Die Standard Template Library

- Die Container-Klasse Set -

Jörg Lange

31.05.2002

Übersicht

- Einordnung & Eigenschaften
- Details
- Operationen
- Hinweise & Beispiele
- Zusammenfassung
- Quellen

Einordnung & Eigenschaften

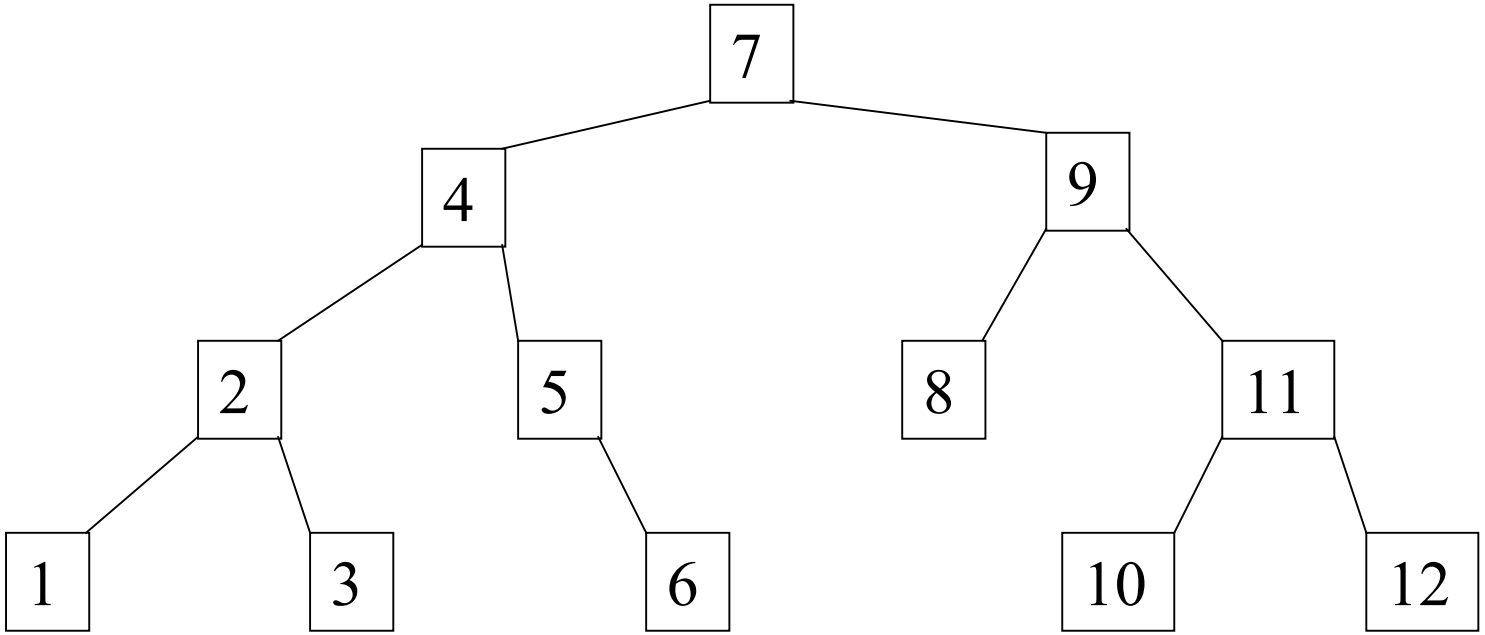
- Set gehört zu den ***Assoziativen Containern***
- Eigenschaften ***Assoziativer Container*** :
  - Elemente werden automatisch sortiert
  - beliebiger Elementtyp, muss sortierbar sein
  - Sortierkriterium ist die ***Äquivalenz***
  - kein Direktzugriff auf Elemente

Einordnung & Eigenschaften

- Eigenschaften ***Assoziativer Container*** :
  - typische Implementation : ***balancierte Binärbäume***
  - Wertänderung → entfernen und neu einfügen des Elements
  - spezielle Set – Eigenschaft:
    - keine Duplikate erlaubt

Einordnung & Eigenschaften

- Interner Aufbau eines Sets



Details

- Definition in Header-Datei <set> im namespace std

```
namespace std {
        template <class T,
                  class Compare = less<T>,
                  class Allocator = allocator>
        class set;
}
```

Set – Operationen

Erzeugen, Kopieren, Zerstören

Ausdruck	Bedeutung
set<Elem> m	erzeugt leeres Set
set<Elem, op>	erzeugt leeres Set, das anhand op sortiert
set<Elem> m1(m2)	erzeugt Set als Kopie eines andereren
set<Elem> m(anf, end)	erzeugt Set und initialisiert es mit Kopien der Elemente des Bereichs [anf, end)
m.~set<Elem>	löscht alle Elemente und gibt Speicherplatz frei

Set – Operationen

Erzeugen, Kopieren, Zerstören

- falls kein Sortierkriterium vorgegeben wird, wird mit

  < sortiert (less<Elem> und weitere definiert in *function.h*)

- Verwendung von Bereichskopien erfolgt meist, wenn set mit Elementen anderer Container initialisiert wird

Set – Operationen

Nicht-Manipulierende Funktionen

Ausdruck	Bedeutung
size_type size( )	liefert aktuelle Anzahl von Elementen
bool empty( )	liefert, ob Container leer ist
size_type max_size( )	liefert die maximal mögliche Elementanzahl
m1 == m2	liefert, ob m1 gleich m2 ist
m1 != m2	liefert,ob m1 ungleich m2 ist
m1 [< | >] m2	liefert, ob m1 kleiner / größer m2 ist
m1 [<= | >=] m2	liefert, ob m1 kleiner/größer oder gleich m2 ist

Set – Operationen

Nicht-Manipulierende Funktionen

- Vergleiche nur zwischen sets mit gleichem Element-Typ
- für Vergleiche mit Containern anderen Typs, dienen vergleichende Algorithmen:
  - bool equal (...)
  - bool lexicographical_compare(...)

Set – Operationen

Spezielle Such - Funktionen

Ausdruck	Bedeutung
size_type count(elem)	liefert Anzahl der Elemente mit Wert elem
iterator find(elem)	liefert Position vom 1.Element elem oder end( )
iterator lower_bound(elem)	liefert erste Position, an der elem eingefügt werden könnte (erstes Element >= elem)
iterator upper_bound(elem)	liefert letzte Position, an der elem eingefügt werden könnte (erstes Element > elem)
pair<iterator, iterator> equal_range(elem)	liefert erste und letzte Position, an der elem eingefügt werden könnte (Bereich mit Elementen == elem)

Set – Operationen

Spezielle Such - Funktionen

- Zugriff auf equal_range Positionen mit

  equal_range(elem).first bzw. equal_range(elem).second

Set – Operationen

Zuweisungen

Ausdruck	Bedeutung
m1 = m2	weist m1 alle Elemente von m2 zu
void swap( )	vertauscht die Elemente von zwei Sets
void swap(m1, m2)	Vertauschung als globale Funktion

Set – Operationen

Iterator Funktionen

- iterator begin( ), iterator end( )
- reverse_iterator rbegin( ), reverse_iterator rend( )
- Elemente werden als Konstante betrachtet
- Algorithmen mit Elementzuweisung funktionieren nicht

  (betrifft modifizierende Algorithmen, z.B. remove( ))

Set – Operationen

Einfügen & Löschen

Ausdruck	Bedeutung
pair<iterator, bool> insert(elem)	fügt Kopie von elem ein und liefert Position des neuen Elements bzw., ob es geklappt hat
iterator insert(pos, elem)	wie oben (pos als Hinweis auf Elementposition)
void insert(anf, end)	fügt Kopien der Elemente des Bereichs [anf, end) ein
size_type erase(elem)	löscht (alle) Element(e) mit Wert elem
void erase(pos)	löscht das Element an Position pos
void erase(anf.end)	löscht alle Elemente in Bereich [anf, end)
void clear( )	löscht alle Elemente (leerer Container)

Set – Operationen

Einfügen & Löschen

- insert(elem) liefert ein Wertepaar vom Typ pair zurück
- pair besteht aus Position und boolschem Wert
- Komponente first gibt Position des neuen Elementes an (bei Erfolg) oder Position des vorhandenen Elements
- Komponente second gibt ob Element eingefügt wurde (also noch nicht vorhanden war)

Hinweise & Beispiele

- Hinweis:
  - ***Assoziatve Container*** basieren bzgl. der Sortierung auf Äquivalenz (Gleichartigkeit) und nicht auf Gleichheit, d.h.
  - zwei Elemente x und y sind gleich (basierend auf <), wenn gilt:

    !(x < y) && !(y < x)
  - nicht - member Funktionen basieren oft auf Gleichheit

Hinweise & Beispiele

- Beispiel 1:
- nicht-case-sensitives set<string> ciss
- Vergleichsfunktion ignoriert Unterscheidung von Groß-/Klein-buchstaben

```
ciss.insert("Seminar");          // neues Element hinzugefügt
ciss.insert("seminar");          // nicht hinzugefügt
```

- Suche nach "seminar" mit ciss.find() und find( )

```
if(ciss..find("seminar")!=ciss.end()) ...                    //  OK
if(find(ciss.begin( ),ciss.end( ),"seminar") != ciss.end( )) .// Falsch
```

Hinweise & Beispiele

- Beispiel 2 :
- man nehme die Vergleichsfunktion **"less_equal<int>"**
- sei **set<int,less_equal<int> > m;** ein int - set mit dieser Funktion
- Einfügen des Elementes 10 (zweimal)

  m.insert(10);

  m.insert(10);
- Resultat: 10 ist zweimal im Set enthalten !!! → nach Definition ist m jetzt nicht mehr ein Set !!!
- → Vergleichsfunktionen zum Sortieren Assoziativer Container müssen für gleiche Werte stets **false** zurückliefern

Hinweise & Beispiele

- Beispiel 3:
- man möchte in einem set Pointer speichern

  set<int*> pset;
- Ausgabeverhalten

  copy(pset.begin( ), pset.end( ), ostream_iterator<int>(cout, " "));

  kompiliert nicht ! Problem: **int*** <==> **int**
- Ausgabe mit doppelter Dereferenzierung :

  for(set<int*>::const_iterator i = m.begin(); i != m.end( ), ++i)

  cout << **i;

Hinweise & Beispiele

- Beispiel 3
- aber: Sortierung erfolgt (bei Default-Vergleichsfunktion) anhand der Pointer-Werte
- Ausgabe erfolgt nach Pointer-Wert
- Vergleichsfunktion implementieren, die auf den Werten arbeitet, auf die der Pointer verweist

Hinweise & Beispiele

- Beispiel 3 set mit Pointern

```
struct DereferenceLess{
        bool operator() ( int* p1, int* p2) const {
            return *p1 < *p2;
        }
};
 int main() {
      set<int*, DereferenceLess> m;
// ...
}
```

Zusammenfassung

- set ist eine Datenstruktur zum Speichern von Elementen, wobei keine Dupilkate erlaubt sind
- Elemente werden sortiert gespeichert, basierend auf Äquivalenz
- Vergleichsoperationen müssen "strict weak ordering" erfüllen, sonst ist das Verhalten nicht mehr definiert
- Achtung bei nicht-member Funktionen, Vergleiche basieren oft auf **Gleichheit**
- Komplexitäten:

find(elem), insert(elem)	logarithmisch
erase(elem)	konstant

Quellen

- Nicolai Josuttis – „Die C++ Standardbibliothek“
- S. Meyers – „ Effective STL“
- www.sgi.com/tech/stl/ "STL-Programmers-Guide"

ENDE