# My interest to know what is .h and where is this printf function source code?

Login as root in linux

```
[jpandya@JMP ~]$ sudo su -
[sudo] password for jpandya:
[root@JMP ~]# cd /
```

```
[root@JMP /]# find ./ -name stdio.h
./usr/share/splint/lib/stdio.h
./usr/lib/i386-redhat-linux4E/include/bits/stdio.h
./usr/lib/i386-redhat-linux4E/include/stdio.h
./usr/lib/bcc/include/stdio.h
./usr/include/c++/4.4.7/tr1/stdio.h
./usr/include/bits/stdio.h
./usr/include/stdio.h
[root@JMP /]# grep "printf" ./usr/include/stdio.h
extern int fprintf (FILE *__restrict __stream,
```

```
extern int printf (__const char *__restrict __format, ...);
```

**//Because, here i see data types in bracket, this seems a declaration of function.**

```
extern int sprintf (char *__restrict __s,
extern int vfprintf (FILE *__restrict __s, __const char *__restrict __format,
extern int vprintf (__const char *__restrict __format, _G_va_list __arg);
extern int vsprintf (char *__restrict __s, __const char *__restrict __format,
extern int snprintf (char *__restrict __s, size_t __maxlen,
     __THROW __attribute__ ((__format__ (__printf__, 3, 4)));
extern int vsnprintf (char *__restrict __s, size_t __maxlen,
     __THROW __attribute__ ((__format__ (__printf__, 3, 0)));
extern int vasprintf (char **__restrict __ptr, __const char *__restrict __f,
     __THROW __attribute__ ((__format__ (__printf__, 2, 0)));
extern int __asprintf (char **__restrict __ptr,
     __THROW __attribute__ ((__format__ (__printf__, 2, 3)));
extern int asprintf (char **__restrict __ptr,
     __THROW __attribute__ ((__format__ (__printf__, 2, 3)));
extern int vdprintf (int __fd, __const char *__restrict __fmt,
     __attribute__ ((__format__ (__printf__, 2, 0)));
extern int dprintf (int __fd, __const char *__restrict __fmt, ...)
     __attribute__ ((__format__ (__printf__, 2, 3)));
extern int obstack_printf (struct obstack *__restrict __obstack,
     __THROW __attribute__ ((__format__ (__printf__, 2, 3)));
extern int obstack_vprintf (struct obstack *__restrict __obstack,
     __THROW __attribute__ ((__format__ (__printf__, 2, 0)));
[root@JMP /]#
```

```
vi /usr/include/stdio.h
....
/* Write formatted output to stdout.

   This function is a possible cancellation point and therefore not
   marked with __THROW.  */
extern int printf (__const char *__restrict __format, ...);
....
```

**What did I learn?**

.h file contains function declaration only. Source code is not in this file.

Also, NOT yet able to find which library may be .o file, gets linked to my program .o and finally I am able to use printf function.

**From web I found below:**

File named libc.so contains compressed and bundled all library object files. This file is getting linked when you execute a c program.

```
[root@JMP lib]# pwd
/lib
[root@JMP lib]# ll libc.so.6
lrwxrwxrwx 1 root root 11 Nov 15 14:07 libc.so.6 -> libc-2.5.so

#man nm - list symbols from object files
nm libc.so.6  | grep printf

...

00046cb0 T printf
000463a0 T printf_size
00046370 T printf_size_info
0003c970 t printf_unknown
000470b0 t printf_unknown
000448a0 W register_printf_function
...
```

Archive libraries (.a) are statically linked i.e when you compile your program with -c option in gcc. So, if there's any change in library, you need to compile and build your code again.

The advantage of .so (shared object) over .a library is that they are linked during the runtime i.e. after creation of your .o file -o option in gcc. So, if there's any change in .so file, you don't need to recompile your main program. But make sure that your main program is linked to the new .so file with ln command.

Courtesy: http://stackoverflow.com

So this means, source codes of are not available in libraries.

Source Code is available via [glibc package in linux]

Download and extract:

https://launchpad.net/glibc

https://launchpad.net/glibc/head/2.18/+download/glibc-2.18.tar.gz

```
[root@JMP glibc-2.18]# pwd
/home/jpandya/Downloads/glibc-2.18
[root@JMP glibc-2.18]# vi libio/stdio.c
[root@JMP glibc-2.18]# vi stdio-common/printf.c
```

**Wow, below is function definition of printf. Great. But because to achieve simpicity, it breaks work in to different modules/funcitons and hence it is calling vfprintf.**

```
/* Write formatted output to stdout from the format string FORMAT.  */
/* VARARGS1 */
int
__printf (const char *format, ...)
{
  va_list arg;
  int done;

  va_start (arg, format);
  done = vfprintf (stdout, format, arg);
  va_end (arg);

  return done;
}
```

```
[root@JMP glibc-2.18]# vi stdio-common/vfprintf.c
```

....

```
  /* Now flush anything from the helper to the S. */
#ifdef COMPILE_WPRINTF
  if ((to_flush = (hp->_wide_data->_IO_write_ptr
                  - hp->_wide_data->_IO_write_base)) > 0)
    {
      if ((int) _IO_sputn (s, hp->_wide_data->_IO_write_base, to_flush)
          != to_flush)
        result = -1;
    }
#else
  if ((to_flush = hp->_IO_write_ptr - hp->_IO_write_base) > 0)
    {
      if ((int) _IO_sputn (s, hp->_IO_write_base, to_flush) != to_flush)
        result = -1;
    }
#endif
....
```

See above code vfprintf is using IO calls.

**Now I know littler more about the great 'C. :)**

*Contributors (CE Department, DDU):*

1. *Prof. Jigar Pandya*