Para bailar la λ

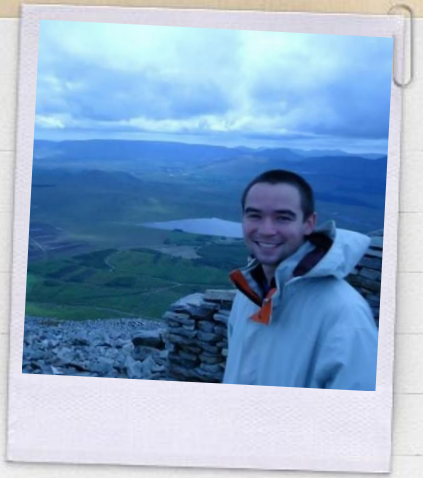
Usage, Détournement et Sublimation des fonctions anonymes au travers des âges

Une pièce en 3 actes de Joël FALCOU

@joel_f

¿ De que vamos a hablar ?

- **Les λ de C++11 à nos jours**
  - Principes de base
  - Evolution
- **Applications inhabituelles**
  - Pseudo-fonction, Overload, IIFE
  - λ - itération, λ-Récursion
  - Tuples et Tagged tuples



Ben Hilburn
@bhilburn

Follow

Leonardo Da Vinci famously wrote in reverse to prevent people from being able to read his writing and steal ideas.

I do the same thing with my code by using C++ lambdas for everything.

4:12 PM - 22 Feb 2019

Les λ-fonctions

Leurs vies, leurs oeuvres

C++98/03: Objet fonctions

- ✗ Principes:
  - ✗ Utilisation d'une structure fournissant une surcharge de l'operator()


Bonnes performances


Localité faible

```cpp
// Object fonction
struct print
{
  void operator()(int i) const
  {
    std::cout << i << "\n";
  }
};

std::vector<int> x;
// Remplissage de x
// …

std::for_each(x.begin(), x.end()
              , print()
              );
```

C++11: Fonctions anonymes

- Principes:
  - Création d'un fonction "locale"
  - Notion de capture

- Bonnes performances
- Bonne localité
- Limitation sur l'API

```
std::vector<int> x =
                  { 1,2,3
                  , 4,5,9
                  , 69,1337,42
                  };

int n = 3;

// lambda-fonction
std::for_each(x.begin(), x.end()
              , [n](int i)
                {
                  std::cout << n*i
                            << "\n";
                }
              );
```

C++14: Utilisabilité

- ✗ Interface:
  - ✗ Paramètres ‘auto’
  - ✗ Paramètres variadiques
  - ✗ Type de retour facultatif
- ✗ Interactions:
  - ✗ Noexcept autorisé
  - ✗ Convertible en pointeur de fonction si stateless
  - ✗ Renommage des captures

```
int x = 4;

auto y = [&r = x, x = x+1](auto v)
        {
          r += v;
          return x * x;
        };

// Que vaut x et n ?
auto n = y(2);
```

C++17: Inexorable constexpr

- ✗ Utilisabilité:
  - ✗ Utilisation des λ-fonctions dans des contextes `constexpr`
  - ✗ Marquage optionnel, toutes les λ-fonctions sont constexpr par défaut
- ✗ Correction:
  - ✗ Capture de `*this` correcte
  - ✗ Pas d'erreur en retournant une λ issue d'un temporaire

```
auto fact = [](int n)
{
  int r = 1;
  for(int i=1;i<=n;i++) r *= i;
  return r;
};


std::array<char, fact(5)> x;
```

C++20: λ-Complétion

- ✗ Retour aux source:
  - ✗ Notion λ-fonctions explicitement templates
  - ✗ Augmente l'utilité de l'appel direct de operator().
- ✗ Support de consteval:
  - ✗ Tout comme les λ-fonctions peuvent être `constexpr`, elles peuvent être marquées `consteval`

```
auto bulk = []<class T>(T v)
{
  std::array<T,16> that{};
  that.fill(v);
  return that;
};

auto x = bulk(4.f);
```

Applications

“La, ça passe”

- Famous last words

Move-in lambdas

- ✗ Capturer un objet move-only:
  - ✗ Capture par référence peu sûre
  - ✗ Capture par valeur impossible
- ✗ Solution [C++14 & up]:
  - ✗ Renommage de la capture
  - ✗ Passage d'un initialiseur explicite

```
// u est move-only
auto u = make_unique<int>(10);

// copie u dans la lambda
auto ko = [u]
{ cout << *u << "\n"; };

// reference u dans la lambda
auto okish = [&u]
{ cout << *u << "\n"; };

// move u dans la lambda
auto ok = [ u{std::move(u)} ]
{ cout << *u << "\n"; };
```

Les IIFEs

- ✗ Immediately-Invoked Function Expression
  - ✗ Initialisation de constante complexe
  - ✗ Initialisation complexe pré-construction
  - ✗ Invariant et sécurité garantit

```
// Facile
const int v = i * 10 + 5;

// Moins facile et pas const
int v = 0;

if (c) v = other_c ? f(i) : 0;
else   v = i * 2;

// IIFE et const
const int v = [&]
{
  if (c) return other_c ? f(i) : 0;
  else   return i * 2;
}();
```

Générateur fonctionnel

- ✗ Générer une fonction à la volée:
  - ✗ Définie par des paramètres externes
  - ✗ A durée de vie courte
- ✗ Solution :
  - ✗ Retourner une lambda depuis une lambda

```
// Courtesy of Markus Werle
#include <iterator>
#include <vector>

auto first = [](auto i)
{
  auto getN = [=](auto const& s)
  {
    auto b = std::begin(s);
    decltype(s) result{b,std::next(b,i)};
    return result;
  };
  return getN;
};

std::vector<int> v{1,2,4,8,16,32};
auto sub = first(3)(v);
```

MAXIMUM OVERLOAD

- ✗ Aggréger des lambdas
  - ✗ Créationd ‘un overload set
  - ✗ Use case: visiteur déclaré online
- ✗ Solution :
  - ✗ Hériter de lambdas
  - ✗ Propagez leur `operator()`
  - ✗ Un petit coup de CTAD et on est parti.

```
// Courtesy of David Brcz
template<class... Ts>
struct overload : Ts...
{
    overload(Ts... t) : Ts(t)... {}
    using Ts::operator()...;
};

template<class... Ts>
overload(Ts...) -> overload<Ts...>;

auto f = overload( [](int)
                   { return "entier\n"; }
                 , [](auto)
                   { return "??\n"; }
                 , [](float)
                   { return "reel\n"; }
                 );
```

Applications II

“Your scientists were so preoccupied with whether or not they could, they didn't stop to think if they should.”

- Ian Malcolm

Lambda récursive

- ✗ Impossible pour une lambda de se capturer elle même
- ✗ Solution :
  - ✗ YAIL !
  - ✗ Une lambda appelant indirectement une autre lambda
  - ✗ Récursivement…

```
auto f_impl = [](auto self, auto n)
            -> decltype(n)
{
  if (n > 2)
    return self(self,n-1)
         + self(self,n-2);
  else
    return 1;
};

auto fibo = [](auto n)
{
  return f_impl(f_impl, n);
};

auto n = fibo(9);
```

Type uniquement nommé

- ✗ Créer un type unique
  - ✗ A partir d'une chaîne de caractères
  - ✗ A la compilation
- ✗ Solution :
  - ✗ Lambda constexpr
  - ✗ Combiné à index_sequence
  - ✗ Macro d'emballage

```
template <char ...> class S {};

template <class F, std::size_t ... I>
auto S_(F f, index_sequence<I...>)
{
  return S<f(I)...>{};
}

#define str(x)                          \
S_( [](auto i){return x[i];}            \
  , make_index_sequence<sizeof(x)>{} \
  );                                    \

auto s = str("foo");
```

La Chambre des Horreurs

- Itérations statiques
  - Liste de types
  - Liste de constantes
- Type-Value Map :
  - Table associant un objet à un objet via leur type
  - Extension des tuples

Conclusion

Merci !!

A toutes les personnes m'ayant signalé les perles présentées ici

Serge Guelton
github.com/serge-sans-paille

@ubsanitizer
https://github.com/ubsan

@MarkusWerle

@bjorn_fahller

@danielelliott3d

@Davidbrcz