

ТИП данных string

- В программировании, строковый тип (англ. string «нить, вереница») — тип данных, значениями которого является произвольная последовательность (строка) символов алфавита. Каждая переменная такого типа (строковая переменная) может быть представлена фиксированным количеством байтов либо иметь произвольную длину.

Тип	Размер в байтах	Диапазон значений
char	1	от -128 до 126
unsigned char	1	от 0 до 255
short	2	от -32 768 до 32767
unsigned short	2	от 0 до 65536
enum	2	от -2 147 483 648 до 2 147 483 647
long	4	от -2 147 483 648 до 2 147 483 647
unsigned long	4	от 0 до 4 294 967 295
int	4	от -2 147 483 648 до 2 147 483 647
unsigned int	4	от 0 до 4 294 967 295
float	4	от $3,4 * 10^{-38}$ до $3,4 * 10^{38}$
double	8	от $1,7 * 10^{-308}$ до $3,4 * 10^{308}$
long double	10	от $3,4 * 10^{-4932}$ до $1,1 * 10^{4932}$
bool	1	true или false

<https://radioprogram.ru/post/1320>

- В какой области памяти хранится `s` (все в стеке или в стеке только указатель, а основная часть в куче), каков ее размер (всегда переопределяется как у динамического массива или выделяется сразу много места), хранятся ли там только символы в привычном однобайтном виде или есть еще какая-то информация о строке, в каком порядке хранятся байты (от младшего к старшему или наоборот)???
- Рассмотрим выделение памяти под строку

- После того, как вы создали строку, часто бывает полезно узнать ее длину. Здесь в игру вступают операции с длиной и емкостью.
- Длина строки
- Длина строки – это довольно просто, это количество символов в строке. Для определения длины строки есть две идентичные функции:
- `size_type string::length() const`
- `size_type string::size() const`
- Обе эти функции возвращают текущее количество символов в строке, исключая завершающий ноль.
- Пример кода:
- `string sSource("012345678");`
- `cout << sSource.length() << endl;`
- Вывод:
- 9

- Хотя, чтобы определить, есть ли в строке какие-либо символы или нет, можно использовать `length()`, но более эффективно использовать функцию `empty()`:
- `bool string::empty() const`
- Возвращает `true`, если в строке нет символов, иначе – `false`.
- Пример кода:
 - `string sString1("Not Empty");`
 - `cout << (sString1.empty() ? "true" : "false") << endl;`
 - `string sString2; // пустая`
 - `cout << (sString2.empty() ? "true" : "false") << endl;`
 - Вывод:
 - `false`
 - `true`

- Есть еще одна функция, связанная с размером, которую вы, вероятно, никогда не будете использовать, но мы опишем ее здесь для полноты картины:
- `size_type string::max_size() const`
- Возвращает максимальное количество символов, которое может содержать строка. Это значение будет варьироваться в зависимости от операционной системы и архитектуры системы.
- Пример кода:
 - `string sString("MyString");`
 - `cout << sString.max_size() << endl;`
 - Вывод:
 - 4294967294

- Емкость строки
- Емкость (вместимость) строки показывает, сколько памяти выделено объектом строки для хранения ее содержимого. Это значение измеряется в строковых символах, исключая символ завершающего нуля. Например, строка с емкостью 8 может содержать 8 символов.

- `size_type string::capacity() const`
- Возвращает количество символов, которое строка может хранить без перераспределения памяти.

- Пример кода:
 - `string sString("01234567");`
 - `cout << "Length: " << sString.length() << endl;`
 - `cout << "Capacity: " << sString.capacity() << endl;`
 - Вывод:
 - Length: 8
 - Capacity: 15

- Обратите внимание, что емкость больше, чем длина строки! Хотя длина нашей строки равна 8, на самом деле она занимала достаточно памяти для 15 символов! Зачем так сделано?
- Здесь важно понимать, что если пользователь хочет поместить в строку больше символов, чем позволяет ее емкость, то для получения большей емкости строка должна быть перераспределена в памяти. Например, если строка имеет длину и емкость 8, то добавление любых символов в строку приведет к перемещению объекта в памяти. Сделав емкость больше размера фактической строки, пользователь получил некоторое буферное пространство для расширения строки до необходимости перераспределения.
- Как оказалось, перераспределение – это плохо по нескольким причинам:
 - Во-первых, перераспределение строки относительно дорого. Сначала необходимо выделить новую память. Затем каждый символ в строке необходимо скопировать в новую память. Если строка большая, это может занять много времени. Наконец, необходимо освободить старую память. Если вы выполняете много перераспределений, этот процесс может значительно замедлить работу вашей программы.
 - Во-вторых, всякий раз, когда строка перераспределяется, адрес содержимого строки в памяти изменяется на новое значение. Это означает, что все ссылки, указатели и итераторы строки становятся недействительными!

- Обратите внимание, что строки не всегда размещаются с емкостью, превышающей длину. Рассмотрим следующую программу:
- `string sString("0123456789abcde");`
- `cout << "Length: " << sString.length() << endl;`
- `cout << "Capacity: " << sString.capacity() << endl;`
- Эта программа выводит:
- Length: 15
- Capacity: 15
- Результаты могут отличаться в зависимости от компилятора.

- Давайте добавим к строке один символ и посмотрим, как изменится емкость:

- `string sString("0123456789abcde");`
- `cout << "Length: " << sString.length() << endl;`
- `cout << "Capacity: " << sString.capacity() << endl;`

- `// Теперь добавим новый символ`
- `sString += "f";`
- `cout << "Length: " << sString.length() << endl;`
- `cout << "Capacity: " << sString.capacity() << endl;`
- Это дает следующий результат:

- Length: 15
- Capacity: 15
- Length: 16
- Capacity: 31

Если вы заранее знаете, что собираетесь создать большую строку, выполняя множество строковых операций, которые увеличивают размер строки, вы можете избежать многократного перераспределения строки в памяти, сразу установив для строки необходимую ей емкость:

- `#include <iostream>`
- `#include <string>`
- `#include <cstdlib> // для rand() и srand()`
- `#include <ctime> // для time()`

- `using namespace std;`

- `int main()`
- `{`
- `std::srand(std::time(nullptr)); // инициализация генератора случайных чисел`

- `string sString{}; // длина 0`
- `sString.reserve(64); // резервируем 64 символа`

- `// Заполняем строку случайными строчными буквами`
- `for (int nCount{ 0 }; nCount < 64; ++nCount)`
- `sString += 'a' + std::rand() % 26;`

- `cout << sString;`
- `}`
- Результат этой программы будет меняться каждый раз. Вот результат одного выполнения:

- `wzpzujwuaokbakgijqdawvzjqlgpciriiiuxhyfkdpkpxyucvytvxwqsbtiehxpy`
- Вместо того чтобы перераспределять `sString` несколько раз, мы устанавливаем емкость один раз, а затем заполняем строку. Это может очень влиять на производительность при формировании больших строк с помощью конкатенации.

Строки типа string

Тип `string` предназначен для работы со строками символов в кодировке Unicode. Ему соответствует базовый класс `System.String` библиотеки .NET.

Создание строки:

1. `string s;` // инициализация отложена
 2. `string t = "qqq";` // инициализация строковым литералом
 3. `string u = new string(' ', 20);` // с пом. конструктора
 4. `string v = new string(a);` // создание из массива символов
- // создание массива символов: `char[] a = { '0', '0', '0' };`

Библиотека <string>

Функции библиотеки <string>:

<code>s.length()</code> ;	возвращает длину строки <i>s</i>
<code>s.find(str)</code> ;	возвращает индекс первого вхождения строки <i>str</i> в <i>s</i>
<code>s.rfind(str)</code> ;	возвращает индекс последнего вхождения строки <i>str</i> в <i>s</i>
<code>s.find_first_of(str)</code> ;	возвращает индекс первого вхождения любого символа из <i>str</i> в <i>s</i>
<code>s.find_last_of(str)</code> ;	возвращает индекс последн. вхождения любого символа из <i>str</i> в <i>s</i>
<code>s.substr(k, n)</code> ;	возвращает строку из <i>n</i> символов строки <i>s</i> , начиная с <i>k</i> -ого
<code>s.insert(k, str)</code> ;	вставляет строку <i>str</i> в строку <i>s</i> , начиная с <i>k</i> -ого символа
<code>s.erase(k, n)</code> ;	удаляет <i>n</i> символов в строке <i>s</i> , начиная с <i>k</i> -ого

Операции со строками

`s.empty()`

Возвращает `true`, если строка пуста

`s.size()`

Возвращает количество символов в строке

`s[n]`

Возвращает `n`-ый символ строки

`s1 + s2`

Возвращает «склейку» строк `s1` и `s2`

`s1 = s2`

Заменяет символы строки `s1` строкой `s2`

`s1 == s2`

Проверяет совпадение строк

`!=, <, <=, >, >=`

Имеют обычное значение

Операции со строками

<i>s.append(str)</i>	добавляет в конец строки строку str. Можно писать как s.append(переменная), так и s.append("строка")
<i>s.assign(str)</i>	присваивает строке s значение строки str. Аналогично записи s=str
<i>int i=s.begin()</i>	записывает в i индекс первого элемента строки
<i>int i=s.end()</i>	аналогично, но последнего
<i>s.clear()</i>	как следует из названия, очищает строку. Т.е. удаляет все элементы в ней
<i>s.compare(str)</i>	сравнивает строку s со строкой str и возвращает 0 в случае совпадения (на самом деле сравнивает коды символов и возвращает их разность)
<i>s.copy</i> (куда, сколько, начиная с какого)	- копирует из строки s в куда (там может быть как строка типа string, так и строка типа char). Последние 2 параметра не обязательные (можно использовать функцию с 1,2 или 3 параметрами)

Строковый тип данных

Пример использования переменной строкового типа:

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    string s, ss; // объявление переменных s и ss строкового типа
    cin >> s; // Считывание строки s до первого разделителя
    getline(cin, ss); // Функция getline() считывает все введённые символы
                    // с пробелами до тех пор, пока не будет нажата
    // клавиша Enter
    gets(s); // Полный аналог функции getline()
    cout << s; // Вывод строки s в консоль
}
```

Работа со строками в C++

Библиотека <string>

Функция `.insert(k, str)` вставляет в строку `ss` начиная с k -ого символа строку `str`.

```
#include <string>
using namespace std;
int main() {
    string ss = "Ой, нужен , ой";
    ss.insert(10, "ХанаХ");
    //ss = "Ой, нужен ХанаХ, ой"
    return 0;
}
```

- `#include <iostream>`
- `#include <string>`
- `using namespace std;`
- `int main () {`
- `string str1 = "Привет";`
- `string str2 = "Мир";`
- `string str3;`
- `int len;`

- `// копируем str1 в str3`
- `str3 = str1;`
- `cout << "str3:" << str3 << endl;`
- `// объединяет str1 и str2`
- `str3 = str1 + str2;`
- `cout << "str1 + str2:" << str3 << endl;`
- `// общая длина str3 после конкатенации`
- `len = str3.size ();`
- `cout << "str3.size ():" << len << endl;`
-
- `return 0;`
- `}`

- Лотосы
- #include <iostream>
- #include<string>
- using namespace std;
- int main()
- { int i;
- string s1 = "Розовые лотосы удивительно красивы";
- string s2 = "можно ";
- string s3 = "есть ";
- string s4 = ", кажется,";
- //string s5=s1+" "+s2+" "+s3+" "+s4;
- //cout <<s5<<endl;
- //cout <<s1<< endl;
- s1.erase(29,22);
- cout <<s1<< endl;
- s1.replace (29,13,s2);
- cout <<s1<< endl;
- s1.append(s3);
- cout <<s1<< endl;
- s1.insert(28,s4);
- cout <<s1<< endl;
- i=s1.size ();
- cout <<i<< endl;
- // cout<<"Hello World";
- return 0;
- }

Дана Фраза: Розовые лотосы удивительно
красивы.

<https://server.179.ru/tasks/cpp/total/161.html>

- Создайте фразу : Розовые лотосы, кажется, можно есть.

