

yapp

Yet Another Print Protocol

A lightweight `printf` replacement as a Windows DLL

Part of the `mtstdlib` project

Version	1.0
Files	yapp.h, yapp.c
Platform	Windows (MinGW/GCC), Linux/macOS
Language	C99
License	MIT

Contents

1	Overview	2
1.1	Why yapp?	2
1.2	File Structure	2
2	Building the Library	2
2.1	Prerequisites	2
2.2	Step 1 — Compile the DLL (Windows)	2
2.3	Step 2 — Compile Your Program	3
2.4	Step 3 — Run	3
2.5	Linux / macOS (shared object)	3
3	API Reference	3
3.1	yapp — Standard Output	4
3.2	yappln — Auto-Newline Output	4
3.3	yapf — Stream Output	5
3.4	Format Specifier Quick Reference	5
4	UTF-8 Support on Windows	6
5	Export Macros	6
6	Complete Example	7
7	Function Summary	7
8	Troubleshooting	8
9	License	8

1 Overview

yapp is a minimal, single-header-plus-source C library that wraps the standard `printf` family into three ergonomic functions: `yapp`, `yappln`, and `yapf`.

On Windows, it ships as a **DLL** and automatically sets the console code page to **UTF-8** the moment it is loaded — no manual setup required.

1.1 Why yapp?

- Drop-in replacement for `printf` with an identical call signature.
- `yappln` saves you from typing `\n` on every line.
- `yapf` lets you target `stderr`, a log file, or any `FILE*` stream.
- **UTF-8 is enabled automatically on Windows** via `DllMain` — no `SetConsoleOutputCP()` calls cluttering your `main()`.

1.2 File Structure

File	Purpose
yapp.h	Public API — include this in your project
yapp.c	Implementation — compile into the DLL

2 Building the Library

2.1 Prerequisites

- **Windows:** MinGW-w64 with GCC (available via `MSYS2` or `WinLibs`)
- **Linux / macOS:** Any GCC or Clang installation

2.2 Step 1 — Compile the DLL (Windows)

Open **PowerShell** or **CMD** in the folder containing `yapp.h` and `yapp.c`:

```
gcc -shared -o yapp.dll yapp.c "-Wl,--out-implib,yapp.a" -  
DYAPP_EXPORTS -Wall -O2
```

Warning

The `-Wl,...` argument contains commas, which PowerShell misinterprets as argument separators. Wrap the entire flag in double quotes as shown above.

This produces two files:

File	Purpose
yapp.dll	The dynamic library loaded at runtime
yapp.a	Import library used by the linker at compile time

2.3 Step 2 — Compile Your Program

```
gcc -o myapp.exe main.c -L. -lyapp -I. -Wall
```

Flag	Meaning
-L.	Look for <code>.a</code> libraries in the current directory
-lyapp	Link against <code>yapp.a</code>
-I.	Look for headers (<code>yapp.h</code>) in the current directory

2.4 Step 3 — Run

Place `yapp.dll` in the **same folder** as your `.exe`:

```
project\  
  myapp.exe  
  yapp.dll      <-- must be here
```

Then run:

```
.\myapp.exe
```

2.5 Linux / macOS (shared object)

```
# Build shared library  
gcc -shared -fPIC -o libyapp.so yapp.c -Wall -O2  
  
# Build program  
gcc -o myapp main.c -L. -lyapp -I. -Wall  
  
# Run  
LD_LIBRARY_PATH=. ./myapp
```

3 API Reference

Include the header at the top of every source file that uses `yapp`:

```
1 #include "yapp.h"
```

All three functions accept **printf-style format strings** and a variadic argument list. The return value is the number of characters written (same contract as `printf`).

3.1 yapp — Standard Output

```
1 int yapp(const char *fmt, ...);
```

Direct replacement for printf. Writes a formatted string to `stdout`. No newline is added automatically.

Parameters

Parameter	Type	Description
<code>fmt</code>	<code>const char*</code>	printf-style format string
<code>...</code>	-	Optional format arguments

Return Value

Number of characters written, or a negative value on error (same as `printf`).

Example

```
1 yapp("Hello, World!\n");
2 yapp("Name: %s | Age: %d\n", "Matej", 20);
3 yapp("Pi = %.4f\n", 3.14159);
```

3.2 yappln — Auto-Newline Output

```
1 int yappln(const char *fmt, ...);
```

Identical to `yapp`, but **appends a newline character** (`\n`) automatically after the formatted output. Useful when every call should occupy its own line.

Return Value

Number of characters written *including* the appended newline.

Example

```
1 /* These two calls produce identical output: */
2 yapp("Loading...\n");
3 yappln("Loading...");          /* no \n needed */
4
5 yappln("Item %d: %s", 1, "apple");
6 yappln("Item %d: %s", 2, "banana");
```

3.3 yapf — Stream Output

```
1 int yapf(void *stream, const char *fmt, ...);
```

Writes formatted output to any **FILE*** **stream** — equivalent to `fprintf`. The *void** type avoids requiring `<stdio.h>` in `yapp.h` itself; pass any valid `FILE*` pointer.

Parameters

Parameter	Type	Description
<code>stream</code>	<code>void*</code>	Target stream (<code>stdout</code> , <code>stderr</code> , or a file)
<code>fmt</code>	<code>const char*</code>	printf-style format string
<code>...</code>	-	Optional format arguments

Example

```
1 #include <stdio.h>    /* for stderr, fopen */
2 #include "yapp.h"
3
4 /* Write to stderr */
5 yapf(stderr, "Error: file not found: %s\n", filename);
6
7 /* Write to a log file */
8 FILE *log = fopen("app.log", "a");
9 if (log) {
10     yapf(log, "[INFO] Server started on port %d\n", 8080);
11     fclose(log);
12 }
```

3.4 Format Specifier Quick Reference

All three functions accept the same format specifiers as `printf`:

Specifier	Type	Example
<code>%d</code>	Signed integer	<code>yapp("%d", 42) → 42</code>
<code>%u</code>	Unsigned integer	<code>yapp("%u", 255u) → 255</code>
<code>%f</code>	Float / double	<code>yapp("%.2f", 3.14) → 3.14</code>
<code>%e</code>	Scientific notation	<code>yapp("%e", 0.001) → 1.000000e-03</code>
<code>%s</code>	String	<code>yapp("%s", "hi") → hi</code>
<code>%c</code>	Single character	<code>yapp("%c", 'A') → A</code>
<code>%x</code>	Hexadecimal (lower)	<code>yapp("%x", 255) → ff</code>
<code>%X</code>	Hexadecimal (upper)	<code>yapp("%X", 255) → FF</code>
<code>%o</code>	Octal	<code>yapp("%o", 8) → 10</code>
<code>%p</code>	Pointer address	<code>yapp("%p", ptr)</code>
<code>%zu</code>	<code>size_t</code>	<code>yapp("%zu", sizeof(int))</code>
<code>%%</code>	Literal percent sign	<code>yapp("100%%") → 100%</code>

4 UTF-8 Support on Windows

By default, the Windows console uses **code page 1250** (Central European), which causes characters like á, č, ž, ř to display as garbage when your source file is saved as UTF-8.

yapp solves this transparently using DllMain:

```

1  BOOL WINAPI DllMain(HINSTANCE hinstDLL,
2                      DWORD      fdwReason,
3                      LPVOID      lpvReserved)
4  {
5      if (fdwReason == DLL_PROCESS_ATTACH) {
6          SetConsoleOutputCP(CP_UTF8); /* stdout/stderr -> UTF-8 */
7          SetConsoleCP(CP_UTF8);      /* stdin          -> UTF-8 */
8      }
9      return TRUE;
10 }
```

Windows calls DllMain automatically whenever the DLL is loaded into a process. You do **not** need to call anything in your `main()` — UTF-8 is active from the very first instruction of your program.

Note

Your source files must be saved in **UTF-8 encoding** (which is the default in VS Code, CLion, and most modern editors). If your editor uses a different encoding, the characters will still not render correctly even with this fix.

5 Export Macros

The header uses a compile-time macro to switch between `dllexport` and `dllimport` automatically:

```

1  #ifdef _WIN32
2      #ifdef YAPP_EXPORTS
3          #define YAPPAPI __declspec(dllexport) /* compiling the DLL */
4      #else
5          #define YAPPAPI __declspec(dllimport) /* using the DLL */
6      #endif
7  #else
8      #define YAPPAPI      /* Linux / macOS */
9  #endif
```

Situation	Macro defined?	Result
Compiling yapp.c → DLL	-DYAPP_EXPORTS	<code>__declspec(dllexport)</code>
Compiling your main.c	(not defined)	<code>__declspec(dllimport)</code>
Linux / macOS	N/A	empty (no annotation needed)

You never need to define `YAPP_EXPORTS` yourself — it is only used during the DLL build step.

6 Complete Example

Listing 1: main.c — full example using all three functions

```
1  #include <stdio.h>      /* for stderr */
2  #include "yapp.h"
3
4  int main(void)
5  {
6      /* --- yapp: same as printf --- */
7      yapp("=== yapp demo ===\n");
8      yapp("Formatted int    : %d\n",    42);
9      yapp("Formatted float  : %.2f\n",  3.14159);
10     yapp("Formatted string: %s\n",    "hello");
11
12     /* --- yappln: automatic newline --- */
13     yappln("--- yappln demo ---");
14     yappln("Line %d", 1);
15     yappln("Line %d", 2);
16
17     /* --- yapf: write to stderr --- */
18     yapf(stderr, "[ERROR] Something went wrong: code %d\n", 404);
19
20     /* --- yapf: write to a file --- */
21     FILE *log = fopen("output.log", "w");
22     if (log) {
23         yapf(log, "Log entry: user=%s, status=%s\n",
24              "matej", "ok");
25         fclose(log);
26         yappln("Log written to output.log");
27     }
28
29     return 0;
30 }
```

Expected Output

```
=== yapp demo ===
Formatted int    : 42
Formatted float  : 3.14
Formatted string: hello
--- yappln demo ---
Line 1
Line 2
Log written to output.log
```

(stderr output appears interleaved in the console but is not shown above.)

7 Function Summary

Function	Stream	Auto newline	Equivalent to
yapp(fmt, ...)	stdout	No	printf(fmt, ...)
yappln(fmt, ...)	stdout	Yes	printf(fmt, ...); putchar('\n')
yapf(stream, fmt, ...)	any FILE*	No	fprintf(stream, fmt, ...)

8 Troubleshooting

Missing argument in parameter list (PowerShell)

PowerShell splits on commas in `-Wl,-out-implib,....`. Wrap the flag in double quotes:

```
"-Wl,-out-implib,yapp.a"
```

Garbled characters in the console

Make sure you are using the **updated** `yapp.c` that contains `DllMain` and recompile the DLL. Also verify your source file is saved in UTF-8.

yapp.dll not found at runtime

The DLL must sit in the **same directory** as your `.exe`, or be on the system PATH. The linker-time `yapp.a` is only needed at compile time.

Linker error: undefined reference to 'yapp'

You forgot `-L. -lyapp` when compiling your program, or the `yapp.a` import library is missing. Rerun the DLL build step.

9 License

yapp is released under the **MIT License**. You are free to use, modify, and distribute it in personal and commercial projects.

yapp v1.0 — part of the mtstdlib project