ArtWorks

Advanced C++

Nouveautés de la librairie standard

Retraits de la librairie

C++ 17

- Eléments de la librairie standard qui sont retirés
  - auto_ptr,
  - unary_function / binary_function,
  - ptr_fun, mem_fun, mem_fun_ref,
  - bind1st, bind2nd,
  - random_shuffle

2

Ranges

C++ 20

```
namespace rng = std::ranges;

std::array<int, 6> myArray {6, 1, 78, -34, 112, 99};
rng::sort (myArray);

auto itor {rng::find_if (
  myArray, [](int n){return n < 0;})};
```

plus besoin d'itérateurs

4

Ranges

C++ 20

```
vector<int> numbers { 1, 2, 3 ,4, 5 };
bool even{ [](int i) { return 0 == i % 2; }};
int square{ [](int i) { return i * i; }};
```

```
vector<int> evenNumbers;
copy_if(cbegin(numbers), cend(numbers),
  back_inserter(evenNumbers), even);

vector<int> results;
std::transform(begin(evenNumbers), end(evenNumbers),
  back_inserter(results), square);
for (int n : results) cout << n << '\n';
```

C++ 98

```
for (int i : numbers | filter(even) | transform(square))
  cout << i << '\n';
```

C++ 20

5

Nouvelle fonctionnalité : emplacement

- insert insère copie ou déplace (arg&&) un objet dans une collection
- emplace crée et insère un nouvel objet
  - les données de construction sont transmises

```
struct A {
  int n;
  A(int n) : n{n}{ cout << "creation\n"; }
  A(const A&){ cout << "copie\n"; }
  A(const A&&){ cout << "move\n"; }
  ~A(){ cout << "destruction\n"; }
};
```

```
{
  vector<A> v;
  v.emplace_back(2);
}
cout << endl;
{
  vector<A> v;
  v.push_back(2);
}
```

```
creation
destruction

creation
move
destruction
destruction
```


Les tableaux à taille fixe

- un array dispose des mêmes méthodes que le vector
  - alternative plus légère

```
array<int, 5> tab { 1, 2, 3, 4, 5 };
for (int nombre : tab)
  cout << nombre << "\n";

cout << "Taille : " << tab.size() << "\n";

tab[0] = 10;
tab.at(1) = 40;
cout << tab[1] << "\n";
cout << tab.at(0) << "\n";
cout << tab.front() << "\n";
cout << tab.back() << "\n";

if (!tab.empty())
  cout << "Le tableau n'est pas vide\n";

tab.fill(5);
```

vérification des limites

réinitialisation de toutes les valeurs

Le conteneur tuple

- structure de dimension fixe d'objets de types différents
  - utile pour un retour multiple depuis une fonction

```
using Personne = tuple<int, double, string>;
Personne Patrick {22, 185.4, "Patrick"};
Personne Marie {23, 185.4, "Marie"};

get<0>(Patrick) = 35;
cout << get<1>(Marie) << "\n";

int n; double d; string s;
std::tie(n, d, s) = t;

auto [a, b, c] {t};


cout << tuple_size <Personne>::value << "\n";

if (Patrick == Marie)
  cout << "Les tuples sont identiques" << "\n";
```

à éviter

unpacking

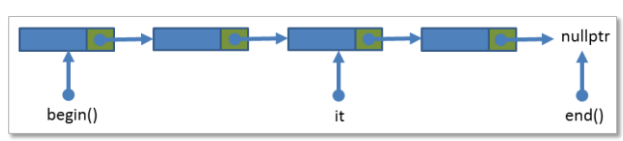
C++17

8

Le conteneur forward_list

- version plus rapide et légère de list
  - pas de reverse itération
  - pas de taille stockée
    - size() est non disponible

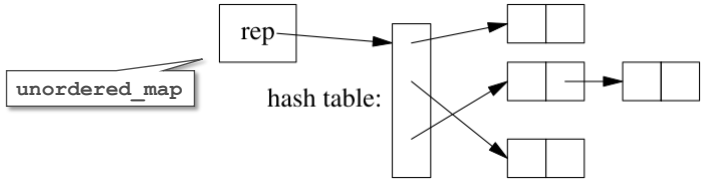
```
forward_list<int> liste;
liste.push_front(2);
liste.push_front(4);
liste.push_front(9);
liste.emplace_after (liste.before_begin(), 11 );
cout << distance (liste.begin(), liste.end()) << endl;
```



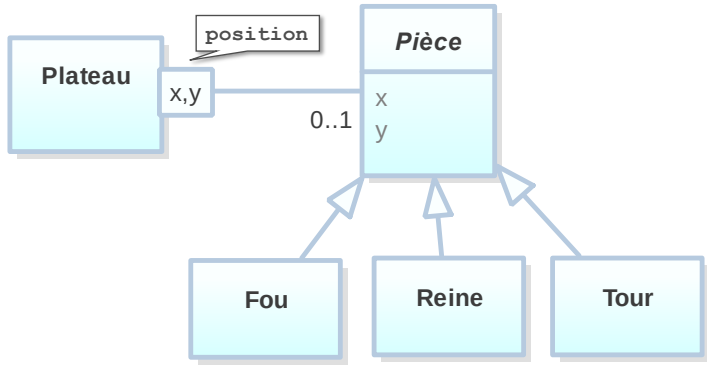
9

Conteneurs associatifs non-ordonnés

- recherche à partir d'une formule de hash-code au lieu d'une relation d'ordre (operator<)
  - unordered_set
    - collection de clés uniques, hachées par les clés
  - unordered_map
    - collection de paires clé-valeur, hachées par les clés, les clés sont uniques
  - unordered_multimap
    - collection de clés, hachées par les clés
  - unordered_multiset
    - collection de paires clé-valeur, hachées par les clés



usage de la classe unordered_map



La classe std::hash<T>

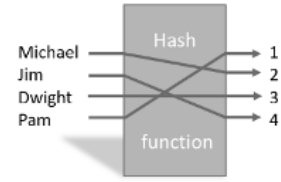
```
struct Position {
 int x, y;
 bool operator== (const Position& autre) const {
  return x == autre.x && y == autre.y;
 }
};

template<>
struct hash<Position>{
 size_t operator()(const Position& p) const {
   return 10 * p.x + p.y;
 }
};
```

opérateur d'égalité obligatoire

pas de relation d'ordre "métier" dans la classe Position

```
unordered_map <Position, Pion> jeu;
jeu[Position{ 2, 3 }] = "FouBlanc";
jeu[Position{ 4, 1 }] = "RoiNoir";
```



les fonctions suivantes sont disponibles:
- hash<int>
- hash<string>
- … combinables avec le XOR bit à bit ^

La classe reference_wrapper<T>

- encapsule une référence dans un objet copiable et assignable.
- permet a des conteneurs de stocker des références
- std::ref et std::cref permettent de les générer

```
#include <functional>

int a{};
vector <reference_wrapper<int>> v{ a };
a = 99;
cout << v[0] << endl;
v[0].get() = 12;
cout << a << endl;

reference_wrapper<const int> w{ a };
w.get() = 12;
```

ne compile pas, constness

std::string_view

C++ 17

- pointeur et longueur sur un buffer interne de string
  - son usage évite des créations temporaires d'instances de string

```
class Person {
  string name;
public:
  Person(string_view name) : name{ name } {}
};
```

```
string name;
Person p{ name };

Person p{ "Tom" };
```

promotion de const char* vers un string_view

std::any

C++ 17

- embarque une instance d'un type quelconque
  - peut être vide et changer dynamiquement de type
  - les objets volumineux sont placés sur le heap

```
try {
  any a;
  if (!a.has_value())
    cout << "empty \n";

  a = 123;
  cout << any_cast<int>(a) << "\n";

  a = 23.78;
  cout << any_cast<double>(a) << "\n";

  a = make_any<string>("hello");
  cout << any_cast<const string&>(a) << "\n";
}
catch (const bad_any_cast& e) {
  cout << e.what() << "\n";
}
```

paramètres de construction

```
empty
123
23.78
hello
```

std::variant

C++ 17

- union générique
  - peut être vide et changer dynamiquement de type
  - allocation sur la pile

types de l'union

```
try {
  variant <int, double> v;
  v = 12;
  cout << get<int>(v) << "\n";

  v = 23.78;
  cout << get<double>(v) << "\n";
  cout << get<int>(v) << "\n";
}
catch (const bad_variant_access& e) {
  cout << e.what() << "\n";
}
```

```
12
23.78
bad variant access
```


16

std::optional<T>

C++ 17

- permet de retourner un résultat optionnel ou d'implémenter une mise en cache de type "lazy"

retour optionnel

```
optional<string> ostr {GetUserResponse()};
if (ostr)
  ProcessResponse(*ostr);
else
  Report("please enter a valid value");
```

- méthodes importantes:
  - emplace
  - reset
    - idem que operator =
  - has_value
    - idem que operator bool
  - value
    - idem que operator *

17

Gestion du temps

- Permet de mesurer efficacement une durée d'exécution

```
using namespace std::chrono;

auto t0 { high_resolution_clock::now()};

// …  do_work();

auto t1 {high_resolution_clock::now()};

cout << duration_cast<milliseconds>(t1-t0).count()
 << "msec\n";
```

retourne un time_point (ns)

conversion en ms

retourne une duration

Littéraux dédiés à la gestion du temps

```
using namespace std::chrono_literals;

auto duration{
  10ns +
  2us +
  3ms +
  100s +
  10min +
  3h};

cout << duration.count() << endl;
```

nanosecondes

microsecondes

microsecondes

Nombres aléatoires

- Couplage d'un générateur à une distribution
  - un moteur produit une séquence aléatoire d'entiers non signés
    - random_device (hardware), default_random_engine,
  - une distribution définit un type, un intervalle et une loi de distribution
    - uniform_int_distribution, uniform_real_distribution, bernoulli_distribution, geometric_distribution, poisson_distribution, binomial_distribution, uniform_real_distribution, exponential_distribution, normal_distribution, gamma_distribution
    - tient compte des valeurs déjà obtenues mais reset() est possible

```
default_random_engine engine;
cout << engine() << "\n";
uniform_int_distribution distribution{0, 99};
cout << distribution(engine) << endl;
```

std::bernoulli_distribution

- la plus simple des distributions

$$P(b|p) = \begin{cases} p & , \ b = \textbf{true} \\ 1-p & , \ b = \textbf{false} \end{cases}$$

```
default_random_engine generator;
bool decision{ bernoulli_distribution{ 0.5 }(generator) };
cout << (decision ? "Go" : "Baby please don't go") << endl;
```

p

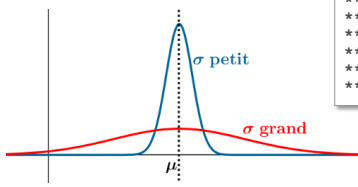
Distribution gaussienne

```
default_random_engine e{ static_cast<unsigned>(time(0)) };
double moyenne {5.3};
double ecart_type {1.5};
normal_distribution n {moyenne, ecart_type};

std::multiset<unsigned> values;
for (size_t i{}; i != 200; ++i) {
  values.insert(static_cast<unsigned>(lround(n(e))));
}

for (unsigned i{}; i < 10; ++i)
  cout << string (values.count(i), '*') << "\n";
```

random seed




```
*****
***************
*****************************************
*******************************************************
*****************************************
****************************
**********
***
```


Expressions régulières avec Regex

- but: vérifier la conformité d'une chaine par rapport à un motif
- quelques motifs:
  - "[[:digit:]]" 1 chiffre
  - "[[:digit:]]+" au moins 1 chiffre
  - "-?[[:digit:]]+" idem, symbole – optionnel
  - "(\\+ | -)?[[:digit:]]+" idem, symbole + ou – optionnel

```
regex motif {"(\\+|-)?[[:digit:]]+"};

string s;
cin >> s;
cout << (regex_match(s,motif) ? "ok" : "not ok") << endl;
```

- http://www.regular-expressions.info/examples.html
- https://regex101.com/