

КАК СТАТЬ АВТОРОМ



Карьерные консультации

Опрос: какие темы вы хотите вид...



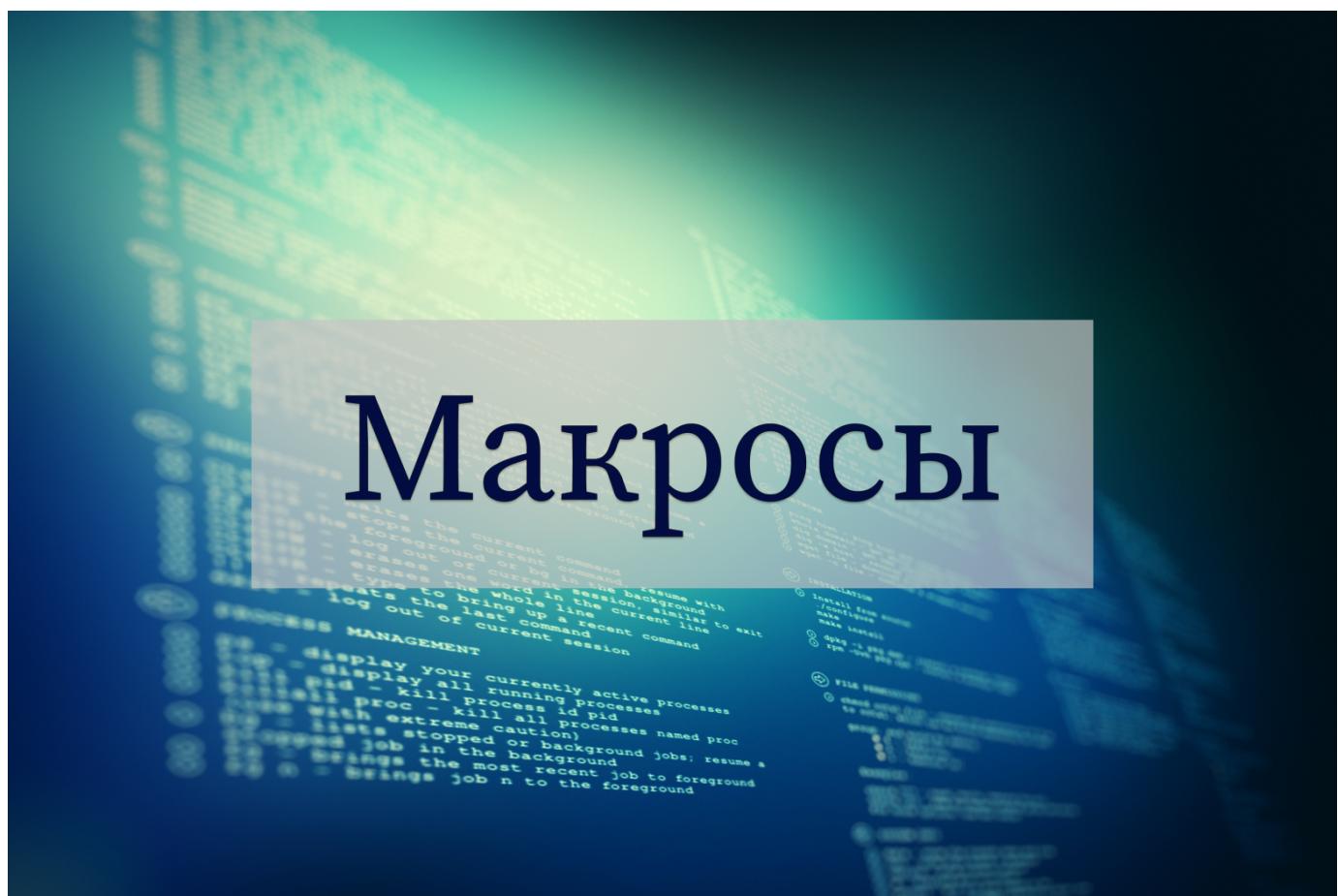
zer0chance 14 марта 2021 в 19:54

Макросы в С и С++

C++ *, C *

Из песочницы

Tutorial



Макросы – один из моих самых любимых инструментов в языках С и С++. Умные люди и умные книжки советуют по максимуму избегать использования макросов, по возможности заменяя их шаблонами, константами и inline-функциями, и на то есть веские основания. С помощью макросов можно создавать не только изящный код, но и плодить не менее изящные баги, которые потом будет очень сложно отловить и пофиксить. Но если соблюдать ряд несложных правил при работе с макросами, они становятся мощным оружием, которое не стреляет по твоим собственным коленям. Но сперва разберемся, что вообще такое макросы в С и С++?

Что есть макросы?

В языках С и С++ есть такой механизм, как *препроцессор*. Он обрабатывает исходный код

программы до того, как она будет скомпилирована. У перпроцессора есть свои директивы, такие как `#include`, `#pragma`, `#if` и тд. Но нам интересна только директива `#define`. В языке Си довольно распространенной практикой является объявление глобальных констант с помощью директивы `#define`:

```
#define PI 3.14159
```

А потом, на этапе препроцессинга, все использование PI будут заменены указанным значением:

```
double area = 2 * PI * r * r;
```

После препроцессинга, который по сути является банальной подстановкой, это выражение превратится в:

```
double area = 2 * 3.14159 * r * r;
```

PI – макрос, в самом простом его исполнении. Естественно, макросы в таком виде не работают как переменные. Им нельзя присваивать новое значение или использовать их адрес.

```
// Так нельзя:  
PI = 3;           // после препроцессинга: 3.14159 = 3  
int *x = &PI;      // после препроцессинга: int *x = &3.14159
```

О макросах важно понимать, что область видимости у них такая же, как у нестатических функций в языке Си, то есть они видны везде, куда их "зайнклюдили". Однако в отличии от функций, объявление макроса можно отменить:

```
#undef PI
```

После этой строчки обращаться к PI будет уже нельзя.

Макросы с параметрами

Самое интересное начинается, когда у макросов появляются параметры. Параметры в макросах работают примерно так же, как аргументы функции. Простой пример – макрос, который определяет больший из переданных ему параметров:

```
#define MAX(a, b) a >= b ? a : b
```

Макрос может состоять не только из одного выражения. Например макрос, который меняет значения двух переменных:

```
#define SWAP(type, a, b) type tmp = a; a = b; b = tmp;
```

Поскольку мы первым параметром передаем тип, данный макрос будет работать с переменными любого типа:

```
SWAP(int, num1, num2)  
SWAP(float, num1, num2)
```

В подобных макросах, вместо передачи типа аргументов первым параметром, полезно использовать оператор *typeof* в языке С или *decltype* в С++. С их помощью можно удобно объявлять переменную *tmp* того же типа, что и переданные аргументы:

```
#define SWAP(a, b) decltype(a) tmp = a; a = b; b = tmp;
```

Макросы также можно записывать в несколько строк, но тогда каждая строка, кроме последней, должна заканчиваться символом '\':

```
#define SWAP(a, b) \  
    decltype(a) tmp = a; \  
    a = b; \  
    b = tmp;
```

Параметр макроса можно превратить в строку, добавив перед ним знак '#':

```
#define PRINT_VAL(val) printf("Value of %s is %d", #val, val);
int x = 5;
PRINT_VAL(x) // -> Value of x is 5
```

А еще параметр можно при克莱ить к чему-то еще, чтобы получился новый идентификатор. Для этого между параметром и тем, с чем мы его склеиваем, нужно поставить '##':

```
#define PRINT_VAL (number) printf("%d", value_##number);
int value_one = 10, value_two = 20;
PRINT_VAL(one) // -> 10
PRINT_VAL(two) // -> 20
```

Техника безопасности при работе с макросами

Есть несколько основных правил, которые нужно соблюдать при работе с макросами.

1. Параметрами макросов не должны быть выражения и вызовы функций.

Ранее я уже объявлял макрос MAX. Но что получится, если попытаться вызвать его вот так:

```
int x = 1, y = 5;
int max = MAX(++x, --y);
```

Со стороны все выглядит нормально, но вот что получится в результате макроподстановки:

```
int max = ++x >= --y ? ++x : --y;
```

В итоге переменная *max* будет равна не 4, как мы ожидали, а 3. Потом можно уйму времени потратить, отлавливая эту ошибку. Так что в качестве аргумента макроса нужно всегда передавать уже конечное значение, а не какое-то выражение или вызов функции. Иначе выражение или функция будут вычислены столько раз, сколько используется этот параметр в теле макроса.

2. Все аргументы макроса и сам макрос должны быть заключены в скобки.

Это правило я уже нарушил при написании макроса MAX. Что получится, если мы захотим использовать этот макрос в составе какого-то математического выражения?

```
int result = 5 + MAX(1, 4);
```

По логике, переменная result должна будет иметь значение 9, однако вот что мы получаем в результате макроподстановки:

```
int result = 5 + 1 > 4 ? 1 : 4;
```

И переменная result внезапно примет значение 1. Чтобы такого не происходило, макрос MAX должен быть объявлен следующим образом:

```
#define MAX(a, b) ((a) >= (b) ? (a) : (b))
```

В таком случае все действия произойдут в нужном порядке.

3. Многострочные макросы должны иметь свою область видимости.

Например у нас есть макрос, который вызывает две функции:

```
#define MACRO() doSomething(); \
doSomethingElse();
```

А теперь попробуем использовать этот макрос в таком контексте:

```
if (some_condition) MACRO()
```

После макроподстановки мы увидим вот такую картину:

```
if (some_condition) doSomething();
doSomethingElse();
```

Нетрудно заметить, что под действие `if` попадет только первая функция, а вторая будет вызываться всегда. Именно для того, чтобы избежать подобных багов, у макросов должна быть объявлена своя область видимости. Для удобства в этих целях принято использовать цикл `do {} while (0);`.

```
#define MACRO() do { \
    doSomething(); \
    doSomethingElse(); \
} while(0)
```

Поскольку в условии цикла стоит ноль, он отработает ровно один раз. Это делается, во первых, для того, чтобы у тела макроса появилась своя область видимости, ограниченная телом цикла, а во вторых, чтобы сделать вызов макроса более привычным, потому что теперь после `MACRO()` нужно будет ставить точку с запятой. Если бы мы просто ограничили тело макроса фигурными скобками, точку с запятой после его вызова поставить бы не получилось.

```
if (some_condition) MACRO();
```

Все потоки Разработка Администрирование Дизайн Менеджмент Маркетинг Научноп



Банальный пример – объявим несколько функций сложения для работы с разными типами данных:

```
#define DEF_SUM(type) type sum_##type (type a, type b) { \
    type result = a + b; \
    return result; \
}
```

Теперь чтобы натянуть таких функций для нужных нам типов, нужно просто использовать пару раз этот макрос в глобальной зоне видимости.

```
DEF_SUM(int)
DEF_SUM(float)
DEF_SUM(double)

int main() {
    sum_int(1, 2);
    sum_float(2.4, 6, 3);
    sum_double(1.43434, 2, 546656);
}
```

Таким образом у нас получился аналог шаблонов из С++. Но стоит сразу обратить внимание, что данный способ не подойдет для типов, название которых состоит более чем из одного слова, например `long long` или `unsigned short`, потому что не получится нормально склеить название функции (`sum_##type`). Для этого сперва придется объявить для них новый тип, состоящий из одного слова.

В современном С++ можно спокойно обходиться без макросов вовсе, используя только шаблоны и `inline`-функции. Но в Си жить с макросами все же удобнее, чем без них. При грамотном использовании макросы позволяют избавиться от большого количества дублирования кода и сделать сам код более симпатичным и удобочитаемым.

[Мой блог в Телеграмме](#)

Теги: Си, С++, С, С++, macro, define, #define, макросы

 +2

 29K

 74



Редакторский дайджест

Присыпаем лучшие статьи раз в месяц



Электропочта



6

0

Карма Рейтинг

Евгений Игнатенко @zer0chance

[Telegram](#)

Реклама



Комментарии 56



- o frobeniusfg 14.03.2021 в 20:32

Кажется, в разделе "Еще немного примеров" пропущен кусок кода.

+2 Ответить



- o bogolt 14.03.2021 в 20:44

Банальный пример – объявим несколько функций сложения для работы с разными типами данных:

Расскажите лучше о примерах которые нельзя реализовать через шаблоны, что куда более надежно чем макросы.

#define SWAP(type, a, b) type tmp = a; a = b; b = tmp;

тут вы оставили переменную 'tmp' в общей области видимости, да и кроме того добавлять тип в этот макрос выглядит не очень красивым решением, уж лучше что-то вроде

```
#define SWAP(a, b) do{ decltype(b) tmp = a; a = b; b = tmp;}while(0)  
Но и это ерунда, потому что в таком случае проще использовать шаблонную функцию, а еще проще взять std::swap
```

+2 Ответить



- o DGG 14.03.2021 в 21:05

КДПВ – 7 (семь) мегабайт!

Зачем?

+4 Ответить



- o FrozenWalrus 15.03.2021 в 04:10

Чтобы ещё раз показать, что макросы – зло.

+4 Ответить



kovserg 14.03.2021 в 21:08

Ну куда же без макросов:

```
#define container_of(ptr, type, member) ({\n    const typeof(((type *)0)->member) * __mptr = (ptr);\\n    (type *)((char *)__mptr - offsetof(type, member)); })
```

Но вообще внешние шаблонизаторы и скрипты для генерации кода могут сделать куда больше и иногда значительно удобней. Например загрузку opengl разных ревизий.

ps: Что бы визуально отличать макросы их рекомендуют называть большими буквами.

+1 Ответить



rsashka 14.03.2021 в 21:09

Именно для того, чтобы избежать подобных багов, у макросов должна быть объявлена своя область видимости. Для удобства в этих целях принято использовать цикл do {} while (0);.

Цель конструкции `do {} while (0);` немного другая. Не объявить область видимости, это и обычными скобками можно сделать

Цикл do/while уникален тем, что является единственной (почти) в языке С грамматической конструкцией, которая формирует блок, и при этом всегда безусловно заканчивается на ;. Благодаря этой замыкающей; мы можем более естественным образом использовать do {...} while (0) внутри составных макросов для объединения их в единый statement.

+2 Ответить



zer0chance 14.03.2021 в 23:43

Этот момент тоже описан в статье в следующем после прокомментированного вами абзаце.

-1 Ответить



Kotofay 15.03.2021 в 02:30

А {...} не делают то же самое?

0 Ответить



 **rsaspka**

15.03.2021 в 02:43

Если оставить только скобки, то в коде будет ошибка компиляции

```
if (true)
    MACRO(foo); // <- "Лишняя" `;`
else // <- Ошибка: оторванный `else`  
/* что-то еще */;
```

**+4**

Ответить



...

**neurocore**

14.03.2021 в 21:44



Лучше – совсем избегать макросы. Из-за огромного числа способов внести побочные эффекты, а также из-за того, что макросы не могут иметь никакую другую область видимости кроме глобальной.

**0**

Ответить



...

**DistortNeo**

14.03.2021 в 22:07

Ну в С это невозможно, а вот современный С++ предоставляет богатые возможности, чтобы единственным сценарием использования макросов была условная компиляция.

**0**

Ответить



...

**neurocore**

14.03.2021 в 23:55

Как это невозможно. Описать функцию вместо макроса? Другое дело, что вас могут пугать незначительные издержки на вызов функции, что в 99% не является критическим местом.

**-3**

Ответить



...

**DistortNeo**

15.03.2021 в 00:34

Не, ну учитывая, что макросы в конечном итоге разворачиваются в код, логично, что без макросов обойтись можно. Просто в случае С это будет жуткое дублирование кода, а это сама по себе плохая практика.

**0**

Ответить



...

**neurocore**

15.03.2021 в 00:39

Что-то я в упор не вижу кейс, где функция вместо макроса приведёт к дублированию кода. Было бы неплохо если бы вы привели пример.

**-2**

Ответить



...

○  DistortNeo
15.03.2021 в 01:24

Да хоть тот же макрос SWAP из статьи. В случае С вам придётся объявить по соответствующей функции для каждого из типов.

◆ +1 Ответить



○  neurocore
15.03.2021 в 10:08

Так, хорошо, в С нет шаблонов, тут соглашусь. Но что заставляет разработчиков в 2021 году писать на С? Разве что для поддержки старого кода.

◆ -8 Ответить



○  Siemargl
15.03.2021 в 15:47

Спроси (или почитай ответ) Линуса или эмбедчиков.

◆ +4 Ответить



○  Tujh
15.03.2021 в 20:08

Вся встройка, VoIP, Linux kernel и многое чего ещё до сих пор на чистом С, без плюсов.

◆ +1 Ответить



○  staticmain
31.03.2021 в 16:04

Простой и надежный язык, если использовать его правильно и регулярно проверяться valgrind и статикой. Намного понятнее С++, быстрее, меньше требует. На TIOBE вышел на первое место, сместив Java на второе.

◆ 0 Ответить



○  staticmain
15.03.2021 в 01:53

- X Macro и все его плюшки.
- Получение номера строки и имени текущего файла исходников для вызываемой функции (например лога)

◆ +3 Ответить



○  Amotum
15.03.2021 в 00:43

Еще макросы, в отличие от лямбд и онлайн-функций, позволяют менять control-flow (т.е. делать `return` или `break` из внешней области).

+3 Ответить



sergegers 15.03.2021 в 00:55

В С++ интроспекция типов, например, пока не может быть сделана без макросов.

0 Ответить



НЛО прилетело и опубликовало эту надпись здесь

neurocore 14.03.2021 в 23:59

Разумеется разные языки. Всё ещё считаю макросы bad practice, учитывая что есть безопасные аналоги: функции и те же перечисления для констант.

-10 Ответить



Tzimie 15.03.2021 в 01:45

Вот один из моих любимых примеров использования макросов. Программа, кстати, очень серьезная: это победитель конкурса на программу на С не длиннее 512 символов без пробелов, которая генерит максимальное число (значность `int` бесконечная)

```
loader.c
define R { return
define P P (
define L L (
define T S (v, y, c,
define C ),
define X x)
define F );}
int r, a;
P y, X
R y - ~y << x;
}
Z (X
R r = x % 2? 0: 1 + Z (x / 2 F
L X
R x / 2 >> Z (x F

define U = S(4,13,-4,
T t)
```

```
{
int
f = L t C
x = r;
R
f - 2?
f > 2?
f - v? t - (f > v) * c: y:
P f, P T L X C
S (v+2, t U y C c, Z (X)))
:
A (T L X C
T Z (X ) F
}
A (y, X
R L y) - 1
? 5 << P y, X
: S (4, x, 4, Z @ F
```

```
define B (x /= 2) % 2 && (
D (X
{
int
f,
d,
c = 0,
t = 7,
u = 14;
while (x && D (x - 1 C B 1))
d = L L D (X ) C
f = L r C
x = L r C
c - r || (
L u) || L r) - f ||
B u = S (4, d, 4, r C
t = A (t, d) C
f / 2 & B c = P d, c C
t U t C
u U u) )
C
c && B
t = P
~u & 2 | B
u = 1 << P L c C u) C
P L c C t) C
c = r C
u / 2 & B
^ ^ + ^ ^
```

```
c = r l, c c  
u U t C  
t = 9 );  
R a = P P t, P u, P x, c)) C  
a F  
}  
main ()  
R D (D (D (D (D (99)))) F
```

0 Ответить



НЛО прилетело и опубликовало эту надпись здесь

rsashka 15.03.2021 в 02:52

```
#define private: public:
```

А разве так прокатит???

0 Ответить



vassabi 15.03.2021 в 03:05

а в чем сомнения? препроцессор находит и применяет макросы до анализа кода – для него это просто такой текстовой файл. Он его открывает, читает, делает проход с макроподстановками и передает дальше.

А вы думаете – почему в языке есть

```
#pragma once
```

или

```
#ifndef __CODE_CPP  
#define __CODE_CPP  
#endif
```

?

+1 Ответить



rsashka 15.03.2021 в 11:46

Причем тут эти макросы?

Я про двоеточие

```
warning: ISO C++11 requires whitespace after the macro name  
error: expected primary-expression before 'public'  
note: in expansion of macro 'private'  
error: ISO C++ forbids declaration of 'type name' with no type [-fpermissive]
```

```
note: in expansion of macro 'private'

error: expected ';' at end of member declaration
note: in expansion of macro 'private'
```

◆ +1 Ответить

Ссылка

НЛО прилетело и опубликовало эту надпись здесь

◦  rsashka
15.03.2021 в 11:49

Я и сейчас раскрываю приватные объявления при сборке юнит тестов, чтобы не заморачиваться с наследованием классов.
Меня удивило предложение использовать двоеточия в макросах

◆ +1 Ответить

Ссылка

НЛО прилетело и опубликовало эту надпись здесь

◦  demp
15.03.2021 в 03:20

Вроде бы это нарушение one definition rule, с теоретически побочным эффектом undefined behaviour

◆ +1 Ответить

Ссылка

НЛО прилетело и опубликовало эту надпись здесь

◦  demp
15.03.2021 в 05:09

В hren.hpp определено

```
class Hren
{
private:
    int a;
};
```

В hren.cpp лежит реализация, всё ок

```
#include "hren.hpp"
```

...

В `hack_hren.cpp` хотим по какой-то причине иметь доступ ко внутренностям

```
#define private public
#include "hren.hpp"
...
```

Разные определения `Hren` в разных единицах трансляции -> ODR.

Еще в <https://stackoverflow.com/a/27779038/1355844> утверждается, что само по себе переопределение ключевых слов – это UB, если в такой единице трансляции включается заголовочный файл стандартной библиотеки.

На практике такого пока не наблюдается, но не хотелось бы получить проблемы на ровном месте, даже в теории.

+2 Ответить

...

НЛО прилетело и опубликовало эту надпись здесь

tuomitch
15.03.2021 в 22:08

Каждый `compilation unit` обрабатывается (препроцессором и затем компилятором) независимо от остальных.

0 Ответить

...

seregazh
15.03.2021 в 04:04

Соглашусь с популярным мнением что препроцессор это не часть языка, и если есть возможность избегать макросов то лучше это сделать.

«Колбаса» начинается когда макрос генерирует макросы которые разворачиваются в другие макроссы, будто это эффективный инструмент для кодогенерации.

Чисто субъективно: `IntelliSense` плачет, я плачу, люди которые читают код после меня тоже страдают – зачем это все? Чтобы «красиво» написать все в 2 строки, а не в 4 понятно?

0 Ответить

...

Siemargl
15.03.2021 в 15:50

В современном С++ можно спокойно обходиться без макросов вовсе, используя только шаблоны и `inline`-функции.

Как в С++ без макросов реализовать конкатенацию `##`?

Ну и `__FILE__` `__LINE__` незаменимы.

▶ Но в C++20 (

0 Ответить



○ Gryphon88
15.03.2021 в 17:39

Ну и __FILE__ __LINE__ незаменимы.

Не поделитесь примером? __LINE__ необходим в protothreads и удобен в реализации КА, но когда, кроме сборки и сообщения об ошибках, нужен __FILE__?

0 Ответить



НЛО прилетело и опубликовало эту надпись здесь

○ Gryphon88
15.03.2021 в 20:21

Сталкивался, решал уникальными строками сообщений об ошибках в xcase. Зря я так, видимо :)

0 Ответить



○ hiewpoint
15.03.2021 в 20:30

Слияние строк на C++ во время компиляции: [stackoverlow.com/questions/38955940/how-to-concatenate-static-strings-at-compile-time/62823211#62823211](https://stackoverflow.com/questions/38955940/how-to-concatenate-static-strings-at-compile-time/62823211#62823211)

Технически реализация стала возможна со стандарта C++11, в данном случае автор кода пользовался возможностями из C++17.

0 Ответить



○ Siemargl
15.03.2021 в 23:31

И как там по человечески будет аналог PRINT_VAL из статьи? (у меня получился сюр при попытке вызова)

```
#define PRINT_VAL(val) printf("Value of %s is %d", #val, val);
```

0 Ответить



○ hiewpoint
16.03.2021 в 00:28

Это уже другой вопрос. Я отвечал про конкатенацию двух или более строк через оператор препроцессора ##, а не про получение самого исходного текста переданного параметра через оператор #.

0 Ответить





Siemargl

16.03.2021 в 02:41

В исходном вопросе различия не было. С++ решение здесь очень ущербно.

Молчу уже про D =)



0

Ответить



...

cr0nk

15.03.2021 в 16:27

потому что теперь после MACRO() нужно будет ставить точку с запятой

Это основная и единственная причина так делать



0

Ответить



...



cr0nk

15.03.2021 в 16:33

```
#define true false
```

Удачной отладки :)



-2

Ответить



...



S1Fed

15.03.2021 в 20:30

Как-то проскочили мимо такой полезной вещи, как стандартные предопределенные макросы: __cplusplus – определяется как целочисленное литеральное значение, если компилируется как C++.

__DATE__ – дата компиляции текущего файла исходного кода.

__FILE__ – имя текущего файла исходного кода.

__LINE__ определяется как целочисленный номер строки в текущем файле исходного кода.

__STDC_NO_THREADS__ – определяется как 1, если реализация не поддерживает необязательные стандартные потоки.

__STDC_VERSION__ – определяется при компиляции в виде C, а также одного из вариантов /std:C11 или C17. Он разворачивается в 201112L для /std:c11 и в 201710L для /std:c17.

__STDCPP_THREADS__ определяется как 1, только если программа может иметь только один поток выполнения и скомпилирована как C++.

__TIME__ – время, в течение которого выполняется преобразование предварительно обработанной единицы трансляции.

В Visual Studio есть еще свои макросы, но это тема на целую статью.



0

Ответить



...

o a-tk

15.03.2021 в 21:45

Правило 0: при любой возможности отдавать предпочтение другим языковым средствам.

PS: у вас никогда std::min не отваливалась после умников оптимизаторов?

◆ +1 Ответить



...

o Tujh

15.03.2021 в 22:50

PS: у вас никогда std::min не отваливалась после умников оптимизаторов?

Чаще всего оно отваливается после, проблема стара как WinAPI

```
#include <windows.h>
```

◆ +3 Ответить



...

o cr0nk

16.03.2021 в 00:36



◆ -2 Ответить



...

o Tujh

16.03.2021 в 18:36

гуглить применение макроса NOMINMAX для WinAPI если ни когда не слышали.

Если в кратце, в недрах WinAPI, кажется в `windef.h` определены макросы `min/max`:

```
#ifndef NOMINMAX
#define min(x,y) ((x) < (y) ? (x) : (y))
#define max(x,y) ((x) > (y) ? (x) : (y))
#endif
```



+1

Ответить



...

o <?php tzlom 16.03.2021 в 03:06

В современном С++ можно спокойно обходиться без макросов вовсе

А вы знаете что `#pragma once` в стандарт не включен? И что иногда без него никак (поэтому все библиотеки используют дурацкий но незаменимый `include guard`)



0

Ответить



...

o Tujh 16.03.2021 в 18:39

думаю речь шла про самый последний стандарт, где вместо инклудов можно использовать модули, но это не точно.



0

Ответить



...

o Tuxman 30.03.2021 в 07:58

- (1) Ожидал увидеть – а сейчас тоже самое, но на безопасных конструкциях С++ `constexpr`, `inline`, `template`,...
- (2) Пост – реклама моего блога вне хабр платформы?



0

Ответить



...

Только полноправные пользователи могут оставлять комментарии. Войдите, пожалуйста.

ПОХОЖИЕ ПУБЛИКАЦИИ

21 апреля 2021 в 21:40

Произвольное число аргументов любых типов на C11 и выше с помощью `_Generic` и `variadic` макросов



+11



5K



34



37 +37

7 января 2013 в 21:32

Создаем расширение (extension) Visual Studio для генерирования C++ директивы #define в header-файле

+17

15K

62

10 +10

20 марта 2012 в 22:34

Пользовательские литералы в C++11

+92

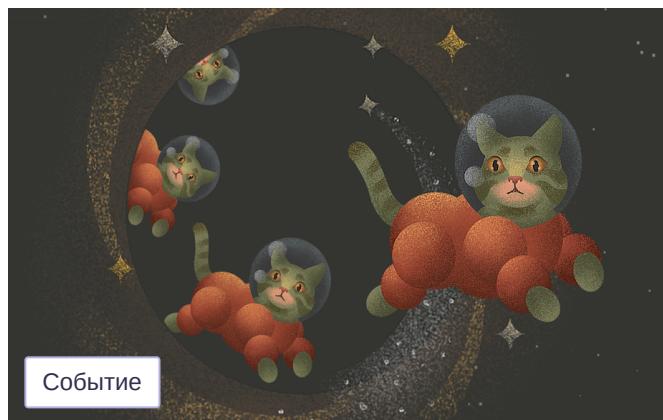
30K

231

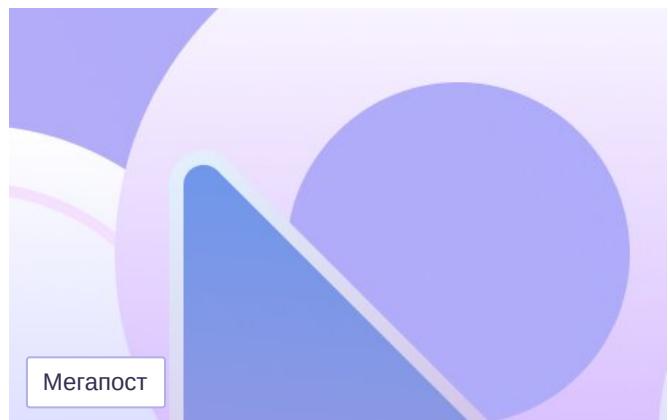
18

МИНУТОЧКУ ВНИМАНИЯ

Разместить



Конкурс технических статей Технотекст 2021



ИТ-рынок сегодня: наём, зарплаты, прогнозы

ЗАКАЗЫ

Разработка сайта для обработки PDF файлов

20000 руб./за проект • 4 просмотра

Консультация по созданию сайта в конструкторе

1500 руб./за проект • 15 просмотров

Парсинг характеристик для сайта на Битрикс

3000 руб./за проект • 1 отклик • 18 просмотров

Сбор контактов из 2gis

1500 руб./за проект • 19 просмотров

Сверстать сайт на html

12000 руб./за проект • 18 откликов • 68 просмотров

Больше заказов на Хабр Фрилансе

ЛУЧШИЕ ПУБЛИКАЦИИ ЗА СУТКИ

сегодня в 13:30

Как в декабре я устроился в \$βΣP*, а в марте меня уволили одним днем

+55

26K

26

247 +247

сегодня в 08:52

5 одноплатников, вышедших в марте 2022 года: необычные системы для самых разных задач

+32

6.3K

18

1 +1

сегодня в 16:00

Профессиональный обман: как мы рассылаем фишинговые письма нашим клиентам

+28

2.8K

13

3 +3

вчера в 22:28

Пожалуйста, прекратите писать shell-скрипты

+28

19K

111

111 +111

сегодня в 15:00

Этапы погружения junior-разработчика

+16

2.2K

17

3 +3

Четыре слагаемых успешной системы образования

Интересно

Реклама



- найти, где в компании хранится критичная информация
- провести аудит прав доступа
- запретить действия с файлами тем, кому это не положено

[Ваш аккаунт](#)[Разделы](#)[Информация](#)[Услуги](#)[Войти](#)[Публикации](#)[Устройство сайта](#)[Реклама](#)[Регистрация](#)[Новости](#)[Для авторов](#)[Тарифы](#)[Хабы](#)[Для компаний](#)[Контент](#)[Компании](#)[Документы](#)[Семинары](#)[Авторы](#)[Соглашение](#)[Мегапроекты](#)[Песочница](#)[Конфиденциальность](#)[Настройка языка](#)[0 сайте](#)[Техническая поддержка](#)

[Вернуться на старую версию](#)

© 2006–2022 «Habr»

ЧИТАЮТ СЕЙЧАС

Как в декабре я устроился в \$БΣР*, а в марте меня уволили одним днем

26K 247 +247

Выходцы из «Яндекса» представили приложение Now, заменяющее собой Instagram**

32K 94 +94

Минцифры пояснило порядок получения отсрочки от армии для IT-специалистов

10K 50 +50

Учёные случайно совершили прорыв в развитии химического оружия

11K 22 +22

Самый длинный в мире авиамаршрут из Гонконга в Нью-Йорк проложат в обход РФ

7.2K 34 +34

Open Source vs «коробки»: что думают хабровчане

Мегапост

РАБОТА

QT разработчик
14 вакансий

Программист С
39 вакансий

Программист C++
114 вакансий

Реклама



Реклама