Урок №40. Инкремент, декремент и побочные эффекты

Юрий |

- Уроки C++

|

Обновл. 7 Сен 2020 |

49026

| 24

На этом уроке мы рассмотрим, что такое инкремент и декремент в языке C++, а также разберемся с таким понятием, как «побочные эффекты».

Оглавление:

Инкремент и декремент

Операции **инкремента** (увеличение на 1) и **декремента** (уменьшение на 1) переменных настолько используемые, что у них есть свои собственные операторы в языке C++. Кроме того, каждый из этих операторов имеет две версии применения: префикс и постфикс.

Оператор	Символ	Пример	Операция
Префиксный инкремент (пре-инкремент)	++	++x	Инкремент x, затем вычисление x
Префиксный декремент (пре-декремент)	−−	−−x	Декремент x, затем вычисление x
Постфиксный инкремент (пост-инкремент)	++	x++	Вычисление x, затем инкремент x
Постфиксный декремент (пост-декремент)	−−	x−−	Вычисление x, затем декремент x

С операторами инкремента/декремента версии префикс всё просто. Значение переменной x сначала увеличивается/уменьшается, а затем уже вычисляется. Например:

```
1 int x = 5;
2 int y = ++x; // x = 6 и 6 присваивается переменной y
```

А вот с операторами инкремента/декремента версии постфикс несколько сложнее. Компилятор создает временную копию переменной x, увеличивает или уменьшает оригинальный x (не копию), а затем возвращает копию. Только после возврата копия x удаляется. Например:

```
1 int x = 5;
2 int y = x++; // x = 6, но переменной y присваивается 5
```

Рассмотрим вышеприведенный код детально. Во-первых, компилятор создает временную копию x, которая имеет то же значение, что и оригинал (5). Затем увеличивается первоначальный x с 5 до 6. После этого компилятор возвращает временную копию, значением которой является 5, и присваивает её переменной y. Только после этого копия x уничтожается. Следовательно, в вышеприведенном примере мы получим y = 5 и x = 6.

Вот еще один пример, показывающий разницу между версиями префикс и постфикс:

```
1  #include <iostream>
2
3  int main()
4  {
5      int x = 5, y = 5;
6      std::cout << x << " " << y << std::endl;
7      std::cout << ++x << " " << --y << std::endl; // версия префикс
8      std::cout << x << " " << y << std::endl;
9      std::cout << x++ << " " << y-- << std::endl; // версия постфикс
10     std::cout << x << " " << y << std::endl;
11
12     return 0;
13 }
```

Результат выполнения программы:

```
5 5
6 4
6 4
6 4
7 3
```

В строке №7 переменные x и y увеличиваются/уменьшаются на единицу непосредственно перед обработкой компилятором, так что сразу выводятся их новые значения. А в строке №9 временные копии (x = 6 и y = 4) отправляются в **cout**, а только после этого исходные x и y увеличиваются/уменьшаются на единицу. Именно поэтому изменения значений переменных после выполнения операторов версии постфикс не видно до следующей строки.

Версия префикс увеличивает/уменьшает значения переменных перед обработкой компилятором, версия постфикс — после обработки компилятором.

Правило: Используйте префиксный инкремент и префиксный декремент вместо постфиксного инкремента и постфиксного декремента. Версии префикс не только более производительны, но и ошибок с ними (по статистике) меньше.

Побочные эффекты

Функция или выражение имеет **побочный эффект**, если она/оно изменяет состояние чего-либо, делает ввод/вывод или вызывает другие функции, которые имеют побочные эффекты.

В большинстве случаев побочные эффекты являются полезными:

```
#include <iostream>

int main()
{
    int x = 5;
    ++x;
    std::cout << x;

    return 0;
}
```

В примере, приведенном выше, оператор присваивания имеет побочный эффект, который проявляется в изменении значения переменной `x`. Оператор `++` имеет побочный эффект инкремента переменной `x`. Вывод `x` имеет побочный эффект внесения изменений в консольное окно.

Также побочные эффекты могут приводить и к неожиданным результатам:

```
#include <iostream>

int add(int x, int y)
{
    return x + y;
}

int main()
{
    int x = 5;
    int value = add(x, ++x); // здесь 5 + 6 или 6 + 6?  Это зависит от компилятора

    std::cout << value; // результатом может быть 11 или 12

    return 0;
}
```

Язык C++ не определяет порядок, в котором вычисляются аргументы функции. Если левый аргумент будет вычисляться первым, то `add(5, 6)` и результат — `11`. Если правый аргумент будет вычисляться первым, то `add(6, 6)` и результат — `12`! А проблема то кроется в побочном эффекте одного из аргументов функции add().

Вот еще один пример:

```
#include <iostream>

int main()
{
    int x = 1;
    x = x++;
    std::cout << x;
```

```
8
9       return 0;
10  }
```

Какой результат выполнения этой программы? Если инкремент переменной x выполняется до операции присваивания, то ответ — 1. Если же после операции присваивания, то ответ — 2.

Есть и другие случаи, в которых C++ не определяет порядок обработки данных, поэтому в разных компиляторах могут быть разные результаты. Но даже в тех случаях, когда C++ и уточняет порядок обработки данных, некоторые компиляторы все равно вычисляют переменные с побочными эффектами некорректно. Этого всего можно избежать, если использовать переменные с побочными эффектами не более одного раза в одном стейтменте.

Правило: Не используйте переменную с побочным эффектом больше одного раза в одном стейтменте.

Оценить статью:

(**296** оценок, среднее: **4,89** из 5)

Урок №39. Арифметические операторы

Урок №41. Условный тернарный оператор, оператор sizeof и Запятая

Комментариев: 24

1.  *Владислав*:
   17 июля 2020 в 19:00

   Вариант x = x++ выдаёт 1. Чисто логически это может выглядеть так:
   Сначала вычисляется выражение x, т.к была получена его копия. Затем идет инкремент, присваивается значение 2 для оригинала x. Но поскольку, выражение было вычислено с результатом 1, то после присвоения оригиналу 2 идёт присвоение вычисленного выражения. Ровно также, например, и для x = (x++)*3, выражение примет вид 1, затем оригиналу присвоится 2, затем выражение примет вид 1*3 = 3 и на выходе x будет иметь значение 3.

   Ответить
2.  *ХейЛонг*:
   19 апреля 2020 в 19:33

   x=x++ вообще не несёт адекватного смысла. инкременты декременты — это упрощённая запись присваивания. то бишь мы тут записали сразу два присваивания одной переменной, и пытаемся разобрать их порядок. мне кажется, что ответ будет одинаковый, потому что это выглядит как x = x = x = (x+1). инкременция происходит для переменной независимо от того, произойдёт она до

или после первого присваивания.
я кстати посчитал x = x++ + ++x — x— — —x; получилось нечто =)

Ответить

3.  *Алексей*:
[28 февраля 2020 в 12:39](#)

Сделал такой пример:

```
int x = 1;
int y = 0;
y = ((x++)*3);
std::cout << x << endl;
std::cout << y << endl;
```

Ответ:

2
3

Люди в комментариях немного разъяснили механизм работы постфиксного инкремента.

Ответить

4.  *Алексей*:
[28 февраля 2020 в 12:04](#)

Попробовал видоизменить немного пример

```
int x = 1;
x = ((x++)*3);
std::cout << x;
```

Ответ получил: 3

Помоему это вообще дичь какая то.

Ответить

1.  *ХейЛонг*:
[19 апреля 2020 в 19:50](#)

хочешь посмеяться? у меня тут ответ получается 4.
сначала 1x3 = 3, а потом вместо увеличения икс просто добавляется к ответу. я подумал, что компилятор устал от моих экспериентов и начал меня троллить.

Ответить

1.  *Александр*:
[13 июля 2020 в 15:08](#)

Хаха, у меня тоже 4
-_-

Ответить

5.  *Владимир*:
[19 ноября 2019 в 01:51](#)

Непонятной является именно логика разработчиков C++ относительно фактической и задекларированной приоритетности постинкремента. С какой целью у него указывается почти наивысший теоретический приоритет, если по факту он самый низкий. Какое значение для нас имеет факт того, что компилятор ка бы уже прибавил еденицу, если мы можем это увидеть только после завершения других операций? Как нас это может согреть и для чего такой преподвыподверт с приоритетами? Почему не указать ему наинизший приоритет — и все стало бы вроде логично?

Ответить

1.  *Богдан*:
[9 февраля 2020 в 21:00](#)

Полностью с вами согласен.

Ответить

6.  *Сергей*:
[2 апреля 2019 в 20:57](#)

Для тех кто не понял постфиксный инкремент.

```
int main()
{
    int x = 2;
    int y = x++ + x;

    // Первым делом x++ заменится на текущее значение 2
    // int y = 2 + x;
    // При этом сам x увеличится сразу, то есть станет 3

    // Соответсвенно когда он будет браться в следующий раз он уже будет 3
    // int y = 2 + 3;

    // На вывод пойдет 5
    cout<<y;

    return 0;
}
```

Постфиксный инкремент увеличивает переменную сразу, но в качестве результата отдает ее предыдущее значение.

Ответить

7.  *Илья*:
[27 декабря 2018 в 17:09](#)

Юрий, мне осталось непонятным, почему

```
x = x++;
```

вызывает неоднозначность?

Укажите, пожалуйста, на ошибку в моих рассуждениях:

приоритет у постфиксного инкрементирования "++" выше,
чем у оператора присваивания "=",
следовательно, первым будет выполняться постинкремент;

постинкремент, как Вы и писали:

"Во-первых, компилятор создает временную копию x, которая имеет то же значение, что и оригинал (5). Затем увеличивается первоначальный x с 5 до 6. После компилятор вычисляет временную копию, значение которой — 5, и присваивает это значение переменной y. Затем копия удаляется."

т.е. оператор постинкремента:
1. делает резервную копию объекта x;
2. производит инкрементирование переменной x;
3. при выходе/завершении возвращает по значению резервную копию объекта x в точку вызова (выражение в правой части оператора присваивания);

к моменту выполнения оператора присваивания
слева от знака "=" находится уже инкрементированная переменная x
справа — временный объект, являющийся копией ещё не инкрементированного значения переменной x

т.е. оператор присваивания просто перезаписывает инкрементированное значение x неинкрементированным.

Ответить

1.  *Константин*:
[11 марта 2019 в 19:55](#)

Илья, ты жим штанги лёжа в зале когда-нибудь практиковал? Как ты считал повторения? Толкнул"ХУХ" в верхней точке произносишь "РАЗ", толкнул"ХУХ" в верхней точке произносишь "ДВА" и т.д. Правильно? Так вот это префиксный инкремент. А вот если ты толкнул"ХУХ"опустил и произнёс "РАЗ" — это постфиксный инкремент. Хочешь глубже понять эту тему — по-больше блинов на штангу надень!

Ответить

1.  *Юрий*:

11 марта 2019 в 21:35

Аахах, шедевр)

Ответить

1.  *Константин*:
24 марта 2019 в 14:22

Юр, ну ответь: C++ не позволяет выводить надписи в какую переменную я заношу значение, когда инициализирую константы в namespace в режиме run-time? Или ты ещё просто не знаешь, как это делается? (Или это только уже на платном курсе познаётся?)

8.  *Константин*:
20 сентября 2018 в 01:41

Привет, Юр!
Пожалуйста дай примеры "непоняток" помимо ин/декрементов. Спасибо!

Ответить

9.  *artem*:
9 июня 2018 в 22:51

Насчет правила не использовать постфиксный инкремент и постфиксный декремент: Когда изучал циклы for, while и do while везде использовали именно постфиксный инкремент и декремент, даже при разработке мини-игр.

Ответить

1.  *Юрий*:
11 июня 2018 в 13:27

То, как писать код — дело каждого лично. В уроках рассматриваются аспекты, которые следует знать и понимать.

Ответить

10.  *Иван*:
23 марта 2018 в 09:37

```
1 int x=1;
2 x=x++;
```

А приоритет у '++' выше ведь, чем у '='? Почему будет неоднозначность?

Ответить

1. *Юрий*:
[23 марта 2018 в 17:05](#)

Да, приоритет у "++" выше, нежели в "=". И **у всех** компиляторах сначала выполнится операция инкремента, результат же зависит от того, в каком направлении будет выполняться следующая операция, т.е. операция присваивания. В случае с:

```
1 int x = 1;
2 x = x++
```

Сначала выполняется операция инкремента, так как у неё приоритет выше — получим:

```
1 int x = 1;
2 x = x++;  // x++ = 2
```

И здесь уже важно то, в каком направлении будет выполняться операция присваивания, либо x++ (значение 2) будет присваиваться переменной x — результат 2, либо переменная x (значение 1) будет присваиваться x++ — результат 1. То есть, либо справа налево, либо слева направо. А это уже вопрос отдельно к каждому компилятору и его разработчику. Но операция инкремента (++) выполняется первой в любом случае.

[Ответить](#)

1.  *Сергей*:
[12 апреля 2018 в 10:23](#)

Юрий, почему происходит нарушение "логики", если у нас программа выполняется сверху вниз, то сначала присваивается переменная, потом происходит инкремент, а потом она (переменная) выводится на экран? Разве на экран не должно выводиться инкрементированное число?

[Ответить](#)

1. *Юрий*:
[14 апреля 2018 в 19:28](#)

В статье об этом ведь рассказывается. Есть две версии инкремента: постфикс и префикс.

В версии префикс число сразу увеличивается и выводиться уже увеличенное:

```
1 int x = 5;
2 int y = ++x; // x = 6 и 6 присваивается переменной y
```

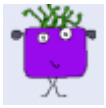
В версии постфикс значение сначала присваивается, затем вычисляется (увеличивается):

```
1 int x = 5;
2 int y = x++; // x = 6 и 5 присваивается переменной y
```

Видите плюсы? Они находятся в разных сторонах от переменной x — так можно легко вычислять результат (если плюсы находятся после переменной, то увеличиваться значение будет после выполнения операции присваивания и наоборот).

Если бы версии постфикс и префикс вычислялись одинаково, то зачем тогда нужны были бы две версии? Сделали бы один общий инкремент/декремент, да й делов то. Но есть случаи, где нужно использовать версии постфикс, есть случаи, где нужно использовать версии префикс — поэтому есть разделение на версии.

Никакого нарушения логики нет, просто нужно понимать разницу, установленную языком C++, между версиями постфикс/префикс операций инкремента/декремента.

2.  *Владимир*:
[28 ноября 2018 в 12:58]

Так вы же в 38 уроке сами указали, что операция присваивания имеет ассоциативность R->L. Следовательно, именно значение инкремента присвоится переменной x. Причём не говорилось, что такие приоритеты имеют какие-то определённые компиляторы, почему здесь есть вопрос к разработчику и компилятору — не понимаю. В том же уроке даже было задание 'B', где как раз и был случай с инкрементом, и ответ там был точно по такому же принципу.

[Ответить]

3.  *Sasha*:
[9 января 2020 в 19:40]

В таблице по-моему был столбец ассоциативность. Для присваивания справа налево, почему же разные компиляторы видят это по-разному

[Ответить]

11.  *Anton*:
[4 февраля 2018 в 02:09]

Странно всё, в первой половине текста рассказывается, что за чем вычисляется, а во второй уже опровержение в стиле "первая часть текста смысла не несёт, так как всё это великий китайский рандом, зависящий от компилятора". Этот урок закинул в дальний уголок памяти, так как пока всё равно в коде ничего сложнее простого повышения значения на единичку не использую.

[Ответить]

1.  *Юрий*:
[4 февраля 2018 в 22:00]

В первой части урока объясняется "инкремент" и "декремент". Во второй части речь идёт о другом понятии — "побочные эффекты". Какое здесь опровержение и чего? Опровержение

не выполнять инкремент и декремент или что? Есть префикс, есть постфикс, есть побочные эффекты. Китайского рандома нет.

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.

Отправить комментарий

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)

- - Все статьи -
-