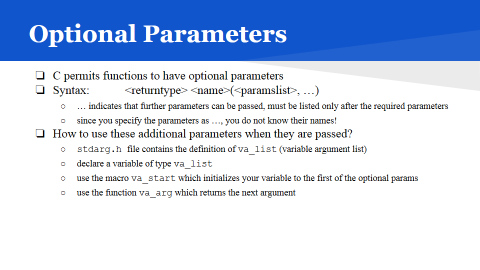
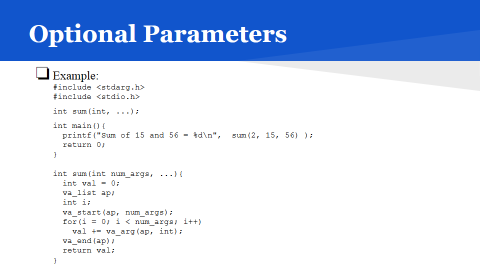
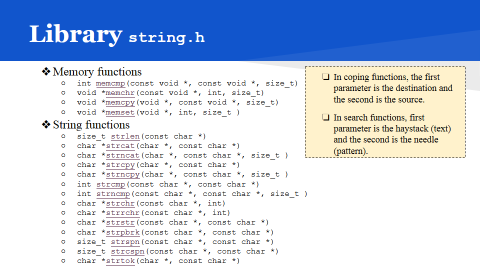
**MACRO**







# **strlen()**

## Description

The C library function **size\_t strlen(const char \*str)** computes the length of the string **str** up to, but not including the terminating null character.

## Declaration

Following is the declaration for strlen() function.

size\_t strlen(const char \*str)

## Parameters

* **str** -- This is the string whose length is to be found.

## Return Value

This function returns the length of string.

## Example

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

#include <stdio.h>

#include <string.h>

int main ()

{

char str[50];

int len;

strcpy(str, "This is tutorialspoint.com");

len = strlen(str);

printf("Length of |%s| is |%d|\n", str, len);

return(0);

}

Length of |This is tutorialspoint.com| is |26|

# **strcat()**

## Description

The C library function **char \*strcat(char \*dest, const char \*src)** appends the string pointed to by **src** to the end of the string pointed to by **dest**.

## Declaration

Following is the declaration for strcat() function.

char \*strcat(char \*dest, const char \*src)

## Parameters

* **dest** -- This is pointer to the destination array, which should contain a C string, and should be large enough to contain the concatenated resulting string.
* **src** -- This is the string to be appended. This should not overlap the destination.

## Return Value

This function returns a pointer to the resulting string dest.

## Example

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

#include <stdio.h>

#include <string.h>

int main ()

{

char src[50], dest[50];

strcpy(src, "This is source");

strcpy(dest, "This is destination");

strcat(dest, src);

printf("Final destination string : |%s|", dest);

return(0);

}

Final destination string : |This is destinationThis is source|

# **strncat()**

## Description

The C library function **char \*strncat(char \*dest, const char \*src, size\_t n)**appends the string pointed to by **src** to the end of the string pointed to by **dest**up to **n** characters long.

## Declaration

Following is the declaration for strncat() function.

char \*strncat(char \*dest, const char \*src, size\_t n)

## Parameters

* **dest** -- This is pointer to the destination array, which should contain a C string, and should be large enough to contain the concatenated resulting string which includes the additional null-character.
* **src** -- This is the string to be appended.
* **n** -- This is the maximum number of characters to be appended.

## Return Value

This function returns a pointer to the resulting string dest.

## Example

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

#include <stdio.h>

#include <string.h>

int main ()

{

char src[50], dest[50];

strcpy(src, "This is source");

strcpy(dest, "This is destination");

strncat(dest, src, 15);

printf("Final destination string : |%s|", dest);

return(0);

}

Final destination string : |This is destinationThis is source|

**Concept strcopy()**

**Syntax :**

**Char \*strcpy( char \*dst, const char\* src)**

**Own Implementation:**

char\* mystrcpy(char \*dst, char\*src)

{

While(\*dst++ = \*stc++);

}

Problems:

1. If src length larger than dst buffer overflow.
2. If dst overlaps with src

Src = I n d I a ‘\0’

1 2 3 4 5 6

Des = I n d I a ‘\0’ \_ \_

1 2 3 4 5 6

Src = I n d I a ‘\0’

Des = d I a ‘\0’ \_ \_

3 4 5 6 7 8

Src = I n d I a ‘\0’

After overlapping..

Whats the case?

Example Link: https://www.tutorialspoint.com/c\_standard\_library/c\_function\_memset.htm

# **C library function - memset()**

The C library function **void \*memset(void \*str, int c, size\_t n)** copies the character **c** (an unsigned char) to the first **n** characters of the string pointed to, by the argument **str**.

Declaration

Following is the declaration for memset() function.

void \*memset(void \*str, int c, size\_t n)

Parameters

* **str** -- This is a pointer to the block of memory to fill.
* **c** -- This is the value to be set. The value is passed as an int, but the function fills the block of memory using the unsigned char conversion of this value.
* **n** -- This is the number of bytes to be set to the value.

#include <stdio.h>

#include <string.h>

int main ()

{

char str[50];

strcpy(str,"This is string.h library function");

puts(str);

memset(str,'$',7);

puts(str);

return(0);

}

Output:

This is string.h library function

$$$$$$$ string.h library function

**Memcpy()**

The C library function **void \*memcpy(void \*str1, const void \*str2, size\_t n)** copies **n** characters from memory area **str2** to memory area **str1**.

## Declaration

Following is the declaration for memcpy() function.

void \*memcpy(void \*str1, const void \*str2, size\_t n)

## Parameters

* **str1** -- This is pointer to the destination array where the content is to be copied, type-casted to a pointer of type void\*.
* **str2** -- This is pointer to the source of data to be copied, type-casted to a pointer of type void\*.
* **n** -- This is the number of bytes to be copied.

## Return Value

This function returns a pointer to destination, which is str1.

## Example

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

#include <stdio.h>

#include <string.h>

int main ()

{

const char src[50] = "http://www.tutorialspoint.com";

char dest[50];

printf("Before memcpy dest = %s\n", dest);

memcpy(dest, src, strlen(src)+1);

printf("After memcpy dest = %s\n", dest);

return(0);

}

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

Before memcpy dest =

After memcpy dest = http://www.tutorialspoint.com

<https://www.tutorialspoint.com/c_standard_library/c_function_memcpy.htm>

# **memchr()**

## Description

The C library function **void \*memchr(const void \*str, int c, size\_t n)**searches for the first occurrence of the character **c** (an unsigned char) in the first **n** bytes of the string pointed to, by the argument **str**.

## Declaration

Following is the declaration for memchr() function.

void \*memchr(const void \*str, int c, size\_t n)

## Parameters

* **str** -- This is the pointer to the block of memory where the search is performed.
* **c** -- This is the value to be passed as an int, but the function performs a byte per byte search using the unsigned char conversion of this value.
* **n** -- This is the number of bytes to be analyzed.

## Return Value

This function returns a pointer to the matching byte or NULL if the character does not occur in the given memory area.

## Example

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

#include <stdio.h>

#include <string.h>

int main ()

{

const char str[] = "http://www.tutorialspoint.com";

const char ch = '.';

char \*ret;

ret = memchr(str, ch, strlen(str));

printf("String after |%c| is - |%s|\n", ch, ret);

return(0);

}

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

String after |.| is - |.tutorialspoint.com|

# **strtok()**

The C library function **char \*strtok(char \*str, const char \*delim)** breaks string **str** into a series of tokens using the delimiter **delim**.

## Declaration

Following is the declaration for strtok() function.

char \*strtok(char \*str, const char \*delim)

## Parameters

* **str** -- The contents of this string are modified and broken into smaller strings (tokens).
* **delim** -- This is the C string containing the delimiters. These may vary from one call to another.

## Return Value

This function returns a pointer to the last token found in the string. A null pointer is returned if there are no tokens left to retrieve.

## Example

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

#include <string.h>

#include <stdio.h>

int main()

{

char str[80] = "This is - www.tutorialspoint.com - website";

const char s[2] = "-";

char \*token;

/\* get the first token \*/

token = strtok(str, s);

/\* walk through other tokens \*/

while( token != NULL )

{

printf( " %s\n", token );

token = strtok(NULL, s);

}

return(0);

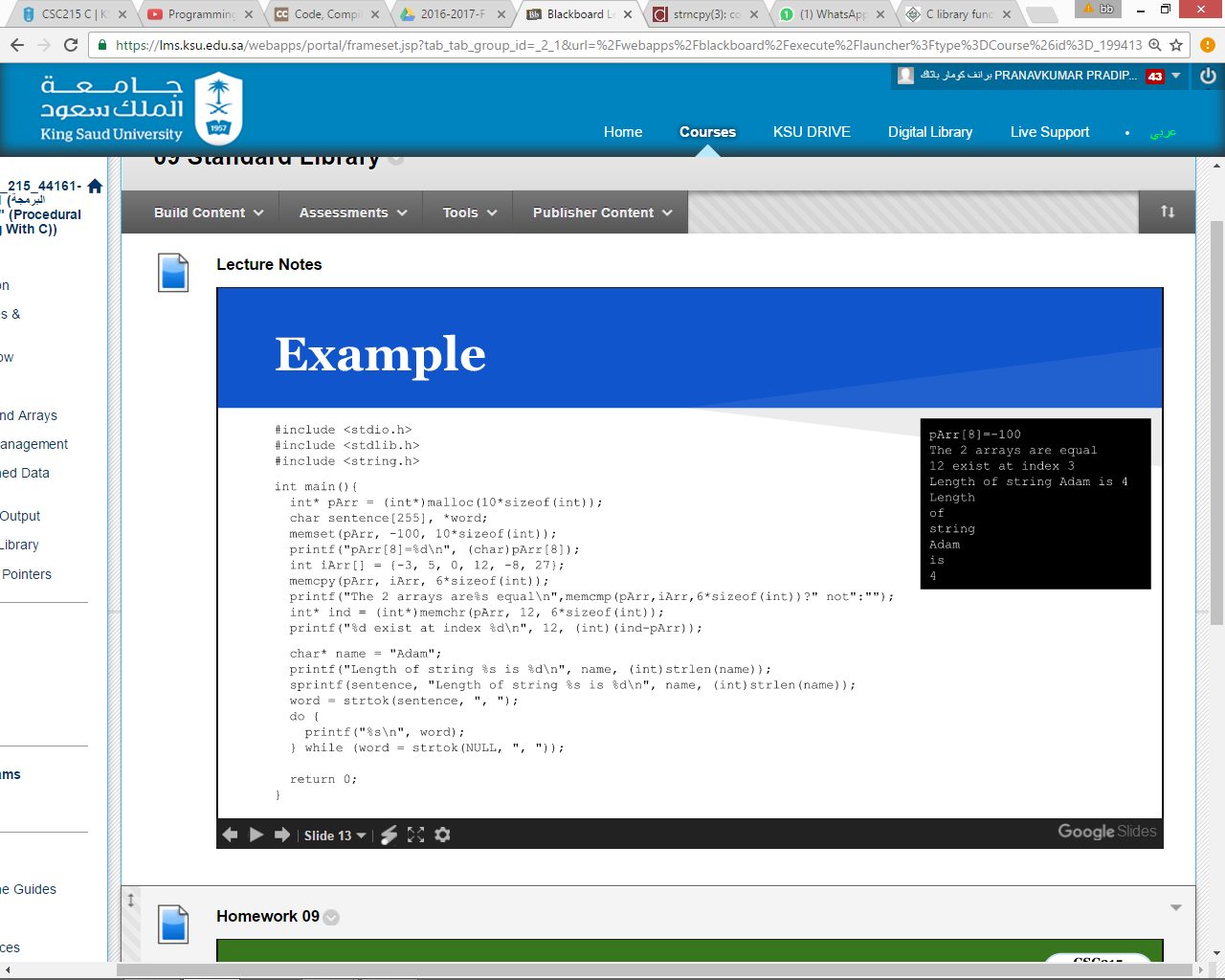
}

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

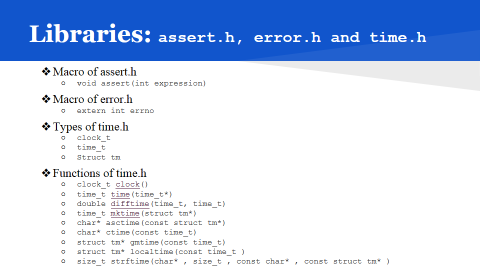
This is

www.tutorialspoint.com

website



# **<time.h>**



The **time.h** header defines four variable types, two macro and various functions for manipulating date and time.

## Library Variables

Following are the variable types defined in the header time.h:

|  |  |
| --- | --- |
| **S.N.** | **Variable & Description** |
| 1 | **size\_t**  This is the unsigned integral type and is the result of the **sizeof**keyword. |
| 2 | **clock\_t**  This is a type suitable for storing the processor time. |
| 3 | **time\_t is**  This is a type suitable for storing the calendar time. |
| 4 | **struct tm**  This is a structure used to hold the time and date. |

The tm structure has the following definition:

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 \*/

};

## Library Macros

Following are the macros defined in the header time.h:

|  |  |
| --- | --- |
| **S.N.** | **Macro & Description** |
| 1 | **NULL**  This macro is the value of a null pointer constant. |
| 2 | **CLOCKS\_PER\_SEC**  This macro represents the number of processor clocks per second. |

## Library Functions

Following are the functions defined in the header time.h:

|  |  |
| --- | --- |
| **S.N.** | **Function & Description** |
| 1 | [**char \*asctime(const struct tm \*timeptr)**](https://www.tutorialspoint.com/c_standard_library/c_function_asctime.htm)  Returns a pointer to a string which represents the day and time of the structure timeptr. |
| 2 | [**clock\_t clock(void)**](https://www.tutorialspoint.com/c_standard_library/c_function_clock.htm)  Returns the processor clock time used since the beginning of an implementation defined era (normally the beginning of the program). |
| 3 | [**char \*ctime(const time\_t \*timer)**](https://www.tutorialspoint.com/c_standard_library/c_function_ctime.htm)  Returns a string representing the localtime based on the argument timer. |
| 4 | [**double difftime(time\_t time1, time\_t time2)**](https://www.tutorialspoint.com/c_standard_library/c_function_difftime.htm)  Returns the difference of seconds between time1 and time2 (time1-time2). |
| 5 | [**struct tm \*gmtime(const time\_t \*timer)**](https://www.tutorialspoint.com/c_standard_library/c_function_gmtime.htm)  The value of timer is broken up into the structure tm and expressed in Coordinated Universal Time (UTC) also known as Greenwich Mean Time (GMT). |
| 6 | [**struct tm \*localtime(const time\_t \*timer)**](https://www.tutorialspoint.com/c_standard_library/c_function_localtime.htm)  The value of timer is broken up into the structure tm and expressed in the local time zone. |
| 7 | [**time\_t mktime(struct tm \*timeptr)**](https://www.tutorialspoint.com/c_standard_library/c_function_mktime.htm)  Converts the structure pointed to by timeptr into a time\_t value according to the local time zone. |
| 8 | [**size\_t strftime(char \*str, size\_t maxsize, const char \*format, const struct tm \*timeptr)**](https://www.tutorialspoint.com/c_standard_library/c_function_strftime.htm)  Formats the time represented in the structure timeptr according to the formatting rules defined in format and stored into str. |
| 9 | [**time\_t time(time\_t \*timer)**](https://www.tutorialspoint.com/c_standard_library/c_function_time.htm)  Calculates the current calender time and encodes it into time\_t format. |

# **asctime()**

## Description

The C library function **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.

#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

# **clock()**

## Description

The C library function **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.

#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);

}

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()**

## Description

The C library function **double 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.

#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...

