Standard Template Library

Container, Iteratoren, Algorithmen

Name *Sebastian Sontag*

Datum *31.05.2019*
Vorlesung *Datenanalyse mit dem C++ Toolkit ROOT*

Einführung

- *C++* Library
- Templates → Generische Programmierung
- Benutzung der Funktionen mit verschiedenen Datentypen
- Standardisierung → gute Wiederverwendbarkeit (Library)

```
template<typename T>
T_max(T a, T b)
{
  if (a < b)
    return b;
  else
    return a;
}
```

Standard Template Library

Container

- Realisierung als *Template*
- Nimmt Elemente eines beliebigen Datentyps auf
- Container können *Klassen* enthalten
- Sequenzielle Container
  - *<vector>, <deque>, <queue>, <stack>, <list>*
- Assoziative Container
  - *<map>, <multimap>, <set>, <multiset>*

Container

https://de.cppreference.com/w/cpp/container

- functions present in C++03
- functions present since C++11

		Sequence containers					Associative containers					
Headers		<array>	<vector>	<deque>	<forward_list>	<list>	<set>		<map>		<unorder	
		array	vector	deque	forward_list	list	set	multiset	map	multimap	unordered_set	un
	(constructor)	(implicit)	vector	deque	forward_list	list	set	multiset	map	multimap	unordered_set	un
	(destructor)	(implicit)	~vector	~deque	~forward_list	~list	~set	~multiset	~map	~multimap	~unordered_set	~u
	operator=	(implicit)	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	
	assign	N/A	assign	assign	assign	assign	N/A	N/A	N/A	N/A	N/A	
Iterators	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin	
	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	
	end	end	end	end	end	end	end	end	end	end	end	
	cend	cend	cend	cend	cend	cend	cend	cend	cend	cend	cend	
	rbegin	rbegin	rbegin	rbegin	N/A	rbegin	rbegin	rbegin	rbegin	rbegin	N/A	
	crbegin	crbegin	crbegin	crbegin		crbegin	crbegin	crbegin	crbegin	crbegin		
	rend	rend	rend	rend	N/A	rend	rend	rend	rend	rend	N/A	
	crend	crend	crend	crend		crend	crend	crend	crend	crend		
Element access	at	at	at	at	N/A	N/A	N/A	N/A	at	N/A	N/A	
	operator[]	operator[]	operator[]	operator[]	N/A	N/A	N/A	N/A	operator[]	N/A	N/A	
	front	front	front	front	front	front	N/A	N/A	N/A	N/A	N/A	
	back	back	back	back	N/A	back	N/A	N/A	N/A	N/A	N/A	
Capacity	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	
	size	size	size	size	N/A	size	size	size	size	size	size	
	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	
	resize	N/A	resize	resize	resize	resize	N/A	N/A	N/A	N/A	N/A	
	capacity	N/A	capacity	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	
	reserve	N/A	reserve	N/A	N/A	N/A	N/A	N/A	N/A	N/A	reserve	
	shrink_to_fit	N/A	shrink_to_fit	shrink_to_fit	N/A	N/A	N/A	N/A	N/A	N/A	N/A	
Modifiers	clear	N/A	clear	clear	clear	clear	clear	clear	clear	clear	clear	
	insert	N/A	insert	insert	insert_after	insert	insert	insert	insert	insert	insert	
	emplace	N/A	emplace	emplace	emplace_after	emplace	emplace	emplace	emplace	emplace	emplace	
	emplace_hint	N/A	N/A	N/A	N/A	N/A	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	
	erase	N/A	erase	erase	erase_after	erase	erase	erase	erase	erase	erase	
	push_front	N/A	N/A	push_front	push_front	push_front	N/A	N/A	N/A	N/A	N/A	
	emplace_front	N/A	N/A	emplace_front	emplace_front	emplace_front	N/A	N/A	N/A	N/A	N/A	
	pop_front	N/A	N/A	pop_front	pop_front	pop_front	N/A	N/A	N/A	N/A	N/A	
	push_back	N/A	push_back	push_back	N/A	push_back	N/A	N/A	N/A	N/A	N/A	
	emplace_back	N/A	emplace_back	emplace_back	N/A	emplace_back	N/A	N/A	N/A	N/A	N/A	
	pop_back	N/A	pop_back	pop_back	N/A	pop_back	N/A	N/A	N/A	N/A	N/A	
	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	
	merge	N/A	N/A	N/A	N/A	merge	N/A	N/A	N/A	N/A	N/A	

std::vector

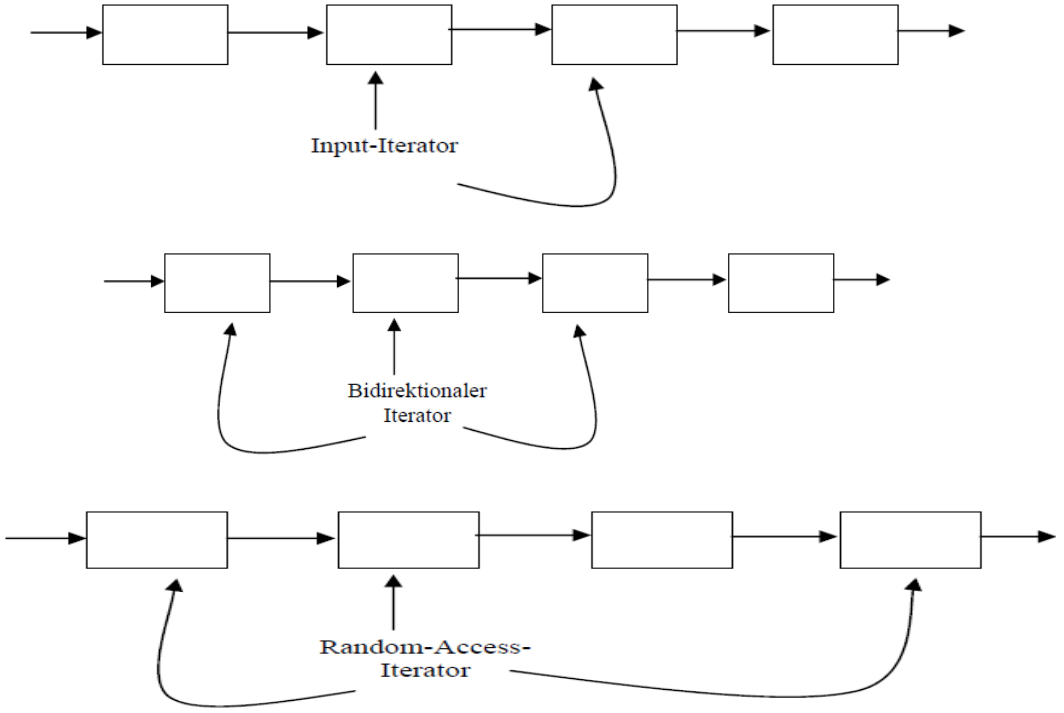
```cpp
#include <vector>
#include <iostream>

using namespace std;

int main(){

    vector<int> vec_1;              // Vektor definieren
    vector<int> vec_2(5);           // Vektor mit 5 Elementen
    vector<int> vec_3(5, 10);       // 5 Elemente mit Wert 10

    cout << "Größe des Containers: " << vec_3.size() << endl;
    cout << "Maximalgröße: " << vec_3.capacity() << endl;

    vec_2[2] = 10;                  // Weise dem 3. Element des Vektors den Wert 10 zu
    vec_2.push_back(5);             // Fügt ein Element am Ende hinzu

    vec_2.insert(vec_2.begin(), 6); // Dem ersten Element wird der Wert 6 zugeordnet
    vec_2.insert(vec_2.end(), 4);   // Dem letzten Element + 1 wird der Wert 4 zugeordnet

    for (int index = 0; index < vec_2.size(); index++){
        cout << vec_2[index] << endl;
    }
}

```

std::stack

- *empty* Testet ob der Container leer ist
- *size* Gibt die Größe zurück
- *top* Zugriff auf das nächste Element
- *push* Fügt Element „oben“ hinzu
- *pop* Löscht das „oberste“ Element

Iteratoren

- Ähnlich wie Pointer → Zeiger auf Elemente eines Containers
- Input-, Output- und Forward-Iteratoren
- Bidirektionale Iteratoren
- Random-Access-Iteratoren



Iteratoren

• **iter*	Aktuelles Element lesen
• **iter = wert*	Wert schreiben
• *iter1 = iter2*	Zuweisung
• *++iter*	Vorwärts
• *--iter*	Rückwärts
• *iter[n]*	Zugriff auf Element n
• *iter1 == iter2*	Vergleich
• *iter1 < iter 2*	Vergleich

Iteratoren

```cpp
#include <vector>
#include <iostream>

using namespace std;

int main(){

    vector<int> vec(5);          // Definition eines Vektors mit 5 Elementen

    vector<int>::iterator ptr;   // Definition eines allgemeinen Zeigers auf einen Container

    ptr = vec.begin() + 2;       // Zuweisung des Zeigers auf 3. Elementen

    vec.insert(ptr, 6);          // Wert 6 an Stelle 3

    for (ptr = vec.begin(); ptr < vec.end(); ptr++){
        cout << *ptr << endl;
    }
}
```

Iteratoren

```
1   // Konstante Iteratoren
2
3   template <typename TDATA>
4
5   void Ausgabe(const TDATA &container){
6
7       typename TDATA::const_iterator ptrIter;
8
9       for (ptrIter = container.begin(); ptrIter < container.end(); ++ptrIter){
10          cout << *ptrIter << endl;
11      }
12  }
```

Iteratoren

```
 1  #include <iostream>
 2  #include <iomanip>
 3  #include <set>
 4  #include <iterator>
 5  #include <string>
 6
 7  using namespace std;
 8
 9  int main(){
10      const int Anzahl = 3;
11      const char* Obst[Anzahl] = {"Apfel", "Erdbeere", "Mango"};
12
13      set<string> setObst (Obst, Obst + Anzahl);
14      cout << "Was ist an Obst vorhanden?" << endl;
15      copy(setObst.begin(), setObst.end(), ostream_iterator<string>(cout, " "));
16      cout << endl;
17  }
18
```

Algorithmen

- Bearbeitung der Elemente eines Containers
- Nicht-modifizierende Algorithmen
  - Verändern die Werte der Elemente oder deren Reihenfolge nicht
  - Können auf alle Standard-Container angewandt werden
  - *count(), count_if(), equal(), find(), find_if(), find_end(), for_each(), max_element(), min_element(), mismatch(), search(), search_n()*
- Modifizierende Algorithmen
  - Verändern die Werte von Elementen
  - Können nicht auf assoziative Container angewandt werden
  - *copy(), copy_backward(), for_each(), generate(), generate_n(), merge(), replace(), replace_if(), replace_copy_if(), fill(), transform()*

Algorithmen

- Löschende Algorithmen
  - Entfernen Elemente aus einem Container
  - Überschreiben die zu löschenden Elemente mit den nachfolgenden Elementen
  - Verkleinern den Speicherplatz des Containers nicht
  - Können nicht auf assoziative Container angewandt werden
  - *remove(), remove_if(), remove_copy_if(), unique(), unique_copy()*

- Mutierende Algorithmen
  - Ändern die Reihenfolge der Elemente in einem Container
  - Sollten auf sortierte Elemente angewandt werden
  - Können nicht auf assoziative Container angewandt werden
  - *next_permutation(), prev_permutation(), partition(), stable_partition(), random_shuffle(), reverse(), reverse_copy(), rotate(), rotate_copy()*

Algorithmen

- Erstellung eines Heaps
  - *make_heap(), pop_heap(), push_heap(), sort_heap()*
- Sortieren von Elementen
  - *partial_sort(), partial_sort_copy(), partition(), stable_partition(), sort(), stable_sort()*
- Anwendung auf sortierte Bereiche
  - *binary_search(), includes(), lower_bound(), upper_bound(), equal_range(), merge(), set_union(), set_intersection(), set_difference(), implace_merge()*
- Berechnung von Werten
  - *accumulate(), adjacent_difference(), inner_product(), partial_sum()*

Implementierung der Algorithmen

```
1  template<class InputIterator, class Function>
2    Function for_each(InputIterator first, InputIterator last, Function fn)
3  {
4    while (first!=last) {
5      fn (*first);
6      ++first;
7    }
8    return fn;
9  }
```

Aufgabe

- Zahlenfolge in std::*vector* schreiben (0,8,5,5,1,4,6,2,2,0,0,9,5,0,1,1,1,8)
- *size()* → Größe bestimmen
- *count()* → Anzahl an 5er bestimmen
- *max_element()* → Größtes Element bestimmen
- *replace()* → 8 mit 3 ersetzen
- *remove()/remove_if()* → 0 entfernen
- *unique()* → Mehrfachaufzählung entfernen
- *rotate()* → zwischen beginn() + 2 und end() - 2 um beginn() + 4 rotieren
- *std::stack* → *push()* → Stack mit Werten füllen
- *top()* & *pop()* → Stack auslesen und ausgeben →Lösung?

Vielen Dank für eure Aufmerksamkeit

Quellen

- http://www.cplusplus.com/reference/algorithm/copy/
- http://www.cplusplus.com/reference/stack/stack/
- https://de.wikipedia.org/wiki/Template_(C%2B%2B)
- https://de.cppreference.com/w/cpp/header
- https://de.wikipedia.org/wiki/C%2B%2B-Standardbibliothek
- http://www.cplusplus.com/reference/vector/vector/
- https://de.wikipedia.org/wiki/Iterator
- https://www.youtube.com/watch?v=gxZJ5JNuWMY