Строки C++

Функции поиска

- find - поиск символов в строке
- rfind - поиск последнего вхождения подстроки
- find_first_of - поиск первого вхождения символов
- find_first_not_of - найти первое вхождение отсутствия символов
- find_last_of - найти последнее вхождение символов
- find_last_not_of - найти последнее вхождение отсутствия символов

std::basic_string::find

size_type find(const basic_string& str, size_type pos = 0) const;	(1)
size_type find(const CharT* s, size_type pos, size_type count) const;	(2)
size_type find(const CharT* s, size_type pos = 0) const;	(3)
size_type find(CharT ch, size_type pos = 0) const;	(4)

Находит первую подстроку, равную переданной последовательности символов. Поиск начинается с позиции pos, т.е. найденная подстрока не может начинаться в позиции, предшествующей pos.

1) Находит первую подстроку, равную str.

2) Находит первую подстроку, равную первым count символам строки, на которую указывает s. s может включать нулевые символы.

3) Находит первую подстроку, равную символьной строке, на которую указывает s. Длина строки определяется по первому вхождению нулевого символа.

4) Находит первый символ ch.

Параметры

str	строка для поиска
count	длина подстроки для поиска
s	указатель на строку поиска
ch	символ для поиска

Возвращаемое значение

позиция первого символа найденной подстроки или **npos**, если подстрока не найдена.

npos – это специальное значение, равное максимальному значению, которое может предоставить тип size_type. Точный смысл данного значения зависит от контекста, но, как правило, оно используется либо как индикатор конца строки в функциях, которые ожидают позицию символа, либо как индикатор ошибки в функциях, которые возвращают позицию в строке.

Пример. find

```
#include <string>
#include <iostream>

void print(std::string::size_type n, std::string const &s)
{
	if (n == std::string::npos) {
		std::cout << "not found\n";
	}
	else {
		std::cout << "found: " << s.substr(n) << '\n';
	}
}

int main()
{
	std::string::size_type n;
	std::string const s = "This is a string";

	// поиск с начала строки
	n = s.find("is");
	print(n, s);

	// поиск с позиции 5
	n = s.find("is", 5);
	print(n, s);

	// поиск первого символа
	n = s.find('a');
	print(n, s);

	// поиск первого символа
	n = s.find('q');
	print(n, s);
}
```


std::basic_string::rfind

size_type rfind(const basic_string& str, size_type pos = npos) const;	(1)
size_type rfind(const CharT* s, size_type pos, size_type count) const;	(2)
size_type rfind(const CharT* s, size_type pos = npos) const;	(3)
size_type rfind(CharT ch, size_type pos = npos) const;	(4)

Находит последнюю подстроку, равную переданной символьной последовательности. Поиск начинается с позиции pos, т.е. в поиске участвует только подстрока в диапазоне [pos, size). Если npos передано в качестве позиции pos, поиск будет произведен по всей строке.

1) Находит последнюю подстроку, равную str.

2) Находит последнюю подстроку, равную первым count символам строки, на которую указывает s. s может включать нулевые символы.

3) Находит последнюю подстроку, равную символьной строке, на которую указывает s. Длина строки определяется по первому вхождению нулевого символа.

4) Находит последний символ ch.

Параметры

str	строка для поиска
count	длина подстроки для поиска
s	указатель на строку поиска
ch	символ для поиска

Возвращаемое значение

позиция первого символа найденной подстроки или npos, если подстрока не найдена.

Пример. rfind

```
#include <string>
#include <iostream>

void print(std::string::size_type n, std::string const &s)
{
	if (n == std::string::npos)
		std::cout << "not found\n";
	else
		std::cout <<n<< " found: " << s.substr(n) << '\n';
}

int main()
{
	std::string::size_type n;
	//                     0123456789*123456789*1
	std::string const s = "This is a is string is";

	// поиск в обратном направлении, начиная с конца строки
	n = s.rfind("is");
	print(n, s);
	// поиск с позиции 2
	n = s.rfind("is", 2);
	print(n, s);
	// поиск первого символа, начиная с конца строки
	n = s.rfind('s');
	print(n, s);
	// поиск первого символа, начиная с конца строки
	n = s.rfind('q');
	print(n, s);
}
```


Функции поиска. Задания

1. Изучите описания функций, расположенные по ссылкам:
   a. http://ru.cppreference.com/w/cpp/string/basic_string/find_first_of
   b. http://ru.cppreference.com/w/cpp/string/basic_string/find_first_not_of
   c. http://ru.cppreference.com/w/cpp/string/basic_string/find_last_of
   d. http://ru.cppreference.com/w/cpp/string/basic_string/find_last_not_of

   Составьте примеры для каждой перегруженной функции.

Операции над строкой

- clear - очищает содержимое строки
- insert - вставка символов
- erase - удаление символов
- push_back - добавление символа в конец строки
- pop_back - удаляет последний символ
- append - добавляет символы в конец строки
- operator+= - добавляет символы в конец строки
- compare - сравнивает две строки
- replace - заменяет каждое вхождение указанного символа
- substr - возвращает подстроку
- resize - изменяет количество хранимых символов
- swap - обменивает содержимое

Пример. clear

```
#include <iostream>
#include <string>
//Эта программа повторяет каждую строку,
//введенную пользователем, пока строка не будет содержать точку('.').
//Каждый символ новой строки('\ n') запускает повторение строки
//и очистку текущего содержимого строки.
int main()
{
	char c;
	std::string str;
	std::cout << "Please type some lines of text. Enter a dot (.) to finish:\n";
	do {
		c = std::cin.get();
		str += c;
		if (c == '\n')
		{
			std::cout << str;
			str.clear();
		}
	} while (c != '.');
	return 0;}
```

Пример. erase

```
#include <iostream>
#include <string>
int main()
{
	std::string str("This is an example sentence.");
	std::cout << str << '\n';
	// "This is an example sentence."
	str.erase(10, 8);
	std::cout << str << '\n';
	// "This is an sentence."
	str.erase(str.begin() + 9);
	std::cout << str << '\n';
	// "This is a sentence."
	str.erase(str.begin() + 5, str.end() - 9);
	std::cout << str << '\n';
	// "This sentence."
	return 0;
}
```

Пример. pop_back

```
#include <iostream>
#include <string>

int main()
{
	std::string str("hello world!");
	str.pop_back();
	std::cout << str << '\n';
	return 0;
}
```

Пример. push_back

```
/*Читается файл по символу,
каждый символ добавляется к строковому объекту
с помощью push_back*/.
#include <iostream>
#include <fstream>
#include <string>

int main()
{
	std::string str;
	std::ifstream file("test.txt", std::ios::in);
	if (file) {
		while (!file.eof()) str.push_back(file.get());
	}
	std::cout << str << '\n';
	return 0;
}
```


Пример. append

```
#include <iostream>
#include <string>

int main()
{
	std::string str;
	std::string str2 = "Writing ";
	std::string str3 = "print 10 and then 5 more";
	// всю строку str2 добавляем в str:
	str.append(str2);                // "Writing «
                                                  // с 6-й позиции 3 символа
	str.append(str3, 6, 3);          // "10 “
                                                  // 5 символов в самое начало
	str.append("dots are cool", 5); // "dots "
	str.append("here: ");            // "here: "
	str.append(10, '.');            // "..........«
                                                 // " and then 5 more"
	str.append(str3.begin() + 8, str3.end());_str.append(5u, '.');             // "....."

	std::cout << str << '\n';
	return 0;
}
```

Пример. compare

```
#include <iostream>
#include <string>

int main()
{
	std::string str1("green apple");
	std::string str2("red apple");
	// сравнивает str1 и str2
	// 0, когда они равны
	if (str1.compare(str2) != 0)
		std::cout << str1 << " is not " << str2 << '\n';
	// с 6-й позиции 5 символов
	if (str1.compare(6, 5, "apple") == 0)
		std::cout << "still, " << str1 << " is an apple\n";

	if (str2.compare(str2.size() - 5, 5, "apple") == 0)
		std::cout << "and " << str2 << " is also an apple\n";

	// с 6-й позиции 5 символов для str1
	// с 4-й позиции 5 символов для str2
	if (str1.compare(6, 5, str2, 4, 5) == 0)
		std::cout << "therefore, both are apples\n";

	return 0;
}
```

Пример. replace

```
#include <iostream>
#include <string>

int main()
{
	std::string base = "this is a test string.";
	std::string str2 = "n example";
	std::string str3 = "sample phrase";
	std::string str4 = "useful.";

	// replace signatures used in the same order as described above:

	// Using positions:                 0123456789*123456789*12345
	std::string str = base;           // "this is a test string."
	str.replace(9, 5, str2);          // "this is an example string."   (1)
	str.replace(19, 6, str3, 7, 6);    //"this is an example phrase." (2)
	str.replace(8, 10, "just a");     // "this is just a phrase."       (3)
	str.replace(8, 6, "a shorty", 7);  //"this is a short phrase."     (4)
	str.replace(22, 1, 3, '!');        //"this is a short phrase!!!"   (5)

// Using iterators:                                     0123456789*123456789*
	str.replace(str.begin(), str.end() - 3, str3);// "sample phrase!!!"

	str.replace(str.begin(), str.begin() + 6, "replace");// "replace phrase!!!"

	str.replace(str.begin() + 8, str.begin() + 14, "is coolness", 7);// "replace is cool!!!"

	str.replace(str.begin() + 12, str.end() - 4, 4, 'o'); // "replace is coooool!!!"

      // "replace is useful."
	str.replace(str.begin() + 11, str.end(), str4.begin(), str4.end

	std::cout << str << '\n';
	return 0;
}
```

Пример. swap

```
#include <iostream>
#include <string>
int main()
{
	std::string buyer("money");
	std::string seller("goods");

	std::cout << "Before the swap, buyer has " << buyer;
	std::cout << " and seller has " << seller << '\n';

	seller.swap(buyer);

	std::cout << " After the swap, buyer has " << buyer;
	std::cout << " and seller has " << seller << '\n';
	return 0;
}
```

Пример. substr

```
#include <iostream>
#include <string>

int main()
{
	std::string str = "We think in generalities, but we live in details.";
	// (quoting Alfred N. Whitehead)

	std::string str2 = str.substr(3, 5);     // "think"

	std::size_t pos = str.find("live");      // position of "live" in str

	std::string str3 = str.substr(pos);     // get from "live" to the end

	std::cout << str2 << ' ' << str3 << '\n';

	return 0;
}
```


Пример. resize

```
#include <iostream>
#include <string>

int main()
{
	std::string str("I like to code in C");
	std::cout << str << '\n';

	unsigned sz = str.size();

	str.resize(sz + 2, '+');
	std::cout << str << '\n';

	str.resize(14);
	std::cout << str << '\n';
	return 0;
}
```

Пример. insert

```
#include <iostream>
#include <string>

int main()
{
	std::string str = "to be question";
	std::string str2 = "the ";
	std::string str3 = "or not to be";
	std::string::iterator it;

	// used in the same order as described above:
	str.insert(6, str2);                 // to be (the )question
	str.insert(6, str3, 3, 4);             // to be (not )the question
	str.insert(10, "that is cool", 8);    // to be not (that is )the question
	str.insert(10, "to be ");            // to be not (to be )that is the question
	str.insert(15, 1, ':');               // to be not to be(:) that is the question
	it = str.insert(str.begin() + 5, ','); // to be(,) not to be: that is the question
	str.insert(str.end(), 3, '.');       // to be, not to be: that is the question(...)
	str.insert(it + 2, str3.begin(), str3.begin() + 3); // (or )

	std::cout << str << '\n';
	return 0;
}
```

Пример. +=

```
#include <iostream>
#include <string>

int main()
{
	std::string name("John");
	std::string family("Smith");
	name += " K. ";         // c-string
	name += family;         // string
	name += '\n';           // character

	std::cout << name;
	return 0;
}
```


Проверка на вместимость строки

- empty - проверяет, является ли строка пустой
- size - возвращает количество символов в строке
- length - возвращает количество символов в строке
- max_size - возвращает максимальное количество символов

Пример. empty, size, length, max_size

```
#include <iostream>
#include <string>
using namespace std;

int main() {
	string s = "a priori";
	if (s.size() != 0) cout << s.size() << endl;
	if (s.length() != 0) cout << "Yes!" << endl;
	if (!s.empty()) cout << s << endl;
	cout << s.max_size() << endl;//2147483647
	return 0;}
```

Задания

1. Напишите программу, которая формирует в цикле файлы с именами **file_001.txt, file_002.txt …file_020.txt** и записывает **1** – в первый файл, 2 – во второй,… 20 – в двадцатый. (Для формирования имен файлов используйте `#include <string>,` `to_string` – для преобразования числа в строку).

2. Напишите программу, которая формирует в цикле файлы с именами **text_001.txt, text_005.tx**t и записывает в них случайное количество случайных текстовых строк.

3. Создайте текстовый файл **task_03.txt**, содержащий пять строк разной длины, которые состоят из символов английского алфавита.

    a. создайте новый файл **task_03a.txt**, в который запишите измененные строки из исходного файла (длина каждой строки должна равняться минимальной длине строки из исходного файла);

    b. создайте новый файл **task_03b.txt**, в который запишите первые символы строк исходного файла;

    c. напишите программу, которая создает текстовый файл **task_03c.txt.** Этот файл должен содержать строки из файла **task_03.txt**, в которых заменены начальные три символа на **ABC**.

    d. напишите программу, которая создает текстовый файл **task_03d.txt.** Этот файл должен содержать строки из файла **task_03.txt**, в которых заменены последние три символа на **XYZ**.

    e. напишите программу, которая создает текстовый файл **task_03e.txt.** Этот файл должен содержать удвоенные строки из файла **task_03.txt**.

    f. напишите программу, которая создает текстовый файл **task_03f.txt.** Этот файл должен содержать коды символов из файла **task_03.txt**, разделенные пробелом.