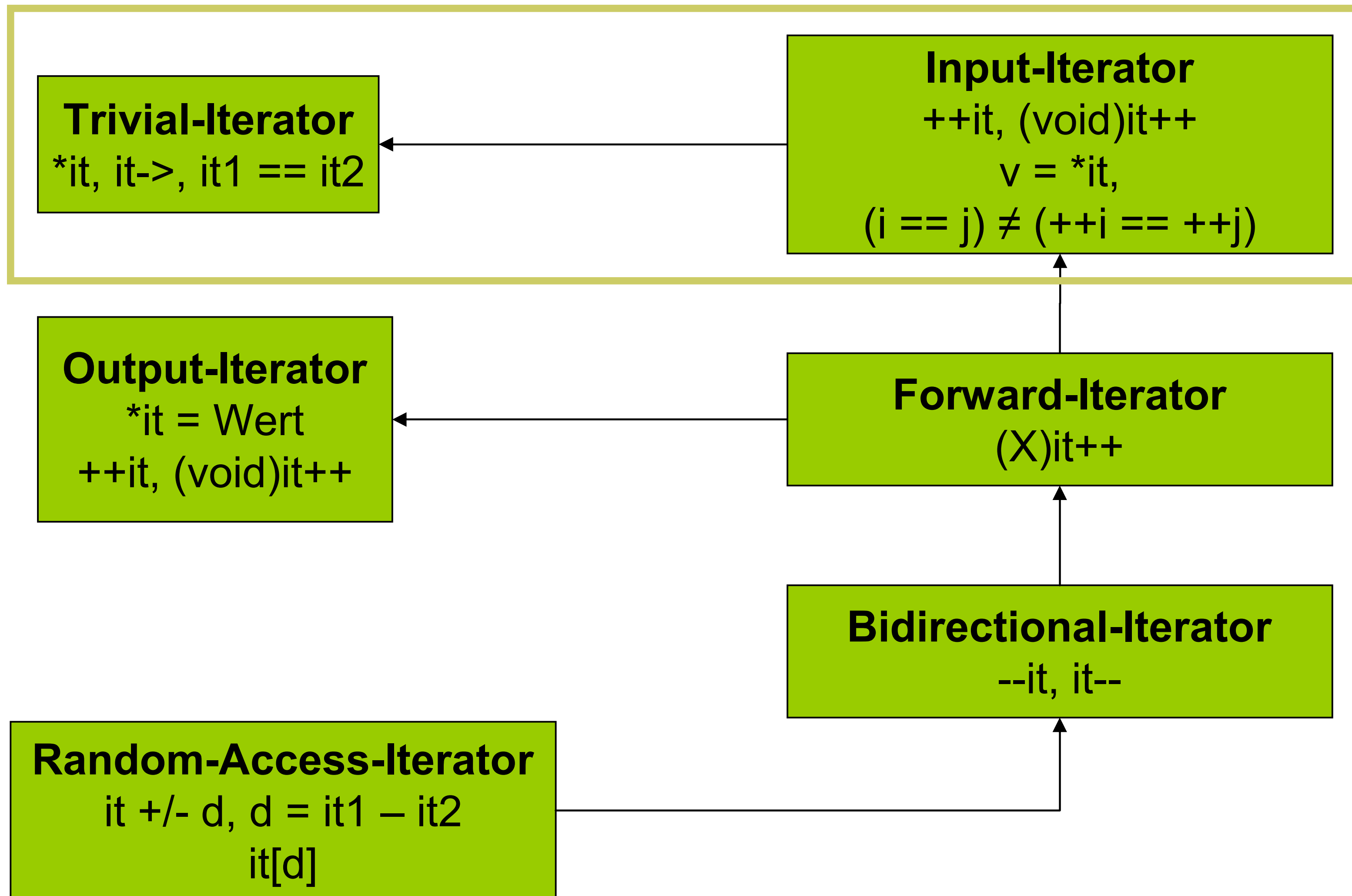
Iteratoren

Ralf Schuchardt

Übersicht

- Konzepte
- const_iterator
- reverse_iterator
- Iterator-Traits
- Iterator-Adapter
  - istreambuf_itererator

Iterator::Konzepte



Iteratoren

- Generalisierung von Zeigern
- durch eine Reihe von Konzepten beschrieben:
  - Input-Iterator
  - Output-Iterator
  - Forward-Iterator
  - Bidirectional-Iterator
  - Random-Access-Iterator

Iterator::Schnittstelle

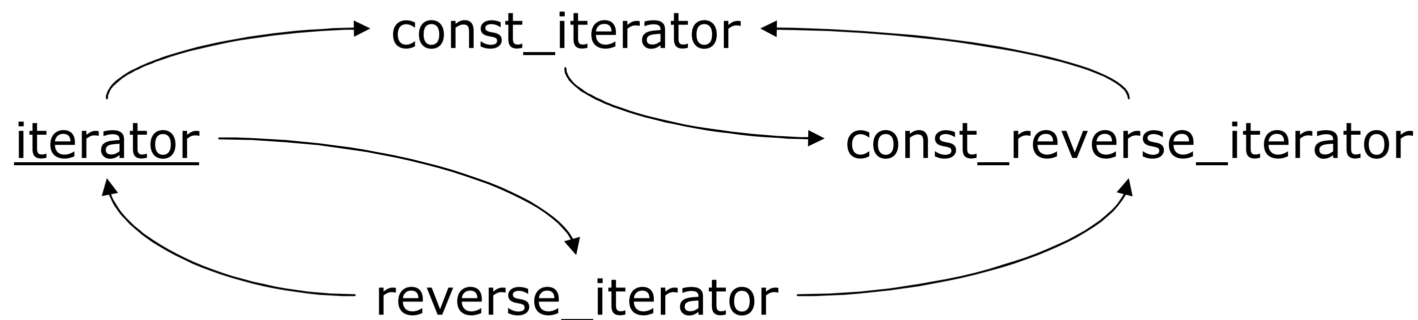
- Schnittstelle zwischen Containern und Algorithmen:
  - Container.begin()
  - Container.end()
  - Container.rbegin()
  - Container.rend()
- *iteratordapter(Container)*
- *stream_iterator(stream)*

Übersicht

- Konzepte
- const_iterator
- reverse_iterator
- Iterator-Traits
- Iterator-Adapter
  - istreambuf_itererator

const_iterator vs. iterator

- Container::const_iterator, Container::const_reverse_iterator



- „insert“ und „erase“ u.a. brauchen oft einen echten „iterator“
- Vergleiche zwischen const_iterator und iterator sind manchmal schwierig

const_iterator vs. iterator (Forts.)

- ```
  typedef std::deque<int> IntDeque;
  typedef IntDeque::iterator It;
  typedef IntDeque::const_iterator ConstIt;
  IntDeque deque; …
  It it = deque.begin();
  ConstIt cit = deque.end();
  ```
- ```
  while (it != cit) std::cout << *it++;
  ```

const_iterator vs. iterator (Forts.)

- $ c++-3.1.exe testconst_a.cc -o testconst_a
  testconst_a.cc: In function `int main()':
  testconst_a.cc:17: no match for `main()::It& != main()::ConstIt&' operator
  /usr/local/lib/gcc-lib/i686-pc-cygwin/3.1/include/g++/bits/stl_deque.h:198: candidates are:
  bool std::_Deque_iterator<_Tp, _Ref, _Ptr>::operator!=(const std::_Deque_iterator<_Tp, _Ref, _Ptr>&) const [with _Tp = int, _Ref = int&, _Ptr = int*]

- ```
  bool operator!=(const Self& x) const
  {return !(*this == x);}
  ```
- ```
  typedef _Deque_iterator<_Tp,_Tp&,_Tp*> iterator;
  typedef _Deque_iterator <_Tp,const _Tp&,const
  _Tp*> const_iterator;
  ```

const_iterator vs. iterator (Forts.)

- ```
  typedef std::deque<int> IntDeque;
  typedef IntDeque::iterator It;
  typedef IntDeque::const_iterator ConstIt;
  IntDeque deque; …
  It it = deque.begin();
  ConstIt cit = deque.end();
  ```
- `while (it != cit) std::cout << *it++; // (a)`
- `while (cit != it) std::cout << *it++; // (b)`
  ```

const_cast<iterator>(const_it_var)

- Holzhammermethode:

```
it = const_cast<iterator>(cit)
```

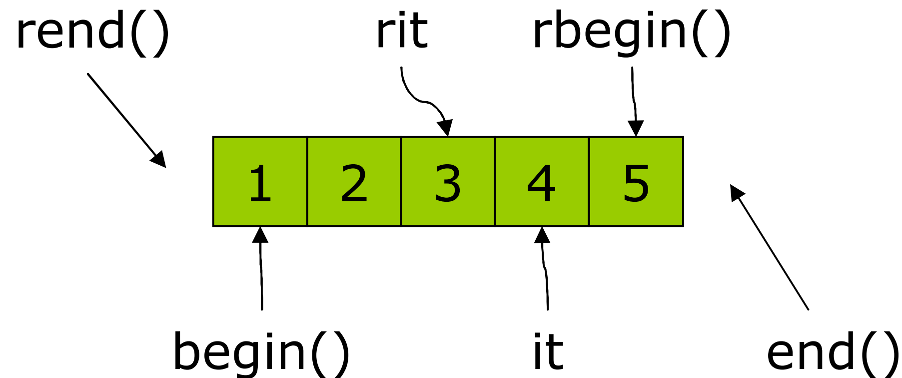
- besser:

```
It it = container.begin();
advance(it, distance<ConstIt>(it, cit));
```

Übersicht

- Konzepte
- const_iterator
- reverse_iterator
- Iterator-Traits
- Iterator-Adapter
  - istreambuf_itererator

begin(), end() und rbegin(), rend()



- cont::reverse_iterator rit = cont::reverse_iterator(it);
- It it = rit.base();

- &*(reverse_iterator(it)) == &*(it - 1)
- klar: It muss mindestens bidirektionaler Iterator sein

Eigenschaften von reverse_iterator

- Einfügen ✓
  - nur über rit.base()
  - vor dem Element, auf das gezeigt wird
  - rit: 5 4 ^ [3] 2 1
  - rit.base(): 1 2 3 ^ [4] 5
- Löschen ✗
  - nur über rit.base()
  - das Element, auf das gezeigt wird
  - --rit.base()
  - (++rit).base(), ✓

Falle!

```
typedef std::vector<int> IntVector;
typedef IntVector::iterator It;
IntVector myvec;
myvec.push_back(…); …
if (!myvec.empty())
  for (It it = ++myvec.begin();
       it != myvec.end();
       ++it) do_something(it);
```

Falle! (Forts.)

- $ c++-3.1 test.cc -o test310

- $ c++ test.cc -o test295
  test.cc: In function `int main()':
  test.cc:6: non-lvalue in increment

„They ain't pointers!“

- ```
  template
  <class _Tp, class _Alloc = allocator<_Tp> >
  class vector :
       protected _Vector_base <_Tp, _Alloc>
  {
   private:
      typedef vector<_Tp, _Alloc> vector_type;
   public:
       typedef
       __gnu_cxx::__normal_iterator <pointer,
                              vector_type> iterator;
  …
  };
  ```
  ```

„They ain't pointers!“ (but may be)

- ```
  template <class _Tp, class _Alloc =
  __STL_DEFAULT_ALLOCATOR(_Tp) >
  class vector
    : protected _Vector_base<_Tp, _Alloc>
  {
      public:
              typedef value_type* pointer;
              typedef value_type* iterator;
  …
  }
  ```
- `It it = ++myvec.begin();`
- `It it = (myvec.begin() + 1);`
  ```

Übersicht

- Konzepte
- const_iterator
- reverse_iterator
- Iterator-Traits
- Iterator-Adapter
  - istreambuf_itererator

advance, distance

- `template <class InputIterator, class Distance>`
  `void advance(InputIterator& i, Distance n);`
- bewegt i um n Elemente vor oder zurück (bei bidirektionalen Iteratoren)
- `template<class InputIterator>`
  `iterator_traits<InputIterator>::difference_type`
  `distance(InputIterator first, InputIterator last);`
- liefert die Anzahl der Elemente zwischen first und last
- last muss von first aus erreichbar sein
  - bei nicht Random-Access-Iteratoren heißt dies meistens, dass first vor last liegen muss
  - bei Random-Access-Iteratoren muss dies evtl. nicht gelten

Iterator_traits::Motivation

- Zeiger und Objekte als Iteratoren
- Benutzung nach Funktionalität ermöglichen
  - Konzepte

Iterator_traits

```
template <class Iterator>
 struct iterator_traits {
   typedef typename Iterator::iterator_category
                                                   iterator_category;
   typedef typename Iterator::value_type            value_type;
   typedef typename Iterator::difference_type       difference_type;
   typedef typename Iterator::pointer               pointer;
   typedef typename Iterator::reference             reference;
 };

template <class T>
 struct iterator_traits<T*> {
   typedef random_access_iterator_tag iterator_category;
   typedef T                          value_type;
   typedef ptrdiff_t                  difference_type;
   typedef T*                         pointer;
   typedef T&                         reference;
 };
```

Iterator::Tags

- `struct input_iterator_tag {};`
- `struct output_iterator_tag {};`
- ```
  struct forward_iterator_tag :
        public input_iterator_tag {};
  ```
- ```
  struct bidirectional_iterator_tag :
        public forward_iterator_tag {};
  ```
- ```
  struct random_access_iterator_tag :
        public bidirectional_iterator_tag {};
  ```
  ```

Iterator::iterator_category

- ```
  inline output_iterator_tag iterator_category
                                       (const output_iterator&);
  ```
- ```
  template <class T, class Distance>
  inline input_iterator_tag
  iterator_category(const input_iterator<T, Distance>&);
  ```
- ```
  template <class T, class Distance>
  inline forward_iterator_tag
  iterator_category(const forward_iterator<T, Distance>&);
  ```
- ```
  template <class T>
  inline random_access_iterator_tag
  iterator_category(const T*);
  ```

Iterator::value_type, distance_type

- ```
  template <class T, class Distance>
  inline T* value_type(const forward_iterator<T, Distance>&);
  ```
- ```
  template <class T, class Distance>
  inline T* value_type(const random_access_iterator<T,
  Distance>&);
  ```
- ```
  template <class T> inline T* value_type(const T*);
  ```
- ```
  template <class T, class Distance>
  inline Distance* distance_type(const forward_iterator<T,
  Distance>&);
  ```
- ```
  template <class T, class Distance>
  inline Distance* distance_type(const random_access_iterator<T,
  Distance>&);
  ```
- ```
  template <class T> inline ptrdiff_t* distance_type(const T*);
  ```

Iterator::distance()

- ```
  template <typename _InputIterator>
  inline typename
  iterator_traits<_InputIterator>::difference_type
  distance (_InputIterator __first,
            _InputIterator __last)
  {
    return
      __distance(__first, __last,
      iterator_traits<_first>::iterator_category());
  }
  ```
  ```

Iterator::distance() (Forts.)

- ```
  template<typename _InputIterator> inline typename
  iterator_traits<_InputIterator>::difference_type
  __distance(_InputIterator __first,
             _InputIterator __last, input_iterator_tag)
  {
      typename iterator_traits <_InputIterator> ::
                                        difference_type
  __n = 0;
      while (__first != __last) {
        ++__first; ++__n;
      }
      return __n;
  }
  ```
- ```
  template<typename _RandomAccessIterator> inline typename
  iterator_traits<_RandomAccessIterator>::difference_type
  __distance(_RandomAccessIterator __first,
             _RandomAccessIterator __last,
             random_access_iterator_tag)
  {return __last - __first;}
  ```

Übersicht

- Konzepte
- const_iterator
- reverse_iterator
- Iterator-Traits
- Iterator-Adapter
  - istreambuf_itererator

Iterator::Adapter

- Reverse-Iterator
- Insert-Iterator
  - output-Iterator
  - back_insert_iterator
    - back_inserter(container)
  - front_insert_iterator
    - front_inserter(container)
  - insert_iterator
    - inserter(container, position)
    - auf Gültigkeit des Iterators achten! (vector, deque)

Iterator::front_/back_inserter

- ```
  template <class Container>
  class back_insert_iterator : public iterator …
  {
    explicit back_insert_iterator(Container& x);
   …
  }
  ```
- zur Vereinfachung: back_inserter(container)
- (it = value) == container.push_back(value);

- ```
  template <class Container>
  class back_insert_iterator : public iterator …
  {
    explicit front_insert_iterator(Container& x);
   …
  }
  ```
- zur Vereinfachung: front_inserter(container)
- nur bei Containern mit push_front() verwendbar
- (it = value) == container.push_front(value)

Iterator::inserter

- ```
  template <class Container>
  class insert_iterator : public iterator …
  {
    explicit insert_iterator(Container& x,
                      typename Container::iterator iter);
   …
  }
  ```
- zur Vereinfachung: inserter(container, iter)
- (it = value) ==
  (iter = container.insert(it, value);
  ++iter)
  ```

front_insert_iterator::Beispiel

```
using std::deque;
deque<int> i;
front_insert_iterator< deque<int> > fit =
front_inserter(i);
for (int i = 1; i < 10; ++i)
{
    fit = i;
}
for (deque<int>::reverse_iterator it = i.rbegin();
     it != i.rend();
     ++it)
{
    std::cout << *it;
}
```

Stream-Iteratoren (Ergänzung)

- istream_iterator it(istream)
- ostream_iterator ot(ostream)

- istreambuf_iterator it(istream)
- ostreambuf_iterator ot(ostream)
  - arbeiten direkt mit Stream-Puffer
  - unformatiert, Leerzeichen werden nicht übersprungen
  - sinnvoll wenn zeichenweise gearbeitet wird

Vergleich

```
time_t t1 = time(0);
for (int i = 1; i <= 10; ++i) {
    ifstream xin("bigfile");
    istreambuf_iterator<char> ist(xin);
    istreambuf_iterator<char> end;
    istream_iterator<char> ist(xin);
    istream_iterator<char> end;
    while (ist != end) {
        char c = *ist++;
        if (c != 127) cerr << "Fehler!" << endl;
    }
    cout << "buffer read #" << i << endl;
}
time_t t2 = time(0);
```

Vergleich

- $ ./vergleich
- file created
- buffer read #1
- buffer read #2
- buffer read #3
- buffer read #4
- buffer read #5
- buffer read #6
- buffer read #7
- buffer read #8
- buffer read #9
- buffer read #10

- try next
- read #1
- read #2
- read #3
- read #4
- read #5
- read #6
- read #7
- read #8
- read #9
- read #10
- Streambuf: 64
- Stream: 273

Literatur

- GCC 3.1
- Nicolai Josuttis:
  Die C++ Standardbibliothek.
  Addison Wesley
- Scott Meyers: Effective STL.
  Addison Wesley
- http://www.sgi.com/tech/stl/
- Bjarne Stroustrup:
  Die C++ Programmiersprache.
  Addison Wesley