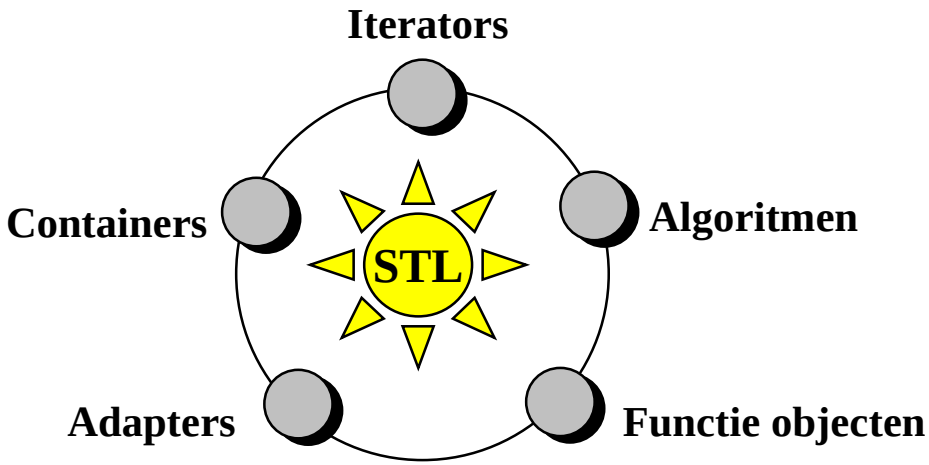


Standard Template Library (STL)

Andy Verkeyn
Vakgroep Informatie Technologie
Universiteit Gent

Versie: November 2000
(Wijzigingen: 11/1998)

Overzicht

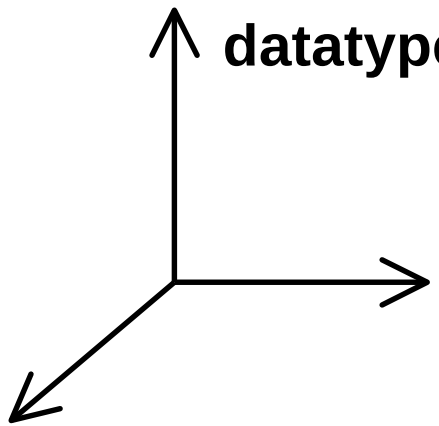
- **Ontwikkeling**
  - HP laboratories
  - Alexander Stepanov
    - **1985: Generieke ADA bibliotheek**
    - **1992: STL**
  - Meng Lee
- **Onderdeel van de ANSI/ISO standaard C++ bibliotheek (aanvaard in 1994)**

Overzicht

- **Bevat 5 soorten componenten**
  - containers
  - iteratoren
  - algoritmen
  - functie objecten (encapsuleren een functie)
  - adapters (veranderen de interface)
- **Heel performant ≅ hand-coded**
  - volledig gebaseerd op templates
    → static linking / type checking
    → code bloat
  - veelvuldig gebruik van inlining

Overzicht

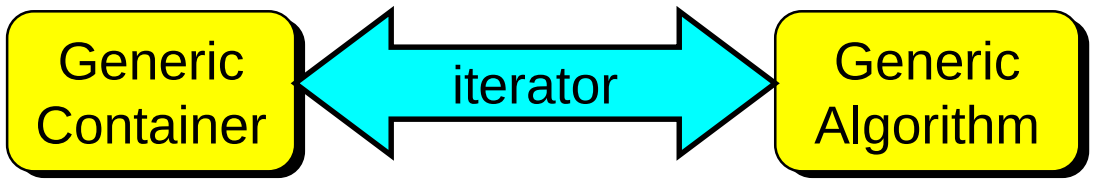
- **Generiek programmeren met orthogonale componenten**



datatype: int, double, Complex,...

container: array, lineaire lijst,...

algoritme:sorteren, zoeken,...



Voorbeeld

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;  // gebruik standaard namespace
void main () {
  vector<int> v;
  int input;
  while(cin >> input)
    v.push_back(input);
  sort(v.begin(), v.end());
  int n = v.size();
  for(int i = 0; i < n; i++)
    cout << v[i] << "\n";
}
```

Voorbeeld - Container

- Container: vector
- Constructors

```
vector<int> v;
// vector met 3 elementen
vector<int> w(3);
// vector met 4 keer getal 3.14
vector<float> u(4, 3.14);
```

Voorbeeld - Container

- **Methodes en operatoren**

```
v.push_back(7);
int waarde = v.pop_back();
int waarde = v.front();
int waarde = v.back();
v[5] = 7;
int aantal = v.size();
bool leeg = v.empty();
w = v;
bool gelijk = (v == w);
v.swap(w);
```

Voorbeeld - Iterator

- **Iteratoren: te gebruiken als pointers**
  - **`v.begin()`**: wijst naar eerste element
  - **`v.end()`** : wijst **voorbij** laatste element (bij conventie)
- **Constructie**

```
vector<int>::iterator i = v.begin();
vector<int>::iterator j = v.end();
```

- **Typisch gebruik**

```
vector<int>::iterator i = v.begin();
while(i != v.end()) {
  cout << *i << endl;
  i++;
}
```

Voorbeeld - Iterator

- **Opgeven van een bereik**
  - Twee iteratoren
  - [i, j[, bv.: [v.begin(), v.end()[
  - **`vector<int> w(v.begin(), v.end());`**
- **De iterator die bij een vector hoort is een random access iterator en laat dus ook ‘pointer arithmetic’ toe.**

```
i = i + n;
i -= n;
i[n];
```

Voorbeeld - Iterator

- **Gebruik van insert/erase**

```
vector<int> v(5, 7);
vector<int> w(3, 13);

// v: 7 7 7 7 7  -->  v: 7 7 7 8 7 7
v.insert(v.begin() + 3, 8);
// voeg v toe vooraan w: 7 7 7 8 7 7 13 13 13
w.insert(w.begin(), v.begin(), v.end());

w.erase(w.end() - 1);         // wis laatste element
w.erase(w.begin() + 3, w.end()); // wis vanaf pos 3
```

- **De `insert` methode voegt een element toe VOOR de opgegeven positie.**

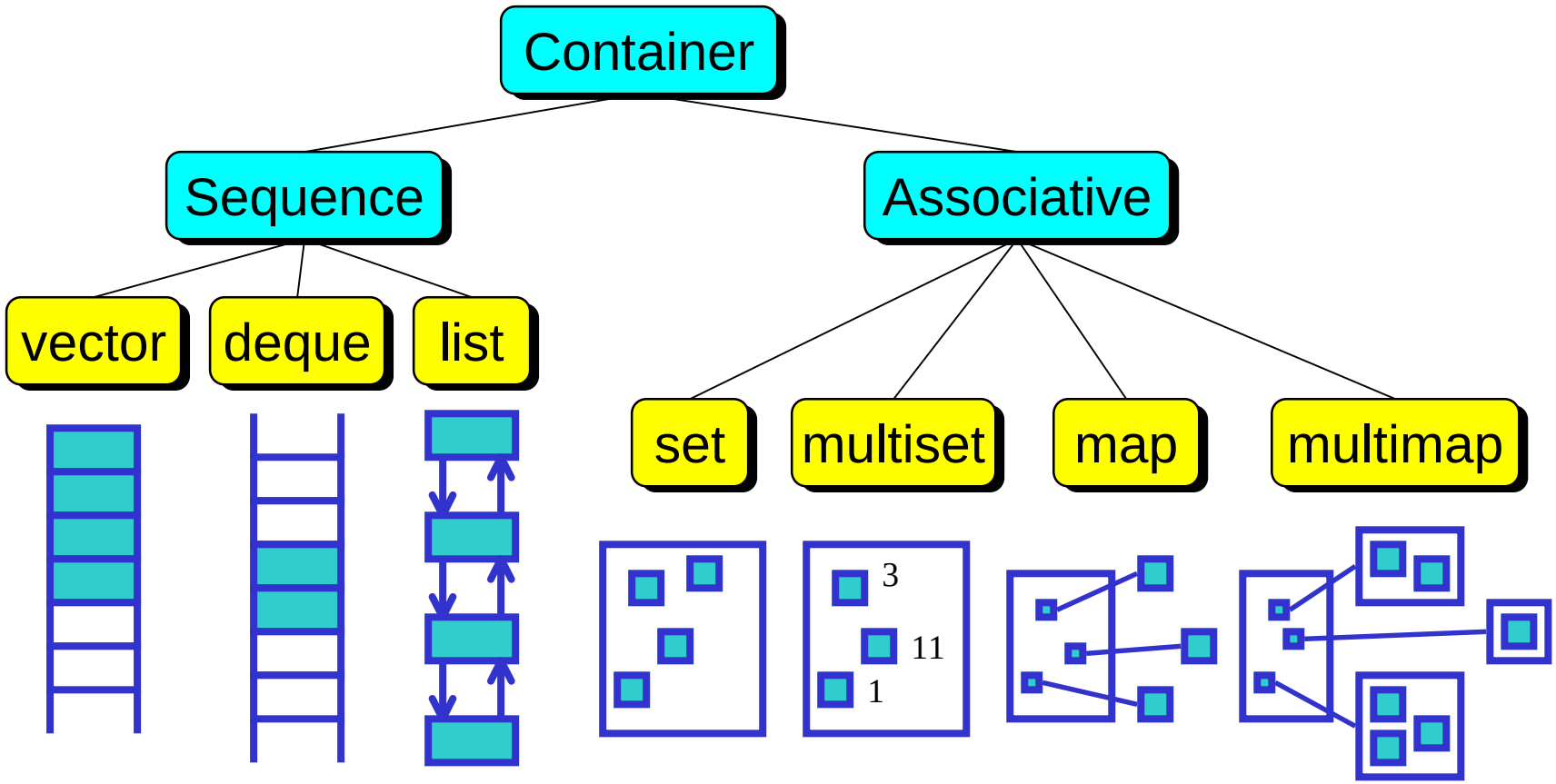
Voorbeeld - Algoritmen

- **Algoritmen**
  - Werken uitsluitend met iteratoren
  - Voorbeelden

```
/* sorteert de elementen in het opgegeven bereik
   (van klein naar groot) */
sort(v.begin(), v.end());

/* geeft een iterator terug die naar het gezocht
   element wijst, end() indien niet gevonden */
binary_search(v.begin(), v.end(), x);
```

Containers

- *"Containers are objects that store other objects"*



Containers

- **Sequentieel**
  - elementen zitten na elkaar
  - **`vector`**: groeit enkel achteraan
  - **`deque`** (Double Ended Queue): kan zowel vooraan als achteraan groeien
  - **`list`**: men kan overal een element tussen voegen

Containers

- **Associatief**
  - elementen worden gesorteerd volgens een sleutel
    $\rightarrow$ snelle toegang
  - implementatie met binaire zoekbomen
    (Red-Black trees)

	duplicaat sleutels	data bij sleutels
set	**Niet toegelaten**	**Neen**
multiset	**Toegelaten**	**Neen**
map	**Niet toegelaten**	**Ja**
multimap	**Toegelaten**	**Ja**

Containers

- **Complexiteit**

	array	vector	deque	list	assoc.
insert/erase begin	na	O(n)	O(1)	O(1)	O(logN)
insert/erase end	na	O(1)	O(1)	O(1)	O(logN)
insert/erase middle	na	O(n)	O(n)	O(1)	O(logN)
access begin	O(1)	O(1)	O(1)	O(1)	O(logN)
access end	O(1)	O(1)	O(1)	O(1)	O(logN)
access middle	O(1)	O(1)	O(1)	O(n)	O(logN)

Containers

- **Type eisen voor T**
  - publieke default constructor
  - publieke copy constructor
  - publieke destructor
  - publieke assignatie operator
- **Alle containers hebben een analoge interface**
  - `size, empty`
  - `begin, end`
  - `insert, erase`
  - `swap`
  - `operator=, operator==`

Containers

- **Alle containers definieren een aantal types**
  - **`value_type`**: type dat de container bevat
  - **`difference_type`**: afstand tussen twee elementen
  - **`iterator`**
  - **`const_iterator`**
  - **`...`**

Containers - Sequentieel

- **Sequentiele containers**
  - `vector<T>`
  - `deque<T>`
  - `list<T>`
- **De gebruiker bepaalt de ordening van de elementen**
- **Bemerk**

  `vector<int>` ≠ `vector<double>`

Containers - Sequentieel

- **Bijkomende methodes vector**

  Toevoegen, verwijderen:

  – push_back, pop_back

  Opvragen:

  – front, back

  – operator[]

- **Bijkomende methodes deque**

  – zie vector

  – push_front, pop_front

Containers - Sequentieel

- **Bijkomende methodes list**
  - zie deque
  - GEEN operator[]
  - enkele typische lijstmanipulaties...

```
// verplaats de elementen [first, last[ van x
// voor de positie pos in de huidige lijst
l.splice(iterator pos, list<T>& x,
         iterator first, iterator last);
// verwijder alle elementen met waarde als inhoud
l.remove(waarde);
// unique, merge, reverse, sort,...
```

Containers - Functie objecten

- ***"Function objects are generalized pointers to functions"***
- **Functie objecten (of "functors")**
  - instanties van klassen die de operator() implementeren.
  - pointers naar C-functies
- **Vaak gebruikt als argument bij algoritmen**
  - om elementen te vergelijken (comparator)
  - als predicaat (resultaat is een logische waarde, **`bool`**)
  - voor bewerkingen

Containers - Functie objecten

- **Standaard functie objecten**

rekenkundig	logisch	vergelijking
plus<T> minus<T> multiplies<T> divides<T> modulus<T> negate<T>	logical_and<T> logical_or<T> logical_not<T>	equal_to<T> not_equal_to<T> greater<T> less<T> greater_equal<T> less_equal<T>

Containers - Functie objecten

- **Voorbeeld**

```
class odd {
  public: bool operator() (int a) { return(a%2==1); }
};
// bool odd(int a) { return(a%2==1); }
void main() {
  vector<int> v;
  // vector v opvullen met int's...
  int n = 0;
  count_if(v.begin(), v.end(), odd(), n);
  // count_if(v.begin(), v.end(), odd, n);
  sort(v.begin(), v.end(), greater<int>());
}
```

Containers - Adapters

- ***"Adapters are template classes that provide interface mapping"***
- **Container adapters**
  - `stack<T, Container<T> >`
  - `queue<T, Container<T> >`
  - `priority_queue<T, Container<T>, Compare<T> >`
- **Methodes**
  - Toevoegen & verwijderen: **`push, pop`**
  - Opvragen:
    - **`top` (enkel stack en priority_queue)**
    - **`front, back` (enkel queue)**

Containers - Adapters

- **Voorbeelden**

```
stack<int, vector<int> > s;
queue<Employee, list<Employee> > q;
priority_queue<int, deque<int>, greater<int> > p;
```

- **queue niet op basis van `vector`**
  (in begin toevoegen is niet efficiënt genoeg)
- **T objecten in een `priority_queue` moeten de gebruikte vergelijkingsoperator implementeren**
- **Bemerk:**

```
stack<int, vector<int> > ≠ stack<int, deque<int> >
```

Containers - Associatief

- **Associatieve containers**
  - `set<Key, Compare<Key> >`
  - `multiset<Key, Compare<Key> >`
  - `map<Key, T, Compare<Key> >`
  - `multimap<Key, T, Compare<Key> >`
- **Worden gesorteerd op de sleutelwaarde**
- **Bijkomende methodes**
  - `find`
  - `count`
  - `lower_bound`
  - `upper_bound`
- **Alle bewerkingen: O(logN)**

Containers - Associatief

- **Als men een standaard functie object gebruikt, moeten de Key objecten zeker de onderliggende vergelijkingsoperator implementeren**

  ```
  set<Employee, greater<Employee> >;
  ```

  greater implementatie...

  ```
  greater(Employee x, Employee y) { return x > y; }
  ```

  ...gebruikt > in Employee en moet dus voorzien zijn!

- **Definitie gelijkheid in functie van vergelijking**
  - Voorbeeld: `k1 == k2 ≡ !comp(k1,k2) && !comp(k2,k1)`
  - Concreet: `k1 == k2 ≡ !(k1 < k2) && !(k2 < k1)`

Containers - pair

- **Hulpklasse:** `pair<T1, T2>`
  - koppel elementen van mogelijks verschillende klasse
  - publieke datamembers: `first, second`
- **map en multimap inhoud:** `pair<Key, T>`

```
m.insert(pair<string,int>("Hallo", 5));
```

- **resultaat van insert in een set:**

```
pair<set::iterator, bool>

cout << *(s.insert(13).first);
if(s.insert(13).second)  cout << "Insert worked";
else                     cout << "Insert failed";
```

Containers - pair

- **Opmerking:**
  - Volgens de C++ standaard moet de **`pair`** klasse een default constructor hebben
  - Dit is echter NIET zo in de RogueWave implementatie van STL (Borland)
  - Oplossing: RWSTD_NO_MEMBER_WO_DEF_CTOR definieren met een preprocessor instructie

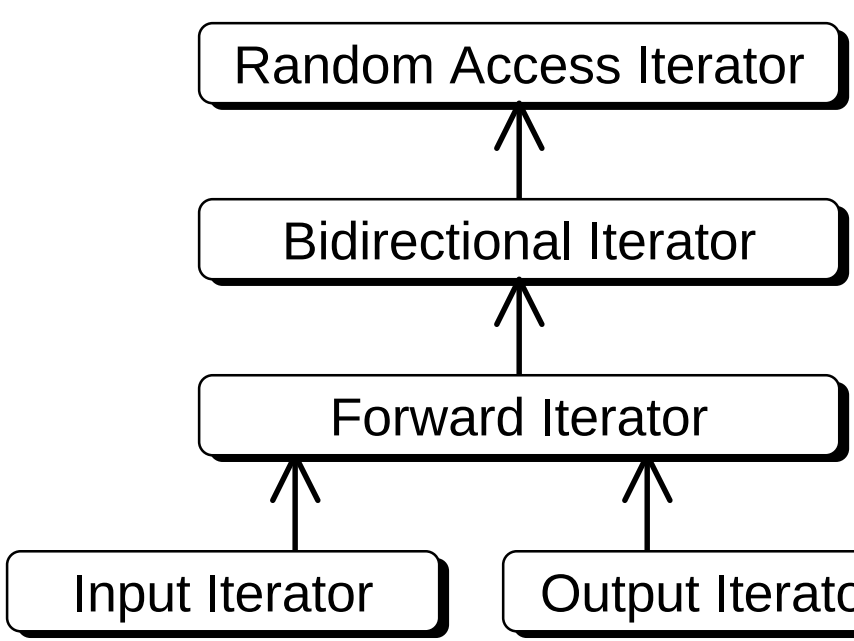
Containers

- **Bijkomende operator map:** `operator[]`
- **Associatieve arrays (dictionary)**
- **Voorbeeld**

```
map<string, int, less<string> > cnt;
pair<string, int> seven(“Hallo”, 7);
cnt.insert(seven);
cnt[“Hallo”] = 13;
cnt[“Hallo”]++;
```

Iteratoren

"Iterators are generalized pointers"




```
i < j; i > j;
i + n; i - n;
i += n; i -= n; i[n];
```

```
i--; --i;
```

```
*i;  i++; ++i;
```

```
i = j;
i == j;  i != j;
```

```
waarde = *i;
i++; ++i;
```

```
*i = waarde;
i++; ++i;
```

incrementeren na elke dereferentie!

Iteratoren

- **Type iterator per container**

array	**random access (pointer)**
vector	**random access**
deque	**random access**
list	**bidirectional**
set, multiset	**bidirectional**
map, multimap	**bidirectional**

Iteratoren

- **Het type is niet expliciet zichtbaar**

  `vector<int>::iterator` → random access iterator

  `list<Employee>::iterator` → bidirectional iterator

  `set<int, less<int> >::iterator` → bidirectional iterator

- **Er is telkens ook een const versie beschikbaar**

```
vector<int>::const_iterator = v.begin();
list<Employee>::const_iterator = l.begin();
set<int, less<int> >::const_iterator = s.begin();
```

- **Gebruik van pointers als een iterator**

```
int c[5] = {50, 40, 30, 20, 10};
sort(c, c + 5, less<int>());
```

Iteratoren

- **Iteratoren van associatieve containers zijn “unassignable” omdat de sleutel niet mag veranderen.**
  - set en multiset
    - **Er maar één iterator implementatie**
    - **`iterator == const_iterator`**
  - map en multimap
    - **Enkel het tweede `pair` element kan men veranderen**

```
map<string, int>::iterator i = m.begin();
(*i).second = 7;        // OK, toegelaten
(*i).first = “Hallo”;   // FOUT, sleutel onwijzigbaar
```

Iteratoren - IO

- **I/O stream iteratoren**
  - Input iterator: `istream_iterator<T, Distance>`
  - Output iterator: `ostream_iterator<T>`
- **`Distance` is de afstand tussen twee elementen**
  - Gedefinieerd in elke container: `difference_type`

```
vector<int>::difference_type
set<Employee, less<Employee> >::difference_type
map<int, string, less<int> >::difference_type
```

Iteratoren - IO

- **Gebruik**

```
typedef vector<int>::difference_type diff_type;
vector<int> v;
istream_iterator<int, diff_type> start(cin),
  end; // EOF: DOS=Ctrl+Z, Unix=Ctrl+D
// invoer van de vector (‘typische’ iteratorlus)
while(start != end) {
  v.push_back(*start);
  start++;
}
// uitvoer van de vector
copy(v.begin(), v.end(),
  ostream_iterator<int>(cout, "\n"));
```

Iteratoren - Adapters

- **Iterator adapters**
  - reverse iteratoren
    - **itereren in de omgekeerde richting**
    - **aanmaken via de container: `rbegin()`, `rend()`**
  - insert iteratoren
    - **assignaties aan “gewone” iteratoren overschrijven elementen**
    - **assignaties aan insert iteratoren voegen elementen toe**
    - **type: output iteratoren**

Iteratoren - Adapters

- **Insert iteratoren**

  `back_insert_iterator<Container>` → **push_back**

  `front_insert_iterator<Container>` → **push_front**

  `insert_iterator<Container>` → **insert**

- **De container moet over de gebruikte methode beschikken**
- **Alternatief om insert iteratoren aan te maken**
  - back_inserter(Container c)
  - front_inserter(Container c)
  - inserter(Container c, Iterator i)

Iteratoren - Adapters

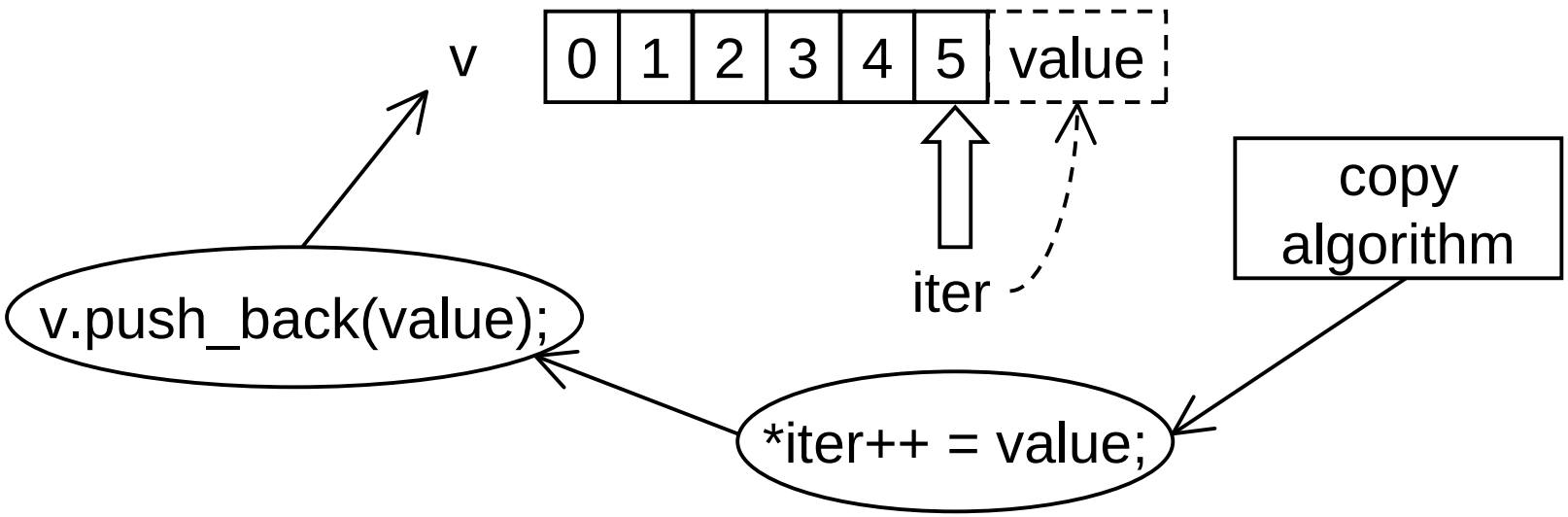
- **Voorbeeld**

```
vector<int> v(3, 7);
back_insert_iterator<vector<int> > bi(v);
// 7 7 7
copy(v.begin(), v.end(),
  ostream_iterator<int> (cout, “\n"));

*bi++ = 10;
*bi++ = 11;

// 7 7 7 10 11
copy(v.begin(), v.end(),
  ostream_iterator<int> (cout, “\n"));
```

Iteratoren - Adapters

- **Typisch gebruik: vermijden van out of bounds**

```
typedef vector<int>::difference_type diff_type;
vector<int> v;
istream_iterator<int, diff_type> start(cin), end;
copy(start, end, back_inserter(v));
```



Iteratoren - Adapters

- **Voorbeeld**

```
typedef vector<int>::difference_type diff_type;

// 0 0 0 0 0 0
vector<int> v(6, 0);
istream_iterator<int, diff_type> start(cin), end;

copy(start, end, inserter(v, v.begin() + 3));
// 0 0 0 1 2 3 4 5 0 0 0
```

Algoritmen

- **Ontwikkeld voor een iterator categorie (en niet voor een bepaalde container)**
- **Varianten**
  - copy
  - if
- **Voorbeeld**

```
replace(first, last, old, new);
replace_if(first, last, pred, new);
replace_copy(first, last, result, old, new);
replace_copy_if(first, last, result, pred, new);
```

Algoritmen

- **Indeling in groepen**
  - non-mutating sequence operations
  - mutating sequence operations
  - sorting operations
  - numeric operations
- **De iterator categorie kan de compiler niet controleren aangezien het template argumenten zijn**
  - parameter naamgeving is doelbewust gekozen

```
OutputIterator copy (InputIterator first,
InputIterator last, OutputIterator result)
```

  → merkwaardige foutmeldingen bij het niet respecteren!

Adapters

- **Soorten adapters**
  - container adapters
  - iterator adapters
  - functor adapters
- **Functor adapters: veranderen een functie object**
  - not1/not2: negatie van unair/binair pred.
  - bind1st/bind2nd: leggen argumenten vast
  - ptr_fun: maakt een object van een functie
- **Voorbeeld**

```
transform(v.begin(), v.end(), v.begin(),
  bind1st(plus<int>(), 7));
```

Implementaties

- **HP (referentie implementatie)**
  - ftp://butler.hpl.hp.com/stl/
- **Silicon Graphics Incorporation**
  - http://www.sgi.com/Technology/STL
- **Rogue Wave Software (Borland, GNU, Sun,...)**
  - http://www.roguewave.com
- **Dinkumware (Microsoft)**
  - http://www.dinkumware.com