Swift's blog

C++ Tricks

By Swift, 6 hours ago,

WARNING: Many of these things are belong to C++11 so use C++11 in order to test anything here :)

This is a great C++11 tutorial for those who want to know more about C++11.

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";
}
// prints "3 4
//         5 6
//         -
```

→ Pay attention

Before contest
Codeforces Round #285 (Div. 2)
5 days

Before contest
Codeforces Round #285 (Div. 1)
5 days

Like You, Mahmudul Tushar and 127 others like this.

→ habib_cse_ruet

Rating: **969**
Contribution: **0**

- Settings
- Blog
- Teams
- Submissions
- Favourites
- Talks
- Contests

habib_cse_ruet

→ Top rated

#	User	Rating
1	tourist	3254
2	Petr	3002
3	rng_58	2898
4	WJMZBMR	2831
5	yeputons	2764
6	vepifanov	2740
7	scott_wu	2719
8	0O0o00OO0Oo0o0Oo	2718
9	Egor	2702
10	sankear	2691

Countries | Cities | Organizations View all →

→ Top contributors

#	User	Contrib.
1	Petr	141
2	Endagorion	137
2	Egor	137
2	Fefer_Ivan	137
5	DmitriyH	134
5	I_love_Tanya_Romanova	134
7	illi	133

```
//         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)

8	marat.snowbear	132
8	Bredor_228_Jaguar_Turnik	132
10	Xellos	130
10	cgy4ever	130

View all →

→ Find user

Handle:

Find

→ Recent actions

Swift → C++ Tricks

shashank21j → [HackerRank] Weekly 13th edition

PrinceOfPersia → Hello 2015 Contest

akash2504 → spoj - Vertex Cover

GiveMinus → cheat

bli0042 → Facebook Hacker Cup 2015

GlebsHP → Заочный этап Открытой олимпиады по программированию 2014-2015

flashion → Robbery problem

Kalanon → First "First Accept"!

hagu → Gym problem

MikeMirzayanov → Codeforces Round #277.5 (Div. 2) Editorial [A-D for now]

Baian98 → I've decided to create a game

dans → Codeforces Round #284 Editorial

besher → DS problem

tncks0121 → Good Bye 2014

beatoriche → Java vs. ActionScript

Egor → Can't solve problem :(

gvikei → Basic Binary Indexed Tree (English version)

andreyv → Catching silly mistakes with GCC

tncks0121 → Good Bye 2014 Editorial (A-G is all available)

Colonelmo → SGU problem 101, hack me

M.D → Bug

KhaustovPavel → Codeforces Round #242 (Div. 2) Editorial

alv-r- → Help needed on problem E — New Year Domino — from Goodbye 2014 Contest

300pound → Problem with 496C test case 20!

Detailed →

```
__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)

```
__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:

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

using namespace std;

#define error(args...) { vector<string> _v;                    \
                         string _s = #args;                    \
                         replace(_s.begin(), _s.end(), ',', ' '); \
                         splitstr(_s, _v);                     \
                         err(_v.begin(), args);                \
                       }

void splitstr(const string &s, vector<string> &v) {
        istringstream in(s);
        copy(istream_iterator<string>(in), istream_iterator<string>(),
back_inserter(v));
}

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

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.

c++, **c++0x**, **tricks**

+220 Swift 6 hours ago 44

Comments (44)

Write comment?

ConfusedGuy

5 hours ago, # | +2

Its awesome. Thanks Swift :)

→ Reply

5 hours ago, # | +2

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

slycelote

→ Reply

Swift

5 hours ago, # ^ | ← Rev. 2 0

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 hours ago, # ^ | +1

http://ideone.com/6l4Wc7

→ Reply

Swift

5 hours ago, # ^ | 0

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

Thank you.

→ Reply

EeOneGuy

5 hours ago, # ^ | 0

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

→ Reply

Swift

4 hours ago, # ^ | ← Rev. 2 0

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

→ Reply

Determinism

4 hours ago, # | ← Rev. 2 0

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

Swistakk

4 hours ago, # | +1

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

→ Reply

Swift

4 hours ago, # ^ | ☆ 0

LMAO =))

→ Reply

GiveMinus

4 hours ago, # | ☆ -41

→ Reply

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

Swistakk

4 hours ago, # | ☆ +1

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

4 hours ago, # ^ | ☆ 0

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

→ Reply

IWillBeRed

4 hours ago, # | ☆ +3

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

→ Reply

Swift

3 hours ago, # ^ | ☆ ← Rev. 2 +3

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

3 hours ago, # | ☆ 0

Thanks for such a nice explanation...

→ Reply

fushar

3 hours ago, # | ☆ +4

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

→ Reply

Swift

3 hours ago, # ^ | ☆ +2

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

87 minutes ago, # ^ | ☆ 0

yeah, that works, I did the same :)

→ Reply

fushar

78 minutes ago, # ^ | ☆ 0

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

→ Reply

Swift

71 minute(s) ago, # ^ | ☆ 0

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

→ Reply

Corei13

3 hours 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

3 hours 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

106 minutes ago, # ^ | ☆ 0

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

88 minutes ago, # ^ | +9

Swift

→ Reply

baklazan

3 hours ago, # | 0

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

na2a

2 hours ago, # | ← Rev. 2 +3

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

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

instead of this

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

GiveMinus

2 hours ago, # ^ | -16

why u downvoted me ?

c++ is bullshit
→ Reply

Xellos

2 hours ago, # ^ | 0

→ Reply

Swift

2 hours ago, # ^ | ☆ 0

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

→ Reply

GiveMinus

71 minute(s) ago, # ^ | ☆ 0

give a kiss baby :)

→ Reply

Swift

68 minutes ago, # ^ | ☆ +9

Here you are:

→ Reply

GiveMinus

52 minutes ago, # ^ | 0

tanx

→ Reply

Swift

2 hours ago, # ^ | 0

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

→ Reply

Ximera

67 minutes ago, # | 0

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 == n];
```

→ Reply

Swift

59 minutes ago, # ^ | ← Rev. 2 +2

Well, Great creativity :) [Actually `" \n"[i == n]` is correct, I think that was a typo.]

`" \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] << (i == n)["
\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

52 minutes ago, # ^ | 0

no

→ Reply

52 minutes ago, # | ☆ ← Rev. 2 +1

technetium28

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

41 minute(s) ago, # ^ | ☆ 0

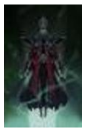
Swift

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

38 minutes ago, # | ☆ 0


HekpoMaH

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

32 minutes ago, # ^ | ☆ ← Rev. 3 +2

Swift

Use this approach:

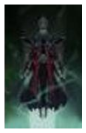
```
vector<int> dp = {4, 5, 6, 4, 8};
for (auto i = ++dp.begin(); i != dp.end(); ++i)
    *i += *(i - 1);
for (auto i: dp)
    cout << i << '\n';
```

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

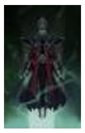
20 minutes ago, # ^ | ☆ 0

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

Swift

14 minutes ago, # ^ | ☆ ← Rev. 3 +1

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

2 minutes ago, # ^ | ☆ +1

XD yeah I agree about this one.
→ Reply