GERMINOLEGRAND

Zeste de savoir

Moderniser ses algorithmes en C++

mardi 11 mars 2025

Table des matières

Introduction

Vous en avez marre d'écrire du code long et répétitif ? Alors il est probable que vous ayez adopté l'usage de la librairie `std::ranges` en C++20 ! Si vous êtes encore coincé · e en C++11, il y a fort à parier que vous ayez écrit vos propres substituts pour ne pas avoir à utiliser les itérateurs.

Nous allons voir ici comment moderniser son code, et comment ne pas attendre avec un exemple de démarche pour substituer une fonction standard qui n'est pas encore disponible dans un standard plus ancien.

1. L'algorithme std::ranges::contains

Vous le savez sans doute (par exemple si vous avez suivi le cours de C++ sur Zeste de Savoir), la bibliothèque standard du C++ regorge d'algorithmes génériques très utiles qui évitent de réinventer la roue, de faire du code suboptimal et bugué, tout en rendant explicite les intentions du développeur · se. Le code est alors plus "expressif" et concis.

```
1 std::vector ages{0, 2, 43, 5, 64, 34, 24, 46};
2
3 if (std::ranges::contains(ages, 34)) {
4     std::print("Age of wonder");
```

```
}
```

Comme on peut le deviner, l'algorithme `std::ranges::contains` permet de savoir si un ensemble contient une valeur.

Elle peut se combiner avec plusieurs fonctionnalités récentes, telles que les ranges, les pipes, lambdas et autres projections.

```cpp
#include <algorithm>
#include <vector>
#include <print>
#include <ranges>
#include <string>

int main() {
    struct Person
    {
        std::string name;
        uint8_t age;
    };
    std::vector<Person> people{
        {"Jules", 0},       {"Amine", 2},
        {"Nada", 43},       {"Khaoula", 5},
        {"Roseline", 64},   {"Gabriela", 34},
        {"Aymeric", 24},    {"Alex", 46},
    };

    auto a_people = people | std::views::filter([](Person const&
        p){
        return std::ranges::contains(p.name, 'A', ::toupper);
    });

    if (std::ranges::contains(a_people, 34, &Person::age)) {
        std::print("Age of wonder");
    }
}
```

Ici on cherche à savoir si parmi la population dont le prénom contient un `a` majuscule ou minuscule, il existe au moins une personne de 34 ans.

i Notez qu'on peut, dans ce cas comme beaucoup d'autres, imaginer d'autres assemblages d'algorithmes

```cpp
if (std::ranges::any_of(people, [](auto& p){
        return std::ranges::contains(p.name, 'A', ::toupper)
            && p.age == 34;
    })) {
    std::print("Age of wonder");
}
```

2. L'évolution de l'algorithme au fil des standards

On peut faire un petit retour en arrière jusqu'en C++98 pour se rendre compte du chemin parcouru.

```cpp
// C++98 (celui-là je vous l'ai mis pour se rendre compte d'où on
    vient, mais ne faites pas ça par pitié)
static const int default_ages[]{0, 2, 43, 5, 64, 34, 24, 46};
std::vector<int> ages(default_ages, std::next(default_ages,
    sizeof(default_ages) / sizeof(*default_ages)));
// C++11
std::vector<int> ages{0, 2, 43, 5, 64, 34, 24, 46};
// C++17
std::vector ages{0, 2, 43, 5, 64, 34, 24, 46};

// C++98
if (std::find(ages.begin(), ages.end(), 34) != ages.end()) {
    std::cout << "Age of darkness";
}

// C++20
if (std::ranges::find(ages, 34) != ages.end()) {
    std::cout << "Age of awakeness";
}

// C++23
if (std::ranges::contains(ages, 34)) {
    std::cout << "Age of wonder";
}
```

2.1. Depuis le C89

Je ne vous fais pas la version C89… bon allez si, pour se faire plaisir 🍍

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {
    static const int default_ages[] = {0, 2, 43, 5, 64, 34, 24,
        46};
    unsigned int ages_size = sizeof(default_ages);
    int* ages = malloc(ages_size * sizeof(int));
    int* a;
    memcpy(ages, default_ages, ages_size);
    for (a = ages; a != ages + ages_size; ++a) {
        if (*a == 34) {
            printf("Age of stone");
        }
    }
    free(ages);
    return 0;
}
```

Bien que ce code compile parfaitement en C++ (moyennant quelques warnings…), de par ce qu'on appelle la compatibilité ascendante qui permet de ne pas avoir à réécrire les centaines de millions de lignes de code écrites par nos prédécesseur · ses, ce qui est bien utile, n'écrivez **JAMAIS** un tel code. Si c'est possible modernisez-le, éventuellement en utilisant des outils tels que Clang-Tidy, il y a fort à parier que des bugs et des failles de sécurité disparaîtront.

Par pitié, si vous faites du C , utilisez le standard le plus récent possible, le C23 par exemple !

2.2. Depuis l'assembleur

Non, là vraiment, ne comptez pas sur moi, je n'écris jamais d'assembleur à la main. Ce qui est utile c'est plutôt de savoir le déchiffrer.

Par exemple, le code en C++23 donne ceci :

```asm
.LC0:
        .string "Age of wonder"
main:
        sub     rsp, 8
        mov     edx, 13
```

```
6         mov     esi, OFFSET FLAT:.LC0
7         mov     edi, OFFSET FLAT:std::cout
8         call    std::basic_ostream<char, std::char_traits<char> >&
              std::__ostream_insert<char, std::char_traits<char>
              >(std::basic_ostream<char, std::char_traits<char> >&,
              char const*, long)
9         xor     eax, eax
10        add     rsp, 8
11        ret
```

? Hein ? Mais c'est super court ! Comment ça se fait ?

Eh bien en fait, le compilateur sait tellement bien optimiser du C++ qui utilise des fonctionnalités standards qu'il a pu déduire que le code ne faisait rien d'autre qu'afficher systématiquement `"Age of wonder"`. Donc il se contente de l'afficher directement sans aucun calcul.

En comparaison, je vous ai mis ce que donne le code en C89 compilé avec le même compilateur très récent (GCC) : il n'est tout simplement pas en mesure de l'optimiser.

Contenu masqué n°1

C'est encore une bonne raison supplémentaire d'utiliser des langages et des standards plus récents et des fonctionnalités standards 😎 .

3. Substituer std::ranges::contains en C++20

Si comme moi vous avez à travailler en C++20, voici comment écrire un code de substitution pour une fonction disponible en C++23.

3.1. Ne pas réinventer la roue

La première étape est de regarder sur le wiki cppreference.com à la page de l'algorithme qui nous intéresse : `std::ranges::contains` .

Elle contient comme souvent une *Possible implementation* :

```
1 struct __contains_fn
2 {
3     template<std::input_iterator I, std::sentinel_for<I> S,
4              class Proj = std::identity,
5              class T = std::projected_value_t<I, Proj>>
```

```
6       requires std::indirect_binary_predicate<ranges::equal_to,
            std::projected<I, Proj>,
7                                               const T*>
8       constexpr bool operator()(I first, S last, const T& value,
            Proj proj = {}) const
9       {
10          return ranges::find(std::move(first), last, value, proj)
                != last;
11      }
12
13      template<ranges::input_range R,
14               class Proj = std::identity,
15               class T =
                     std::projected_value_t<ranges::iterator_t<R>,
                     Proj>>
16      requires std::indirect_binary_predicate<ranges::equal_to,
17                                              std::projected<ranges:
                                                    :iterator_t<R>,
                                                    Proj>,
18                                              const T*>
19      constexpr bool operator()(R&& r, const T& value, Proj proj =
            {}) const
20      {
21          return (*this)(ranges::begin(r), ranges::end(r),
                std::move(value), proj);
22      }
23  };
24
25  inline constexpr __contains_fn contains {};
```

i

Parfois, comme c'est le cas ici, le code peut rebuter de prime abord. Vous n'écririez jamais vous-même un code aussi complexe, mais la bibliothèque standard, elle, doit prévoir tous les cas particuliers, et s'entremêle avec beaucoup de fonctionnalités. Il n'est pas indispensable de tout comprendre pour pouvoir adapter.

Évidemment c'est une implémentation qui fonctionne… en C++26, le standard le plus récent. Mais en l'adaptant un peu, on peut ici sans problème l'amener en C++20 !

3.2. Mettre dans son propre namespace et corriger le nom

On n'a pas le droit de faire la même chose quand on n'est pas nous même en train d'implémenter la bibliothèque standard. Par exemple, on n'a pas le droit d'écrire dans le `namespace std` (hormis quelques exceptions bien spécifiques) ni utiliser les noms qui commencent par deux underscores. Il faut donc corriger tout cela, et rajouter des `std::` là où c'est nécessaire.

```cpp
namespace utils
{
struct contains_fn
{
    template<std::input_iterator I, std::sentinel_for<I> S,
             class Proj = std::identity,
             class T = std::projected_value_t<I, Proj>>
    requires std::indirect_binary_predicate<std::ranges::equal_to,
        std::projected<I, Proj>,
                                            const T*>
    constexpr bool operator()(I first, S last, const T& value,
        Proj proj = {}) const
    {
        return std::ranges::find(std::move(first), last, value,
            proj) != last;
    }

    template<std::ranges::input_range R,
             class Proj = std::identity,
             class T = std::projected_value_t<std::ranges::iterato
                 r_t<R>,
                 Proj>>
    requires std::indirect_binary_predicate<std::ranges::equal_to,
                                            std::projected<std::ra
                                                nges::iterator_t<R
                                                >,
                                                Proj>,
                                            const T*>
    constexpr bool operator()(R&& r, const T& value, Proj proj =
        {}) const
    {
        return (*this)(std::ranges::begin(r), std::ranges::end(r),
            std::move(value), proj);
    }
};

inline constexpr contains_fn contains {};

}
```

3.3. Remplacer ce qui n'est pas encore en C++20

Tout ça ne compile malheureusement pas encore, à cause de `std::projected_value_t` qui arrive en C++26, il faut donc appliquer la même méthode, consulter la page du wiki, et le tour est joué !

```cpp
template< std::indirectly_readable I,
          std::indirectly_regular_unary_invocable<I> Proj >
using projected_value_t =
    std::remove_cvref_t<std::invoke_result_t<Proj&,
        std::iter_value_t<I>&>>;
```

Ce qui nous donne :

```cpp
namespace utils
{
template< std::indirectly_readable I,
          std::indirectly_regular_unary_invocable<I> Proj >
using projected_value_t =
    std::remove_cvref_t<std::invoke_result_t<Proj&,
        std::iter_value_t<I>&>>;

struct contains_fn
{
    template<std::input_iterator I, std::sentinel_for<I> S,
             class Proj = std::identity,
             class T = projected_value_t<I, Proj>>
    requires std::indirect_binary_predicate<std::ranges::equal_to,
        std::projected<I, Proj>,
                                            const T*>
    constexpr bool operator()(I first, S last, const T& value,
        Proj proj = {}) const
    {
        return std::ranges::find(std::move(first), last, value,
            proj) != last;
    }

    template<std::ranges::input_range R,
             class Proj = std::identity,
             class T =
                 projected_value_t<std::ranges::iterator_t<R>,
                 Proj>>
    requires std::indirect_binary_predicate<std::ranges::equal_to,
                                            std::projected<std::ra
                                                nges::iterator_t<R
                                                >,
                                                Proj>,
                                            const T*>
    constexpr bool operator()(R&& r, const T& value, Proj proj =
        {}) const
    {
        return (*this)(std::ranges::begin(r), std::ranges::end(r),
            std::move(value), proj);
```

```
    }
};

inline constexpr contains_fn contains {};

}
```

3.4. Prévoir l'avenir

Lorsqu'on écrit un substitut, il est préférable d'anticiper qu'on va moderniser le code un jour et qu'il faudra supprimer ce substitut et remplacer ses usages.

La bonne pratique est d'utiliser l'attribut `[[deprecated]]`, de préférence avec un message indiquant quoi faire `[[deprecated("C++23: use std::ranges::contains instead")]]`.

Évidemment, il ne s'agit pas d'avoir des warnings tant qu'on est en C++20… on peut donc utiliser soit la macro `__cplusplus` pour vérifier le standard utilisé :

```
#if __cplusplus > 202302L
    [[deprecated("C++23: use std::ranges::contains instead")]]
#endif
```

Ou pour être plus précis, on peut consulter le tableau des macros de test de fonctionnalité :

```
#ifdef __cpp_lib_ranges_contains
    [[deprecated("Use std::ranges::contains instead")]]
#endif
```

Ce qui nous donne :

```
#include <iostream>
#include <vector>
#include <algorithm>

namespace utils
{
template< std::indirectly_readable I,
          std::indirectly_regular_unary_invocable<I> Proj >
using projected_value_t =
    std::remove_cvref_t<std::invoke_result_t<Proj&,
        std::iter_value_t<I>&>>;

struct contains_fn
{
```

```
14      template<std::input_iterator I, std::sentinel_for<I> S,
15               class Proj = std::identity,
16               class T = projected_value_t<I, Proj>>
17      requires std::indirect_binary_predicate<std::ranges::equal_to,
            std::projected<I, Proj>,
18                                              const T*>
19      constexpr bool operator()(I first, S last, const T& value,
            Proj proj = {}) const
20      {
21          return std::ranges::find(std::move(first), last, value,
                proj) != last;
22      }
23
24      template<std::ranges::input_range R,
25               class Proj = std::identity,
26               class T =
                     projected_value_t<std::ranges::iterator_t<R>,
                     Proj>>
27      requires std::indirect_binary_predicate<std::ranges::equal_to,
28                                              std::projected<std::ra
                                                    nges::iterator_t<R
                                                    >,
                                                    Proj>,
29                                              const T*>
30      constexpr bool operator()(R&& r, const T& value, Proj proj =
            {}) const
31      {
32          return (*this)(std::ranges::begin(r), std::ranges::end(r),
                std::move(value), proj);
33      }
34  };
35
36  #ifdef __cpp_lib_ranges_contains
37      [[deprecated("Use std::ranges::contains instead")]]
38  #endif
39  inline constexpr contains_fn contains {};
40
41  }
42
43  int main()
44  {
45      std::vector ages{0, 2, 43, 5, 64, 34, 24, 46};
46
47      // C++20
48      if (utils::contains(ages, 34)) {
49          std::cout << "Age of wonder";
50      }
51  }
```

— https://godbolt.org/z/Kfc95vr4x

Lorsque C++23 sera activé, on aura un joli warning :

```
<source>: In function 'int main()':
<source>:47:16: warning: 'utils::contains' is deprecated: Use
    std::ranges::contains instead [-Wdeprecated-declarations]
   47 |     if (utils::contains(ages, 34)) {
      |                ^~~~~~~~
<source>:39:30: note: declared here
   39 | inline constexpr contains_fn contains {};
      |                              ^~~~~~~~
```

4. Je suis coincé en C++11/14/17 ! Que faire ?

Tout n'est pas perdu !

Mais il faudra faire preuve d'un peu plus de réflexion et le coder soi-même.

```cpp
#include <algorithm>

namespace utils
{
    template<class R, class V>
#ifdef __cpp_lib_ranges_contains
    [[deprecated("Use std::ranges::contains instead")]]
#endif
    constexpr bool contains(R&& range, V&& value)
    {
        return std::find(range.begin(), range.end(),
            std::forward<V>(value)) != range.end();
    }
}
```

Cette version peut paraître plus simple (et elle l'est, pour ce qui est de la lire et de la comprendre), mais elle est moins optimisable par le compilateur et moins intégrée que ne l'est la version C++23, contient moins de garde-fous sur les types utilisables, et bien sûr ne contient pas la fonctionnalité de projection.

Un code simple que l'on comprend est parfois préférable à un code complexe bien que plus performant, surtout si on doit le maintenir soi-même.

Conclusion

Le pas-à-pas que nous avons vu ici peut se décliner sur d'autres fonctionnalités qui continueront d'arriver au fil des nouvelles versions du langage. Elles arrivent généralement plus vite que l'on n'a le temps de les apprendre et de les maîtriser, je ne peux que vous encourager à faire preuve de curiosité à ce sujet 🍊 .

J'espère que vous aurez appris des choses, que cela vous aura aidé, ou que cela vous aura tout simplement permis de découvrir ou redécouvrir certains aspects du C++ !

Contenu masqué

Contenu masqué n°1

```
.LC0:
        .string "Age of stone"
main:
        push    r12
        mov     edi, 128
        push    rbp
        push    rbx
        call    malloc
        movdqa  xmm0, XMMWORD PTR default_ages.0[rip]
        mov     r12, rax
        lea     rbp, [rax+128]
        mov     rbx, rax
        movups  XMMWORD PTR [rax], xmm0
        movdqa  xmm0, XMMWORD PTR default_ages.0[rip+16]
        movups  XMMWORD PTR [rax+16], xmm0
        jmp     .L3
.L2:
        add     rbx, 4
        cmp     rbx, rbp
        je      .L7
.L3:
        cmp     DWORD PTR [rbx], 34
        jne     .L2
        mov     edi, OFFSET FLAT:.LC0
        xor     eax, eax
        add     rbx, 4
        call    printf
        cmp     rbx, rbp
        jne     .L3
.L7:
        mov     rdi, r12
        call    free
```

Contenu masqué

```
33          pop     rbx
34          xor     eax, eax
35          pop     rbp
36          pop     r12
37          ret
38  default_ages.0:
39          .long   0
40          .long   2
41          .long   43
42          .long   5
43          .long   64
44          .long   34
45          .long   24
46          .long   46
```

Retourner au texte.