1. Project Overview

`yscanf.h` is an open-source and free project developed by *Summer PLUS Studio*, designed to significantly accelerate input operations in C programs.

When used for console input (standard input), `yscanf` is **noticeably faster than the standard library function** `scanf`.
When used for file input via `freopen`, `freopen + yscanf` **is significantly faster than** `freopen + scanf` **or** `fopen + fscanf`.

`yscanf` is a **high-performance input function intended to replace** `scanf` in scenarios where input format is known and performance is critical.

It is implemented on top of a **large fread-based buffer**, which greatly reduces the number of system calls.
This makes it particularly suitable for:

- Competitive programming
- High-performance computing
- Engineering scenarios with strictly defined input formats

Based on preliminary internal testing, for programs processing millions of input values:

- `yscanf` is typically **120–200 ms faster than** `scanf`
- About **50 ms faster than** `getchar`**-based input**
- At the cost of approximately **4 MB of additional memory usage**

All functions related to `yscanf` are prefixed with **`y`**
(e.g. `yscanf`, `ynext_char`, `yskip_space_input`, etc.),
so they can safely coexist with other libraries or multiple header files **without symbol name conflicts or linkage issues**.

EOF Requirement for Standard Input

When using `yscanf` to read from **standard input (`stdin`)**,
the user **must explicitly provide an End-Of-File (`EOF`) signal** after finishing input, so the program can correctly detect the end of input.

`EOF` input methods on different operating systems:

- **Windows**:
  Press Ctrl + Z, then press Enter
- **macOS / Linux**:
  Press Ctrl + D

When the input source is a **text file or file redirection** (e.g. using `freopen`),

the file itself naturally contains an `EOF` marker, and **no additional user action is required**.

If `EOF` is not properly provided in interactive input, yscanf may continue waiting for input, causing the program to appear blocked or unable to terminate.

2. Function Prototype

```
int yscanf(const char *fmt, ...);
```

The return value semantics are similar to `scanf`:

- Returns the number of successfully assigned arguments
- Returns `-1` if an illegal or unsupported format specifier is encountered

3. Supported Format Specifiers

`yscanf` supports **only** the following format specifiers:

- `%d`
  Read an `int`
- `%u`
  Read an `unsigned int`
- `%lld`
  Read a `long long`
- `%llu`
  Read an `unsigned long long`
- `%lf`
  Read a `double` (scientific notation **not supported**)
- `%g`
  Read a `double` (scientific notation supported, e.g. `1e-3`)
- `%s`
  Read a string (whitespace-delimited, null-terminated automatically)
- `%c`
  Read a single `char`

Any format specifier **not listed above** is considered **illegal**, and `yscanf` will immediately return `-1`.

4. Whitespace Handling Rules

The whitespace handling behavior of `yscanf` is designed to be **consistent with standard `scanf`**.

4.1 Whitespace in the format string

Any whitespace character in the format string
(i.e. characters for which `isspace()` returns true)

will cause `yscanf` to skip **any number of whitespace characters** in the input stream.

4.2 %c without leading space

`%c` does not skip whitespace by default.

Example:

```
yscanf("%c",&ch);
```

This may read a space, newline, or tab character.

4.3 " %c" with leading space

A space before %c causes yscanf to skip all whitespace characters before reading the character.

Example:

```
yscanf(" %c",&ch);
```

4.4 Definition of whitespace

Whitespace is determined strictly by isspace() and includes, but is not limited to:

- `' '` (space)
- `'\n'` (newline)
- `'\t'` (tab)
- `'\r','\f','\v'`

---

5. Compatibility with scanf (Important)

`yscanf` is **not a full replacement for** `scanf`.
The following incompatibilities or differences exist:

1. Width specifiers are not supported (e.g. `%5d`)
2. Assignment suppression (*) is not supported
3. Scan sets are not supported (e.g. `%[a-z]`)
4. Special specifiers such as `%p`,`%n`, etc. are not supported
   Since `yscanf` writes directly to memory, users must ensure type correctness
5. Error handling is simplified; illegal formats immediately return `-1`
6. Error tolerance for malformed input is **not guaranteed to match** `scanf`
7. **Do not mix** `yscanf` **with** `scanf` **or** `getchar` on the same input stream
   If single-character input is needed, use: `yscanf("%c",&ch);`or the provided `getchar` replacement: `ygetchar();`

Therefore, `yscanf` is best suited for **controlled input scenarios with known formats**, rather than general-purpose user input parsing.

6. Usage Recommendations

1. Recommended for competitive programming, performance-critical code, and projects with predefined input formats
2. Do **not** mix `yscanf` functions with standard input functions such as `scanf` or `getchar`
3. Always check the return value of `yscanf` to ensure all expected inputs were successfully read

7. Additional Notes

This release is a **public test version**.
If you encounter any issues during use, feedback is welcome.

Contact Email:
`yuzhouhunter@outlook.com`