


dccllyde's blog

My C++ template, incl. fancy colored debug printouts

By [dccllyde](#), [history](#), 5 months ago, 

My C++ template, incl. fancy colored debug printouts

Use [this template](#) and you'll gain 350 rating right away! OK maybe not, but it has a bunch of cool features I built up while climbing from 1800 to 2150 in February-May of this year, and I bet some of them can be helpful to others too. Please, comment with feedback/suggestions /feature requests about how to make the template more useful :)

Links

- [Current template.cpp](#) — includes updates made after this blog was posted. This is the recommended version for most people.
- [Minimal version with debug printouts only](#) — Stripped-down version that compiles faster while still providing the `dbg()` setup, which is IMO the most novel and cool feature in the template. Recommended if your computer is too slow to compile the main version. But please try the precompiled headers trick first, since that helps the compile times a lot!

Specific cool tricks in the template

Debug printouts

Here's an output screenshot:

```

show this on left 822 : [ a, b, c, "", s, q ] : 14 -0.030000 '@' "" "Test String" {1, 6, 4, 3, 5, 3, 1}
diff colors      823 : [ make_pair(a, c) ] : (14, '@')
expressions work too 824 : [ a, b ] : 14 -0.030000
825 : [ 2*b, gcd(a, a+4) ] : -0.060000 2

834 : [ dat ] : {{{(-5, 0), {"tourist", "jiangly", "um_nik", "slime", "ksun48"}}, {(3, 5), {"abc", "def"}}, {(4, -1), {"apple", "peach", "banana"}}}
836 : [ pdh(dat) ] : ""

0: ((-5, 0), {"tourist", "jiangly", "um_nik", "slime", "ksun48"})
1: ((3, 5), {"abc", "def"})
2: ((4, -1), {"apple", "peach", "banana"})
..

14 -0.03 @
845 : [ pdh(vbig, 10) ] : ""
0: (0, 0)
1: (1, -1)
2: (2, -4)
3: (3, -9)
4: (4, -16)
5: (5, -25)
6: (6, -36)
7: (7, -49)
8: (8, -64)
9: (9, -81)
... (full length 100)

849 : [ pdhf(vbig, [&]{auto x}{return x.second;}, 5) ] : ""
0: 0
1: -1
2: -4
3: -9
4: -16
... (full length 100)

done! 851 : [ ] :
896 : [ TIME() ] : 0.000218

```

...and here's the code I used :

```

void solve() {
    ll a = 14; double b = -3e-2; char c = '@'; string s = "Test String";
    vector<ll> q = {1, 6, 4, 3, 5, 3, 1};
    dbg(a, b, c, "", s, q);
    dbg("show this on left", make_pair(a, c)); // dbg = "debug with 1st arg
as comment"
    dbgP("diff colors", a, b); // uppercase letters change the color; all
terminal colors are available
    dbg("expressions work too", 2*b, gcd(a, a+4));
    el; // empty line

    // complex/nested data structures work too.
    map<pair<ll, ll>, V<string>> dat = {
        {{3, 5}, {"abc", "def"}},
        {{4, -1}, {"apple", "peach", "banana"}},
        {{-5, 0}, {"tourist", "jiangly", "um_nik", "slime", "ksun48"}}
    };
    dbgY(dat);
    // that may be pretty messy to read all on one line though, so we can use:
    dbg(pdh(dat));
    el;

    // show how "regular output" lines look visibly different from dbg() lines.

```

```

cout << a << ' ' << b << ' ' << c << '\n';

// what if we have a big object and we want to "get the idea" of its
contents
// without printing the whole thing?
vector<pll> vbig; FOR(k, 0, 100) {vbig.emplace_back(k, -k*k);}
dbgR(pdh(vbig, 10)); // short for "print_details_helper"

el;
// Advanced: pdhf lets me specify a function, so f(x) will be printed for
each x in the structure.
dbg(pdhf(vbig, [&](auto x){return x.second;}, 5)); // pdhf =
"print_details_helper_func"

dbgcbold("done!");
return;
}

```

Summary of key dbg() features

- `dbg(a, b, c)` prints line number, exact text you passed in, and the values of all those variables/expressions/objects. Optional `dbgC` version uses the first argument as a marker or comment, which is placed to the left. All printouts are indented to keep them visually separate from the normal outputs to cout.
- Different color options so you can easily see which printouts came from which `dbg()` call without comparing line numbers every time. Default color is cyan if you just use `dbg()`; specify other colors with `dbgP` (purple), `dbgY` (yellow), and so on. All the ANSI terminal colors are provided except black; see the template code for the full list. `dbgBold` provides a special colorset with a bright red background.
- Handles pairs, tuples, and STL data structures including arbitrary nesting.
- All `dbg()` and `el` stuff is removed at preprocessor stage if we didn't pass flag `-DDCCLYDE_LOCAL` when compiling. That means I don't need to bother commenting out these lines before submitting to online judges. (That's a big upgrade from regular `cerr` printouts, which won't cause WA at most judges but can easily cause TLE.)
- `pdh` ("print details helper") functionality prints one entry per line; useful if we're printing a complex object since printing on one line can be hard to visually process. Template includes some extensions, demod above.

Real-life example

The demo above may make it seem like the different colors are just visually noisy instead of useful. Here's an example of how I'd use the printouts in "real life" on a recent div2 C. (OK, in a real competition I wouldn't use this many comments or printouts unless the implementation

was more complicated, but I want to make the example understandable by everyone who sees this blog.)

[Submission link](#)

Just the solve() function

```
/**
    Idea: After the very first step, it should always be possible to remove
    exactly one 1 per step. So the plan is:
    1) Choose the first step to remove smallest possible number of 1s
    2) Answer will be 1 + num ones remaining after the first step
*/

void solve() {
    ll s(R, C);
    V<string> dat; rv(R, dat);
    dbgR(MP(R,C), dat); // I usually dbg() the inputs so I can confirm I
    read them correctly.
    el;

    // Step 1: Find smallest possible number of 1s deleted in first step.
    ll fewest_deleted = 3;
    FOR(r, 0, R-1) FOR(c, 0, C-1) {
        ll ones_in_2x2 = 0;
        FOR(j, 0, 2) FOR(k, 0, 2) {
            if (dat[r + j][c + k] == '1') {++ones_in_2x2;}
        }
        ll ones_deleted_here = max(ones_in_2x2 - 1, 0LL);
        ckmin(fewest_deleted, ones_deleted_here);
        dbg(MP(r, c), ones_in_2x2, "", ones_deleted_here, fewest_deleted);
    }

    ll ones_total = 0;
    FOR(r, 0, R) FOR(c, 0, C) {
        if (dat[r][c] == '1') {++ones_total;}
    }

    ll out;
    if (fewest_deleted == 0) {
        // It's initially possible to pick an L shape that doesn't hit any
        1s.

        // That isn't a legal move though.
        out = ones_total;
    }
}
```

```

    } else {
        out = 1 + (ones_total - fewest_deleted);
    }
    dbgY(fewest_deleted, ones_total, out);
    ps(out);
    return;
}

```

Here's what I see when I run the sample test cases locally:

```

loading num cases!!! 865 : [ ] :

CASE 873 : [ CASE ] : 1
824 : [ MP(R,C), dat ] : (4, 3) {"101", "111", "011", "110"}

836 : [ MP(r, c), ones_in_2x2, "", ones_deleted_here, fewest_deleted ] : (0, 0) 3 "" 2 2
836 : [ MP(r, c), ones_in_2x2, "", ones_deleted_here, fewest_deleted ] : (0, 1) 3 "" 2 2
836 : [ MP(r, c), ones_in_2x2, "", ones_deleted_here, fewest_deleted ] : (1, 0) 3 "" 2 2
836 : [ MP(r, c), ones_in_2x2, "", ones_deleted_here, fewest_deleted ] : (1, 1) 4 "" 3 2
836 : [ MP(r, c), ones_in_2x2, "", ones_deleted_here, fewest_deleted ] : (2, 0) 3 "" 2 2
836 : [ MP(r, c), ones_in_2x2, "", ones_deleted_here, fewest_deleted ] : (2, 1) 3 "" 2 2
852 : [ fewest_deleted, ones_total, out ] : 2 9 8

8

CASE 873 : [ CASE ] : 2
824 : [ MP(R,C), dat ] : (3, 4) {"1110", "0111", "0111"}

836 : [ MP(r, c), ones_in_2x2, "", ones_deleted_here, fewest_deleted ] : (0, 0) 3 "" 2 2
836 : [ MP(r, c), ones_in_2x2, "", ones_deleted_here, fewest_deleted ] : (0, 1) 4 "" 3 2
836 : [ MP(r, c), ones_in_2x2, "", ones_deleted_here, fewest_deleted ] : (0, 2) 3 "" 2 2
836 : [ MP(r, c), ones_in_2x2, "", ones_deleted_here, fewest_deleted ] : (1, 0) 2 "" 1 1
836 : [ MP(r, c), ones_in_2x2, "", ones_deleted_here, fewest_deleted ] : (1, 1) 4 "" 3 1
836 : [ MP(r, c), ones_in_2x2, "", ones_deleted_here, fewest_deleted ] : (1, 2) 4 "" 3 1
852 : [ fewest_deleted, ones_total, out ] : 1 9 9

9

CASE 873 : [ CASE ] : 3
824 : [ MP(R,C), dat ] : (2, 2) {"00", "00"}

836 : [ MP(r, c), ones_in_2x2, "", ones_deleted_here, fewest_deleted ] : (0, 0) 0 "" 0 0
852 : [ fewest_deleted, ones_total, out ] : 0 0 0

0

CASE 873 : [ CASE ] : 4
824 : [ MP(R,C), dat ] : (2, 2) {"11", "11"}

836 : [ MP(r, c), ones_in_2x2, "", ones_deleted_here, fewest_deleted ] : (0, 0) 4 "" 3 3
852 : [ fewest_deleted, ones_total, out ] : 3 4 2

2

880 : [ TIME() ] : 0.000984

```

For short solutions, I normally just use random different colors for each printout. In more complicated programs I try to somewhat organize my color usage. For example, I sometimes use: Red = print out inputs as soon as I read them to confirm I loaded them correctly; Yellow = higher-level info, maybe printing out results of a big intermediate step or printing status once per outer loop cycle; Cyan (default) = less important stuff, or printouts coming from inside a work loop; Purple = printouts from inside a function call or nested loop.

"Unhackable" and faster unordered_map and unordered_set replacements

`unordered_map` and `unordered_set` are notoriously hackable and so generally should not be used on Codeforces; see [classic blog post](#). Also, they are usually around 5x slower than the alternative `gp_hash_table`; see [another blog](#). My template defines replacements `umap` and `uset` which fix both of those problems, basically by implementing the recommendations from those linked posts. Bonus: my versions also happily hash arbitrary pairs, tuples, or iterable STL objects. I make it easy to toggle which implementation is used by `umap` and `uset`. Caveat: pairs and tuples work well, but hashing more complex stuff like

`std::vector` s can often lead to TLE.

Precompiled header

When compiling on judge server, the template uses a few includes:

```
#include <tgmath.h>
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
```

When compiling locally, the template instead just uses `#include "/home/dccllyde/puzzles/code/templates/superheader.h"`. That's because `superheader.h` is a local file that has all those same includes, except I've set up a precompiled header which makes my compilations several seconds faster. The difference is approximately 6s -> 1.5s on my older laptop, but only 2.5s -> 1.0s on my current machine. Basically, depending on your computer this hack might be pretty important for keeping the compile time manageable despite all the template's `#includes` and `#defines`.

I'm not recommending a specific guide for how to build a precompiled header because the correct steps may depend on your OS and machine configuration. However, I was able to make it work pretty quickly once I searched Google for `precompiled header C++ gcc ubuntu`, so I hope you'll have good luck here too.

I/O shortcuts

These aren't a huge deal, but they do save me from typing the same annoying lines of code in every problem.

```
// INPUT
lls(R, C); // declare and read from stdin two Long Longs called R and C. Can
also use ints, chars, strings to read those other datatypes.
ll1(a, b); // same as lls except decrement the variables by 1 after reading;
useful when problems give 1-indexed inputs but I want to work with 0-indexed
V<int> q;
rv(R, q); // resize q to size R and then fill all entries from stdin
rv1(R, q); // same as rv except decrement by 1 after reading

// OUTPUT
ps(a, b, R); // same as cout << a << ' ' << b << ' ' << R << '\n';
ps1(a, b); // same as cout << a+1 << ' ' << b+1 << '\n'; notice we ADD 1 when
outputting.
pv(q); // output space-separated all on one line, with newline at the end
pv1(q); // same as pv but add 1 to each output
```

Compilation and usage notes

Sample compile+run command: `g++ --std=c++20 -DDCCLYDE_LOCAL template.cpp && ./a.out`

The template requires C++17 or higher. It also works better on 64-bit systems, just because lately I use `long long` instead of `int` by default due to the negligible speed difference on modern machines including online judges.

I submit to Codeforces using `GNU C++20 (64)` and AtCoder using `C++ (GCC 9.2.1)`.

The template will be nicer to use if your editor has a shortcut for code folding between `#pragma region` and `#pragma endregion`. I use VS Code and I have keyboard shortcuts bound for "Fold All Regions" and "Unfold All Regions". I also use the "Better Comments" extension, which explains why some of the comments have special flags at the start (they show up in different colors on my screen).

What a good template can (and can't) do for you

The main things I think a template can help with are:

1. Reduce repetitive typing: save 1-5 minutes on every problem by using small prewritten functions/macros for I/O, binary searches, etc. On CF this is pretty important, since getting problems A and B in 1-3 mins instead of 5-10 minutes helps my rating a lot. Less code typed for each problem also means less chances for typos, so it becomes a little easier to code correctly.
2. Make debugging faster and easier: even when the intermediate steps are producing complicated data structures, the `dbg()` functionality lets me track down unexpected/buggy behaviour very fast. Before I had this template I used to code new debug printouts on every problem, and coding the printouts would distract me from focusing fully on thinking through my logic to actually find the mistake.

Overall, the template makes it way easier and faster for me to go from "having the right idea" to "getting AC". I always feel frustrated if I think of the correct plan but don't end up getting AC. My template makes the implementation process much smoother which helps me feel motivated and confident during contests and training, especially with new ideas/algorithms that I'm not very comfortable with.

Unfortunately, a template can't help you have the right solution ideas in the first place. For that you can instead read the thousands of blogs that already exist with more general advice about how to train ;)

I've read some CF blogs where strong coders say that a good coder shouldn't rely on a long template. I also see a very wide variety in template lengths even among LGMs. Overall, I think template size can just be personal preference, and you should experiment to see what feels best to you.

Special thanks

When creating my template, I started from **Benq**'s TemplateLong.cpp ([GitHub](#)). My version is very different, but I still use lots of minor convenience features from his template.

The dbg() functionality is my heavily modified version of ideas stolen from **tourist** and [liv_1n9_w08](#), see [blog](#).

Of course I've taken ideas or code from a ton of other sources too, mostly other CF blogs. Thanks everyone!

Questions, suggestions, advice?

I'm very interested to hear to advice about how to accomplish the same things in a better way, or feature requests that would make the template easier to use or more helpful, or even just random questions about why I did things a certain way. Some of my "weird" decisions are made on purpose to get around non-obvious issues that I didn't discuss in this blog, and others may be just because I'm clumsy with some of the tools I'm using here, so even asking "silly" questions can be useful — either I'll explain something you didn't see, or I'll learn something from your idea.

This is my first blog post after using Codeforces off and on for 6+ years, so I'm also interested to hear advice about how to make a more useful blog next time.

🔗 template, c++ template, debugging, preprocessor

▲ +68 ▼ ☆

👤 [dccllyde](#)

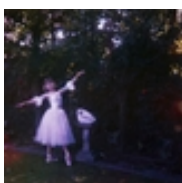
📅 5 months ago

💬 [12](#)



Comments (12)

[Write comment?](#)



oversolver

5 months ago, # | ☆

rip compilation time

→ [Reply](#)

▲ +34 ▼

5 months ago, # ^ | ☆

▲ +10 ▼

Thanks, that's a good concern to raise. I ran some compile time tests on my laptop and got:

- Base template.cpp: 0.421s
- C.cpp (the example I used for debug printouts, see [submission](#)): 0.440s
- C_halftemplate.cpp (same example rewritten to use the "minimal version that adds only the dbg() functionality", see [submission](#)): 0.315s
- C_notemplate.cpp (same example rewritten so it doesn't use the template at all, see [submission](#)): 0.208s

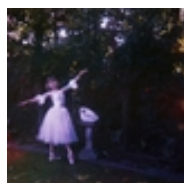


dccllyde

I got those numbers by running each command 10x and averaging the timing results, like `time for k in {1..10}; do g++ --std=c++20 -DDCCLYDE_LOCAL template.cpp; done;`. Note these are somewhat faster than the times I mentioned in the blog above, because those times were measured using my normal compile script which adds lots of slow g++ compile flags.

Overall you're right that the fancy template stuff increases compile time, but the difference isn't big enough to make me really care. Still, if someone does see slower compile times on their computer, the results with `C_halftemplate.cpp` show that it should be possible to at least use the fancy `dbg()` setup while causing a much smaller compile time increase. Alternatively, if someone wants to try using most of the template's functionality but still needs to reduce compile time, the first thing I'd recommend would be to move the `umap` and `uset` stuff into a separate file and copy-paste in only when it's needed, since `gp_hash_table` requires some hefty includes.

→ [Reply](#)



oversolver

5 months ago, # ^ | ☆

▲ +11 ▼

Maybe it is hardware factor, but in my PC this template compiles more than 2 seconds while my usual code in less than 1 second, and I use precompiled stdc++.h

→ [Reply](#)

5 months ago, # ^ | ☆

▲ 0 ▼



dccllyde

If you get a chance, could you try compiling the [current version](#) and let me know how long it takes? I just pushed a [commit](#) that will probably help your compile time a lot. I figured out that your precompiled header probably wasn't getting picked up because I needed to make stdc++.h the first `#include`.

After this change, people who have precompiled stdc++.h should get that speedup. Precompiling all includes at once using superheader.h gives a small extra boost but not a huge deal. If someone continues seeing a big compile time difference, I'm interested to see the output of your compile command with flags `-H` and `-Winvalid-pch` (or other equivalents if you don't use g++) so I can learn how to make this more portable.

→ [Reply](#)

5 months ago, <#> | ☆

← Rev. 2

▲ +27 ▼

```
template<class A,class B,class C> void re(tuple<A,B,C>& t) {auto& [a,b,c]=t;
re(a,b,c);}
```

```
template<class A,class B,class C, class D> void re(tuple<A,B,C,D>& t)
{auto& [a,b,c,d]=t; re(a,b,c,d);}
```



CountZero

why so serious?

```
void re(auto &...args) { ((cin >> args), ...); }
```

```
template <class... Ts> void read(tuple<Ts...> &t) {
    std::apply([](Ts &...args) { re(args...); }, t);
}
```

this works for any tuple.

→ [Reply](#)

5 months ago, <#> [^](#) | ☆

← Rev. 2

▲ 0 ▼

Thanks, that's really useful! These tricks can clean up several other parts of the template too — anyplace where I used `is_tuplelike`. I'm going to update all of those and then update the blog.

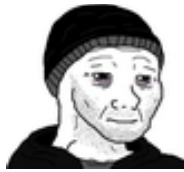


dccllyde

Update: I made those changes and pushed to GitHub, so the link in the blog will now give the new version.

For other people's reference, the trick in the above comment's `re()` function is called a "fold expression". That and `std::apply` were both totally new to me, so thanks again for teaching me these cool tricks!

→ [Reply](#)



Absurd_

5 months ago, # | ☆

▲ +11 ▼

I used to have a big template. But now I refrained from it. No regrets.

→ [Reply](#)

5 months ago, # | ☆

← Rev. 2

▲ +41 ▼

If you're using C++20, you could try using [concepts](#). Something like:

Concepts

```

namespace concepts {
    template <typename T>
    concept is_readable = requires(T x) { std::cin >> x; };

    template <typename T>
    concept is_writable = requires(T x) { std::cout << x; };

    template <typename T>
    concept is_iterable = requires(T x) { x.begin(); x.end(); };

    template <typename T>
    concept is_tuple_like = requires(T x) { typename
std::tuple_size<T>::type; };
}

```

Reading

```

namespace reading {
    template <class... Ts> void re(Ts&... ts);

    template <concepts::is_readable T> void re(T& t) { std::cin >>
t; }

    template <typename T>
    requires(!concepts::is_readable<T> && concepts::is_iterable<T>)
    void re(T& t);

    template <typename T>
    requires(!concepts::is_readable<T> && !concepts::is_iterable<T>
&& concepts::is_tuple_like<T>)
    void re(T& t) { apply([&](auto&... args) { (re(args), ...); },
t); }

    template <typename T>
    requires(!concepts::is_readable<T> && concepts::is_iterable<T>)

```



the_hyp0cr1t3

```

    void re(T& t) { for(auto& x: t) re(x); }

    template <class... Ts> void re(Ts&... ts) { (re(ts), ...); }
}

```

Writing

```

#define sep ", "
#define tr(...) cout << "(" << __LINE__ << ")[#"__VA_ARGS__ "]:["
pr(__VA_ARGS__), cout << "]\n"

namespace writing {
    template <class T, class... Ts>
    void pr(const T& t, const Ts&... ts);

    template <concepts::is_writable T> void pr(const T& t) {
std::cout << t; }

    template <typename T>
    requires(!concepts::is_writable<T> && concepts::is_iterable<T>)
    void pr(const T& t);

    template <typename T>
    requires(!concepts::is_writable<T> && !concepts::is_iterable<T>
&& concepts::is_tuple_like<T>)
    void pr(const T& t) {
        std::cout << '(';
        apply([&](const auto&... args) {
            bool fst = true;
            ((fst ? (fst = false, pr(args)) : (std::cout << sep,
pr(args)))), ...);
        }, t);
        std::cout << ')';
    }

    template <typename T>
    requires(!concepts::is_writable<T> && concepts::is_iterable<T>)
    void pr(const T& t) {
        std::cout << '{';
        for(bool fst = true; const auto& x: t)
            std::cout << (fst? fst = false, "" : sep), pr(x);
        std::cout << '}';
    }

    template <class T, class... Ts>
    void pr(const T& t, const Ts&... ts) {

```

```

        pr(t), ((std::cout << sep, pr(ts)), ...);
    }

    void ps() { std::cout << std::endl; }

    template <class... Ts>
    void ps(const Ts&... ts) { pr(ts...), ps(); }
}

```

→ [Reply](#)

5 months ago, # ^ | ☆

▲ +14 ▼



dccllyde

Thanks for the tip! I do compile with `--c++20` on my own machine, but I don't want to make the template incompatible with C++17 quite yet. AtCoder and CodeJam don't have C++20 yet, and also I sometimes get stuck coding on a different machine that doesn't have C++20 access locally. Concepts are on my list of improvements to make once I fully switch to C++20 though.

→ [Reply](#)



Irvideckis

5 months ago, # ^ | ☆

▲ +25 ▼

It's a shame this is still an issue in 2022

→ [Reply](#)



dccllyde

5 months ago, # | ☆

▲ 0 ▼

Auto comment: topic has been updated by dccllyde (previous revision, new revision, compare).

→ [Reply](#)

5 months ago, # | ☆

▲ +8 ▼



Manan_shah

Nice blog!

Pretty similar to my blog where I had written about debugging, inputs/outputs and other cool stuff:

[getting started with CP](#)

→ [Reply](#)

Desktop version, switch to [mobile version](#).
[Privacy Policy](#)

Supported by



ITMO UNIVERSITY