TDDI82 - Objektorienterad problemlösning

Storseminarium – standardbibliotek

Christoffer Holm

Institutionen för datavetenskap

LINKÖPING UNIVERSITY

1. Upplägg av storseminarium
2. Behållare
3. Algoritmer
4. Modifierande algoritmer
5. Lambdafunktioner
6. Övningsuppgifter

1 Upplägg av storseminarium
2 Behållare
3 Algoritmer
4 Modifierande algoritmer
5 Lambdafunktioner
6 Övningsuppgifter

LINKÖPING UNIVERSITY

Upplägg av storseminarium

Introduktion

- Storseminariet är en kombination mellan föreläsning och labbpass
- Vi kommer upprepa följande upplägg:
  1. Ni löser en uppgift
  2. Diskussion i halvklass
  3. Genomgång av material
- Under tiden som ni löser uppgifter kommer lärare och assistenter gå runt i salen och hjälpa till.
- Om ni blir klara innan utsatt tid: prata med en assistent, de kan ge er utmaningar!

Upplägg av storseminarium

Ungefärlig tidslinje

- 13:20 - 13:40: Uppgift #1
- 14:00 - 14:30: Uppgift #2 + 15 min rast, välj själv när
- 15:00 - 15:40: Uppgift #3 + 15 min rast, välj själv när
- 16:15 - 17:00: Övningsuppgifter

Sluttiderna är då ni ska vara tillbaka i salen för då börjar genomgången.

1 Upplägg av storseminarium
2 Behållare
3 Algoritmer
4 Modifierande algoritmer
5 Lambdafunktioner
6 Övningsuppgifter

LINKÖPING UNIVERSITY

Behållare

Uppgift #1

Skriv ett program som läser in heltal från `std::cin` till en lämplig behållare. Användaren avslutar inmatningen genom att trycka `ctrl+D`.
Skriv sedan ut alla *unika* heltal som användaren matade in. Notera att dessa ska skrivas ut i *sorterad* ordning (i stigande ordning). Inga dubletter av värden får förekomma i utskriften, trots att det potentiellt förekommer dubletter i inmatningen.
Tips: Ditt val av behållare kommer ha en stor påverkan på komplexiteten av din lösning, så fundera noggrannt.

Algoritmer

Vad gör detta program?

```
1  std::vector<int> values { /* ... */ };
2
3  int x { 0 };
4  int y { 1 };
5
6  for (auto it = values.begin(); it != values.end(); ++it)
7  {
8    if (*it == y)
9    {
10     ++x;
11   }
12 }
13
14 std::cout << x << std::endl;
```

Algoritmer

Jämför med:

```
1 std::vector<int> values { /* ... */ };
2 int x { std::count(values.begin(), values.end(), 1) };
3 std::cout << x << std::endl;
```

Algoritmer

Vad gör denna kod?

```
1  std::vector<int> values { /* ... */ };
2
3  int y { 1 };
4  auto it = values.begin();
5
6  while (it != values.end())
7  {
8    if (*it == y)
9    {
10     break;
11   }
12   ++it;
13 }
14
15 // vad är it här?
```

Algoritmer

Jämför förra exemplet med:

```
1 std::vector<int> values { /* ... */ };
2 auto it = std::find(values.begin(), values.end(), 1);
3
4 // vad är it här?
```

Algoritmer

cppreference.com

Öppna följande sida: `https://en.cppreference.com/w/cpp/algorithm`

Algoritmer

Uppgift #2

Läs in en behållare av heltal från `std::cin` (precis på samma sätt som i förra uppgiften). När användaren är klar med sin inmatning trycker denne `ctrl+D`.
Använd sedan lämpliga algoritmer för att hitta det största värdet i behållaren och skriv sedan ut hur många gånger det största värdet förekom i behållaren.
Notera: Poängen med uppgiften är att söka upp lämpliga algoritmer på cppreference.

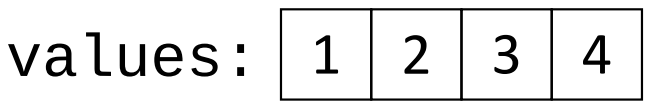
1 Upplägg av storseminarium
2 Behållare
3 Algoritmer
4 Modifierande algoritmer
5 Lambdafunktioner
6 Övningsuppgifter

LINKÖPING UNIVERSITY

Modifierande algoritmer

`std::fill`

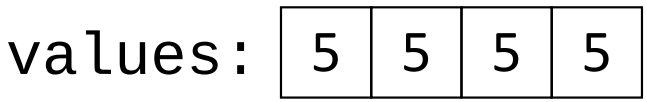
```
1 std::vector<int> values { 1, 2, 3, 4 };
2 std::fill(values.begin(), values.end(), 5);
```



Modifierande algoritmer

`std::fill`

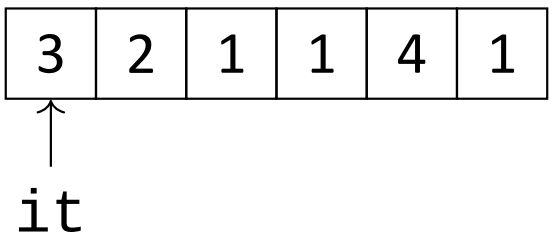
```
1 std::vector<int> values { 1, 2, 3, 4 };
2 std::fill(values.begin(), values.end(), 5);
```



Modifierande algoritmer

Borttagning av värden

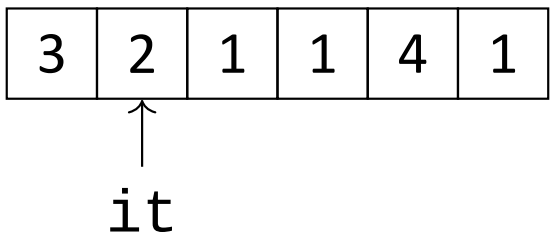
```
1  std::vector<int> v { 3, 2, 1, 1, 4, 1 };
2  for (auto it { v.begin() }; it != v.end(); ++it)
3  {
4    if (*it == 1)
5    {
6      v.erase(it);
7    }
8  }
```



Modifierande algoritmer

Borttagning av värden

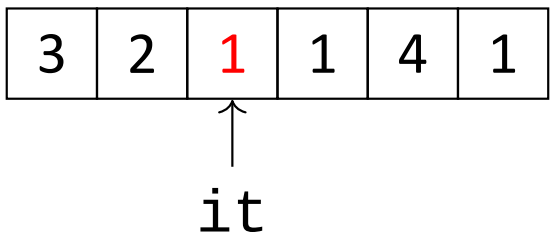
```
1  std::vector<int> v { 3, 2, 1, 1, 4, 1 };
2  for (auto it { v.begin() }; it != v.end(); ++it)
3  {
4    if (*it == 1)
5    {
6      v.erase(it);
7    }
8  }
```



Modifierande algoritmer

Borttagning av värden

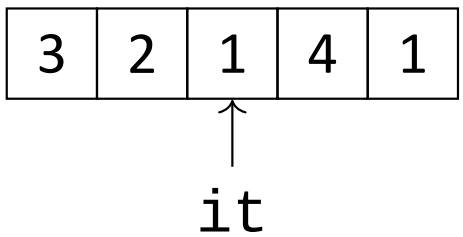
```
1  std::vector<int> v { 3, 2, 1, 1, 4, 1 };
2  for (auto it { v.begin() }; it != v.end(); ++it)
3  {
4    if (*it == 1)
5    {
6      v.erase(it);
7    }
8  }
```



Modifierande algoritmer

Borttagning av värden

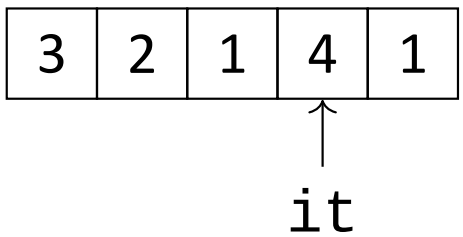
```
1  std::vector<int> v { 3, 2, 1, 1, 4, 1 };
2  for (auto it { v.begin() }; it != v.end(); ++it)
3  {
4    if (*it == 1)
5    {
6      v.erase(it);
7    }
8  }
```



Modifierande algoritmer

Borttagning av värden

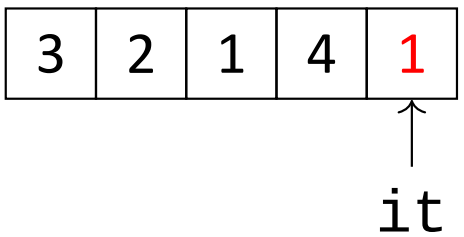
```
1  std::vector<int> v { 3, 2, 1, 1, 4, 1 };
2  for (auto it { v.begin() }; it != v.end(); ++it)
3  {
4    if (*it == 1)
5    {
6      v.erase(it);
7    }
8  }
```



Modifierande algoritmer

Borttagning av värden

```
1  std::vector<int> v { 3, 2, 1, 1, 4, 1 };
2  for (auto it { v.begin() }; it != v.end(); ++it)
3  {
4    if (*it == 1)
5    {
6      v.erase(it);
7    }
8  }
```



Modifierande algoritmer

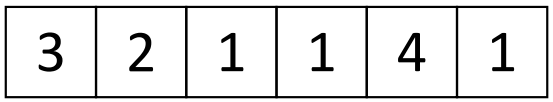
Lösning

```
1  std::vector<int> v { 3, 2, 1, 1, 4, 1 };
2  for (auto it { v.begin() }; it != v.end();)
3  {
4    if (*it == 1)
5    {
6      it = v.erase(it);
7    }
8    else
9    {
10     ++it;
11   }
12 }
```

Modifierande algoritmer

Borttagning med algoritm

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



Modifierande algoritmer

Borttagning med algoritm

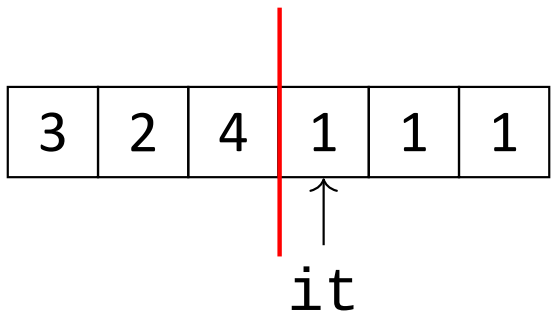
```
1 std::vector<int> v { 3, 2, 1, 1, 4, 1 };
2 auto it = std::remove(v.begin(), v.end(), 1);
3 v.erase(it, v.end());
```

3	2	4	1	1	1

Modifierande algoritmer

Borttagning med algoritm

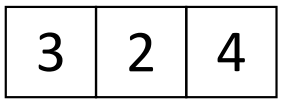
```
1 std::vector<int> v { 3, 2, 1, 1, 4, 1 };
2 auto it = std::remove(v.begin(), v.end(), 1);
3 v.erase(it, v.end());
```



Modifierande algoritmer

Borttagning med algoritm

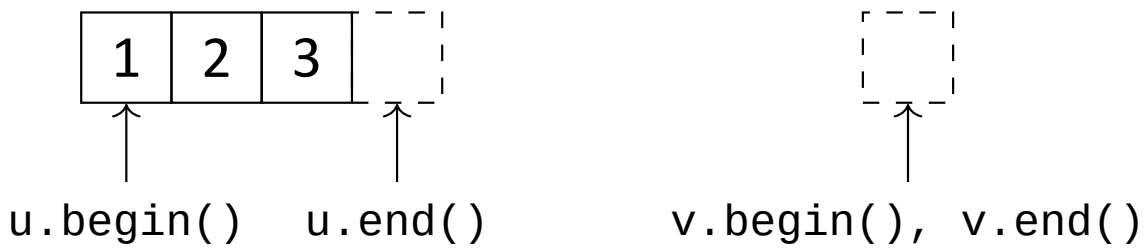
```
1 std::vector<int> v { 3, 2, 1, 1, 4, 1 };
2 auto it = std::remove(v.begin(), v.end(), 1);
3 v.erase(it, v.end());
```



Modifierande algoritmer

Kopiering till annan behållare

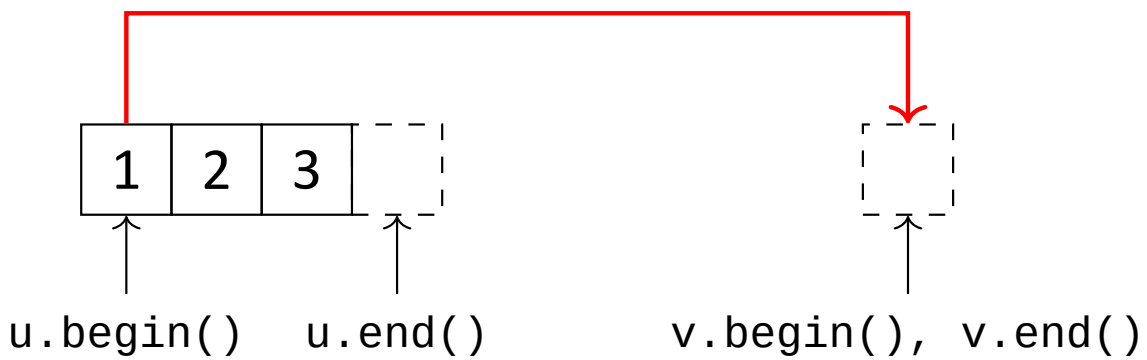
```
1 std::vector<int> u { 1, 2, 3 };
2
3 std::vector<int> v { };
4
5 std::copy(u.begin(), u.end(), v.begin());
```



Modifierande algoritmer

Kopiering till annan behållare

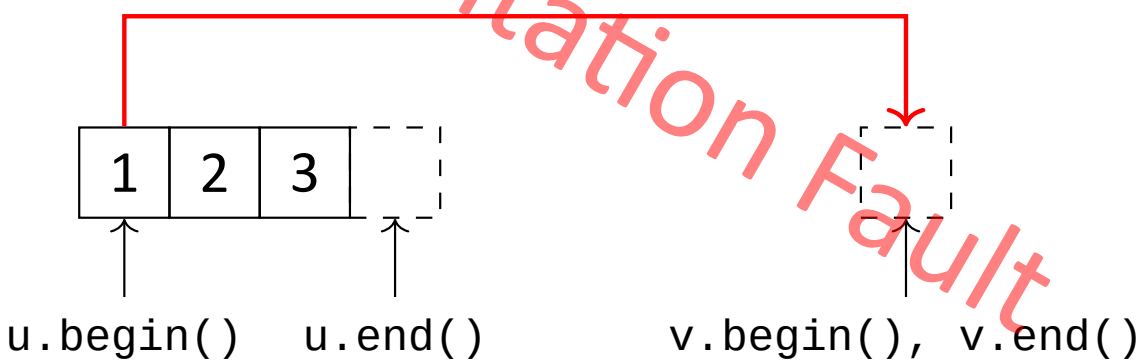
```
1 std::vector<int> u { 1, 2, 3 };
2
3 std::vector<int> v { };
4
5 std::copy(u.begin(), u.end(), v.begin());
```



Modifierande algoritmer

Kopiering till annan behållare

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

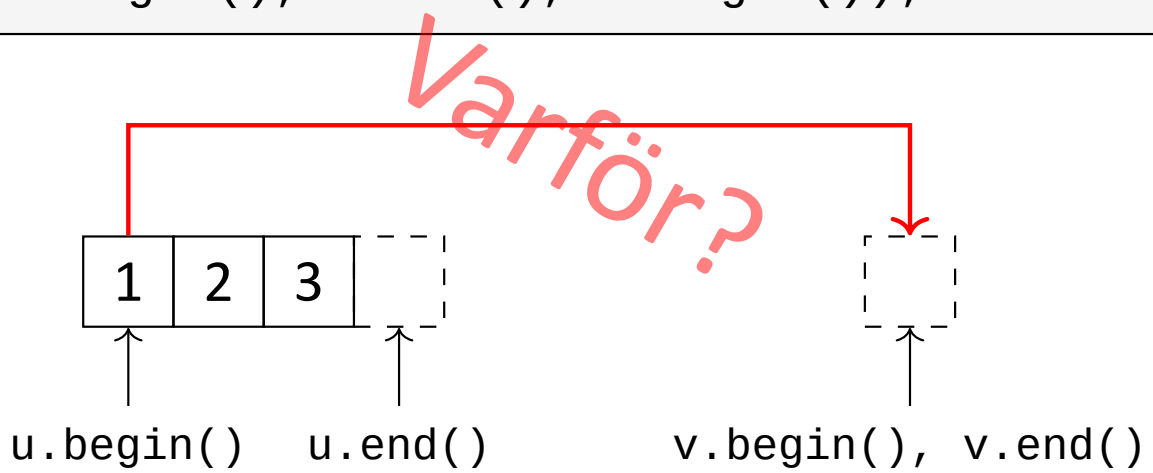


Segmentation Fault

Modifierande algoritmer

Kopiering till annan behållare

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



Modifierande algoritmer

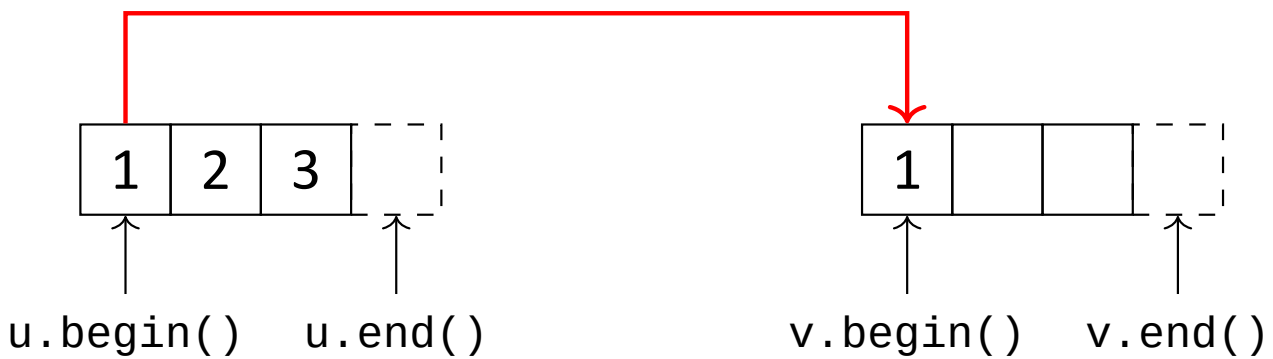
Lösning

```
1 std::vector<int> u { 1, 2, 3 };
2 // se till att `v` har lika många platser som `u`
3 std::vector<int> v(u.size());
4
5 std::copy(u.begin(), u.end(), v.begin());
```

Modifierande algoritmer

Lösning

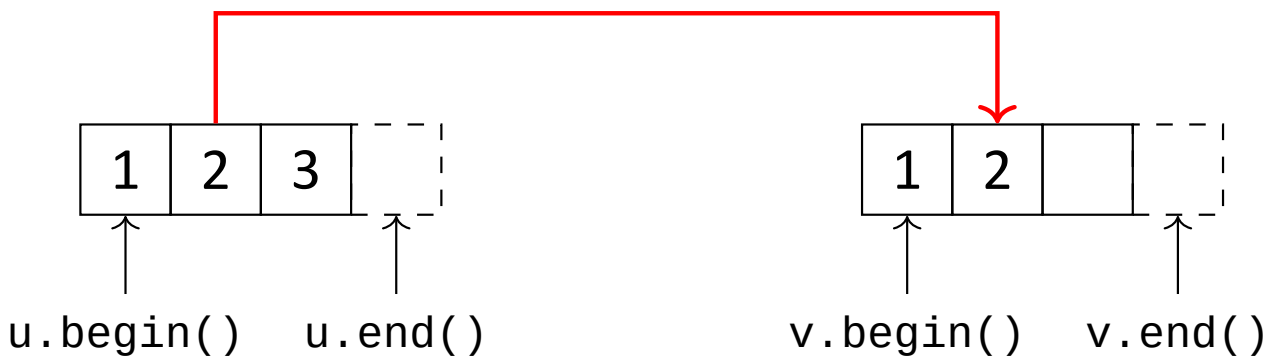
```
1 std::vector<int> u { 1, 2, 3 };
2 // se till att `v` har lika många platser som `u`
3 std::vector<int> v(u.size());
4
5 std::copy(u.begin(), u.end(), v.begin());
```



Modifierande algoritmer

Lösning

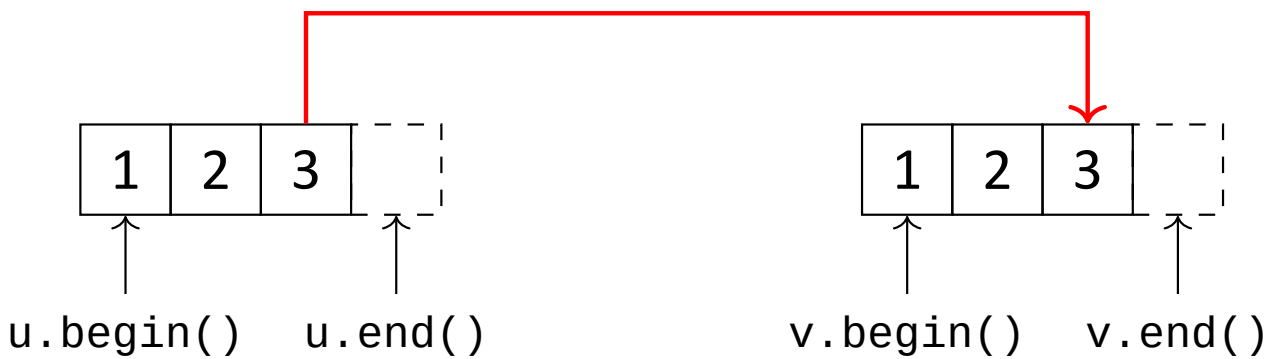
```
1 std::vector<int> u { 1, 2, 3 };
2 // se till att `v` har lika många platser som `u`
3 std::vector<int> v(u.size());
4
5 std::copy(u.begin(), u.end(), v.begin());
```



Modifierande algoritmer

Lösning

```
1 std::vector<int> u { 1, 2, 3 };
2 // se till att `v` har lika många platser som `u`
3 std::vector<int> v(u.size());
4
5 std::copy(u.begin(), u.end(), v.begin());
```



Modifierande algoritmer

Lösning

```
1 std::vector<int> u { 1, 2, 3 };
2 // se till att `v` har lika många platser som `u`
3 std::vector<int> v(u.size());
4
5 std::copy(u.begin(), u.end(), v.begin());
```

Funkar!

Modifierande algoritmer

Intressant problem: Inläsning

```
1  std::vector<int> v { };
2
3  int x { };
4  while (std::cin >> x)
5  {
6    v.push_back(x);
7  }
```

Modifierande algoritmer

Inläsning med algoritmer!

```
1 std::vector<int> v { };
2 std::copy(std::istream_iterator<int>{ std::cin },
3           std::istream_iterator<int>{ },
4           std::back_inserter(v));
```

Modifierande algoritmer

Inläsning med algoritmer!

```
1 std::vector<int> v { };
2 std::copy(std::istream_iterator<int>{ std::cin },
3           std::istream_iterator<int>{ },
4           std::back_inserter(v));
```

Modifierande algoritmer

Inläsning med algoritmer!

```
1 std::vector<int> v { };
2 std::copy(std::istream_iterator<int>{ std::cin },
3           std::istream_iterator<int>{ },
4           std::back_inserter(v));
```

Modifierande algoritmer

Inläsning med algoritmer!

```
1 std::vector<int> v { };
2 std::copy(std::istream_iterator<int>{ std::cin },
3           std::istream_iterator<int>{ },
4           std::back_inserter(v));
```

Modifierande algoritmer

Utskrift med algoritmer

```
1 std::vector<int> v { 1, 2, 3 };
2 for (int x : v)
3 {
4   std::cout << x << " ";
5 }
```

Modifierande algoritmer

Utskrift med algoritmer

```
1 std::vector<int> v { 1, 2, 3 };
2 std::copy(v.begin(), v.end(),
3           std::ostream_iterator<int>{ std::cout, " " });
```

Modifierande algoritmer

Uppgift #3

Implementera ett program som gör följande:

1. Läs in strängar från `std::cin` till en `std::vector`, användaren trycker `ctrl+D` när denne är klar.
2. Sortera orden
3. Ta bort alla dubletter ur vektorn
4. Skriv ut orden som finns i vektorn till `std::cout`, separerade med mellanslag.

Alla dessa steg kan göras med STL algoritmer och iteratorer.

1 Upplägg av storseminarium
2 Behållare
3 Algoritmer
4 Modifierande algoritmer
5 Lambdafunktioner
6 Övningsuppgifter

LINKÖPING UNIVERSITY

lambdafunktioner

`std::transform`

```
1  int add_2(int x)
2  {
3    return x + 2;
4  }
5
6  int main()
7  {
8    std::vector<int> u { 1, 2, 3 };
9    std::vector<int> v(u.size());
10
11   std::transform(u.begin(), u.end(), v.begin(), add_2);
12 }
```

lambdafunktioner

lambdafunktioner

```
1  std::vector<int> u { 1, 2, 3 };
2  std::vector<int> v(u.size());
3
4  std::transform(u.begin(), u.end(), v.begin(),
5                 [](int x)
6                 {
7                   return x + 2;
8                 });
```

lambdafunktioner

lambdafunktioner

```
1  std::vector<int> u { 1, 2, 3 };
2  std::vector<int> v(u.size());
3
4  std::transform(u.begin(), u.end(), v.begin(),
5                 [](int x)
6
7                 {
8                   return x + 2;
                  });
```

lambdafunktion

lambdafunktioner

lambdafunktioner

```
1  std::vector<int> u { 1, 2, 3 };
2  std::vector<int> v(u.size());
3
4  std::transform(u.begin(), u.end(), v.begin(),
5                 [](int x)
6                 {
7                   return x + 2;
8                 });
```

omfång (capture)

lambdafunktioner

lambdafunktioner

```
1  std::vector<int> u { 1, 2, 3 };
2  std::vector<int> v(u.size());
3
4  std::transform(u.begin(), u.end(), v.begin(),
5                 [](int x) Parametrar
6                 {
7                   return x + 2;
8                 });
```

lambdafunktioner

lambdafunktioner

```
1  std::vector<int> u { 1, 2, 3 };
2  std::vector<int> v(u.size());
3
4  std::transform(u.begin(), u.end(), v.begin(),
5                 [](int x)
6                 {
7                   return x + 2;
8                 });
```

Funktionskropp

lambdafunktioner

lambdafunktioners omfång

```
1  std::vector<int> v { 1, 2, 3 };
2
3  int n { };
4  std::cin >> n;
5
6  if (n == 1)
7  {
8    // notera att std::transform kan skriva över de gamla värdena
9    // om vi anger början av v som den tredje parametern.
10   std::transform(v.begin(), v.end(), v.begin(),
11                  [](int x) { return x + 1; });
12 }
13 else if (n == 2)
14 {
15   std::transform(v.begin(), v.end(), v.begin(),
16                  [](int x) { return x + 2; });
17 }
18 // ...
```

lambdafunktioner

lambdafunktioners omfång

```
1  std::vector<int> v { 1, 2, 3 };
2
3  int n { };
4  std::cin >> n;
5
6  std::transform(v.begin(), v.end(), v.begin(),
7                 [n](int x) { return x + n; });
```

lambdafunktioner

lambdafunktioners omfång

```
1  std::vector<int> v { 1, 2, 3 };
2
3  int n { };
4  std::cin >> n;
5
6  std::transform(v.begin(), v.end(), v.begin(),
7                 [n](int x) { return x + n; });
```

Lambdafunktioner

Mer om omfång

```
1  int n { 2 };
2  std::vector<int> u { 1, 2, 3 };
3  std::vector<int> v(u.size());
4
5  auto add_n = [n](int x) { return x + n; };
6
7  // resultat = { 3, 4, 5 }
8  std::transform(u.begin(), u.end(), v.begin(), add_n);
9  n = 3;
10
11 // resultat = { 3, 4, 5 } ???
12 std::transform(u.begin(), u.end(), v.begin(), add_n);
```

Lambdafunktioner

Fånga referenser

```
1  int n { 2 };
2  std::vector<int> u { 1, 2, 3 };
3  std::vector<int> v(u.size());
4
5  auto add_n = [&n](int x) { return x + n; };
6
7  // resultat = { 3, 4, 5 }
8  std::transform(u.begin(), u.end(), v.begin(), add_n);
9  n = 3;
10
11 // resultat = { 4, 5, 6 }
12 std::transform(u.begin(), u.end(), v.begin(), add_n);
```

Lambdafunktioner

Fånga referenser

```
1  int n { 2 };
2  std::vector<int> u { 1, 2, 3 };
3  std::vector<int> v(u.size());
4
5  auto add_n = [&n](int x) { return x + n; };
6
7  // resultat = { 3, 4, 5 }
8  std::transform(u.begin(), u.end(), v.begin(), add_n);
9  n = 3;
10
11 // resultat = { 4, 5, 6 }
12 std::transform(u.begin(), u.end(), v.begin(), add_n);
```

Lambdafunktioner

Fånga flera variabler

```
1  int x { };
2  int y { };
3
4  auto my_lambda = [x, &y](int a, int b)
5  {
6    int c { a + b };
7    int z { x + y };
8    return a*x + b*y + c*z;
9  };
```

Lambdafunktioner

Uppgift #4

Skriv en funktion som läser in ett heltal `n` från `std::cin` samt en godtycklig mängd heltal från `std::cin` till en `std::vector`.
Sortera sedan alla heltal i vektorn baserat på deras absoluta avstånd till `n` (använd `std::abs` från `<cmath>`).
Sist ska du skriva ut hela listan till `std::cout`, ett värde per rad.

1 Upplägg av storseminarium
2 Behållare
3 Algoritmer
4 Modifierande algoritmer
5 Lambdafunktioner
6 Övningsuppgifter

LINKÖPING UNIVERSITY

Övningsuppgifter

Uppgift #5

Skriv ett program som läser in strängar från `std::cin` och räknar hur många gånger varje *unik* sträng förekom.
Skriv sedan ut hur många gånger varje unikt ord förekom till `std::cout`. Orden ska skrivas ut i *sorterad* ordning.
Tips: Välj en lämplig behållare för att hålla koll på antalet förekomster.

Övningsuppgifter

Uppgift #6

Läs in heltal från `std::cin` till en vektor.
Kontrollera om alla tal i vektorn är jämna. Skriv ut strängen `"Alla är jämna!"` om alla tal faktiskt var jämna, annars skriver du ut `"Det finns ojämna..."`.
Notera: Detta är en övning i att hitta algoritmer samt att använda lambdafunktioner.

Övningsuppgifter

Uppgift #7

Läs in heltal från `std::cin` till en vektor.
Sortera sedan alla tal i vektorn som ligger mellan det första negativa talet och slutet (i fallande ordning).
Skriv till sist ut alla tal till `std::cout`, ett tal per rad.

www.liu.se

LINKÖPING UNIVERSITY