сишные трюки
(0xE выпуск)

крис касперски ака мыщъх, a.k.a. souriz, a.k.a. nezumi, a.k.a. elraton, no-email

сегодняшний выпуск трюков посвящен двум любопытным, но малоизвестным возможностям языка си — "триграфам" (trigraph) и "диграфам" (digraph), в основном встречающихся в соревнованиях по непонятному программированию, однако, в некоторых (достаточно редких) случаях "разваливающих" программу, написанную без учета их существования

совет 1 - триграфы: жизнь без скобок

Три- и диграфы широко используются в натуральных языках для обозначения "чужеродных" символов, отсутствующих в "своем" алфавите. Последовательность из двух (реже — из трех) "своих" символов кодирует один "чужой". Просто как и все гениальное! Например, хангыль (корейское фонематическое письмо) состоит из блоков типа чамо, кодирующих отдельные слоги или даже целые слова. Всего существует 51 чамо, 24 из которых эквивалентны буквам обычного алфавита, а оставшиеся 27 представляют собой комбинации из двух или трех букв (т. е. диграфы и триграфы соответственно).

Язык Си использует девять символов, не входящих в наборы ISO 646 и EBCDIC до сих пор используемые в некоторых терминалах. В результате, квадратные и фигурные скобки невозможно ни набрать с клавиатуры, ни отобразить на экране такого терминала, а потому, начиная с самых первых редакций Стандарта, в язык ввели поддержку ***триграфов***, "скрестив" символы базового набора ISO 646 с двумя знаками вопроса (см. таблицу 1).

триграф	эквивалент
??=	#
??/	\
??'	^
??(	[
??)	]
??!	|
??<	{
??>	}
??-	~

Таблица 1 триграфы и соответствующие им символы

Программа, написанная с использованием триграфов, может выглядеть, например, так (см. листинг 1):

```
??=include <stdio.h>                          /* #            */

int main(void)
??<                                           /* {            */
        char n??(5??);                        /* [ и ]        */

        n??(4??) = '0' - (??-0 ??' 1 ??! 2);  /* ~, ^ и |     */
        printf("%c??/n", n??(4??));           /* ??/ = \      */
        printf("??=??=??=");                  /* ###          */
??>
```

Листинг 1 исходный текст программы, написанной с использованием триграфов

Попробуйте подсунуть эту абракадабру коллегам и спросите, что она делает. Обратите внимание на строку, выделенную полужирным шрифтом: здесь тригафы используются внутри строковых констант и вместо вывода ожидаемых "??=??=??=" функция printf напечатает три символа решетки!!! А ведь не зная о существовании триграфов, о такую комбинацию можно спотыкнуться чисто случайно, долго ломая голову почему программа работает не так, как это задумывалось.

Триграфы поддерживают практически все современные компиляторы, однако, если Microsoft Visual C++ задействует триграфы по умолчанию, то Borland C++ для увеличения скорости трансляции использует внешний препроцессор, реализованный в файле trigraph.exe, входящий в штатный комплект поставки компилятора и вызываемый программистом самостоятельно. GCC поддерживает триграфы, но по умолчанию не обрабатывает их внутри строковых констант и делает это только при явном указании ключа "`-trigraphs`".

Подробнее о триграфах можно почитать в Стандарте на Си, любом хорошем учебнике или на Википедии — http://en.wikipedia.org/wiki/C_trigraph.

совет 2 – диграфы: элегантный мир

Недостаточная выразительность и отвратительная читабельность триграфов привели к тому, что в последней редакции Стандарта ANSI C99 появилась достойная альтернатива в виде ***диграфов***, использующих всего два символа вместо трех, комбинируя угловые скобки со знаком процента или двоеточия, интерпретируемых как квадратные и фигурные скобки соответственно. Согласитесь, что так нагляднее (причем намного) и психологически гораздо естественнее!

Решетка ("#") кодируется двоеточием следующим за знаком процента (см. таблицу 2), остальные же символы – "\", "^", "|" и "~" не получили адекватной репрезентации и по-прежнему должны кодироваться через триграфы, что, впрочем, не создает большой проблемы в силу их невысокой распространенности (по сравнению с фигурными и угловыми скобками).

Конечно, тут можно поспорить, попинать разработчиков Стандарта и вообще развести флейм на шестьсот квадратных миль, но… против Священного Писания (Стандарта в смысле) не попрешь, несмотря на то, что в настоящее время не имеется ни одного компилятора, в полной мере поддерживающего C99.

диграф	эквивалент
<:	[
:>	]
<%	{
%>	}
%:	#
%:%:	##

Таблица 2 диграфы и соответствующие им символы

Пример программы, написанной с использованием диграфов (точнее на смеси диграфов и триграфов) приведен ниже:

```
%:include <stdio.h>                          /* #           */

int main(void)
<%                                           /* {           */
        char n<:5:>;                         /* [ и ]       */

        n<:4:> = '0' - (??-0 ??' 1 ??! 2);   /* ~, ^ и |    */
        printf("%c??/n", n<:4:>);            /* ??/ = \     */
        printf("%:%:%:");
??>
```

Листинг 2 исходный текст программы, написанной с использованием ди- и триграфов

Попытка ее трансляции компиляторами Microsoft Visual C++ и Borland C++ вызывает сообщение об ошибке и проваливается. Ничего не поделаешь — эти компиляторы диграфов не переваривают! Последние версии GCC выполняют постановку диграфов везде, за исключением строковых констант, причем, попытка использования ключа "`-digraphs`" не решает проблемы, поскольку, данный ключ предназначен для вывода внутренней отладочной информации, генерируемой компилятором в ходе трансляции программы и главным образом интересной его разработчикам.

В Сети встречается достаточно много исходных текстов программ, написанных под UNIX (а UNIX — это синоним GCC) с использованием диграфов. Возникает резонный вопрос — и как же это чудо прогресса портировать на Windows? Можно, конечно, посоветовать версию GCC для win32, но это будет плохой совет, особенно если из программы требуется вырезать всего один кусок, надеясь вставить его в готовый проект на Microsoft Visual C++, который, между прочим, компилятор GCC скорее всего не захочет транслировать, особенно,

если программист активно использовал нестандартные расширения от Microsoft (а, поскольку, редко кто изучает Си по Стандарту, практически все мы используем те или иные расширения, зачастую, даже не подозревая об их нестандартности).

На самом деле, решение состоит в создании простейшего внешнего препроцессора, выполняющего "сквозной" поиск диграфов и заменяющий их на соответствующие им символы. Если препроцессор писать лень, эту операцию можно осуществить в любом текстовом редакторе через Replace.

совет 3 — диграфы и шаблоны

В отношении Си диграфы лексически нейтральны. Они не нарушают целостность языка и гарантируют отсутствие коллизий — ни при каких обстоятельствах, никакой диграф не может "пересекаться" ни с одной "родной" конструкцией языка, что совсем неудивительно, поскольку, именно для Си диграфы и разрабатывались.

А вот язык Си++, претендующий на обратную совместимости с Си, при работе с диграфами наталкивается на серьезные проблемы, приобретая неоднозначность интерпретации — одна и та же конструкция может быть прочитана двояко в зависимости от того выполнять ли в ней подстановку диграфов или нет.

Рассмотрим ситуацию на следующем примере. Допустим, в глобальном пространстве имен (global namespace), мы имеем класс, именуемый (для определенности) X. Допустим так же, что мы хотим передавать класс X какому-нибудь другому классу в качестве аргумента (например, классу "`std::vector`"), причем, передавать не абы как, а непременно в виде шаблона (template), ведь шаблоны это, во-первых, очень модно, а, во-вторых, жутко (не)удобно! Но, как бы там ни было (о вкусах не спорят) — задача поставлена и ее надо решать.

Очевидное решение — написать "`std::vector<::X>`" и на _некоторых_ компиляторах это будет работать. Как вы уже, наверное, поняли этими компиляторами окажутся Microsoft Visual C++, Borland C++, ранние версии GCC… то есть все те, кто не поддерживает диграфов и потому трактует конструкцию "`std::vector<::X>`" весьма однозначно.

Проблемы возникают при попытке скормить эту штуку свежим версиям GCC (или любому другому компилятору с поддержкой диграфов). Комбинация "<:" заменяется на "[", в результате чего вся конструкция превращается в "`std::vector[:X>`", выдавая ошибку транслятора и вызывая естественное недоумение программиста: что здесь не так?! Ведь еще вчера компилировалось!!!

Одно из возможных решений состоит в разделении "<" и "::X>" символом пробела. Конструкция принимает вид "`std::vector< ::X>`" и конфликтов с диграфами уже не возникает.

Анализ исходных текстов, выловленных на бескрайних просторах Сети, показывает, что очень многие Си++ программы страдают подобными конфликтами и отказываются компилироваться свежими версиями GCC. Забавно, но некоторые разработчики прямо указывают требуемую версию транслятора в FAQ, предостерегая от использования более новых ("багистных") версий. На самом деле, это не баг. Это фича! И теперь вы знаете как с ней обращаться.