

Iterator for loop

```
for(std::vector<int>::iterator it = v.begin(); it
!= v.end(); ++it) {
    it->doSomething();
}
```

Lambda

[captures] (parameters) mutable -> return-type
 {body}

[] = argument att fånga (behandlas som globala variabler).

() = normala funktions argument.

-> return -type = datatype att returnera.

mutable = lägg till om du vill ändra på något.

Non-const function.

{ } = function goes here.

Lambda går att skriva helt utan (), -> return -type och mutable.

Felhantering

```
try {
    foo() //foo kastar ett error med
    throw( "Feltext") någonstans i koden
}
catch( exception & e)
{
    std::cout << e.what() << std::endl; //
    blir "Feltext"
}
```

for_each loop med lambda

```
vector<int> v;
int sum{0};
for_each( v.begin(), v.end(), sum -> void {
    sum += x;
});
```

Läsa från fil med ifstream

```
ifstream file("/some/path")
string line;
vector<string> file_content;
while (getline( file, line)) {
    file_content.push_back(line);
}
```

Range-based for loop

```
vector<int> v;
for(int i : v) { // i = förväntad datatype
    //do something
}
```

Hitta saker med egen jämförelse

```
find_if (InputIt first, InputIt last,
UnaryPredicate boolFunction) //returns iterator to
element if found, otherwise returns last
find_if_not() finns också.
```

Kommandoradsargument med validering

```
int main(int argc, char* argv[]) { // plocka ut
argument.
    vector<string> arg(argv, argv + argc);
    int num_of_iterations;
    istream arg(arg.at(1));
    if( !(arg >> num_of_iterations)) //om
    arg inte kan göras till int så kastas logic_error
    {
        throw logic_error( "Error
        meddelande");
    }
}
```

Huvudklass till subclass (Employee ← Boss)

```
Boss b = dynamic_cast<Boss>(employee);
if (b != nullptr) {
    std::cout << b -> get_bonus
}
```

Konstruktörer/tilldelning

```
Ptr(Ptr const& rhs); //kopieringskonstruktor
Ptr& operator = (Ptr const& rhs); //kopierings -
tilldelning
Ptr(Ptr && this) //flyttkonstruktor
Ptr& operator = (Ptr && rhs(?)) //flytt tilldelning
```

Slumpa tal

```
#include <random>

std::random_device rnd;
std::uniform_int_distribution<int>
num(1, 100); // tal mellan 1-100
std::cout << num(rnd) << std::endl;
```



By **Dockson**

cheatography.com/dockson/

Published 6th December, 2015.

Last updated 12th May, 2016.

Page 2 of 2.

Sponsored by **Readable.com**

Measure your website readability!

<https://readable.com>