TDP004
STL

Eric Ekström

Institutionen för datavetenskap

LINKÖPING UNIVERSITY

STL - Intro

Vad är programmering?

STL - Intro

Vad är programmering?

Vissa säger att allting är

1. Läs in data
2. Lagra data
3. Processa data
4. Returnera data

LINKÖPING UNIVERSITY

STL - std::vector

```cpp
#include <vector>

int main()
{
  std::vector<int> v {};
  v.push_back(1);
  v.push_back(5);
  v.push_back(2);

  std::cout << v.at(0) << v.at(1) << v.at(2) << std::endl;
  return 0;
}
```

STL - std::set

```
#include <set>

int main()
{
  std::set<double> s { 1.0, 3.0, -1.0 };

  s.insert(3.14);
  s.erase(1.0);
}
```

En sorterad mängd av värden.

- Snabb insättning och åtkomst
- Ingen indexering
- Tillåter inte dubbletter.

STL - std::deque

```
#include <deque>

int main()
{
  std::deque<char> d { 'h', 'e', 'j' };

  d.push_back('!');
  d.pop_front();
}
```

- Generellt lite långsammare än std::vector, men
- Insättning i början och slutet snabbt

STL - std::map

```
std::map<std::string, int> m { {"a", 1},
                               {"b", 2} };

// Åtkomst av befintligt värde
std::cout << m["a"];
m["b"] = 7;

// Sätter in ett nytt värde
m["c"] = 2;
```

- Associativ behållare.
- Sparar ett par av [nyckel, värde]
- Nycklar måste gå att jämföra

LINKÖPING UNIVERSITY

Iteratorer

```
int main()
{
  vector<int> v {1, 2, 3};
  for (int i{0}; i < v.size(); ++i)
  {
    cout << v[i] << endl;
  }
}
```

Iteratorer

```
int main()
{
  set<int> v {1, 2, 3};
  for (int i{0}; i < v.size(); ++i)
  {
    cout << v[i] << endl; // Fungerar inte!
  }
}
```

Iteratorer

```
int main()
{
  vector<int> v {1, 2, 3};
  for (int e : v)
  {
    cout << e << endl;
  }
}
```

Iteratorer

```
int main()
{
  set<int> v {1, 2, 3};
  for (int e : v)
  {
    cout << e << endl; // Fungerar!
  }
}
```

Iteratorer

```
for (int x : v)
{
  cout << x << endl;
}
```

Iteratorer

```
using iterator = std::vector<int>::iterator;

for (iterator it {container.begin()}; it != container.end(); ++it)
{
  auto x {*it};
  cout << x << endl;
}
```

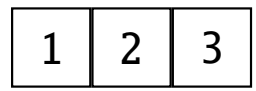
Iteratorer

Iteratorer är generaliserade pekare.

- `begin()` och `end()` ger iteratorer till början respektive slutet av en behållare.
- En iterator har (som minst) funktionerna
  - `operator++`
  - `operator*`
  - `operator==`
  - `operator!=`
- Olika implementation beroende på behållare

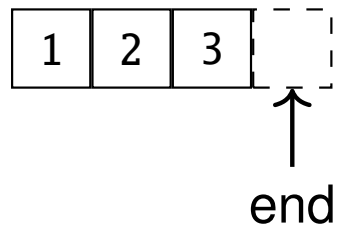
Iteratorer

Vart pekar `end()`?



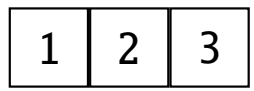
Iteratorer

Vart pekar `end()`?



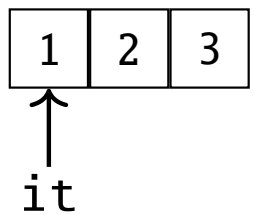
Iteratorer - Exempel

```
vector<int> v{1,2,3};
```



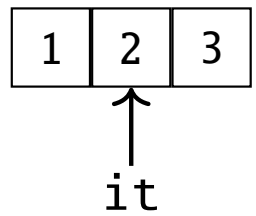
Iteratorer - Exempel

```
vector<int>::iterator it{v.begin()};
```



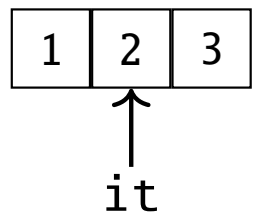
Iteratorer - Exempel

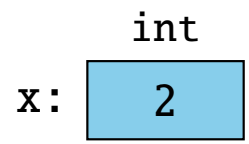
```
++it;
```



Iteratorer - Exempel

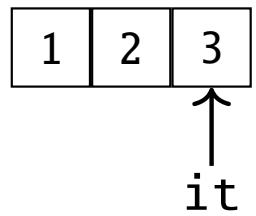
```
int x{*it};
```





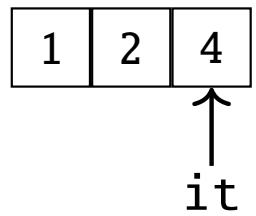
Iteratorer - Exempel

```
++it;
```



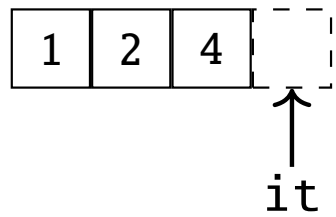
Iteratorer - Exempel

```
*it = 4;
```



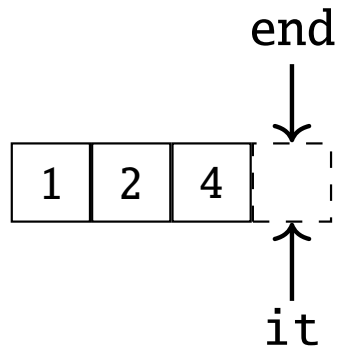
Iteratorer - Exempel

```
++it;
```



Iteratorer - Exempel

```
it == v.end();
```



1 Intro
2 Behållare
3 Iteratorer
4 Standardbiblioteket
5 Algoritmer
6 Lambda-uttryck
7 Blandade exempel

Standardbiblioteket

- Tillgängligt för alla
- Löser vanliga problem
- Effektivt
- Uppdelat i komponenter

Standardbiblioteket

Standard **T**emplate **L**ibrary

Standardbiblioteket

Designprinciper:

- Ska vara så generellt som möjligt

Standardbiblioteket

Designprinciper:

- Ska vara så generellt som möjligt
- Löser vanliga problem

Standardbiblioteket

Designprinciper:

- Ska vara så generellt som möjligt
- Löser vanliga problem
- Måste fungera med användarens kod

Standardbiblioteket

Designprinciper:

- Ska vara så generellt som möjligt
- Löser vanliga problem
- Måste fungera med användarens kod
- Effektivt nog att ersätta egengjorda lösningar

Standardbiblioteket

Designprinciper:

- Ska vara så generellt som möjligt
- Löser vanliga problem
- Måste fungera med användarens kod
- Effektivt nog att ersätta egengjorda lösningar
- Robust felhantering

1 Intro
2 Behållare
3 Iteratorer
4 Standardbiblioteket
5 Algoritmer
6 Lambda-uttryck
7 Blandade exempel

LINKÖPING UNIVERSITY

Algoritmer

Vad gör denna kod?

```
int main()
{
  std::vector<int> v {1, 2, 3, 1, 4, 1};
  int result {0};
  for (auto it{v.begin()}; it != v.end(); ++it)
  {
    if (*it == 1)
    {
      result++;
    }
  }
}
```

Algoritmer

Vad gör denna kod?

```
int main()
{
  std::vector<int> v {1, 2, 3, 1, 4, 1};
  int result {std::count(v.begin(), v.end(), 1)};
}
```

Algoritmer

Vad gör denna kod?

```
int main()
{
  std::vector<int> v {1, 2, 3, 1, 4, 1};
  int result {std::count(v.begin(), v.end(), 1)};
}
```

Vilka algoritmer finns? www.cppreference.com!

Algoritmer

```
std::vector<int> v {1, 2, 3, 1, 4, 1};
auto it { std::find(v.begin(), v.end(), 2) };
```

Vilka mönster ser vi?

Algoritmer

```
std::vector<int> v {1, 2, 3, 1, 4, 1};
auto it { std::find(v.begin(), v.end(), 2) };
```

Vilka mönster ser vi?

- Alla algoritmer tar två iteratorer
- Ger ut ett resultat
  - std::count -> int
  - std::find -> std::vector<int>::iterator

Modifierande algoritmer

Om vi vill ändra i en behållare då?

```
std::vector<int> v {1, 2, 3, 1, 4, 1};
std::fill(v.begin(), v.end(), 8);
```

Modifierande algoritmer

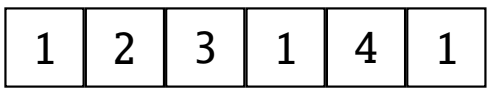
Hur gör vi om vi vill ta bort värden?

```
std::vector<int> v {1, 2, 3, 1, 4, 1};

for (auto it {v.begin()}; it != v.end(); ++it)
{
  if (*it == 1)
  {
    v.erase(it);
  }
}
```

Fungerar inte!

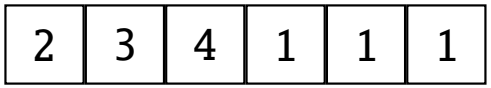
Modifierande algoritmer

```
std::vector<int> v {1, 2, 3, 1, 4, 1};

v.erase(std::remove(v.begin(), v.end(), 1),
        v.end());
```



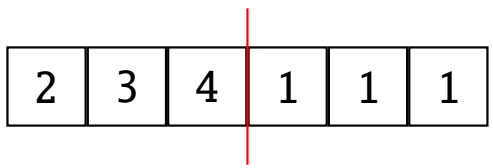
Modifierande algoritmer

```
std::vector<int> v {1, 2, 3, 1, 4, 1};

v.erase(std::remove(v.begin(), v.end(), 1),
        v.end());
```



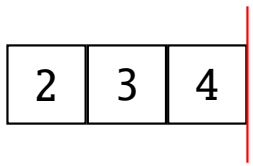
Modifierande algoritmer

```
std::vector<int> v {1, 2, 3, 1, 4, 1};

v.erase(std::remove(v.begin(), v.end(), 1),
        v.end());
```



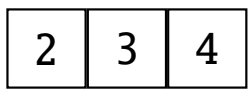
Modifierande algoritmer

```
std::vector<int> v {1, 2, 3, 1, 4, 1};

v.erase(std::remove(v.begin(), v.end(), 1),
        v.end());
```



Modifierande algoritmer

```
std::vector<int> v {1, 2, 3, 1, 4, 1};

v.erase(std::remove(v.begin(), v.end(), 1),
        v.end());
```



Modifierande algorithmer

```
std::vector<int> old {1, 2, 3, 1, 4, 1};

std::vector<string> v {};

std::copy(old.begin(), old.end(), v.begin());
```

Modifierande algorithmer

```
std::vector<int> old {1, 2, 3, 1, 4, 1};

std::vector<string> v ( old.size() );

std::copy(old.begin(), old.end(), v.begin());
```

Inläsning med algoritmer

```
int x { };
while (std::cin >> x)
{
  v.push_back(x);
}
```

Inläsning med algoritmer

```
int x { };
while (std::cin >> x)
{
  v.push_back(x);
}
```

```
using namespace std;

int main()
{
  vector<int> v {};
  copy(istream_iterator<int>{cin},
       istream_iterator<int>{},
       back_inserter(v));
}
```

Utskrift med algoritmer

```
for (int x : v)
{
  cout << x << " ";
}
```

Utskrift med algoritmer

```
for (int x : v)
{
  cout << x << " ";
}
```

```
using namespace std;

int main()
{
  vector<int> v {};
  copy(v.begin(), v.end(),
       ostream_iterator<int>{cout, " "});
}
```

std::transform

```
int add_2(int n)
{
  return n + 2;
}

int main()
{
  std::vector<int> v {1, 2, 3};
  std::vector<int> result (v.size());
  std::transform(v.begin(), v.end(),
                 result.begin(), add_2);
}
```

std::transform

```
int add_2(int n)
{
  return n + 2;
}

int main()
{
  std::vector<int> v {1, 2, 3};
  std::vector<int> result (v.size());
  std::transform(v.begin(), v.end(),
                 result.begin(), add_2);
}
```

Måste vi skriva funktionen separat?

LINKÖPING UNIVERSITY

Lambda

```
int main()
{
  std::vector<int> w(v.size());
  std::transform(v.begin(), v.end(), w.begin(),
                 [](int x)
                 {
                   return x + 2;
                 });
}
```

Lambda - capture

```
int main()
{
  int n {};
  std:.cin >> n;
  std::vector<int> w(v.size());

  if (n == 1)
  {
    std::transform(v.begin(), v.end(), w.begin(),
                   [](int x) {return x + 1;});
  }
  else if (n == 2)
  {
    std::transform(v.begin(), v.end(), w.begin(),
                   [](int x) {return x + 2;});
  }
  //...
}
```

Dåligt!

Lambda - capture

```
std::transform(v.begin(), v.end(), w.begin(),
               [n](int x)
               {
                 return x + n;
               });
```

Lambda - capture

```
int n { 2 };
std::vector<int> w(v.size());

auto add = [n](int x) { return x + n; };

std::transform(v.begin(), v.end(), w.begin(), add);
n = 3;
std::transform(v.begin(), v.end(), w.begin(), add);
n = 5;
std::transform(v.begin(), v.end(), w.begin(), add);
```

Lambda - capture

```
int n { 2 };
std::vector<int> w(v.size());

auto add = [&n](int x) { return x + n; };

std::transform(v.begin(), v.end(), w.begin(), add);
n = 3;
std::transform(v.begin(), v.end(), w.begin(), add);
n = 5;
std::transform(v.begin(), v.end(), w.begin(), add);
```

Lambda-uttryck

```
[capture](parametrar) -> returtyp
{
  funktionsblock
}
```

1 Intro
2 Behållare
3 Iteratorer
4 Standardbiblioteket
5 Algoritmer
6 Lambda-uttryck
7 Blandade exempel

LINKÖPING UNIVERSITY

Exempel #1

Vi vill kontrollera att varje tal i en behållare är jämnt.

Exempel #1

Vi vill kontrollera att varje tal i en behållare är jämnt.

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

int main()
{
  std::vector<int> v {28,12,8,14,9,20,4};

  bool all_even = std::all_of(v.begin(), v.end(),
                              [](int i)
                              {
                                return i % 2 == 0;
                              });
}
```

Exempel #2

Sortera alla tal som ligger mellan det första negativa talet och slutet (i fallande ordning).

Exempel #2

Sortera alla tal som ligger mellan det första negativa talet och slutet (i fallande ordning).

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

int main()
{
  std::vector<int> v {28,12,8,14,9,20,4};

  auto it { std::find_if(v.begin(), v.end(),
                         [](int i)
                         {
                           return i < 0;
                         })};
  std::sort(it, v.end(),
            [](int lhs, int rhs)
            {
              return lhs > rhs;
            });
}
```

Exempel #3

Räkna förekomsten av varje tecken.

Exempel #3

Räkna förekomsten av varje tecken.

```
#include <algorithm>
#include <vector>
#include <map>

int main()
{
  std::vector<char> v {'a','i','e','i','y','a','i'};
  std::map<char, int> m {};

  std::for_each(v.begin(), v.end(),
                [&m](char c)
                {
                  m[c]++;
                }
}
```

Exempel #4

Går det att sortera en `std::map`? `std::list`?

Exempel #4

Går det att sortera en `std::map`? `std::list`?

```
#include <algorithm>
#include <map>
#include <list>

int main()
{
  std::map<int, int> m {};
  std::list<int> l {};

  std::sort(m.begin(), m.end());
  std::sort(l.begin(), l.end());
}
```

Exempel #4

Går det att sortera en `std::map`? `std::list`?

```
#include <algorithm>
#include <map>
#include <list>

int main()
{
  std::map<int, int> m {};
  std::list<int> l {};

  std::sort(m.begin(), m.end());
  std::sort(l.begin(), l.end());
}
```

KOMPILERAR EJ!

Iteratorkategorier

	Category				
Operationer	**Input**	**Output**	**Forward**	**Bidirectional**	**Random Access**
==, !=	✓	✓	✓	✓	✓
*, ->	Read	Write	Read/Write	Read/Write	Read/Write
++	✓	✓	✓	✓	✓
–	-	-	-	✓	✓
+, +=, -, -=	-	-	-	-	✓
<, <=, >, >=	-	-	-	-	✓
i[n]	-	-	-	-	✓

www.ida.liu.se/~TDP004/