&std::remove_cvref_t<decltype(*this)>::memfun, this)
```

Of course this involves only library feature (no core language features), so it should be able to be rewritten to make available in current versons of c++ compilers/libraries that UPP aims to conform to atm.

An additional benefit of above macro definition is that it relieves the library user of the burden of an otherwise useless typedef

```
class MyClass
{
    typedef MyClass CLASSNAME;
//....
}
```

to be able to use thisfn

BTW, thisfn is provided in the UPP library for a good reason: there are occasions where using thisfn is so much easier than using an lambda.

eg.

```
 menu.Set( THISFN(MenuMain) );
```

vs

```
 menu.Set ( [=, this] ( Bar & bar )
  {
   MenuMain ( bar );
  }
```

);

And imagine when the callback accept a few more parameters.
[/code]

U++ is currently aiming at compliance to C++14. What if you want to use it in your project and you know your compiler/c++library are modern enough?

One way is to put the following in a header file and make sure you include it after <Core/Core.h> is included.

```
#ifdef THISFN
#    undef THISFN
#    define THISFN(memfun) ....
#endif
```

Reference:
Why use `std::bind_front` over lambdas in C++20?

Subject: Re: Make THISFN simpler and more powerful
Posted by Lance on Wed, 09 Nov 2022 23:32:44 GMT
View Forum Message <> Reply to Message

This is kind of C++20 related, I am going to reuse this thread for c++ language specific problems or what not.

Here I got a problem: how to detect if a class is Moveable?

A naive first thought would be

```
#include <Core/Core.h>
#include <concepts>

template <class T>
inline constexpr bool is_moveable = std::derived_from<T, Upp::Moveable<T>>;

using namespace Upp;

CONSOLE_APP_MAIN
{
 LOG(is_moveable<Upp::Rect>);
 LOG(is_moveable<Upp::String>);
}
```

Output

```
true
false
```

It turns out Moveable has an optional second template parameter which failed the test in [b]String[/]'s case.

What makes Moveable so special is that it introduced a friend AssertMoveable0 in <Core/Topt.h>

```
template <class T>
inline void AssertMoveable0(T *t) { AssertMoveablePtr(&**t, *t); }
// COMPILATION ERROR HERE MEANS TYPE T WAS NOT MARKED AS Moveable

template <class T, class B = EmptyClass>
struct Moveable : public B {
 friend void AssertMoveable0(T *) {}
};
```

Trying to use it directly, I come up with something like

```
#include <Core/Core.h>
#include <concepts>

template <class T>
inline constexpr bool is_moveable = /*std::derived_from<T, Upp::Moveable<T>>;*/
 requires (T t){ Upp::AssertMoveable0(&t); };

using namespace Upp;

CONSOLE_APP_MAIN
{
 LOG(is_moveable<Upp::Rect>);
 LOG(is_moveable<Upp::String>);
}
```

This time both give correct result, but unfortuately, any class T would give a true result regardless of whether it's derived from Upp::Moveable.

How would you check if a class type is Upp::Moveable ?

BTW, something like this

LOG(std::is_base_of<Ctrl, EditField>);

will fail to compile because of a missing guard in the definition of LOG in <Core/Diag.h>

#define LOG(a) UPP::VppLog() << a << UPP::EOL

I would think here change it to

#define LOG(a) UPP::VppLog() << (a) << UPP::EOL

might be a reasonable thing to do.

Subject: Re: Make THISFN simpler and more powerful
Posted by Oblivion on Thu, 10 Nov 2022 05:29:55 GMT
View Forum Message <> Reply to Message

Maybe a combination of is_constructibles can help here:

```
using namespace Upp;

struct A {
 A(A&&) = delete;
 A&& operator=(A&&) = delete;
};

CONSOLE_APP_MAIN
{
 StdLogSetup(LOG_COUT);

 bool a = std::is_constructible_v<Rect, Rect&&>;
 bool b = std::is_constructible_v<String, String&&>;
 bool c = std::is_constructible_v<A, A&&>;
 RLOG(a);
 RLOG(b);
 RLOG(c);
}
```

Best regards,
Oblivion

Subject: Re: Make THISFN simpler and more powerful

Posted by Lance on Thu, 10 Nov 2022 11:52:33 GMT
View Forum Message <> Reply to Message

Thank you, Oblivion! You code checks if a class have a move ctor?

I was looking for a way to check if a class T has Upp::Moveable<T, optional U> as its base class such that it can be put in a Upp::Vector<T>.

atm I believe the AssertMoveable0 path is dead end. Now the problem becomes:

There is a class T and some class U, devise a way to detect (at compile time of course) if T is defined with something like

class SomeT : public Moveable<SomeT, AnyU>
{...};

Subject: Re: Make THISFN simpler and more powerful
Posted by Lance on Thu, 10 Nov 2022 20:41:07 GMT
View Forum Message <> Reply to Message

Some failed attempts.

```
template <class T>
concept UppMoveable = requires (T t){
   Upp::Vector<T> v;
   v.Add(t);
};
```

This will fail in compilation with a strange error message.

Turns out you cannot do variable declaration in a requires-expression.

Something like this

```
template <class T>
concept UppMoveable = requires (T t, Upp::Vector<T> v){
   v.Add(t);
};
```

compiles fine but doesn't give desired results. Compiler only check if expressions inside a requires-expression are well-formed. Evaluation doesn't take place.

So this won't work. Back to old conclusion: we need a way to check is a class T is

declared/defined with something like

class MyString : public Moveable<MyString, optionalB>{...};

to determine if it can be placed in a Upp::Vector.

Here are some well written articles about concepts and requires-expression.

C++20 Concepts - a Quick Introduction

Requires-expression

Subject: Re: Make THISFN simpler and more powerful
Posted by Lance on Fri, 11 Nov 2022 01:50:42 GMT
View Forum Message <> Reply to Message

Problem solved with a hackish modification to the definition of the Moveable template in <Core/Topt.h>

```
template <class T, class B = EmptyClass>
struct Moveable : public B {
 typedef B MoveableBase; // Add this line
 friend void AssertMoveable0(T *) {}
};
```

Now it works like a charm :)

```
#include <Core/Core.h>
#include <concepts>
#include <type_traits>

template <class T>
concept UppMoveable = std::is_trivial_v<T> || requires{
 typename T::MoveableBase;
 requires std::derived_from<T,
          Upp::Moveable<T, typename T::MoveableBase>
        >;
};

CONSOLE_APP_MAIN
{
 using namespace Upp;

 struct D{
  D(){}
 };
```

```
 struct F{
  F() = default;
 };

 struct E : Moveable<E, D>{
  E(){}
 };


 DUMP(UppMoveable<String>);//true
 DUMP(UppMoveable<D>);//false
 DUMP(UppMoveable<F>);//true, note the difference between D and F
 DUMP(UppMoveable<E>);//true
 DUMP(UppMoveable<int64>);//true
}
```

This is a hack. I am still interested to know if there is a way to do it without resorting to touch <Core/Topt.h>.

Subject: Re: Make THISFN simpler and more powerful
Posted by Lance on Tue, 08 Oct 2024 16:38:32 GMT
View Forum Message <> Reply to Message

OK, this is not even c++ language related. But c++ is to be blamed.

Strange padding.

```
#include <iostream>

struct A{
    int a;
    unsigned b:16; // for demonstration purpose only
    bool c:1;
};

int main()
{
   std::cout<<"sizeof(A) is "<<sizeof(A)<<std::endl;
}
```

You would expect the result be 8, but on windows MSBT22x64 and CLANG64 will give 12.
(CLANG64 and GCC64 on linux give 8, which is naturally and as expected)

If you think that's not weired enough, it goes further.

```
#include <iostream>

struct A{
 int a;
 union{
  int dummy;
  struct{
      unsigned v1:16;
      bool     v2:1;
  };
 };
};


int main()
{
 A a;
 a.v2 = true;
 a.dummy = 0;

 std::cout<<"a.v2 is "<<a.v2<<std::endl;
}
```

The output is true. The compiler is smart enough to disregard the union request and put dummy and the unnamed struct object side by side, instead of start from the same address.

Maybe there is something I did wrong?

Subject: Re: Make THISFN simpler and more powerful
Posted by Lance on Tue, 08 Oct 2024 16:42:02 GMT
View Forum Message <> Reply to Message

oh, I said c++ is to be blamed. The lack of basic binary standard is creating a lot a pains and extra work for programmers. And I don't know c++ enough to say how much additional benefits it provides by allowing compiler vendors the freedom to make their own choice on these area. It's more work for CLANG for sure, as it has to imitate vc++ on Windows and g++ otherwise.

Subject: Re: Make THISFN simpler and more powerful
Posted by Didier on Tue, 08 Oct 2024 17:25:33 GMT
View Forum Message <> Reply to Message

Hello Lance,

What compilers do with bit field is not covered by a standard and is implementation defined... so they do just about whatever they want.
BUT, something important is missing in you're code ! If you really want everything to be side by side, bit wise, you have to specify PACKED option on you're structs
search the web : __attribute__((packed, aligned(X))) or __PACKED__
This will force the compiler to 'pack' all the bits together not waisting anything : so you will have a stable size.

Note : positionning of the inside the struct is implementation defined ... so some compilers put them in on order, and others the other way around ;)
==> you're code won't be very portable

Subject: Re: Make THISFN simpler and more powerful
Posted by Lance on Tue, 08 Oct 2024 18:01:45 GMT
View Forum Message <> Reply to Message

Didier wrote on Tue, 08 October 2024 13:25Hello Lance,

What compilers do with bit field is not covered by a standard and is implementation defined... so they do just about whatever they want.
BUT, something important is missing in you're code ! If you really want everything to be side by side, bit wise, you have to specify PACKED option on you're structs
search the web : __attribute__((packed, aligned(X))) or __PACKED__
This will force the compiler to 'pack' all the bits together not waisting anything : so you will have a stable size.

Note : positionning of the inside the struct is implementation defined ... so some compilers put them in on order, and others the other way around ;)
==> you're code won't be very portable

Thanks for your reply, Didier.

I am fine with padding. I am having issues with the way MSVC padding this one to unnecessary increase object size.

Also, for union, I expect objects taking same memory address. MSVC failed to deliver. I don't know what the standard says. But it's a tradition dated back to old C.

Are compiler free to reorder data members with the same access privileges? I will have to double check. Thanks again.

Subject: Re: Make THISFN simpler and more powerful
Posted by Lance on Wed, 09 Oct 2024 03:59:40 GMT

It's less a padding issue than a bitfield issue.

The question is answered
here.

Subject: Re: Make THISFN simpler and more powerful
Posted by Lance on Wed, 09 Oct 2024 14:09:28 GMT

I don't like that fact that compilers are allowed to stuff random padding bits in a bitfield as they like, but it's actually standard compliant.

In the above example, change unsigned to byte (Upp::byte of course) actually removes the extra cost on total storage usage. But the padding MSVC inserts vs GCC's sequential packed bits will result in binary incompatibilities. Worse, some old c tricks no longer work with MSVC.

A somewhat more realistic though simplified example.

```
class SomeFormat{
...
private:
   Font font;
   Color paper, ink, highlight;
   union{
      int32 dummy;
      struct{
         byte info1:3;
         byte info2:5;

         // allow individual font properties
         // to be Null for multi-tier composition
         bool faceNotNull:1;
         bool heightNotNull:1;
         bool widthNotNull:1;
         bool boldNotNull:1;
         bool strikeoutNotNull:1;
         bool underlineNotNull:1;
         bool italicNotNull:1;

      };
   };


};
```

In old c days, if we want to check if all Font properties are set, we can simply

```
bool SomeFormat::AllFontPropertiesSet()const
{
#define SOMEFORMAT_MASK (((1<<7)-1)<<8)
   return (dummy & SOMEFORMAT_MASK) == SOMEFORMAT_MASK;
#define SOMEFORMAT_MASK
}
```

And to mark all font properties as set(non-Null)

```
SomeFormat::SetAllFontProperties()
{
#define SOMEFORMAT_MASK (((1<<7)-1)<<8)
   return dummy |= SOMEFORMAT_MASK;
#define SOMEFORMAT_MASK
}
```

etc. With GCC, you can still do things like that. Total predictability. Fully appreciated.

End of the day, what benefits MSVC is going to achieve by padding random bits? I can see if a bitfield crosses a machine's fast-integer boundary (a few bits in previous FAST-INTEGER and a few in the following), there will be extra cpu cost involved. Other than that, what's going to be saved?

Thumbs down for MSVC on this regard.

Subject: Re: Make THISFN simpler and more powerful
Posted by Lance on Wed, 09 Oct 2024 14:17:43 GMT
View Forum Message <> Reply to Message

C++23 provides a remedy to write portable bitfield code, without having to resort to unnecessary bit by bit initialization/update.

Here is an example.

```
union Flags{
 int32 dummy;
 struct{
  byte borderLeft:3;
  byte borderRight:3;
  byte borderTop:3;
  byte borderBottom:3;
```

```
 byte halign:2;
 byte valign:2; //16th bit

 bool  faceNotNull:1;
 bool  boldNotNull:1;
 bool  heightNotNull:1;
 bool  widthNotNull:1;
 bool  underlineNotNull:1;
 bool  italicNotNull:1;
 bool  strikeoutNotNull:1; //23rd bit
};

Flags() : dummy(0){ static_assert(sizeof(*this)==sizeof(dummy)); }

static constexpr int32 FontMask()
{
 Flags f;
 f.faceNotNull = true;
 f.boldNotNull = true;
 f.heightNotNull = true;
 f.widthNotNull = true;
 f.underlineNotNull = true;
 f.italicNotNull = true;
 f.strikeoutNotNull = true;
 return f.dummy;
}

void Border(int border)
{
 borderLeft = borderRight = borderTop = borderBottom = border;
}

void SetAllFontProperties()
{
 dummy |= FontMask();
}

void ClearAllFontProperties()
{
 dummy &= ~FontMask();
}

bool AllFontPropertiesSet()const
{
 return ( dummy & FontMask() ) == FontMask();
}
};
```

It's a lot more work. But there is one added benefit: if you change the bitfields definition and add a few bits in front of the font properties group, you have a better chance not to break existing code unknowingly.

Subject: Re: Make THISFN simpler and more powerful
Posted by Didier on Wed, 09 Oct 2024 17:32:35 GMT
View Forum Message <> Reply to Message

In fact the extra padding inserted by compilers is intented for alignement issues (although for bitfields this should not be an issue) this is why you're code works.
But apparently MSVC has his own way of working and pack(1) doesn't seem to be taken into account ... very strange

---

Subject: Re: Make THISFN simpler and more powerful
Posted by Lance on Thu, 10 Oct 2024 12:01:36 GMT
View Forum Message <> Reply to Message

Didier wrote on Wed, 09 October 2024 13:32In fact the extra padding inserted by compilers is intented for alignement issues (although for bitfields this should not be an issue) this is why you're code works.
But apparently MSVC has his own way of working and pack(1) doesn't seem to be taken into account ... very strange

:)

MSC is doing the same thing. It's in ISO C too. Even a c struct is an opaque object that should be accessed through functions if more than one compilers is involved.

Subject: Re: Make THISFN simpler and more powerful
Posted by Lance on Thu, 10 Oct 2024 12:06:48 GMT
View Forum Message <> Reply to Message

Lance wrote on Wed, 09 October 2024 10:17C++23 provides a remedy to write portable bitfield code, without having to resort to unnecessary bit by bit initialization/update.

Here is an example.

```
union Flags{
 int32 dummy;
 struct{
  byte borderLeft:3;
  byte borderRight:3;
```

```
 byte borderTop:3;
 byte borderBottom:3;
 byte halign:2;
 byte valign:2; //16th bit

 bool  faceNotNull:1;
 bool  boldNotNull:1;
 bool  heightNotNull:1;
 bool  widthNotNull:1;
 bool  underlineNotNull:1;
 bool  italicNotNull:1;
 bool  strikeoutNotNull:1; //23rd bit
};

Flags() : dummy(0){ static_assert(sizeof(*this)==sizeof(dummy)); }

static constexpr int32 FontMask()
{
 Flags f;
 f.faceNotNull = true;
 f.boldNotNull = true;
 f.heightNotNull = true;
 f.widthNotNull = true;
 f.underlineNotNull = true;
 f.italicNotNull = true;
 f.strikeoutNotNull = true;
 return f.dummy;
}

void Border(int border)
{
 borderLeft = borderRight = borderTop = borderBottom = border;
}

void SetAllFontProperties()
{
 dummy |= FontMask();
}

void ClearAllFontProperties()
{
 dummy &= ~FontMask();
}

bool AllFontPropertiesSet()const
{
 return ( dummy & FontMask() ) == FontMask();
}
```

```
};
```

It's a lot more work. But there is one added benefit: if you change the bitfields definition and add a few bits in front of the font properties group, you have a better chance not to break existing code unknowingly.

with std=c++23, gcc and msvc accept the above code. But clang rejects it, for good reason. The union default constructor used in constexpr ...FontMask()const should be constexpr modified too.

```
 constexpr Flags() : dummy(0){
  static_assert(sizeof(*this)==sizeof(dummy));
      }
```

Subject: Re: Make THISFN simpler and more powerful
Posted by Lance on Tue, 05 Nov 2024 01:25:15 GMT
View Forum Message <> Reply to Message

I had a brief discussion with regard to constexpr with Mirek a few days ago
here

Today I did a simple test with a more complicated case.

```
#include <Core/Core.h>

using namespace Upp;

CONSOLE_APP_MAIN
{
 Color c(200,50,76);
 if ( c == Color(200, 50, 76) )
 {
  RLOG("Yes");
 }else{
  RLOG("NO");
 }
}
```

Compiled to the following 338 lines of assembly code

```
 .text
 .file "TestColorConstant.cpp"
 .globl main                     # -- Begin function main
```

```
 .p2align 4, 0x90
 .type main,@function
main:                                   # @main
 .cfi_startproc
# %bb.0:
 pushq %rbp
 .cfi_def_cfa_offset 16
 .cfi_offset %rbp, -16
 movq %rsp, %rbp
 .cfi_def_cfa_register %rbp
 subq $32, %rsp
 movl $0, -4(%rbp)
 movl %edi, -8(%rbp)
 movq %rsi, -16(%rbp)
 movq %rdx, -24(%rbp)
 movl -8(%rbp), %edi
 movq -16(%rbp), %rsi
 movq -24(%rbp), %rdx
 callq _ZN3Upp9AppInit__EiPPKcS2_@PLT
 leaq _Z14ConsoleMainFn_v(%rip), %rdi
 callq _ZN3Upp12AppExecute__EPFvvE@PLT
 callq _ZN3Upp9AppExit__Ev@PLT
 callq _ZN3Upp11GetExitCodeEv@PLT
 addq $32, %rsp
 popq %rbp
 .cfi_def_cfa %rsp, 8
 retq
.Lfunc_end0:
 .size main, .Lfunc_end0-main
 .cfi_endproc
                                        # -- End function
 .globl _Z14ConsoleMainFn_v             # -- Begin function _Z14ConsoleMainFn_v
 .p2align 4, 0x90
 .type _Z14ConsoleMainFn_v,@function
_Z14ConsoleMainFn_v:                    # @_Z14ConsoleMainFn_v
 .cfi_startproc
# %bb.0:
 pushq %rbp
 .cfi_def_cfa_offset 16
 .cfi_offset %rbp, -16
 movq %rsp, %rbp
 .cfi_def_cfa_register %rbp
 subq $16, %rsp
 leaq -4(%rbp), %rdi
 movl $200, %esi
 movl $50, %edx
 movl $76, %ecx
 callq _ZN3Upp5ColorC2Eiii
```

```
 leaq -8(%rbp), %rdi
 movl $200, %esi
 movl $50, %edx
 movl $76, %ecx
 callq _ZN3Upp5ColorC2Eiii
 movl -8(%rbp), %esi
 leaq -4(%rbp), %rdi
 callq _ZNK3Upp5ColoreqES0_
 testb $1, %al
 jne .LBB1_1
 jmp .LBB1_2
.LBB1_1:
 callq _ZN3Upp6VppLogEv@PLT
 movq %rax, %rdi
 leaq .L.str(%rip), %rsi
 callq _ZN3UpplsERNS_6StreamEPKc
 movq %rax, %rdi
 xorl %esi, %esi
 callq _ZN3Upp6StreamlsENS_7EOLenumE
 jmp .LBB1_3
.LBB1_2:
 callq _ZN3Upp6VppLogEv@PLT
 movq %rax, %rdi
 leaq .L.str.1(%rip), %rsi
 callq _ZN3UpplsERNS_6StreamEPKc
 movq %rax, %rdi
 xorl %esi, %esi
 callq _ZN3Upp6StreamlsENS_7EOLenumE
.LBB1_3:
 addq $16, %rsp
 popq %rbp
 .cfi_def_cfa %rsp, 8
 retq
.Lfunc_end1:
 .size _Z14ConsoleMainFn_v, .Lfunc_end1-_Z14ConsoleMainFn_v
 .cfi_endproc
                                # -- End function
 .section .text._ZN3Upp5ColorC2Eiii,"axG",@progbits,_ZN3Upp5ColorC2Eiii,comdat
 .weak _ZN3Upp5ColorC2Eiii       # -- Begin function _ZN3Upp5ColorC2Eiii
 .p2align 4, 0x90
 .type _ZN3Upp5ColorC2Eiii,@function
_ZN3Upp5ColorC2Eiii:                    # @_ZN3Upp5ColorC2Eiii
 .cfi_startproc
# %bb.0:
 pushq %rbp
 .cfi_def_cfa_offset 16
 .cfi_offset %rbp, -16
 movq %rsp, %rbp
```

```
 .cfi_def_cfa_register %rbp
 subq $32, %rsp
 movq %rdi, -8(%rbp)
 movl %esi, -12(%rbp)
 movl %edx, -16(%rbp)
 movl %ecx, -20(%rbp)
 movq -8(%rbp), %rax
 movq %rax, -32(%rbp)              # 8-byte Spill
 movl -12(%rbp), %eax
 movb %al, %dl
 movl -16(%rbp), %eax
 movb %al, %cl
 movl -20(%rbp), %eax
                                   # kill: def $al killed $al killed $eax
 movzbl %dl, %edi
 movzbl %cl, %esi
 movzbl %al, %edx
 callq _ZN3Upp3RGBEhhh
 movl %eax, %ecx
 movq -32(%rbp), %rax              # 8-byte Reload
 movl %ecx, (%rax)
 addq $32, %rsp
 popq %rbp
 .cfi_def_cfa %rsp, 8
 retq
.Lfunc_end2:
 .size _ZN3Upp5ColorC2Eiii, .Lfunc_end2-_ZN3Upp5ColorC2Eiii
 .cfi_endproc
                                   # -- End function
 .section .text._ZNK3Upp5ColoreqES0_,"axG",@progbits,_ZNK3Upp5ColoreqES0_,comdat
 .weak _ZNK3Upp5ColoreqES0_        # -- Begin function _ZNK3Upp5ColoreqES0_
 .p2align 4, 0x90
 .type _ZNK3Upp5ColoreqES0_,@function
_ZNK3Upp5ColoreqES0_:              # @_ZNK3Upp5ColoreqES0_
 .cfi_startproc
# %bb.0:
 pushq %rbp
 .cfi_def_cfa_offset 16
 .cfi_offset %rbp, -16
 movq %rsp, %rbp
 .cfi_def_cfa_register %rbp
 movl %esi, -4(%rbp)
 movq %rdi, -16(%rbp)
 movq -16(%rbp), %rax
 movl (%rax), %eax
 cmpl -4(%rbp), %eax
 sete %al
 andb $1, %al
```

```
 movzbl %al, %eax
 popq %rbp
 .cfi_def_cfa %rsp, 8
 retq
.Lfunc_end3:
 .size _ZNK3Upp5ColoreqES0_, .Lfunc_end3-_ZNK3Upp5ColoreqES0_
 .cfi_endproc
                                        # -- End function
 .section .text._ZN3UpplsERNS_6StreamEPKc,"axG",@progbits,_ZN3UpplsERNS_6StreamEPKc
,comdat
 .weak _ZN3UpplsERNS_6StreamEPKc       # -- Begin function _ZN3UpplsERNS_6StreamEPKc
 .p2align 4, 0x90
 .type _ZN3UpplsERNS_6StreamEPKc,@function
_ZN3UpplsERNS_6StreamEPKc:             # @_ZN3UpplsERNS_6StreamEPKc
 .cfi_startproc
# %bb.0:
 pushq %rbp
 .cfi_def_cfa_offset 16
 .cfi_offset %rbp, -16
 movq %rsp, %rbp
 .cfi_def_cfa_register %rbp
 subq $16, %rsp
 movq %rdi, -8(%rbp)
 movq %rsi, -16(%rbp)
 movq -8(%rbp), %rdi
 movq -16(%rbp), %rsi
 callq _ZN3Upp6Stream3PutEPKc@PLT
 movq -8(%rbp), %rax
 addq $16, %rsp
 popq %rbp
 .cfi_def_cfa %rsp, 8
 retq
.Lfunc_end4:
 .size _ZN3UpplsERNS_6StreamEPKc, .Lfunc_end4-_ZN3UpplsERNS_6StreamEPKc
 .cfi_endproc
                                        # -- End function
 .section .text._ZN3Upp6StreamlsENS_7EOLenumE,"axG",@progbits,_ZN3Upp6StreamlsENS_7
EOLenumE,comdat
 .weak _ZN3Upp6StreamlsENS_7EOLenumE   # -- Begin function
_ZN3Upp6StreamlsENS_7EOLenumE
 .p2align 4, 0x90
 .type _ZN3Upp6StreamlsENS_7EOLenumE,@function
_ZN3Upp6StreamlsENS_7EOLenumE:         # @_ZN3Upp6StreamlsENS_7EOLenumE
 .cfi_startproc
# %bb.0:
 pushq %rbp
 .cfi_def_cfa_offset 16
 .cfi_offset %rbp, -16
```

```
 movq %rsp, %rbp
 .cfi_def_cfa_register %rbp
 subq $32, %rsp
 movq %rdi, -8(%rbp)
 movl %esi, -12(%rbp)
 movq -8(%rbp), %rdi
 movq %rdi, -24(%rbp)               # 8-byte Spill
 callq _ZN3Upp6Stream6PutEolEv
 movq -24(%rbp), %rax               # 8-byte Reload
 addq $32, %rsp
 popq %rbp
 .cfi_def_cfa %rsp, 8
 retq
.Lfunc_end5:
 .size _ZN3Upp6StreamlsENS_7EOLenumE, .Lfunc_end5-_ZN3Upp6StreamlsENS_7EOLenumE
 .cfi_endproc
                                    # -- End function
 .section .text._ZN3Upp3RGBEhhh,"axG",@progbits,_ZN3Upp3RGBEhhh,comdat
 .weak _ZN3Upp3RGBEhhh              # -- Begin function _ZN3Upp3RGBEhhh
 .p2align 4, 0x90
 .type _ZN3Upp3RGBEhhh,@function
_ZN3Upp3RGBEhhh:                    # @_ZN3Upp3RGBEhhh
 .cfi_startproc
# %bb.0:
 pushq %rbp
 .cfi_def_cfa_offset 16
 .cfi_offset %rbp, -16
 movq %rsp, %rbp
 .cfi_def_cfa_register %rbp
 movb %dl, %al
 movb %sil, %cl
 movb %dil, %dl
 movb %dl, -1(%rbp)
 movb %cl, -2(%rbp)
 movb %al, -3(%rbp)
 movzbl -1(%rbp), %eax
 movzbl -2(%rbp), %ecx
 shll $8, %ecx
 orl %ecx, %eax
 movzbl -3(%rbp), %ecx
 shll $16, %ecx
 orl %ecx, %eax
 popq %rbp
 .cfi_def_cfa %rsp, 8
 retq
.Lfunc_end6:
 .size _ZN3Upp3RGBEhhh, .Lfunc_end6-_ZN3Upp3RGBEhhh
 .cfi_endproc
```

```
                                        # -- End function
 .section .text._ZN3Upp6Stream6PutEolEv,"axG",@progbits,_ZN3Upp6Stream6PutEolEv,comdat
 .weak _ZN3Upp6Stream6PutEolEv         # -- Begin function _ZN3Upp6Stream6PutEolEv
 .p2align 4, 0x90
 .type _ZN3Upp6Stream6PutEolEv,@function
_ZN3Upp6Stream6PutEolEv:               # @_ZN3Upp6Stream6PutEolEv
 .cfi_startproc
# %bb.0:
 pushq %rbp
 .cfi_def_cfa_offset 16
 .cfi_offset %rbp, -16
 movq %rsp, %rbp
 .cfi_def_cfa_register %rbp
 subq $16, %rsp
 movq %rdi, -8(%rbp)
 movq -8(%rbp), %rdi
 movl $10, %esi
 callq _ZN3Upp6Stream3PutEi
 addq $16, %rsp
 popq %rbp
 .cfi_def_cfa %rsp, 8
 retq
.Lfunc_end7:
 .size _ZN3Upp6Stream6PutEolEv, .Lfunc_end7-_ZN3Upp6Stream6PutEolEv
 .cfi_endproc
                                        # -- End function
 .section .text._ZN3Upp6Stream3PutEi,"axG",@progbits,_ZN3Upp6Stream3PutEi,comdat
 .weak _ZN3Upp6Stream3PutEi            # -- Begin function _ZN3Upp6Stream3PutEi
 .p2align 4, 0x90
 .type _ZN3Upp6Stream3PutEi,@function
_ZN3Upp6Stream3PutEi:                  # @_ZN3Upp6Stream3PutEi
 .cfi_startproc
# %bb.0:
 pushq %rbp
 .cfi_def_cfa_offset 16
 .cfi_offset %rbp, -16
 movq %rsp, %rbp
 .cfi_def_cfa_register %rbp
 subq $32, %rsp
 movq %rdi, -8(%rbp)
 movl %esi, -12(%rbp)
 movq -8(%rbp), %rcx
 movq %rcx, -24(%rbp)                  # 8-byte Spill
 movq 24(%rcx), %rax
 cmpq 40(%rcx), %rax
 jae .LBB8_2
# %bb.1:
 movq -24(%rbp), %rdx                  # 8-byte Reload
```

```
 movl -12(%rbp), %eax
 movb %al, %cl
 movq 24(%rdx), %rax
 movq %rax, %rsi
 addq $1, %rsi
 movq %rsi, 24(%rdx)
 movb %cl, (%rax)
 jmp .LBB8_3
.LBB8_2:
 movq -24(%rbp), %rdi               # 8-byte Reload
 movl -12(%rbp), %esi
 movq (%rdi), %rax
 callq *(%rax)
.LBB8_3:
 addq $32, %rsp
 popq %rbp
 .cfi_def_cfa %rsp, 8
 retq
.Lfunc_end8:
 .size _ZN3Upp6Stream3PutEi, .Lfunc_end8-_ZN3Upp6Stream3PutEi
 .cfi_endproc
                                    # -- End function
 .type .L.str,@object                # @.str
 .section .rodata.str1.1,"aMS",@progbits,1
.L.str:
 .asciz "Yes"
 .size .L.str, 4

 .type .L.str.1,@object              # @.str.1
.L.str.1:
 .asciz "NO"
 .size .L.str.1, 3

 .section ".linker-options","e",@llvm_linker_options
 .ident "Ubuntu clang version 18.1.3 (1ubuntu1)"
 .section ".note.GNU-stack","",@progbits
 .addrsig
 .addrsig_sym _ZN3Upp9AppInit__EiPPKcS2_
 .addrsig_sym _ZN3Upp12AppExecute__EPFvvE
 .addrsig_sym _Z14ConsoleMainFn_v
 .addrsig_sym _ZN3Upp9AppExit__Ev
 .addrsig_sym _ZN3Upp11GetExitCodeEv
 .addrsig_sym _ZNK3Upp5ColoreqES0_
 .addrsig_sym _ZN3UpplsERNS_6StreamEPKc
 .addrsig_sym _ZN3Upp6VppLogEv
 .addrsig_sym _ZN3Upp6StreamlsENS_7EOLenumE
 .addrsig_sym _ZN3Upp3RGBEhhh
 .addrsig_sym _ZN3Upp6Stream3PutEPKc
```

```
 .addrsig_sym _ZN3Upp6Stream6PutEolEv
 .addrsig_sym _ZN3Upp6Stream3PutEi
```

While the if constexpr version

```
#include <Core/Core.h>

using namespace Upp;

CONSOLE_APP_MAIN
{
 constexpr Color c(200,50,76);
 if constexpr ( c == Color(200, 50, 76) )
 {
  RLOG("Yes");
 }else{
  RLOG("NO");
 }
}
```

compiles to the following 214 lines of assembly code

```
 .text
 .file "TestColorConstant.cpp"
 .globl main                     # -- Begin function main
 .p2align 4, 0x90
 .type main,@function
main:                            # @main
 .cfi_startproc
# %bb.0:
 pushq %rbp
 .cfi_def_cfa_offset 16
 .cfi_offset %rbp, -16
 movq %rsp, %rbp
 .cfi_def_cfa_register %rbp
 subq $32, %rsp
 movl $0, -4(%rbp)
 movl %edi, -8(%rbp)
 movq %rsi, -16(%rbp)
 movq %rdx, -24(%rbp)
 movl -8(%rbp), %edi
 movq -16(%rbp), %rsi
 movq -24(%rbp), %rdx
 callq _ZN3Upp9AppInit__EiPPKcS2_@PLT
 leaq _Z14ConsoleMainFn_v(%rip), %rdi
 callq _ZN3Upp12AppExecute__EPFvvE@PLT
 callq _ZN3Upp9AppExit__Ev@PLT
```

```
 callq _ZN3Upp11GetExitCodeEv@PLT
 addq $32, %rsp
 popq %rbp
 .cfi_def_cfa %rsp, 8
 retq
.Lfunc_end0:
 .size main, .Lfunc_end0-main
 .cfi_endproc
                                # -- End function
 .globl _Z14ConsoleMainFn_v           # -- Begin function _Z14ConsoleMainFn_v
 .p2align 4, 0x90
 .type _Z14ConsoleMainFn_v,@function
_Z14ConsoleMainFn_v:                  # @_Z14ConsoleMainFn_v
 .cfi_startproc
# %bb.0:
 pushq %rbp
 .cfi_def_cfa_offset 16
 .cfi_offset %rbp, -16
 movq %rsp, %rbp
 .cfi_def_cfa_register %rbp
 subq $16, %rsp
 movl .L__const._Z14ConsoleMainFn_v.c(%rip), %eax
 movl %eax, -4(%rbp)
 callq _ZN3Upp6VppLogEv@PLT
 movq %rax, %rdi
 leaq .L.str(%rip), %rsi
 callq _ZN3UpplsERNS_6StreamEPKc
 movq %rax, %rdi
 xorl %esi, %esi
 callq _ZN3Upp6StreamlsENS_7EOLenumE
 addq $16, %rsp
 popq %rbp
 .cfi_def_cfa %rsp, 8
 retq
.Lfunc_end1:
 .size _Z14ConsoleMainFn_v, .Lfunc_end1-_Z14ConsoleMainFn_v
 .cfi_endproc
                                # -- End function
 .section .text._ZN3UpplsERNS_6StreamEPKc,"axG",@progbits,_ZN3UpplsERNS_6StreamEPKc
,comdat
 .weak _ZN3UpplsERNS_6StreamEPKc       # -- Begin function _ZN3UpplsERNS_6StreamEPKc
 .p2align 4, 0x90
 .type _ZN3UpplsERNS_6StreamEPKc,@function
_ZN3UpplsERNS_6StreamEPKc:            # @_ZN3UpplsERNS_6StreamEPKc
 .cfi_startproc
# %bb.0:
 pushq %rbp
 .cfi_def_cfa_offset 16
```

```
 .cfi_offset %rbp, -16
 movq %rsp, %rbp
 .cfi_def_cfa_register %rbp
 subq $16, %rsp
 movq %rdi, -8(%rbp)
 movq %rsi, -16(%rbp)
 movq -8(%rbp), %rdi
 movq -16(%rbp), %rsi
 callq _ZN3Upp6Stream3PutEPKc@PLT
 movq -8(%rbp), %rax
 addq $16, %rsp
 popq %rbp
 .cfi_def_cfa %rsp, 8
 retq
.Lfunc_end2:
 .size _ZN3UpplsERNS_6StreamEPKc, .Lfunc_end2-_ZN3UpplsERNS_6StreamEPKc
 .cfi_endproc
                                # -- End function
 .section .text._ZN3Upp6StreamlsENS_7EOLenumE,"axG",@progbits,_ZN3Upp6StreamlsENS_7
EOLenumE,comdat
 .weak _ZN3Upp6StreamlsENS_7EOLenumE   # -- Begin function
_ZN3Upp6StreamlsENS_7EOLenumE
 .p2align 4, 0x90
 .type _ZN3Upp6StreamlsENS_7EOLenumE,@function
_ZN3Upp6StreamlsENS_7EOLenumE:         # @_ZN3Upp6StreamlsENS_7EOLenumE
 .cfi_startproc
# %bb.0:
 pushq %rbp
 .cfi_def_cfa_offset 16
 .cfi_offset %rbp, -16
 movq %rsp, %rbp
 .cfi_def_cfa_register %rbp
 subq $32, %rsp
 movq %rdi, -8(%rbp)
 movl %esi, -12(%rbp)
 movq -8(%rbp), %rdi
 movq %rdi, -24(%rbp)               # 8-byte Spill
 callq _ZN3Upp6Stream6PutEolEv
 movq -24(%rbp), %rax               # 8-byte Reload
 addq $32, %rsp
 popq %rbp
 .cfi_def_cfa %rsp, 8
 retq
.Lfunc_end3:
 .size _ZN3Upp6StreamlsENS_7EOLenumE, .Lfunc_end3-_ZN3Upp6StreamlsENS_7EOLenumE
 .cfi_endproc
                                # -- End function
 .section .text._ZN3Upp6Stream6PutEolEv,"axG",@progbits,_ZN3Upp6Stream6PutEolEv,comdat
```

```
 .weak _ZN3Upp6Stream6PutEolEv        # -- Begin function _ZN3Upp6Stream6PutEolEv
 .p2align 4, 0x90
 .type _ZN3Upp6Stream6PutEolEv,@function
_ZN3Upp6Stream6PutEolEv:              # @_ZN3Upp6Stream6PutEolEv
 .cfi_startproc
# %bb.0:
 pushq %rbp
 .cfi_def_cfa_offset 16
 .cfi_offset %rbp, -16
 movq %rsp, %rbp
 .cfi_def_cfa_register %rbp
 subq $16, %rsp
 movq %rdi, -8(%rbp)
 movq -8(%rbp), %rdi
 movl $10, %esi
 callq _ZN3Upp6Stream3PutEi
 addq $16, %rsp
 popq %rbp
 .cfi_def_cfa %rsp, 8
 retq
.Lfunc_end4:
 .size _ZN3Upp6Stream6PutEolEv, .Lfunc_end4-_ZN3Upp6Stream6PutEolEv
 .cfi_endproc
                                    # -- End function
 .section .text._ZN3Upp6Stream3PutEi,"axG",@progbits,_ZN3Upp6Stream3PutEi,comdat
 .weak _ZN3Upp6Stream3PutEi          # -- Begin function _ZN3Upp6Stream3PutEi
 .p2align 4, 0x90
 .type _ZN3Upp6Stream3PutEi,@function
_ZN3Upp6Stream3PutEi:                 # @_ZN3Upp6Stream3PutEi
 .cfi_startproc
# %bb.0:
 pushq %rbp
 .cfi_def_cfa_offset 16
 .cfi_offset %rbp, -16
 movq %rsp, %rbp
 .cfi_def_cfa_register %rbp
 subq $32, %rsp
 movq %rdi, -8(%rbp)
 movl %esi, -12(%rbp)
 movq -8(%rbp), %rcx
 movq %rcx, -24(%rbp)              # 8-byte Spill
 movq 24(%rcx), %rax
 cmpq 40(%rcx), %rax
 jae .LBB5_2
# %bb.1:
 movq -24(%rbp), %rdx              # 8-byte Reload
 movl -12(%rbp), %eax
 movb %al, %cl
```

```
 movq 24(%rdx), %rax
 movq %rax, %rsi
 addq $1, %rsi
 movq %rsi, 24(%rdx)
 movb %cl, (%rax)
 jmp .LBB5_3
.LBB5_2:
 movq -24(%rbp), %rdi               # 8-byte Reload
 movl -12(%rbp), %esi
 movq (%rdi), %rax
 callq *(%rax)
.LBB5_3:
 addq $32, %rsp
 popq %rbp
 .cfi_def_cfa %rsp, 8
 retq
.Lfunc_end5:
 .size _ZN3Upp6Stream3PutEi, .Lfunc_end5-_ZN3Upp6Stream3PutEi
 .cfi_endproc
                                    # -- End function
 .type .L__const._Z14ConsoleMainFn_v.c,@object # @__const._Z14ConsoleMainFn_v.c
 .section .rodata.cst4,"aM",@progbits,4
 .p2align 2, 0x0
.L__const._Z14ConsoleMainFn_v.c:
 .long 4993736                      # 0x4c32c8
 .size .L__const._Z14ConsoleMainFn_v.c, 4

 .type .L.str,@object                # @.str
 .section .rodata.str1.1,"aMS",@progbits,1
.L.str:
 .asciz "Yes"
 .size .L.str, 4

 .section ".linker-options","e",@llvm_linker_options
 .ident "Ubuntu clang version 18.1.3 (1ubuntu1)"
 .section ".note.GNU-stack","",@progbits
 .addrsig
 .addrsig_sym _ZN3Upp9AppInit__EiPPKcS2_
 .addrsig_sym _ZN3Upp12AppExecute__EPFvvE
 .addrsig_sym _Z14ConsoleMainFn_v
 .addrsig_sym _ZN3Upp9AppExit__Ev
 .addrsig_sym _ZN3Upp11GetExitCodeEv
 .addrsig_sym _ZN3UpplsERNS_6StreamEPKc
 .addrsig_sym _ZN3Upp6VppLogEv
 .addrsig_sym _ZN3Upp6StreamlsENS_7EOLenumE
 .addrsig_sym _ZN3Upp6Stream3PutEPKc
 .addrsig_sym _ZN3Upp6Stream6PutEolEv
 .addrsig_sym _ZN3Upp6Stream3PutEi
```

Without going into details of assembly code, we know if constexpr indeed provides additional optimization opportunities.

Note in oder for the second example to compile, certain modifications are required in <Core/Color.h>, and a minimum std=c++20 might be required.

Subject: Re: Make THISFN simpler and more powerful
Posted by Lance on Tue, 05 Nov 2024 01:33:16 GMT
View Forum Message <> Reply to Message

Overview of New Features in C++23

Subject: Re: Make THISFN simpler and more powerful
Posted by Lance on Wed, 06 Nov 2024 23:49:55 GMT
View Forum Message <> Reply to Message

Writing a custom iterator in modern C++

Comparisons in C++20 by Barry Revzin

What is the <=> ("spaceship", three-way comparison) operator in C++?
I didn't know I also need to provide an operator == with customized spaceship operator.

Subject: Re: Make THISFN simpler and more powerful
Posted by Lance on Fri, 29 Nov 2024 01:30:27 GMT
View Forum Message <> Reply to Message

std::print in C++23

Keeping Time in C++: How to use the std::chrono API

Github workflow files for building U++ on Windows, Linux & MacOS

How to move packages from bazaar to github repo and UppHub
