



Debugging

- Viraj Chandra



Goal

To understand:

- What is Debugging?
- Efficient Ways
- Flag in Online Judge



What is Debugging?

Process of finding and fixing errors (bugs) in a program to ensure it runs correctly.

Types:

- ✓ • Syntax Errors – Missing semicolons, incorrect brackets
- ✓ • Runtime Errors – Division by zero, Out-of-bounds
- ✓ • Logical Errors – Flaws in the logic
- ✓ • Time Limit Exceeded
- ✓ • Wrong Answer
- ✓ • Memory Limit Exceeded



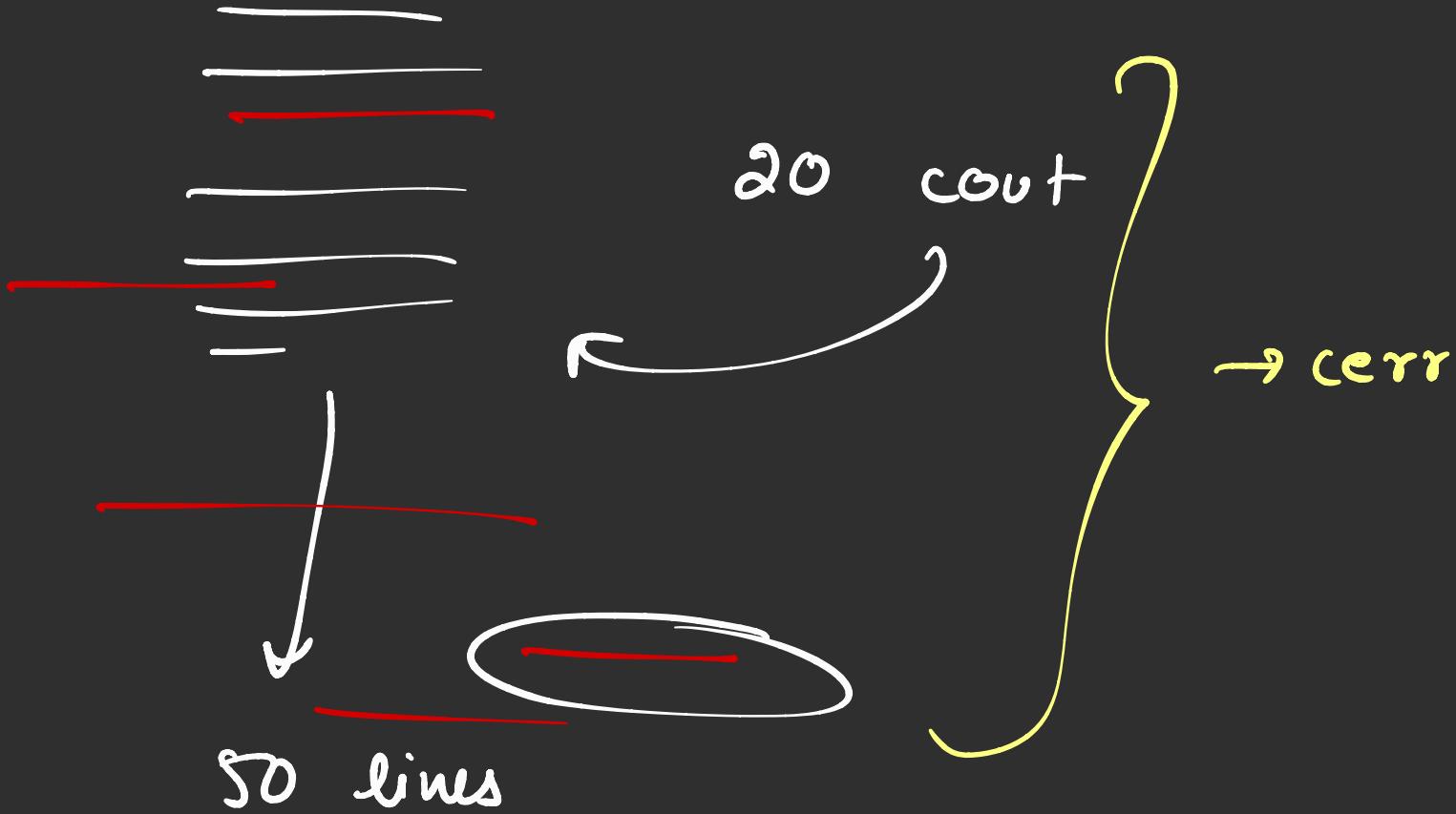
Bug
↓

papers

Scientists → Debugging

- Pen Paper
- add cout
- print values
- dry run
- place checkers

- time taking
- waiting
- remove cout/flag
/checker.



✓ `cout << ans << endl;`



Newbie Way!

Write print statements.

Examples:

- ✓ • cout << arr << endl; (everywhere)
- ✓ • cout << will_cp_rating_increase << endl; (everytime)

xD



Newbie Way!

Problems?

- ✗ • Remove cout << arr << endl; (everywhere)
- ✓ • Remove cout << will_cp_rating_increase << endl; (everytime)

Longer code implementation == More pain



↗ **console**

Efficient Way 1 - “cerr”

“cerr” is an unbuffered standard error stream in C++. Unlike cout, it does not require std::endl to flush.

Useful for debugging since output from **cerr does not interfere** with the standard output (cout) expected by the online judge.

Let us understand with code. }



```
for ( auto it : mp )  
    cerr << it.first << ...
```

→ template ?? →



Efficient Way 1 - “cerr”

CPH JUDGE: RESULTS **CPU**

Local: example

^ TC 1 Failed 453ms

Input: Copy

Expected Output: Copy

Received Output: Set Copy

Standard Error: 1 2 3 4 5

C++ example.cpp M X

```
C++ example.cpp > ⚙ main()
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main()
5 {
6     vector<int> v = {1, 2, 3, 4, 5};
7     for (auto it : v)
8         cerr << it << " ";
9     return 0;
10}
11
```

A screenshot of a judge interface showing a failed submission. A blue arrow points from the 'CPU' label in the top navigation bar to the 'Failed' status under 'TC 1'. A red vertical line highlights the 'Received Output' section, where the output '1 2 3 4 5' is shown. A red arrow points from the bottom of this section to the standard error output at the bottom of the page. The code editor on the right shows a C++ program that prints the numbers 1 through 5 to standard error using the `cerr` stream.



Efficient Way 2 - “cerr” with power

✓ “**cerr**” seems useful, but still has pain because writing **cerr** for larger implementations still exists.

Time for template? **YES** ✓

Let us understand with code.



Efficient Way 2 - “cerr” with power

```
1 #define debug(x) _print(x); cerr << endl;
2 void _print(ll t) {cerr << t;}
3 void _print(int t) {cerr << t;}
4 void _print(string t) {cerr << t;}
5 void _print(char t) {cerr << t;}
6 void _print(double t) {cerr << t;}
7
8 template <class T, class V> void _print(pair <T, V> p);
9 template <class T> void _print(vector <T> v);
10 template <class T> void _print(set <T> v);
11 template <class T> void _print(multiset <T> v);
12 template <class T, class V> void _print(pair <T, V> p) {cerr << "{"; _print(p.f); cerr << ","; _print(p.s); cerr << "}"}
13 template <class T> void _print(vector <T> v) {cerr << "[ "; for (T i : v) {_print(i); cerr << " ";} cerr << "]";}
14 template <class T> void _print(set <T> v) {cerr << "[ "; for (T i : v) {_print(i); cerr << " ";} cerr << "]";}
15 template <class T> void _print(multiset <T> v) {cerr << "[ "; for (T i : v) {_print(i); cerr << " ";} cerr << "]";}
16 template <class T, class V> void _print(map <T, V> v) {cerr << "[ "; for (auto i : v) {_print(i); cerr << " ";} cerr << "]";}
```



cout



cerr

cerr << arr << endl;

TC

TLE

debug(x)



CF X

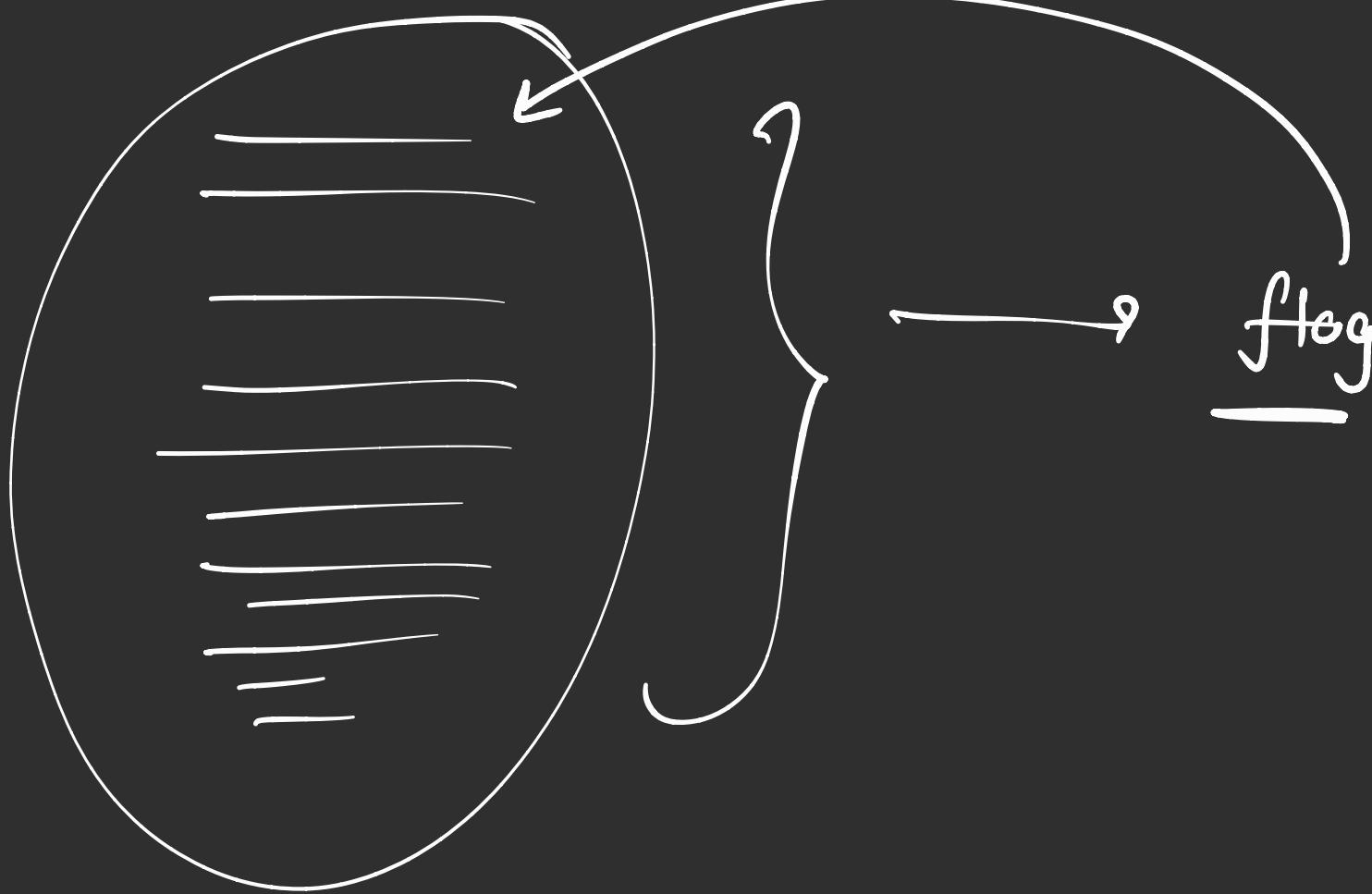
local



g++ compile -o fin.exe

↳ flags

-D



Local

fig -> khnh k coding karlo }

→ debug(x) →
points

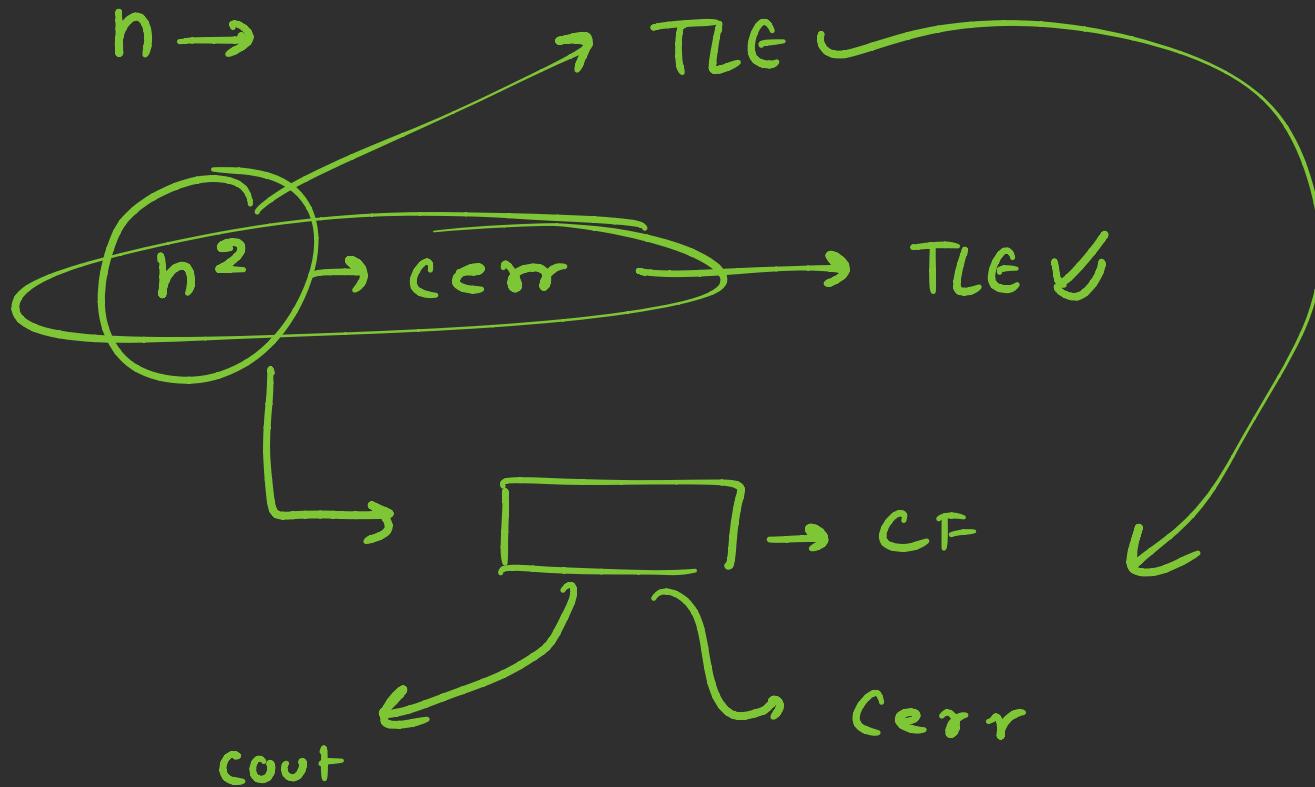
points y

CF Server

fig → ' x '

→ debug(x) → X

X





Efficient Way 3 - “cerr” with flag

“cerr” now even better, but still has added complexity of printing useless lines.

Can we do even better?

Let us understand with code.



Efficient Way 3 - “cerr” with flag



```
1 // Debug Overloads
2 #ifdef khnhcodingkarlo
3 #define debug(x) _print(x); cerr << endl;
4 #else
5 #define debug(x)
6 #endif
7
8 void _print(ll t) {cerr << t;}
9 void _print(int t) {cerr << t;}
10 void _print(string t) {cerr << t;}
11 void _print(char t) {cerr << t;}
12 void _print(double t) {cerr << t;}
13
14 template <class T, class V> void _print(pair <T, V> p);
15 template <class T> void _print(vector <T> v);
16 template <class T> void _print(set <T> v);
17 template <class T> void _print(multiset <T> v);
18 template <class T, class V> void _print(pair <T, V> p) {cerr << "{"; _print(p.f); cerr << ","; _print(p.s); cerr << "}";}
19 template <class T> void _print(vector <T> v) {cerr << "[ "; for (T i : v) {_print(i); cerr << " ";} cerr << "]";}
20 template <class T> void _print(set <T> v) {cerr << "[ "; for (T i : v) {_print(i); cerr << " ";} cerr << "]";}
21 template <class T> void _print(multiset <T> v) {cerr << "[ "; for (T i : v) {_print(i); cerr << " ";} cerr << "]";}
22 template <class T, class V> void _print(map <T, V> v) {cerr << "[ "; for (auto i : v) {_print(i); cerr << " ";} cerr << "]";}
23
```



Efficient Way 3 - “cerr” with flag

The screenshot shows the Code settings search interface. The search bar at the top contains the text "cph cpp". To the right of the search bar, a button displays "4 Settings Found" next to a filter icon. Below the search bar, there are two tabs: "User" and "Workspace", with "User" being the active tab. To the right of the tabs, it says "Last synced: 0 secs ago". The main content area has a dark background with light-colored text. It displays a result under the heading "Cph > Language > Cpp: Args". The description below the heading reads: "Space seperated additional flags passed to g++ (for C++ compiling your file. Example '-Wmaybe-uninitialized -std=c++20 -O2 -Dkhnhcodingkarlo'". A red underline highlights the word "khnhcodingkarlo" in the example flags.