



# libc

- ▶ Most functions from course so far are from libc
- ▶ Standardized by the Portable Operating System Interface for Computer Environments (ISO/IEC 9945, also known as POSIX).
- ▶ Widely used implementation is the GNU libc of the GNU project
- ▶ Used by default with the GNU C compiler (gcc)
- ▶ Other implementations exist, sometimes with subtle incompatibilities!

# Copying memory

```
#include <string.h>

void *
memcpy (void *restrict to,
        const void *restrict from,
        size_t size);
```

C99 restrict keyword says the pointer is the *only* thing that accesses the underlying object (no aliasing).

# Copying memory

```
#include <string.h>

void *
memmove (void *to,
         const void *from,
         size_t size);
```

# Copying strings

```
#include <string.h>

char *
strcpy (char *restrict to,
        const char *restrict from);

char *
strncpy (char *restrict to,
         const char *restrict from);

char *
strdup (const char *s);
```

## Concatenating strings

```
#include <string.h>

char *
strcat (char *restrict to,
        const char *restrict from);
```

## Searching

```
#include <string.h>

void *
memchr (const void *block, int c, size_t size);
void *
memrchr(const void *block, int c, size_t size);

char *
strchr (const char *string, int c);
char *
strrchr(const char *string, int c);

char *
strstr (const char *haystack,
        const char *needle);

char *
strcasestr (const char *haystack,
            const char *needle);
```

## Truncated copies

```
#include <string.h>

char *
strncpy (char *restrict to,
         const char *restrict from,
         size_t size); // Dangerous!

char *
strndup (const char *s,
         size_t size);

char *
strncat (char *restrict to,
         const char *restrict from,
         size_t size);
```

## Binary Search

```
#include <search.h>

typedef int
(*comparison_fn_t)(const void *,
                  const void *);

void *
bsearch (const void *key,
         const void *array,
         size_t count,
         size_t size,
         comparison_fn_t compare);
```

# Time

- ▶ Simple time: `time_t`, typically gives the number of seconds of elapsed time since the Unix Epoch (1.1.1970).
- ▶ High-resolution time: `struct timeval`
- ▶ Local time or broken-down time: `struct tm`

# Getting and setting the time

```
#include <time.h>

time_t
time (time_t *result);

int
stime (const time_t *newtime);
```

# Breaking it down

```
#include <time.h>

struct tm {
    int tm_sec;    int tm_min;    int tm_hour;
    int tm_mday;  int tm_mon;    int tm_year;
    int tm_wday;  int tm_yday;    int tm_isdst;
    long int tm_gmtoff;  const char *tm_zone;
};

struct tm *
localtime (const time_t *time);

struct tm *
gmtime (const time_t *time); /* UTC was ↘
→formerly called GMT */
```

# Printing it out

```
#include <time.h>

char *
asctime (const struct tm *brokentime);
/* => "Tue Dec 26 13:46:22 2017\n" */

size_t
strftime (char *s,
          size_t size,
          const char *template,
          const struct tm *brokentime)
/* Important: %z == RFC 822/ISO 8601:1988 */
```

# Sleeping

```
#include <unistd.h>

unsigned int
sleep (unsigned int seconds);

int
nanosleep (const struct timespec *req,
           struct timespec *remaining);
```

# Processing variable argument lists: Example

```
void m (const char *format, ...) {
    va_list ap;
    va_start (ap, format);
    while (*format) switch (*(format++)) {
        case 'i':{int i = va_arg (ap, int); break;}
        case 'f':{float f = va_arg (ap, float);
                 break;}
        case 'p':{void *p = va_arg (ap, void *);
                 break;}
        default: exit (1);
    }
    va_end(ap);
}

m ("iffp", 4, 4.5, 5.5, &data);
```

# Processing variable argument lists

```
#include <stdarg.h>

void va_start(va_list ap, last);

type va_arg(va_list ap, type);

void va_end(va_list ap);
```

Other Libraries

## Other libraries

- ▶ libcurl – network client
- ▶ libz – gzip compression
- ▶ libtidy – parse HTML
- ▶ libmagic – determine file type
- ▶ libjansson – JSON manipulation
- ▶ libsqlite3 – SQL database

## Conclusion

- ▶ **Libraries are essential**
  - ▶ Avoid re-inventing the wheel!
- ▶ **A lot of possibilities**
  - ▶ Use free software libraries: be in control!
  - ▶ Still many too choose from!
  - ▶ Good API design is hard, learn from existing APIs!
  - ▶ Be aware of potential compatibility issues, even with libc.

## Conclusion

## Bibliography

- ▶ The GNU C Library:  
<https://www.gnu.org/software/libc/>