Complete C time.h Library Documentation

Table of Contents

Data Types

time_t

```
typedef long time_t;  // Usually long int, represents seconds since epoch
```

clock_t

```
typedef long clock_t; // Usually long int, represents processor time
```

struct tm

```
struct tm {
    int tm_sec;   // seconds (0-60, 60 for leap second)
    int tm_min;   // minutes (0-59)
    int tm_hour;  // hours (0-23)
    int tm_mday;  // day of month (1-31)
    int tm_mon;   // month (0-11, 0=January)
    int tm_year;  // year since 1900
    int tm_wday;  // day of week (0-6, 0=Sunday)
    int tm_yday;  // day of year (0-365)
```

```
    int tm_isdst; // daylight saving time flag
};
```

Macros and Constants

`CLOCKS_PER_SEC`

```
#define CLOCKS_PER_SEC 1000000  // Number of clock ticks per second
```

`NULL`

```
#define NULL ((void*)0)  // Null pointer constant
```

Time Manipulation Functions

1. `clock()`

Prototype: `clock_t clock(void);`

Description: Returns processor time used by the program

Return: Number of clock ticks, or -1 on error

```
#include <stdio.h>
#include <time.h>

int main() {
    clock_t start = clock();

    // Some computation
    for(int i = 0; i < 1000000; i++);

    clock_t end = clock();

    printf("clock() start: %ld\n", start);
    printf("clock() end: %ld\n", end);
    printf("CPU time used: %f seconds\n",
           ((double)(end - start)) / CLOCKS_PER_SEC);
```

```
    return 0;
}
```

2. time()

Prototype: `time_t time(time_t *timer);`

Description: Returns current calendar time

Return: Seconds since epoch (Jan 1, 1970), or -1 on error

```
#include <stdio.h>
#include <time.h>

int main() {
    time_t current_time;
    time_t result = time(&current_time);

    printf("time() return value: %ld\n", result);
    printf("time() via parameter: %ld\n", current_time);

    // Alternative usage
    time_t direct_time = time(NULL);
    printf("time(NULL): %ld\n", direct_time);
    return 0;
}
```

3. difftime()

Prototype: `double difftime(time_t time1, time_t time0);`

Description: Calculates difference between two times

Return: Difference in seconds as double

```
#include <stdio.h>
#include <time.h>
#include <unistd.h>  // for sleep()

int main() {
    time_t start = time(NULL);
    sleep(3);  // Wait 3 seconds
    time_t end = time(NULL);

    double diff = difftime(end, start);
```

```
    printf("difftime() result: %f seconds\n", diff);
    printf("difftime() as integer: %.0f seconds\n", diff);
    return 0;
}
```

Time Conversion Functions

4. gmtime()

Prototype: `struct tm *gmtime(const time_t *timer);`

Description: Converts time_t to struct tm in UTC

Return: Pointer to struct tm, or NULL on error

```
#include <stdio.h>
#include <time.h>

int main() {
    time_t current_time = time(NULL);
    struct tm *utc_time = gmtime(&current_time);

    if (utc_time != NULL) {
        printf("gmtime() result:\n");
        printf("  Year: %d\n", utc_time->tm_year + 1900);
        printf("  Month: %d\n", utc_time->tm_mon + 1);
        printf("  Day: %d\n", utc_time->tm_mday);
        printf("  Hour: %d\n", utc_time->tm_hour);
        printf("  Minute: %d\n", utc_time->tm_min);
        printf("  Second: %d\n", utc_time->tm_sec);
        printf("  Day of week: %d\n", utc_time->tm_wday);
        printf("  Day of year: %d\n", utc_time->tm_yday);
        printf("  DST flag: %d\n", utc_time->tm_isdst);
    } else {
        printf("gmtime() returned NULL\n");
    }
    return 0;
}
```

5. localtime()

Prototype: `struct tm *localtime(const time_t *timer);`

Description: Converts time_t to struct tm in local time

Return: Pointer to struct tm, or NULL on error

```
#include <stdio.h>
#include <time.h>

int main() {
    time_t current_time = time(NULL);
    struct tm *local_time = localtime(&current_time);

    if (local_time != NULL) {
        printf("localtime() result:\n");
        printf("  Year: %d\n", local_time->tm_year + 1900);
        printf("  Month: %d\n", local_time->tm_mon + 1);
        printf("  Day: %d\n", local_time->tm_mday);
        printf("  Hour: %d\n", local_time->tm_hour);
        printf("  Minute: %d\n", local_time->tm_min);
        printf("  Second: %d\n", local_time->tm_sec);
        printf("  Day of week: %d\n", local_time->tm_wday);
        printf("  Day of year: %d\n", local_time->tm_yday);
        printf("  DST flag: %d\n", local_time->tm_isdst);
    } else {
        printf("localtime() returned NULL\n");
    }
    return 0;
}
```

6. mktime()

Prototype: `time_t mktime(struct tm *timeptr);`

Description: Converts struct tm to time_t

Return: time_t value, or -1 on error

```
#include <stdio.h>
#include <time.h>

int main() {
    struct tm custom_time = {0};
    custom_time.tm_year = 2024 - 1900;  // 2024
    custom_time.tm_mon = 11;            // December (0-based)
    custom_time.tm_mday = 25;           // 25th
```

```
    custom_time.tm_hour = 12;           // 12 PM
    custom_time.tm_min = 30;            // 30 minutes
    custom_time.tm_sec = 45;            // 45 seconds
    custom_time.tm_isdst = -1;          // Let system determine DST

    time_t result = mktime(&custom_time);

    printf("mktime() input:\n");
    printf("  Date: %d-%02d-%02d %02d:%02d:%02d\n",
           custom_time.tm_year + 1900, custom_time.tm_mon + 1,
           custom_time.tm_mday, custom_time.tm_hour,
           custom_time.tm_min, custom_time.tm_sec);

    if (result != -1) {
        printf("mktime() result: %ld\n", result);
        printf("mktime() normalized struct:\n");
        printf("  Day of week: %d\n", custom_time.tm_wday);
        printf("  Day of year: %d\n", custom_time.tm_yday);
        printf("  DST flag: %d\n", custom_time.tm_isdst);
    } else {
        printf("mktime() returned -1 (error)\n");
    }
    return 0;
}
```

Time Formatting Functions

7. `asctime()`

Prototype: `char *asctime(const struct tm *timeptr);`

Description: Converts struct tm to string representation

Return: Pointer to static string, or NULL on error

```
#include <stdio.h>
#include <time.h>

int main() {
    time_t current_time = time(NULL);
    struct tm *local_time = localtime(&current_time);

    char *time_string = asctime(local_time);
```

```
    if (time_string != NULL) {
        printf("asctime() result: %s", time_string);  // Note: already ha
        printf("asctime() string length: %lu\n", strlen(time_string));
    } else {
        printf("asctime() returned NULL\n");
    }
    return 0;
}
```

8. ctime()

Prototype: `char *ctime(const time_t *timer);`

Description: Converts time_t to string representation

Return: Pointer to static string, or NULL on error

```
#include <stdio.h>
#include <time.h>

int main() {
    time_t current_time = time(NULL);
    char *time_string = ctime(&current_time);

    if (time_string != NULL) {
        printf("ctime() result: %s", time_string);  // Note: already has
        printf("ctime() without newline: %.24s\n", time_string);
        printf("ctime() string length: %lu\n", strlen(time_string));
    } else {
        printf("ctime() returned NULL\n");
    }
    return 0;
}
```

9. strftime()

Prototype: `size_t strftime(char *s, size_t maxsize, const char *format, const struct tm *timeptr);`

Description: Formats time according to format specifiers

Return: Number of characters written (excluding null terminator), or 0 on error

```c
#include <stdio.h>
#include <time.h>

int main() {
    time_t current_time = time(NULL);
    struct tm *local_time = localtime(&current_time);
    char buffer[200];

    // Various format examples
    size_t result1 = strftime(buffer, sizeof(buffer), "%Y-%m-%d %H:%M:%S"
    printf("strftime() ISO format: %s (returned: %zu)\n", buffer, result1

    size_t result2 = strftime(buffer, sizeof(buffer), "%A, %B %d, %Y", lc
    printf("strftime() long format: %s (returned: %zu)\n", buffer, result

    size_t result3 = strftime(buffer, sizeof(buffer), "%I:%M %p", local_t
    printf("strftime() 12-hour: %s (returned: %zu)\n", buffer, result3);

    size_t result4 = strftime(buffer, sizeof(buffer), "Week %U, Day %j",
    printf("strftime() week/day: %s (returned: %zu)\n", buffer, result4);

    size_t result5 = strftime(buffer, sizeof(buffer), "%c", local_time);
    printf("strftime() locale format: %s (returned: %zu)\n", buffer, resu

    // Test with small buffer
    char small_buffer[5];
    size_t result6 = strftime(small_buffer, sizeof(small_buffer), "%Y-%m-
    printf("strftime() small buffer result: %zu\n", result6);
    if (result6 > 0) {
        printf("strftime() small buffer content: %s\n", small_buffer);
    }

    return 0;
}
```

Complete Example Program

Here's a comprehensive example that demonstrates all time.h functions:

```c
#include <stdio.h>
#include <time.h>
```

```c
#include <string.h>
#include <unistd.h>

void print_tm_struct(const struct tm *tm_ptr, const char *label) {
    printf("%s:\n", label);
    printf("  tm_sec: %d\n", tm_ptr->tm_sec);
    printf("  tm_min: %d\n", tm_ptr->tm_min);
    printf("  tm_hour: %d\n", tm_ptr->tm_hour);
    printf("  tm_mday: %d\n", tm_ptr->tm_mday);
    printf("  tm_mon: %d\n", tm_ptr->tm_mon);
    printf("  tm_year: %d\n", tm_ptr->tm_year);
    printf("  tm_wday: %d\n", tm_ptr->tm_wday);
    printf("  tm_yday: %d\n", tm_ptr->tm_yday);
    printf("  tm_isdst: %d\n", tm_ptr->tm_isdst);
}

int main() {
    printf("=== COMPLETE time.h FUNCTION DEMONSTRATION ===\n\n");

    // 1. clock()
    printf("1. clock() function:\n");
    clock_t start_clock = clock();
    printf("   Initial clock(): %ld ticks\n", start_clock);

    // Some work
    for(volatile int i = 0; i < 1000000; i++);

    clock_t end_clock = clock();
    printf("   Final clock(): %ld ticks\n", end_clock);
    printf("   CLOCKS_PER_SEC: %ld\n", CLOCKS_PER_SEC);
    printf("   CPU time: %f seconds\n\n",
           ((double)(end_clock - start_clock)) / CLOCKS_PER_SEC);

    // 2. time()
    printf("2. time() function:\n");
    time_t current_time_1;
    time_t result_time = time(&current_time_1);
    time_t current_time_2 = time(NULL);
    printf("   time(&var): %ld\n", result_time);
    printf("   via parameter: %ld\n", current_time_1);
    printf("   time(NULL): %ld\n\n", current_time_2);

    // 3. difftime()
    printf("3. difftime() function:\n");
```

```
time_t time1 = time(NULL);
sleep(2);
time_t time2 = time(NULL);
double time_diff = difftime(time2, time1);
printf("   time1: %ld\n", time1);
printf("   time2: %ld\n", time2);
printf("   difftime(time2, time1): %f seconds\n\n", time_diff);

// 4. gmtime()
printf("4. gmtime() function:\n");
struct tm *utc_time = gmtime(&current_time_1);
if (utc_time) {
    print_tm_struct(utc_time, "   UTC time");
} else {
    printf("   gmtime() returned NULL\n");
}
printf("\n");

// 5. localtime()
printf("5. localtime() function:\n");
struct tm *local_time = localtime(&current_time_1);
if (local_time) {
    print_tm_struct(local_time, "   Local time");
} else {
    printf("   localtime() returned NULL\n");
}
printf("\n");

// 6. mktime()
printf("6. mktime() function:\n");
struct tm custom_tm = {0};
custom_tm.tm_year = 2024 - 1900;
custom_tm.tm_mon = 0;  // January
custom_tm.tm_mday = 1;
custom_tm.tm_hour = 0;
custom_tm.tm_min = 0;
custom_tm.tm_sec = 0;
custom_tm.tm_isdst = -1;

printf("   Input: 2024-01-01 00:00:00\n");
time_t mktime_result = mktime(&custom_tm);
printf("   mktime() result: %ld\n", mktime_result);
if (mktime_result != -1) {
    printf("   Normalized tm_wday: %d\n", custom_tm.tm_wday);
```

```
    printf("   Normalized tm_yday: %d\n", custom_tm.tm_yday);
}
printf("\n");

// 7. asctime()
printf("7. asctime() function:\n");
char *asctime_str = asctime(local_time);
if (asctime_str) {
    printf("   asctime() result: %s", asctime_str);
    printf("   String length: %lu\n", strlen(asctime_str));
} else {
    printf("   asctime() returned NULL\n");
}
printf("\n");

// 8. ctime()
printf("8. ctime() function:\n");
char *ctime_str = ctime(&current_time_1);
if (ctime_str) {
    printf("   ctime() result: %s", ctime_str);
    printf("   String length: %lu\n", strlen(ctime_str));
} else {
    printf("   ctime() returned NULL\n");
}
printf("\n");

// 9. strftime()
printf("9. strftime() function:\n");
char format_buffer[100];

size_t len1 = strftime(format_buffer, sizeof(format_buffer), "%Y-%m-%
printf("   ISO format: %s (length: %zu)\n", format_buffer, len1);

size_t len2 = strftime(format_buffer, sizeof(format_buffer), "%A, %B
printf("   Long format: %s (length: %zu)\n", format_buffer, len2);

size_t len3 = strftime(format_buffer, sizeof(format_buffer), "%%Y=%%Y
printf("   Literal %% test: %s (length: %zu)\n", format_buffer, len3)

// Test buffer overflow
char tiny_buffer[5];
size_t len4 = strftime(tiny_buffer, sizeof(tiny_buffer), "%Y-%m-%d",
printf("   Small buffer (size 5): returned %zu\n", len4);
```

```
    printf("\n=== END OF DEMONSTRATION ===\n");

    return 0;
}
```

Format Specifiers for strftime()

Specifier	Description	Example
%a	Abbreviated weekday name	Mon
%A	Full weekday name	Monday
%b	Abbreviated month name	Jan
%B	Full month name	January
%c	Complete date and time	Mon Jan 1 12:00:00 2024
%d	Day of month (01-31)	01
%H	Hour (00-23)	12
%I	Hour (01-12)	12
%j	Day of year (001-366)	001
%m	Month (01-12)	01
%M	Minute (00-59)	30
%p	AM/PM	PM
%S	Second (00-61)	45
%U	Week of year (00-53, Sunday first)	00
%w	Weekday (0-6, 0=Sunday)	1
%W	Week of year (00-53, Monday first)	00
%x	Date representation	01/01/24
%X	Time representation	12:30:45
%y	Year without century (00-99)	24
%Y	Year with century	2024
%Z	Timezone name	EST
%%	Literal % character	%

Notes

1. **Thread Safety**: Functions like `gmtime()`, `localtime()`, `asctime()`, and `ctime()` return pointers to static data and are not thread-safe. Use their `_r` variants for thread safety where available.
2. **Epoch**: time_t represents seconds since January 1, 1970, 00:00:00 UTC (Unix epoch).
3. **Year Representation**: In struct tm, tm_year is years since 1900, so add 1900 for actual year.
4. **Month Representation**: tm_mon is 0-based (0=January, 11=December).
5. **Error Handling**: Always check return values. Functions returning pointers may return NULL on error, and mktime() returns -1 on error.
6. **Buffer Sizes**: When using strftime(), ensure your buffer is large enough. The function returns 0 if the buffer is too small.

This documentation covers all standard time.h functions with complete examples showing how to print and use their return values.