Тема 11 Асоциативни контейнери

4.4 Асоциативни контейнери

4.5 Алгоритми, специализирани за асоциативни контейнери

4.6 Алгоритми за генериране на пермутации

4.7. Алгоритми за контейнер Heap

4.4 Асоциативни контейнери

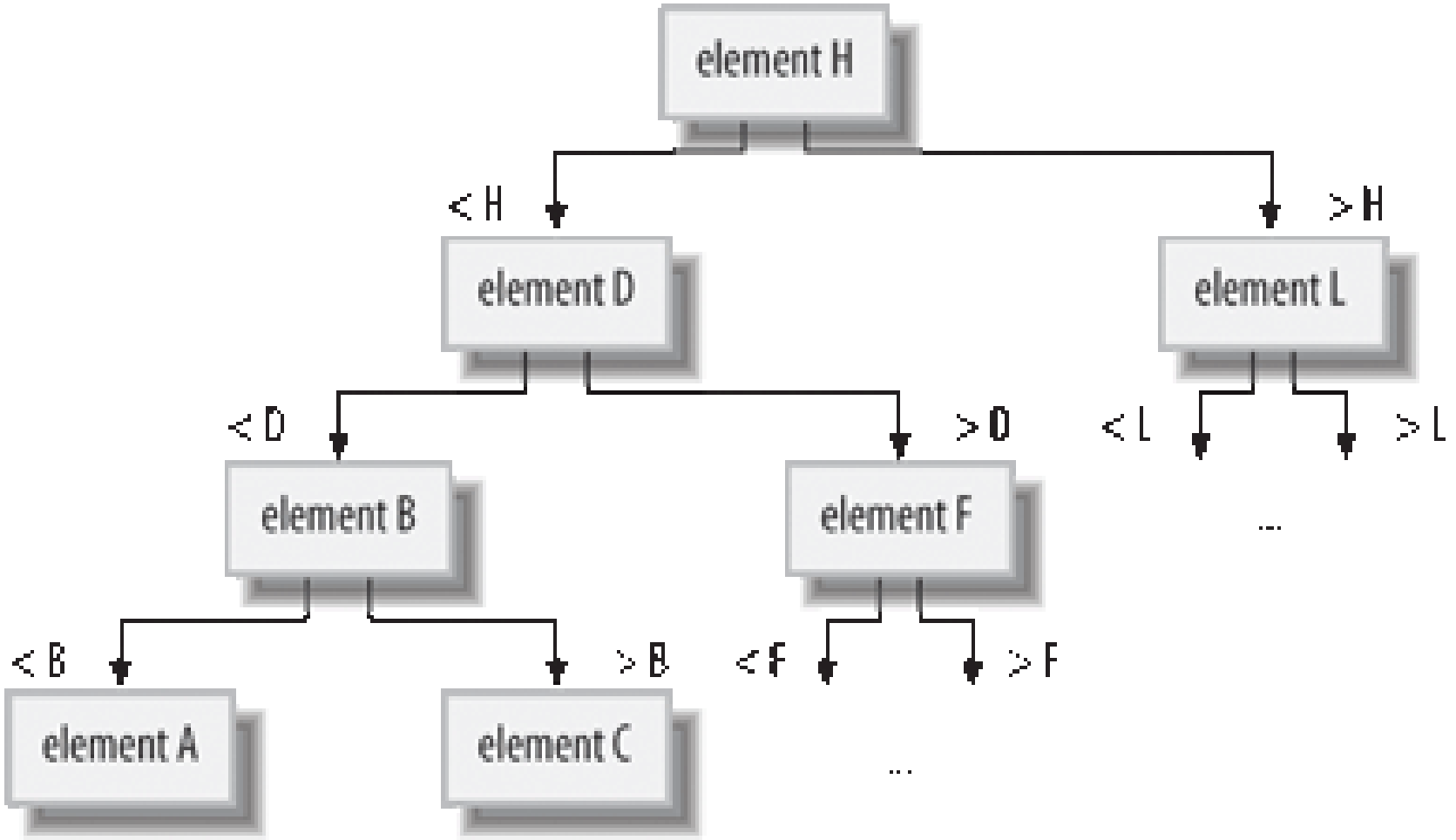
Дефиниция:

Асоциативни контейнери предоставят възможности за бързо получаване на данните основано на двоично търсене по ключове.

Основни характеристики на асоциативните контейнери:

- Организацията на елементите в абстрактни структури (двоично дърво), с ключове за идентификация на елементите;
- Операциите търсене, добавяне и премахване на елемент е пропорционално на логаритъм от размерността-(log(N));
- Добавянето на елемент не предизвиква невалидност на итераторите, поддържани за последователността;
- Премахването-предизвиква невалидност само на итератора към изтрития елемент.

4.4 Асоциативни контейнери



4.4 Асоциативни контейнери

Видове:

- Множество - set;
  - Множество с повторения (мултимножество) - multiset;
- Карта - map;
  - Карта с повторение (мултикарта) - multimap

Разлика:

При set (multiset) – записваните данни са едновременно данни и ключ за сравнение;

При map (multimap) - съхраняват данните и ключовият израз отделно, т.е. ключа не е част от записваните данни.

4.4 Асоциативни контейнери
Множество и мултимножество

Декларация на (set) и (multiset) в библиотеката:

```
template <class Key, class Compare = less<Key>,
          template <class U> class Allocator = allocator>
class set {
    typedef Key key_type;
    typedef Key value_type;
   ...
};
```

4.4 Асоциативни контейнери Множество и мултимножество

Примери:

set <int, less<int> > integerSet; //множество от цели числа,

// подредени в нарастващ ред

multiset <int, less<int> > integerMultiset; //мултимножество

Често се декларира с подразбиращата се функция за сравнение:

set <int> integerSet;

multiset <int> integerMultiset;

Редът на елементите в контейнера се определя чрез извикване на функцията за сравняване Compare- подразбира се клас шаблонна функция на STL less .

4.4 Асоциативни контейнери Множество и мултимножество

Сортиращият израз, които е имплементиран от тази функция трябва да отговаря на следните правила:

- **Да е антисиметричен:**

1. ако x < y е истина (true), тогава y < x е false;
2. ако Compare(x,y) е true, тогава Compare(y,x) е false

- **Да е транзитивен:**

1. ако x < y е истина (true) и y < z е истина (true), тогава x < z е истина (true);
2. Ако Compare(x,y) е true и Compare(y,z) е true, тогава Compare(x,z) е true.

- **Да не е рефлексивен**:

1. x < x е false;
2. Compare(x,x) е false

4.4 Асоциативни контейнери Множество и мултимножество

- **Особености при определяне на еквивалентност между два елемента, т.е** elem1==elem2 **:**
  - Подразбираща се подредба (с оператор <). Търсенето се основава на сравнението на следните условия:

(! (elem1<elem2 || elem2<elem1));

  - Подредба с дефиниран сравнител Compare. Търсенето се основава на сравнението с използване на Compare:

Compare(elem1, elem2) == false && Compare(elem2, elem1) == false

4.4 Асоциативни контейнери Множество и мултимножество

- **Особености при определяне на подредбата. x,y са два последователни елемента в редицата:**
  - *Compare()(y, x) трябва да върне резултат false.*
  - *За подразбиращият се обект функция се използва less<Key>*
    - *Ключовете за сортиране не трябва да допускат намаляване. Проверката за еквивалентност x==y зависи от типа:*
  - *При set се проверява и Compare()(x, y) за true.*
  - *При multiset не се прави тази проверка и се допуска липса на уникалност.*

4.4 Асоциативни контейнери Множество и мултимножество

- **Особености на конструкторите :**

1. Чрез експлицитно зададен итерационен интервал, представен от два итератора.
   - Допуска създаване от друг тип контейнер!
2. Чрез копиращ конструктор от друг асоциативен контейнер от същия тип.

4.4 Асоциативни контейнери Множество и мултимножество

- **Дефиниран е operator=**

Особености :

Обектът (множество) не се копира, когато се присвоява един обект на друг !

- **Основни член функции:**

1. добавяне-insert Предоставя се за единичен елемент по стойността му или за интервал, по зададени начало и край [beg,end);
2. брой на елементите в последователността-size;
3. търсене-find;

4.4 Асоциативни контейнери Множество и мултимножество

4. изтриване на елемент(и) - **erase**. Предоставя се за единичен елемент по стойността му или за интервал, по подадени итератори - начало и край [beg,end);
5. изтриване на всички елементи - **clear**;
6. размяна на стойностите на две последователности-**swap**;
7. брой на срещанията на даден елемент в последователността - **count**.

   За множество връща само 0 или 1;
8. определяне на елемент, не по-малък (по-голям или равен) от определена стойност- **lower_bound** (**upper_bound**). Връща итератор към намерения елемент (или итератор зад последния);
9. определяне на двойката елементи, които се получават от lower_bound и upper_bound - **equal_range- резултат двойка итератори**

4.4 Асоциативни контейнери
Множество и мултимножество

Примери:

- Пример **count при мултимножество:**

с параметър – O2:
O3 O3 O2 O2 O2 O1 O1 O1
Резултат 3

- Пример **count при множество** с параметър – O2:

O3 O2 O1;
Резултат 1

- Пример **equal_range** с параметър – O2:

O3 O3 O2 O2 O2 O1 O1 O1

4.4 Асоциативни контейнери Множество и мултимножество

- **Особености при достъпа:**

1. Не допускат директен достъп до елементите от последователността, поради което трябва да се използват итератори;
2. Всички асоциативни контейнери имат дефинирани член функциите begin, end, rbegin, rend за получаване на итератори;
3. Предоставят функциите empty, size, max_size и swap;
4. Всички елементи се обявяват като **непроменяеми** (const) относно ключа, защото не бива да се допуска с промяна на елемент да се наруши реда-**erase+insert**;
5. Общите алгоритми от библиотеката, които сменят съдържанието на редицата не работят. За целта се предоставят собствени член функции – erase, clear и insert;

4.4 Асоциативни контейнери Множество и мултимножество

Примерни програми за използване на set (multiset):

```
class CStudentData {
public:
    CStudentData() : m_strName (""), m_dUspeh(0) {}
    CStudentData(const string& n, double u ) :
       m_strName (n), m_dUspeh (u) {}
    string     m_strName;
    double    m_dUspeh;
  friend ostream& operator<< (ostream& os, const CStudentData& e);
};
ostream& operator<< (ostream& os, const CStudentData& e) {
    os << "Student: " << e.m_strName << " " << e.m_dUspeh;
    return os;
}
```

4.4 Асоциативни контейнери
Множество и мултимножество

```
class CStudentHead {
public:
    CStudentData MainData;
  CStudentHead (const string& strFN, const CStudentData& e) :
      FacultyNumber (strFN), MainData (e) {}
  string FacultyNumber;
    bool operator< (const CStudentHead& e) const {
      return FacultyNumber < e.FacultyNumber; }
    bool operator== (const CStudentHead& e) const {
      return FacultyNumber == e.FacultyNumber; }
};
```

4.4 Асоциативни контейнери Множество и мултимножество

```
set <CStudentHead, less<CStudentHead> > student_set;
multiset <CStudentHead, less<CStudentHead> > student_multiset;
```

Съкратено:

```
set <CStudentHead> student_set;
multiset <CStudentHead> student_multiset;

// Създаване на обектите с общи данни:
CStudentData st1 ("ivan", 5.54);
CStudentData st2 ("stela", 3.54);
CStudentData st3 (“simo", 4.5);

// Създаване на обектите с данни за добавяне и търсене:
CStudentHead s1 ("61460102", st1);
CStudentHead s2 ("61460101", st2);
CStudentHead s3 ("61460103", st3);
```

4.4 Асоциативни контейнери
Множество и мултимножество

Добавяне в множество:

```
pair<set <CStudentHead>::iterator, bool>StudentPair1 = student_set.insert (s1);
if (StudentPair1.second) cout << "insert ok\n"; else cout << "not inserted\n";
pair<set <CStudentHead>::iterator, bool> StudentPair2 = student_set.insert (s2);
if (StudentPair2.second) cout << "insert ok\n"; else cout << "not inserted\n";
pair<set <CStudentHead>::iterator, bool> StudentPair3 = student_set.insert (s3);
if (StudentPair3.second) cout << "insert ok\n"; else cout << "not inserted\n";
```

Резултат:

```
insert OK
insert OK
insert OK
```

C++11 – Допуска:

pair<set <CStudentHead>::iterator, bool> ⇔ *auto*

Пример:

```
auto StudentPair1 = student_set.insert(s1);
```

4.4 Асоциативни контейнери Множество и мултимножество

- Примери-без да се обработва резултата:

```
student_set.insert (s1);
student_set.insert (s2);
student_set.insert (s3);
StudentPair1 = student_set.insert (s3);
if (StudentPair1.second) cout << "insert ok\n"; else cout << "not inserted\n";
 StudentPair1 = student_set.insert (s2);
if (StudentPair1.second) cout << "insert ok\n"; else cout << "not inserted\n";
 StudentPair1 = student_set.insert (s1);
if (StudentPair1.second) cout << "insert ok\n"; else cout << "not inserted\n";
```

Резултат:

```
not inserted
not inserted
not inserted
```

4.4 Асоциативни контейнери Множество и мултимножество

Определяне броят на елементите в контейнера:

```
int iSize = student_set.size();
```

Деклариране на итератори и извеждане на данните:

```
set <CStudentHead >::iterator start = student_set.begin();
set <CStudentHead>::iterator end =  student_set.end();
set <CStudentHead>::iterator it;
for( it=start; it != end; it++ )
  cout << endl << (*it).MainData;
```

Търсене и извеждане:

```
// търсене
it = student_set.find(s2);
// извеждане
if( it != student_set.end() )
   cout << endl << "Found " << (*it).MainData << endl;
```

Изтриване:

```
student_set.erase(s2);
```

4.4 Асоциативни контейнери
Карта (map) и мултикарта (multimap)

Дефиниция:

Картата и мултикартата са контейнери, които съхраняват като елементи двойки ключ/стойност. Елементите се поддържат автоматично сортирани в съответствие със сортиращия израз. Разликата между двата е, че multimap допуска дублиране на елементите докато map не допуска това.

Декларацията на map в библиотеката може да се представи по следния начин:

```
template <class Key, class T, class Compare = less<Key>, class
  Allocator = allocator<pair<const Key,T> > >
class map {   ... };
```

4.4 Асоциативни контейнери Карта (map) и мултикарта (multimap)

Декларация на обект, пример с ключ от тип int :

map<int,string,less<int> > myMap;

- Първият шаблонен аргумент е типа на ключовия обект а вторият-на стойността. Елементите на map и multimap трябва да отговарят на следните изисквания:
- Двойката ключ/стойност да има копиращ конструктор и оператор за присвояване;
- Ключовият обект да има оператор за сравнение в съответствие със сортиращия критерии.
- Подразбира се оператора за сравнение на ключовете less<Key>:

map<int,string> myMap;

4.4 Асоциативни контейнери
Карта (map) и мултикарта (multimap)

- Дефиниране на сортиращ критерий:

Примери при ключове x, y

- Подразбира се критерии - нарастващ ред на елементите относно ключа :
  - Ключовият обект да предоставя оператора <;
  - Условието за сортиране се използва и за определяне на еквивалентност между обектите:

    (! (x<y || y<x));

  - Условието е същото като при set, за ключовия обект

- Чрез обект функция Compare. Условията за еквивалентност се изчисляват чрез извикване на функцията:

  Compare(x, y) == false && Compare(y, x) == false

4.4 Асоциативни контейнери Карта (map) и мултикарта (multimap)

- Принципните разлики спрямо множествата са следните:

1. Съхраняват двойки ключ/стойност.
2. Възможността при картата да се използва последователността като *асоциативен масив*.
   - Имат оператор за индексиране operator[]и се осигурява *асоциативно търсене* и достъп до стойността на елемента, **ключа на който е подаден като индекс;**
   - В зависимост от типа на използвания ключ индекса може да бъде както число, така и string или друг тип данни. За примера :

```
string strData = MyMap[2].second;
```

   - Извършва търсене на стойността на ключа 2 и връща данните по този ключ;
   - Висока ефективност се осигурява при търсене по известен ключ на елемента;
   - Търсенето по известна стойност на елемента има ниска ефективност.

4.4 Асоциативни контейнери Карта (map) и мултикарта (multimap)

- Автоматичното поддържане на сортировка налага следните особености:
- Не може директно да се променя ключовата стойност, за да не се нарушава реда на сортировка. Както при множествата, достъпът до елементите е чрез итератори и ключовата стойност е достъпна, но е константна (непроменяема). За да се промени даден елемент се изтрива елемента по старият ключ и се добавя новия;
- Стойността на елемента (value) може да се променя без проблеми, тъй като не участва в сортиращия израз.

4.4 Асоциативни контейнери
Карта (map) и мултикарта (multimap)

Създаването на елементите става при конструиране на обектите. Елементите на карта и мултикарта се създават като се използва помощния клас двойка:

pair<const Key, value>

Достъпът до елементите на двойката се осъществява с помощта на член променливите на двойката:

first – Ключовата стойност (Key);

second- Стойността на елемента (value)

например:

```
int iKey=myMap.first;
string strValue= myMap.second;
```

4.4 Асоциативни контейнери
Карта (map) и мултикарта (multimap)

Примерни програми за използване на map (multimap):

```
class CStudentProtocolData {
  string m_strName;
  string m_strFacultyNumber;
public:
  CStudentProtocolData(const string& name, const string& fnum) :
    m_strName(name), m_strFacultyNumber(fnum) {}
  friend ostream& operator<< (ostream& os, const CStudentProtocolData& e){
    return os<< "Student: " << e.m_strName << " " <<
     e.m_strFacultyNumber;
  }
};
```

4.4 Асоциативни контейнери
Карта (map) и мултикарта (multimap)

```
// Дефиниране на карта от двойки, съставена от точките
 // на студента и  данните му - обект CStudentProtocolData:
// Валиден за C++11
 map< int, CStudentProtocolData> OOP_protocol = {
   make_pair(55, CStudentProtocolData("ivan", "61460102")),
   make_pair(63, CStudentProtocolData("stela", "61460101")),
   make_pair(100, CStudentProtocolData(“simo", "61460103"))
 };
```

4.4 Асоциативни контейнери
Карта (map) и мултикарта (multimap)

```
// Извеждане на картата на конзолен изход,
// auto⇔map<int, CStudentProtocolData >::const_iterator

  auto it = OOP_protocol.begin();
  while ( it != OOP_protocol.end()){
    cout << (*it).first << " " << (*it).second << endl;
    it++;
  }
```

4.4 Асоциативни контейнери
Карта (map) и мултикарта (multimap)

```
// Търсене на студент, получил оценка 55, 63 и 42.
// auto⇔map<int, CStudentProtocolData >::const_iterator
  auto itm = OOP_protocol.find(55);
  if (itm == OOP_protocol.end() )  cout << "not found";
  else  cout << (*itm).second;
// Търсене с оператор за индекс [] => връща намерения
// обект. Ако не се намери ?
CStudentProtocolData d = OOP_protocol[55];
  cout << d << endl;
  d = OOP_protocol[63];
  cout << d << endl;
  d = OOP_protocol[42];
  cout << d << endl;
```

4.4 Асоциативни контейнери
Карта (map) и мултикарта (multimap)

```
// Изтриване на студента, с оценка 63 точки
OOP_protocol.erase (63);
// Изтриване на първия от останалите
OOP_protocol.erase (OOP_protocol.begin() );
// Изтриване по целия итерационен интервал
OOP_protocol.erase (OOP_protocol.begin(), OOP_protocol.end() );
if (OOP_protocol.empty() ) cout << "empty protocol";
```

Изход: *empty protocol*

4.4 Асоциативни контейнери
Карта (map) и мултикарта (multimap)

***erase** прави невалидни итераторите и референциите към изтрития елемент-разлика с последователните контейнери:*

- *при асоциативните контейнери не се връща резултат от функцията. Причини:Да не се загуби време за намиране (двоично търсене) на стойността на итератора.*

```
multimap< int, CStudentProtocolData> OOP_grupa;
OOP_grupa.insert (make_pair(1, OOP_protocol[0]));// в група 1
OOP_grupa.insert (make_pair(1, OOP_protocol[1]));      // 1
OOP_grupa.insert (make_pair(2, OOP_protocol[2]));      // 2
```

count получава като аргумент ключова стойност и връща като резултат броят елементи, с тази ключова стойност:

```
multimap <int, CStudentProtocolData >::size_type count=
    OOP_grupa.count(1);
cout << count;
```

Изход: *2*

4.4 Асоциативни контейнери
Карта (map) и мултикарта (multimap)

```
typedef multimap <int, CStudentProtocolData >::iterator ItMap;
pair< ItMap, ItMap > result = OOP_grupa.equal_range (1);
copy (result.first,
      result.second,
      ostream_iterator<pair<int, CStudentProtocolData > > (cout , "\n")
     );
```

Изход:

stela 61460101
simo 61460103
ivan 61460102

4.4 Асоциативни контейнери
Карта (map) и мултикарта (multimap)

1) ForwardIterator lower_bound(ForwardIterator first, ForwardIterator last, const Type &value);

2) ForwardIterator lower_bound(ForwardIterator first, ForwardIterator last, const Type &value, Compare comp);

Алгоритъмът работи върху подредени редици.

1)В случай, че елемент от редицата е не по-малък (т.е. по-голям или равен) на подадения елемент value, се връща итератора към този елемент. В противен случай се връща итератор зад последния елемент-last. Това е първият итератор, по който може да се вмъкне value без да се наруши подредбата.

2)В случай, че предикатът comp, приложен върху елемент от редицата и подадения елемент value върне false, се връща итератора към този елемент. В противен случай се връща итератор зад последния елемент-last.

4.5 Алгоритми, специализирани за асоциативни контейнери

- Сечение на множества- ***set_intersection***;
- Обединение на множества ***set_union***;
- Разлика на множества- ***set_difference***;
- Симетрична разлика на множества- ***set_symmetric_difference***;
- Подмножество- ***includes***

4.5 Алгоритми, специализирани за асоциативни контейнери

- Примерен формат на алгоритмите:

```
template<class InIt1, class InIt2, class OutIt>
    OutIt set_symmetric_difference(InIt1 first1,
    InIt1 last1,
        InIt2 first2, InIt2 last2, OutIt x);
template<class InIt1, class InIt2, class OutIt, class
    Pred>
    OutIt set_symmetric_difference(InIt1 first1,
    InIt1 last1,
        InIt2 first2, InIt2 last2, OutIt x, Pred pr);
```

4.5 Алгоритми, специализирани за асоциативни контейнери

Пример: Да се състави клас CTest, който да има член променлива vector<string>.

Да се напише конструктор на класа, чрез който член променливата да се запълва с думи, записани в текстови файл, разделени с интервал. Името на файла се подава като параметър на конструктора на класа:

CTest(const string& strFileName).

Да се напишат и следните член функции на класа:

- void WriteVector() - извежда съдържанието на член променлива vector на класа;
- bool Find(const string& _str) -търси стринга параметър _str във вектора член променлива на класа и връща резултат true при откриването му ;
- void WriteNotEqualStrings(vector<string>& _vStr) - извежда в конзолния изход всички стрингове, освен еднаквите между вектора параметър и вектора член променлива на класа.

Пример: Файлът съдържа: AB A AA BB C D E, _vStr съдържа: AB A D B C

Изход: B AA BB E

4.5 Алгоритми, специализирани за асоциативни контейнери

Реализация на WriteNotEqualStrings чрез множество:

```
void CTest::WriteNotEqualStrings( vector<string>& _vStr ) {
set<string> setStr1( _vStr.begin(), _vStr.end());
set<string> setStr2( m_vstr.begin(), m_vstr.end());
set<string> setStrResult;
set_symmetric_difference( setStr1.begin(), setStr1.end(),
    setStr2.begin(),setStr2.end(), inserter(setStrResult,
    setStrResult.begin()) );
// извеждане на резултата от операцията
  copy(setStrResult.begin(), setStrResult.end(),
          ostream_iterator<const string, char>(cout, " ") );
  cout << endl;
}
```

4.5 Алгоритми, специализирани за асоциативни контейнери

Реализация на WriteNotEqualStrings без множества:

```
void CTest::WriteNotEqualStrings( vector<string>& _vStr ) {
vector<string> vStrR;
  sort( m_vstr.begin(), m_vstr.end());
  sort( _vStr.begin(),_vStr.end() );
  set_symmetric_difference( _vStr.begin(), _vStr.end(),
    m_vstr.begin(), m_vstr.end(), back_inserter(vStrR) );
  copy(vStrR.begin(), vStrR.end(), ostream_iterator<const
    string, char>(cout, " ") );
  cout << endl;
}
```

4.6 Алгоритми за генериране на пермутации

- next_permutation() – прав ред;
- prev_permutation() - обратен ред.

```
template< class BidirectionalIterator >
bool next_permutation( BidirectionalIterator first,
                       BidirectionalIterator last );
template< class BidirectionalIterator, class Compare >
bool next_permutation( BidirectionalIterator first,
            BidirectionalIterator last, Compare comp );
```

Изисква началният итерационен интервал да е подреден в нарастващ ред

Пример: {a,b,c}

abc, acb, bac, bca, cab, cba

4.7. Алгоритми за контейнер Heap

Дефиниция: Контейнерът heap е форма на двоично дърво, представено като масив. STL предоставя max-heap представяне, при което ключовата стойност във всеки връх е по-голяма или равна на ключовата стойност на неговите дъщерни върхове.

Операции:

- make_heap();
- pop_heap();
- push_heap();
- sort_heap().

Асоциативни контейнери

Въпроси?