Sanitator's blog

Pretty and powerful 4-liner for debugging (C++)

By **Sanitator**, history, 4 years ago, translation, 

Ever wanted to look in console at variables you use? Meet these helper functions!

dbg()

DeBuG

Suppose we have some nested containers(`vector, string, bitset, set, map`) or arrays, which for simplicity we may consider a multidimensional array. `dbg()` can neatly print the name, bounds and, at last, the values from the required sub-array with automatic bound checking:

For example:

```
int j[2][2][3] = {{{4,5,6},{10,11,12}}, {{1,2,3}, {7,8,9}}};
dbg(j);
dbg(j, 0,0, 0,1, 0,1);

output:
[[[4, 5, 6],
  [10, 11, 12]],
 [[1, 2, 3],
  [7, 8, 9]]]

[[[4, 5],
  [10, 11]]]
```

Another example:

You pass the name of array and **[two closed bounds]** for each dimension(btw, you can omit several last bounds). If they are too large, `dbg()` reduces them. By default the bounds are set on the start and the end of each dimension.

+If you pass bounds `[l;r]` to the dimension that is map or set, the output goes from the **l**th largest to the **r**th largest key, or to the last element of dimension(if **r** is too big).

+ `dbg()` works with **c-arrays whose sizes of dimensions are constant and known at compile time**.
first example
second example

```
/*-----------------------------------------------*/
```

dbgm()

DeBuG Multiple

You can print the names of several variables first and values next:

```
    string s = {"codeforces"};
    int t = 5; char u = 'R';
    pair<pair<double, unsigned int>, pair<int, string>> v = {{234.34534, 42},
{133, "IOI"}};

    dbgm(s,t,u,v);

    output:

    [s,t,u,v]: "codeforces" | 5 | R | ((234.345340, 42), (133, "IOI")) |
```

```
/*-----------------------------------------------*/
```

Here's my code. It's hugely inspired by this submission by **tourist**.

The compact version is created from the extended one by means of http://removelinebreaks.net/.

```
/*-----------------------------------------------*/
```

Hope these functions save your precious minutes during contests. Enjoy!

Thanks to this post and this suggestion by **HosseinYousefi**

Full version of the dbg*() library is here. For printing tuples I used the code from this blog

UPD1: a link to the full library added
UPD2: tuple printing added

tricks, debug, c++, dbg

+39 ☆ Sanitator 4 years ago 25

Comments (24)

Show archived | Write comment?

sneaking

4 years ago, # | ☆ +66

I'd rather use my shitty debugging skills which I understand than using something cryptic like this.

→ Reply

Roach00

4 years ago, # ^ | ☆ ← Rev. 3 +20

true , lol but still we should appreciate the poster for trying to help community.

→ Reply

Sanitator

4 years ago, # ^ | ☆ ← Rev. 2 +40

Well, I just wanted to share my shitty debugging skills

→ Reply

Errichto

4 years ago, # | ☆ +157

One of your "four lines" has more than 2000 characters...

→ Reply

4 years ago, # ^ | ☆ +16

I would also like to thank **MikeMirzayanov** for the great Codeforces Custom invocation which doesn't have automatic line breaking

→ Reply

Sanitator

4 years ago, # ^ | ☆ ← Rev. 4 -29
→ Reply

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

Sanitator

4 years ago, # | ☆ +8

While your function seems to work, why did you obfuscate everything in a single line? Good luck changing the function in case you have a bug or want to add something new to it...

→ Reply

ajecc

4 years ago, # ^ | ☆ ← Rev. 6 +3

Because I provided extended code in the post. And it's easier to copy-paste 4 lines of code, than 42

→ Reply

Sanitator

4 years ago, # | ☆ +33

Every time I see the things people use to debug in C++ (how did this ever pass review I wonder?), I'm glad I switched to D quite a while ago.

example

```
import std.stdio;
void main () {
        auto a = [[[2, 3], [4]], [[6, 2], [4, 5]]];
        writeln (a);
        writefln ("%(%(%s and %), %)", a);
        auto d = [["fjs", "sdf"], ["sas"]];
        writeln (d);
}
```

Result:

```
[[[2, 3], [4]], [[6, 2], [4, 5]]]
[2, 3] and [4], [6, 2] and [4, 5]
[["fjs", "sdf"], ["sas"]]
```

Gassa

Perhaps in 20 years, the C++ committee will agree on a feasible means of debug output to have in the standard library. But life, or ICPC eligibility, or whatever other contests, they are here and now.

→ Reply

Sanitator

4 years ago, # ^ | ☆ ← Rev. 2 -16

Moses said

→ Reply

riadwaw

4 years ago, # ^ | ☆ ← Rev. 2 +8

While I agree with the sentiment that C++ is changing too slowly in some areas where it should be changing faster, code you linked can be trivially written to support any number of args without copypaste

→ Reply

Gassa

4 years ago, # ^ | ☆ +3

You are right, the strange addition to `testlib.h` could perhaps be rewritten as a recursive template to accept any number of arguments. And it will be cleaner and shorter. And it may look trivial once it's done.

However, the mere existence of this commit shows that doing it "right" still requires some effort: learning a bit more of C++ features and/or making sure the result compiles on a bunch of compiler versions.

Alright, there's no point I'm trying to make here, just an observation.

→ Reply

Rishabh_Verma

4 years ago, # | ☆ ← Rev. 2 +9

You are gonna save a lot of time of guys like us, in the upcoming journey of CP.

Thanks a lot.

→ Reply

CountZero

4 years ago, # | ☆ +25

```
let x = vec![vec![1, 2, 3], vec![4, 5]];
println!("{:?}", x);
```

laughs in rust

→ Reply

Sanitator

4 years ago, # | ☆ ← Rev. 6 -34

I left my team

→ Reply

jyttoby

4 years ago, # | ☆ +14

Anyway, you will never have this in a formal competition.

→ Reply

srijan123j

4 years ago, # | ☆ ← Rev. 2 0

.

→ Reply

y0u_kn0w_wh0

4 years ago, # | ☆ +5

Sanitator, Its very helpful. I came across a `bug` , hope you fix it soon.

code

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

#define nl '\n'
#define dbg(...) cout << "[" << #__VA_ARGS__ << "]: ", cout <<
to_string(__VA_ARGS__) << endl
/*compact version*/
template <typename T, size_t N> int SIZE(const T (&t)[N]){ return
N; } template<typename T> int SIZE(const T &t){ return t.size(); }
string to_string(string &s, int x1=0, int x2=1e9){ return '"' +
((x1 < s.size()) ? s.substr(x1, x2-x1+1) : "") + '"'; } string
to_string(const char* s) { string tmp(s); return to_string(tmp); }
string to_string(bool b) { return (b ? "true" : "false"); } string
to_string(char c){ return string({c}); } template<size_t N> string
to_string(bitset<N> &b, int x1=0, int x2=1e9){ string t = ((x1 <
b.size()) ? (b.to_string()).substr(x1, x2-x1+1) : "");
reverse(begin(t), end(t)); return '"' + t + '"'; } template
<typename A, typename... C> string to_string(A (&v), int x1=0, int
x2=1e9, C... coords); int l_v_l_v_l = 0, t_a_b_s = 0; template
<typename A, typename B> string to_string(pair<A, B> &p) {
l_v_l_v_l++; string res = "(" + to_string(p.first) + ", " +
to_string(p.second) + ")"; l_v_l_v_l--; return res; } template
<typename A, typename... C> string to_string(A (&v), int x1, int
x2, C... coords) { int rnk = rank<A>::value; string tab(t_a_b_s, '
```

```
); string res = "", bool first = true, if(l_v_l_v_l == 0) res +=
nl; res += tab + "["; x1 = min(x1, SIZE(v)), x2 = min(x2, SIZE(v));
auto l = begin(v); advance(l, x1); auto r = l; advance(r, (x2-x1) +
(x2 < SIZE(v))); for (auto e = l; e != r; e = next(e)) { if
(!first) { res += ", "; } first = false; l_v_l_v_l++; if(e != l){
if(rnk > 1) { res += nl; t_a_b_s = l_v_l_v_l; }; } else{ t_a_b_s =
0; } res += to_string(*e, coords...); l_v_l_v_l--; } res += "]";
if(l_v_l_v_l == 0) res += nl; return res; }
/*end of compact version*/

int main()
{
  int n = 3;
  string a[3] = {"p","q","r"};
  string b[n] = {"p","q","r"};
  dbg(a);
  dbg(b);
}
```

output

```
[a]:
["p", "q", "r"]

[b]: true
```

→ Reply

4 years ago, # ^ | ☆ ← Rev. 3 +5

though it has some limitations

Frankly speaking, currently I can't come up with idea how to not manually pass the sizes of dimensions of a variable sized array to `dbg()` , so the only thing I could suggest is to set constant sizes of dimensions of arrays at compile time. Any ideas without specific functions for 2D, 3D... arrays?

Sanitator

first option

```
const int N = 3, M = 2;
int main(){
    for(int i = 0; i < 2; ++i){
        char b[N][M] = {'p','q','r','s'};
        dbg(b);
    }
}
```

first option

```
int main(){
    for(int i = 0; i < 2; ++i){
        char b[3][2] = {'p','q','r','s'};
        dbg(b);
    }
}
```

→ Reply

Nizil

3 years ago, # ^ | ☆ 0

Do you have an idea for 1D arrays?

→ Reply

liv_1n9_w08

4 years ago, # | ☆ +8

I wonder where were LanceTheDragonTrainer

→ Reply

HosseinYousefi

4 years ago, # | ☆ +4

I got notified of this post because you tagged me in it today.

Since it's not practical to actually type this code, we should have it somewhere in our template and use it in online contests like codeforces.

So why not using a complete header file like PrettyPrint?

Put it in your directory and then add couple of lines in your code, something like:

```
#if !defined(ONLINE_JUDGE)
#include "prettyprint.hpp"
#endif
```

→ Reply

z4120

3 years ago, # | ☆ ← Rev. 2 +3

Note: GDB has built-in pretty-printing of data structures. So if you know how to use GDB it would be much easier.

Using GDB also have the advantage of being able to print user-defined struct without defining operator<< for them.

It's recommended to check the environment before the real contest day, because it's possible that you need to explicitly turn them on to use them, or there are multiple versions of GDB installed and not all of them have the feature. (See for

example https://stackoverflow.com/questions/49654414/how-to-enable-gdb-pretty-printing-for-c-stl-objects-in-eclipse-cdt . However if `gdb_printers` can't be found in the machine then you're out of luck)

→ Reply

3 years ago, # | ☆ 0

C++ macro debug support color, line number, print pair, stl container...

```
int n = 5;
vector<int> v = {1,2,3,4};
pair<int,string> p = {1, "codeforces"};
debug(n, v, p);
```

```
Line 60: n = 5; v = [1, 2, 3, 4]; p = {1, codeforces};
```

For linux, windows 10 enable virtual terminal processing

Spoiler

giahuyng98

```
#include <bits/stdc++.h>
#ifdef LOCAL
#define _NTH_ARG(_1, _2, _3, _4, _5, _6, N, ...) N
#define _FE_1(_CALL, x) _CALL(x)
#define _FE_2(_CALL, x, ...) _CALL(x) _FE_1(_CALL, __VA_ARGS__)
#define _FE_3(_CALL, x, ...) _CALL(x) _FE_2(_CALL, __VA_ARGS__)
#define _FE_4(_CALL, x, ...) _CALL(x) _FE_3(_CALL, __VA_ARGS__)
#define _FE_5(_CALL, x, ...) _CALL(x) _FE_4(_CALL, __VA_ARGS__)
#define _FE_6(_CALL, x, ...) _CALL(x) _FE_5(_CALL, __VA_ARGS__)
#define FOR_EACH_MACRO(MACRO, ...)
\
  _NTH_ARG(__VA_ARGS__, _FE_6, _FE_5, _FE_4, _FE_3, _FE_2, _FE_1)
\
  (MACRO, __VA_ARGS__)
#define watch(x) cerr << "\033[1;32m" #x " = \033[1;34m" << (x) <<
"\033[0m; ";
#define debug(...)
\
  cerr << "\033[2;31mLine " << __LINE__ << ": \033[0;m";
\
  FOR_EACH_MACRO(watch, __VA_ARGS__)
\
  cerr << endl
#else
#define debug(...)
#endif
```

```
using namespace std;

template <class T1, class T2>
ostream &operator<<(ostream &os, const pair<T1, T2> &p) {
  return os << '{' << p.first << ", " << p.second << '}';
}

template <class T, class = decltype(begin(declval<T>())),
          class = enable_if_t<!is_same<T, string>::value>>
ostream &operator<<(ostream &os, const T &c) {
  os << '[';
  for (auto it = begin(c); it != end(c); ++it)
    os << (it == begin(c) ? "" : ", ") << *it;
  return os << ']';
}

int main() {
  int n = 5;
  vector<int> v = {1,2,3,4};
  pair<int,string> p = {1, "codeforces"};
  debug(n, v, p);
}
```

→ Reply