TDP004 - Objektorienterad programmering

Standard Template Library

Pontus Haglund & Rasmus Jonsson

Institutionen för datavetenskap

LINKÖPING UNIVERSITY

1 Mål med föreläsningen
2 Filströmmar
3 Containers
4 Iteratorer
5 Algoritmer

LINKÖPING UNIVERSITY

1 Mål med föreläsningen
2 Filströmmar
3 Containers
4 Iteratorer
5 Algoritmer

LINKÖPING UNIVERSITY

Mål med föreläsningen

Efter föreläsningen skall studenten kunna:

- Använda strängar och strängfunktioner.
- Använda olika containers i c++.
- Använda filströmmar.
- Förstå iterator och deras användningsområden
- Använda de vanligaste algoritmerna i c++
- Hitta lämpliga algoritmer för särskilda fall.
- Använda funktionsobjekt i c++

1 Mål med föreläsningen
2 Filströmmar
3 Containers
4 Iteratorer
5 Algoritmer

LINKÖPING UNIVERSITY

Filströmmar

`fstream`

```
#include <fstream>

int main()
{
  ifstream fs { "test.txt" };
  int x;
  fs >> x;
}
```

Filströmmar

öppningslägen

```
#include <fstream>

int main()
{
  fstream fs { "test.txt", ios::out | ios::app };
  int x;
  fs >> x;
}
```

Filströmmar

skriva till fil

```
#include <fstream>
int main()
{
  ofstream fs { "test.txt" };
  int x {10};
  fs << "x = " << x << endl;
  fs << setw(6) << "hej";
}
```

Filströmmar

fil till behållare

```
#include <fstream>
int main()
{
  ifstream ifs{"data.txt"};
  vector<int> v{};
  int data;
  while (ifs >> data)
  {
    v.push_back(data);
  }
}
```

1 Mål med föreläsningen
2 Filströmmar
3 Containers
4 Iteratorer
5 Algoritmer

LINKÖPING UNIVERSITY

Containers

Strängar

```
string s{"Detta är en sträng"};
```

Containers

Vektorer

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

Containers

array

```
array<int, 6> a{1, 2, 3};
```

Containers

list

```
list<int> l{1, 2, 3};
```

Containers

set

```
set<int> s{1, 2, 3, 3};
```

Containers

map

```
map<string, int> m{};
map<string, int> m{pair<string, int> {"a",1},
                   pair<string, int> {"b",2}};
cout << m["b"] << endl;
m["c"] = 3
cout << m["c"] << endl;
```

Containers

loopa genom map

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

for( pair<string, int> const& p: m )
{
  cout << p.first << ": " << p.second << endl;
}
```

1 Mål med föreläsningen
2 Filströmmar
3 Containers
4 Iteratorer
5 Algoritmer

Iteratorer

Iteratorer

```
vector<int> v{1, 2, 3};
vector<int>::iterator it = begin(v);
cout << *it << endl;
```

Iteratorer

Stega iteratorer

```
vector<int> v{1, 2, 3};
vector<int>::iterator it = begin(v);
++it;
cout << *it << endl;
```

Iteratorer

Loopa med iteratorer

```
vector<int> v{1, 2, 3};
vector<int>::iterator it = begin(v);
while ( it != end(v) )
{
  cout << *it << endl;
  ++it;
}
```

Iteratorer

For-loop med iteratorer

```
vector<int> v{1, 2, 3};
for( vector<int>::iterator it{ begin(v) }; it != end(v); ++it )
{
  cout << *it << endl;
}
```

Iteratorer

auto och iteratorer

```
vector<int> v{1, 2, 3};
for( auto it{ begin(v) }; it != end(v); ++it )
{
  cout << *it << endl;
}
```

Iteratorer

Constant containers och pekare

```
vector<int> const v{1, 2, 3};
for( auto it{ cbegin(v) }; it != cend(v); ++it )
{
  cout << *it << endl;
}
```

Iteratorer

Reverse iterator

```
vector<int> v{1, 2, 3};
for( auto it{ rbegin(v) }; it != rend(v); ++it )
{
  cout << *it << endl;
}
```

1 Mål med föreläsningen
2 Filströmmar
3 Containers
4 Iteratorer
5 Algoritmer

LINKÖPING UNIVERSITY

Algoritmer

`find`

```
vector<int> v {1, 2, 3};
auto it { find(begin(v), end(v), 3) };
if( it == v.end() )
{
  // vi hittade ingenting!
}
```

Algoritmer

count

```
vector<int> v {1, 2, 3, 1, 4, 1};
int count {count(begin(v), end(v), 1)};
cout << count << endl; // borde skriva ut 3
```

Algoritmer

copy

```
int main(int argc, char* argv[])
{
  vector<string> args{argv, argv + argc};
  // här måste vi ha paranteser, anropar en specifik konstruktor
  vector<string> v(args.size());
  copy(begin(args), end(args), begin(v));
}
```

Algoritmer

copy med backinserter

```
int main(int argc, char* argv[])
{
  vector<string> args{argv, argv + argc};
  // här måste vi ha paranteser, anropar en specifik konstruktor
  vector<string> v(args.size());
  copy(begin(args), end(args), backinserter(v));
}
```

Algoritmer

`copy` med `backinserter` från `cin`

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

Algoritmer

skriva till `cout` med `copy`

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

Algoritmer

`transform`

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

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

Algoritmer

`transform` med `lambda`

```
vector<int> v{1, 2, 3};
vector<int> new_v(v.size());
transform( begin(v), end(v), begin(new_v),
                []( int const i) -> int{ return i+2; } );
```

Algoritmer

`transform` med `lambda`

```
vector<string> args{argv, argv + argc};
vector<int> v(args.size());
transform( args.begin(), args.end(), v.begin(),
             [](string const& str) -> int{ return stoi(str); } )
```

Algoritmer

`lambda` captures

```
int x{};
auto f {[x](){return x;}};
cout << f() << endl;
x = 2;
cout << f() << endl;
```

Algoritmer

`for_each` den tråkigaste algoritmen

```
vector<string> args{argv, argv + argc};
for_each(begin(args), end(args),
         [](string const& str) -> void{cout << str << endl;});
```

Algoritmer

`cppreferens`

Använd cppreferens för att hitta lämpliga algoritmer:
`https://en.cppreference.com/w/cpp/header/algorithm`

Eller titta på specifika behållare, exempelvis:
`https://en.cppreference.com/w/cpp/string/basic_string`

www.liu.se

LINKÖPING UNIVERSITY