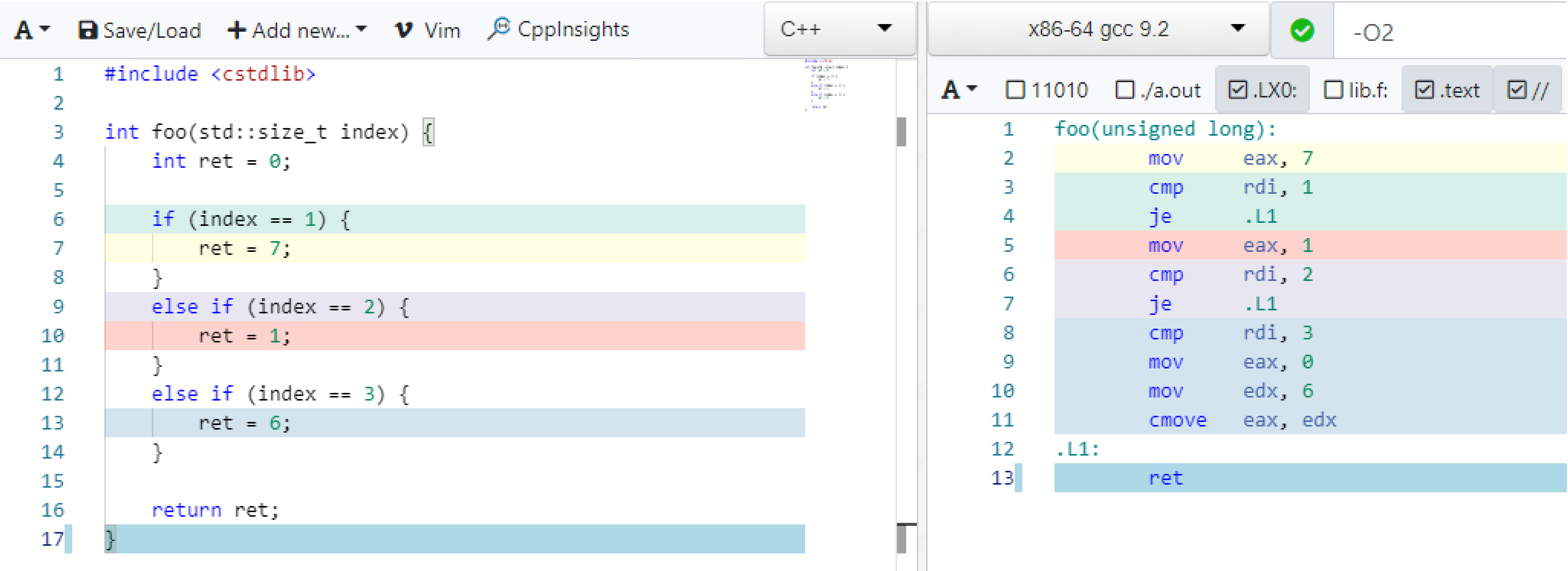
Forcing the Compiler to Generate Jump Tables

Lennox Shou Hao Ho
lennox.ho@intel.com

A Trivial Example



C++

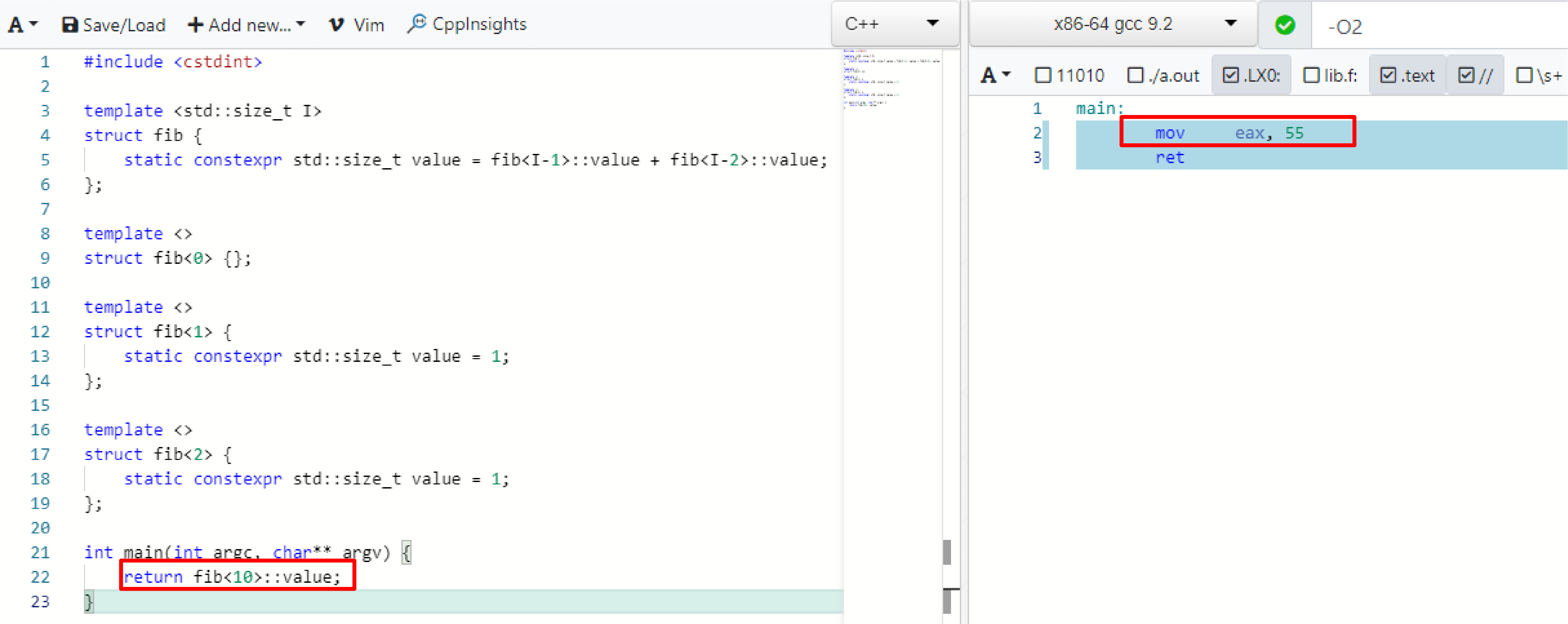
x86-64 gcc 9.2 -O2

☐ 11010 ☐ ./a.out ☑ .LX0: ☐ lib.f: ☑ .text ☑ //

```cpp
#include <cstdlib>

int foo(std::size_t index) {
    int ret = 0;

    if (index == 1) {
        ret = 7;
    }
    else if (index == 2) {
        ret = 1;
    }
    else if (index == 3) {
        ret = 6;
    }

    return ret;
}
```

```asm
foo(unsigned long):
        mov     eax, 7
        cmp     rdi, 1
        je      .L1
        mov     eax, 1
        cmp     rdi, 2
        je      .L1
        cmp     rdi, 3
        mov     eax, 0
        mov     edx, 6
        cmove   eax, edx
.L1:
        ret
```

But First, Automatically Generate if-else cases



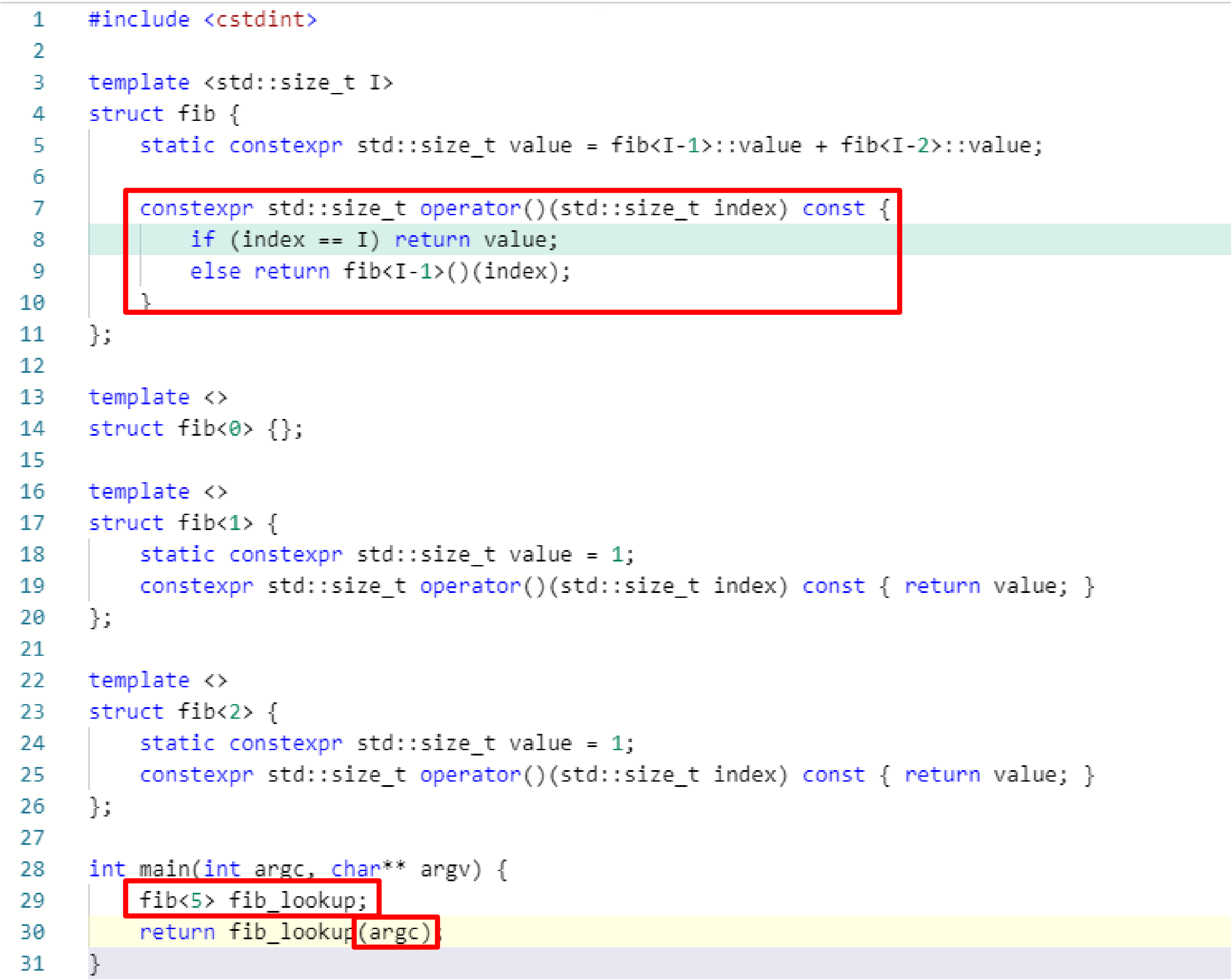
x86-64 gcc 9.2 -O2

☐ 11010 ☐ ./a.out ☑ .LX0: ☐ lib.f: ☑ .text ☑ // ☐ \s+

```cpp
#include <cstdint>

template <std::size_t I>
struct fib {
    static constexpr std::size_t value = fib<I-1>::value + fib<I-2>::value;
};

template <>
struct fib<0> {};

template <>
struct fib<1> {
    static constexpr std::size_t value = 1;
};

template <>
struct fib<2> {
    static constexpr std::size_t value = 1;
};

int main(int argc, char** argv) {
    return fib<10>::value;
}
```

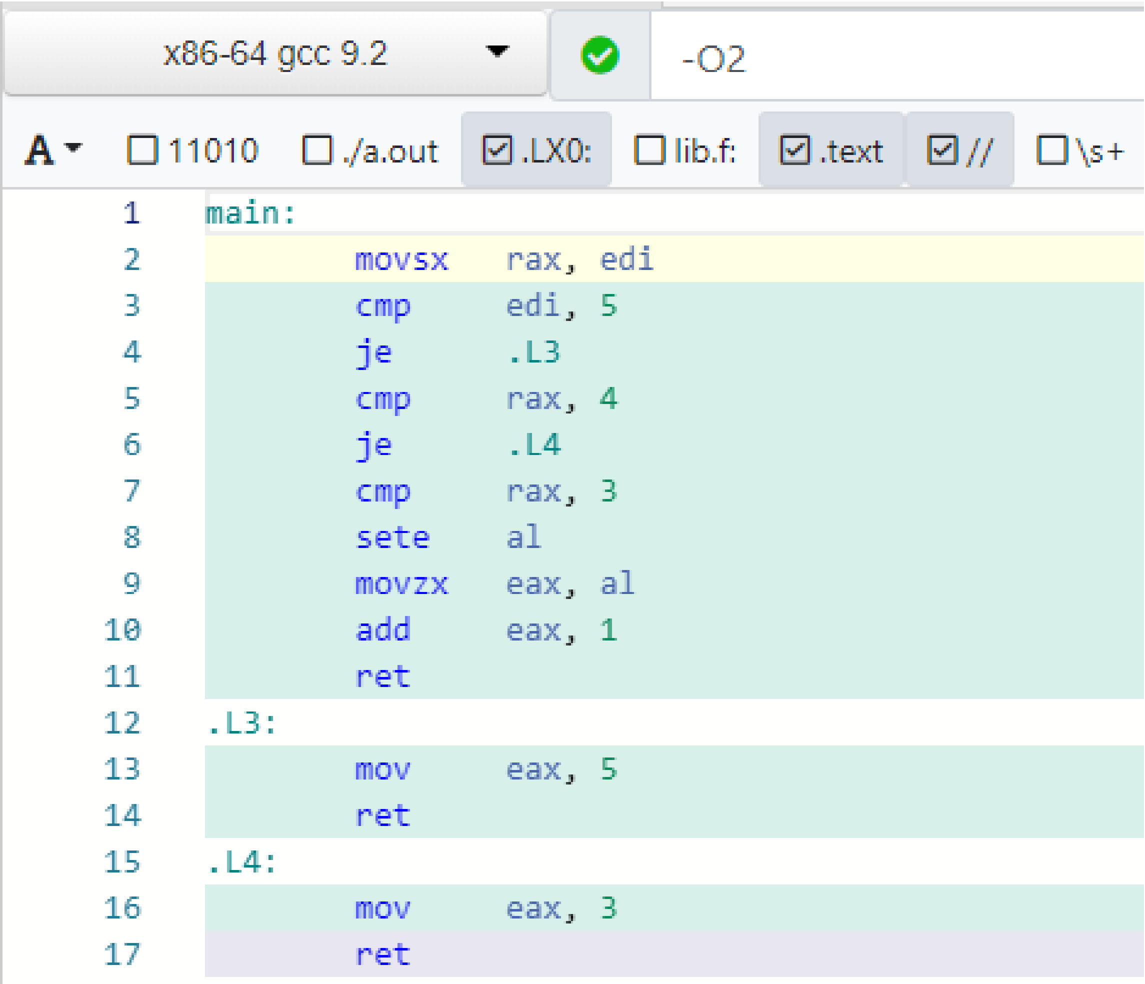
```asm
main:
        mov     eax, 55
        ret
```

But First, Automatically Generate if-else cases



```cpp
#include <cstdint>

template <std::size_t I>
struct fib {
    static constexpr std::size_t value = fib<I-1>::value + fib<I-2>::value;

    constexpr std::size_t operator()(std::size_t index) const {
        if (index == I) return value;
        else return fib<I-1>()(index);
    }
};

template <>
struct fib<0> {};

template <>
struct fib<1> {
    static constexpr std::size_t value = 1;
    constexpr std::size_t operator()(std::size_t index) const { return value; }
};

template <>
struct fib<2> {
    static constexpr std::size_t value = 1;
    constexpr std::size_t operator()(std::size_t index) const { return value; }
};

int main(int argc, char** argv) {
    fib<5> fib_lookup;
    return fib_lookup(argc);
}
```

Fibonacci : 5 numbers

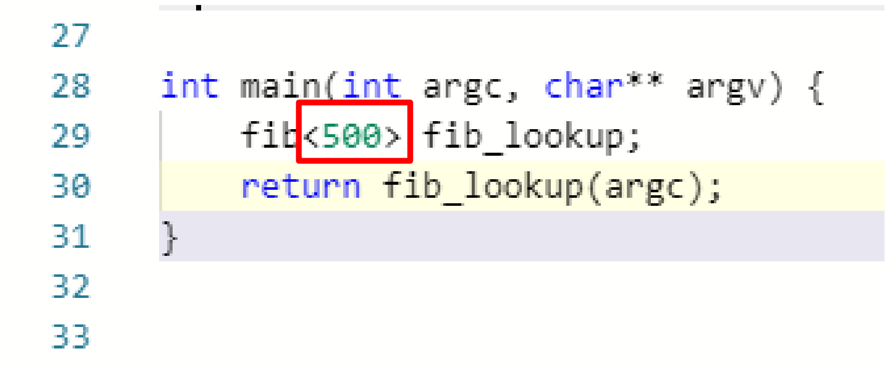


x86-64 gcc 9.2 ▾ ✓ -O2

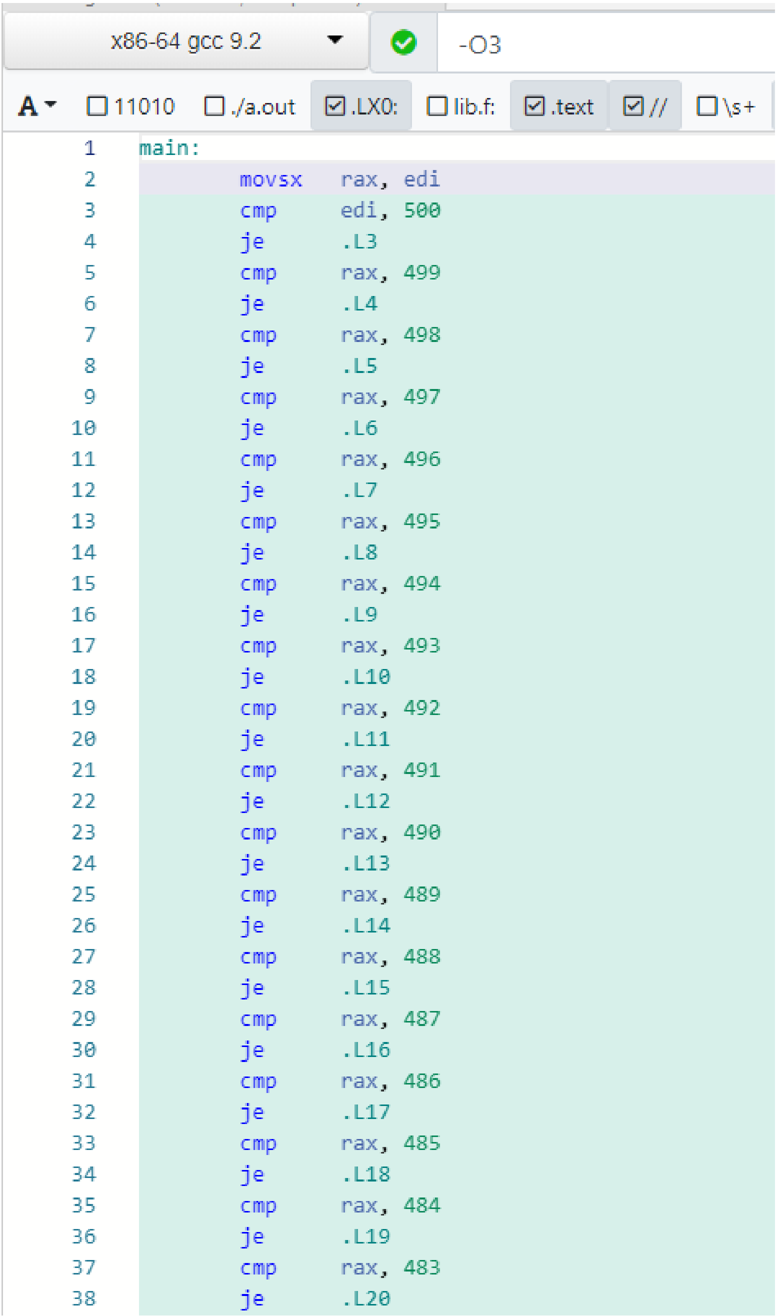
A ▾ ☐ 11010 ☐ ./a.out ☑ .LX0: ☐ lib.f: ☑ .text ☑ // ☐ \s+

```
main:
        movsx   rax, edi
        cmp     edi, 5
        je      .L3
        cmp     rax, 4
        je      .L4
        cmp     rax, 3
        sete    al
        movzx   eax, al
        add     eax, 1
        ret
.L3:
        mov     eax, 5
        ret
.L4:
        mov     eax, 3
        ret
```

Fibonacci : 500 numbers



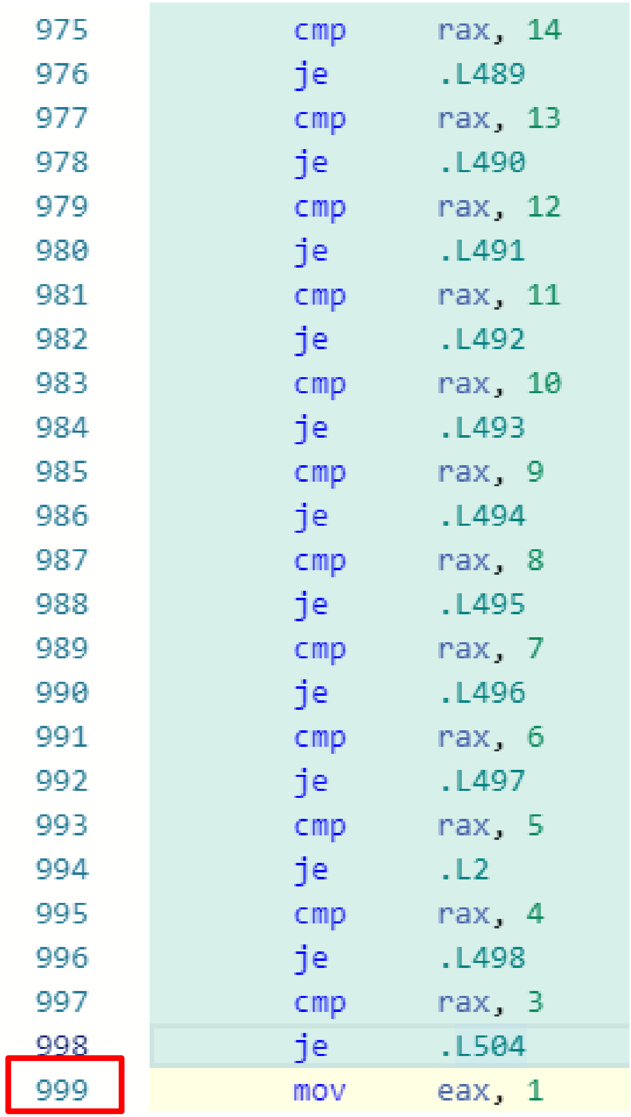
```
int main(int argc, char** argv) {
    fib<500> fib_lookup;
    return fib_lookup(argc);
}
```



x86-64 gcc 9.2 ▾ -O3

```
main:
        movsx   rax, edi
        cmp     edi, 500
        je      .L3
        cmp     rax, 499
        je      .L4
        cmp     rax, 498
        je      .L5
        cmp     rax, 497
        je      .L6
        cmp     rax, 496
        je      .L7
        cmp     rax, 495
        je      .L8
        cmp     rax, 494
        je      .L9
        cmp     rax, 493
        je      .L10
        cmp     rax, 492
        je      .L11
        cmp     rax, 491
        je      .L12
        cmp     rax, 490
        je      .L13
        cmp     rax, 489
        je      .L14
        cmp     rax, 488
        je      .L15
        cmp     rax, 487
        je      .L16
        cmp     rax, 486
        je      .L17
        cmp     rax, 485
        je      .L18
        cmp     rax, 484
        je      .L19
        cmp     rax, 483
        je      .L20
```

Fibonacci : 500 numbers

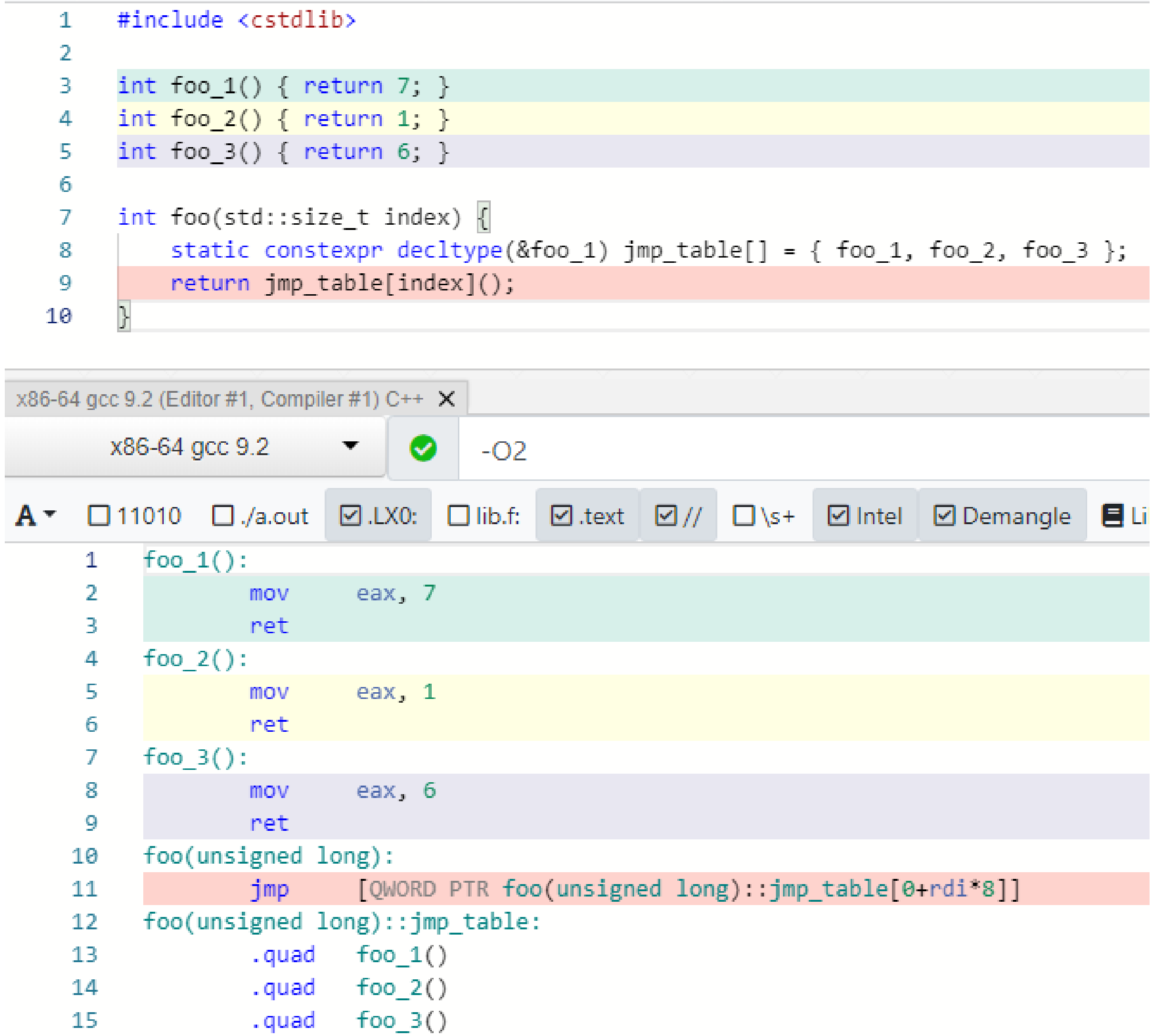


```
975        cmp     rax, 14
976        je      .L489
977        cmp     rax, 13
978        je      .L490
979        cmp     rax, 12
980        je      .L491
981        cmp     rax, 11
982        je      .L492
983        cmp     rax, 10
984        je      .L493
985        cmp     rax, 9
986        je      .L494
987        cmp     rax, 8
988        je      .L495
989        cmp     rax, 7
990        je      .L496
991        cmp     rax, 6
992        je      .L497
993        cmp     rax, 5
994        je      .L2
995        cmp     rax, 4
996        je      .L498
997        cmp     rax, 3
998        je      .L504
999        mov     eax, 1
```

More Practical Problem : Jump Table + Variant Visit

- Currently all major implementations of Variant classes (std::variant, boost::variant, boost::variant2) perform linear lookup
- This is generally fine, until N is large and/or the visitor function is non-trivial
- Want the choice to use jump tables if we so choose

How to Generate Jump Table? Basic Observation



```cpp
#include <cstdlib>

int foo_1() { return 7; }
int foo_2() { return 1; }
int foo_3() { return 6; }

int foo(std::size_t index) {
    static constexpr decltype(&foo_1) jmp_table[] = { foo_1, foo_2, foo_3 };
    return jmp_table[index]();
}
```

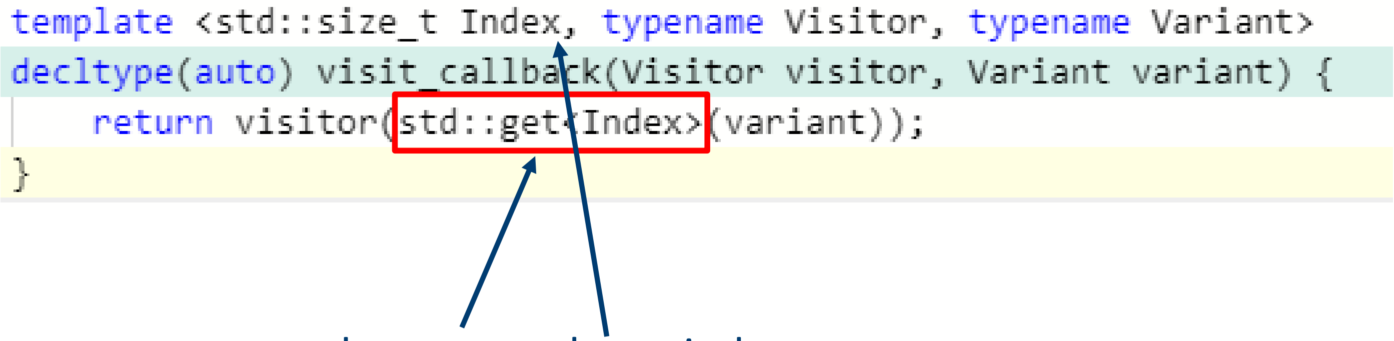
x86-64 gcc 9.2 (Editor #1, Compiler #1) C++ ✕

x86-64 gcc 9.2 ▾ ✓ -O2

A ▾ □ 11010 □ ./a.out ☑ .LX0: □ lib.f: ☑ .text ☑ // □ \s+ ☑ Intel ☑ Demangle Li

```asm
foo_1():
        mov     eax, 7
        ret
foo_2():
        mov     eax, 1
        ret
foo_3():
        mov     eax, 6
        ret
foo(unsigned long):
        jmp     [QWORD PTR foo(unsigned long)::jmp_table[0+rdi*8]]
foo(unsigned long)::jmp_table:
        .quad   foo_1()
        .quad   foo_2()
        .quad   foo_3()
```

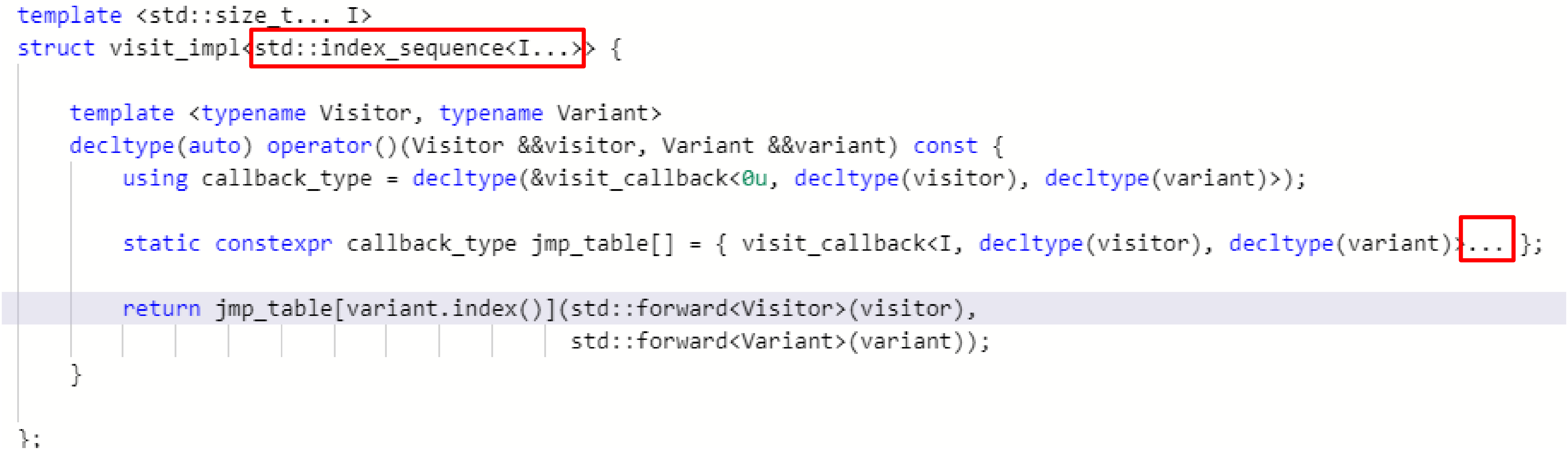
Use Function Template to Generate Callbacks



```
template <std::size_t Index, typename Visitor, typename Variant>
decltype(auto) visit_callback(Visitor visitor, Variant variant) {
    return visitor(std::get<Index>(variant));
}
```

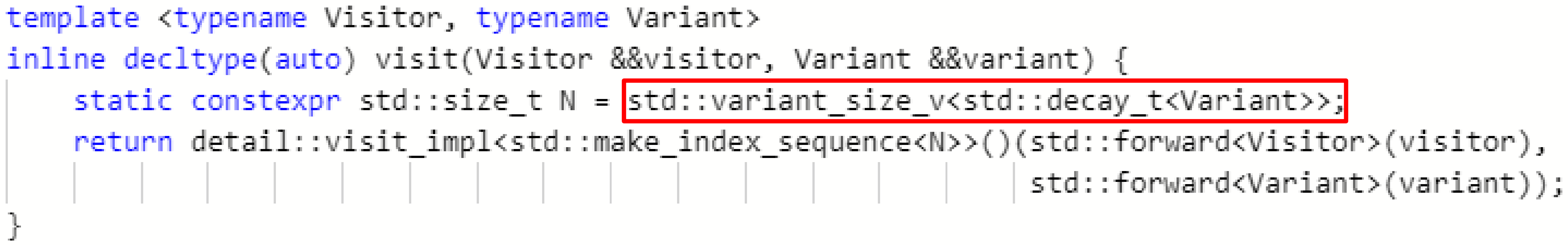
std::get can take an index

Use Variadic Template to Generate Indices



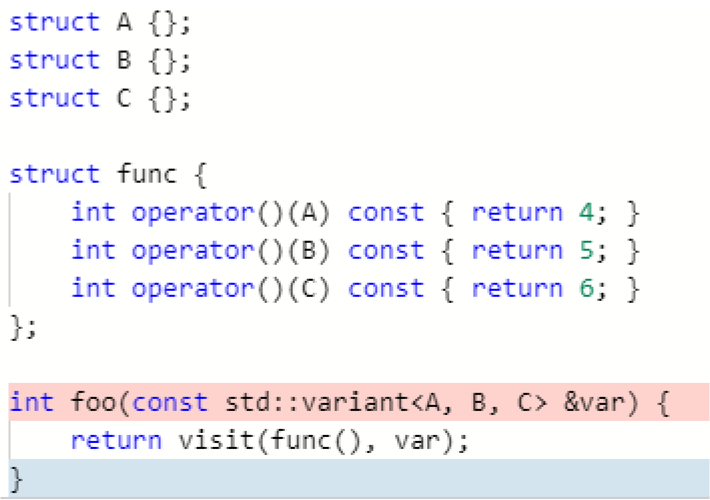
```cpp
template <std::size_t... I>
struct visit_impl<std::index_sequence<I...>> {

    template <typename Visitor, typename Variant>
    decltype(auto) operator()(Visitor &&visitor, Variant &&variant) const {
        using callback_type = decltype(&visit_callback<0u, decltype(visitor), decltype(variant)>);

        static constexpr callback_type jmp_table[] = { visit_callback<I, decltype(visitor), decltype(variant)>... };

        return jmp_table[variant.index()](std::forward<Visitor>(visitor),
                                          std::forward<Variant>(variant));
    }

};
```

Get Variant Size

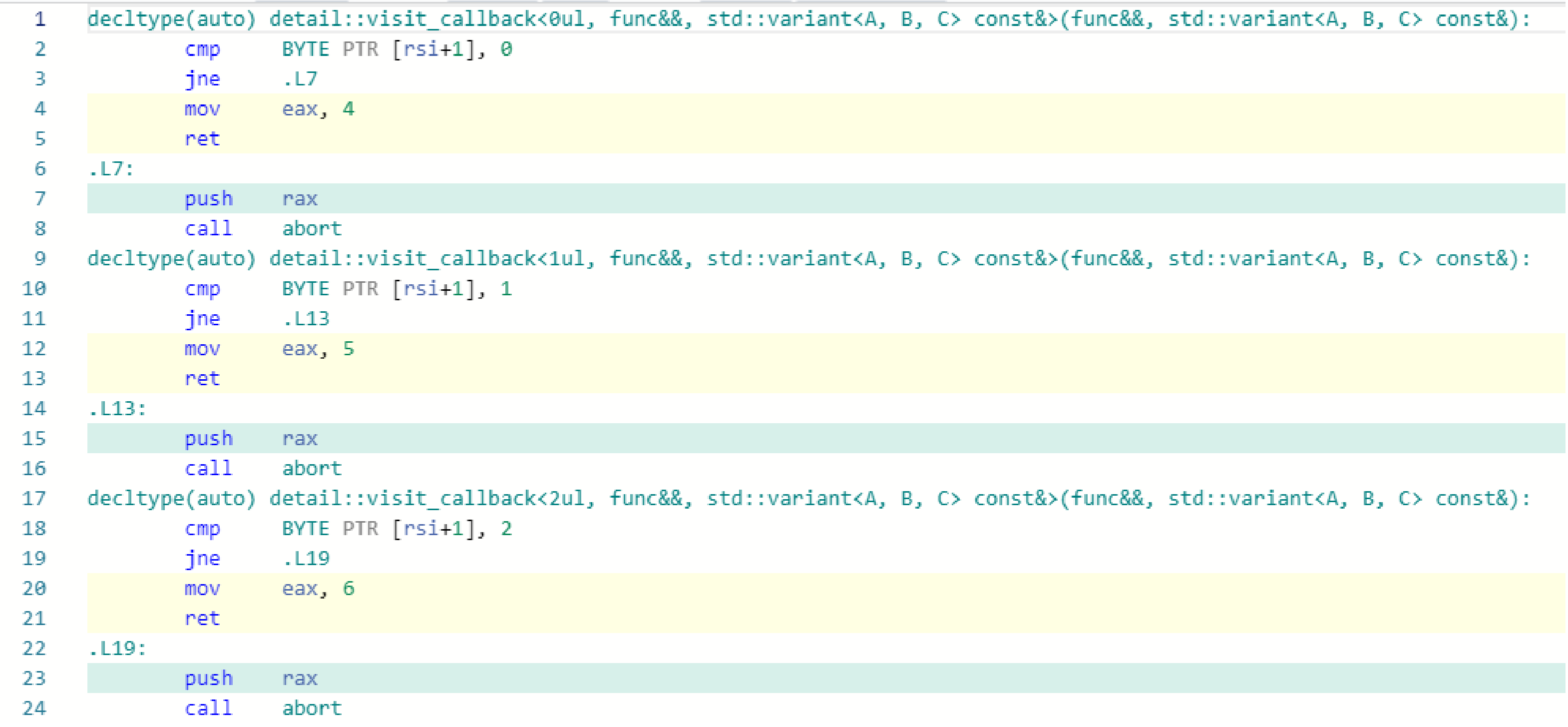


```cpp
template <typename Visitor, typename Variant>
inline decltype(auto) visit(Visitor &&visitor, Variant &&variant) {
    static constexpr std::size_t N = std::variant_size_v<std::decay_t<Variant>>;
    return detail::visit_impl<std::make_index_sequence<N>>()(std::forward<Visitor>(visitor),
                                                             std::forward<Variant>(variant));
}
```

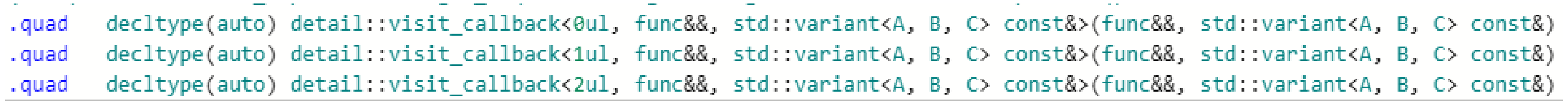
Generated ASM



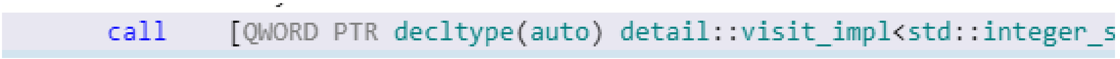
```
struct A {};
struct B {};
struct C {};

struct func {
    int operator()(A) const { return 4; }
    int operator()(B) const { return 5; }
    int operator()(C) const { return 6; }
};

int foo(const std::variant<A, B, C> &var) {
    return visit(func(), var);
}
```



```
decltype(auto) detail::visit_callback<0ul, func&&, std::variant<A, B, C> const&>(func&&, std::variant<A, B, C> const&):
        cmp     BYTE PTR [rsi+1], 0
        jne     .L7
        mov     eax, 4
        ret
.L7:
        push    rax
        call    abort
decltype(auto) detail::visit_callback<1ul, func&&, std::variant<A, B, C> const&>(func&&, std::variant<A, B, C> const&):
        cmp     BYTE PTR [rsi+1], 1
        jne     .L13
        mov     eax, 5
        ret
.L13:
        push    rax
        call    abort
decltype(auto) detail::visit_callback<2ul, func&&, std::variant<A, B, C> const&>(func&&, std::variant<A, B, C> const&):
        cmp     BYTE PTR [rsi+1], 2
        jne     .L19
        mov     eax, 6
        ret
.L19:
        push    rax
        call    abort
```



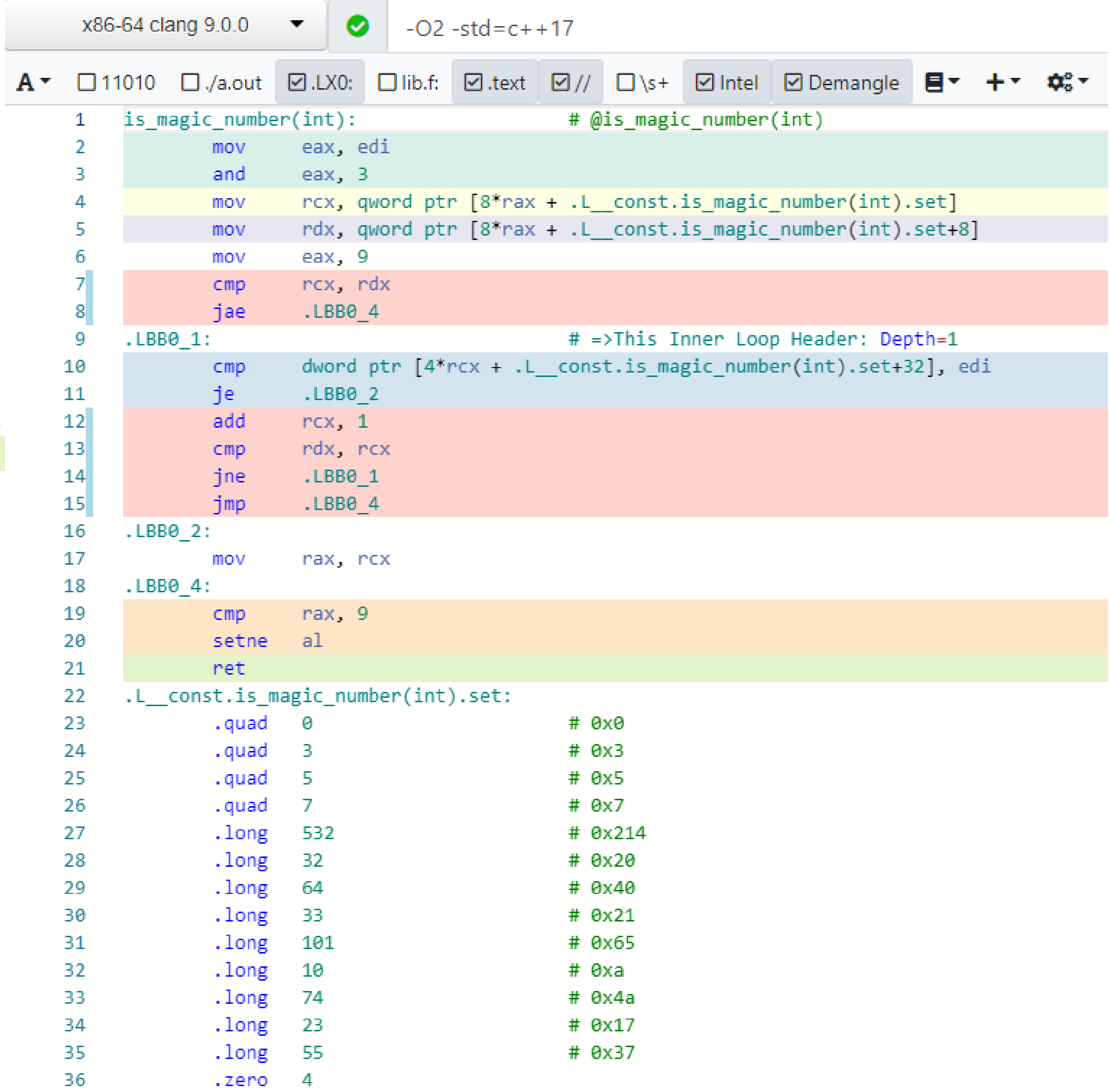
```
.quad   decltype(auto) detail::visit_callback<0ul, func&&, std::variant<A, B, C> const&>(func&&, std::variant<A, B, C> const&)
.quad   decltype(auto) detail::visit_callback<1ul, func&&, std::variant<A, B, C> const&>(func&&, std::variant<A, B, C> const&)
.quad   decltype(auto) detail::visit_callback<2ul, func&&, std::variant<A, B, C> const&>(func&&, std::variant<A, B, C> const&)
```



```
        call    [QWORD PTR decltype(auto) detail::visit_impl<std::integer_s
```

Sneak Peek – constexpr hash map/set

```cpp
bool is_magic_number(int num) {
    constexpr std::array values = { 33, 23, 532, 32, 10, 55, 74, 101, 64 };
    constexpr std::size_t num_buckets = 4;

    constexpr auto set = make_constexpr_hash_set<num_buckets, int_hash>(values);
    return set.contains(num);
}
```



x86-64 clang 9.0.0 -O2 -std=c++17

A▾ □ 11010 □ ./a.out ☑ .LX0: □ lib.f: ☑ .text ☑ // □ \s+ ☑ Intel ☑ Demangle

```asm
is_magic_number(int):                   # @is_magic_number(int)
        mov     eax, edi
        and     eax, 3
        mov     rcx, qword ptr [8*rax + .L__const.is_magic_number(int).set]
        mov     rdx, qword ptr [8*rax + .L__const.is_magic_number(int).set+8]
        mov     eax, 9
        cmp     rcx, rdx
        jae     .LBB0_4
.LBB0_1:                                # =>This Inner Loop Header: Depth=1
        cmp     dword ptr [4*rcx + .L__const.is_magic_number(int).set+32], edi
        je      .LBB0_2
        add     rcx, 1
        cmp     rdx, rcx
        jne     .LBB0_1
        jmp     .LBB0_4
.LBB0_2:
        mov     rax, rcx
.LBB0_4:
        cmp     rax, 9
        setne   al
        ret
.L__const.is_magic_number(int).set:
        .quad   0                       # 0x0
        .quad   3                       # 0x3
        .quad   5                       # 0x5
        .quad   7                       # 0x7
        .long   532                     # 0x214
        .long   32                      # 0x20
        .long   64                      # 0x40
        .long   33                      # 0x21
        .long   101                     # 0x65
        .long   10                      # 0xa
        .long   74                      # 0x4a
        .long   23                      # 0x17
        .long   55                      # 0x37
        .zero   4
```