

Русификация языка Си++

проект

Русскоязычные эквиваленты для служебных слов Си++

ООО "Интерстрон"
Email: info@mlc.ru
Tel: (095) 976-2033

Общие положения	1
Общий список служебных слов Си++ согласно Стандарту.....	1
Принципы перевода служебных слов.....	2
Перевод служебных слов Си++	3
Перевод служебных слов препроцессора Си++	11

Общие положения

Одной из особенностей проекта С11 является возможность использовать символы кириллицы в программах на Си++. Эта возможность проявляется в следующих трех аспектах:

- * задание русскоязычных текстов в комментариях;
- * использование символов кириллицы в идентификаторах;
- * допущение русскоязычных эквивалентов служебных слов.

Первый аспект допускается Стандартом, при реализации не требует каких-либо дополнительных усилий и обеспечивается уже в нынешнем компиляторе.

Для поддержки символов кириллицы в идентификаторах необходима очень небольшая модификация компилятора. Очевидно, что необходимо обеспечить одновременное присутствие в программе как полностью кириллических и полностью латинских идентификаторов, а также идентификаторов, в составе которых имеются символы из обоих множеств.

Введение русскоязычных эквивалентов для служебных слов Си++ представляет собой более ответственную задачу; прежде всего это относится к качеству их перевода.

Общий список служебных слов Си++ согласно Стандарту

(73 слова в алфавитном порядке)

and	const_cast	friend	private	this
and_eq	continue	goto	protected	throw
asm	default	if	public	true
auto	delete	inline	register	try
bitand	do	int	reinterpret_cast	typedef
bitor	double	long	return	typeid
bool	dynamic_cast	mutable	short	typename
break	else	namespace	signed	union
case	enum	new	sizeof	unsigned
catch	explicit	not	static	using
char	extern	not_eq	static_cast	virtual
class	false	operator	struct	void
compl	float	or	switch	volatile
const	for	or_eq	template	wchar_t
				while
				xor
				xor_eq

Принципы перевода служебных слов

Основные принципы, которыми следует руководствоваться при выборе переводов конкретных слов:

1. Перевод не обязательно должен быть буквальным; прежде всего он должен обеспечивать прежде всего смысловую идентичность.
2. Перевод должен быть единообразным: например, если используются сокращения, то стиль сокращений должен быть единым.
3. Необходимо учесть особенности русского языка, прежде всего, наличие родов для прилагательных. Например, в объявлении вида

```
int a;
```

описатель типа `int` можно трактовать как "целая переменная", "целый объект", "целое данное". В таких случаях можно как отбрасывать родовые окончания, так и допускать несколько вариантов. Общее решение в данном случае состоит в следующем: основной (официальный) перевод подобных служебных слов представляет собой единообразное сокращение, полученное отбрасыванием окончаний. Наряду с официальным имеется ряд альтернативных переводов, которые поддерживаются компилятором, но не рекомендуются для активного использования.

4. Некоторые служебные слова можно переводить различными грамматическими формами: как существительные, как глаголы, как предлоги. Например, для `goto` допустимы следующие переводы:

```
иди  
идти  
на  
переход
```

В подобных случаях предпочтение дается **существительным**, как наиболее нейтральным формам. Это решение позволит избежать неопределенности в случае глаголов и ненужной эмоциональной окраски (например, для варианта "иди"). Так, перевод слова `continue` в глагольной форме вызовет дилемму вроде "продолжать" или "продолжить", или "продолж" (не говоря о не вполне точном соответствии этого глагола семантике оператора `continue`).

5. В некоторых случаях допускаются отступления от перечисленных правил, если они приводят к неудобочитаемым, неблагозвучным или семантически неточным переводам, а также если для данного слова имеется традиционная русскоязычная нотация.

Перевод служебных слов Си++

Ниже приводится предлагаемый перевод служебных слов. Служебные слова сгруппированы в соответствии со смысловыми связями.

В левой колонке дается оригинальное служебное слово Си++, во второй колонке - перевод, предлагаемый как основной, официальный. Третья колонка содержит альтернативные переводы. Наконец, в последней колонке представлены варианты переводов, которые рассматривались, но были отброшены по каким-либо причинам. Здесь же содержатся краткие объяснения этих причин и прочие комментарии.

Оригинальное служебное слово	Официальный перевод	Альтернативные переводы	Рассматривавшиеся и отброшенные варианты
<code>and</code>	<code>лог_и</code>	нет; традиционное обозначение - <code>&&</code>	Перевод обозначения операции <code>and</code> как <code>и</code> был отвергнут ввиду недопустимости служебных слов, состоящих из одной буквы. Поэтому для единообразия и другие операции - <code>or</code> , <code>xor</code> , <code>not</code> - переводятся несколько более громоздко, однако единообразно и в стиле языка Си/Си++. Следует иметь в виду, что в Стандарте данные обозначения для логических операций введены как альтернативы известным лексемам <code>&&</code> , <code> </code> , <code>!</code> , <code>^</code> и, скорее всего, не будут широко использоваться.
<code>and_eq</code>	<code>и_присв</code>	нет; традиционное обозначение <code>&=</code>	
<code>asm</code>	<code>ассем</code>	<code>ассемблер</code>	
<code>auto</code>	<code>автом</code>	<code>автоматич</code> <code>автоматический</code> <code>автоматическая</code> <code>автоматическое</code> <code>автоматические</code>	<code>авто</code> Предпочтение было отдано варианту <code>автом</code> , так как прямая транслитерация вызывает ненужные ассоциации (<code>авто</code> - распространенное сокращение для "автомобиль").
<code>bitand</code>	<code>бит_и</code>	нет; традиционное обозначение <code>&</code>	
<code>bitor</code>	<code>бит_или</code>	нет; традиционное обозначение <code> </code>	
<code>bool</code>	<code>лог</code>	<code>логич</code> <code>логический</code> <code>логическая</code> <code>логическое</code> <code>логические</code>	<code>бул</code> <code>булев</code> Обычно, в знак уважения в Д.Булю, основателю алгебры логики, логические объекты называют "булевскими". Однако, в русском переводе сокращения <code>бул</code> , <code>булев</code> выглядят не слишком благозвучно, поэтому был выбран нейтральный вариант <code>лог</code> .
<code>break</code>	<code>прервать</code>		<code>прерывание</code> <code>заверш</code> Для перевода слова <code>break</code> был выбран глагольный перевод <code>прерывание</code> , <code>прервать</code> . Предпочтение отдано глагольной форме, так как существительное <code>прерывание</code> несет специфическую нагрузку, подразумевая аппаратное прерывание.
<code>case</code>	<code>вариант</code>		<code>вар</code>
<code>catch</code>	<code>перехват</code>		См. <code>throw</code> .

<code>char</code>	сим	симв символьный символьная символьное символьные	симв литер	Описатель сим выглядит так же наглядно, как и симв , поэтому был выбран более компактный вариант. (Более точный вариант литер был отвергнут из-за множественных толкований этого слова.)
<code>class</code>	класс			Для слова <code>class</code> , обозначающего ключевое в Си++ понятие, невозможно предложить ничего, кроме прямого перевода.
<code>compl</code>	бит_не	нет; традиционное обозначение <code>~</code>		
<code>const</code>	конст	константа константный константная константное константные		
<code>const_cast</code>	конст_прив		конст_привед	На редкость неуклюжее имя; перевод неизбежно оказался таким же неуклюжим. Единственное утешение: эта конструкция используется относительно редко.
<code>continue</code>	итерация		еще итерация продолж	Как и во многих других случаях, вариант с существительными для слов <code>continue</code> , <code>return</code> снимает глагольную неопределенность. Кроме того, для слова <code>continue</code> буквальный перевод вида продолжить , продолжать и т.п. не позволяет выявить смысл соответствующего оператора; вариант итерация кажется наиболее предпочтительным, будучи одновременно недвусмысленным и семантически точным.
<code>default</code>	проч	прочие прочее	умолч	Аналогично <code>switch</code> , для слова <code>default</code> предпочтительным кажется не буквальный перевод - умолчание , - а более точный в данном контексте вариант прочее или проч . Последний вариант прямо соотносится с известной традицией и поэтому выглядит вполне естественно.
<code>delete</code>	удалить		удл удал	Комментарий см. в <code>new</code> .
<code>do</code>	цикл		выполни вып	Попытка перевести <code>do</code> буквально, как выполнить , выполни , вып (последнее вообще неприемлемо, так как вызывает ассоциации с "выпить", "выплюнуть" и т.д.) кажется существенно хуже по сравнению с семантически более точным существительным цикл , которое, к тому же, ввиду своей краткости не нуждается в сокращении.
<code>double</code>	двойн	двойной двойная двойное двойные двойн_точн двойной_точности	удв	Вариант был отброшен ввиду своей неочевидности.

<code>dynamic_cast</code>	<code>дин_прив</code>		<code>дин_привед</code>	На редкость неуклюжее имя; перевод неизбежно оказался таким же неуклюжим. Единственное утешение: эта конструкция используется относительно редко.
<code>else</code>	<code>иначе</code>			Все достаточно очевидно и понятно.
<code>enum</code>	<code>перечисл</code>	<code>перечисление</code>		Не слишком элегантный, но точный перевод. Полный вариант - <code>перечисление</code> - кажется громоздким, а все остальные варианты сокращений, кроме предложенного, - <code>пер</code> , <code>переч</code> и т.д. - явно неблагозвучны. Заметим, что практически во всех языках программирования, имеющих перечислимые типы, они вводятся без использования специальных служебных слов.
<code>explicit</code>	<code>явн</code>	<code>явная</code> <code>явный</code>		
<code>extern</code>	<code>внеш</code>	<code>внешний</code> <code>внешняя</code> <code>внешнее</code> <code>внешние</code>	<code>внешн</code>	Вариант <code>внешн</code> выглядит более элегантно по сравнению с более громоздким <code>внешн</code> .
<code>false</code>	<code>ложь</code>		<code>нет</code>	<code>ложь</code> - единственно возможный перевод. Вариант <code>нет</code> отдает любительщиной. См. также <code>true</code> .
<code>float</code>	<code>плав</code>	<code>плавающий</code> <code>плавающая</code> <code>плавающее</code> <code>плавающие</code>	<code>плв</code>	См. комментарий для <code>long</code>
<code>for</code>	<code>для</code>		<code>цикл_для</code>	Точный перевод в данном случае согласуется в русскоязычной традиции в программировании, идущей от первых российских компиляторов Алгола-60. В принципе, можно было бы предложить вариант <code>цикл_для</code> , дополнительно проясняющий семантику данного оператора, однако он был отвергнут из-за большей громоздкости.
<code>friend</code>	<code>друж</code>	<code>друг</code> <code>дружеств</code> <code>дружественный</code> <code>дружественная</code> <code>дружественные</code>	<code>дружеств</code>	Соблазн заменить буквальный перевод каким-либо более наглядным (хотелось дополнительно прояснить понятие дружественных функций и классов в Си++) был преодолен, так как все популярные описания Си++ оперируют именно этим названием. Введение другого, даже более наглядного обозначения привело бы к необходимости ссылок типа "в оригинале это свойство определяется писателем <code>friend</code> , что в русскоязычных книгах переводится как "дружественный".
<code>goto</code>	<code>переход</code>		<code>на иди</code> <code>идти</code>	Выбор сделан в пользу существительного. Помимо общего принципа предпочтительности существительных, относительно длинное слово для явно дискуссионного и для многих сомнительного оператора перехода кажется более оправданным, позволяя легче обнаружить его в тексте программы.
<code>if</code>	<code>если</code>			Все достаточно очевидно и понятно.

<code>inline</code>	подставл	подставляемая подставляемые	подст откр открыт	Предложенный вариант перевода нельзя считать окончательным ввиду его громоздкости (более компактный вариант подст отвергнут из-за ненаглядности); однако другого подходящего слова предложить пока не удалось. Заметим, что в русскоязычной программистской литературе, начиная с 60-х г.г. для подпрограмм данного вида использовался термин "открытая".
<code>int</code>	цел	целый целая целое целые		
<code>long</code>	длин	длинный длинная длинное длинные	длн	Дополнительные, помимо отбрасывания окончаний, сокращения внутри слова (длн , плв), кажутся неоправданными, даже если они продиктованы стремлением "втиснуть" все обозначения стандартных типов в три символа.
<code>mutable</code>	изменч	изменчивый изменяемая изменяемое изменяемые	измен	Описатель <code>mutable</code> говорит именно об <i>изменчивости</i> соответствующего члена класса: даже если объект этого класса объявлен как константный, данный его член может изменять свое значение, в отличие от других членов. Вариант измен был отвергнут из-за своей неочевидности.
<code>namespace</code>	область		обл	Полный перевод слова <code>namespace</code> - область - кажется более наглядным и предпочтительным по сравнению с обл (служебные слова заголовочного характера, каким является <code>namespace</code> , не стоит делать слишком короткими).
<code>new</code>	создать		нов новый	Точный перевод операций <code>new</code> и <code>delete</code> как новый (нов) и удалить несимметричен, что маскирует очевидную симметричность семантики этих операций. В русском переводе (то есть для русскоязычного пользователя языка) эта несимметричность проявится со всей очевидностью. Поэтому решение перевести <code>new</code> и <code>delete</code> в виде пары "взаимоисключающих" глаголов кажется более предпочтительным.
<code>not</code>	лог_не	нет; традиционное обозначение !		не ; см. комментарий к <code>and</code>
<code>not_eq</code>	не_равно	нет; традиционное обозначение !=		

`operator` операция

Английское слово `operator` в данном контексте однозначно переводится как "операция". Например, выражение `a+b` задает **операцию** сложения (обозначенную **знаком операции** - символом "+"); соответственно, языковая конструкция, позволяющая определить собственную версию **операции** сложения, содержит служебное слово **операция**. Вульгарный буквальный перевод `operator` как "оператор" затемняет смысл соответствующей конструкции языка и приводит к путанице, так как в русскоязычной программистской литературе понятием "оператор" исторически обозначается конструкция языка, задающая некоторое элементарное действие без образования нового значения (англоязычный прототип - `statement`). Именно такая трактовка слов `operator` и `statement` была принята в русском переводе книги Эллис и Страуструпа.

<code>or</code>	лог_или	нет; традиционное обозначение <code> </code>	или; см. комментарий к <code>and</code>
<code>or_eq</code>	или_присв	нет; традиционное обозначение <code> =</code>	
<code>private</code>	себе	скрытый скрытая скрытое скрытые	для_себя прив скрыт
<code>protected</code>	своим	защищ защищенный защищенная защищенное защищенные	для_своих защищ защ
<code>public</code>	всем	доступный доступная доступное доступные	для_всех пуб публ общедост

Для часто используемых в объектно-ориентированном программировании спецификаторов `private`, `public` и `protected` в качестве официальных предлагаются несколько нетривиальных по стилю переводы в виде притяжательных местоимений **себе**, **всем** и **своим**. Представляется, что такие трактовки явно точнее оригинальных, заметно повышают наглядность этих спецификаторов и позволяют легче запомнить их назначение. Их необычность кажется допустимой, кроме того, по той причине, что они и используются не вполне стандартным образом: не как модификаторы для каждого отдельного объявления (как в языке Java), а как своего рода заголовки внутри тела класса.

Варианты **для_себя**, **для_всех**, **для_своих** были отброшены ввиду своей неоправданной громоздкости. В списке представлены также традиционные (буквальные) переводы этих спецификаторов; кажется очевидным, что все они существенно хуже по тем или иным причинам.

<code>register</code>	<code>рег</code>	регистровый регистровая регистровое регистровые	регистр Был выбран сокращенный вариант перевода, который не вызывает неверных ассоциаций с регистрами аппаратуры. В противном случае объявление вида регистр A ; можно было бы трактовать как задание некоторого аппаратного регистра с обозначением A . Представляется, что сокращение рег менее прямолинейно и поэтому более соответствует семантике описателя как просто подсказке компилятору.
<code>reinterpret_cast</code>	<code>тип_прив</code>		тип_привед На редкость неуклюжее имя; перевод неизбежно оказался таким же неуклюжим. Единственное утешение: конструкция используется относительно редко. Для самого неуклюжего слова из группы xxx-cast-reinterpret_cast предложен чуть менее громоздкий, но несколько более точный перевод, нежели буквальный.
<code>return</code>	<code>возврат</code>		вернуть возвратить Как и во многих других случаях, вариант с существительными для слова return снимает глагольную неопределенность.
<code>short</code>	<code>кор</code>	короткий короткая короткое короткие	
<code>signed</code>	<code>со_знаком</code>	знаковый знаковая знаковое знаковые	знак Спецификаторы signed и unsigned имеют двойной смысл: во-первых, они обозначают знаковый (беззнаковый) целочисленные типы и в этом качестве могут использоваться независимо; во-вторых, они могут использоваться как дополнительные спецификаторы к спецификатору некоторого целочисленного типа, явно обозначая его (знаковый или беззнаковый) вариант, например, unsigned long i ; Перевод этих спецификаторов как прилагательных, очевидно, затруднителен: так, вариант знак не воспринимается как сокращение от "знаковый", а, скорее, может пониматься как некоторый отдельный тип знак , обозначающий, например, однобайтовые литеральные символы. Поэтому был выбран более громоздкий, но совершенно недвусмысленный вариант перевода.
<code>sizeof</code>	<code>размер</code>		Кто придумает что-то другое?
<code>static</code>	<code>стат</code>	статический статическая статическое статические	
<code>static_cast</code>	<code>стат_прив</code>		стат_привед На редкость неуклюжее имя; перевод неизбежно оказался таким же неуклюжим. Единственное утешение: эта конструкция используется относительно редко.
<code>struct</code>	<code>структ</code>	структура	Варианты с полным и сокращенным переводом слова struct примерно одинаково приемлемы с точки зрения наглядности.

<code>switch</code>	выбор	<p>перекл переключатель</p> <p>Для оператора выбора было решено отказаться от буквального перевода служебных слов. Использование для <code>switch</code> перевода типа перекл кажется несколько неуклюжим (не говоря уже о переключатель) и искажающим смысл. Традиционный перевод - выбор - обращается к русскоязычной традиции (идушей от Паскаля), заметно понятнее и компактнее. См. также <code>default</code>.</p>
<code>template</code>	шаблон	<p>Можно было бы проявить оригинальность и выбрать вариант вроде образец, однако это противоречило бы складывающейся традиции обозначения данного понятия в русскоязычной литературе.</p>
<code>this</code>	этот	
<code>throw</code>	исключение	<p>искл исключ передать передача ситуация</p> <p>В данном случае буквальный перевод служебных слов, предполагающий глагольную форму ("пытаться" для <code>try</code>, "поймать" для <code>catch</code>, "бросить" для <code>throw</code>) приводит к нелепым конструкциям. Вариант с существительными выглядит гораздо более стройным, естественным и не вызывающим посторонних ассоциаций. Кроме того, полный вариант слова исключение заметно лучше неуклюжих сокращений искл, исключ.</p> <p>Остаются сомнения, стоит ли конструкцию, обозначающую некоторое (вполне нетривиальное) действие - возбуждение исключительной ситуации, - обозначать несколькими аморфным словом исключение (которое, строго говоря, и не обозначает никакого действия). Быть может, слово передать или передача выглядит естественнее. (Варианты бросить, возбудить, возбуждение, равно как и их сокращения, были отброшены как абсолютно неприемлемые.)</p>
<code>true</code>	истина	<p>да</p> <p>истина - единственно возможный перевод. Вариант да отдает любительщиной.</p>
<code>try</code>	контроль	<p>Перевод слова <code>try</code> как контроль семантически более точен и соответствует переводу понятия <code>try-block</code> как "блок-с-контролем", предложенному в Зеленой книге (Эллис, Страуструп. "Справочное руководство по языку Си++ с комментариями").</p> <p>См. также <code>catch</code>, <code>throw</code>.</p>
<code>typedef</code>	тип	<p>оптип</p>

<code>typeid</code>	<code>есть_тип</code>		тип естьтип	Альтернативный, более буквальный перевод: <code>typedef</code> - оптип , <code>typeid</code> - тип был отвергнут по причине очень широкой распространенности конструкции <code>typedef</code> и, наоборот, относительной редкости <code>typeid</code> . Поэтому для частой конструкции был выбран более простой вариант тип , а для редкой операции <code>typeid</code> - более громоздкий, но более точный перевод <code>есть_тип</code> .	
<code>typename</code>	<code>имя_типа</code>		имятипа		
<code>union</code>	<code>совмещ</code>	<code>совмещение</code>	союз объедин	объединение	Все варианты буквальных переводов слова <code>union</code> (союз, объединение) отвергнуты из-за того, что они скрывают назначение конструкции (конструкция <code>union</code> не "объединяет" свои члены, а, скорее, допускает "совмещение" этих членов в памяти). Так что в этом отношении предпочтительнее перевод <code>совмещение</code> или его сокращенный вариант.
<code>unsigned</code>	<code>без_знака</code>	<code>беззнаковый</code> <code>беззнаковая</code> <code>беззнаковое</code> <code>беззнаковые</code>	беззнак	См. также <code>signed</code>	
<code>using</code>	<code>использ</code>	<code>использовать</code>	исп	Сокращение вида <code>исп</code> может приводить к двусмысленности ("использовать" или "исполнить").	
<code>virtual</code>	<code>вирт</code>	<code>виртуальный</code> <code>виртуальная</code> <code>виртуальное</code> <code>виртуальные</code>	виртуал		
<code>void</code>	<code>пуст</code>	<code>пустой</code> <code>пустая</code> <code>пустое</code> <code>пустые</code>			
<code>volatile</code>	<code>ненадеж</code>	<code>ненадежный</code> <code>ненадежная</code> <code>ненадежное</code> <code>ненадежные</code>	подвиж	Обычно следует избегать использования прилагательных с приставкой "не". Однако в данном случае использование слова с "не" выглядит оправданно, так как сама семантика данного описателя описывается им более точно: считается, что значение объекта, объявленного с данным описателем, может изменяться асинхронно по отношению к потоку управления программы; в этом смысле объект является ненадежным, то есть нельзя делать никаких предположений о его значении в любой момент времени. В то же время он не является "подвижным": его положение в памяти не может изменяться.	
<code>wchar_t</code>	<code>шсим</code>	<code>широкий_симв</code> <code>широкая_симв</code> <code>широкое_симв</code> <code>широкие_симв</code>	ш_сим	шир_сим	Вариант <code>шсим</code> кажется естественнее буквального <code>ш_сим</code> или <code>шир_сим</code> (последнее неприемлемо ввиду аналогий типа "ширяться").
<code>while</code>	<code>пока</code>				

`xor` `искл_или` нет; традиционное обозначение `^` `бит_искл_или` `мод2`

Перевод обозначений операций `^` и `^=` сделан несколько отличным от других аналогичных операций ввиду сложного названия данной операции - "исключающее или" в западной терминологии или "сложение по модулю 2" в российской математической традиции. Полный вариант `бит_искл_или`, хотя он и сочетается с другими операциями `бит_и`, `бит_или`, `бит_не`, был отброшен ввиду своей громоздкости, тем более, что "логического" варианта этой операции нет.

`xor_eq` `искл_присв` нет; традиционное обозначение `^=`

Перевод служебных слов препроцессора Си++

Оригинальное служебное слово	Официальный перевод	Альтернативные переводы	Рассматривавшиеся и отброшенные варианты
<code>#define</code>	<code>#макрос</code>		<p><code>#опред</code> <code>#определить</code></p> <p>В данном случае предпочтение было отдано существительным, в первую очередь, потому, что это позволило избежать сокращений и получить краткие и выразительные названия.</p> <p>Заметим, что "макрос" - корректная форма единственного числа для данного понятия. Именно такой перевод английского <i>macro</i> был принят в отечественной профессиональной литературе, начиная с 70-х гг. (см., например, М.Кемпбел-Келли, "Введение в макросы": Пер. с англ./ Под ред. Э.З.Любимского.- М.: Сов. Радио, 1978). Общеупотребительный в устной речи (иногда проникающий и в печатные тексты) вульгаризм "макро" возник только в последнее время.</p>
<code>defined</code>	<code>есть</code>		<p>Если директива <code>#ifdef</code> переводится как <code>#если_есть</code>, то некоторую ее модификацию естественно перевести аналогично. Пример: <code>#если (есть M1 есть M2) ...</code></p>
<code>#elif</code>	<code>#инес</code>	<code>#иначе_если</code>	<p><code>#ин_если</code> <code>#ин_ес</code></p> <p>Перевод директивы <code>#elif</code> как <code>#инес</code> выглядит наиболее кратко и вместе с тем читабельно. Кроме того, именно так традиционно переводится служебное слово <code>elif</code> в тех языках программирования, в которых оно имеется (например, в Алгол-68).</p>
<code>#else</code>	<code>#иначе</code>		<p>Было бы странно, если бы директивы <code>#if</code> и <code>#else</code> были бы переведены как-либо иначе, имея ввиду одноименные служебные слова самого языка.</p>
<code>#endif</code>	<code>#конес</code>	<code>#конец_если</code>	<p><code>#кон_если</code> <code>#конесли</code> <code>#илсе</code></p> <p>Директива <code>#endif</code> переведена как <code>#конес</code>, - в духе <code>#если</code> и <code>#инес</code>. "Стильный" перевод как <code>#илсе</code> - перевернутой начальной директивы, сам по себе очень привлекательный (и используемый, в частности, для аналогичного служебного слова в Алголе-68), был отвергнут, так как в данном случае он противоречил бы общему стилю данных предложений.</p>

<code>#error</code>	<code>#ошибка</code>		
<code>#if</code>	<code>#если</code>		
<code>#ifdef</code>	<code>#если_есть</code>	<code>#еслиесть</code>	<code>#если_опред</code>
<code>#ifndef</code>	<code>#если_нет</code>	<code>#еслинет</code>	<code>#если_не_опред</code>
		<p>Предлагаемый перевод <code>#ifdef</code> <code>#ifndef</code> как, соответственно, <code>#если_есть</code> и <code>#если_нет</code>, представляется наиболее точным из возможных: директива "срабатывает", если ранее уже было (еще не было) определение макроса с указанным именем.</p>	
<code>#include</code>	<code>#вставка</code>	<code>#вставить</code>	<code>#внести</code>
		<p>Здесь также использовано существительное; быть может, перевод <code>#вставить</code> так же нагляден и точен, но для единообразия (чтобы директивы определения макроса, его отмены и использования представлялись существительными) была выбрана указанная форма.</p>	
<code>#line</code>	<code>#строка</code>		
<code>#pragma</code>	<code>#прагмат</code>	<code>#прагма</code>	
		<p>Практически единственный возможный перевод. Вариант <code>#опция</code>, в принципе, тоже допустим, но он несет оттенок жаргона, используемого в устной речи. К предложенному варианту перевода полностью относится комментарий для директивы <code>#макрос</code> (имея в виду распространенность вульгарного перевода "прагма").</p>	
<code>#undef</code>	<code>#отмена</code>	<code>#отмен</code>	<code>#отменить</code>
		<p>Предпочтение было отдано существительным; см. <code>#define</code>.</p>	
<code>__DATE__</code>	<code>__ДАТА__</code>		
<code>__FILE__</code>	<code>__ФАЙЛ__</code>		
<code>__LINE__</code>	<code>__СТРОКА__</code>		
<code>__TIME__</code>	<code>__ВРЕМЯ__</code>		
<code>__cplusplus</code>	<code>__СИПЛЮСПЛЮС</code>		
<code>__STDC__</code>	<code>__СТАНДСИ__</code>	<code>__СТДСИ__</code>	
		<p>Имена предопределенных макросов либо имеют совершенно однозначный и краткий перевод, либо представляют собой неудобопроизносимые сокращения, возникшие исторически по произволу создателей языков Си и Си++. Поэтому не следует надеяться исправить дело, пытаясь ввести для них некие "красивые" эквиваленты на русском языке.</p>	

