Programming

Daniel Balle 2018

Iterators

Sequence containers like *vectors* or *strings* offer random access iterators for either ordered traversal using `begin` and `end`, or reversed traversal with `rbegin` and `rend`.

```
c = some container
for (auto it = c.begin(); it != c.end(); i++) ...
for (auto rit = c.rbegin(); rit != c.rend(); i++)
    // use *rit
```

Remark. Such containers can be used for *range-based* for loops.

Range-Based For Loop

Range-based for loops are used to execute statements through any `range` defined by *iterators* `begin` and `end`.

```
for (auto & i : c) ...
```

Remark. Using a reference `&` is almost always preferred. For *read-only* purposes, preface with `const`.

Element Access

Noteworthy element access functions are `front` and `back` which return a direct reference for respectively the first and last element of a sequence container.

```
c = some container
c.front() += 10;
c.back() = 42;
```

Remark. Complement basic functions `operator[]` and `at`.

---

constexpr

`constexpr`

Sorting

`std:sort` can be used to sort elements in a range defined by random access *iterators* `[first, last)` using either the default `operator<` or a custom *binary* function `cmp`.

```
#include <algorithm>
bool cmp(const Type1 &a, const Type2 &b) { ... }
sort(c.begin(), c.end())
sort(c.begin()+4, c.end(), cmp)
```

Performs $O(n \log_2 n)$ comparison according to the `C++` standard. Implementations differ but usually use *introsort* or hybrids.

Remark. For a *stable* sorting algorithm, use `stable_sort`.

String Stream

Type `stringstream` defined in `sstream` allows a string to be treated like a stream, thus allowing extraction `<<` and insertion `>>`.

```
#include <iostream>
#include <sstream>
string s, int i;
getline(cin, s);
stringstream(s) >> i;
```

Reverse

```
std:reverse(c.begin(), c.end())
```

Vector

C++, Data Structures