Name mangling

C++’s function resolution

scc@teamt5.org

TEAMT5
杜 浦 數 位 安 全
Persistent Cyber Threat Hunters

Outline

- Namespace and Member function
- static, anonymous namespace, static member function
- Overloading and Candidate functions
- SFINAE
- ~~Further issues: concepts and constraints~~

Outline

- Namespace and Member function
- static, anonymous namespace, static member function
- Overloading and Candidate functions
- SFINAE
- ~~Further issues: concepts and constraints~~

Introduced in C++20.
But we might need
std::enable_if and
constexpr if .

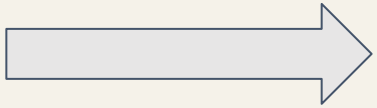
C++ use mangling to separate functions.

Named namespace

```
namespace SCC {
int foo();
}
```

Named namespace

```
namespace SCC {
int foo();
}
```



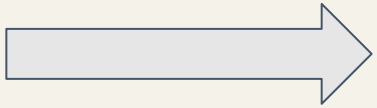
```
SCC::foo()
```

Member functions

```
class SCC {
int foo();
}
```

Member functions

```
class SCC {
int foo();
}
```



```
SCC::foo()
```

What’s the difference?

No any differences.

static function

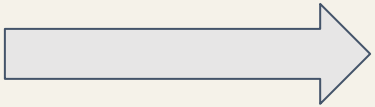
TEAMT5
杜 浦 數 位 安 全

```
static int foo();
```

static function

TEAMT5
杜 浦 數 位 安 全

```
static int foo();
```



```
foo()
```

Linker couldn’t see.

TEAMT5
杜 浦 數 位 安 全

yangzhixuan@yangzhiuandeAir:/tmp/foo

```
→  foo c++ a.cpp -c --static ; bat a.cpp;
a.cpp:1:19: warning: non-void function does not return a value [-Wretu
rn-type]
static int foo() {}
                  ^
1 warning generated.

File: a.cpp

1  static int foo() {}

→  foo nm a.o
0000000000000000 t ltmp0
→  foo
```

Linker could see.

TEAMT5
杜 浦 數 位 安 全

yangzhixuan@yangzhiuandeAir:/tmp/foo

```
➜  foo c++ a.cpp -c --static ; bat a.cpp;
a.cpp:1:12: warning: non-void function does not return a value [-Wreturn-type]
int foo() {}
           ^
1 warning generated.

       File: a.cpp

   1   int foo() {}

➜  foo nm a.o
0000000000000000 T __Z3foov
0000000000000000 t ltmp0
0000000000000010 s ltmp1
➜  foo
```

Unnamed namespace

TEAMT5
杜 浦 數 位 安 全

Is same as static qualifiers.

```
namespace {
    int data;
    int foo();
}
```

Unnamed namespace

TEAMT5
杜 浦 數 位 安 全

Is same as static qualifiers.

```
namespace {
    int data;
    int foo();
}
```

⟺

```
static int data;
static int foo();
```

static member function

TEAMT5
杜 浦 數 位 安 全

Only can be associated to static data.

(Not having this pointer)

Where is the this pointer?

Python:

```
class SCC:
    def __init__(self):
        pass
```

Assembly

```cpp
struct SCC {
    int foo() { return data; }
    int data;
};

struct SCC2 {
    int foo() { return 1; }
    int data;
};

int main() {
    auto s = new SCC();
    SCC2 ss;
    s->foo();
    ss.foo();
}
```

```asm
_ZN3SCC3fooEv:
        push    rbp
        mov     rbp, rsp
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     eax, DWORD PTR [rax]
        pop     rbp
        ret
_ZN4SCC23fooEv:
        push    rbp
        mov     rbp, rsp
        mov     QWORD PTR [rbp-8], rdi
        mov     eax, 1
        pop     rbp
        ret
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     edi, 4
        call    _Znwm
        mov     DWORD PTR [rax], 0
        mov     QWORD PTR [rbp-8], rax
        mov     rax, QWORD PTR [rbp-8]
        mov     rdi, rax
        call    _ZN3SCC3fooEv
        lea     rax, [rbp-12]
        mov     rdi, rax
        call    _ZN4SCC23fooEv
        mov     eax, 0
        leave
        ret
```

https://godbolt.org/z/cn7so8bE1

How different compilers mangle the same functions [edit]

There isn't a standardized scheme by which even trivial C++ identifiers are mangled, and consequently different compilers (or even different versions of the same compiler, or the same compiler on different platforms) mangle public symbols in radically different (and thus totally incompatible) ways. Consider how different C++ compilers mangle the same functions:

Compiler	`void h(int)`	`void h(int, char)`	`void h(void)`
Intel C++ 8.0 for Linux	`_Z1hi`	`_Z1hic`	`_Z1hv`
HP aC++ A.05.55 IA-64			
IAR EWARM C++			
GCC 3.*x* and higher			
Clang 1.*x* and higher[3]			
GCC 2.9.*x*	`h__Fi`	`h__Fic`	`h__Fv`
HP aC++ A.03.45 PA-RISC			
Microsoft Visual C++ v6–v10 (mangling details)	`?h@@YAXH@Z`	`?h@@YAXHD@Z`	`?h@@YAXXZ`
Digital Mars C++			
Borland C++ v3.1	`@h$qi`	`@h$qizc`	`@h$qv`
OpenVMS C++ v6.5 (ARM mode)	`H__XI`	`H__XIC`	`H__XV`
OpenVMS C++ v6.5 (ANSI mode)		`CXX$__7H__FIC26CDH77`	`CXX$__7H__FV2CB06E8`
OpenVMS C++ X7.1 IA-64	`CXX$_Z1HI2DSQ26A`	`CXX$_Z1HIC2NP3LI4`	`CXX$_Z1HV0BCA19V`
SunPro CC	`__1cBh6Fi_v_`	`__1cBh6Fic_v_`	`__1cBh6F_v_`
Tru64 C++ v6.5 (ARM mode)	`h__Xi`	`h__Xic`	`h__Xv`
Tru64 C++ v6.5 (ANSI mode)	`__7h__Fi`	`__7h__Fic`	`__7h__Fv`
Watcom C++ 10.6	`W?h$n(i)v`	`W?h$n(ia)v`	`W?h$n()v`

There isn't a standardized scheme

How different compilers mangle the same functions [edit]

There isn't a standardized scheme by which even trivial C++ identifiers are mangled, and consequently different compilers (or even different versions of the same compiler, or the same compiler on different platforms) mangle public symbols in radically different (and thus totally incompatible) ways. Consider how different C++ compilers mangle the same functions:

How different compilers mangle the same functions [edit]

There isn't a standardized scheme by which even trivial C++ identifiers are mangled, and consequently different compilers (or even different versions of the same compiler, or the same compiler on different platforms) mangle public symbols in radically different (and thus totally incompatible) ways. Consider how different C++ compilers mangle the same functions:

Compiler	`void h(int)`	`void h(int, char)`	`void h(void)`
Intel C++ 8.0 for Linux	`_Z1hi`	`_Z1hic`	`_Z1hv`
HP aC++ A.05.55 IA-64			
IAR EWARM C++			
GCC 3.*x* and higher			
Clang 1.*x* and higher[3]			
GCC 2.9.*x*	`h__Fi`	`h__Fic`	`h__Fv`
HP aC++ A.03.45 PA-RISC			
Microsoft Visual C++ v6–v10 (mangling details)	`?h@@YAXH@Z`	`?h@@YAXHD@Z`	`?h@@YAXXZ`
Digital Mars C++			
Borland C++ v3.1	`@h$qi`	`@h$qizc`	`@h$qv`
OpenVMS C++ v6.5 (ARM mode)	`H__XI`	`H__XIC`	`H__XV`
OpenVMS C++ v6.5 (ANSI mode)		`CXX$__7H__FIC26CDH77`	`CXX$__7H__FV2CB06E8`
OpenVMS C++ X7.1 IA-64	`CXX$_Z1HI2DSQ26A`	`CXX$_Z1HIC2NP3LI4`	`CXX$_Z1HV0BCA19V`
SunPro CC	`__1cBh6Fi_v_`	`__1cBh6Fic_v_`	`__1cBh6F_v_`
Tru64 C++ v6.5 (ARM mode)	`h__Xi`	`h__Xic`	`h__Xv`
Tru64 C++ v6.5 (ANSI mode)	`__7h__Fi`	`__7h__Fic`	`__7h__Fv`
Watcom C++ 10.6	`W?h$n(i)v`	`W?h$n(ia)v`	`W?h$n()v`

So, how did C++ select functions?

Overload resolution

Unfortunately

TEAMT5
杜 浦 數 位 安 全

https://en.cppreference.com/w/cpp/language/overload_resolution

https://eel.is/c++draft/over

…

3 steps roughly

1. 利用 function name lookup 建立 overload set
2. 建立候選人清單 (candidate set)
3. Ranking，找 best overload

3 steps roughly

TEAMT5 杜浦數位安全

1. 利用 function name lookup 建立 overload set
   a. 找出所有 **visible** 的 function declaration
   b. 建立出一個 overload set，過程中它可能會需要 ADL, template argument deduction… (只要 function name 一樣就好，argument 沒符合並沒關係)
2. 建立候選人清單 (candidate set)
3. Ranking，找 best overload

3 steps roughly

1. 利用 function name lookup 建立 overload set
2. 建立候選人清單 (candidate set)
   a. argument 數量正確
   b. 如果呼叫的 argument 數量多於 function parameter 數量，那 function parameter 需要有 ellipsis parameter，就是 C 裡面也有的那個 … eg: `int printz(...);`。
   c. 如果呼叫的 argument 數量小於 function parameter 數量，那 function parameter 需要有 default parameter, eg: `int fn(int = 0);`。
   d. 符合 constraint
   e. argument 的型態要對，armument 可能有 implicit conversion sequence 存在，也就是說如果 argument 轉型可以傳進 function 那也算對
3. Ranking，找 best overload

3 steps roughly

TEAMT5 杜浦數位安全

1. 利用 function name lookup 建立 overload set
2. 建立候選人清單 (candidate set)
3. Ranking，找 best overload
    a. Standard conversion sequences
        i. Exact Match
        ii. Promotion
        iii. Conversion
    b. User-defined conversion sequences
    c. Ellipsis conversion sequences

TEAMT5
杜 浦 數 位 安 全

真的很多很細，等我看完再分享

https://eel.is/c++draft/over.match

https://www.youtube.com/watch?v=iDX2d7poJnI

Name Lookup and Overload Resolution in C++

SFINAE

TEAMT5
杜 浦 數 位 安 全

Substitution **F**ailure **I**s **N**ot **A**n **E**rror

template 參數在替換 explicitly specified 或 deduced type 的時候：

如果失敗的並不會給出 error，而是不將其從 overload set 移除

```
struct Test {
  typedef int foo;
};

template <typename T>
void f(typename T::foo) {}   // Definition #1

template <typename T>
void f(T) {}                 // Definition #2

int main() {
  f<Test>(10);               // Call #1.
  f<int>(10);                // Call #2. 無編譯錯誤(即使沒有 int::foo)
}
```

https://en.wikipedia.org/wiki/Substitution_failure_is_not_an_error

```
struct Test {
  typedef int foo;
};

template <typename T>
void f(typename T::foo) {}   // Definition #1

template <typename T>
void f(T) {}                 // Definition #2

int main() {
  f<Test>(10);               // Call #1.
  f<int>(10);                // Call #2. 無編譯錯誤(即使沒有 int::foo)
}
```

https://en.wikipedia.org/wiki/Substitution_failure_is_not_an_error

```
struct Test {
  typedef int foo;
};

template <typename T>
void f(typename T::foo) {}   // Definition #1

template <typename T>
void f(T) {}                 // Definition #2

int main() {
  f<Test>(10);               // Call #1.
  f<int>(10);                // Call #2. 無編譯錯誤(即使沒有 int::foo)
}
```

https://en.wikipedia.org/wiki/Substitution_failure_is_not_an_error

原本想分享 C++23

TEAMT5 杜浦數位安全

```
struct X {
    template<typename Self>
    void foo(this Self&&, int);
};
struct D : X {};

void ex(X& x, D& d) {
    x.foo(1);            // Self = X&
    move(x).foo(2);  // Self = X
    d.foo(3);            // Self = D&
}
```

Explicit object parameter (P0847R7)

Thank you for your patience.

scc@teamt5.org