

# Регулярные выражения

Шипицын Кирилл Васильевич

18.12.2020



# Возможности

- Проверка текста на соответствие шаблону
- Поиск текста по шаблону
- Извлечение текста по шаблону
- Замена текста по шаблону

# Виды регулярных выражений

## Виды синтаксиса:

- BRE = POSIX Basic Regular Expressions
- ERE = POSIX Extended Regular Expressions
- PCRE = Perl-Compatible Regular Expressions
- C++, .NET, JavaScript, Ruby, Java, везде разный синтаксис

# Regex101

regular expressions 101

@regex101 donate contact bug reports & feedback wiki

</>

SAVE & SHARE

Update Regex ctrl+s

Fork Regex

FLAVOR

PCRE (PHP) ✓

ECMAScript (JavaScript)

Python

Golang

TOOLS

Code Generator

Regex Debugger

SPONSOR

REGULAR EXPRESSION v1

2 matches, 127 steps (~15ms)

/ [^@]+@example.com /gm

TEST STRING

user1234@example.com

testuser@example.com

user@a.example.com

test@forexample.com

SUBSTITUTION

EXPLANATION

> / [^@]+@example.com /gm

> Match a single character not present in the list below

[^@]+

Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)

@ matches the character @ literally (case sensitive)

@example matches the characters @example literally

MATCH INFORMATION

Match 1

Full match 0-20 user1234@example.com

Match 2

Full match 20-41 testuser@example.com

QUICK REFERENCE

Search reference

All Tokens

Common Tokens ✓

General Tokens

Anchors

Meta Sequences

A single character... [abc]

A character exc... [^abc]

A character in th... [a-z]

A character not ... [^a-z]

A character i... [a-zA-Z]

Any single character .



# Regex101

regular expressions 101

@regex101 donate contact bug reports & feedback wiki

</>

SAVE & SHARE

Update Regex ctrl+s

Fork Regex

FLAVOR

PCRE (PHP) ✓

ECMAScript (JavaScript)

Python

Golang

TOOLS

Code Generator

Regex Debugger

SPONSOR

REGULAR EXPRESSION v1

2 matches, 127 steps (~15ms)

TEST STRING

SWITCH TO UNIT TESTS

user1234@example.com

testuser@example.com

user@a.example.com

test@forexample.com

SUBSTITUTION

EXPLANATION

▼ / [^@]+@example.com / gm

▼ Match a single character not present in the list below

[^@]+

Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)

@ matches the character @ literally (case sensitive)

@example matches the characters @example literally

MATCH INFORMATION

Match 1

Full match 0-20 user1234@example.com

Match 2

Full match 20-41 testuser@example.com

QUICK REFERENCE

Search reference

All Tokens

★ Common Tokens ✓

General Tokens

Anchors

Meta Sequences

A single character... [abc]

A character exc... [^abc]

A character in th... [a-z]

A character not ... [^a-z]

A character i... [a-zA-Z]

Any single character .

# Regex101

regular expressions 101

@regex101 donate contact bug reports & feedback wiki

**SAVE & SHARE**

- Update Regex `ctrl+s`
- Fork Regex

**FLAVOR**

- PCRE (PHP) ✓
- ECMAScript (JavaScript)
- Python
- Golang

**TOOLS**

- Code Generator
- Regex Debugger

**REGULAR EXPRESSION** v1

2 matches, 127 steps (~15ms)

`/[^\@]+\@example.com/ gm`

**TEST STRING**

user1234@example.com  
testuser@example.com  
user@a.example.com  
test@forexample.com

**EXPLANATION**

▼ / [^\@]+\@example.com / gm

- ▼ Match a single character not present in the list below
- [^\@]+  
Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)
- @ matches the character @ literally (case sensitive)
- @example matches the characters @example literally

**MATCH INFORMATION**

Match 1

Full match 0-20 user1234@example.com

Match 2

Full match 20-41 testuser@example.com

**QUICK REFERENCE**

Search reference

- All Tokens
- ★ Common Tokens ✓
- General Tokens
- ⚓ Anchors
- Meta Sequences

A single character... [abc]  
A character exc... [^\abc]  
A character in th... [a-z]  
A character not ... [^a-z]  
A character l... [a-zA-Z]  
Any single character .

**SUBSTITUTION**

SPONSOR

# Regex101

regular expressions 101

@regex101 donate contact bug reports & feedback wiki

</>

SAVE & SHARE

Update Regex ctrl+s

Fork Regex

FLAVOR

PCRE (PHP) ✓

ECMAScript (JavaScript)

Python

Golang

TOOLS

Code Generator

Regex Debugger

SPONSOR

REGULAR EXPRESSION v1

2 matches, 127 steps (~15ms)

/ [^@]+@example.com /gm

TEST STRING

user1234@example.com

testuser@example.com

user@a.example.com

test@forexample.com

SUBSTITUTION

EXPLANATION

▼ / [^@]+@example.com / gm

▼ Match a single character not present in the list below

[^@]+

Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)

@ matches the character @ literally (case sensitive)

@example matches the characters @example literally

MATCH INFORMATION

Match 1

Full match 0-20 user1234@example.com

Match 2

Full match 20-41 testuser@example.com

QUICK REFERENCE

Search reference

All Tokens

★ Common Tokens ✓

General Tokens

Anchors

Meta Sequences

A single character... [abc]

A character exc... [^abc]

A character in th... [a-z]

A character not ... [^a-z]

A character i... [a-zA-Z]

Any single character .

# Regex101

regular expressions 101

@regex101 donate contact bug reports & feedback wiki

</>

SAVE & SHARE

Update Regex ctrl+s

Fork Regex

FLAVOR

PCRE (PHP) ✓

ECMAScript (JavaScript)

Python

Golang

TOOLS

Code Generator

Regex Debugger

SPONSOR

REGULAR EXPRESSION v1

2 matches, 127 steps (~15ms)

/ [^@]+@example.com /gm

TEST STRING

user1234@example.com

testuser@example.com

user@a.example.com

test@forexample.com

SUBSTITUTION

EXPLANATION

</> / [^@]+@example.com / gm

</> Match a single character not present in the list below

</> [^@]+

</> Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)

</> @ matches the character @ literally (case sensitive)

</> @example matches the characters @example literally

MATCH INFORMATION

Match 1

Full match 0-20 user1234@example.com

Match 2

Full match 20-41 testuser@example.com

QUICK REFERENCE

Search reference

All Tokens

Common Tokens ✓

General Tokens

Anchors

Meta Sequences

A single character... [abc]

A character exc... [^abc]

A character in th... [a-z]

A character not ... [^a-z]

A character i... [a-zA-Z]

Any single character .



# Regex101

regular expressions 101

@regex101 donate contact bug reports & feedback wiki

</>

SAVE & SHARE

Update Regex ctrl+s

Fork Regex

FLAVOR

PCRE (PHP) ✓

ECMAScript (JavaScript)

Python

Golang

TOOLS

Code Generator

Regex Debugger

SPONSOR

REGULAR EXPRESSION v1

2 matches, 127 steps (~15ms)

/ [^@]+@example.com /gm

TEST STRING

user1234@example.com

testuser@example.com

user@a.example.com

test@forexample.com

SUBSTITUTION

EXPLANATION

▼ / [^@]+@example.com /gm

▼ Match a single character not present in the list below

[^@]+

Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)

@ matches the character @ literally (case sensitive)

@example matches the characters @example literally

MATCH INFORMATION

Match 1

Full match 0-20 user1234@example.com

Match 2

Full match 20-41 testuser@example.com

QUICK REFERENCE

Search reference

All Tokens

Common Tokens ✓

General Tokens

Anchors

Meta Sequences

A single character... [abc]

A character exc... [^abc]

A character in th... [a-z]

A character not ... [^a-z]

A character i... [a-zA-Z]

Any single character .

# Regex101

regular expressions 101

@regex101 donate contact bug reports & feedback wiki

</>

SAVE & SHARE

Update Regex ctrl+s

Fork Regex

FLAVOR

</> PCRE (PHP) ✓

</> ECMAScript (JavaScript)

</> Python

</> Golang

TOOLS

Code Generator

Regex Debugger

SPONSOR

REGULAR EXPRESSION v1

2 matches, 127 steps (~15ms)

/ [^@]+@example.com / gm

TEST STRING

user1234@example.com

testuser@example.com

user@a.example.com

test@forexample.com

SUBSTITUTION

SWITCH TO UNIT TESTS

EXPLANATION

</> / [^@]+@example.com / gm

</> Match a single character not present in the list below

</> [^@]+

</> Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)

</> @ matches the character @ literally (case sensitive)

</> @example matches the characters @example literally

MATCH INFORMATION

Match 1

Full match 0-20 user1234@example.com

Match 2

Full match 20-41 testuser@example.com

QUICK REFERENCE

Search reference

All Tokens

★ Common Tokens ✓

General Tokens

Anchors

Meta Sequences

A single character... [abc]

A character exc... [^abc]

A character in th... [a-z]

A character not ... [^a-z]

A character i... [a-zA-Z]

Any single character .

CQG

# Regex101

regular expressions 101

@regex101 donate contact bug reports & feedback wiki

SAVE & SHARE

Update Regex `ctrl+s`

Fork Regex

FLAVOR

</> PCRE (PHP) ✓

</> ECMAScript (JavaScript)

</> Python

</> Golang

TOOLS

Code Generator

Regex Debugger

SPONSOR

REGULAR EXPRESSION v1

2 matches, 127 steps (~15ms)

TEST STRING

user1234@example.com  
testuser@example.com  
user@a.example.com  
test@forexample.com

SUBSTITUTION

EXPLANATION

/ / `[^@]+@example.com` / gm

Match a single character not present in the list below

`[^@]+`

Quantifier — Matches between one and unlimited times, as many times as possible, giving back as needed (greedy)

@ matches the character @ literally (case sensitive)

@example matches the characters @example literally

MATCH INFORMATION

Match 1

Full match 0-20 user1234@example.com

Match 2

Full match 20-41 testuser@example.com

QUICK REFERENCE

Search reference

All Tokens

★ Common Tokens ✓

General Tokens

Anchors

Meta Sequences

A single character... [abc]

A character exc... [^abc]

A character in th... [a-z]

A character not ... [^a-z]

A character i... [a-zA-Z]

Any single character .

# Основы регулярных выражений

Алфавит – Unicode

Служебные символы – `[]{}()|\.+*?^$`

Экранирование	<code>\&lt;sym&gt;</code>
Конкатенация	<code>&lt;first&gt;&lt;second&gt;</code>
Чередование	<code>&lt;first&gt; &lt;second&gt;</code>
Группировка	<code>(&lt;first&gt;)</code>
Любой символ	<code>.</code>



# Пример

```
#include (<|"").....\.(h|hpp)("|>)
```

# Пример

**#include** (<|"").....\.(h|hpp)("|>)

# Пример

```
#include (<|").....\.(h|hpp)("|>)
```

# Пример

```
#include (<|"").....\.(h|hpp)("|>)
```



# Пример

```
#include (<|"").....\.(h|hpp)("|>)
```

# Пример

```
#include (<|"").....\.(h|hpp)("|>)
```

# Пример

`#include (<|"").....\.(h|hpp)("|>)`

## Invalid

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

```
#include "stdio.h"
```

```
#include 'helper.hpp'
```

```
#include "HELPER.HPP"
```

## Valid

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

```
#include "stdlib.h"
```

```
#include <helper.hpp>
```

```
#include "helper.hpp"
```

# Символьные классы

[abcde] – любой из перечисленных символов

[^abcde] – любой символ кроме перечисленных

[a-t] – задание диапазона символов

Класс	Эквивалент	Описание	Отрицание
\d	[0-9]	Цифры	\D
\s	[ \f\n\r\t\v]	Пробелы	\S
\w	[a-zA-Z0-9_]	Буква, цифра, подчеркивание	\W



```
#include (<|"")\w\w\w\w\w\w\w\.(h|hpp)("|>)
```

[illegible]

```
#include (<|"")\w\w\w\w\w\w\w\.(h|hpp)("|>)
```

```
#include (<|")\w\w\w\w\w\w\w\.(h|hpp)("|>)
```

Invalid	Valid
#include <std.io.h>	#include <stdlib.h>
#include "std-io.h"	#include "stdlib.h"
#include <ab/csv.hpp>	#include <helper.hpp>
#include "../csv.hpp"	#include "helper.hpp"

# Квантификаторы

Синтаксис:  $\langle \text{expr} \rangle \langle \text{quant} \rangle$

Ищется **максимальное** вхождение

$\{n\}$	Ровно n раз
$\{m, n\}$	От m до n раз
$\{m, \}$	Не менее m раз
$\{, n\}$	Не более n раз
$\{0, 1\} = ?$	0 или 1 раз
$\{0, \} = *$	0 и более раз (любое количество раз)
$\{1, \} = +$	1 и более раз



# Пример

```
#include[ \t]*
```

```
(<|")[\.\\\/:\w]+\.(h|hpp)("|>)
```

# Пример

```
#include[ \t]*
```

```
(<|")[\.\.\.\./:\w]+\.(h|hpp)("|>)
```

# Пример

```
#include[ \t]*
```

```
(<|")[\.\.\./:\w]+\.(h|hpp)("|>)
```

# Пример

`#include[ \t]*`

`(<|")[\.\\\/:\w]+\.(h|hpp)("|>)`

## Valid

```
#include "C:\\Users\\Admin\\Desktop\\project\\helper.hpp"
```

```
#include "../helper.hpp"
```

```
#include "project/helper.hpp"
```

```
#include<stdio.h>
```

# Символы позиционирования

Позиционируют регулярное выражение в строке относительно начала/конца строки/слова.

^	Начало текста	^cat	catdog catdog
\$	Конец текста	dog\$	catdog catdog
\b	Граница слова	\bcat	catcat catcat
\B	Не граница слова	\Bcat	catcat catcat



# Модификаторы

(?<mod>), (?-<mod>) – в начале выражения  
включает/выключает модификатор

# Модификатор

# g(global)

Искать все вхождения

\d

g

1 2 3 4 5

-g

1 2 3 4 5

# Модификатор m(multi-line)

^\$ соответствуют началу/концу строк

**^is\$**

m

is  
this  
his

-m

is  
this  
his

# Модификатор i(insensitive)

Отключить чувствительность к регистру

[a-z]

i

a b c d e

-i

a b c d e

# Модификатор x(extended)

Можно писать комментарии после #. Пробелы игнорируются.

\d #Цифры

\s #Пробелы

\w #Буква, цифра, подчеркивание

x	-x
1 b 2 c 3 d	1 b 2 c 3 d



# Модификатор s(single-line)

Точка (.) соответствует символу новой строки

.\*

S	-S
123	123
456	456

# Проблема

`<td>.*</td>`

```
<table>
```

```
<tr><th>Firstname</th><th>Lastname</th><th>Age</th></tr>
```

```
<tr><td>Jill</td><td>Smith</td><td>50</td></tr>
```

```
<tr><td>Eve</td><td>Jackson</td><td>94</td></tr>
```

```
</table>
```

# Проблема

<td>.\*</td>

```
<table>
```

```
<tr><th>Firstname</th><th>Lastname</th><th>Age</th></tr>
```

```
<tr><td>Jill</td><td>Smith</td><td>50</td></tr>
```

```
<tr><td>Eve</td><td>Jackson</td><td>94</td></tr>
```

```
</table>
```

# Ленивые квантификаторы

Ищут минимальное вхождение.

Синтаксис:

`<expr><quant>?`

# Решение

`<td>.*?<\/td>`

```
<table>
```

```
<tr><th>Firstname</th><th>Lastname</th><th>Age</th></tr>
```

```
<tr><td>Jill</td><td>Smith</td><td>50</td></tr>
```

```
<tr><td>Eve</td><td>Jackson</td><td>94</td></tr>
```

```
</table>
```



# Группировка и захваты

(<expr>)

– захватывающая группа

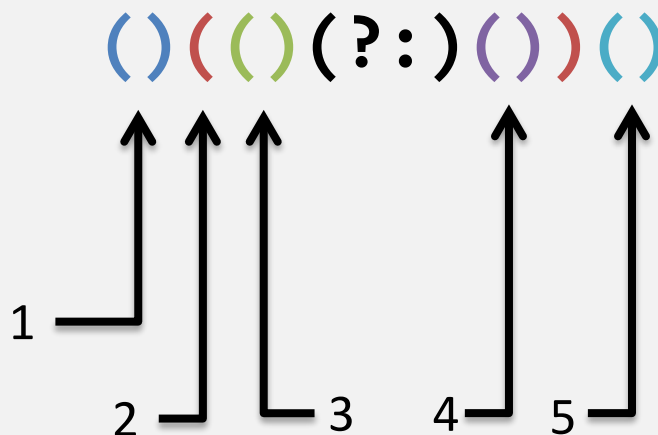
(?:<expr>)

– незахватывающая группа

(?<name><expr>)

– именованная группа

Нумерация групп



# Пример

$$\begin{aligned} & ^{(? : 8 | \backslash + 7 [ \backslash - ] ? ) ?} \\ & ( \backslash ( ? \backslash d \{ 3 \} \backslash ) ? [ \backslash - ] ? ) ? \\ & ( [ \backslash d \backslash - ] \{ 7 , 10 \} ) \$ \end{aligned}$$

# Пример

$^{(? : 8 | \backslash + 7 [ \backslash - ] ? ) ?}$

$( \backslash ( ? \backslash d \{ 3 \} \backslash ) ? [ \backslash - ] ? ) ?$

$( [ \backslash d \backslash - ] \{ 7 , 10 \} ) \$$

# Пример

$^{(? : 8 | \backslash + 7 [ \backslash - ] ? ) ?}$   
 $( \backslash ( ? \backslash d \{ 3 \} \backslash ) ? [ \backslash - ] ? ) ?$   
 $( [ \backslash d \backslash - ] \{ 7 , 10 \} ) \$$

# Пример

$^((?:8|\backslash+7[\backslash- ]?))?$   
 $(\backslash(?:\d{3}\backslash)?[\backslash- ]?)?$   
 $([\backslash\d\backslash- ]{7,10})\$$



# Пример

`^(?:8|\+7[\- ]?)?  
(\(?\d{3}\)?[\- ]?)?  
([\d\- ]{7,10})$`

Full match	+79261234567
Group 1	926
Group 2	1234567

# Ссылки назад

В регулярном выражении можно сослаться на уже совпавшие группы.

Синтаксис:

`\<n>`,  $n \in [0; 99]$

`\g{<n>}`

`\g{<name>}`

`\1`, `\2`, ...

`\g{1}`, `\g{2}`, ...

`\g{<a>}`, `\g{<b>}`, ...



Пример

`(' | ").*?\g{1}`



Пример

`('|").*?\g{1}`



Пример

( ' | " ) . \* ? \g{1}

# Пример

`(' | ").*?\g{1}`



# Пример

( ' | " ) . \* ? \g{1}

**Invalid**

"abc '

**Valid**

"abc"

" "

'abc'

# Условный оператор

Проверяет было ли найдено соответствие для предыдущей группы по имени или номеру (либо для выражения нулевой длины).

Синтаксис:

`(?(<name>)<first>|<second>)`

`(?(?=<expr>)<first>|<second>)`

Пример `(?(<name><first>|<second>)`

`(?<comment>^\s/\s/)?`

`.*`

`(?(<comment>)(\n)|(; \n))$)`

Пример `(?(<name><first>|<second>)`

`(?<comment>^\s\s)?`

`.*`

`(?(<comment>)(\n)|(; \n))$)`

Пример  $(?(<name><first>|<second>)$

$(?<comment>^\\/\\/?$

$\cdot *$

$(?(<comment>)(\\n)|(;\\n))\$$

Пример `(?(<name><first>|<second>)`

`(?<comment>^\s/\s/)?`

`.*`

`(?(<comment>)(\n)|(; \n))$)`



Пример `(?(<name><first>|<second>)`

`(?<comment>^\s/\s/)?`

`.*`

`(?(<comment>)(\n)|(; \n))$)`

Пример `(?(<name><first>|<second>)`

`(?<comment>^\n/\/)?`

`.*`

`(?(<comment>)(\n)|(; \n))$)`

# Пример

(?(<name><first>|<second>)

(?<comment>^\n/\/)?

.\*

(?(<comment>)(\n)|(; \n))\$)

Invalid

```
int x = 0
```

Valid

```
int x = 0;
```

```
// int x = 0
```

```
// int x = 0;
```

Пример

$(? ( ? = \langle \text{expr} \rangle ) \langle \text{first} \rangle | \langle \text{second} \rangle )$

$(? ( ? = ^ \backslash / \backslash / ) ( . * \backslash n ) | ( . * ; \backslash n ) \$ )$

Пример

$(? ( ? = \langle \text{expr} \rangle ) \langle \text{first} \rangle | \langle \text{second} \rangle )$

$( ? ( ? = ^ \backslash / \backslash / ) ( . * \backslash n ) | ( . * ; \backslash n ) \$ )$

Пример

$(? ( ? = \langle \text{expr} \rangle ) \langle \text{first} \rangle | \langle \text{second} \rangle )$

$(? ( ? = ^ \backslash / \backslash / ) ( . * \backslash n ) | ( . * ; \backslash n ) \$ )$



Пример

$(? ( ? = \langle \text{expr} \rangle ) \langle \text{first} \rangle | \langle \text{second} \rangle )$

$(? ( ? = ^ \backslash / \backslash / ) ( . * \backslash n ) | ( . * ; \backslash n ) \$ )$

# Пример

(? ( ? = <expr> ) <first> | <second> )

(? ( ? = ^ \\ / ) ( . \* \n ) | ( . \* ; \n ) \$ )

## Invalid

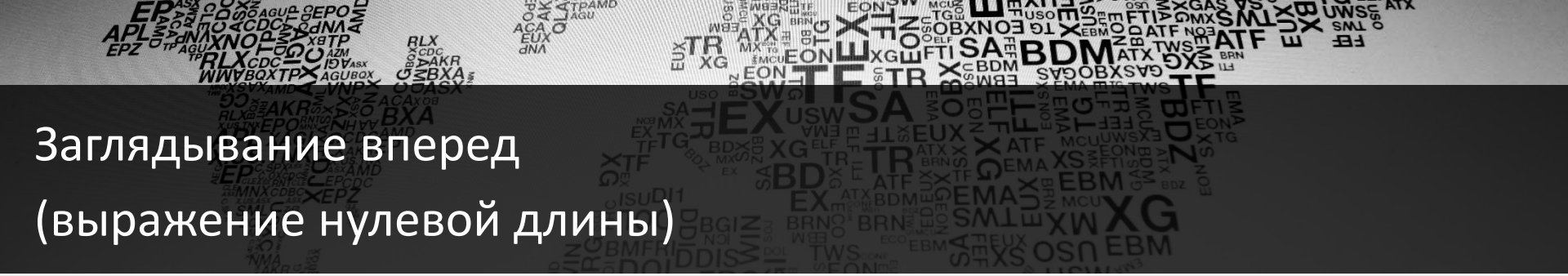
```
int x = 0
```

## Valid

```
int x = 0;
```

```
// int x = 0
```

```
// int x = 0;
```



## Заглядывание вперед (выражение нулевой длины)

Просматривание последующего текста без включения в найденное.

Синтаксис:

( ?=<expr> ) – позитивное

( ?!<expr> ) – негативное



# Пример

(?= <expr>)

\d+(?=,)

5, 10, 15, 20, 25, 30

Пример

(?=<expr>)

\d+(?=,)

5, 10, 15, 20, 25, 30



# Пример

(?!<expr>)

\d+(?! ,)

5, 10, 15, 20, 25, 30




# Пример

(?!<expr>)

\d+(?! ,)

5, 10, 15, 20, 25, 30



## Оглядывание назад (выражение нулевой длины)

Просматривание предыдущего текста без включения в найденное.

**Квантификаторы запрещены.**

Синтаксис:

$(?<=<expr>)$  – позитивное

$(?<!\<expr>)$  – негативное

Пример

(?<=<expr>)

(?<=, \s)(\d+)

5, 10, 15, 20, 25, 30

Пример

(?<=<expr>)

(?<=, \s)(\d+)

5, 10, 15, 20, 25, 30

Пример

(?<!<expr>)

(?<!, \s)(\d+)

5, 10, 15, 20, 25, 30

Пример

(?<!<expr>)

(?<!, \s)(\d+)

5, 10, 15, 20, 25, 30





# Замена текста

Большинство программ позволяют производить замену по регулярным выражениям.

В замещающем тексте можно использовать захваченные группы.

# Пример

Поиск	(\d+)\u
Замена	\1 или \$1

Исходный текст:	Результат:
const auto x = 1 <u>u</u> ;	const auto x = 1;

# Пример `std::regex_match`

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

int main()
{
    std::string fnames[] = {"foo.txt", "bar.txt", "zoidberg"};

    std::regex txt_regex("[a-z]+\\.txt");
    for (const auto &fname : fnames) {
        std::cout << fname << ": " << std::regex_match(fname, txt_regex) << '\n';
    }
}
```

# Пример `std::regex_match`

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

int main()
{
    std::string fnames[] = {"foo.txt", "bar.txt", "zoidberg"};

    std::regex txt_regex("[a-z]+\\.txt");
    for (const auto &fname : fnames) {
        std::cout << fname << ": " << std::regex_match(fname, txt_regex) << '\n';
    }
}
```

# Пример `std::regex_match`

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

int main()
{
    std::string fnames[] = {"foo.txt", "bar.txt", "zoidberg"};

    std::regex txt_regex("[a-z]+\\.txt");
    for (const auto &fname : fnames) {
        std::cout << fname << ": " << std::regex_match(fname, txt_regex) << '\n';
    }
}
```

# Пример `std::regex_match`

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

int main()
{
    std::string fnames[] = {"foo.txt", "bar.txt", "zoidberg"};

    std::regex txt_regex("[a-z]+\\.txt");
    for (const auto &fname : fnames) {
        std::cout << fname << ": " << std::regex_match(fname, txt_regex) << '\n';
    }
}
```



# Пример `std::regex_match`

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

int main()
{
    std::string fnames[] = {"foo.txt", "bar.txt", "zoidberg"};

    std::regex txt_regex("[a-z]+\\.txt");
    for (const auto &fname : fnames) {
        std::cout << fname << ": " << std::regex_match(fname, txt_regex) << '\n';
    }
}
```

```
foo.txt: 1
bar.txt: 1
zoidberg: 0
```

# Пример `std::regex_search`

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

int main()
{
    std::string s("this subject has a submarine as a subsequence");
    std::smatch m;
    std::regex e("\\b(sub)([^\ ]*)");

    std::cout << "Target sequence: " << s << std::endl;
    std::cout << "Regular expression: /\\b(sub)([^\ ]*)/" << std::endl;
    std::cout << "The following matches and submatches were found:" << std::endl;

    while (std::regex_search (s,m,e))
    {
        for (auto x:m) std::cout << x << " ";
        std::cout << std::endl;
        s = m.suffix().str();
    }

    return 0;
}
```

# Пример `std::regex_search`

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

int main()
{
    std::string s("this subject has a submarine as a subsequence");
    std::smatch m;
    std::regex e("\\b(sub)([^\ ]*)");

    std::cout << "Target sequence: " << s << std::endl;
    std::cout << "Regular expression: /\\b(sub)([^\ ]*)/" << std::endl;
    std::cout << "The following matches and submatches were found:" << std::endl;

    while (std::regex_search (s,m,e))
    {
        for (auto x:m) std::cout << x << " ";
        std::cout << std::endl;
        s = m.suffix().str();
    }

    return 0;
}
```

# Пример `std::regex_search`

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

int main()
{
    std::string s("this subject has a submarine as a subsequence");
    std::smatch m;
    std::regex e("\\b(sub)([^\ ]*)");

    std::cout << "Target sequence: " << s << std::endl;
    std::cout << "Regular expression: /\\b(sub)([^\ ]*)/" << std::endl;
    std::cout << "The following matches and submatches were found:" << std::endl;

    while (std::regex_search (s,m,e))
    {
        for (auto x:m) std::cout << x << " ";
        std::cout << std::endl;
        s = m.suffix().str();
    }

    return 0;
}
```

# Пример `std::regex_search`

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

int main()
{
    std::string s("this subject has a submarine as a subsequence");
    std::smatch m;
    std::regex e("\\b(sub)([^\ ]*)");

    std::cout << "Target sequence: " << s << std::endl;
    std::cout << "Regular expression: /\\b(sub)([^\ ]*)/" << std::endl;
    std::cout << "The following matches and submatches were found:" << std::endl;

    while (std::regex_search (s,m,e))
    {
        for (auto x:m) std::cout << x << " ";
        std::cout << std::endl;
        s = m.suffix().str();
    }

    return 0;
}
```



# Пример `std::regex_search`

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

int main()
{
    std::string s("this subject has a submarine as a subsequence");
    std::smatch m;
    std::regex e("\\b(sub)([^\ ]*)");

    std::cout << "Target sequence: " << s << std::endl;
    std::cout << "Regular expression: /\\b(sub)([^\ ]*)/" << std::endl;
    std::cout << "The following matches and submatches were found:" << std::endl;

    while (std::regex_search(s,m,e))
    {
        for (auto x:m) std::cout << x << " ";
        std::cout << std::endl;
        s = m.suffix().str();
    }

    return 0;
}
```



# Пример `std::regex_search`

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

int main()
{
    std::string s("this subject has a submarine as a subsequence");
    std::smatch m;
    std::regex e("\\b(sub)([^\ ]*)");

    std::cout << "Target sequence: " << s << std::endl;
    std::cout << "Regular expression: /\\b(sub)([^\ ]*)/" << std::endl;
    std::cout << "The following matches and submatches were found:" << std::endl;

    while (std::regex_search (s,m,e))
    {
        for (auto x:m) std::cout << x << " ";
        std::cout << std::endl;
        s = m.suffix().str();
    }

    return 0;
}
```

# Пример `std::regex_search`

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

int main()
{
    std::string s("this subject has a submarine as a subsequence");
    std::smatch m;
    std::regex e("\\b(sub)([^\ ]*)");

    std::cout << "Target sequence: " << s << std::endl;
    std::cout << "Regular expression: /\\b(sub)([^\ ]*)/" << std::endl;
    std::cout << "The following matches and submatches were found:" << std::endl;

    while (std::regex_search (s,m,e))
    {
        for (auto x:m) std::cout << x << " ";
        std::cout << std::endl;
        s = m.suffix().str();
    }

    return 0;
}
```

# Пример `std::regex_search`

Target sequence: this subject has a submarine as a subsequence

Regular expression: `/\b(sub)([^\s]*)/`

The following matches and submatches were found:

subject sub ject

submarine sub marine

subsequence sub sequence

# Пример `std::regex_match`

```
#include <iostream>
#include <regex>
#include <string>
int main()
{
    std::string text = "Quick brown fox";
    std::regex vowel_re("a|o|e|u|i");
    std::cout << std::regex_replace(text, vowel_re, "[$&]") << '\n';
}
```

# Пример `std::regex_match`

```
#include <iostream>
#include <regex>
#include <string>
int main()
{
    std::string text = "Quick brown fox";
    std::regex vowel_re("a|o|e|u|i");
    std::cout << std::regex_replace(text, vowel_re, "[$&]") << '\n';
}
```

# Пример `std::regex_match`

```
#include <iostream>
#include <regex>
#include <string>
int main()
{
    std::string text = "Quick brown fox";
    std::regex vowel_re("a|o|e|u|i");
    std::cout << std::regex_replace(text, vowel_re, "[$&]") << '\n';
}
```



# Пример `std::regex_match`

```
#include <iostream>
#include <regex>
#include <string>
int main()
{
    std::string text = "Quick brown fox";
    std::regex vowel_re("a|o|e|u|i");
    std::cout << std::regex_replace(text, vowel_re, "[$&]") << '\n';
}
```

# Пример `std::regex_match`

```
#include <iostream>
#include <regex>
#include <string>
int main()
{
    std::string text = "Quick brown fox";
    std::regex vowel_re("a|o|e|u|i");
    std::cout << std::regex_replace(text, vowel_re, "[$&]") << '\n';
}
```

Q[u][i]ck br[o]wn f[o]x



# Ссылки

[regex101.com](http://regex101.com)

[rexegg.com](http://rexegg.com)

[regular-expressions.info](http://regular-expressions.info)

[habr.com/ru/post/166777/](http://habr.com/ru/post/166777/)

# Q&A