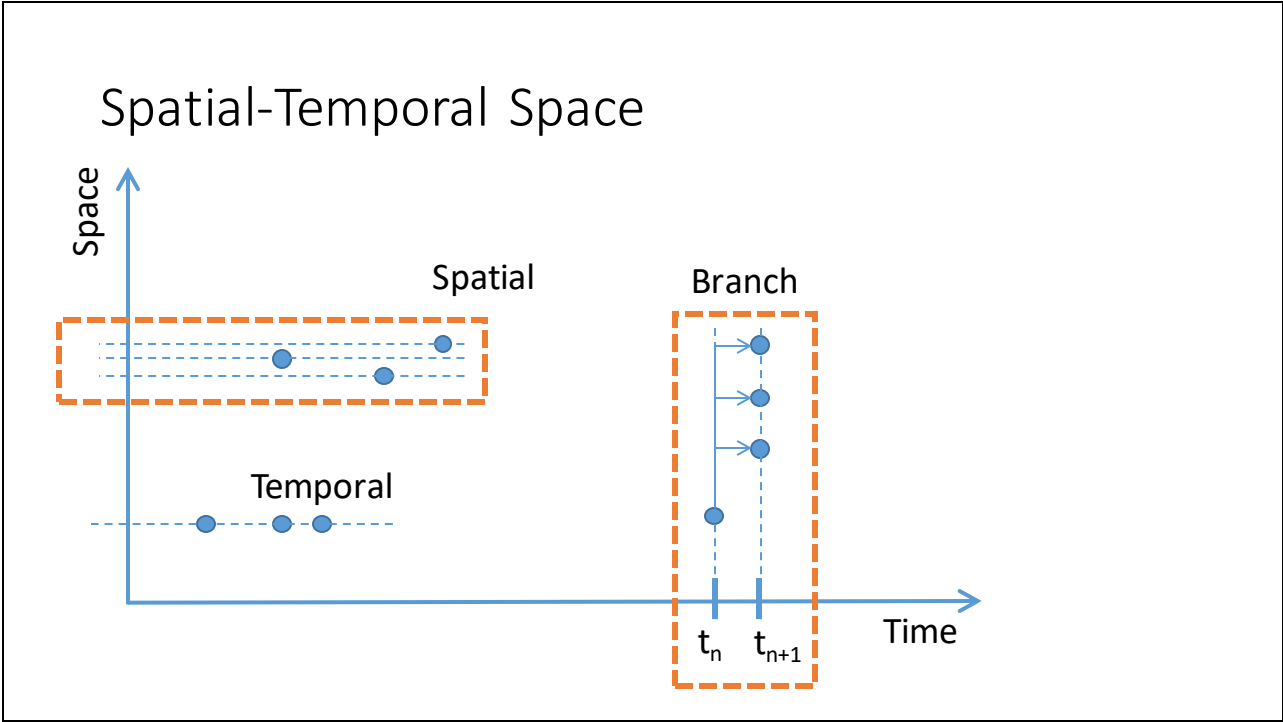


STL and Algorithms (Part 2)

Atanas Semerdzhiev

Locality

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



Сортировки

`#include <algorithm>`

Основни бележки

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

Минута (понякога не) е много

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

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

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

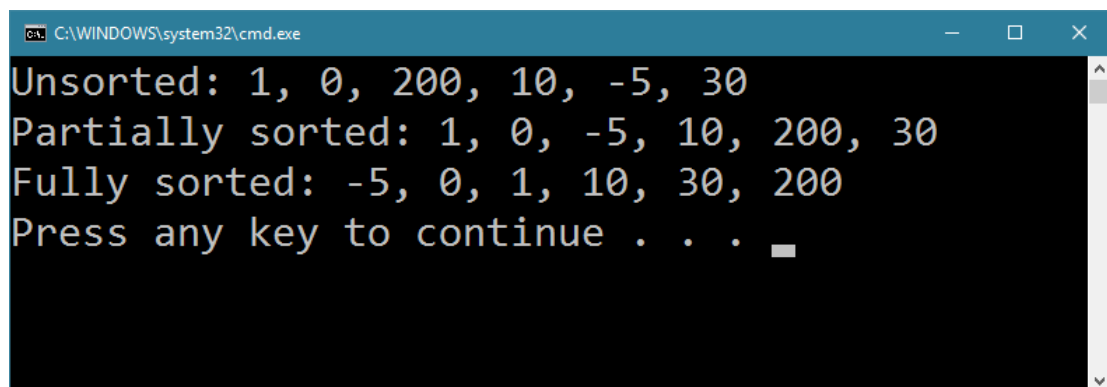
    out << *it;
    ++it;

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

    return out;
}
```

Функция sort()

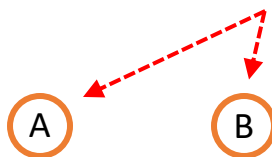
```
std::vector<int> v{ 1, 0, 200, 10, -5, 30 };  
std::cout << "Unsorted: " << v << "\n";  
  
std::sort(v.begin() + 2, v.begin() + 5);  
std::cout << "Partially sorted: " << v << "\n";  
  
std::sort(v.begin(), v.end());  
std::cout << "Fully sorted: " << v << "\n";
```



```
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 . . .
```

sort() отблизо

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*

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);
}
```

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 $[\text{begin}, \text{end})$.
<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 $[\text{begin}, \text{end})$ е сортиран.
<code>is_sorted_until(begin, end)</code>	Връща първата позиция в $[\text{begin}, \text{end})$, която нарушава подредбата на елементите в нарастващ ред или <code>end</code> , ако няма такава.

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

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);
```

```
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());
}
```

Полезни функции

```
#include <algorithm>
```


iter_swap()

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

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

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/>

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/>

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/>

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/>

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/>

Извеждане на вектор

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

Чрез lambda

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

```
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";
}
```

```
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 . . .
```

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>

Функтори

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



Употреба

// Обектът се създава по обичайния начин

```
MyFunctor f;
```

// Работата с него прилича на тази с функция

```
int result = f(1, 2);
```

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;

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);
}
```


Още един пример

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

    return fn;
}
```

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/>

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>());
```

Още за колекциите

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;

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>

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`;

Пример за употреба

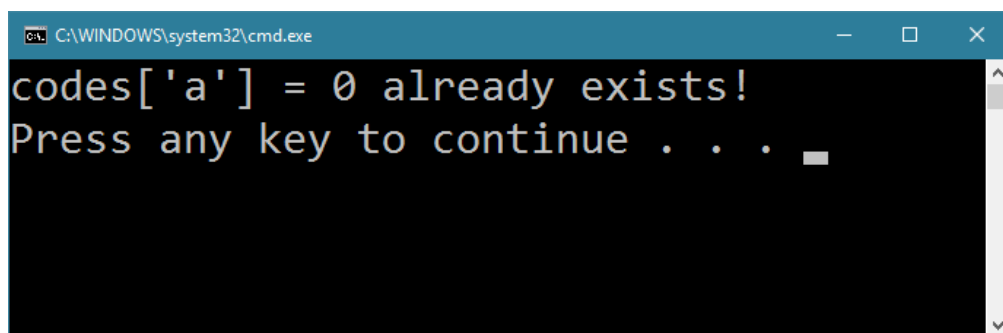
```
std::map<char, int> codes;

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";
```

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";  
}
```



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

Тогава как да променим записа?

```
// Всъщност е много лесно :-)
```

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

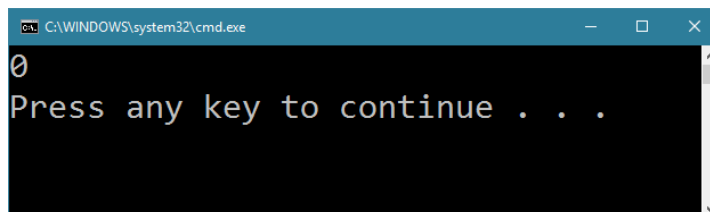
```
codes['a'] = 97; // OK, променя записа
```

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

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

```
// Какво ще се случи?
```

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



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/

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;

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));
```

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>

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`?

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;

string

```
#include <string>
```

String

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

Йерархия

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

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;

```
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!"
```

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().

Полезни функции (1)

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

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

Полезни функции (2)

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

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

Конвертиране

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