Handout – Itérateurs et quelques usages

Itérateurs

Un itérateur pointe sur un élément d'une collection. Un itérateur supporte :

- L'opérateur d'incrémentation (++, et aussi `--`, `+=`, `-=`) pour le déplacer
- L'opérateur de déréférencement (`*`) pour obtenir la valeur d'un élément

Exemple

```
for (vector<int>::iterator it = a.begin();
     it != a.end(); ++it) {
    print(*it);
}
```

est équivalent à

```
for (int i = 0; i < a.size(); ++i) {
    print(a[i]);
}
```

- `a.begin()` est un pointeur sur `a[0]`
- `a.end()` est l'itérateur "past-the-end"

Attention:

- `it` (position) n'est pas la même chose que `*it` (valeur)
- `*it.x` n'est pas interprété comme `*(it.x)`, mais comme `(*it).x`. `it->x` est équivalent à `(*it).x`.

Ranges

Un *Range* est défini par un intervalle semi ouvert. Le range $[\ell, r)$ contient les éléments $\ell, \ell+1, \ldots, r-2, r-1$. Toute la collection est alors `v.begin(), v.end()`.

Utilisation

Trier

```
vector<int> a{1, 4, 5, 5, 2, 5};
sort(a.begin(), a.end());
// a: 1, 2, 4, 5, 5, 5
```

Le côté gauche est toujours inclusif et le côté exclusif :

```
vector<int> a{1, 4, 2, 5, 2, 5};
sort(a.begin()+1, a.end()-2);
// a: 1, 2, 4, 5, 2, 5
```

Changer l'ordre

```
vector<int> a{1, 2, 3, 4, 5};
reverse(a.begin(), a.end()); // a est 5, 4, 3, 2, 1
```

```
vector<int> a{1, 2, 3, 4, 5};
rotate(a.begin(), a.begin() + 1, a.end());
// a est 2, 3, 4, 5, 1
rotate(a.begin(), a.begin() + 1, a.end());
// a est 3, 4, 5, 1, 2
rotate(a.begin(), a.begin() + 3, a.end());
// a est 1, 2, 3, 4, 5
```

Minimum

```
vector<int> a{1, 4, 5, 5, 2, 5};
vector<int>::iterator it =
    min_element(a.begin(), a.end());
if (it == a.end()) {
    print("La liste est vide");
} else {
    print("Minimum:", *it);
}
```

Compter

```
vector<int> a{1, 4, 5, 5, 2, 5};
print("il y a ", count(a.begin(), a.end(), 5),
      " fois un 5 dans a");
// donne "il y a 3 fois un 5 dans a\n"
```

Trouver un élément

```
vector<int> a{1, 4, 5, 5, 2, 5};
vector<int>::iterator it = find(a.begin(), a.end(), 5);
if (it == a.end()) {
    print("il n'y a pas de 5.");
} else {
    print("l'élément ", *it, " est dans la liste");
}
```

Remplir un vector

```
vector<int> a{1, 4, 5, 5, 2, 5};
fill(a.begin(), a.end, 0); // a: 0, 0, 0, 0, 0, 0
```

```
vector<int> a{1, 4, 5, 5, 2, 5};
iota(a.begin(), a.end(), 0); // a: 0, 1, 2, 3, 4, 5
```

Enlever des éléments

```
vector<int> a{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
a.erase(a.end() - 2);
// a: 0, 1, 2, 3, 4, 5, 6, 7, 9
a.erase(a.begin() + 3, a.begin() + 5);
// a: 0, 1, 2, 5, 6, 7, 9
a.erase(a.begin(), a.end()); // a est vide
```

Enlever les doublons

```
vector<int> a{1, 4, 5, 5, 2, 5};
sort(a.begin(), a.end()); // trier
// a: 1, 2, 4, 5, 5, 5
a.erase(unique(a.begin(), a.end()), a.end());
// a: 1, 2, 4, 5
```

Des fonctions comme arguments

Trier par ordre décroissant

```
bool is_greater_than(int lhs, int rhs) {
    return lhs > rhs;
}

vector<int> a{2, 6, 3, 5, 8, 4, 2, 7, 8};
sort(a.begin(), a.end(), is_greater_than);
// a: 8 8 7 6 5 4 3 2 2
```

Trouver un élément impair

```
bool impair(int n) {
    return n % 2 == 1;
}

vector<int> a{2, 6, 3, 5, 8, 4, 2, 7, 8};
vector<int>::iterator it =
    find_if(a.begin(), a.end(), impair);
if (it != a.end()) {
    print("nombre impair trouvé :", *it);
} else {
    print("toutes les valeurs sont paires");
}
```

Enlever des éléments qui satisfont une contrainte

```
bool impair(int n) {
    return n % 2 == 1;
}

vector<int> a{2, 6, 3, 5, 8, 4, 2, 7, 8};
vector<int>::iterator it =
    remove_if(a.begin(), a.end(), impair);
// a: 2 6 8 4 2 8 2 7 8
a.erase(it, a.end());
// a: 2 6 8 4 2 8
```

Documentation

https://en.cppreference.com/w/cpp/algorithm