**clock\_t clock(void)** returns the number of clock ticks elapsed since the program was launched. To get the number of seconds used by the CPU, you will need to divide by CLOCKS\_PER\_SEC.

On a 32 bit system where CLOCKS\_PER\_SEC equals 1000000 this function will return the same value approximately every 72 minutes.

**Declaration**

Following is the declaration for clock() function.

clock\_t clock(void)

**Parameters**

* **NA**

**Return Value**

This function returns the number of clock ticks elapsed since the start of the program. On failure, the function returns a value of -1.

**Example**

The following example shows the usage of clock() function.

[Live Demo](http://tpcg.io/84Cc7C)

#include <time.h>

#include <stdio.h>

int main () {

clock\_t start\_t, end\_t, total\_t;

int i;

start\_t = clock();

printf("Starting of the program, start\_t = %ld\n", start\_t);

printf("Going to scan a big loop, start\_t = %ld\n", start\_t);

for(i=0; i< 10000000; i++) {

}

end\_t = clock();

printf("End of the big loop, end\_t = %ld\n", end\_t);

total\_t = (double)(end\_t - start\_t) / CLOCKS\_PER\_SEC;

printf("Total time taken by CPU: %f\n", total\_t );

printf("Exiting of the program...\n");

return(0);

}

Let us compile and run the above program that will produce the following result −

Starting of the program, start\_t = 0

Going to scan a big loop, start\_t = 0

End of the big loop, end\_t = 20000

Total time taken by CPU: 0.000000

Exiting of the program...

**clock\_t clock(void)** returns the number of clock ticks elapsed since the program was launched. To get the number of seconds used by the CPU, you will need to divide by CLOCKS\_PER\_SEC.

On a 32 bit system where CLOCKS\_PER\_SEC equals 1000000 this function will return the same value approximately every 72 minutes.

**Declaration**

Following is the declaration for clock() function.

clock\_t clock(void)

**Parameters**

* **NA**

**Return Value**

This function returns the number of clock ticks elapsed since the start of the program. On failure, the function returns a value of -1.

**Example**

The following example shows the usage of clock() function.

[Live Demo](http://tpcg.io/84Cc7C)

#include <time.h>

#include <stdio.h>

int main () {

clock\_t start\_t, end\_t, total\_t;

int i;

start\_t = clock();

printf("Starting of the program, start\_t = %ld\n", start\_t);

printf("Going to scan a big loop, start\_t = %ld\n", start\_t);

for(i=0; i< 10000000; i++) {

}

end\_t = clock();

printf("End of the big loop, end\_t = %ld\n", end\_t);

total\_t = (double)(end\_t - start\_t) / CLOCKS\_PER\_SEC;

printf("Total time taken by CPU: %f\n", total\_t );

printf("Exiting of the program...\n");

return(0);

}

Let us compile and run the above program that will produce the following result −

Starting of the program, start\_t = 0

Going to scan a big loop, start\_t = 0

End of the big loop, end\_t = 20000

Total time taken by CPU: 0.000000

Exiting of the program...

-----------------------------------------------------------------------------------

**difftime(time\_t time1, time\_t time2)** returns the difference of seconds between **time1** and **time2** i.e. **(time1 - time2)**. The two times are specified in calendar time, which represents the time elapsed since the Epoch (00:00:00 on January 1, 1970, Coordinated Universal Time (UTC)).

**Declaration**

Following is the declaration for difftime() function.

double difftime(time\_t time1, time\_t time2)

**Parameters**

* **time1** − This is the time\_t object for end time.
* **time2** − This is the time\_t object for start time.

**Return Value**

This function returns the difference of two times (time1 - time2) as a double value.

**Example**

The following example shows the usage of difftime() function.

[Live Demo](http://tpcg.io/o6QW0i)

#include <stdio.h>

#include <time.h>

int main () {

time\_t start\_t, end\_t;

double diff\_t;

printf("Starting of the program...\n");

time(&start\_t);

printf("Sleeping for 5 seconds...\n");

sleep(5);

time(&end\_t);

diff\_t = difftime(end\_t, start\_t);

printf("Execution time = %f\n", diff\_t);

printf("Exiting of the program...\n");

return(0);

}

Let us compile and run the above program that will produce the following result −

Starting of the program...

Sleeping for 5 seconds...

Execution time = 5.000000

Exiting of the program...

-------------------------------------------------------------------------------------------

**time\_t mktime(struct tm \*timeptr)** converts the structure pointed to by **timeptr** into a time\_t value according to the local time zone.

## Declaration

Following is the declaration for mktime() function.

time\_t mktime(struct tm \*timeptr)

## Parameters

* **timeptr** − This is the pointer to a time\_t value representing a calendar time, broken down into its components. Below is the detail of timeptr structure

struct tm {

int tm\_sec; /\* seconds, range 0 to 59 \*/

int tm\_min; /\* minutes, range 0 to 59 \*/

int tm\_hour; /\* hours, range 0 to 23 \*/

int tm\_mday; /\* day of the month, range 1 to 31 \*/

int tm\_mon; /\* month, range 0 to 11 \*/

int tm\_year; /\* The number of years since 1900 \*/

int tm\_wday; /\* day of the week, range 0 to 6 \*/

int tm\_yday; /\* day in the year, range 0 to 365 \*/

int tm\_isdst; /\* daylight saving time \*/

};

## Return Value

This function returns a time\_t value corresponding to the calendar time passed as argument. On error, a -1 value is returned.

## Example

The following example shows the usage of mktime() function.

#include

#include

int main () {

int ret;

struct tm info;

char buffer[80];

info.tm\_year = 2001 - 1900;

info.tm\_mon = 7 - 1;

info.tm\_mday = 4;

info.tm\_hour = 0;

info.tm\_min = 0;

info.tm\_sec = 1;

info.tm\_isdst = -1;

ret = mktime(&info);

if( ret == -1 ) {

printf("Error: unable to make time using mktime\n");

} else {

strftime(buffer, sizeof(buffer), "%c", &info );

printf(buffer);

}

return(0);

}

Let us compile and run the above program that will produce the following result −

Wed Jul 4 00:00:01 2001

-----------------------------------------------------------------------------

**time\_t time(time\_t \*seconds)** returns the time since the Epoch (00:00:00 UTC, January 1, 1970), measured in seconds. If **seconds** is not NULL, the return value is also stored in variable **seconds**.

**Declaration**

Following is the declaration for time() function.

time\_t time(time\_t \*t)

**Parameters**

* **seconds** − This is the pointer to an object of type time\_t, where the seconds value will be stored.

**Return Value**

The current calendar time as a time\_t object.

**Example**

The following example shows the usage of time() function.

[Live Demo](http://tpcg.io/BDXhsE)

#include <stdio.h>

#include <time.h>

int main () {

time\_t seconds;

seconds = time(NULL);

printf("Hours since January 1, 1970 = %ld\n", seconds/3600);

return(0);

}

Let us compile and run the above program that will produce the following result −

Hours since January 1, 1970 = 393923

-----------------------------------------------------------------------------------------------

**char \*asctime(const struct tm \*timeptr)** returns a pointer to a string which represents the day and time of the structure **struct timeptr**.

**Declaration**

Following is the declaration for asctime() function.

char \*asctime(const struct tm \*timeptr)

**Parameters**

The **timeptr** is a pointer to tm structure that contains a calendar time broken down into its components as shown below −

struct tm {

int tm\_sec; /\* seconds, range 0 to 59 \*/

int tm\_min; /\* minutes, range 0 to 59 \*/

int tm\_hour; /\* hours, range 0 to 23 \*/

int tm\_mday; /\* day of the month, range 1 to 31 \*/

int tm\_mon; /\* month, range 0 to 11 \*/

int tm\_year; /\* The number of years since 1900 \*/

int tm\_wday; /\* day of the week, range 0 to 6 \*/

int tm\_yday; /\* day in the year, range 0 to 365 \*/

int tm\_isdst; /\* daylight saving time \*/

};

**Return Value**

This function returns a C string containing the date and time information in a human-readable format **Www Mmm dd hh:mm:ss yyyy**, where *Www* is the weekday, *Mmm* the month in letters, *dd* the day of the month, *hh:mm:ss* the time, and *yyyy* the year.

**Example**

The following example shows the usage of asctime() function.

[Live Demo](http://tpcg.io/xeqVkg)

#include <stdio.h>

#include <string.h>

#include <time.h>

int main () {

struct tm t;

t.tm\_sec = 10;

t.tm\_min = 10;

t.tm\_hour = 6;

t.tm\_mday = 25;

t.tm\_mon = 2;

t.tm\_year = 89;

t.tm\_wday = 6;

puts(asctime(&t));

return(0);

}

Let us compile and run the above program that will produce the following result −

Sat Mar 25 06:10:10 1989

-------------------------------------------------------------------------------------------

**char \*ctime(const time\_t \*timer)** returns a string representing the localtime based on the argument **timer**.

The returned string has the following format: **Www Mmm dd hh:mm:ss yyyy**, where *Www* is the weekday, *Mmm* the month in letters, *dd* the day of the month, *hh:mm:ss* the time, and *yyyy* the year.

**Declaration**

Following is the declaration for ctime() function.

char \*ctime(const time\_t \*timer)

**Parameters**

* **timer** − This is the pointer to a time\_t object that contains a calendar time.

**Return Value**

This function returns a C string containing the date and time information in a human-readable format.

**Example**

The following example shows the usage of ctime() function.

[Live Demo](http://tpcg.io/wz1UTb)

#include <stdio.h>

#include <time.h>

int main () {

time\_t curtime;

time(&curtime);

printf("Current time = %s", ctime(&curtime));

return(0);

}

Let us compile and run the above program that will produce the following result −

Current time = Mon Aug 13 08:23:14 2012

-----------------------------------------------------------------------------------------------------

**struct tm \*gmtime(const time\_t \*timer)** uses the value pointed by timer to fill a **tm** structure with the values that represent the corresponding time, expressed in Coordinated Universal Time (UTC) or GMT timezone.

**Declaration**

Following is the declaration for gmtime() function.

struct tm \*gmtime(const time\_t \*timer)

**Parameters**

* **timeptr** − This is the pointer to a time\_t value representing a calendar time.

**Return Value**

This function returns pointer to a tm structure with the time information filled in. Below is the detail of timeptr structure −

struct tm {

int tm\_sec; /\* seconds, range 0 to 59 \*/

int tm\_min; /\* minutes, range 0 to 59 \*/

int tm\_hour; /\* hours, range 0 to 23 \*/

int tm\_mday; /\* day of the month, range 1 to 31 \*/

int tm\_mon; /\* month, range 0 to 11 \*/

int tm\_year; /\* The number of years since 1900 \*/

int tm\_wday; /\* day of the week, range 0 to 6 \*/

int tm\_yday; /\* day in the year, range 0 to 365 \*/

int tm\_isdst; /\* daylight saving time \*/

};

**Example**

The following example shows the usage of gmtime() function.

[Live Demo](http://tpcg.io/13DKiT)

#include <stdio.h>

#include <time.h>

#define BST (+1)

#define CCT (+8)

int main () {

time\_t rawtime;

struct tm \*info;

time(&rawtime);

/\* Get GMT time \*/

info = gmtime(&rawtime );

printf("Current world clock:\n");

printf("London : %2d:%02d\n", (info->tm\_hour+BST)%24, info->tm\_min);

printf("China : %2d:%02d\n", (info->tm\_hour+CCT)%24, info->tm\_min);

return(0);

}

Let us compile and run the above program that will produce the following result −

Current world clock:

London : 14:10

China : 21:10

----------------------------------------------------------------------------------------------------

**struct tm \*localtime(const time\_t \*timer)** uses the time pointed by **timer** to fill a **tm** structure with the values that represent the corresponding local time. The value of **timer** is broken up into the structure **tm** and expressed in the local time zone.

## Declaration

Following is the declaration for localtime() function.

struct tm \*localtime(const time\_t \*timer)

## Parameters

* **timer** − This is the pointer to a time\_t value representing a calendar time.

## Return Value

This function returns a pointer to a **tm** structure with the time information filled in. Following is the tm structure information −

struct tm {

int tm\_sec; /\* seconds, range 0 to 59 \*/

int tm\_min; /\* minutes, range 0 to 59 \*/

int tm\_hour; /\* hours, range 0 to 23 \*/

int tm\_mday; /\* day of the month, range 1 to 31 \*/

int tm\_mon; /\* month, range 0 to 11 \*/

int tm\_year; /\* The number of years since 1900 \*/

int tm\_wday; /\* day of the week, range 0 to 6 \*/

int tm\_yday; /\* day in the year, range 0 to 365 \*/

int tm\_isdst; /\* daylight saving time \*/

};

## Example

The following example shows the usage of localtime() function.

#include <stdio.h>

#include <time.h>

int main () {

time\_t rawtime;

struct tm \*info;

time( &rawtime );

info = localtime( &rawtime );

printf("Current local time and date: %s", asctime(info));

return(0);

}

Let us compile and run the above program that will produce the following result −

Current local time and date: Thu Aug 23 09:12:05 2012

--------------------------------------------------------------------------------------------------------------------

**size\_t strftime(char \*str, size\_t maxsize, const char \*format, const struct tm \*timeptr)** formats the time represented in the structure **timeptr** according to the formatting rules defined in **format** and stored into **str**.

**Declaration**

Following is the declaration for strftime() function.

size\_t strftime(char \*str, size\_t maxsize, const char \*format, const struct tm \*timeptr)

**Parameters**

* **str** − This is the pointer to the destination array where the resulting C string is copied.
* **maxsize** − This is the maximum number of characters to be copied to str.
* **format** − This is the C string containing any combination of regular characters and special format specifiers. These format specifiers are replaced by the function to the corresponding values to represent the time specified in tm. The format specifiers are −

|  |  |  |
| --- | --- | --- |
| **Specifier** | **Replaced By** | **Example** |
| %a | Abbreviated weekday name | Sun |
| %A | Full weekday name | Sunday |
| %b | Abbreviated month name | Mar |
| %B | Full month name | March |
| %c | Date and time representation | Sun Aug 19 02:56:02 2012 |
| %d | Day of the month (01-31) | 19 |
| %H | Hour in 24h format (00-23) | 14 |
| %I | Hour in 12h format (01-12) | 05 |
| %j | Day of the year (001-366) | 231 |
| %m | Month as a decimal number (01-12) | 08 |
| %M | Minute (00-59) | 55 |
| %p | AM or PM designation | PM |
| %S | Second (00-61) | 02 |
| %U | Week number with the first Sunday as the first day of week one (00-53) | 33 |
| %w | Weekday as a decimal number with Sunday as 0 (0-6) | 4 |
| %W | Week number with the first Monday as the first day of week one (00-53) | 34 |
| %x | Date representation | 08/19/12 |
| %X | Time representation | 02:50:06 |
| %y | Year, last two digits (00-99) | 01 |
| %Y | Year | 2012 |
| %Z | Timezone name or abbreviation | CDT |
| %% | A % sign | % |

* **timeptr** − This is the pointer to a tm structure that contains a calendar time broken down into its components as shown below −

struct tm {

int tm\_sec; /\* seconds, range 0 to 59 \*/

int tm\_min; /\* minutes, range 0 to 59 \*/

int tm\_hour; /\* hours, range 0 to 23 \*/

int tm\_mday; /\* day of the month, range 1 to 31 \*/

int tm\_mon; /\* month, range 0 to 11 \*/

int tm\_year; /\* The number of years since 1900 \*/

int tm\_wday; /\* day of the week, range 0 to 6 \*/

int tm\_yday; /\* day in the year, range 0 to 365 \*/

int tm\_isdst; /\* daylight saving time \*/

};

**Return Value**

If the resulting C string fits in less than size characters (which includes the terminating null-character), the total number of characters copied to str (not including the terminating null-character) is returned otherwise, it returns zero.

**Example**

The following example shows the usage of strftime() function.

[Live Demo](http://tpcg.io/AI8DkS)

#include <stdio.h>

#include <time.h>

int main () {

time\_t rawtime;

struct tm \*info;

char buffer[80];

time( &rawtime );

info = localtime( &rawtime );

strftime(buffer,80,"%x - %I:%M%p", info);

printf("Formatted date & time : |%s|\n", buffer );

return(0);

}

Let us compile and run the above program that will produce the following result −

Formatted date & time : |08/23/12 - 12:40AM|