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

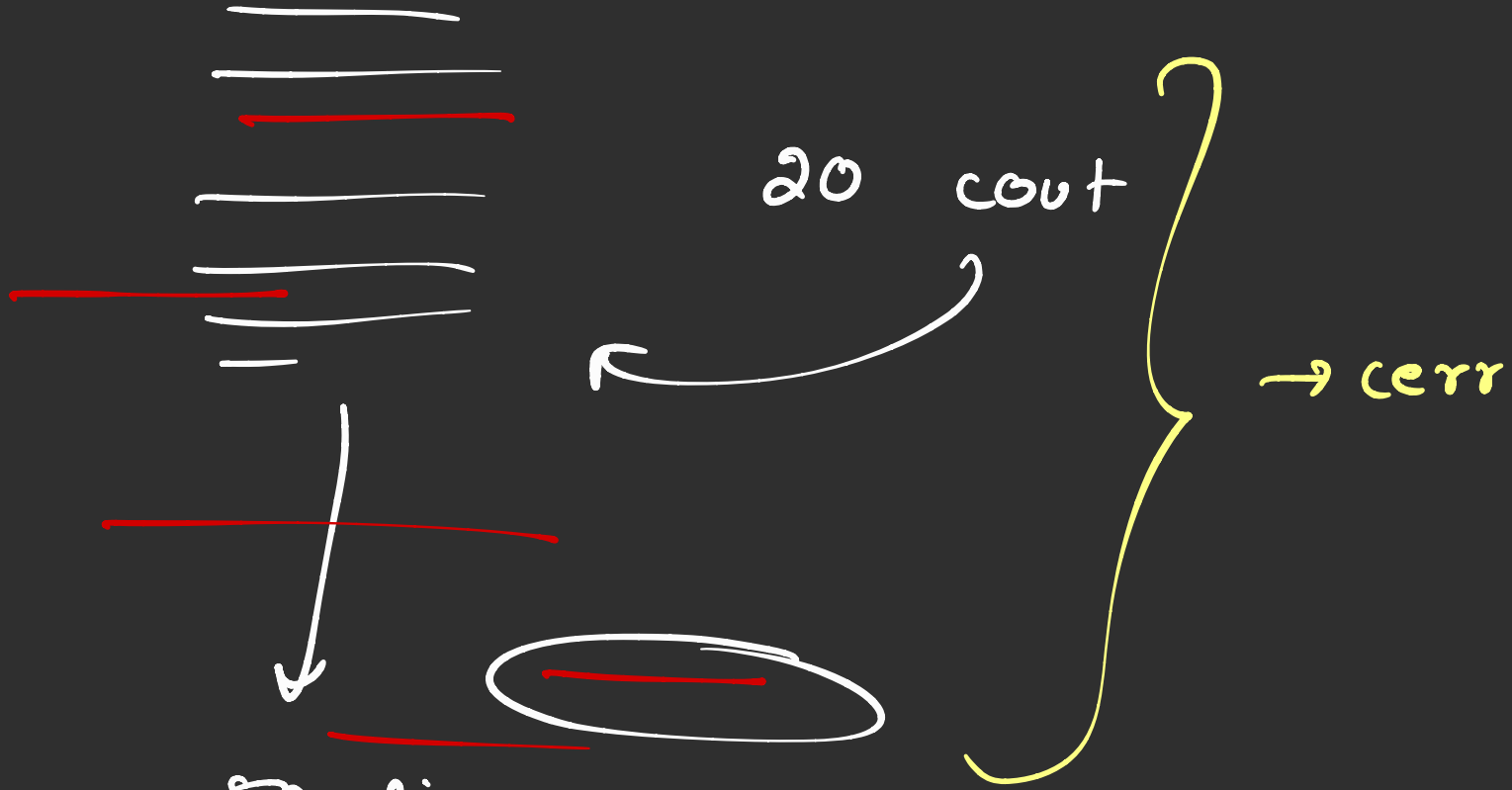
→ dry run

→ place checkers

→ time taking

→ writing

→ remove cout/flag /checker.



✓ cout << ons << 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

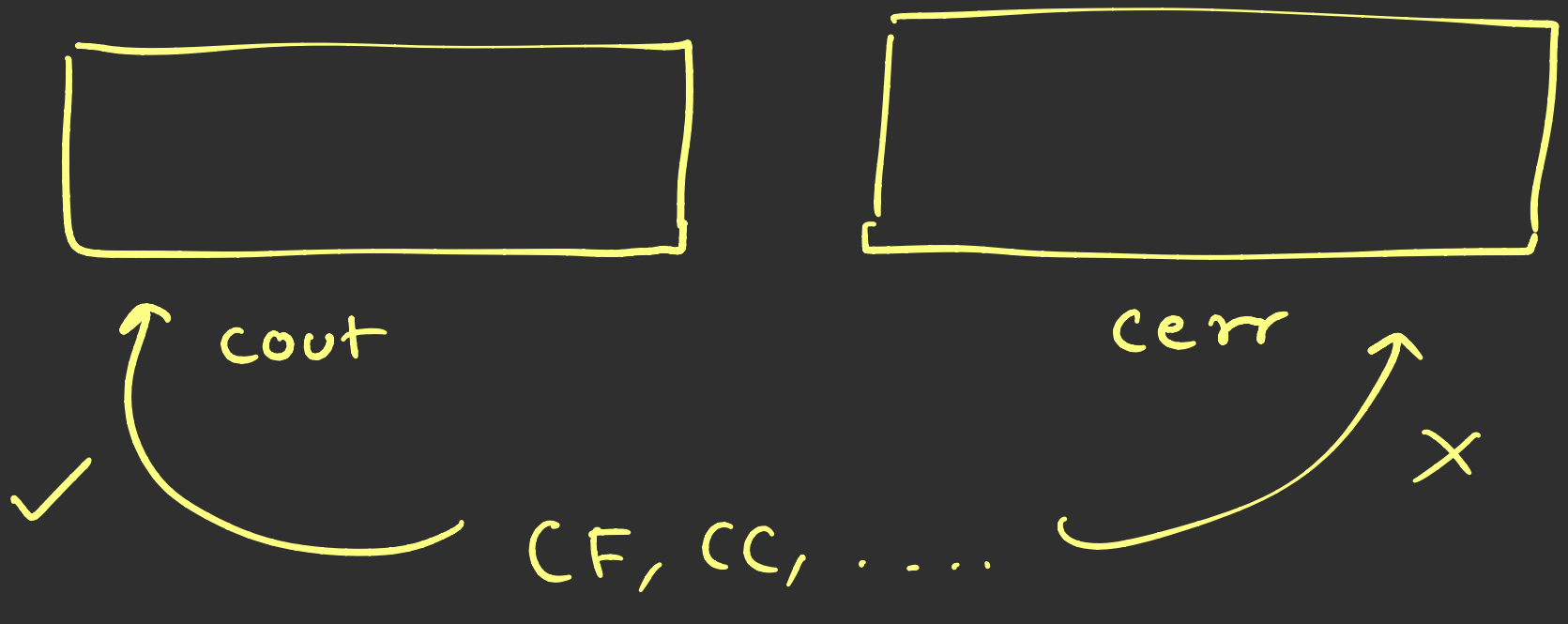
console

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.

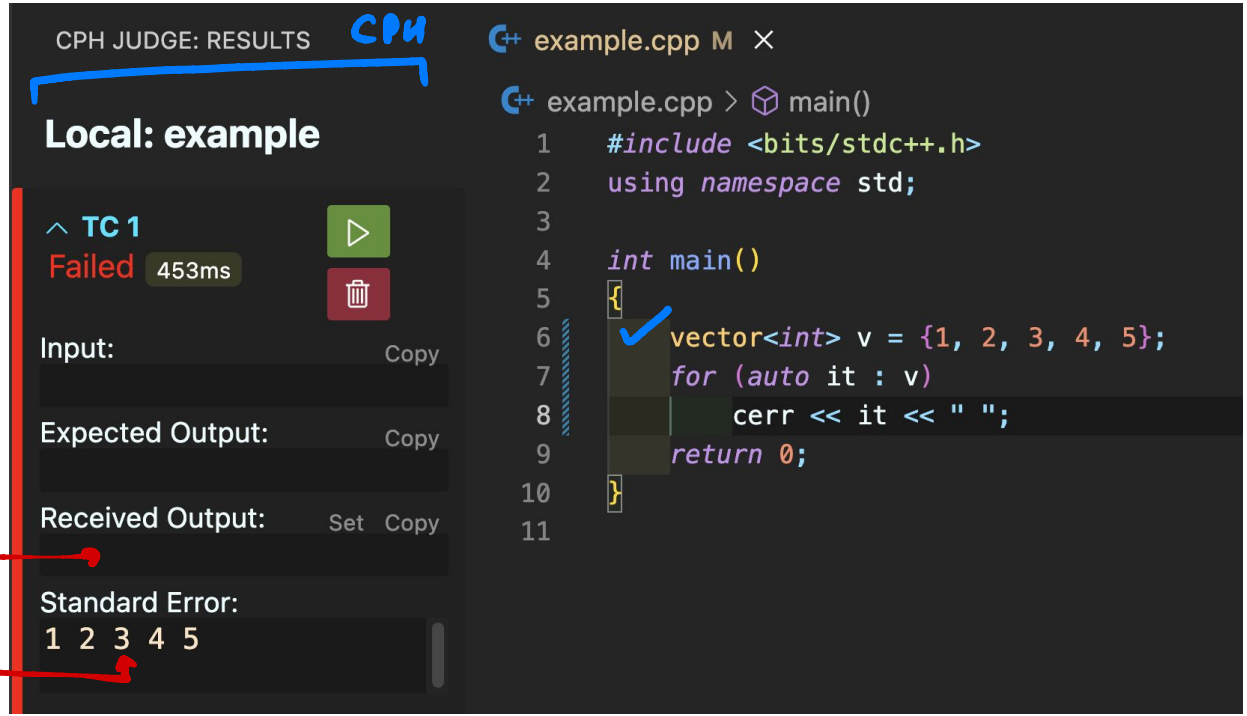
Let us understand with code.



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

→ template ?? →

Efficient Way 1 - “cerr”



Efficient Way 2 - “cerr” with power

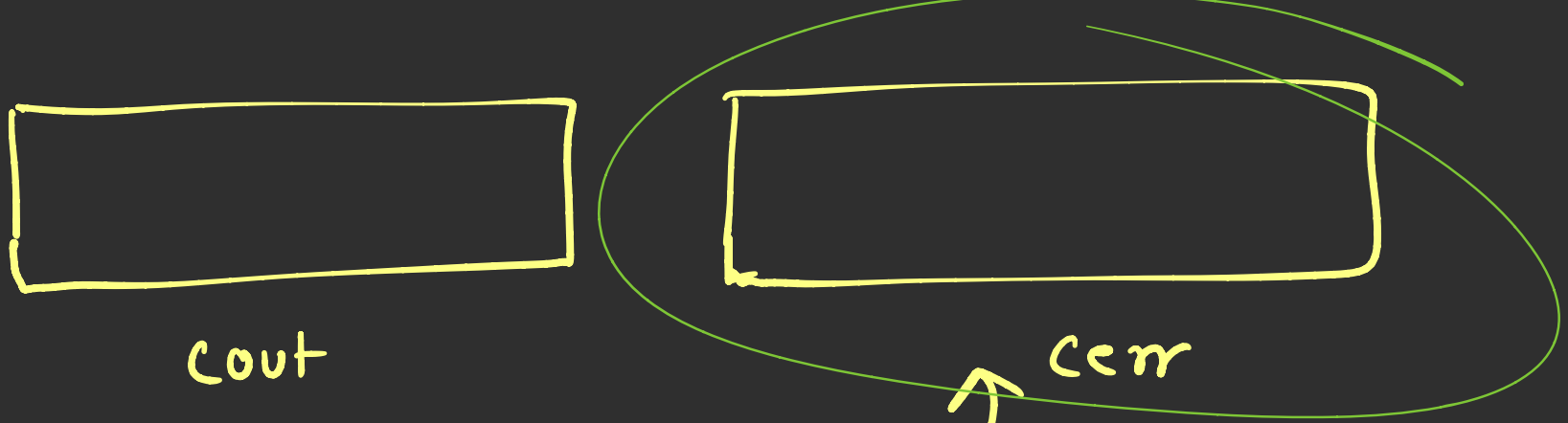
“cerr” seems useful, but still has pain because writing cerr for larger implementations still exists.

Time for template? **YES**

Let us understand with code.

Efficient Way 2 - “cerr” with power

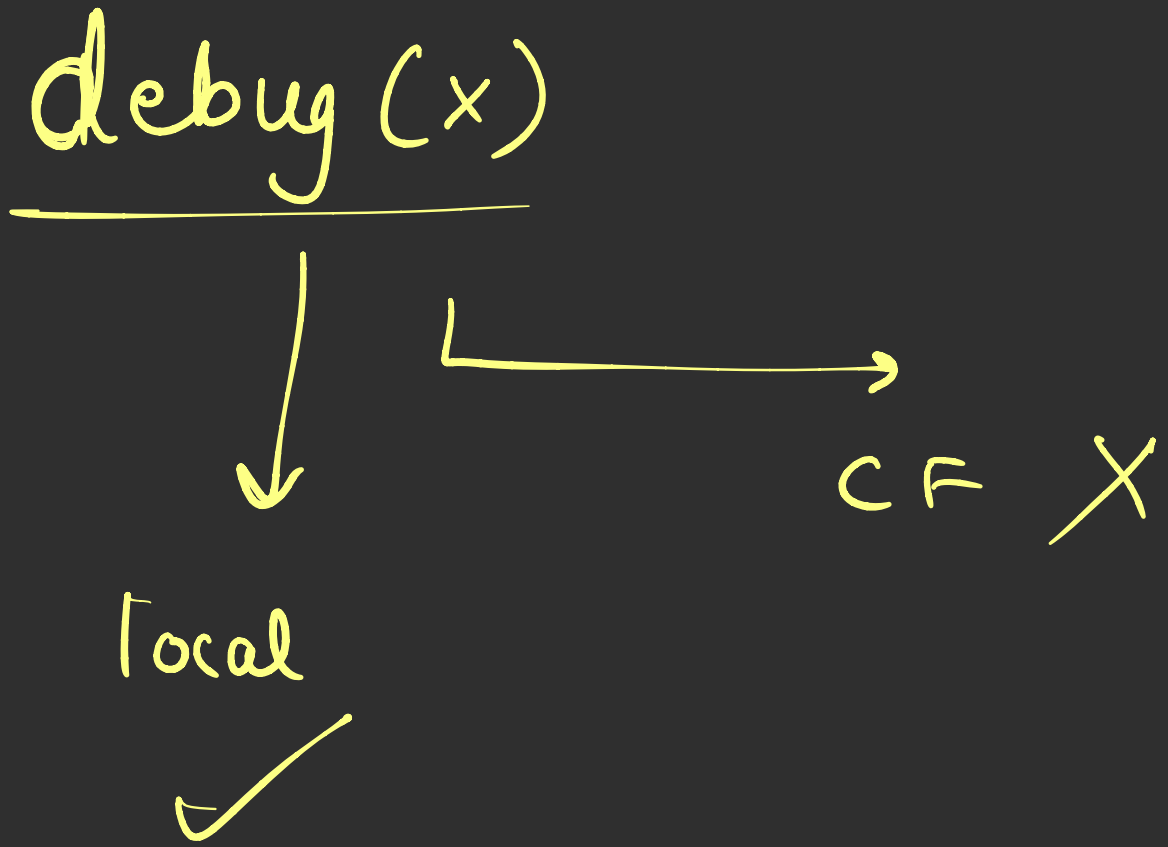
```cpp
#define debug(x) _print(x); cerr << endl;
void _print(ll t) {cerr << t;}
void _print(int t) {cerr << t;}
void _print(string t) {cerr << t;}
void _print(char t) {cerr << t;}
void _print(double t) {cerr << t;}

template <class T, class V> void _print(pair <T, V> p);
template <class T> void _print(vector <T> v);
template <class T> void _print(set <T> v);
template <class T> void _print(multiset <T> v);
template <class T, class V> void _print(pair <T, V> p) {cerr << "{"; _print(p.f); cerr << ","; _print(p.s); cerr << "}";}
template <class T> void _print(vector <T> v) {cerr << "[ "; for (T i : v) {_print(i); cerr << " ";} cerr << "]";}
template <class T> void _print(set <T> v) {cerr << "[ "; for (T i : v) {_print(i); cerr << " ";} cerr << "]";}
template <class T> void _print(multiset <T> v) {cerr << "[ "; for (T i : v) {_print(i); cerr << " ";} cerr << "]";}
template <class T, class V> void _print(map <T, V> v) {cerr << "[ "; for (auto i : v) {_print(i); cerr << " ";} cerr << "]";}
```



TC

TLE

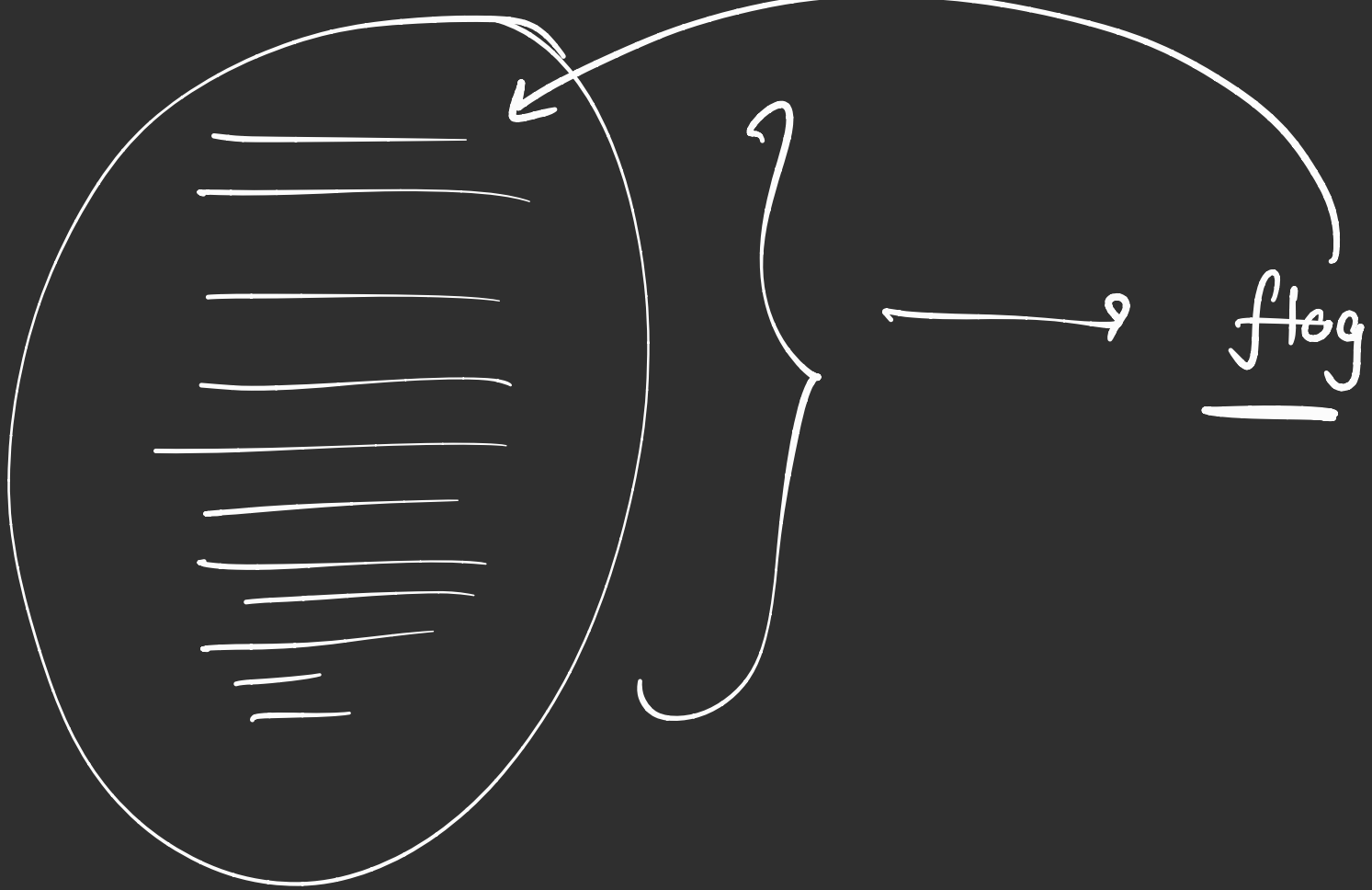
cerr << arr << endl;



g++ compile -o fin.exe

↳ flags

-D_____



Local

fig -> khmh kcodingkarlo

→ debug(x) →

prints

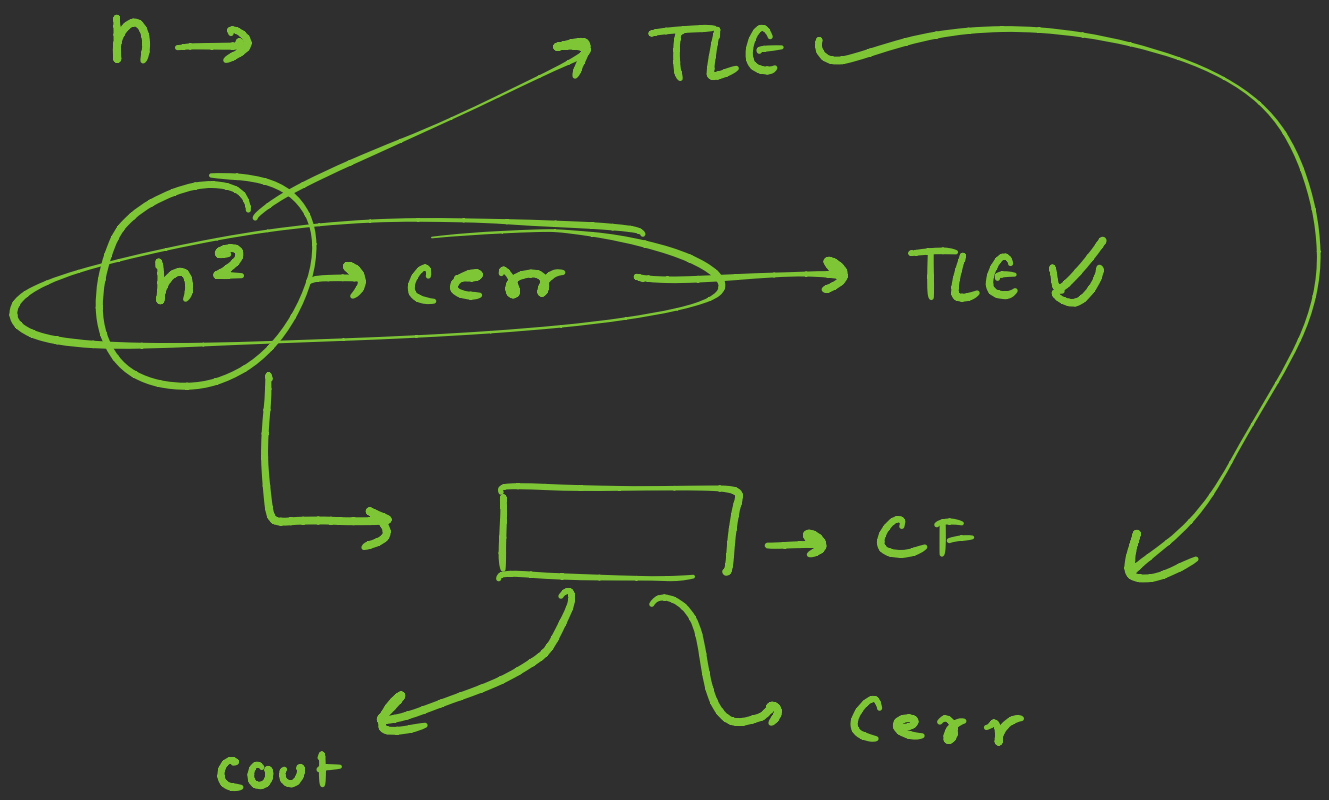
prints ✓

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

```cpp
// Debug Overloads
#ifdef khnhcodingkarlo
#define debug(x) _print(x); cerr << endl;
#else
#define debug(x)
#endif

void _print(ll t) {cerr << t;}
void _print(int t) {cerr << t;}
void _print(string t) {cerr << t;}
void _print(char t) {cerr << t;}
void _print(double t) {cerr << t;}

template <class T, class V> void _print(pair <T, V> p);
template <class T> void _print(vector <T> v);
template <class T> void _print(set <T> v);
template <class T> void _print(multiset <T> v);
template <class T, class V> void _print(pair <T, V> p) {cerr << "{"; _print(p.f); cerr << ","; _print(p.s); cerr << "}";}
template <class T> void _print(vector <T> v) {cerr << "[ "; for (T i : v) {_print(i); cerr << " ";} cerr << "]";}
template <class T> void _print(set <T> v) {cerr << "[ "; for (T i : v) {_print(i); cerr << " ";} cerr << "]";}
template <class T> void _print(multiset <T> v) {cerr << "[ "; for (T i : v) {_print(i); cerr << " ";} cerr << "]";}
template <class T, class V> void _print(map <T, V> v) {cerr << "[ "; for (auto i : v) {_print(i); cerr << " ";} cerr << "]";}
```

Efficient Way 3 - “cerr” with flag