

C++ STL

assoc. prof. Atanas Semerdzhiev

1

ОСНОВНИ ИЗТОЧНИЦИ

- Bjarne Stroustrup (2013) **The C++ Programming Language**, 4th ed. <http://www.stroustrup.com/4th.html>
- Bjarne Stroustrup (2014) **Programming: Principles and Practice Using C++**, 2nd ed. <http://www.stroustrup.com/programming.html>
- Nicolai M. Josuttis (2012) "**The C++ Standard Library - A Tutorial and Reference**", 2nd ed. <http://cppstdlib.com/>
- David Vandevoorde, Nicolai Josuttis (2017) **C++ Templates - The Complete Guide**, 2nd ed. <http://www.tmplbook.com/>

2

История на стандарта

First C++ Standard

- C++98: ISO/IEC 14882:1998.
- C++03: “technical corrigendum” (“TC”), ISO/IEC 14882:2003

Second C++ Standard

- C++11: ISO/IEC 14882:2011
- C++14: ISO/IEC 14882:2014
- C++17: ISO/IEC 14882:2017

3

За стандартната библиотека

- Като замисъл и имплементация, всяка част от библиотеката има собствена логика, която може да не е консистентна с останалите!
- STL = Standard Template Library (Стандартна библиотека с шаблони)
- Бележка: template може да се произнася като „темплейт“ (UK) или „темплит“ (USA).
- Standard Library ≠ Standard Template Library

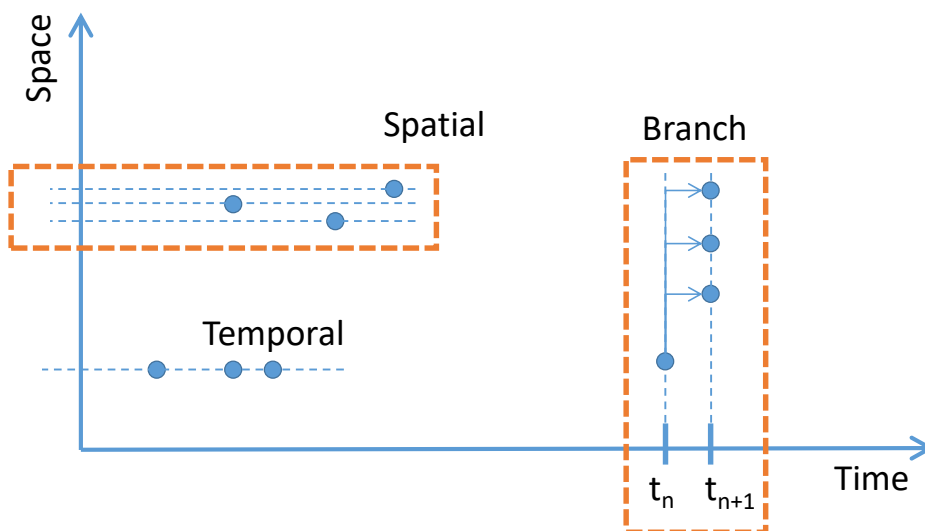
4

Locality

- Temporal Locality
- Spatial Locality
- Branch Locality
- Следствие за обхождането и работата с различни структури
- Пример: Обхождане на двумерен масив

5

Spatial-Temporal Space



6

Сложност

Видове сложност

- Времева (temporal)
- Пространствена (spatial)

Big-O класове

$O(1)$
 $O(\log N)$
 $O(N)$
 $O(N \log N)$
 $O(N^2)$
 $O(2^N)$
 $O(N!)$

7

Big-O in practice

	1	2	3	10	100	1000	1 000 000
$O(1)$	1	1	1	1	1	1	1
$O(\log N)$	1	1	2	4	7	10	20
$O(N)$	1	2	3	10	100	1000	1 000 000
$O(N \log N)$	1	2	5	34	665	9964	19 931 569
$O(N^2)$	1	4	9	100	10 000	1 000 000	10^{12}
$O(2^N)$	1	4	8	1024	$> 10^{30}$	$> 10^{301}$	$> 10^{301029}$
$O(N!)$	1	2	6	3 628 800	$> 10^{157}$	$> 10^{2567}$	$> 10^{5565708}$

8

Big-O in STL

- Стандартът изисква определена сложност за отделните елементи;
- Той обаче НЕ определя какъв алгоритъм трябва да реализират те, въпреки, че зад повечето от тях стои конкретна идея (напр. динамичен масив за vector).

9

Namespaces

Всички идентификатори от стандартната библиотека се намират в пространството (namespace) `std`.

Можем да ги използваме чрез:

1. Fully-qualified names

```
std::cout << "1";
```

2. using-declaration

```
using std::cout;  
cout << "1";
```

3. using-directive

```
using namespace std;  
cout << "1";  
// За предпочитане е  
// да се избягва!
```

10

```
// Понякога името на даден namespace може да е прекалено  
// дълго или да има вероятност да се промени.
```

```
#include <filesystem>  
#include <iostream>  
  
namespace fs = std::experimental::filesystem;  
  
int main()  
{  
    fs::path pathToDir("C:\\Temp");  
    pathToDir /= "subdir1";  
    std::cout << pathToDir << std::endl;  
}
```

11

Header Files

1. По конвенция, новите header-файлове нямат разширение. Например:

```
#include <iostream>  
#include <utility>  
#include <algorithm>
```

Идентификаторите, които се дефинират в тях са в namespace std.

2. За поддържане на съвместимост, с компилатора ви вероятно идват две версии на header-файловете. Например:

```
#include <string.h>  
#include <stdlib.h>
```

и

```
#include <cstring>  
#include <cstdlib>
```

12

Обработка на изключения

13

Пример: хвърляне на изключение

```
try
{
    p = new int[SIZE];
}
catch (std::bad_alloc& e)
{
    std::cerr << "ERROR: allocation failed (exception: "
                << e.what()
                << ")\n";
}
```

По-детайлен пример можете да намерите тук: <https://github.com/semerdzhiev/oop-samples/tree/master/Exceptions>

14

Видове изключения

1. Language support

- Използват се от средствата на езика C++;
- Например `bad_cast`, което се хвърля от `dynamic_cast`.

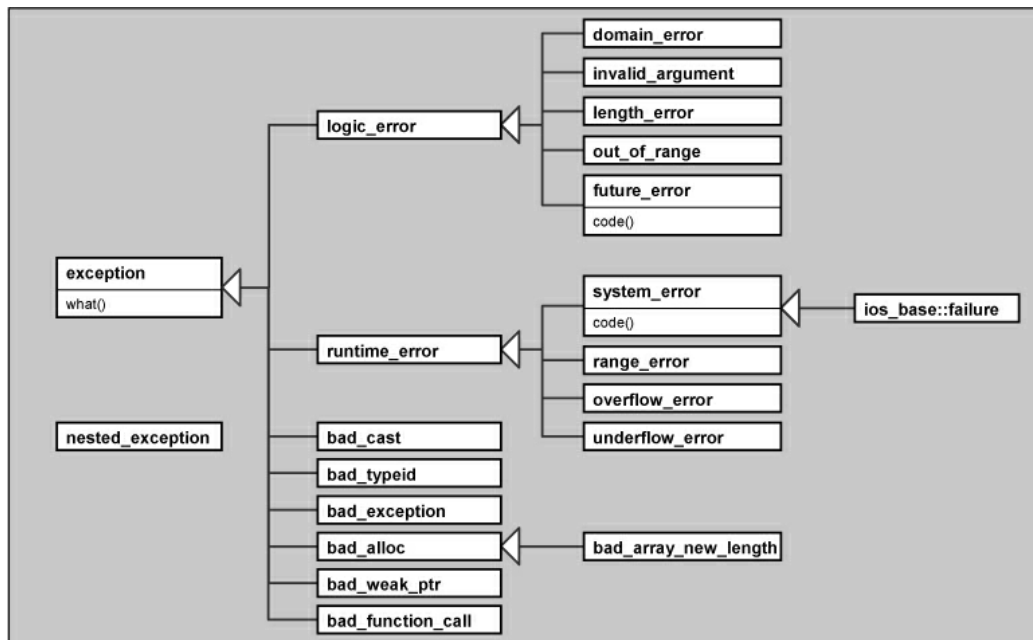
2. Logic errors

- Възникват от нарушения в логиката на програмата;
- Могат да се избегнат ако напишем програмата „по-добре“.

3. Runtime errors

- Възникват по време на изпълнение;
- Не са в резултат от грешки в програмната логика. Например: няма достатъчно памет, липсващ конфигурационен файл и т.н.

15



Източник: Nicolai M. Josuttis (2012) *The C++ Standard Library: A Tutorial and Reference*, 2nd Edition

16

Информация за грешката

В общия случай, информация за това какво се е объркало в програмата можете да получите по два начина:

1. От **типа на изключението**, което сте хванали;
2. От символния низ, който връща функцията `what()`.

`what()` е виртуална функция дефинирана в `exception` по следния начин:

```
virtual const char* what() const noexcept;
```

17

Код на грешка

- Някои изключения, като например `system_error` носят допълнително и код на грешката, която е възникнала;
- Той може да се получи от член-функцията `code()`;
- Възможните кодове са изброени в `std::errc` (за повече информация вижте <http://en.cppreference.com/w/cpp/error/errc>).

18

Кое изключение да хвърлим?

Добра (но не винаги удачна) практика е:

1. Ако е възможно, да използвате някой от стандартните класове;
2. Ако те не ви вършат работа, да наследите някой от тях и да реализирате нужното поведение;

19

Добавяне на съобщение за грешка

```
if (i > SIZE)
    throw std::out_of_range("Invalid array index");
else
    std::cout << array[i];
```

Не всички класове от йерархията имат конструктор, който приема параметри!

Например `bad_alloc` няма.

20

nullptr

21

За указателите и нулите

- Какво се случва, когато присвоим нула (0) на указател?
- Какво се случва, когато сравним два null-указателя?

```
int* p = 0;  
int* q = 0;  
  
if (p == q)  
{  
    //...  
}
```

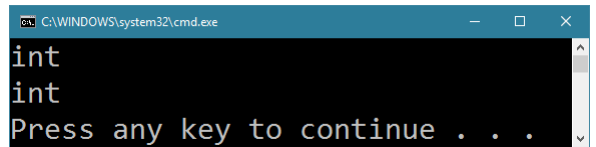
22

Проблем: разлика между null и 0

```
void f(int)
{
    std::cout << "int\n";
}

void f(char*)
{
    std::cout << "char*\n";
}

int main()
{
    f(0);
    f(NULL);
    return 0;
}
```



23

Проблем: ориентиране в кода

```
int* pData = 0;
int Index = 0;
double Income = 0.0;
char c = '\0';
char d = 0;
f(0);
std::cout << 0 << "\n" << pData;
```

24

Примери за употреба на nullptr

```
char *p = nullptr;
char *q = 0;

p == q;           // true
p == nullptr;    // true
q == nullptr;    // true

f(nullptr); // извежда
            // "char*"

int(*pf)(int);

int (MyClass::*pm)(int);

pf = nullptr; // OK
pm = nullptr; // OK
```

25

Допълнителни източници

- Herb Sutter, Bjarne Stroustrup (2003) **A name for the null pointer: nullptr**. <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2003/n1488.pdf>
- Herb Sutter, Bjarne Stroustrup (2007) **A name for the null pointer: nullptr (revision 4)**. <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2007/n2431.pdf>

26

Наредени двойки

```
#include <utility>
```

27

pair

```
std::pair<int, char> p1;  
std::pair<int, char> p2(2, 'b');  
  
p1.first = 1;  
p1.second = 'a';  
  
p1.swap(p2); // Алтернатива: swap(p1, p2)  
  
std::cout << "p1(" << p1.first << ", " << p1.second << ")\n";  
std::cout << "p2(" << p2.first << ", " << p2.second << ")\n";
```

За повече информация: <http://www.cplusplus.com/reference/utility/pair/>

28

pair

```
std::pair<int, char> p1(10, 'a');  
std::pair<int, char> p2(20, 'a');  
std::pair<int, char> p3(10, 'b');
```

```
p1 < p2; // true  
p1 < p3; // true  
p1 == p1; // true  
p1 != p2; // true
```

29

pair

```
std::pair<int, char> p1(10, 'a');  
  
std::get<0>(p1) = 1;  
std::get<1>(p1) = 'a';  
  
std::cout << "p1(" << std::get<0>(p1)  
          << ", " << std::get<1>(p1) << ")\n";
```

30

Наредени n-орки

```
#include <tuple>
```

31

tuple

```
std::tuple<int, double, char> t1;  
std::tuple<int, double, char> t2(10, 5.0, 'b');  
  
std::get<0>(t1) = 10;  
std::get<1>(t1) = 1.5;  
std::get<2>(t1) = 'a';  
  
t1 < t2;  
t1 < std::make_tuple(20, 7.0, 'c');  
t1 < std::make_tuple<int, double, char>(20, 7.0, 'c');
```

32

ВАЖНО!

Въпреки, че работата с наредени n-орки (tuples) може на пръв поглед да изглежда лесна, тяхната реализация се основава върху няколко механизма на езика, които не са тривиални.

За повече информация прочетете главите от основните източници, които третираат:

- Variadic Templates
- Move Semantics

Полезен може да ви бъде и:

- Eli Bendersky (2014) **Variadic templates in C++**.
<http://eli.thegreenplace.net/2014/variadic-templates-in-c/>

33

tuple

```
typedef std::tuple<int, double, char> IntDoubleChar;

IntDoubleChar t1(10, 5.0, 'c');

auto t2 = std::tuple_cat(t1, std::make_tuple(20, 30));

std::cout << "Size of t1 is "
            << std::tuple_size<IntDoubleChar>::value
            << "\nSize of t2 is "
            << std::tuple_size<decltype(t2)>::value
            << "\n";
```

34

Помощни функции

```
#include <algorithm>
```

35

Намиране на минимум

```
int value;
```

```
value = std::min(10, 20); // 10
```

```
value = std::min(10, 10.0); // грешка!
```

```
std::pair<int,int> m = std::minmax(20, 10);
```

```
std::cout << m.first << " <= " << m.second << "\n";
```

36

Когато стандартното сравнение не ни върши работа

```
bool SmallerLastBit(int a, int b)
{
    return (a & 1) < (b & 1);
}

int main()
{
    std::cout << std::min(500, 1, SmallerLastBit)
               << " has smaller last bit\n";
}
```

37

Разменяне на стойностите на два елемента

```
int x = 1, y = 2;

std::swap(x, y);

std::cout << "x = " << x
          << ", y = " << y << "\n";
```

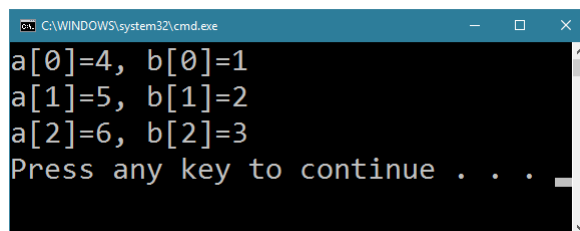
38

swap е предефинирана за масиви (C++11)

```
int a[] = { 1, 2, 3 };
int b[] = { 4, 5, 6 };

std::swap(a, b);

for (int i = 0; i < 3; i++)
{
    std::cout
        << "a[" << i << "]= " << a[i]
        << ", b[" << i << "]= " << b[i] << "\n";
}
```



```
C:\WINDOWS\system32\cmd.exe
a[0]=4, b[0]=1
a[1]=5, b[1]=2
a[2]=6, b[2]=3
Press any key to continue . . .
```

39

Аргументи на swap

```
// Типовете на аргументите на swap()
// трябва да съвпадат!
int a[] = { 1, 2, 3 };
int c[] = { 7, 8, 9, 0 };
std::swap(a, c); // Грешка!

int x = 1;
long y = 1;
std::swap(x, y); // Грешка!
```

40

Оптимизиране на разменянето

- `swap()` може да се предефинира за произволен тип.
- Тя може да оптимизира разменянето на стойности, като използва някакво свойство на типа – например за контейнерен тип може да размени указатели, вместо да копира съдържание.
- Така `swap()` може да работи по-бързо от еквивалентния код:

```
Temp = A;  
    A = B;  
    B = Temp;
```

- Това е направено за стандартните контейнери

41

Обща версия на `swap`

В зависимост от вида и версията на компилатора, с който работите, шаблонната, обща версия на `swap` може да има няколко варианта:

- извършваща традиционно разменяне на елементи;
- за `movable` типове (C++11);
- за масиви (C++11).

Версията за масиви не просто разменя съответните им елементи, а на свой ред извиква `swap` за тях. Така размяната може да се извърши по-ефикасно, стига типът на елементите да го поддържа.

42

Дефиниране на swap за собствен тип T

Един възможен подход:

- Добавете член-функция swap към T, която извършва разменянето.
- Дефинирайте глобална функция swap същия namespace като T, която разменя два обекта. Тази функция ще съдържа само едно извикване на T::swap.

За справка: <https://en.cppreference.com/w/cpp/algorithm/swap>

43

```
class X {  
    int *pBuffer = nullptr;  
    size_t size = 0;  
  
public:  
    X(size_t size)  
        : size(size)  
    {  
        pBuffer = new int[size];  
    }  
  
    ~X()  
    {  
        delete[] pBuffer;  
    }  
  
    void swap(X& other) noexcept  
    {  
        std::swap(pBuffer, other.pBuffer);  
        std::swap(size, other.size);  
    }  
};
```

```
void swap(X& a, X& b) noexcept  
{  
    a.swap(b);  
}  
  
int main()  
{  
    X a(10), b(20);  
    swap(a, b);  
}
```

44

iter_swap()

```
// Функцията е еквивалентна на следното
template <class ForwardIterator1,
          class ForwardIterator2>
inline void iter_swap(ForwardIterator1 a,
                      ForwardIterator2 b)
{
    using std::swap;
    swap(*a, *b);
}
```

Източник: http://www.cplusplus.com/reference/algorithm/iter_swap/

45

Контейнери и итератори

46

Контейнери

- На следния адрес е направено нагледно сравнение между контейнерите и операциите, които те поддържат:
<http://www.cplusplus.com/reference/stl/>
- Важно: стандартът не фиксира алгоритмите и структурите от данни, които контейнерите реализират! Те следват от изисквания върху поведението и сложността им.

47

Видове контейнери

- Sequence containers
 - наредени колекции, в които всеки елемент има уникална позиция
 - array, vector, deque, list, forward_list
- Associative containers
 - сортирани колекции; позицията на елементите зависи от техния ключ.
 - set, multiset, map, multimap.
- Unordered (associative) containers
 - колекции без наредба; позицията на елементите няма значение и може да се променя.
 - unordered_set, unordered_multiset, unordered_map, unordered_multimap.

48

Типична реализация

- Sequence containers
 - Масив или свързан списък
- Associative containers
 - Двоично дърво (например червено-черно дърво)
- Unordered (associative) containers
 - Хеш

49

Основни операции с контейнерите

Операция	Коментар
<code>empty()</code>	Може да бъде по-бързо от <code>size() == 0</code>
<code>size()</code>	
<code>max_size()</code>	
<code>swap()</code>	Работи по-различно за класа <code>array</code>
<code>clear()</code>	
Оператори за сравнение	За наредените контейнери; По лексикографската наредба

50

Итератори

- Основна идея
- Основни операции:

Операция	Описание
<code>operator *</code>	Текущ елемент
<code>operator ++</code>	Придвижва итератора напред
<code>operator --</code>	Придвижва итератора назад
<code>operator ==</code> <code>operator !=</code>	Проверява дали два итератора сочат една и съща позиция
<code>operator =</code>	Присвоява позиция на итератор

51

Итератори

Контейнерите, които могат да бъдат итерирани предлагат следните операции (но може да реализират и допълнителни):

- `begin()` – връща итератор към първия елемент
- `end()` – връща итератор към последния елемент

Освен това те предлагат и два типа итератори: `iterator` и `const_iterator`. Например:

```
std::vector<int>::iterator
```

52

std::vector

- По идея моделира динамичен масив
- Основни свойства
- Предимства и недостатъци
- Примерна реализация:
<https://github.com/semerdzhiev/sdp-samples/tree/master/Dynamic%20Array>
- Източници:
<http://www.cplusplus.com/reference/vector/vector/>
<http://en.cppreference.com/w/cpp/container/vector>

53

std::vector

Гарантирано във вектора елементите са подредени в непрекъснат блок в паметта

Това значи, че:

$$\&v[i] == \&v[0] + i$$

54

Примери за употреба

```
std::vector<int> v;

v.push_back(0);
v.push_back(1);
v.push_back(2);

v[1] = 10;

for (int i = 0; i < v.size(); i++)
    std::cout << v[i] << " "
               << v.at(i) << "\n";

std::vector<int> v2(v);
v2.clear();

v.pop_back();
v.pop_back();

v.shrink_to_fit();
```

55

Тип на променлива, по която обхождаме

```
// Вероятно ще сработи, но не е коректно.
// Може да предизвика грешка
for (int i = 0; i < v.size(); i++)
    std::cout << v[i] << "\n";

// OK
for (std::vector<int>::size_type i = 0; i < v.size(); i++)
    std::cout << v[i] << "\n";

// ВАЖНО: разлика между std::size_t и
// std::<container>::size_type
```

За повече информация вижте: <http://stackoverflow.com/questions/409348/iteration-over-vector-in-c>

56

Обхождане с итератор

```
std::vector<int>::iterator it;

for (it = v.begin(); it != v.end(); ++it)
    std::cout << *it << "\n";

// в един for... ако се събере на екрана :- )
for (std::vector<int>::iterator it = v.begin();
     it != v.end();
     ++it)
    std::cout << *it << "\n";
```

За повече информация вижте: <http://stackoverflow.com/questions/409348/iteration-over-vector-in-c>

57

auto / decltype

```
for (auto it = v.begin(); it != v.end(); ++it)
    std::cout << *it << "\n";

for (decltype(v)::size_type i = 0; i < v.size(); ++i)
    std::cout << v[i] << "\n";
```

58

auto/decltype

ВАЖНО: Механизмът под `auto` и `decltype` не е тривиален. Ако се използват без разбиране, това може да доведе до трудни за откриване грешки в кода. За повече информация вижте например: Scott Meyers (2014) **Effective Modern C++**, глави 1 – 4

В много от примерите в курса употребата на `auto` е избягната, за да се покаже експлицитно какъв тип се използва. Това НЕ значи, че в практиката няма да бъде удачно да се използва `auto` на даденото място.

59

Константен итератор

```
std::vector<int>::const_iterator cit;

for (cit = v.cbegin(); cit != v.cend(); ++cit)
{
    std::cout << *cit << "\n"; // OK
    *cit = 0; // грешка! (би работило за iterator)
}
```

60

Преобразуване (cast) на итератори

- `iterator` се преобразува до `const_iterator`, но не и обратното:

```
std::vector<int>::iterator it;  
std::vector<int>::const_iterator cit;  
  
cit = it; // OK  
it = cit; // грешка!
```

61


Следствие

```
std::vector<int>::iterator it;  
std::vector<int>::const_iterator cit;  
  
// OK  
for (cit = v.begin(); cit != v.end(); ++cit)  
    std::cout << *cit << "\n";  
  
// Грешка!  
for (it = v.cbegin(); it != v.cend(); ++it)  
    std::cout << *it << "\n";
```

62

Възможна грешка: смесване на итератори

```
for (it = v.cbegin(); it != v.end(); ++it)
    std::cout << *it << "\n";
```



Погрешка е
изпуснато
едно 'с' в
началото

63

Упражнение: работа с вектор

Напишете програма, която получава на стандартния вход поредица от цели числа и след това:

1. Съхранява числата във вектор от подходящ тип
2. Извежда съдържанието на вектора на екрана
3. Извежда средното аритметично на числата на екрана

Реализирайте поне една от стъпките 2 и 3 с итератор

Съвет: за конвертиране на числата от символен низ към `int`, използвайте наготово функцията `atoi()`.

64

Инициализиране (1)

```
// ръчно запълване
std::vector<int> v;
v.push_back(0);
v.push_back(1);
v.push_back(2);
v.push_back(3);

// Braced initialization (C++11)
std::vector<int> v{0, 1, 2, 3};
```

65

Инициализиране (2)

```
// копиране на друг вектор
std::vector<int> vectorCopy(v);

// копиране с итератори...
std::vector<int> vectorRange(v.cbegin(), v.cend());

// ...то работи за различни колекции
std::list<int> l(v.cbegin(), v.cend());

// запълване с fill constructor - четири единици
std::vector<int> vectorFill(4, 1);
```

66

Извеждане (1)

```
template <class Iter>
void Print(Iter start, Iter end)
{
    for (; start != end; ++start)
        std::cout << *start;

    std::cout << "\n";
}
```

67

Извеждане (2)

```
Print(v.begin(), v.end());
Print(v.cbegin(), v.cend());
Print(l.begin(), l.end());

// Print работи не само с итератори
int arr[] = { 1, 2, 3, 4 };
Print(arr, arr+4);
```

68

Обратно итериране

```
// Извежда 3210
for (std::vector<int>::reverse_iterator it = v.rbegin();
     it != v.rend();
     ++it)
{
    std::cout << *it;
}
```

69

Random Access Iterator

```
std::vector<int>::iterator it = v.begin();

// ... запълваме вектора с 0, 1, 2, 3, 4, 5, 6

int elem;
elem = *it;           // elem == 0
elem = *(it + 5);     // elem == 5
elem = it[5];         // elem == 5

std::advance(it, 5);  // функцията работи за всички итератори
elem = *it;           // elem == 5
```

70

Категории итератори в STL

- **Input Iterator:** може да се използва за извличане на стойностите на елементите в колекцията, но не и да ги променя;
- **Output Iterator:** може да се използва за променяне на стойностите на елементите в колекцията, но не и да ги извлича;
- **Forward Iterator:** може и да чете и да променя;
- **Bidirectional Iterator:** може да се движи и напред и назад;
- **Random Access Iterator:** позволяват да се достъпи произволен друг елемент, който се намира на някакъв отстъп (offset) от текущата им позиция.

71

Категории итератори в STL

Итератор	Достъп до елементите	Посока на движение	Произволен достъп (random access)
Input	Read	Напред	Не
Output	Write	Напред	Не
Forward	Read/Write	Напред	Не
Bidirectional	Read/Write	Напред и назад	Не
Random Access	Read/Write	Напред и назад	Да

72

Итератори и колекции

Ненаредените колекции предлагат най-малкото forward итератори, но създателите на имплементацията могат да ги реализират и с bidirectional итератори.

list, set, multiset, map и multimap предлагат bidirectional итератори.
vector, deque и array предлагат random-access итератори.

73

end() НЕ ни дава валидна позиция!

```
std::vector<int> v;  
  
// ...запълваме вектора...  
  
std::vector<int>::iterator it;  
  
it = v.end();  
  
std::cout << *it // Грешка!!!  
          << std::endl;
```

74

end() не е като NULL!

```
std::vector<int> v;  
  
// ...запълваме вектора...  
  
std::vector<int>::iterator it;  
  
it = v.end();  
--it; // OK  
  
std::cout << *it // OK  
          << std::endl;
```

75

Други операции с вектори

Операция	Описание
insert(elem,pos)	Вмъква elem на дадена позиция pos
insert(elem,n,pos)	Вмъква n-копия на elem на дадена позиция
erase(pos)	Води до shift
reserve(n)	Резервира поне капацитет n
resize(n)	Може да намали или увеличи размера на вектора; Ако го увеличава, новите елементи се създават с default constructor;

76

std::list

- Моделира работата на свързан списък
- Основни свойства
- Предимства и недостатъци
- Примерна реализация (вкл. итератор):
<https://github.com/semerdzhiev/sdp-samples/tree/master/LinkedList>
- Източници:
<http://www.cplusplus.com/reference/list/list/>
<http://en.cppreference.com/w/cpp/container/list>

77

Разлики на list спрямо vector

- Имплементира двусвързан списък
- Няма достъп до произволен елемент
- Вмъкването и изтриването на елементи е бързо, стига да сте на нужната позиция (в противен случай $O(n)$ за достигането ѝ)
- Изтриването и вмъкването на инвалидират указатели и итератори към други елементи.
- Поддържа операции front(), push_front(), and pop_front()
- Няма операции за преоразмеряване или капацитет (но поддържа resize())
- Много допълнителни операции, като например remove, remove_if, splice, reverse и др. Поддържа обратно-вървящи итератори.

78

Въпроси за упражнение

Защо vector няма следните операции:

- push_front / pop_front

Защо list няма следните операции:

- at / предефиниран operator[]?
- reserve / shrink_to_fit

Защо forward_list няма следните операции:

- push_back / pop_back?

79

list::remove_if

```
bool isOdd(int n) { return n & 1; }

int main()
{
    std::list<int> l;

    // ...запълваме списъка с елементи...

    l.remove_if(isOdd); // Премахва нечетните числа от l
}
```

80

Други контейнери: Array и Deque

- Основни свойства
- Типична имплементация
- За повече информация:
 - Array: <http://www.cplusplus.com/reference/array/array/>
 - Deque: <http://www.cplusplus.com/reference/deque/deque/>

81

Кога какво да използваме

Как да изберем между четирите контейнера:

- Array
- Vector
- List
- Deque

Полезна статия:

- Bjarne Stroustrup - Are lists evil?
http://www.stroustrup.com/bs_faq.html#list

82

Стек и опашка

```
#include <stack>
#include <queue>
```

83

std::stack

Основните операции на структурата от данни се реализират от:

- `top()`
- `pop()`
- `push()`

Оптимизиран е за бързодействие, като е направен компромис с удобството на работа и безопасността.

`pop()` не връща стойност и не може да се използва в цикъл в конструкция от тип `while(s.pop(x))`.

За проверки за размера на стека се използват `empty()` и `size()`.

84

Пример

```
std::stack<int> st;

st.push(10);
st.push(20);

std::cout << st.top() << "\n";
st.pop();
std::cout << st.top() << "\n";
st.pop();

st.pop();           // runtime error!
std::cout << st.top(); // runtime error!
```

85

Стек и опашка като адаптери

- Стекът и опашката в STL са реализирани като адаптер върху някой от другите контейнери:
 - За стек може да се използват vector, deque или list;
 - За опашка може да се използват deque или list.
- Освен ако не е указано друго, по подразбиране и за двете се използва deque.
- За повече информация вижте:
<http://www.cplusplus.com/reference/queue/queue/>
<http://www.cplusplus.com/reference/stack/stack/>

87

Указване на контейнер

Ако искате да използвате друг контейнер, можете да го укажете, когато създавате стека/опашката:

```
#include <stack>
#include <vector>
```

```
int main()
{
    std::stack<int, std::vector<int> > st;
}
```

88

std::priority_queue

STL поддържа и клас за приоритетна опашка.

Той също се реализира като адаптер, но трябва да бъде върху клас, който поддържа достъп до произволен елемент (например vector или deque).

Контейнер по подразбиране е deque.

Вътрешно класът използва произволния достъп, за да поддържа пирамида (heap).

За повече информация вижте:

http://www.cplusplus.com/reference/queue/priority_queue/

89

Сортировки

```
#include <algorithm>
```

90

Основни бележки

- Устойчивост (stability)
- Сложност
 - Еднаква във всички случаи
 - Променлива (най-добър, общ, най-лош случай)
- Основни алгоритми
- Quicksort
 - Алгоритъм
 - Основни свойства
 - Имплементация

91

Въпроси за упражнение

С увеличаване на броя N на елементите, които се сортират, вероятността времевата сложност на бързото сортиране (quicksort) да бъде съществено по-голяма от $O(N \log N)$ намалява значително.

Вярно ли е това твърдение?

- Да
- Не
- Зависи

92

Нека изпълняваме бързо сортиране (quicksort) върху масив с N -елемента.

Делителния елемент (pivot) за всеки pass избираме да бъде първият елемент в частта от масива, която се сортира.

Кое/кои от следните условия водят до най-лошия случай на изпълнение - $O(N^2)$?

- A. На всеки pass се избира делителен елемент (pivot), който е прекалено малък (например най-малкият елемент в масива).
- B. На всеки pass се избира делителен елемент (pivot), който е прекалено голям (например най-големият елемент в масива).
- C. Всички (или почти всички) елементи в масива са еднакви
- D. Масивът е (почти) сортиран
- E. Масивът е (почти) сортиран в обратен ред

93

```
template <typename ElementType>
std::ostream& operator<<(std::ostream& out,
                        const std::vector<ElementType> & v)
{
    std::vector<ElementType>::const_iterator it = v.cbegin();

    out << *it;
    ++it;

    for (; it != v.cend(); ++it)
        out << ", " << *it;

    return out;
}
```

94

Функция sort()

```
std::vector<int> v{ 1, 0, 200, 10, -5, 30 };
std::cout << "Unsorted: " << v << "\n";

std::sort(v.begin() + 2, v.begin() + 5);
std::cout << "Partially sorted: " << v << "\n";

std::sort(v.begin(), v.end());
std::cout << "Fully sorted: " << v << "\n";
```

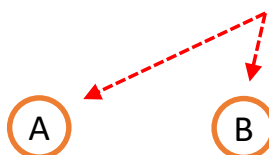
95

```
C:\WINDOWS\system32\cmd.exe
Unsorted: 1, 0, 200, 10, -5, 30
Partially sorted: 1, 0, -5, 10, 200, 30
Fully sorted: -5, 0, 1, 10, 30, 200
Press any key to continue . . .
```

96

sort() отблизко

Random-access iterator



```
std::sort(v.begin(), v.end());
```

Елементите се сортират:

от (A) включително до (B) неключително

sort() може да не е стабилна

*swap() трябва да може да работи върху
елементите на колекцията*

97

Потребителски дефинирани сравнения

```
bool LesserLastDigit(int A, int B)
{
    return A % 10 < B % 10;
}

int main()
{
    std::vector<int> v{1, 0, 200, 10, -5, 30};

    std::sort(v.begin(), v.end(), LesserLastDigit);
}
```

98

Допълнителни функции за сортиране

Функция	Описание
<code>stable_sort</code>	Устойчиво сортиране
<code>partial_sort(begin, middle, end)</code>	Частично сортиране; След изпълнението на функцията, в първите N позиции ($N = \text{middle} - \text{begin}$) ще са най-малки елементи от <code>[begin, end)</code> .
<code>nth_element(b,n,e)</code>	Пренарежда елементите така, че n-тият да е на мястото си
<code>is_sorted(begin,end)</code>	Проверява дали диапазона <code>[begin, end)</code> е сортиран.
<code>is_sorted_until(begin,end)</code>	Връща първата позиция в <code>[begin,end)</code> , която нарушава подредбата на елементите в нарастващ ред или <code>end</code> , ако няма такава.

За повече информация: <http://www.cplusplus.com/reference/algorithm/>

99

ВАЖНО!

Тъй като разгледаните функции са шаблонни, а итераторите имат същия синтаксис като на указателите, можете да правите например следното:

```
int arr[] = { 1, 5, 300, -1, 10 };

const int SIZE = 5;

std::sort(arr, arr + SIZE);
```

100

```
template <class InputIterator>
void print(InputIterator begin, InputIterator end)
{
    for (; begin != end; ++begin)
        std::cout << *begin << "\n";
}

int main()
{
    int arr[] = { 1, 2, 3, 4, 5 };
    std::vector<int> v{ 1, 2, 3, 4, 5 };
    print(arr, arr + 5);
    print(v.begin(), v.end());
}
```

101

Полезни функции

```
#include <algorithm>
```

102

Търсене

Функция	Описание
<code>find(b,e,v)</code>	Намира първото срещане на <code>v</code>
<code>find_if(b,e,p)</code> <code>find_if_not(b,e,p)</code>	Търсене на елемент с предикат
<code>count(b,e,v)</code>	Брой на срещанията на <code>v</code> във <code>[b,e)</code>
<code>search(b1,e1,b2,e2)</code>	Проверява дали <code>[b2,e2)</code> е подредица в <code>[b1,e1)</code> . Ако да, връща позицията на първото ѝ срещане; ако не, връща <code>e1</code> .
<code>find_end(b1,e1,b2,e2)</code>	Намира последното срещане
<code>find_first_of(b1,e1,b2,e2)</code>	Първо срещане на елемент от <code>[b2,e2)</code> в <code>[b1,e1)</code>
<code>binary_search(b,e,v)</code>	Двоично търсене (само за сортирани колекции)

За повече информация: <http://www.cplusplus.com/reference/algorithm/>

103

Пренареждане и запълване

Функция	Описание
<code>fill(b,e,v)</code>	Запълва [b,e) със <code>v</code>
<code>fill_n(b,n,v)</code>	Записва <code>n</code> -броя <code>v</code> , започвайки от <code>b</code>
<code>replace(b,e,l,v)</code>	Заменя всяко срещане на <code>l</code> със <code>v</code>
<code>reverse(b,e)</code>	Обръща наопаки [b, e)
<code>rotate(b,m,e)</code>	Лява ротация (<code>m</code> става първи елемент)
<code>random_shuffle(b,e)</code>	Пренарежда елементите в произволен ред
<code>generate(b,e,f)</code>	Запълва [b,e) със стойности, които се генерират от последователни извиквания на функцията <code>f</code> .

За повече информация: <http://www.cplusplus.com/reference/algorithm/>

104

Копиране

Функция	Описание
<code>copy(b1,e1,b2)</code>	Копира [b1,e1) на ново място, започващо на <code>b2</code> (Ако <code>b2</code> ∈ [b1,e1) да се използва <code>copy_backwards</code>)
<code>copy_backwards(b1,e1,e2)</code>	Копира отзад напред (Ако <code>e2</code> ∈ [b1,e1) да се използва <code>copy</code>)
<code>copy_n(b1,n,b2)</code>	Копира <code>n</code> -елемента в <code>b2</code> ; започва от <code>b1</code>
<code>copy_if(b1,e1,b2,p)</code>	filter операция Копира само елементи, за които предикатът <code>p</code> е истина;

За повече информация: <http://www.cplusplus.com/reference/algorithm/>

105

Операции с множества и редици

Функция	Описание
<code>equal(e1,b1,e2,b2)</code>	Дали двете редици са еднакви
<code>unique</code>	Премахва последователни повторения (напр. 1, 2, 2, 3, 3, 3, 2, 2 → 1, 2, 3, 2)
<code>set_union(e1,b1,e2,b2,r)</code>	Работят върху две колекции и връщат резултата в трета (r); Колекциите трябва да са сортирани
<code>set_intersection</code>	
<code>set_difference</code>	
<code>set_symmetric_difference</code>	

За повече информация: <http://www.cplusplus.com/reference/algorithm/>

106

map

Функция	Описание
<code>for_each(b,e,f)</code>	map операция Прилага f върху всички елементи в [b,e)
<code>transform(b,e,r,f)</code>	map операция Аналогично на <code>for_each</code> , но f не бива да променя елементите. Резултатът се поставя в друга колекция, започвайки от r
<code>transform(b1,e1,b2,e2,f,r)</code>	Аналогично на <code>transform</code> , но с двуместен map

За повече информация: <http://www.cplusplus.com/reference/algorithm/>

107

Извеждане на вектор

```
void printWithSpace(int x) {  
    std::cout << ' ' << x;  
}  
  
int main() {  
    // ...  
    for_each(v.cbegin(), v.cend(), printWithSpace);  
}
```

108

Чрез lambda

```
for_each(v.cbegin(),  
        v.cend(),  
        [](int x) { std::cout << ' ' << x; } );
```

109

```
int f(int x) { return x * 10; }

int main()
{
    std::vector<int> v1{ 1, 2, 3, 4 };

    std::vector<int> v2{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

    std::cout << "v1: " << v1 << "\n"
               << "v2: " << v2 << "\n";

    std::transform(v1.cbegin(), v1.cend(), v2.begin()+3, f);

    std::cout << "Transformed: " << v2 << "\n";
}
```

110

```
C:\WINDOWS\system32\cmd.exe
v1: 1, 2, 3, 4
v2: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Transformed: 1, 2, 3, 10, 20, 30, 40, 8, 9, 10
Press any key to continue . . .
```

111

fold/accumulate

STL дефинира и fold/accumulate операция:

- `accumulate(begin, end, init)`
- `accumulate(begin, end, init, op)`

За повече информация вижте:

- <http://en.cppreference.com/w/cpp/algorithm/accumulate>

112


Функтори

113

Увод

Функционален обект (Functional object) или функтор (Functor) е обект на клас, който дефинира `operator()`. Например:

```
class MyFunctor {  
public:  
    int operator()(int a, int b) {  
        return a + b;  
    }  
} f;
```

 Функционален обект

114

Употреба

```
// Обектът се създава по обичайния начин  
MyFunctor f;  
  
// Работата с него прилича на тази с функция  
int result = f(1, 2);
```

115

Няколко бележки

- Предимството на този подход е, че можем да енкапсулираме някаква сложна логика в класа, а този, който работи с неговите обекти да не може да го различи от обикновена функция;
- Функторът има състояние (функцията – не винаги);
- Затваряне (closure);
- lambda се реализира с функтори;

116

Как да направим функция, която едновременно работи с други функции или функтори?

```
// Отговор: като използваме шаблони
template <typename F>
int Process(int A, int B, F fn)
{
    return fn(A, B);
}
```

117

Още един пример

```
template <typename Iterator, typename F>
F MyForEach(Iterator begin, Iterator end, F fn)
{
    for (; begin != end; ++begin)
        fn(*begin);

    return fn;
}
```

118

Функтори в STL

Сравнение: greater, less, not_equal_to, greater_equal и др.

Логически: logical_not, logical_and и др.

Аритметични: plus, minus, divides, modulus и др.

Конвертиране: ptr_fun, ptr_mem и др.

За повече информация: <http://www.cplusplus.com/reference/functional/>

119

Пример

```
int arr[] = { 1, 5, 300, -1, 10 };  
const int SIZE = 5;  
  
// Сортира в намаляващ ред  
std::sort(arr, arr + SIZE, std::greater<int>());  
  
// Сортира в нарастващ ред  
std::sort(arr, arr + SIZE, std::less<int>());
```

120

Още за колекциите

121

map

- Имплементират асоциативна колекция (речник);
- Името идва от там, е ключът е свързан (mapped) със стойност;
- Сортирането и търсенето на елементи става по ключ;
- Ключовете в един map са константи – след като веднъж добавите двойка (ключ, стойност), ключът не може да се променя;
- При създаването на обект от тип map можете да укажете как да се сравняват ключовете (по подразбиране е `std::less<Key>`)
- Наредбата може да бъде частична или пълна;

122

Относно наредбата

- Наредбата, която се използва за сравняване на ключовете може да бъде частична или пълна;
- Въпреки това, колекцията поддържа съхранените в нея елементи в строго определен, последователен ред, в който всеки елемент има собствена, уникална позиция.

За повече информация: <https://www.sgi.com/tech/stl/StrictWeakOrdering.html>

123

Сравнение с други СД

- Обикновено се имплементира с двоично дърво;
- Обикновено по-бавна от `unordered_map`, но...
- ...позволява итериране на елементите по наредбата на ключовете;
- Може да има само един запис с даден ключ K.
- Ако трябва да може да се пазят няколко записа с еднакъв ключ K, може да се използва `multimap`;

124

Пример за употреба

```
std::map<char, int> codes;

codes.insert(std::pair<char, int>('a', 97));
codes['b'] = 98;

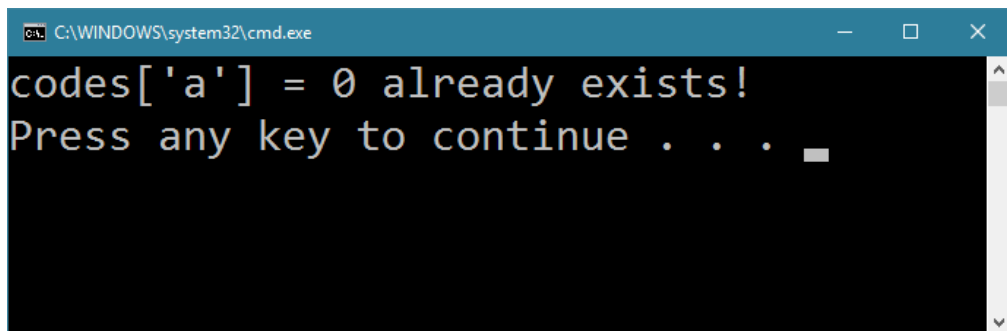
std::cout << "a:" << codes.at('a') << "\n"; // C++11
std::cout << "b:" << codes['b'] << "\n";
```

125

Колизия на ключ

```
codes.insert(std::pair<char, int>('a', 0));  
std::pair<std::map<char, int>::iterator, bool> rval;  
rval = codes.insert(std::pair<char, int>('a', 97));  
if (!rval.second) {  
    std::cout << "codes['" << rval.first->first  
               << "'] = " << rval.first->second  
               << " already exists!\n";  
}
```

126



```
C:\WINDOWS\system32\cmd.exe  
codes['a'] = 0 already exists!  
Press any key to continue . . .
```

127

Тогава как да променим записа?

```
// Всъщност е много лесно :-)
```

```
codes.insert(std::pair<char, int>('a', 0));
```

```
codes['a'] = 97; // ОК, променя записа
```

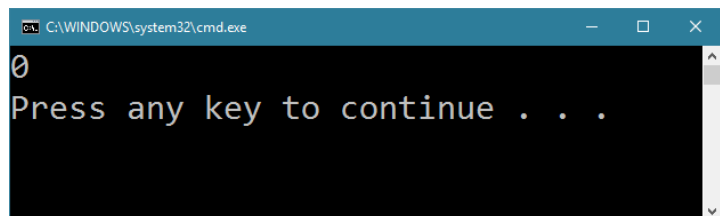
```
std::cout << codes['a'] << "\n"; // Извежда 97
```

128

```
std::map<char, int> codes;
```

```
// Какво ще се случи?
```

```
std::cout << codes['a'] << "\n";
```



129

Още операции с map

Функция	Описание
<code>find(key)</code>	Връща итератор към елемента или към <code>end()</code> , ако елемент с такъв ключ няма;
<code>erase(key)</code>	Изтрива елемента с ключ <code>key</code> , ако такъв има. Връща броя изтрити елементи (0 или 1); $O(\log N)$
<code>erase(it)</code>	Изтрива елемента сочен от <code>it</code> ; Амортизирана $O(1)$
<code>erase(begin, end)</code>	Изтрива елементите в <code>[begin, end)</code>
<code>count(key)</code>	Брой на елементите с ключ <code>key</code> ; $O(\log N)$
<code>begin, end, cbegin, ...</code>	Итераторите връщат наредени двойки!

За повече информация: www.cplusplus.com/reference/map/

130

Граници на елемент с ключ K

- Горна граница (`upper_bound`)
 - Елементът следващ K или `end()`;
- Долна граница (`lower_bound`)
 - Първият елемент, който НЕ Е преди K;
 - Ако колекцията съдържа K, това е самото K;
 - В противен случай това е първият елемент, който би бил след K, ако K беше в колекцията;

131

Insertion hinting (amortized $O(1)$ vs $O(\log N)$)

```
std::map<char, int> codes;
codes['a'] = 97;
codes['c'] = 99;
//...

std::map<char, int>::iterator it = codes.lower_bound('b');

if (it != codes.end() && !(codes.key_comp()('b', it->first)))
    it->second = 98;
else
    codes.insert(it, std::pair<char, int>('b', 98));
```

132

C++98 vs C++11

Според C++98, hint трябва да бъде позицията **преди** тази, на която ще се вмъква. Според C++11 е тази **след** нея;

Коректното поведение е това според C++11 (в C++98 е допусната грешка). За повече информация:

- <http://stackoverflow.com/questions/32758548/stdmap-insert-hint-location-difference-between-c98-and-c11>
- <http://cplusplus.github.io/LWG/lwg-defects.html#233>
- <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2005/n1780.html>

133

Повод за размисъл

Може ли ключовете в един `map` да бъдат от тип `double`?

Hint: Каква особеност има при сравняването на числа от тип `double` за това дали са еднакви?

Допълнителен въпрос: Може ли в `switch` оператор да `switch-ваме` по `double`?

134

set

- Моделира множество (всеки елемент може да се среща само по веднъж);
- Семантиката на много от операциите е сходна с тази на `map` – `lower_bound`, `upper_bound`, `find`, `erase` и т.н.;
- Отново реализацията обикновено използва дърво;
- Елементите могат да се итерират и всеки има своя позиция, въпреки, че наредбата им може да е частична;

135

string

```
#include <string>
```

136

String

- Представя символен низ
- За повече информация за имплементациите на string:
 - <http://info.prekert.com/blog/cpp-stdstring-implementations>

137

Йерархия

- `basic_string<T>`
 - `string`
 - `u16string`
 - `u32string`
 - `wstring`
- За конвертиране вижте например:
 - http://www.cplusplus.com/reference/locale/wstring_convert/

138

Относно символите

- Single-byte vs Multi-byte
- Multi-byte encoding
 - Fixed-width encoding
 - Variable-width encoding
- Wide character
 - Реално (по време на изпълнение) представяне на символите
- Важно:
 - `wchar_t` може да варира като размер; зависи от компилатора и съхранява символи, които в общия случай може да не са Unicode символи;

139

```
std::string str = "Helloworld!";
std::string::size_type pos;
pos = str.find('w');
// Важно: не сравнявайте int и npos!
if (pos == std::string::npos)
    std::cout << "Character not found!";
str[pos] = 'W';
str.insert(pos, ", ");
std::cout << str << "\n"; // Извежда "Hello, World!"
```

140

c_str() и data()

- И двете връщат масив от символи;
- Както в C++98, така и в C++11, c_str() връща C-style низ;
- data е по-особена:
 - В C++98 връща масив, който може да НЕ Е терминиран с '\0';
 - В C++11 прави същото като c_str().

141

Полезни функции (1)

Функция	Описание
substr	Връща нов string обект, в който е копирано парче от оригиналния обект (по подразбиране се копира целият низ) http://www.cplusplus.com/reference/string/string/substr/
append	Конкатенация http://www.cplusplus.com/reference/string/string/append/
compare	Сравнение (подобно на strcmp) http://www.cplusplus.com/reference/string/string/compare/
operator+ operator+=	Конкатенация

За повече информация: www.cplusplus.com/reference/map/

142

Полезни функции (2)

Функция	Описание
copy	Копира част от низа в масив http://www.cplusplus.com/reference/string/string/substr/
assign operator=	Присвояват нова стойност на обекта, като изтриват предишната
swap	Разменя съдържанието на низа с това на друг низ. По-бързо е от стандартното T=A, A=B, B=T
capacity	Заета памет
size	Размер на низа

За повече информация: www.cplusplus.com/reference/map/

143

Конвертиране

- Функции подобни на `atoi`, `atod`, ...:
 - `stoi`, `stod`, `stol`, ...
- Конвертиране на число до низ:
 - `to_string`, `to_wstring`

144

Някои бележки

1. Има ли разлика между дадените по-долу и кое е по-добре да използваме (кога)?

```
const std::string str = "Hello world!";  
const char buffer[] = "Hello world!";  
const char* pText = "Hello world!";
```

2. Small String Optimization (SSO)

145