Swift's blog

C++ Tricks

By Swift, 5 years ago,

I see lots of programmers write code like this one:

```
pair<int, int> p;
vector<int> v;
// ...
p = make_pair(3, 4);
v.push_back(4); v.push_back(5);
```

while you can just do this:

```
pair<int, int> p;
vector<int> v;
// ...
p = {3, 4};
v = {4, 5};
```

1. Assign value by a pair of {} to a container

I see lots of programmers write code like this one:

```
pair<int, int> p;
// ...
p = make_pair(3, 4);
```

while you can just do this:

```
pair<int, int> p;
// ...
p = {3, 4};
```

even a more complex `pair`

```
pair<int, pair<char, long long> > p;
// ...
p = {3, {'a', 8ll}};
```

What about `vector`, `deque`, `set` and other containers?

```
vector<int> v;
v = {1, 2, 5, 2};
for (auto i: v)
    cout << i << ' ';
cout << '\n';
// prints "1 2 5 2"


deque<vector<pair<int, int>>> d;
d = {{{3, 4}, {5, 6}}, {{1, 2}, {3, 4}}};
for (auto i: d) {
    for (auto j: i)
        cout << j.first << ' ' << j.second << '\n';
    cout << "-\n";
}
```

→ Pay attention

Before contest
Kotlin Heroes: Practice 3
2 days

Like 25 people like this. Sign Up to see what your friends like.

→ Top rated

#	User	Rating
1	MiFaFaOvO	3681
2	tourist	3404
3	TLE	3356
4	apiadu	3351
5	mnbvmar	3281
6	LHiC	3276
7	Um_nik	3268
8	300iq	3267
9	yosupo	3249
10	ainta	3226

Countries | Cities | Organizations View all →

→ Top contributors

#	User	Contrib.
1	Errichto	190
2	antontrygubO_o	186
3	tourist	182
4	Radewoosh	169
5	vovuh	167
5	pikmike	167
7	ko_osaga	161
8	Um_nik	160
9	Petr	155
10	McDic	152

View all →

→ Find user

Handle:

Find

→ Recent actions

coding.quest1 → Atcoder Educational DP Contest :- Video Tutorials (till problem C)

FieryPhoenix → Codeforces Round #621 (Div. 1 + Div. 2) Editorial

Halit → Is Object-Oriented Programming Useful in Competitive Programming?

JuanFernandez → [Timus 1522] Help to prove solution.

chokudai → AtCoder Beginner Contest 155 Announcement

MiFaFaOvO → Screencast collection

Laggy → Teams going to ICPC World Finals 2020 — WIP List

djm03178 → Codeforces Round #620 (Div. 2) Editorial

```
// prints "3 4
//         5 6
//         -
//         1 2
//         3 4
//         -"


set<int> s;
s = {4, 6, 2, 7, 4};
for (auto i: s)
    cout << i << ' ';
cout << '\n';
// prints "2 4 6 7"


list<int> l;
l = {5, 6, 9, 1};
for (auto i: l)
    cout << i << ' ';
cout << '\n';
// prints "5 6 9 1"


array<int, 4> a;
a = {5, 8, 9, 2};
for (auto i: a)
    cout << i << ' ';
cout << '\n';
// prints "5 8 9 2"


tuple<int, int, char> t;
t = {3, 4, 'f'};
cout << get<2>(t) << '\n';
```

Note that it doesn't work for `stack` and `queue`.

2. Name of argument in macros

You can use '#' sign to get exact name of an argument passed to a macro:

```
#define what_is(x) cerr << #x << " is " << x << endl;
// ...
int a_variable = 376;
what_is(a_variable);
// prints "a_variable is 376"
what_is(a_variable * 2 + 1)
// prints "a_variable * 2 + 1 is 753"
```

3. Get rid of those includes!

Simply use

```
#include <bits/stdc++.h>
```

This library includes many of libraries we do need in contest like `algorithm`, `iostream`, `vector` and many more. Believe me you don't need to include anything else!

4. Hidden function (not really hidden but not used often)

one)

```
__gcd(value1, value2)
```

You don't need to code Euclidean Algorithm for a gcd function, from now on we can use. This function returns gcd of two numbers.

e.g. __gcd(18, 27) = 9.

two)

ITryMyBest → Atcoder Beginner Contest 043 [D]

ben_dover → Recent Project Euler Fastest Solvers (Eulerians)

FieryPhoenix → Codeforces Round #621 (Div. 1 + Div. 2)

Medeowex → Codeforces Round #619 (Div. 2) Editorial

tourist → XX Open Cup: GP of Gomel

Geralt → Codeforces round #621 Winner Stream

CodingKnight → A serious question about the security of online compilers

support → moshennika-A rising cheater.

SPyofgame → Can we optimize the space Counting Sort using ?

pikmike → Educational Codeforces Round 82 [Rated for Div. 2]

mshiladityam → Please check if this question requires Graph Theory, DFs Or some advanced concept

djm03178 → Codeforces Round #620 (Div. 2)

beet → Automating the verification of your library — Online Judge Verify Helper

Araska → Maximum Matching

Una_Shem → Hash Code 2020

Bhj2001 → Invitation to ICM Technex 2020 — IIT(BHU) Varanasi

repeating → Maximum rating change in CF history

Detailed →

`__builtin_ffs(x)`

This function returns 1 + least significant 1-bit of x. If x == 0, returns 0. Here x is `int` , this function with suffix 'l' gets a `long` argument and with suffix 'll' gets a `long long` argument.

e.g. __builtin_ffs(10) = 2 because 10 is '...10 **1** 0' in base 2 and first 1-bit from right is at index 1 (0-based) and function returns 1 + index.

three)

`__builtin_clz(x)`

This function returns number of leading 0-bits of x which starts from most significant bit position. x is `unsigned int` and like previous function this function with suffix 'l gets a `unsigned long` argument and with suffix 'll' gets a `unsigned long long` argument. If x == 0, returns an undefined value.

e.g. __builtin_clz(16) = 27 because 16 is ' **...** 10000'. Number of bits in a `unsigned int` is 32. so function returns 32 — 5 = 27.

four)

`__builtin_ctz(x)`

This function returns number of trailing 0-bits of x which starts from least significant bit position. x is `unsigned int` and like previous function this function with suffix 'l' gets a `unsigned long` argument and with suffix 'll' gets a `unsigned long long` argument. If x == 0, returns an undefined value.

e.g. __builtin_ctz(16) = 4 because 16 is '...1 **0000** '. Number of trailing 0-bits is 4.

five)

`__builtin_popcount(x)`

This function returns number of 1-bits of x. x is `unsigned int` and like previous function this function with suffix 'l' gets a `unsigned long` argument and with suffix 'll' gets a `unsigned long long` argument. If x == 0, returns an undefined value.

e.g. __builtin_popcount(14) = 3 because 14 is '... **111** 0' and has three 1-bits.

Note: There are other `__builtin` functions too, but they are not as useful as these ones.

Note: Other functions are not unknown to bring them here but if you are interested to work with them, I suggest this website.

5. Variadic Functions and Macros

We can have a variadic function. I want to write a sum function which gets a number of ints, and returns sum of them. Look at the code below:

```
int sum() { return 0; }

template<typename... Args>
int sum(int a, Args... args) { return a + sum(args...); }

int main() { cout << sum(5, 7, 2, 2) + sum(3, 4); /* prints "23" */ }
```

In the code above I used a template. sum(5, 7, 2, 2) becomes 5 + sum(7, 2, 2) then sum(7, 2, 2), itself, becomes 7 + sum(2, 2) and so on... I also declare another sum function which gets 0 arguments and returns 0.

I can even define a any-type sum function:

```
int sum() { return 0; }

template<typename T, typename... Args>
T sum(T a, Args... args) { return a + sum(args...); }

int main() { cout << sum(5, 7, 2, 2) + sum(3.14, 4.89); /* prints "24.03" */ }
```

Here, I just changed `int` to `T` and added `typename T` to my template.

In C++14 you can also use `auto sum(T a, Args... args)` in order to get sum of mixed types. (Thanks to **slycelote** and **Corei13**)

We can also use variadic macros:

```
#define a_macro(args...) sum(args)

int sum() { return 0; }

template<typename T, typename... Args>
auto sum(T a, Args... args) { return a + sum(args...); }

int main() { cout << a_macro(5, 7, 2, 2) + a_macro(3.14, 4.89); /* prints "24.03" */ }
```

Using these 2, we can have a great debugging function: (thanks to **Igorjan94**) — *Updated!*

```
#include <bits/stdc++.h>

using namespace std;

#define error(args...) { string _s = #args; replace(_s.begin(), _s.end(), ',', ' ');
stringstream _ss(_s); istream_iterator<string> _it(_ss); err(_it, args); }

void err(istream_iterator<string> it) {}
template<typename T, typename... Args>
void err(istream_iterator<string> it, T a, Args... args) {
        cerr << *it << " = " << a << endl;
        err(++it, args...);
}

int main() {
        int a = 4, b = 8, c = 9;
        error(a, b, c);
}
```

Output:

```
a = 4
b = 8
c = 9
```

This function helps a lot in debugging.

6. Here is C++0x in CF, why still C++?

Variadic functions also belong to C++11 or C++0x, In this section I want to show you some great features of C++11.

one) Range-based For-loop

Here is a piece of an old code:

```
set<int> s = {8, 2, 3, 1};
for (set<int>::iterator it = s.begin(); it != s.end(); ++it)
    cout << *it << ' ';
// prints "1 2 3 8"
```

Trust me, that's a lot of code for that, just use this:

```
set<int> s = {8, 2, 3, 1};
for (auto it: s)
    cout << it << ' ';
// prints "1 2 3 8"
```

We can also change the values just change `auto` with `auto &` :

```
vector<int> v = {8, 2, 3, 1};
for (auto &it: v)
    it *= 2;
for (auto it: v)
    cout << it << ' ';
// prints "16 4 6 2"
```

two) The Power of `auto`

You don't need to name the type you want to use, C++11 can infer it for you. If you need to loop over iterators of a set<pair<int, pair<int, int> > > from begin to end, you need to type `set<pair<int, pair<int, int> > >::iterator` for me it's so suffering! just use auto it = s.begin()

also x.begin() and x.end() now are accessible using begin(x) and end(x).

There are more things. I think I said useful features. Maybe I add somethings else to post. If you know anything useful please share with Codeforces community :)

From **Ximera**'s comment:

this code:

```
for(i = 1; i <= n; i++) {
    for(j = 1; j <= m; j++)
        cout << a[i][j] << " ";
    cout << "\n";
}
```

is equivalent to this:

```
for(i = 1; i <= n; i++)
    for(j = 1; j <= m; j++)
        cout << a[i][j] << " \n"[j == m];
```

And here is the reason: `" \n"` is a `char*`, `" \n"[0]` is `' '` and `" \n"[1]` is `'\n'`.

From **technetium28**'s comment:

Usage of `tie` and `emplace_back`:

```
#define mt make_tuple
#define eb emplace_back
typedef tuple<int,int,int> State; // operator< defined

int main(){
  int a,b,c;
  tie(a,b,c) = mt(1,2,3); // assign
  tie(a,b) = mt(b,a); // swap(a,b)

  vector<pair<int,int>> v;
  v.eb(a,b); // shorter and faster than pb(mp(a,b))

  // Dijkstra
  priority_queue<State> q;
  q.emplace(0,src,-1);
  while(q.size()){
    int dist, node, prev;
    tie(dist, ode, prev) = q.top(); q.pop();
    dist = -dist;
    // ~~ find next state ~~
    q.emplace(-new_dist, new_node, node);
  }
}
```

And that's why `emplace_back` faster: `emplace_back` is faster than `push_back` 'cause it just construct value at the end of vector but `push_back` construct it somewhere else and then move it to the vector.

Also in the code above you can see how `tie(args...)` works. You can also use `ignore` keyword in `tie` to ignore a value:

```
tuple<int, int, int, char> t (3, 4, 5, 'g');
int a, b;
tie(b, ignore, a, ignore) = t;
cout << a << ' ' << b << '\n';
```

Output: `5 3`

I use this macro and I love it:

```
#define rep(i, begin, end) for (__typeof(end) i = (begin) - ((begin) > (end)); i !=
(end) - ((begin) > (end)); i += 1 - 2 * ((begin) > (end)))
```

First of all, you don't need to name the type you want to use. Second of all it goes forwards and backwards based on (begin > end) condition. e.g. `rep(i, 1, 10)` is 1, 2, ..., 8, 9 and `rep(i, 10, 1)` is 9, 8, ..., 2, 1

It works well with different types e.g.

```
vector<int> v = {4, 5, 6, 4, 8};
rep(it, end(v), begin(v))
    cout << *it << ' ';
// prints "8 4 6 5 4"
```

Also there is another great feature of C++11, lambda functions!

Lambdas are like other languages' closure. It defines like this:

```
[capture list](parameters) -> return value { body }
```

one) Capture List: simple! We don't need it here, so just put `[]`

two) parameters: simple! e.g. int x, string s

three) return value: simple again! e.g. pair<int, int> which can be omitted most of the times (thanks to **Jacob**)

four) body: contains function bodies, and returns return value.

e.g.

```
auto f = [] (int a, int b) -> int { return a + b; };
cout << f(1, 2); // prints "3"
```

You can use lambdas in `for_each`, `sort` and many more STL functions:

```
vector<int> v = {3, 1, 2, 1, 8};
sort(begin(v), end(v), [] (int a, int b) { return a > b; });
for (auto i: v) cout << i << ' ';
```

Output:

```
8 3 2 1 1
```

From **Igorjan94**'s comment:

Usage of `move`:

When you work with STL containers like `vector`, you can use `move` function to just move container, not to copy it all.

```
vector<int> v = {1, 2, 3, 4};
vector<int> w = move(v);

cout << "v: ";
for (auto i: v)
    cout << i << ' ';

cout << "\nw: ";
for (auto i: w)
    cout << i << ' ';
```

Output:

```
v:
w: 1 2 3 4
```

As you can see `v` moved to `w` and not copied.

7. C++0x Strings

one) Raw Strings (From **IvayloS**'s comment)

You can have UTF-8 strings, Raw strings and more. Here I want to show raw strings. We define a raw string as below:

```
string s = R"(Hello, World!)"; // Stored: "Hello, World!"
```

A raw string skips all escape characters like `\n` or `\"` . e.g.

```
string str = "Hello\tWorld\n";
string r_str = R"(Hello\tWorld\n)";
cout << str << r_str;
```

Output:

```
Hello   World
Hello\tWorld\n
```

You can also have multiple line raw string:

```
string r_str =
R"(Dear Programmers,
I'm using C++11
Regards, Swift!)";
cout << r_str;
```

Output:

```
Dear Programmer,
I'm using C++11
Regards, Swift!
```

two) Regular Expressions (regex)

Regular expressions are useful tools in programming, we can define a regular expression by `regex` e.g. `regex r = "[a-z]+";` . We will use raw string for them because sometimes they have `\` and other characters. Look at the example:

```
regex email_pattern(R"(^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$)"); // This
email pattern is not totally correct! It's correct for most emails.

string
valid_email("swift@codeforces.com"),
invalid_email("hello world");

if (regex_match(valid_email, email_pattern))
    cout << valid_email << " is valid\n";
else
    cout << valid_email << " is invalid\n";

if (regex_match(invalid_email, email_pattern))
    cout << invalid_email << " is valid\n";
else
    cout << invalid_email << " is invalid\n";
```

Output:

```
swift@codeforces.com is valid
hello world is invalid
```

Note: You can learn Regex in this website.

three) User-defined literals

You already know literals from C++ like: `0xA` , `1000ll` , `3.14f` and so on...

Now you can have your own custom literals! Sounds great :) So let's see an example:

```
long long operator "" _m(unsigned long long literal) {
        return literal;
}

long double operator "" _cm(unsigned long long literal) {
        return literal / 100.0;
}

long long operator "" _km(unsigned long long literal) {
        return literal * 1000;
}
```

```
int main() {
        // See results in meter:
        cout << 250_m << " meters \n"; // Prints 250 meters
        cout << 12_km << " meters \n"; // Prints 12000 meters
        cout << 421_cm << " meters \n"; // Prints 4.21 meters
}
```

Note that a literal should start with an underscore (`_`). We declare a new literal by this pattern:

```
[returnType] operator "" _[name]([parameters]) { [body] }
```

note that parameters only can be one of these:

`(const char *)`

`(unsigned long long int)`

`(long double)`

`(char)`

`(wchar_t)`

`(char16_t)`

`(char32_t)`

`(const char *, size_t)`

`(const wchar_t *, size_t)`

`(const char16_t *, size_t)`

`(const char32_t *, size_t)`

Literals also can used with templates.

Discussion of Delete2

c++, c++0x, tricks

+971 Swift 5 years ago 196

Comments (196)

Write comment?

mm_kishor

5 years ago, # | +7

Its awesome. Thanks Swift :)

→ Reply

slycelote

5 years ago, # | +7

Your `sum` function returns an incorrect result for `sum(1, 1.5)` . To fix, declare the return type as `auto` .

→ Reply

Swift

5 years ago, # ^ | ← Rev. 2 +5

My sum function designed to sum numbers from one type. I mean integers, doubles, ... not mix of these types. BTW, How should I use auto in that function?

I mean you can't have a `auto` return type for any function as far as I know.

→ Reply

slycelote

5 years ago, # ^ | +6

http://ideone.com/6l4Wc7

→ Reply

5 years ago, # ^ | +5

Interesting! my Xcode can't compile that code. I'll edit blog post.

Swift Thank you.
→ Reply

4 years ago, # ^ | 0

Ahmadshallouf

my xcode doesn't allow __gcd or any function that start with __builtin. and doesn't allow bits/stdc++.h

how do you do it in your Xcode ?

and thanks for the entry.
→ Reply

3 years ago, # ^ | 0

_Noor

Download this file :https://gist.github.com/reza-ryte-club/97c39f35dab0c45a5d924dd9e50c445f and then include stdc++.h to your code, before submitting your code change it to bits/stdc++.h .
→ Reply

5 years ago, # ^ | +5

EeOneGuy

Why not? http://pastie.org/9817864
→ Reply

5 years ago, # ^ | ← Rev. 2 +5

Swift

Your code has `decltype` (actually because of `->`). Xcode won't compile code without it. However IDEONE compiles it. So I edited my post.
→ Reply

5 years ago, # ^ | +5

nic11

Isn't `decltype` C++14?
→ Reply

5 years ago, # ^ | +5

Swift

I suppose not.
→ Reply

3 months ago, # ^ | 0

interestingLSY

It's supported since c++14. :)
→ Reply

5 years ago, # | ← Rev. 2 +5

determinism

It's better to use auto& in range-based loop when the object is not primitive (e.g pair, vector). UPD: I realized that you mention it at the end, but there are some code written poorly because of that in the first part.
→ Reply

4 years ago, # ^ | 0

samiemad

actually, compiler optimizations will get rid of the extra copy operations if you are not modifying the element. so I don't think it will be any slower in runtime compared to auto&.

You can use auto& if you are too suspicious, but I don't think that the first part is categorized as 'written poorly'. it is just OK.
→ Reply

3 years ago, # ^ | 0

retrograd

`const auto&` is even better if you want to be really strict about it.
→ Reply

5 years ago, # | +18

Swistakk

"these things are belong to C++11" — https://www.youtube.com/watch?v=8fvTxv46ano :)

→ Reply

Swift
5 years ago, # ^ | 0

LMAO =))

→ Reply

GiveMinus
5 years ago, # | -72
→ Reply

The comment is hidden because of too negative feedback, click here to view it

Swistakk
5 years ago, # | +4

mukel already has written nice "C++11 for dummies" tutorial http://codeforces.com/blog/entry/10124 . I think it's a good idea to provide that link directly in entry.

→ Reply

Swift
5 years ago, # ^ | 0

Excellent tutorial, I'll add it at top of blog.

→ Reply

IWillBeRed
5 years ago, # | +10

Could you give link to compiler that you use? Because I get CE on my GNU 4.7.1:)

→ Reply

Swift
5 years ago, # ^ | ← Rev. 2 +5

In CF, use `GNU C++0x 4` instead of `GNU C++ 4.7` .

Get latest GCC, and from your terminal/cmd use one of these flags `-std=gnu++11` or `-std=c++11` You can download it for your computer: Windows —

→ Reply

shashanktandon
5 years ago, # | 0

Thanks for such a nice explanation...

→ Reply

fushar
5 years ago, # | +5

Anyone knows how to include <bits/stdc++.h> on OS X? I am already using gcc but it cannot found that header...

→ Reply

Swift
5 years ago, # ^ | 0

1. Go to: `/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/include/c++/v1`
2. Create a folder named `bits`
3. Add a file into that named `stdc++.h`
4. Edit it and include libraries

→ Reply

J4T8Z9
5 years ago, # ^ | 0

yeah, that works, I did the same :)

→ Reply

fushar
5 years ago, # ^ | 0

What is the content of the file (stdc++.h)?

→ Reply

Swift
5 years ago, # ^ | 0

Here: https://gist.github.com/eduarc/6022859

→ Reply

5 years ago, # ^ | 0

Ah, forgot to say. Thank you! It worked :)

fushar

Ah, forgot to say. Thank you! it worked :)
→ Reply

josemanuel101

5 years ago, # ^ | ← Rev. 2 0

Thanks for sharing! Works like a breeze. For those who don't have Xcode, but have the command line developer tools installed, go to: `/Library/Developer/CommandLineTools/usr/include/c++/v1` in step one.
→ Reply

MrNull

4 years ago, # ^ | 0

there is another way: install GCC using brew terminal package manager!
→ Reply

Corei13

5 years ago, # | +4

The second sum function (with `auto`) is `C++14` standard, not `C++11`. `C++11` doesn't allow function without a return type.
→ Reply

Swift

5 years ago, # ^ | 0

Thanks for sharing your knowledge to us! That's why Xcode couldn't compile that. Now I tested it with C++14 and everything is OK. So let's make it clear in blog.
→ Reply

Corei13

5 years ago, # ^ | +32

And it is still possible to write sum (or other) functions for mixed type using `std::common_type`

```
template <typename A, typename B>
auto sum(A a, B b) -> typename common_type<A, B>::type {
    return static_cast<typename common_type<A, B>::type>(a) +
static_cast<typename common_type<A, B>::type>(b);
}

template <typename A, typename B, typename... Args>
auto sum(A a, B b, Args... args) -> typename common_type <A, B,
Args...>::type {
    return sum(sum(a, b), args...);
}

int main() {
    cout << sum(5, 7, 2, 2) + sum(3.14, 4.89) << endl;      //
24.03
    cout << sum (complex <double>(1, 2), 1.3, 2) << endl;   //
(4.3,2)
}
```
→ Reply

Swift

5 years ago, # ^ | +65

→ Reply

Baklazan

5 years ago, # | +3

As for `__gcd()`, it may be a little tricky at some compilers.
→ Reply

5 years ago, # | ← Rev. 2 +30

The best thing is that you can write like this (C++11 vs C++) :D

The best thing is that you can write like this (C++11 vs C++) :D

Na2a

```
vector<pair<int, int>> v;
```

instead of this

```
vector<pair<int, int> > v;
```

→ Reply

GiveMinus

5 years ago, # ^ | -54

→ Reply

The comment is hidden because of too negative feedback, click here to view it

Xellos

5 years ago, # ^ | +27

→ Reply

Swift

5 years ago, # ^ | 0

If C++ is that bad, why all of your codes are in this language?

→ Reply

GiveMinus

5 years ago, # ^ | 0

give a kiss baby :)

→ Reply

Swift

5 years ago, # ^ | +65

Here you are:

→ Reply

GiveMinus
5 years ago, # ^ | +1

tanx

→ Reply

Batman
5 years ago, # ^ | +9

Cause he don't do them...

(cheat)

→ Reply

Swift
5 years ago, # ^ | 0

Yep. I also do this in my post: `deque<vector<pair<int, int>>> d;`

→ Reply

Ximera
5 years ago, # | ← Rev. 2 +31

May be you can tell something more about this

```
for(i = 1; i <= n; i++) {
    for(j = 1; j <= m; j++)
        cout << a[i][j] << " ";
    cout << "\n";
}

for(i = 1; i <= n; i++)
    for(j = 1; j <= m; j++)
        cout << a[i][j] << " \n"[j == m];
```

→ Reply

Swift
5 years ago, # ^ | ← Rev. 3 +32

Well, Great creativity :)

`" \n"` is a char*, " \n"[0] is ' ' and " \n"[1] is '\n'.

Also this is a correct one too:

```
for (int i = 1; i <= n; i++)
                for (int j = 1; j <= m; j++)
                        cout << a[i][j] << (j == m)[" \n"];
```

It's because e.g. a[8] and 8[a] are the same thing both of them are (a + 8)* and (8 + a)*.

→ Reply

GiveMinus
5 years ago, # ^ | -13

no

→ Reply

Ximera
5 years ago, # ^ | 0

Actually `" \n"[j == m]` was correct, but that doesn't matter at all now :)

→ Reply

Swift
5 years ago, # ^ | 0

Oops! You're right!

→ Reply

Nizil
4 months ago, # ^ | 0

They aren't exactly equivalent to the original because in the original there is one extra space at the end of each line. I still like the idea.

→ Reply

__builtin__wolfy
5 years ago, # ^ | +1

For a while, I thought that this is Iverson's bracket :D

→ Reply

5 years ago, # | ← Rev. 2 +14

Do you know tie and emplace ?

tubo28

Do you know tie and emplace ?

```
#define mt make_tuple
#define eb emplace_back
typedef tuple<int,int,int> State; // operator< defined

int main(){
  int a,b,c;
  tie(a,b,c) = mt(1,2,3); // assign
  tie(a,b) = mt(b,a); // swap(a,b)

  vector<pair<int,int>> v;
  v.eb(a,b); // shorter and faster than pb(mp(a,b))

  // Dijkstra
  priority_queue<State> q;
  q.emplace(0,src,-1);
  while(q.size()){
    int dist, node, prev;
    tie(dist, ode, prev) = q.top(); q.pop();
    dist = -dist;
    // ~~ find next state ~~
    q.emplace(-new_dist, new_node, node);
  }
}
```

→ Reply

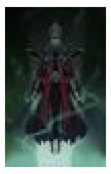
5 years ago, # ^ | 0

Swift

Such a great feature.

`emplace_back` is faster than `push_back` 'cause it just construct value at the end of vector but `push_back` construct it somewhere else and then move it to the vector.

→ Reply


HekpoMaH

5 years ago, # | 0

Can you get the previous element in an, let's say, vector using `auto` ? Here is why `auto` is not the best option for dp-like tasks where you need information from the previous elements.

→ Reply

5 years ago, # ^ | ← Rev. 3 +4

Swift

Use this approach:

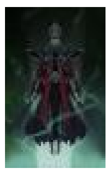
```
vector<int> dp = {4, 5, 6, 4, 8};
for (auto i = ++dp.begin(); i != dp.end(); ++i)
    *i += *(i - 1);
for (auto i: dp)
    cout << i << '\n';
```

Output:

```
4
9
15
19
27
```

Use range-based for-loop only when you want exact element, when you need to access other elements use normal for-loop, but this doesn't mean that you can't use auto in that for-loop.

→ Reply


HekpoMaH

5 years ago, # ^ | 0

Hm, I didn't know it could be done. Still, it is easier with normal for loop.

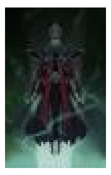
→ Reply

5 years ago, # ^ | ← Rev. 3 +3

Swift

Btw, using `auto` is just for inferring type you are working with. If your type is `int` , it's better to use that ('cause it's just 3 characters) but if your type is `std::vector<std::pair<std::set<int>, bool>>::iterator` so I think using `auto` is a must :)

→ Reply


HekpoMaH

5 years ago, # ^ | 0

XD yeah I agree about this one.

→ Reply

Rezwan.Arefin01

3 years ago, # ^ | 0

Just saying. Cumulative sum can be done only with this-

```
vector<int> dp = {4, 5, 6, 4, 8};
partial_sum(dp.begin(), dp.end(), dp.begin());
```

→ Reply

rlac

5 years ago, # | +13

In 2, I use:

```
#define DB(x) cerr << __LINE__ << ": " << #x << " = " << (x) << endl
```

In this way I get the number of the line in which this instruction is executed. It's useful when we have more than one variable with the same name. Also, x needs to be enclosed in parenthesis due to operators precedence.

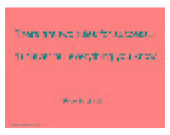
→ Reply

aremo

5 years ago, # | 0

would you please tell me about vector ,i don't know anything about that !

→ Reply


yarak

5 years ago, # ^ | ← Rev. 2 0

vector

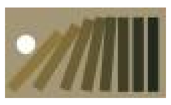
→ Reply

yzmyyff

5 years ago, # | 0

Its useful! Thanks for sharing.

→ Reply


determinism

5 years ago, # | ← Rev. 2 +6

You say that "Variadic functions also belong to C++11", but that's not really correct. Even C had variadic functions. New feature in C++11 is variadic templates.

→ Reply

Swift

5 years ago, # ^ | +3

Yeah. You're right. Here I used variadic template so I said it's for C++11.

→ Reply

Baklazan

5 years ago, # | +1

I thing you should consider defining short version of your blog post, now that it is on the main page.

→ Reply

Swift

5 years ago, # ^ | 0

OK. I'll do it.

→ Reply

kien_coi_1997

5 years ago, # | +27

In my country, at this time, we are not allowed to use C++11 in national contest.

→ Reply

I_love_Hoang_Yen

5 years ago, # ^ | 0

Is C++11 being used in IOI? If this is the case, I guess it should not be hard to convince the judge committee to change.

→ Reply

5 years ago, # | 0

if i have a vector < pair<int, pair<int, int> > > a;

could i use emplace_back to insert {1, {2, 3}}? i tries to emplace_back(1, 2, 3); but of course it's an could I use emplace_back to insert {1, {2, 3} }? I tries to emplace_back(1, 2, 3), but of course it's an error.

thanks in advance :-)

Tensor

→ Reply

riadwaw

5 years ago, # ^ | 0

You could emplace_back(^{1}, mp(2,3))

→ Reply

Tensor

5 years ago, # ^ | 0

thank you for replying. i was looking forward for a method like that above something like (1, 2, 3); as i don't like using macros, something that's faster to write.

thanks in advance :)

→ Reply

Swift

5 years ago, # ^ | 0

Don't use `pair<int, pair<int, int>>` ! Code less and use `tuple<int, int, int>` :

```
vector<tuple<int, int, int>> v;
v.emplace_back(1, 2, 3);
```

→ Reply

Baklazan

5 years ago, # ^ | +3

Well, actually sometimes `pair<int, pair<int,int> > x;` may make more sense than `tuple<int,int,int> x;` , for instance when `x.second` are coordinates of some point and `x.first` is some property of this point.

→ Reply

Swift

5 years ago, # ^ | ← Rev. 2 +10

When working with tuples, you don't really use get(tuple) you do use tie:

```
tie(point_property, pointx, pointy) = some_tuple;
```

And that makes sense.

→ Reply

riadwaw

5 years ago, # ^ | 0

then you probably have that point as a variable, not as two coordinates.

→ Reply

Baklazan

5 years ago, # ^ | 0

I often use

```
#define X first
#define Y second
#define pii pair<int, int>

pii point;
```

→ Reply

Rubanenko

5 years ago, # ^ | +25

Yeah let's write ugly unreadable code with nested pairs and macros instead of class/struct.

→ Reply

Baklazan

5 years ago, # ^ | +8

I totally agree that classes/structs are more readable. I just wanted to point out that in some cases `tuple<int,int,int>` is less readable (at least for me) than `pair<int, pair<int,int> >` .

→ Reply

4 years ago, # ^ | 0

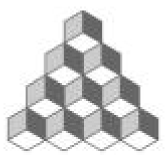
The real solution to this would be something that lets us write

EvgeniSergeev

The real solution to this would be something that lets us write

```
struct dist_xy {
    const int dist, x, y;
};
```

and then would supply a commonsense `bool operator<(..)` automatically.
→ Reply


AkshajK

5 years ago, # | 0

Thanks for this! I'm sure many of us would also be interested in a Java tricks article! :)
→ Reply

Rubanenko

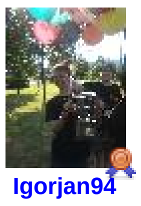
5 years ago, # ^ | +38

The advantage of Java is that there are no tricks.
→ Reply

Swift

5 years ago, # ^ | ← Rev. 2 0

I can also write an article about **Swift**'s tricks. But no one here, cares about that language :)
→ Reply


Igorjan94

5 years ago, # | ← Rev. 2 +3

your debugging function doesn't work for `#args` with spaces
so, I think it's better to rewrite split to more universal

```
vector<string> split(const string& s, char c) {
    vector<string> v;
    stringstream ss(s);
    string x;
    while (getline(ss, x, c))
        v.eb(x); //emplace_back
    return std::move(v);
}
```

(Note no copying because of move, another cpp trick)
and macro will be:

```
#define err(args...) {\
    vector<string> _v = split(#args, ',');\
    err(_v.begin(), args);\
}
```

→ Reply

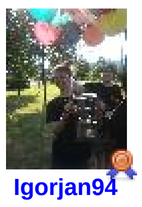
Swift

5 years ago, # ^ | 0

It also brings default space before arguments, e.g. `err(a, b)` outputs:

```
a = value1
 b = value2
```

but it's better for arguments like `a + b` so I'll replace it with my code.
→ Reply

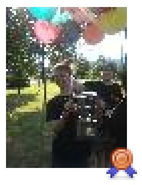

Igorjan94

5 years ago, # ^ | ← Rev. 3 0

oh, yep, I forgot I changed your err to

```
void err(vector<string>::iterator it) {}
template<typename T, typename... Args>
void err(vector<string>::iterator it, T a, Args... args) {
        cerr << it->substr((*it)[0] == ' ') << " = " << a <<
'\n';
        err(++it, args...);
}
```

→ Reply



5 years ago, # ^ | 0

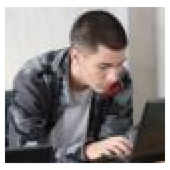
if you are interested in it, I also have writeln and readln on variadic templates, which helps to write smth like this:

```
int n; vector<pair<int, pair<int, long long>>> a; long long l;
```

Igorjan94

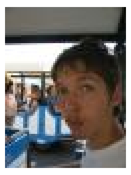
```
int n; vector<pair<int, pair<int, long long>>> a; long long l;
char c; string s; double d; // just any combination of
fundamental types + vector/pair
readln(n, a, l, c, s, d);
writeln(n, a, l, c, s, d);
```

you can find it here 9388829(I deleted all spaces for more compact view)
if trailing space is unimportant, half of code can be deleted:)
it can be simply extended on user's types by overloading ostream and istream operators
this template is with cin/cout, and this->9316393 with scanf/printf
yes, looks awful, and for only prewritten use:)
→ Reply


dj3500

5 years ago, # ^ | +6

Actually this use of `std::move` is superfluous. The compiler will move the return value automatically (search for: return value optimization).
→ Reply


Jacob

5 years ago, # | ← Rev. 3 +1

One can omit return type in lambda expression in most cases.

P.S. I have to say, 'tie' looks awesome, I need to start using it.
→ Reply

Kvark161

5 years ago, # | +4

You haven't to specify return type in lambda functions if all return values are the same type.

```
auto f1 = [](int a, int b) {return a < b;}; // ok: return type is bool

auto f2 = [](int a, double b) {
             if (a == 0)
                 return b;
             else
                 return a;}; // error: is return type double or int?

auto f3 = [](int a, double b)->double {
             if (a == 0)
                 return b;
              else
                 return a;}; // ok: return type is double

auto f4 = [](double a, double b) {
             if (a < 0)
                 return a;
             else
                 return pow(a, b);}; // ok: return type is double
```

see more about lambda functions
→ Reply

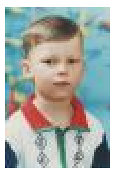
Tensor

5 years ago, # | +1

you can even write your own recursive functions inside the main in lambdas, that's really cool and useful for less code.

But here instead of using auto you should specify the return type and the parameters type of the lambda expression.

see my submission here
→ Reply


anthonycherepkov

5 years ago, # | 0

Thanks. Useful information.
→ Reply

hsnprsd

5 years ago, # | 0

Thank you so much :) I learned a lot :D
→ Reply

5 years ago, # | -16

+669 for vain' blog !why?

+809 for vain blog !why?
→ Reply

CFpolice

5 years ago, # ^ | 0

You are **GiveMinus**! Both of you have a comment "give a kiss baby :)"

give a kiss baby :)
→ Reply

Swift

5 years ago, # ^ | +21

+726 for a lot of useful info, that's why.
→ Reply

Xellos

5 years ago, # | ← Rev. 20 +9

```
warning: ISO C does not permit named variadic macros [-Wvariadic-macros]
#define error(args...)
                     ^
```

could write:

```
#define error(...) { vector<string> _v = split(#__VA_ARGS__, ',');
err(_v.begin(), __VA_ARGS__);}
```

→ Reply

n.eugene

5 years ago, # | 0

The example which is now given for `move` (define `w = move(v)` and then output contents of `v`) is actually undefined behaviour. What the compiler will actually do in this situation is just swap the contents of the two vectors (`v` with the empty `w`); however, in theory `v` is now "junk" and should not be touched at all (it can not even be a vector with arbitrary contents, but just something referring to some arbitrary place in memory, which might, in theory, no longer correspond to any correct contents of a vector, and it can do basically anything when its methods (such as the range-based for loop) are called).
→ Reply

dj3500

5 years ago, # ^ | +25

http://cplusplus.com/reference/vector/vector/operator=

"The move assignment (2) moves the elements of x into the container (x is left in an unspecified **but valid state**)."

We'd better call `v.clear()` after `w = move(v)` to bring `v` to a determinate (empty, actually) state. And then we can access it.
→ Reply

SirNickolas

5 years ago, # ^ | 0

Didn't know that. Thanks for the correction!
→ Reply

dj3500

5 years ago, # | ← Rev. 2 0

Variadic functions and macros are awesome. Now I've got unique functions for debug, input and output, no more gi2, gi3, ... !!!
→ Reply

sparks

5 years ago, # | ← Rev. 3 +20

I like the string literals fucntionality. Sometime it can make code much simpler, especially for competitions:

```
#include <iostream>
using namespace std;

int main() {
        string test = R"END(
                let's test a multiline string
                that can have special chars like ''
                or even ""
                and not to forget \
                and no need to escape!
                This rocks !)END";
```

IvayloS

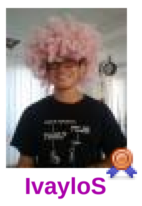
```
                This rocks !)END";
        cout << test << endl;
        return 0;
}
```

And the result on ideone can be seen here.
→ Reply

Swift

5 years ago, # ^ | 0

I didn't know about this! Thank you. Could you please write a tutorial about this, I'll move it to this post.
→ Reply


IvayloS

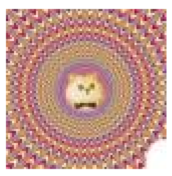
5 years ago, # ^ | +5

c++11 also introduces a set of new string literals. Some of them are really useful for professional programming, but not very helpful for competitions(like UTF-8, UTF-16 and UTF-32 literals) and thus they are not that much of an interest(you can read about them in the wiki article that I link to). However one type of string literal is particularly interesting — the raw string literal. To write a raw string literal you need to prefix the opening quotes with R and immediately after the quotes you should write some delimiter, the delimiter can be a string of up to 16 characters and should not contain whitespace or control characters, You should terminate the string with the same delimiter before the closing quote and also the string should be in brackets(after the delimiter). Here is an example usage:

```
int main() {
        string test = R"END(
                let's test a multiline string
                that can have special chars like ''
                or even ""
                and not to forget \
                and no need to escape!
                This rocks !
                )END";
        cout << test << endl;
        return 0;
}
```

And the output can be seen here.

Note that the string can span multiple lines and that you don't need to escape special characters in it. In this case I use END as my delimiter.
→ Reply


vsamsonov

5 years ago, # | ← Rev. 4 +17

Following is also useful for GCC. Very fast ASM bit operations:

Note, that **offset** can be >=32, any valid offset will work. However, I didn't know if inline assembly allowed in CF. Should work.

```
/* Read bit and set to zero */
inline bool btr (volatile void * mem, size_t offset) {
        bool result;
        __asm__ (
                "btr %2, %1; setc %0;"
                : "=r" (result), "+m" (* (volatile long *) mem)
                : "r" (offset)
                : "cc");
        return result;
}

/* Read bit and set to one */
inline bool bts (volatile void * mem, size_t offset) {
        bool result;
        __asm__ (
                "bts %2, %1; setc %0;"
                : "=r" (result), "+m" (* (volatile long *) mem)
                : "r" (offset)
                : "cc");
        return result;
}

/* Bit value */
inline bool bittest (volatile void * mem, size_t offset) {
        bool result;
        __asm__ (
        __asm__ (
                "bt %1, %2; setc %0;"
                : "=r" (result)
                : "r" (offset), "m" (* (volatile long *) mem)
                : "cc");
        return result;
}

/* Set bit to one */
inline void bitset1 (volatile void * mem, size_t offset) {
        __asm__ ("bts %1, %0;" : "+m" (* (volatile long *) mem) : "r" (offset)
: "cc");
}

/* Set bit to zero */
inline void bitset0 (volatile void * mem, size_t offset) {
        __asm__ ("btr %1, %0;" : "+m" (* (volatile long *) mem) : "r" (offset)
: "cc");
}
```

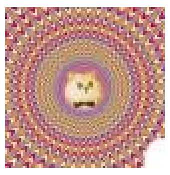
→ Reply

andreyv

5 years ago, # ^ | 0

Why do you need `volatile` everywhere?

→ Reply


vsamsonov

5 years ago, # ^ | ← Rev. 2 0

Just to make sure that value is actually changed. It gives information to the compiler that memory is changed indirectly (inside **asm** block), to avoid unexpected optimizations. Modern compilers have aggressive optimizations. If you used some value from memory, compiler probably saved it to intermediate register. Let's imagine, that you then called bitset on that memory and used value again. Compiler may decide: "Ok, he didn't even touched that **mem** variable, I'll use the old value". But it's wrong. You changed it inside **asm** block. Everything inside **asm** — direct instructions to processor, compiler doesn't know what you are doing there.

→ Reply

andreyv

5 years ago, # ^ | +11

Yes, GCC does not know what is inside the asm block. However, GCC does know which variables are used and modified — you specified this yourself in the asm block input/output operands! In particular, `"+m"` should tell GCC that this variable/location in memory is read and modified.

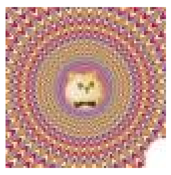
You can see that GCC indeed reloads the value as it should here: http://goo.gl/Jz8SYH. If GCC thought the variable was unmodified, it would do

```
movl    $31, %eax
```

instead (comment out the `btr()` call to see this).

Bottom line: `volatile` is not needed in correct code. The only valid uses for `volatile` I can think of are signal handler flags and hardware registers that are mapped in memory.

→ Reply


vsamsonov

5 years ago, # ^ | 0

Well, it seems like `volatile` is indeed redundant in this case. Clobber "+m" should take care of all things. I put it there just in case. Because redundant information isn't a problem, but lack of information is. `volatile` also comes in handy in multithreaded programs, when you are messing up with custom synchronization/locking technique. Actually anything that involves shared memory involves volatile somehow. In regular programs volatile rarely used, because everything is already written (like synchronization primitives/threadsafe data structures...) and program uses high-level functions for this.

→ Reply

5 years ago, # ^ | +8

I'm sorry for being a nerd, but `volatile` can't andreyv

I'm sorry for being a nerd, but `volatile` can't be used to implement thread synchronization primitives too. Even `volatile sig_atomic_t` won't do. You are confusing `volatile` with atomic operations, which are two different things.
→ Reply

IvayloS

5 years ago, # | 0

Please note that regex is part of the standard but it is not part of g++(at least prior to 4.9). Have a look here. I'm not 100% sure but I think code with regex will not compile on codeforces.
→ Reply

riadwaw

5 years ago, # ^ | 0

actually, regex's compile fine on g++4.6 or 4.7 (I don't remember) but they just worked incorrectly.
→ Reply

IvayloS

5 years ago, # ^ | 0

As is mentioned in the bug I relate to, some of the functionality is not working as expected and some of not implemented at all. As per the comments in the bug I think this is fixed in 4.9. However I think codeforces uses an earlier version.
→ Reply

LittleDreamer

5 years ago, # | 0

array<int, 4> a; a = {5, 8, 9, 2};

This code fails on c++11 compilation with error error: no match for 'operator=' in 'a' no known conversion for argument 1 from '' to 'const std::array<int, 4ul>&'

Need additional braces a = {{5, 8, 9, 2}};
→ Reply

levonog

5 years ago, # | ← Rev. 3 -19

I use some tricks too, for example:

Input in vector n elements:

`for ( int i = 0 ; i < n ; cin >> vec [ i++ ] );`

Or analog of:

```
for(i = 1; i <= n; i++) {
    for(j = 1; j <= m; j++)
        cout << a[i][j] << " ";
    cout << "\n";
}
```

//

```
for(i = 1; i <= n; i++ , cout << endl)
    for(j = 1; j <= m; j++)
        cout << a[i][j] << " ";
```

→ Reply

nic11

5 years ago, # ^ | +14

I would call it not a C++ trick, but a creative way to use for in C++. It's indeed shorter (just a little), but the code is unreadable IMHO.
→ Reply

multisystem

5 years ago, # | +11

This is really priceless!

Just another two tricks that might help.

`std::string to_string( int value ); // Converts a numeric value to std::string.`

`int stoi( const std::string& str, std::size_t* pos = 0, int base = 10 ); // Interprets a signed integer value in the string str.`

For more information, review std::to_string and std::stoi.
→ Reply

TERMINATOR_228

5 years ago, # | +1

Thanks, very interesting. Let's do blogs like this often!

→ Reply

xpertcoder

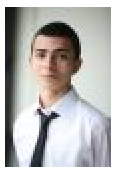
5 years ago, # | ← Rev. 2 0

Can someone tell what I am doing wrong with trick `__builtin_popcount` where it's written `function with suffix 'l' gets a unsigned long argument and with suffix 'll' gets a unsigned long long argument` in this problem

485C - Bits

Solution 9506498 gives WA because of overflow.

→ Reply


sparik

5 years ago, # ^ | +1

`1ll<<i`

→ Reply

xpertcoder

5 years ago, # ^ | 0

Thanks man!! and after that contest I cursed `__builtin_popcount` for making me lose points :P .

I wonder then what is the difference between `__builtin_popcount` and `__builtin_popcountll` as both solution give AC. I thought `__builtin_popcount` should give wrong result if I send long long as an argument.

9506854 --> __builtin_popcountll

and 9506856 __builtin_popcount

→ Reply

Alsh_compiler

5 years ago, # | 0

please show us some tricks in swift language :D :D

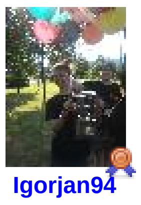
→ Reply

Hepic_Antony_Skarlatos

5 years ago, # | 0

One of the best quick C++/STL tutorials,I have ever read. Congratulations to people who helped for this tut.

→ Reply


Igorjan94

5 years ago, # | ← Rev. 2 +11

It is not part of c++11(only one of this), but useful cpp functions

```
    vector<int> a(n), b(n), c(n);
    iota(a.begin(), a.end(), 1); //c++11
// a = 1..10
    random_shuffle(a.begin(), a.end());
// a = random permutation of a
    partial_sum(a.begin(), a.end(), b.begin());
// b[i] = sum(a[j], j <= i)
    adjacent_difference(a.begin(), a.end(), c.begin());
// c[i] = a[i] - (i == 0 ? 0 : a[i - 1])
    cout << accumulate(a.begin(), a.end(), 123) << "\n";
// x = 123 + sum(a[i])
    cout << inner_product(a.begin(), a.end(), b.begin(), 234) << "\n";
// x = 234 + sum(a[i] * b[i])
```

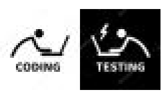
All functions have two iterators as input, some of them have outputIterators and init values. All operators, used in these functions can be user-defined or standard:

```
    cout << accumulate(a.begin(), a.end(), 1, multiplies<int>()) << "\n";
// x = product(a[i])
// foldl in functional languages
    adjacent_difference(a.begin(), a.end(), c.begin(), [](int a, int b){return
a * b;});
// c[i] = a[i] * (i == 0 ? 1 : a[i - 1])
```

These functions are defined in <numeric>

→ Reply

→ Reply



Hepic_Antony_Skarlatos

5 years ago, # | ← Rev. 3 +3

Swift ,I think you forgot a semicolon in your perfect tutorial,right here:

"""""" auto f = [] (int a, int b) -> int { return a + b; } ..HERE.. cout << f(1, 2); // prints "3" """"""

→ Reply

Swift

5 years ago, # ^ | 0

Thanks, now corrected.

→ Reply

DarthKnight

5 years ago, # | +11

Using `complex` , `p.real() = x` or `cin >> p.real()` don't work in C++11 but they do in C++98.

→ Reply

Swift

5 years ago, # ^ | 0

You can use `p.real(x)` in C++11. I don't know any way to `cin` real.

→ Reply

__builtin__wolfy

5 years ago, # | 0

Here is a trick that might interest you. In C++, a class can inherit from a template instantiation of itself. So you can write `class X: vector<X> {...};` for example. Class X inherits the members of vector and you can use this trick to implement multidimensional arrays, tries, and other useful data structure without using pointers. More here.

→ Reply

AKP

5 years ago, # | ← Rev. 2 -11

C++11 Tricks or Traps?

One should not use this:

```
vector<int> s(5);
for(int i=0;i<5;i++) s[i]=(101*i)%37;
for(int z:s) cout<<s[z]<<' ';
```

instead of this:

```
vector<int> s(5);
for(int i=0;i<5;i++) s[i]=(101*i)%37;
for(int z=0;z<s.size();z++) cout<<s[z]<<' ';
```

or, am I missing something?

→ Reply

natsukagami

5 years ago, # ^ | ← Rev. 2 +8

```
for(int z:s) cout<<s[z]<<' ';
```

should be

```
for(int z:s) cout<< z <<' ';
```

→ Reply

AKP

5 years ago, # ^ | 0

Oh I see, misunderstood that, thanks.

→ Reply

Swift

5 years ago, # ^ | 0

You trapped in your own mistake!

→ Reply

nakeep

5 years ago, # | 0

`for(auto& e: ...)` will cause compile error on `vector<bool>` . use universal reference instead: `for(auto&& e: ...)`

→ Reply

z4120

6 weeks ago, # ^ | 0

Note that it's equivalent to this code:

```
std::vector<bool> a{0,0,0};
for(auto x:a) x=1;
```

It will set all elements of `a` to 1. In order to copy the values, it's necessary to use

```
std::vector<bool> a{0,0,0};
for(bool x:a) x=1; // no-operation
```

→ Reply

yhylord

5 years ago, # | 0

There is a tiny typo in the section 6, dijkstra's part: `tie(dist, ode, prev) = q.top(); q.pop();`

should be: `tie(dist, node, prev) = q.top(); q.pop();`

→ Reply

lonerz

4 years ago, # | +46

Here's another trick:

For max/min functions, **these functions don't need to take two parameters, they can take more** :)

Instead of writing,

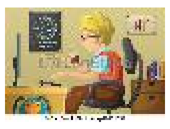
```
int a = 5, b = 6, c = 2, d = 10;
cout << max(a,max(b,max(c,d))) << endl;
```

You can just use "{ }" braces around your parameters and insert a list into the max function (works the same for min function) like below:

```
int a = 5, b = 6, c = 2, d = 10;
cout << max( {a,b,c,d} ) << endl;
```

Here's a source code for reference: http://ideone.com/lllqIK

→ Reply


foundLoveOfMyLife

4 years ago, # ^ | 0

Hey is there a shortcut to Something like:

a = max(a , Something being computed);

I always wanted something like: a+=Something being computed for max too. Although a function with variable parameters can be defined in a template but I don't like working with templates! :)

→ Reply

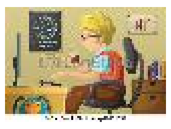
TimonKnigge

4 years ago, # ^ | ← Rev. 3 0

What's wrong with templates? This would work just fine:

```
    template<class T>
    void maxx(T &l, T r) {
        if (l < r) l = r;
    }
```

→ Reply


foundLoveOfMyLife

4 years ago, # ^ | 0

Probably I fear them! Can you suggest some source to read more about templates and classes and stuff!

→ Reply

harrypotter0

14 months ago, # ^ | 0

How does this works? Why "&" only before l and not before r?

→ Reply

14 months ago, # ^ | 0

Since we are only changing l while we iterate and not r.

→ Reply

→ Reply

jheel4

Bredor

4 years ago, # | +5

Here's another trick:

You can write `return 14 / 88` instead of `return 0`

→ Reply

harrypotter0

14 months ago, # ^ | 0

How is it useful?

→ Reply

Faster

4 years ago, # | -8

Can I write a void which like

```
void read(T &a,Args... args) {
    cin << a;
    read(args...);
}
```

and got the result `a=1, b=2, c=3, d=4` if I have input 4 numbers 1, 2, 3, 4 when run `read(a,b,c,d)` ?

→ Reply

Swift

4 years ago, # ^ | 0

Yes. Why do you ask? You can simply test it by doing so!

→ Reply

Faster

4 years ago, # ^ | 0

I got this error

```
/home/tunc/Documents/try_C++11.cpp: In instantiation of ‘void
read(T&, Args ...) [with T = int; Args = {int, int, int}]’:
/home/tunc/Documents/try_C++11.cpp:36:14:   required from here
/home/tunc/Documents/try_C++11.cpp:14:9: error: no match for
‘operator<<’ (operand types are ‘std::istream {aka
std::basic_istream<char>}’ and ‘int’)
     cin << A;
         ^
/home/tunc/Documents/try_C++11.cpp:14:9: note: candidates are:
In file included from /usr/include/c++/4.8/bitset:1578:0,
                 from /usr/include/x86_64-linux-
gnu/c++/4.8/bits/stdc++.h:65,
                 from /home/tunc/Documents/try_C++11.cpp:1:
/usr/include/c++/4.8/debug/bitset:405:5: note: template<class
_CharT, class _Traits, long unsigned int _Nb>
std::basic_ostream<_CharT, _Traits>&
std::__debug::operator<<(std::basic_ostream<_CharT, _Traits>&,
const std::__debug::bitset<_Nb>&)
     operator<<(std::basic_ostream<_CharT, _Traits>& __os,
     ^
etc.
```

when I ran that code. How to fix it?

→ Reply

_index

4 years ago, # ^ | ← Rev. 2 +1

lol, change

```
cin << a
```

to

```
cin >> a;
```

→ Reply

4 years ago, # ^ | Rev. 3 0

I changed it, but when i ran with `1 2 3 4` the result was

Faster

I changed it, but when I ran with `1 2 3 4` the result was `1 0 0 0`. How to fix it?

p/s: haha, I learnt to code for a while but now I still get that mistake =)) so ashame =))

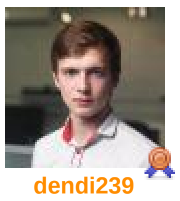
→ Reply

Nizil

4 months ago, # ^ | 0

You probably need to pass the rest of the arguments by reference somehow, not only the first one.

→ Reply


dendi239

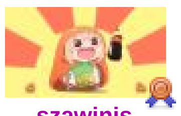
8 days ago, # ^ | ← Rev. 3 0

Here's how.

```
template <>
void read() {}

template <class Arg, class Rest>
void read(Arg &arg, Rest
&...rest) {
    cin >> arg;
    read(rest...);
}
```

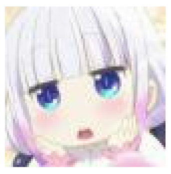
→ Reply


szawinis

4 years ago, # | 0

The Dijkstra code that uses emplace_back + tie has a little typo: node is spelt as ode

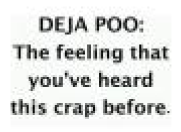
→ Reply


Hossam

3 years ago, # | 0

Thanks a lot! I am beginning to love C++ <3

→ Reply

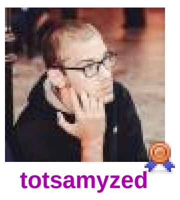

DEJA POO:
The feeling that
you've heard
this crap before.
SarvagyaAgarwal

3 years ago, # | 0

How do I define the "rep" macro if i want to include the end indexes too ?

Like -> rep(i,1,10) prints 1...10 rep(i,10,1) prints 10....1 .

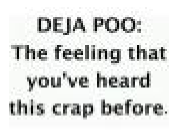
→ Reply


totsamyzed

3 years ago, # ^ | +6

An ugly way, but it works. link

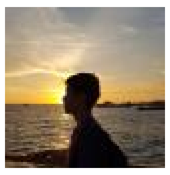
→ Reply


DEJA POO:
The feeling that
you've heard
this crap before.
SarvagyaAgarwal

3 years ago, # ^ | +1

The link you mentioned isn't working . Can you post it on ideone ?

→ Reply


thienlongtpct

3 years ago, # ^ | 0

```
#define ftoa(i, x, y, a) for(int i = (x); i != (((x) < (y)) ?
(((y)-(x))/a+1)*a+(x) : (x)-(((x)-(y))/a+1)*a); i += ((x) <
(y)) ? (a) : -(a))
```

I have use this code and try 1000 test cases to make sure that it is correct.

Here is 3 codes:

By `ftoa`

By normal `for`

Make test case

Note: to make the test cases you download these 3 codes and then run the third one. It will automatically run.

→ Reply

tera_coder 3 years ago, # | 0

Thanks for the great tips; but are all of them usable without C++14?

→ Reply

AlexandruValeanu 3 years ago, # ^ | 0

Most of them are. Is there any reason why you would use C++11?

→ Reply

tera_coder 3 years ago, # ^ | 0

Because of onsite contest limitations.

→ Reply

tera_coder 3 years ago, # ^ | 0

Which one's aren't needing C++14? Thank you.

→ Reply

egor.okhterov 3 years ago, # | 0

Why would you use

```
array<int, 4> a;
```

instead of

```
int a[4];
```

?

→ Reply

Swift 3 years ago, # ^ | +5

To use it as elements of vector for example.

`vector<array<int, 4>> v`

→ Reply

egor.okhterov 3 years ago, # ^ | 0

What are the advantages of `vector<array<int, 4>> v;` over `vector<vector<int>> v;` ?

→ Reply

MrDindows 3 years ago, # ^ | 0

Memory will be allocated only once.

→ Reply

AlexandruValeanu 3 years ago, # ^ | 0

Because you can compare arrays, access elements with bound-checking or get iterators support.

→ Reply

ShafinKhadem 22 months ago, # ^ | 0

I think biggest advantage is `map<long long int, array<int,4> > m`

→ Reply

M.A.H.M.O.O.D 3 years ago, # | ← Rev. 5 -13

Why the downvotes I didn't say anything wrong did I ???

Here's a submission by me using what I described (the check function)23252012(I got WA because the idea is wrong not the implementation)

My life now is a lot easier...Thank you **Swift**.

:)

I'm not sure if this is well known but in C++ you can give a default value to a function for example:

void DFS(int node, int par = -1){

...

}

}

int main(){

// input a graph

DFS(1);

// rest of the code

}

the DFS function works as a normal function but when you don't provide a second parameter it will take the default value you have given it as its value...hope this helps.

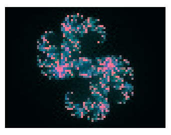
:)
→ Reply

seenu 3 years ago, # | 0

thanks Swift
→ Reply

iit_sujal 3 years ago, # | 0

Great Work Man
→ Reply

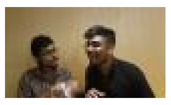

ekzhang 3 years ago, # | +10

Old post, but one important mistake: there should be no `std::move()` call at the end of your `split()` function. `std::move()` should never be used to move automatic objects out of functions.

Source
→ Reply

Swift 2 years ago, # | 0

*Auto comment: topic has been updated by **Swift** (previous revision, new revision, compare).*
→ Reply


mochow 2 years ago, # | -10

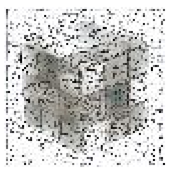
Now that C++17 is here in CF, is there anything new and useful in the newer edition that we can use in competitive programming?
→ Reply

KarlisS 2 years ago, # ^ | ← Rev. 2 +8

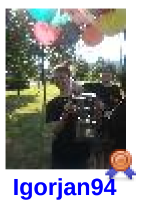
Gcd, structured bindings, clamp.
→ Reply

iLoveIOI 2 years ago, # ^ | 0

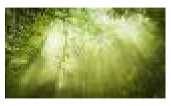
how do you write GCD function in c++17
→ Reply


Jakube 2 years ago, # ^ | 0

std::gcd
→ Reply


Igorjan94 2 years ago, # ^ | +1

Here are you
→ Reply


atlasworld 2 years ago, # | 0

nice blog !
→ Reply

2 years ago, # | ← Rev. 2 0

Also, one more cool thing C++(11?) has is the `throw` instruction and `try/catch`. You can get

retrograd

Also, one more cool thing C++(11?) has is the `throw` instruction and `try/catch`. You can get out of recursive call stacks and treat "No solution" / "Solution found" cases much more easily.

Example:

```
try {
   DFS(0);
   PrintSolution();
} catch (int) {
   PrintNoSolution();
}
```

→ Reply

22 months ago, # | ← Rev. 2 0

ShafinKhadem

Thanks a lot for the awesome tutorial, specially for the debug function. But it doesn't work perfectly if there is space in the macro parameter, e.g. `error(get<0> (tuple1), get<0> (tuple2));` Besides, replacing comma with spaces is also unnecessary, when we can tokenize based on comma:

```
#define bug(args...) { cout<<__LINE__<<": "; string s = #args; istringstream
ss(s); err(ss, args); }

void err(istringstream &ss) { cout<<"\n"; }
template<typename T, typename... Args>
void err(istringstream &ss, const T &a, const Args & ... args) {
    string token;
    getline(ss, token, ',');
    cout << token << " = " << a << "; ";
    err(ss, args...);
}
```

→ Reply

14 months ago, # ^ | 0

harrypotter0

ShafinKhadem Could you provide some working of this debugger?

Thank you.

→ Reply

14 months ago, # ^ | +1

ShafinKhadem

After some days, I realized that tokenizing on comma is a bad idea, as it fails in cases like bug(func(a,b),func(c,d)), but if we tokenize based on space, we can easily avoid and add some spaces to make it work. Now-a-days I use it like this:

```
#include <bits/stdc++.h>
using namespace std;

#define bug(args ...) cerr << __LINE__ << ": ", err(new
istringstream(string(#args)), args), cerr << '\n'
void err(istringstream *iss) {}
template<typename T, typename ... Args> void err(istringstream
*iss, const T &_val, const Args & ... args) {
    string _name;
    *iss >> _name;
    if (_name.back()==',') _name.pop_back();
    cerr << _name << " = " << _val << "; ", err(iss, args ...);
}

int func(int a, int b) {
    return a+b;
}

int main() {
    int x = 1, y = 2, n = 3, m = 4;
    bug(x, y, func(x,y), m, n, func(m,n));
    bug(m, n, m*n, x, y, x*y);
    return 0;
}
```

Notes: After every token u must add both comma and space and there should not be space in single token (e.g. func(x,y), x*y). It won't compile in versions older than c++11.

→ Reply

14 months ago, # ^ | 0

Okay thanks for the explanation and fast reply :)

harrypotter0

→ Reply

14 months ago, # | +3

CountZero

use std::tie to write complex comparators:

```
// before
bool cmp(int i, int j) {
  if (x[i] != x[j]) return x[i] < x[j];
  if (y[i] != y[j]) return y[i] < y[j];
  return z[i] < z[j];
}
// after
bool cmp(int i, int j) {
  return tie(x[i], y[i], z[i]) < tie(x[j], y[j], z[j]);
}
```

range-for:
you can use it for input:

```
vector<int> v(n);
for (auto& x: v) cin >> x;
```

works with C-style arrays too:

```
int v[5];
for (auto& x: v) cin >> x;
```

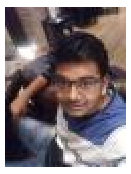
actually you can use std::array instead of C-style arrays:

```
// before
int a[maxn], b[maxn], c[maxn], d[maxn];
// after
array<int, maxn> a, b, c, d;
```

how to reference the global variable if there's local one with the same name:

```
int v[5];
void f() {
  bool v = false;
  ::v[0] += 1;
}
```

→ Reply



mynk322.0

13 months ago, # | 0

It's awesome thanks for the blog!!

→ Reply

redimer

7 months ago, # | ← Rev. 2 +3

c++ 17 Better (for faster execution) used *int* instead *short* or *bool* or *__int64*

example: const int MAX = 1e4; vector< int > v(MAX); //instead vector< bool > v(MAX); int score;

//to use logical operations: for (int a=0;a<1e4;++a) for (int b=0;b<1e4;++b) score += (v[a] ^ v[b]);

you can be sure of solving the problem http://acmp.ru/index.asp?main=task&id_task=659

→ Reply

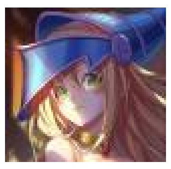
adi_1992

5 months ago, # | ← Rev. 3 0

define rep(i, begin, end) for (__typeof(end) i = (begin) — ((begin) > (end)); i != (end) — ((begin) > (end)); i += 1 — 2 * ((begin) > (end)))

does not work with set and map container as iterators dont support operator>

→ Reply



DarkMagician09

5 months ago, # | -8

That (bits/stdc++.h) Library doesn't actually include everything like these two.

```
#include<unordered_map>
#include<regex>
```

If you didn't know or you miss this information, because I searched for hours on the error for calling unordered_map in my code when including that bits only :D , so I suggest editing the post for these two.
→ Reply

NRK7

5 months ago, # ^ | +11

If you use c++11 or later (which I think everyone should), using bits/stdc++.h includes them too.
→ Reply

heyyolol

5 months ago, # | +8

Are variables in namespace initialised to 0 for c++? Thanks in advance.
→ Reply

NeverRegret

4 weeks ago, # ^ | 0

For better clearification

C++ does not initialize most variables to a given value (such as zero) automatically. Thus when a variable is assigned a memory location by the compiler, the default value of that variable is whatever (garbage) value happens to already be in that memory location!
→ Reply

chrome

4 months ago, # | 0

In c++17 we can do things like this:

```
std::map<std::string, int> m = {{"first", 1}, {"second", 2}};

for (auto &[key, value] : m) {
  std::cout << key << " " << value << std::endl;
}

struct state {
  int a, b, c;
};

std::vector<state> v = {{1, 2, 3}, {4, 5, 6}};

for (auto &it : v) {
  // we can use this, instead of std::tie operator
  auto [a, b, c] = it;
  std::cout << a * b * c << std::endl;
}
```
→ Reply

ritik_patel05

3 months ago, # ^ | 0

That's something new I saw. Thanks :)
→ Reply

IWanna_MightiestDisciple

3 months ago, # | 0

eagerly waiting for +1000 :D <3 such a good blog post :D
→ Reply

Manan_shah

7 days ago, # | 0

Another useful thing would be to precompile the <bits/stdc++.h> header to reduce the compilation time. Just compile it as you normally compile in the folder having that file. The compiled file would have a .gch extension.
→ Reply