++i + ++i — Lurkmore

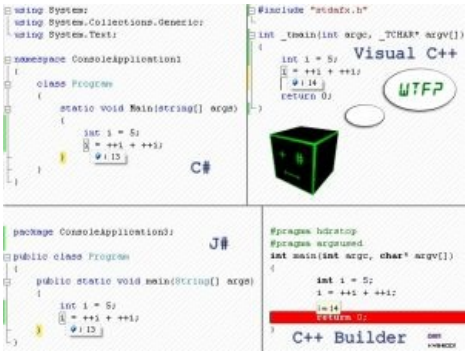
A long time ago, in a galaxy far, far away...

События и явления, описанные в этой статье, были давно, и помнит о них разве что пара-другая олдфагов. Но Анонимус не забывает!

ZOMG TEH DRAMA!!!11

Обсуждение этой статьи неиллюзорно доставляет не хуже самой статьи. Рекомендуем ознакомиться и причаститься, а то и поучаствовать, иначе впечатление будет неполным.

`int i = 5; i = ++i + ++i;` — типичный код-майндфак для программистов.



Строгое описание происходящего

В данном примере происходит неоднократное изменение переменной в пределах одной точки следования, такая ситуация описывается в стандартах C и C++ как UB. Иными словами, даже попытки ответить на этот вопрос иначе как «UB» демонстрируют недостаточную квалификацию отвечающего. Другое дело, что после «UB» можно указать некоторые подробности, и мы этим займёмся, поскольку не утоленное вовремя любопытство приводит к драмам в обсуждении.

Конкретно неопределённость в этой, как некоторым кажется, кристально ясной конструкции в данном случае заключается в том, что, согласно стандартам C и C++, побочные эффекты (то есть инкремент в данном случае) могут быть применены в *любой* удобный для компилятора момент между двумя точками следования. Конструкцию `i = ++i + ++i;` компилятор вправе понять и как

```
tmp=i; tmp++; i = tmp; tmp++; i += tmp;
```

и как

```
tmp=i; tmp++; tmp++; i = tmp + tmp;
```

и какими-нибудь другими способами. Нужна такая свобода для проведения низкоуровневых оптимизаций в обычных случаях типа `a = ++b + ++c;`, дабы между делом сэкономить пару тактов на халяву.

Хотя, оптимизатор тут вообще ни при чём. Дело в том, что наше интуитивное понимание работы этого кода основывается на том, что прединкремент возвращает *значение*, получившееся после прибавления единицы. На самом же деле любой нормальный прединкремент возвращает не получившееся значение, а *ссылку* на эту же переменную. Поэтому мы складываем не числа, а две одинаковые ссылки, то есть переменную `i` саму с собой! Иными словами происходит буквально следующее:

1. Левый `++i` прибавляет единицу к i и возвращает ссылку на неё. I = 6.
2. Правый `++i` прибавляет ещё одну единицу к i и также возвращает ссылку на неё. I = 7.
3. Оператор сложения разыменовывает ссылки, получая `i = i + i`. Так как после второго шага I = 7, то извлекается именно это число, давая выражение `i = 7 + 7`, откуда и получается 14.

Нельзя, но если всё-таки сделаем?

Почему нельзя, мы уже видели (даже наиболее «альтернативно одаренные» не возьмутся оспаривать, что это ведет ко глюку). Однако всякий пытливый ум, несомненно, изучит этот глюк в поисках того, **как именно** может сглючить данный код, **в какую сторону** стандарты языка допускают маневры компилятора, а в какую — таки нет.

Варианты «правильного» ответа колеблются между 13 и 14, хотя на LISPe (пруф) анонимусы вроде как получили 12, 13 (пруф), а на Питоне вообще 10 (поскольку там вообще нет оператора инкремента ++). Результат зависит от последовательности операций. Там, где результат 13, сначала вычисляется первый инкремент, на место первого операнда идёт 6, вычисляется второй инкремент, туда идет 7, и результаты складываются: 6+7=13. 14 можно получить, вычислив оба инкремента, а потом уже сумму. Таким образом, этот трюк хорошо использовать для ошеломления быдлокодеров, уверенных в непогрешимости своего уютненького язычка.

Вообще, по определению предынкремент выполняется **до** вычисления выражения. Если считать ++i + ++i одним выражением, то мы должны выполнить два инкремента, потом вычислить выражение и получить 14. Если считать каждый ++i выражением, а сумму — выражением, принимающим результаты других выражений как аргументы, то, с точки зрения логики и математики, ничего таки не изменится, а вот компилятор, вычислив одно слагаемое (имеет право — самостоятельное выражение ведь!), упустит тот момент, что при вычислении второго такого же самостийного слагаемого первое тоже, сцуко, подло изменилось. Указать синтаксисом, где начинается и заканчивается выражение, невозможно — скобки задают приоритет внутри выражения. «Выражение» — другая инстанция. Например, в A=(B=C+14) есть выражение C+14, результат которого присваивается B, причем присваивание — тоже является выражением, и его результат присваивается A. И никакие скобки не могут побудить ЭТО стать одним выражением. Разъединить выражение, напротив, можно: (a=++i) + (b=++i), что напрямую задает вычисление a, вычисление b и только потом вычисление суммы a+b. Это, теоретически, должно детерминировать значение a+b как 13 (если компилятор настолько строго выдерживает формальные определения языка), но зато содержит UB относительно значений a и b, поскольку оптимизировать порядок вычисления a и b компилятору никто не запрещает (см. точки следования). Очевидно, что значение (a=++i) / (b=++i) по этой причине вообще не детерминировано.

Оно на башорге

Исходная цитата

> KoloDen
>
> Привет, я общительный пацан, люблю поболтать, особенно с классными девченками. Но, чтобы поговорить со мной, ответьте на простую задачку анти-спам бота. Вот она: int i = 5; i = ++i + ++i; Вопрос: Чему равно i?
>
> Stefmania 14
>
> KoloDen Гы. Признайся, ты не девченка, а 40-летний одмин, да?

Последствия

11 мая 2007 года случилось страшное — была заапрувлена вышеприведенная цитата. С тех пор разнокалиберные программисты потеряли покой и сон. Дело в том, что в зависимости от используемого языка программирования данное выражение может давать и 13, и 12, и еще больше 9000 вариантов ответа.

Вот одно из множества обсуждений

> Anhen, 11.05.2007 14:03:27:
>
> KoloDen Привет, я общительный пацан, люблю поболтать, особенно с классными девченками. Но, чтобы поговорить со мной, ответьте на простую задачку анти-спам бота. Вот она: int i = 5; i = ++i + ++i; Вопрос: Чему равно i?
>
> Stefmania 14
>
> KoloDen Гы. Признайся, ты не девченка, а 40-летний хрен одмин, да?
>
> DarkMist, 14:03:54: хм 8-)
>
> Anhen, 14:06:01: что хм? 13 или 14?
>
> DarkMist, 14:06:32: бля я завис. то что 14 это точно а вот почему я не могу понять
>
> Anhen, 14:07:43: тогда откуда ты знаешь что точно 14?
>
> DarkMist, 14:09:18: бля 8-) я понял
>
> DarkMist, 14:09:26: сцуко, хитро 8-)
>
> Anhen, 14:09:27: ну?
>
> Anhen, 14:09:48: ну?!?!
>
> DarkMist, 14:09:48: откуда знаю что 14: perl -e "$i = 5; $i = ++$i + ++$i; print $i"
>
> DarkMist, 14:10:08: почему 14: пришлось открывать вижи и смотреть асмовый код
>
> Anhen, 14:10:12: и?

> DarkMist, 14:11:39: когда вычисляется выражение, сначала вычисляются его операнды но оператор ++i - это не i + 1, а i += 1 то есть сначала к i прибавляется 1, потом к i еще раз прибавляется единица а потом к i прибавляется i получается 7+7, то есть 14
>
> Anhen, 14:12:33: черт красиво!
>
> Anhen, 14:13:32: а вот в пхп 13
>
> DarkMist, 14:39:02: пхп сосет 8-)
>
> Anhen, 14:39:21: пхп логичен
>
> DarkMist, 14:40:06: в данном случае - правильный ответ 14, он согласуется с логикой, а вот 13 — нет
>
> Anhen, 14:40:39: имхо 13 логичнее (5+1) + (5+1)+1
>
> DarkMist, 14:48:04: еще раз. ++i это не i + 1 это так же логично как обман зрения. только здесь не обман зрения, а инерция мышления
>
> Anhen, 14:49:24: i += 1 это i = i+1
>
> DarkMist, 14:50:10: да
>
> DarkMist, 14:51:22: исходный statement выглядит так: i = (i += 1, i) + (i += 1, i) в этом случае все смотриться логично, aren't you?
>
> Anhen, 14:58:45: я остановила работу всего джавского отдела
>
> Anhen, 14:58:52: сидят пытаются получить 14
>
> DarkMist, 15:00:27: на жабе шо ле 8-)?
>
> Anhen, 15:00:59: ага
>
> Anhen, 15:01:07: у них 13
>
> Anhen, 15:01:13: и куча теорий
>
> Anhen, 15:02:28: не знаю, наша контора считает что логика на стороне 13
>
> Anhen, 15:07:09: на флексе 13 подключились дельфисты :))))))
>
> Anhen, 15:07:20: бугага башорг зло
>
> Anhen, 15:08:49: и в сишарпе 13
>
> Anhen, 15:09:13: а у дельфистов нет инкрементов
>
> DarkMist, 15:21:28: на перле и c++ 14. все остальное от лукавого 8-)
>
> Anhen, 15:22:04: мои коллеги просили передать, что ты сволочь и башорг твой блядский тоже цытата
>
> DarkMist, 15:23:52: 8-)))

— *http://darky2000.livejournal.com/72865.html*

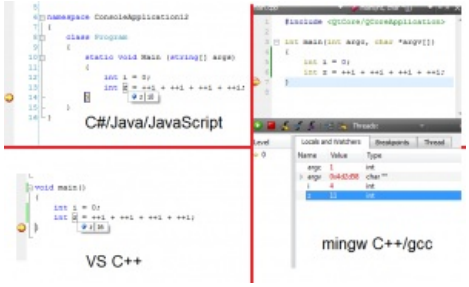
Реализации

C и C++

В этих языках может получиться и 13, и 14, и вообще чёрт-те-что. Разные компиляторы C++ выдают 13 и 14. Это пример неопределённого поведения. Неопределённое поведение — самый типичный для этих языков способ выстрелить себе в ногу.

Более того, один и тот же компилятор может выдавать разные значения в зависимости от опций оптимизатора. Некоторые об этой фигне еще и предупреждают, например gcc -Wall выдает warning: operation on ‘i’ may be undefined.

Даже в одной и той же программе может получаться разный результат:



Демонстрация прекрасной портабельности исходников между

```
int i=5,j=5;
i=++i+ ++i;
printf("i=%i j=%i", i, ++j + ++j); //Вывод: i=14  j=13
```

компиляторами

Вот описанный майндфак в квадрате на плюсах:

```
int i = 0; int z = ++i + ++i + ++i + ++i;
```

Алсо, в C++17 сабж уже не даёт UB (описали-таки поведение в стандарте), так что правильный ответ — 14 (оба раза). Можно проверить на вышеприведённом коде в любом онлайн компиляторе.

Java

В Java получается 13 (первый пре-инкремент увеличивает i на 1 и возвращает 6, второй пре-инкремент увеличивает i на 1 и возвращает 7), и данное поведение жёстко определено, так как в спецификации языка чётко описан порядок вычисления (не путать с приоритетом операторов).

C#

Как истинный клон Джавы, даёт нам результат с числом 13.

Objective-C

Компилятор clang, который является де-факто стандартом (а других просто нет), предупреждает о Multiple unsequenced modifications to 'i', но исправно возвращает 13.

awk

Awk, как и Жаба, выдаёт 13:

```
awk 'BEGIN { i=5;j=5;i=++i+ ++i; print i, ++j+ ++j}'

13 13
```

ActionScript 3.0

```
var test:int = 5;
trace(++test + ++test);//13
test = 5;
trace(test++ + test++);//11
```

ActionScript 2.0

```
i=5; trace(++i + ++i); //13
i=5; trace(i++ + i++); //11
```

Perl

Perl выдает 14:

```
my $i = 5;
$i = ++$i + ++$i;
print $i;
```

Pawn

```
main()
{
        new i = 5, j = 5;
        i = ++i + ++i;

        printf "%d %d", i, ++j + ++j;
}
```

Вывод: `13 13`

PHP

PHP выдаёт 13:

```
$i = 5;
$i = ++$i + ++$i;
echo $i;
```

JavaScript

JavaScript в V8 выдаёт 13:

```
i = 5
i = ++i + ++i
document.write(i)
```

Bash

GNU bash 4.1.5 тоже выдаёт 13:

```
~$ i=5; echo $((++i + ++i))
13
```

Forth

Никаких UB. Слова выполняются строго последовательно. Что написано, то и будет получено.

```
5 1+ dup + .
```

Выдает 12.

```
5 1+ dup 1+ + .
```

Выдает 13.

Fortran 90

Ожидаемое 13 ((а эти товарисчи и инкремент написать нормально не могут))

```
    program mindfuck
    integer i
    i = 5
    i = INC (i) + INC (i)
    print *,i
    contains
    integer function INC (i)
    integer,intent(inout)::i
            i=i+1
            INC = i
    end function INC
    end program mindfuck
```

Common Lisp

CLISP 2.49 возвращает 13, поведение определено в спецификации.

```
[1]> (defvar i 5)
I
[2]> (+ (incf i) (incf i))
13
```

Python

Python выдаёт 10, в нем нет инкремента в таком виде:

```
i = 5
i = ++i + ++i
print i
```

Но при этом, если реализовать инкремент самим, будет 13:

```
class Foo:
    def __init__(self, num):
        self.num = num

    def inc(self):
        self.num += 1
        return self.num

i = Foo(5)
print(i.inc() + i.inc())
```

Это потому, что интерпретатор вычисляет все по порядку.

Но если чуток подумать и довести все до логического ума, то все таки 14

```
class Foo:
    def __init__(self, num):
        self.num = num

    def inc(self):
        self.num += 1
        return self

    def __add__(self, right):
        return Foo(self.num + right.num)

    def __repr__(self):
        return repr(self.num)

i = Foo(5)
print(i.inc() + i.inc())
```

VB

Visual Basic выдаёт 10 тоже:

```
i = 5
i = ++i + ++i
log.WriteLine("i = " & i)
```

Powershell

Powershell выдаёт 13:

```
$i=5
$i=++$i + ++$i
Write-Host $i
```

Rexx

Rexx выдаёт 10

```
i=5
i=++i + ++i
SAY i
```

Delphi

В Delphi нельзя присвоить значение результата инкремента, потому что инкремент — не функция, но процедура, посему сия операция будет несовместима по типу, и компилятор откажется собирать программу, ругнувшись [Error] Incompatible types: 'Integer' and 'procedure, untyped pointer or untyped parameter' Однако, если реализовать инкремент как функцию, чтоб она работала именно так, как надо (изменяя переменную, от которой запускается), выдаст 13

```
function MyInc(var i: integer): integer;
begin
  inc(i);
  Result:=i;
end;

begin
  i:=5
  i:=MyInc(i){Теперь i=6}+MyInc(i){А теперь i=7}; // i:= 6 + 7;
  Writeln(i); Readln // i= 13;
end;
```

Pascal

В Паскале нет оператора ++, но есть два оператора + (как в математике): бинарный (например, 2+2) и унарный (например, +2). Итого получается: +i — это +5, то есть то же 5; ++i — это +(+5) — ну, если мы очень настаиваем, чтобы у 5 ни в коем случае не меняли знак, почему бы и нет? В итоге, команда математически превращается в +(+5)+(+(+5))=10. Если реализовать инкремент как функцию, выдается 13.

```
program ippProblem;

var
        i:Cardinal;
```

```
function inc(var x:Cardinal):Cardinal;
begin
        x := x + 1;
        inc := x;
end;

begin
        i := 5;
        writeln(inc(i) + inc(i));
end.
```

Развитие идеи

Утверждается, что есть люди, которым приведённого выражения недостаточно для полной потери церебральной девственности. Такие люди могут захотеть отступить от классики и изучить вопрос последствий выражения

Not Safe For ~~Work~~ Sanity

```
 int i = 0; i += i++ + ++i
```

K&R определил += как разновидность именно оператора присваивания, а не краткую форму записи i=i+..., то есть к инстанции «выражение» относится всё, что справа от +=. Поэтому += можно не бояться, так как предынкремент вычисляется до значения выражения, затем вычисляется выражение, затем — сразу же и никак иначе — постинкремент, а потом уже мы покидаем инстанцию данного выражения и используем его результат по назначению (в данном случае — для операции +=). А вот само выражение по-прежнему UB, потому что разные варианты разбиения его на отдельные инстанции самостоятельных выражений опять по той же схеме дают разные значения.

А ещё можно сделать так:

```
 static int i = i++ + ++i;
```

Результат — 2 или 1, но хуй кто поймёт.
А с помощью Free Pascal Compiler дельфисты могут присоединиться к толпам любителей инкрементов. Не получится. Все эти ваши плюсы в строке fpc воспринимает как обычный унарный плюс. Бинарная операция сложения тут ровно одна, в итоге при попытке записать си-подобный вариант

```
program increment_ub;
var i : Integer;
begin
        i := 5;
        i := ++i + ++i;
        WriteLn(i);
end.
```

результат ожидаемо оказывается 10.

А если переопределить стандартную процедуру inc() следующим образом:

```
function inc(i:word):word;
begin
system.inc(i);
inc:=i;
end;
```

то получим 12. Потому что, в отличие от «++i», аргумент не изменяется. А вот если перед параметром поставить ключевое слово «var», то результат будет 13

Башорг

++i + ++i Aalien Asuka Bash.im BB-код CAPSLOCK Creator DarkRider IT happens Jozhig T9 Zadolba.li Zoi Админ Аппрув Бездна Биоланте Блондинка Бросить лом в унитаз поезда Взрывающийся вертолёт Влад Чесноков Вордстримовские войны Если трактористы — женщины Зеленоград Извините за неровный почерк Конина блядская Котомальчик Криветко Локальные мемы башорга Лопата Минет На башорг Накипело Патчить KDE2 под FreeBSD Пельмени Плюсообмен Пчёлы против мёда Рекурсия Урановые ломы Чувак, купивший доллары Шредер Я кончил и закурил Ящик пива Ящитаю

Языки программирования

++i + ++i 1C AJAX BrainFuck C Sharp C++ Dummy mode Erlang Forth FUBAR God is real, unless explicitly declared as integer GOTO Haskell Ifconfig Java JavaScript LISP My other car Oracle Pascal Perl PHP Prolog Pure C Python RegExp Reverse Engineering Ruby SAP SICP Tcl TeX Xyzzy Анти-паттерн Ассемблер Быдлокодер Выстрелить себе в ногу Грязный хак Дискета ЕГГОГ Индусский код Инжалид дежице Капча КОИ-8 Костыль Лог Метод научного тыка Очередь Помолясь Проблема 2000 Программист Процент эс Рекурсия Свистелки и перделки Спортивное программирование СУБД Тестировщик Умение разбираться в чужом коде Фаза Луны Фортран Хакер

Языки программирования