**Функционал std::string**

**Создание и удаление:**

**конструктор** — создает или копирует строку;

**деструктор** — уничтожает строку.

**Размер и ёмкость:**

**capacity()** — возвращает количество символов, которое строка может хранить без дополнительного перевыделения памяти;

**empty()** — возвращает логическое значение, указывающее, является ли строка пустой;

**length(), size()** — возвращают количество символов в строке;

**max\_size()** — возвращает максимальный размер строки, который может быть выделен;

**reserve()** — расширяет или уменьшает ёмкость строки.

**Доступ к элементам:**

   [], **at()** — доступ к элементу по заданному индексу.

**Изменение:**

   =, **assign()** — присваивают новое значение строке;

   +=, **append()**, **push\_back()** — добавляют символы к концу строки;

**insert()** — вставляет символы в произвольный индекс строки;

**clear()** — удаляет все символы строки;

**erase()** — удаляет символы по произвольному индексу строки;

**replace()** — заменяет символы произвольных индексов строки другими символами;

**resize()** — расширяет или уменьшает строку (удаляет или добавляет символы в конце строки);

**swap()** — меняет местами значения двух строк.

**Ввод/вывод:**

   >>, **getline()** — считывают значения из входного потока в строку;

   << — записывает значение строки в выходной поток;

**c\_str()** — конвертирует строку в строку C-style с нуль-терминатором в конце;

**copy()** — копирует содержимое строки (которое без нуль-терминатора) в массив типа char;

**data()** — возвращает содержимое строки в виде массива типа char, который не заканчивается нуль-терминатором.

**Сравнение строк:**

   ==, != — сравнивают, являются ли две строки равными/неравными (возвращают значение типа bool);

   <, <=, >, >= — сравнивают, являются ли две строки меньше или больше друг друга (возвращают значение типа bool);

**compare()** — сравнивает, являются ли две строки равными/неравными (возвращает -1, 0 или 1).

**Подстроки и конкатенация:**

   + — соединяет две строки;

**substr()** — возвращает подстроку.

**Поиск:**

**find** — ищет индекс первого символа/подстроки;

**find\_first\_of** — ищет индекс первого символа из набора символов;

**find\_first\_not\_of** — ищет индекс первого символа НЕ из набора символов;

**find\_last\_of** — ищет индекс последнего символа из набора символов;

**find\_last\_not\_of** — ищет индекс последнего символа НЕ из набора символов;

**rfind** — ищет индекс последнего символа/подстроки.

**Поддержка**[**итераторов**](https://ravesli.com/urok-198-iteratory-stl/)**и**[**распределителей**](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%BB%D0%BE%D0%BA%D0%B0%D1%82%D0%BE%D1%80)**(allocators):**

**begin()**, **end()** — возвращают «прямой» итератор, указывающий на первый и последний (элемент, который идет за последним) элементы строки;

**get\_allocator()** — возвращает распределитель;

**rbegin()**, **rend()** — возвращают «обратный» итератор, указывающий на последний (т.е. «обратное» начало) и первый (элемент, который предшествует первому элементу строки — «обратный» конец) элементы строки. Отличие от begin() и end() в том, что движение итераторов происходит в обратную сторону.

Хотя функционал достаточно широк, все же есть несколько заметных упущений:

   поддержка [**регулярных выражений**](https://ravesli.com/uroki-po-regexp/);

   конструкторы для создания строк из чисел;

   прописные/строчные функции;

   токенизация/разбиение строк на массивы;

   простые функции для получения левой или правой части строки;

   обрезка пробелов;

   конвертация из UTF-8 в UTF-16 и наоборот.

**Создание std::string из чисел**

Одно заметное упущение в классе std::string — это отсутствие возможности создавать строки из чисел. Например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | #include <iostream>  #include <string>    int main()  {      std::string sFive(5);        return 0;  } |

Здесь мы получим ошибку неудачной конвертации значения типа int в std::basic\_string. Самый простой способ конвертировать числа в строки — это **задействовать класс std::ostringstream**, который находится в [**заголовочном файле**](https://ravesli.com/urok-21-zagolovochnye-fajly/) sstream.

std::ostringstream уже настроен для приема разных входных данных: символов, чисел, строк и т.д.

А с помощью **std::istringstream** можно выполнять обратную конвертацию — выводить строки (либо через оператор вывода >>, либо через функцию str()).

Например, создадим std::string из разных входных данных:

|  |  |
| --- | --- |
|  | #include <iostream>  #include <sstream>  #include <string>    template <typename T>  inline std::string ToString(T tX)  {      std::ostringstream oStream;      oStream << tX;      return oStream.str();  }    int main()  {      std::string sFive(ToString(5));      std::string sSevenPointEight(ToString(7.8));      std::string sB(ToString('B'));      std::cout << sFive << std::endl;      std::cout << sSevenPointEight << std::endl;      std::cout << sB << std::endl;  } |

Результат:

5  
7.8  
B

Обратите внимание, здесь отсутствует проверка на ошибки. Может случиться так, что конвертация tX в std::string будет неудачной. В таком случае, хорошим вариантом было бы подключить генерацию исключения.

**Конвертация std::string в числа**

|  |  |
| --- | --- |
|  | #include <iostream>  #include <sstream>  #include <string>    template <typename T>  inline bool FromString(const std::string& sString, T &tX)  {      std::istringstream iStream(sString);      return (iStream >> tX) ? true : false; // извлекаем значение в tX, возвращаем true (если удачно) или false (если неудачно)  }    int main()  {      double dX;      if (FromString("4.5", dX))          std::cout << dX << std::endl;      if (FromString("TOM", dX))          std::cout << dX << std::endl;  } |

## Доступ к символам std::string

Есть два практически идентичных способа доступа к символам std::string. Наиболее простой и быстрый — использовать [**перегруженный оператор индексации []**](https://ravesli.com/urok-138-peregruzka-operatora-indeksatsii/).

### char& string::operator[](size\_type nIndex) const char& string::operator[](size\_type nIndex) const

   Обе эти функции возвращают символ под индексом nIndex.

   Передача неверного индекса приведет к неопределенным результатам.

   Использование [**функции length()**](https://ravesli.com/urok-202-dlina-i-yomkost-std-string/) в качестве индекса допустимо только для константных строк и возвращает значение, сгенерированное [**конструктором по умолчанию std::string**](https://ravesli.com/urok-201-sozdanie-unichtozhenie-i-konvertatsiya-std-string/). Это не рекомендуется делать.

   Поскольку char& — это тип возврата, то вы можете использовать его для изменения символов строки.

Например:

|  |  |
| --- | --- |
|  | #include <iostream>  #include <string>   int main()  {  std::string sSomething("abcdefg");  std::cout << sSomething[4] << std::endl;  sSomething[4] = 'A';  std::cout << sSomething << std::endl;  } |

Результат:

e  
abcdAfg

Другой способ доступа к символам std::string медленнее, чем вышеприведенный вариант, так как использует [**исключения**](https://ravesli.com/urok-181-isklyucheniya-zachem-oni-nuzhny/) для проверки корректности nIndex. Если вы не уверены в корректности передаваемого nIndex, то вы должны использовать именно этот способ (тот, что описан ниже) для доступа к символам строки.

### char& string::at(size\_type nIndex) const char& string::at(size\_type nIndex) const

   Обе эти функции возвращают символ под индексом nIndex.

   Передача неверного индекса приведет к генерации исключения out\_of\_range.

   Поскольку char& — это тип возврата, то вы можете использовать его для изменения символов строки.

## Конвертация std::string в строки C-style

Многие функции (включая все функции языка C++) ожидают форматирования строк как строк C-style, а не как std::string. По этой причине std::string предоставляет 3 разных способа конвертации std::string в строки C-style.

### const char\* string::c\_str() const

   Возвращает содержимое std::string в виде константной строки C-style.

   Добавляется нуль-терминатор.

   Строка C-style принадлежит std::string и не должна быть удалена.

Например:

|  |  |
| --- | --- |
|  | #include <iostream>  #include <string>    int main()  {  std::string sSomething("abcdefg");  std::cout << strlen(sSomething.c\_str());  } |

Результат:

7

### const char\* string::data() const

   Возвращает содержимое std::string в виде константной строки C-style.

   Не добавляется нуль-терминатор.

   Строка C-style принадлежит std::string и не должна быть удалена.

Например:

|  |  |
| --- | --- |
|  | #include <iostream>  #include <string>    int main()  { std::string sSomething("abcdefg");  const char \*szString = "abcdefg";  // Функция memcmp() сравнивает две вышеприведенные строки C-style и возвращает 0, если они равны  if (memcmp(sSomething.data(), szString, sSomething.length()) == 0)  std::cout << "The strings are equal";  else  std::cout << "The strings are not equal";  } |

Результат:

The strings are equal

### size\_type string::copy(char \*szBuf, size\_type nLength) const size\_type string::copy(char \*szBuf, size\_type nLength, size\_type nIndex) const

   Отличие второго варианта этой функции от первого состоит в том, что копирование не более nLength символов передаваемой строки в szBuf начинается с символа под индексом nIndex. В первой же функции копирование всегда начинается с символа под индексом [0].

   Количество скопированных символов возвращается.

   Caller отвечает за то, чтобы не произошло [**переполнения**](https://ravesli.com/urok-31-tselochislennyj-tip-dannyh-integer/#toc-4) строки szBuf.

Например:

|  |  |
| --- | --- |
|  |  |

Результат:

ipsum

Если вы не гонитесь за максимальной эффективностью, то c\_str() — это самый простой и безопасный способ конвертации std::string в строки C-style.

Вставлять символы/строки в **[std::string](https://ravesli.com/urok-200-std-string-i-std-wstring/" \t "_blank)** можно с помощью **функции insert()**.

**string& string::insert(size\_type index, const string& str)  
string& string::insert(size\_type index, const char\* str)**

   Обе функции вставляют символы/строки, начиная с определенного index std::string.

   Возвращают [**скрытый указатель \*this**](https://ravesli.com/urok-121-skrytyj-ukazatel-this/), что позволяет «связывать» объекты.

[**Генерируют исключение**](https://ravesli.com/urok-182-obrabotka-isklyuchenij/) out\_of\_range, если index некорректен.

   Генерируют исключение length\_error, если результат превышает максимально допустимое количество символов.

   Во второй версии функции insert() str не должен быть NULL.

Например:

|  |  |
| --- | --- |
|  | #include <iostream>  #include <string>    int main()  {  std::string sString("bbb");  std::cout << sString << std::endl;    sString.insert(2, std::string("mmm"));  std::cout << sString << std::endl;    sString.insert(5, "aaa");  std::cout << sString << std::endl;  } |

Результат:

bbb  
bbmmmb  
bbmmmaaab

А вот версия функции insert(), которая позволяет вставить с определенного index std::string подстроку.

**string& string::insert(size\_type index, const string& str, size\_type startindex, size\_type num)**

   Эта функция вставляет с определенного index std::string указанное количество символов (num) строки str, начиная со startindex-а.

   Возвращает скрытый указатель \*this, что позволяет «связывать» объекты.

   Генерирует исключение out\_of\_range, если index или startindex некорректны.

   Генерирует исключение length\_error, если результат превышает максимально допустимое количество символов.

Например:

|  |  |
| --- | --- |
|  | #include <iostream>  #include <string>    int main()  {  std::string sString("bbb");    const std::string sInsert("012345");  sString.insert(1, sInsert, 2, 4);  // вставляем подстроку sInsert длиной 4, начиная с символа под индексом 2,  // в строку sString, начиная с индекса 1  std::cout << sString << std::endl;  } |

Результат:

b2345bb

А вот версия функции insert(), с помощью которой в std::string можно вставить часть [**строки C-style**](https://ravesli.com/urok-79-stroki-c-style/).

**string& string::insert(size\_type index, const char\* str, size\_type len)**

   Эта функция вставляет с определенного index std::string указанное количество символов (len) строки C-style str.

   Возвращает скрытый указатель \*this, что позволяет «связывать» объекты.

   Генерирует исключение out\_of\_range, если index некорректен.

   Генерирует исключение length\_error, если результат превышает максимально допустимое количество символов.

   Игнорирует специальные символы (такие как ").

Например:

|  |  |
| --- | --- |
|  | #include <iostream>  #include <string>    int main()  {  std::string sString("bbb");    sString.insert(2, "acdef", 4);  std::cout << sString << std::endl;  } |

Результат:

bbacdeb

А вот версия функции insert(), которая вставляет в std::string один и тот же символ несколько раз.

**string& string::insert(size\_type index, size\_type num, char c)**

   Эта функция вставляет с определенного index std::string указанное количество вхождений (num) символа c.

   Возвращает скрытый указатель \*this, что позволяет «связывать» объекты.

   Генерирует исключение out\_of\_range, если index некорректен.

   Генерирует исключение length\_error, если результат превышает максимально допустимое количество символов.

Например:

|  |  |
| --- | --- |
|  | #include <iostream>  #include <string>    int main()  {  std::string sString("bbb");    sString.insert(2, 3, 'a');  std::cout << sString << std::endl;  } |

Результат:

bbaaab

И, наконец, функция insert() имеет три разные версии, которые работают с [**итераторами**](https://ravesli.com/urok-198-iteratory-stl/).

**void insert(iterator it, size\_type num, char c)  
iterator string::insert(iterator it, char c)  
void string::insert(iterator it, InputIterator begin, InputIterator end)**

   Первая версия функции вставляет в std::string указанное количество вхождений (num) символа c перед итератором it.

   Вторая версия функции вставляет в std::string одиночный символ c перед итератором it и возвращает итератор в позицию вставленного символа.

   Третья версия функции вставляет в std::string все символы диапазона (begin, end) перед итератором it.

   Все функции генерируют исключение length\_error, если результат превышает максимально допустимое количество символов.

Самый эффективный способ - просто перебирать строку, пока не найдете нецифровой символ. Если есть какие-либо нецифровые символы, вы можете считать строку не числом.

bool is\_number(const std::string& s)

{

std::string::const\_iterator it = s.begin();

while (it != s.end() && std::isdigit(\*it)) ++it;

return !s.empty() && it == s.end();

}

Или, если вы хотите сделать это способом C ++ 11:

bool is\_number(const std::string& s)

{

return !s.empty() && std::find\_if(s.begin(),

s.end(), [](unsigned char c) { return !std::isdigit(c); }) == s.end();

}

Как указано в комментариях ниже, это работает только для положительных целых чисел. Если вам нужно обнаружить отрицательные целые числа или дроби, вам следует выбрать более надежное решение на основе библиотеки. Хотя добавление поддержки отрицательных целых чисел довольно тривиально.

РАЗБИЕНИЕ НА ТОКЕНЫ

#include <iostream>

#include <vector>

#include <fstream>

#include <Windows.h>

#include <string>

using namespace std;

int main()

{

SetConsoleOutputCP(1251);

ifstream fin("Text.txt");

std::vector<std::string> token;

std::string s;

while (fin >> s)

token.push\_back(s);

for (int i = 0; i < token.size(); ++i)

cout << "Token[" << i << "] = " << token[i] << endl;

}