7.23 Date and time `<time.h>`

7.23.1 Components of time

1 The header **`<time.h>`** defines two macros, and declares several types and functions for manipulating time. Many functions deal with a *calendar time* that represents the current date (according to the Gregorian calendar) and time. Some functions deal with *local time*, which is the calendar time expressed for some specific time zone, and with *Daylight Saving Time*, which is a temporary change in the algorithm for determining local time. The local time zone and Daylight Saving Time are implementation-defined.

2 The macros defined are **`NULL`** (described in 7.17); and

```
CLOCKS_PER_SEC
```

which expands to an expression with type **`clock_t`** (described below) that is the number per second of the value returned by the **`clock`** function. |

3 The types declared are **`size_t`** (described in 7.17);

```
clock_t
```

and

```
time_t
```

which are arithmetic types capable of representing times; and

```
struct tm
```

which holds the components of a calendar time, called the *broken-down time*.

4 The range and precision of times representable in **`clock_t`** and **`time_t`** are implementation-defined. The **`tm`** structure shall contain at least the following members, in any order. The semantics of the members and their normal ranges are expressed in the comments.[274)]

```
int tm_sec;    // seconds after the minute — [0, 60]
int tm_min;    // minutes after the hour — [0, 59]
int tm_hour;   // hours since midnight — [0, 23]
int tm_mday;   // day of the month — [1, 31]
int tm_mon;    // months since January — [0, 11]
int tm_year;   // years since 1900
int tm_wday;   // days since Sunday — [0, 6]
int tm_yday;   // days since January 1 — [0, 365]
int tm_isdst;  // Daylight Saving Time flag
```

274) The range [0, 60] for **`tm_sec`** allows for a positive leap second.

The value of `tm_isdst` is positive if Daylight Saving Time is in effect, zero if Daylight Saving Time is not in effect, and negative if the information is not available.

7.23.2 Time manipulation functions

7.23.2.1 The `clock` function

Synopsis

1
```
#include <time.h>
clock_t clock(void);
```

Description

2 The `clock` function determines the processor time used.

Returns

3 The `clock` function returns the implementation's best approximation to the processor time used by the program since the beginning of an implementation-defined era related only to the program invocation. To determine the time in seconds, the value returned by the `clock` function should be divided by the value of the macro `CLOCKS_PER_SEC`. If the processor time used is not available or its value cannot be represented, the function returns the value `(clock_t)(-1)`.[275)]

7.23.2.2 The `difftime` function

Synopsis

1
```
#include <time.h>
double difftime(time_t time1, time_t time0);
```

Description

2 The `difftime` function computes the difference between two calendar times: `time1 - time0`.

Returns

3 The `difftime` function returns the difference expressed in seconds as a `double`.

275) In order to measure the time spent in a program, the `clock` function should be called at the start of the program and its return value subtracted from the value returned by subsequent calls.

7.23.2.3 The `mktime` function

Synopsis

1
```
#include <time.h>
time_t mktime(struct tm *timeptr);
```

Description

2 The **`mktime`** function converts the broken-down time, expressed as local time, in the structure pointed to by **`timeptr`** into a calendar time value with the same encoding as that of the values returned by the **`time`** function. The original values of the **`tm_wday`** and **`tm_yday`** components of the structure are ignored, and the original values of the other components are not restricted to the ranges indicated above.[276)] On successful completion, the values of the **`tm_wday`** and **`tm_yday`** components of the structure are set appropriately, and the other components are set to represent the specified calendar time, but with their values forced to the ranges indicated above; the final value of **`tm_mday`** is not set until **`tm_mon`** and **`tm_year`** are determined.

Returns

3 The **`mktime`** function returns the specified calendar time encoded as a value of type **`time_t`**. If the calendar time cannot be represented, the function returns the value **`(time_t)(-1)`**.

4 EXAMPLE What day of the week is July 4, 2001?

```
#include <stdio.h>
#include <time.h>
static const char *const wday[] = {
      "Sunday", "Monday", "Tuesday", "Wednesday",
      "Thursday", "Friday", "Saturday", "-unknown-"
};
struct tm time_str;
/* ... */
```

276) Thus, a positive or zero value for **`tm_isdst`** causes the **`mktime`** function to presume initially that Daylight Saving Time, respectively, is or is not in effect for the specified time. A negative value causes it to attempt to determine whether Daylight Saving Time is in effect for the specified time.

```
time_str.tm_year   = 2001 - 1900;
time_str.tm_mon    = 7 - 1;
time_str.tm_mday   = 4;
time_str.tm_hour   = 0;
time_str.tm_min    = 0;
time_str.tm_sec    = 1;
time_str.tm_isdst  = -1;
if (mktime(&time_str) == (time_t)(-1))
      time_str.tm_wday = 7;
printf("%s\n", wday[time_str.tm_wday]);
```

7.23.2.4 The `time` function

Synopsis

1
```
#include <time.h>
time_t time(time_t *timer);
```

Description

2 The **`time`** function determines the current calendar time. The encoding of the value is unspecified.

Returns

3 The **`time`** function returns the implementation's best approximation to the current calendar time. The value **`(time_t)(-1)`** is returned if the calendar time is not available. If **`timer`** is not a null pointer, the return value is also assigned to the object it points to.

7.23.3 Time conversion functions

1 Except for the **`strftime`** function, these functions each return a pointer to one of two types of static objects: a broken-down time structure or an array of **`char`**. Execution of any of the functions that return a pointer to one of these object types may overwrite the information in any object of the same type pointed to by the value returned from any previous call to any of them. The implementation shall behave as if no other library functions call these functions.

7.23.3.1 The `asctime` function

Synopsis

1
```
#include <time.h>
char *asctime(const struct tm *timeptr);
```

Description

2 The **`asctime`** function converts the broken-down time in the structure pointed to by **`timeptr`** into a string in the form

```
Sun Sep 16 01:03:52 1973\n\0
```

using the equivalent of the following algorithm.

```
char *asctime(const struct tm *timeptr)
{
     static const char wday_name[7][3] = {
          "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
     };
     static const char mon_name[12][3] = {
          "Jan", "Feb", "Mar", "Apr", "May", "Jun",
          "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
     };
     static char result[26];

     sprintf(result, "%.3s %.3s%3d %.2d:%.2d:%.2d %d\n",
          wday_name[timeptr->tm_wday],
          mon_name[timeptr->tm_mon],
          timeptr->tm_mday, timeptr->tm_hour,
          timeptr->tm_min, timeptr->tm_sec,
          1900 + timeptr->tm_year);
     return result;
}
```

Returns

3 The **asctime** function returns a pointer to the string.

7.23.3.2 The ctime function

Synopsis

1
```
     #include <time.h>
     char *ctime(const time_t *timer);
```

Description

2 The **ctime** function converts the calendar time pointed to by **timer** to local time in the form of a string. It is equivalent to

```
     asctime(localtime(timer))
```

Returns

3 The **ctime** function returns the pointer returned by the **asctime** function with that broken-down time as argument.

Forward references: the **localtime** function (7.23.3.4).

7.23.3.3 The `gmtime` function

Synopsis

1
```
#include <time.h>
struct tm *gmtime(const time_t *timer);
```

Description

2 The **`gmtime`** function converts the calendar time pointed to by **`timer`** into a broken-down time, expressed as UTC.

Returns

3 The **`gmtime`** function returns a pointer to the broken-down time, or a null pointer if the specified time cannot be converted to UTC.

7.23.3.4 The `localtime` function

Synopsis

1
```
#include <time.h>
struct tm *localtime(const time_t *timer);
```

Description

2 The **`localtime`** function converts the calendar time pointed to by **`timer`** into a broken-down time, expressed as local time.

Returns

3 The **`localtime`** function returns a pointer to the broken-down time, or a null pointer if the specified time cannot be converted to local time.

7.23.3.5 The `strftime` function

Synopsis

1
```
#include <time.h>
size_t strftime(char * restrict s,
     size_t maxsize,
     const char * restrict format,
     const struct tm * restrict timeptr);
```

Description

2 The **`strftime`** function places characters into the array pointed to by **`s`** as controlled by the string pointed to by **`format`**. The format shall be a multibyte character sequence, beginning and ending in its initial shift state. The **`format`** string consists of zero or more conversion specifiers and ordinary multibyte characters. A conversion specifier consists of a **`%`** character, possibly followed by an **`E`** or **`O`** modifier character (described below), followed by a character that determines the behavior of the conversion specifier. All ordinary multibyte characters (including the terminating null character) are copied

unchanged into the array. If copying takes place between objects that overlap, the behavior is undefined. No more than **maxsize** characters are placed into the array.

3 Each conversion specifier is replaced by appropriate characters as described in the following list. The appropriate characters are determined using the **LC_TIME** category of the current locale and by the values of zero or more members of the broken-down time structure pointed to by **timeptr**, as specified in brackets in the description. If any of the specified values is outside the normal range, the characters stored are unspecified.

%a is replaced by the locale's abbreviated weekday name. [**tm_wday**]

%A is replaced by the locale's full weekday name. [**tm_wday**]

%b is replaced by the locale's abbreviated month name. [**tm_mon**]

%B is replaced by the locale's full month name. [**tm_mon**]

%c is replaced by the locale's appropriate date and time representation. [all specified in 7.23.1]

%C is replaced by the year divided by 100 and truncated to an integer, as a decimal number (**00−99**). [**tm_year**]

%d is replaced by the day of the month as a decimal number (**01−31**). [**tm_mday**]

%D is equivalent to "**%m/%d/%y**". [**tm_mon**, **tm_mday**, **tm_year**]

%e is replaced by the day of the month as a decimal number (**1−31**); a single digit is preceded by a space. [**tm_mday**]

%F is equivalent to "**%Y−%m−%d**" (the ISO 8601 date format). [**tm_year**, **tm_mon**, **tm_mday**]

%g is replaced by the last 2 digits of the week-based year (see below) as a decimal number (**00−99**). [**tm_year**, **tm_wday**, **tm_yday**]

%G is replaced by the week-based year (see below) as a decimal number (e.g., 1997). [**tm_year**, **tm_wday**, **tm_yday**]

%h is equivalent to "**%b**". [**tm_mon**]

%H is replaced by the hour (24-hour clock) as a decimal number (**00−23**). [**tm_hour**]

%I is replaced by the hour (12-hour clock) as a decimal number (**01−12**). [**tm_hour**]

%j is replaced by the day of the year as a decimal number (**001−366**). [**tm_yday**]

%m is replaced by the month as a decimal number (**01−12**). [**tm_mon**]

%M is replaced by the minute as a decimal number (**00−59**). [**tm_min**]

%n is replaced by a new-line character.

%p is replaced by the locale's equivalent of the AM/PM designations associated with a 12-hour clock. [**tm_hour**]

%r is replaced by the locale's 12-hour clock time. [**tm_hour**, **tm_min**, **tm_sec**]

%R is equivalent to "**%H:%M**". [**tm_hour**, **tm_min**]

%S is replaced by the second as a decimal number (**00−60**). [**tm_sec**]

%t is replaced by a horizontal-tab character.

%T is equivalent to "**%H:%M:%S**" (the ISO 8601 time format). [**tm_hour**, **tm_min**, **tm_sec**]

`%u`	is replaced by the ISO 8601 weekday as a decimal number (`1-7`), where Monday is 1. [`tm_wday`]
`%U`	is replaced by the week number of the year (the first Sunday as the first day of week 1) as a decimal number (`00-53`). [`tm_year`, `tm_wday`, `tm_yday`]
`%V`	is replaced by the ISO 8601 week number (see below) as a decimal number (`01-53`). [`tm_year`, `tm_wday`, `tm_yday`]
`%w`	is replaced by the weekday as a decimal number (`0-6`), where Sunday is 0. [`tm_wday`]
`%W`	is replaced by the week number of the year (the first Monday as the first day of week 1) as a decimal number (`00-53`). [`tm_year`, `tm_wday`, `tm_yday`]
`%x`	is replaced by the locale's appropriate date representation. [all specified in 7.23.1]
`%X`	is replaced by the locale's appropriate time representation. [all specified in 7.23.1]
`%y`	is replaced by the last 2 digits of the year as a decimal number (`00-99`). [`tm_year`]
`%Y`	is replaced by the year as a decimal number (e.g., `1997`). [`tm_year`]
`%z`	is replaced by the offset from UTC in the ISO 8601 format "`-0430`" (meaning 4 hours 30 minutes behind UTC, west of Greenwich), or by no characters if no time zone is determinable. [`tm_isdst`]
`%Z`	is replaced by the locale's time zone name or abbreviation, or by no characters if no time zone is determinable. [`tm_isdst`]
`%%`	is replaced by `%`.

4 Some conversion specifiers can be modified by the inclusion of an `E` or `O` modifier character to indicate an alternative format or specification. If the alternative format or specification does not exist for the current locale, the modifier is ignored.

`%Ec`	is replaced by the locale's alternative date and time representation.
`%EC`	is replaced by the name of the base year (period) in the locale's alternative representation.
`%Ex`	is replaced by the locale's alternative date representation.
`%EX`	is replaced by the locale's alternative time representation.
`%Ey`	is replaced by the offset from `%EC` (year only) in the locale's alternative representation.
`%EY`	is replaced by the locale's full alternative year representation.
`%Od`	is replaced by the day of the month, using the locale's alternative numeric symbols (filled as needed with leading zeros, or with leading spaces if there is no alternative symbol for zero).
`%Oe`	is replaced by the day of the month, using the locale's alternative numeric symbols (filled as needed with leading spaces).
`%OH`	is replaced by the hour (24-hour clock), using the locale's alternative numeric symbols.

`%OI` is replaced by the hour (12-hour clock), using the locale's alternative numeric symbols.
`%Om` is replaced by the month, using the locale's alternative numeric symbols.
`%OM` is replaced by the minutes, using the locale's alternative numeric symbols.
`%OS` is replaced by the seconds, using the locale's alternative numeric symbols.
`%Ou` is replaced by the ISO 8601 weekday as a number in the locale's alternative representation, where Monday is 1.
`%OU` is replaced by the week number, using the locale's alternative numeric symbols.
`%OV` is replaced by the ISO 8601 week number, using the locale's alternative numeric symbols.
`%Ow` is replaced by the weekday as a number, using the locale's alternative numeric symbols.
`%OW` is replaced by the week number of the year, using the locale's alternative numeric symbols.
`%Oy` is replaced by the last 2 digits of the year, using the locale's alternative numeric symbols.

5 **`%g`**, **`%G`**, and **`%V`** give values according to the ISO 8601 week-based year. In this system, weeks begin on a Monday and week 1 of the year is the week that includes January 4th, which is also the week that includes the first Thursday of the year, and is also the first week that contains at least four days in the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year; thus, for Saturday 2nd January 1999, **`%G`** is replaced by **`1998`** and **`%V`** is replaced by **`53`**. If December 29th, 30th, or 31st is a Monday, it and any following days are part of week 1 of the following year. Thus, for Tuesday 30th December 1997, **`%G`** is replaced by **`1998`** and **`%V`** is replaced by **`01`**.

6 If a conversion specifier is not one of the above, the behavior is undefined.

7 In the **`"C"`** locale, the **`E`** and **`O`** modifiers are ignored and the replacement strings for the following specifiers are:

`%a` the first three characters of **`%A`**.
`%A` one of "**`Sunday`**", "**`Monday`**", ... , "**`Saturday`**".
`%b` the first three characters of **`%B`**.
`%B` one of "**`January`**", "**`February`**", ... , "**`December`**".
`%c` equivalent to "**`%a %b %e %T %Y`**".
`%p` one of "**`AM`**" or "**`PM`**".
`%r` equivalent to "**`%I:%M:%S %p`**".
`%x` equivalent to "**`%m/%d/%y`**".
`%X` equivalent to **`%T`**.
`%Z` implementation-defined.

Returns

8 If the total number of resulting characters including the terminating null character is not more than **`maxsize`**, the **`strftime`** function returns the number of characters placed into the array pointed to by **`s`** not including the terminating null character. Otherwise, zero is returned and the contents of the array are indeterminate.