

Ravesli [Ravesli](#)


- [Уроки по C++](#)
- [OpenGL](#)
- [SFML](#)
- [Qt5](#)
- [RegExr](#)
- [Ассемблер](#)
- [Купить .PDF](#)

Урок №38. Приоритет операций и правила ассоциативности

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 7 Сен 2020 |

 43174

[1](#)  9

Чтобы правильно вычислять выражения (например, $4 + 2 * 3$), мы должны знать, что делают определенные операторы и в каком порядке они выполняются. Последовательность, в которой они выполняются, называется **приоритетом операций**. Следуя обычным правилам математики (в которой умножение следует перед сложением), выражение, приведенное выше в данном абзаце как пример, обрабатывается следующим образом: $4 + (2 * 3) = 10$.

В языке C++ все операторы (операции) имеют свой уровень приоритета. Те, в которых он выше, выполняются первыми. В таблице, приведенной ниже, можно увидеть, что приоритет операций умножения и деления (5) выше, чем в операциях сложения и вычитания (6). Компилятор использует приоритет операторов для определения порядка обработки выражений.

А что делать, если у двух операторов в выражении одинаковый уровень приоритета, и они размещены рядом? Какую операцию компилятор выполнит первой? А здесь уже компилятор будет использовать **правила ассоциативности**, которые указывают направление выполнения операций: слева направо или справа налево. Например, в выражении $3 * 4 / 2$ операции умножения и деления имеют одинаковый уровень приоритета (5-й уровень). А ассоциативность пятого уровня соответствует выполнению операций слева направо, таким образом: $(3 * 4) / 2 = 6$.

Оглавление:

1. [Таблица приоритета и ассоциативности операций](#)
2. [Как возвести число в степень в C++?](#)
3. [Тест](#)

Таблица приоритета и ассоциативности операций

Несколько примечаний:

→ 1 означает самый высокий уровень приоритета, а 17 — самый низкий. Операции с более высоким уровнем приоритета выполняются первыми.

→ L -> R означает слева направо.

→ R -> L означает справа налево.

Ассоциативность	Оператор	Описание	Пример
1. Нет	::	Глобальная область видимости (унарный)	::name
	::	Область видимости класса (бинарный)	class_name::member_name
	()	Круглые скобки	(expression)
	()	Вызов функции	function_name(parameters)
	()	Инициализация	type name(expression)
	{}	uniform-инициализация (C++11)	type name {expression}
	type()	Конвертация типа	new_type(expression)
	type{}	Конвертация типа (C++11)	new_type {expression}
	[]	Индекс массива	pointer[expression]
	.	Доступ к члену объекта	object.member_name
2. L -> R	->	Доступ к члену объекта через указатель	object_pointer->member_name
	++	Пост-инкремент	lvalue++
	—	Пост-декремент	lvalue—
	typeid	Информация о типе во время выполнения	typeid(type) or typeid(expression)
	const_cast	Cast away const	const_cast(expression)
	dynamic_cast	Type-checked cast во время выполнения	dynamic_cast(expression)
	reinterpret_cast	Конвертация одного типа в другой	reinterpret_cast(expression)
	static_cast	Type-checked cast во время компиляции	static_cast(expression)
	+	Унарный плюс	+expression
	—	Унарный минус	-expression
	++	Пре-инкремент	++lvalue
	—	Пре-декремент	—lvalue
	!	Логическое НЕ (NOT)	!expression
	~	Побитовое НЕ (NOT)	~expression
	(type)	C-style cast	(new_type)expression
3. R -> L	sizeof	Размер в байтах	sizeof(type) or sizeof(expression)
	&	Адрес	&lvalue
	*	Разыменование	*expression
	new	Динамическое выделение памяти	new type
	new[]	Динамическое выделение массива	new type[expression]
	delete	Динамическое удаление памяти	delete pointer
	delete[]	Динамическое удаление массива	delete[] pointer
4. L -> R	->*	Member pointer selector	object_pointer->*pointer_to_member

	.	*	Member object selector	object.*pointer_to_member
	*		Умножение	expression * expression
5. L -> R	/		Деление	expression / expression
	%		Деление с остатком	expression % expression
6. L -> R	+		Сложение	expression + expression
	—		Вычитание	expression — expression
7. L -> R	<<		Побитовый сдвиг влево	expression << expression
	>>		Побитовый сдвиг вправо	expression >> expression
	<		Сравнение: меньше чем	expression < expression
8. L -> R	<=		Сравнение: меньше чем или равно	expression <= expression
	>		Сравнение: больше чем	expression > expression
	>=		Сравнение: больше чем или равно	expression >= expression
9. L -> R	==		Равно	expression == expression
	!=		Не равно	expression != expression
10. L -> R	&		Побитовое И (AND)	expression & expression
11. L -> R	^		Побитовое исключающее ИЛИ (XOR)	expression ^ expression
12. L -> R			Побитовое ИЛИ (OR)	expression expression
13. L -> R	&&		Логическое И (AND)	expression && expression
14. L -> R			Логическое ИЛИ (OR)	expression expression
	?:		Тернарный условный оператор	expression ? expression : expression
	=		Присваивание	lvalue = expression
	*=		Умножение с присваиванием	lvalue *= expression
	/=		Деление с присваиванием	lvalue /= expression
	%=		Деление с остатком и с присваиванием	lvalue %= expression
	+=		Сложение с присваиванием	lvalue += expression
	-=		Вычитание с присваиванием	lvalue -= expression
15. R -> L	<<=		Присваивание с побитовым сдвигом влево	lvalue <<= expression
	>>=		Присваивание с побитовым сдвигом вправо	lvalue >>= expression
	&=		Присваивание с побитовой операцией И (AND)	lvalue &= expression
	=		Присваивание с побитовой операцией ИЛИ (OR)	lvalue = expression
	^=		Присваивание с побитовой операцией «Исключающее ИЛИ» (XOR)	lvalue ^= expression
16. R -> L	throw		Генерация исключения	throw expression
17. L -> R	,		Оператор Запятая	expression, expression

Некоторые операторы вы уже знаете из предыдущих уроков: +, -, *, /, (), =, < и >. Их значения одинаковы как в математике, так и в языке C++.

Однако, если у вас нет опыта работы с другими языками программирования, то большинство из этих операторов вам сейчас могут быть непонятны. Это нормально. Мы рассмотрим большую их часть на уроках этой главы, а об остальных расскажем по мере необходимости.

Эта таблица предназначена в первую очередь для того, чтобы вы могли в любой момент обратиться к ней для решения возможных проблем приоритета или ассоциативности.

Как возвести число в степень в C++?

Вы уже должны были заметить, что оператор $^$, который обычно используется для обозначения возведения в степень в обычной математике, не является таковым в языке C++. В языке C++ это побитовая операция XOR. А для возведения числа в степень в языке C++ используется функция `pow()`, которая находится в [заголовочном файле](#) `cmath`:

```
1 #include <cmath>
2
3 double x = pow(3.0, 4.0); // 3 в степени 4
```

Обратите внимание, параметры и возвращаемые значения функции `pow()` являются типа `double`. А поскольку типы с плавающей точкой известны [ошибками округления](#), то результаты `pow()` могут быть неточными (чуть меньше или чуть больше).

Если вам нужно возвести в степень целое число, то лучше использовать собственную функцию, например:

```
1 // Примечание: Экспонент не должен быть отрицательным
2 int pow(int base, int exp)
3 {
4     int result = 1;
5     while (exp)
6     {
7         if (exp & 1)
8             result *= base;
9         exp >>= 1;
10        base *= base;
11    }
12
13    return result;
14 }
```

Не переживайте, если здесь что-то не понятно. Просто помните о проблеме переполнения, которая может произойти, если один из аргументов будет слишком большим.

Тест

Из школьной математики нам известно, что выражения внутри скобок выполняются первыми. Например, в выражении $(2 + 3) * 4$ часть $(2 + 3)$ выполняется первой.

В этом задании есть 4 выражения, в которых отсутствуют какие-либо скобки. Используя приоритет операций и правила ассоциативности, приведенные выше, добавьте скобки в каждое выражение так, как если бы их обрабатывал компилятор.

Подсказка: Используйте колонку «Пример» в таблице приоритета и ассоциативности операций, чтобы определить, является ли оператор [унарным](#) (имеет один операнд) или бинарным (имеет два

операнда).

Например: $x = 2 + 3 \% 4$

Бинарный оператор $\%$ имеет более высокий приоритет, чем оператор $+$ или $=$, поэтому он выполняется первым: $x = 2 + (3 \% 4)$. Затем выполняется бинарный оператор $+$, так как он имеет более высокий приоритет, чем оператор $=$.

Ответ: $x = (2 + (3 \% 4))$.

Дальше нам уже не нужна таблица, чтобы понять ход обработки этого выражения компилятором.

Задания:

- *Выражение №1:* $x = 3 + 4 + 5$
- *Выражение №2:* $x = y = z$
- *Выражение №3:* $z *= ++y + 5$
- *Выражение №4:* $a || b \&\& c || d$

Ответ

Выражение №1: $x = 3 + 4 + 5$

Уровень приоритета бинарного оператора $+$ выше, чем оператора $=$, поэтому: $x = (3 + 4 + 5)$. Ассоциативность бинарного оператора $+$ слева направо, поэтому **ответ:** $x = ((3 + 4) + 5)$.

Выражение №2: $x = y = z$

Ассоциативность бинарного оператора $=$ справа налево, поэтому **ответ:** $x = (y = z)$.

Выражение №3: $z *= ++y + 5$

Унарный оператор $++$ имеет наивысший приоритет, поэтому: $z *= (++y) + 5$. Затем идет бинарный оператор $+$, поэтому **ответ:** $z *= ((++y) + 5)$.

Выражение №4: $a || b \&\& c || d$

Бинарный оператор $\&\&$ имеет приоритет выше, чем $||$, поэтому: $a || (b \&\& c) || d$. Ассоциативность бинарного оператора $||$ слева направо, поэтому **ответ:** $(a || (b \&\& c)) || d$.

Оценить статью:

★★★★★ (269 оценок, среднее: 4,90 из 5)



← [Глава №2. Итоговый тест](#)

[Урок №39. Арифметические операторы](#)

Комментариев: 9



1. *Виктор:*

[5 июня 2020 в 14:00](#)

Чё-то нифига не понятно.

`while (exp)`

1. `exp==false(0)`, цикл не выполняется и сразу `return`. Тут понятно.
2. `exp==true(1)`, то получаем бесконечный цикл. Не увидел условие выхода из цикла.
3. `exp==2`, 3 и т.д., то есть при другом возможном значении?

[Ответить](#)



2. *Георгий:*

[5 февраля 2020 в 01:46](#)

«Если вам нужно возвести в степень целое число, то лучше использовать собственную функцию»

Что-то я не понял, а зачем нам `pow()` тогда вообще? Нам привели пример как пользоваться `pow`, но сказали использовать какой-то не понятный вариант. Вообще ничего не понял в этой главе с возведением в степень. Можно объяснить? Почему мы не можем возвести в степень целочисленный тип через `pow()`? Если через `double` лучше не делать, то зачем `pow()`?

[Ответить](#)



1. *PP189:*

[31 марта 2020 в 13:01](#)

Использование собственной функции нужно для возведения целых чисел в степень. Например, степени десятки для перевода в различные системы счисления.

`Pow()` применяется для чисел с плавающей точкой: он может принять аргументы `int`, но вернет их в возможно не точном виде типа `double`. При этом, даже такое неточное значение может пригодиться в математических уравнениях.

[Ответить](#)



2. *Борис:*

[26 апреля 2020 в 18:12](#)

Элементарно, Ватсон! Целые числа возводить в целую же степень — проще простого, поэтому легко реализовать такую функцию самому с целочисленными типами, где гарантированно всё будет без ошибок округления. А вот `pow()` корректно возводит в степень любые числа, даже дробные и отрицательные (как основание степени, так и показатель). Огромная разница.

[Ответить](#)



3. Андрей:

[26 ноября 2019 в 14:23](#)

Возвести целое число в целую степень разве не проще через обычный цикл?

```
1 int pow2(int base, int exp) {
2     int result = 1;
3     for (int i = 0; i < exp; i++) {
4         result *= base;
5     }
6     return result;
7 }
```

Зачем такая витиеватость?

[Ответить](#)

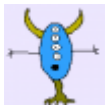


1. Андрей:

[26 ноября 2019 в 14:25](#)

Понятно. Количество умножений сократили.

[Ответить](#)



4. zashiki:

[19 июня 2019 в 13:25](#)

здравствуйте, может, где то упоминалось, но не найду: операторы ввода вывода << и >> (к примеру в std::cout<<) — это и есть побитовый сдвиг влево вправо?

Если нет, то какое место они занимают во всей этой системе операторов?

[Ответить](#)



1. Chestor:

[1 июля 2019 в 23:29](#)

<< и >> это всё те же операции битового сдвига влева и вправо соответственно, НО, в пространстве имён std эти две операции ПЕРЕОПРЕДЕЛЕННЫ. Думаю, о переопределении, речь пойдёт в след. уроках и Вы узнаете, что это такое. Но можете сейчас погуглить)))

[Ответить](#)



1. zashiki:

[15 июля 2019 в 00:09](#)

Спасибо за ответ! "Гуглить сейчас" не стоило)

[Ответить](#)

Добавить комментарий

Ваш E-mail не будет опубликован. Обязательные поля помечены *

Имя *

Email *

Комментарий

☐ Сохранить моё Имя и E-mail. Видеть комментарии, отправленные на модерацию

☐ Получать уведомления о новых комментариях по электронной почте. Вы можете [подписаться](#) без комментирования.

[TELEGRAM](#)  [КАНАЛ](#)



[ПАБЛИК](#) 

ТОП СТАТЬИ

- [📖 Словарь программиста. Сленг, который должен знать каждый кодер](#)
- [🔌 Урок №1. Введение в программирование](#)
- [✍️ 70+ бесплатных ресурсов для изучения программирования](#)
- [📈 Урок №1: Введение в создание игры «Same Game»](#)
- [⚙️ Урок №4. Установка IDE \(Интегрированной Среды Разработки\)](#)

- [Ravesli](#)
- - [О проекте](#) -
- - [Пользовательское Соглашение](#) -
- - [Все статьи](#) -
- Copyright © 2015 - 2020