C library function

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|
```

Description

The C library function **int memcmp(const void *str1, const void *str2, size_t n))** compares the first **n** bytes of memory area **str1** and memory area **str2**.

Declaration

Following is the declaration for memcmp() function.

```
int memcmp(const void *str1, const void *str2, size_t n)
```

Parameters

- **str1** − This is the pointer to a block of memory.
- **str2** − This is the pointer to a block of memory.
- **n** − This is the number of bytes to be compared.

Return Value

- if Return value < 0 then it indicates str1 is less than str2.
- if Return value > 0 then it indicates str2 is less than str1.
- if Return value = 0 then it indicates str1 is equal to str2.

Example

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

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

int main () {
   char str1[15];
   char str2[15];
   int ret;

   memcpy(str1, "abcdef", 6);
   memcpy(str2, "ABCDEF", 6);

   ret = memcmp(str1, str2, 5);

   if(ret > 0) {
      printf("str2 is less than str1");
   } else if(ret < 0) {
      printf("str1 is less than str2");
   } else {
      printf("str1 is equal to str2");
   }

   return(0);
}
```

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

```
str2 is less than str1
```

Description

The C library function **void *memcpy(void *dest, const void *src, size_t n)** copies **n** characters from memory area **src** to memory area **dest**.

Declaration

Following is the declaration for memcpy() function.

```
void *memcpy(void *dest, const void * src, size_t n)
```

Parameters

- **dest** − This is pointer to the destination array where the content is to be copied, type-casted to a pointer of type void*.
- **src** − 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];
   strcpy(dest,"Heloooo!!");
   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 = Heloooo!!
After memcpy dest = http://www.tutorialspoint.com
```

Description

The C library function **void *memmove(void *str1, const void *str2, size_t n)** copies **n** characters from **str2** to **str1**, but for overlapping memory blocks, memmove() is a safer approach than memcpy().

Declaration

Following is the declaration for memmove() function.

```
void *memmove(void *str1, const void *str2, size_t n)
```

Parameters

- **str1** − This is a pointer to the destination array where the content is to be copied, type-casted to a pointer of type void*.
- **str2** − This is a 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 the destination, which is str1.

Example

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

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

int main () {
   char dest[] = "oldstring";
   const char src[]  = "newstring";

   printf("Before memmove dest = %s, src = %s\n", dest, src);
   memmove(dest, src, 9);
   printf("After memmove dest = %s, src = %s\n", dest, src);

   return(0);
}
```

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

```
Before memmove dest = oldstring, src = newstring
After memmove dest = newstring, src = newstring
```

Description

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.

Return Value

This function returns a pointer to the memory area str.

Example

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

```
#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);
}
```

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

```
This is string.h library function
$$$$$$$ string.h library function
```

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);
}
```

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

```
Final destination string : |This is destinationThis is source|
```

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);
}
```

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

```
Final destination string : |This is destinationThis is source|
```

Description

The C library function **int strcmp(const char *str1, const char *str2)** compares the string pointed to, by **str1** to the string pointed to by **str2**.

Declaration

Following is the declaration for strcmp() function.

```
int strcmp(const char *str1, const char *str2)
```

Parameters

- **str1** − This is the first string to be compared.
- **str2** − This is the second string to be compared.

Return Value

This function return values that are as follows −

- if Return value < 0 then it indicates str1 is less than str2.
- if Return value > 0 then it indicates str2 is less than str1.
- if Return value = 0 then it indicates str1 is equal to str2.

Example

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

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

int main () {
   char str1[15];
   char str2[15];
   int ret;


   strcpy(str1, "abcdef");
   strcpy(str2, "ABCDEF");

   ret = strcmp(str1, str2);

   if(ret < 0) {
      printf("str1 is less than str2");
   } else if(ret > 0) {
      printf("str2 is less than str1");
   } else {
      printf("str1 is equal to str2");
   }

   return(0);
}
```

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

```
str2 is less than str1
```

Description

The C library function **int strncmp(const char *str1, const char *str2, size_t n)** compares at most the first **n** bytes of **str1** and **str2**.

Declaration

Following is the declaration for strncmp() function.

```
int strncmp(const char *str1, const char *str2, size_t n)
```

Parameters

- **str1** − This is the first string to be compared.
- **str2** − This is the second string to be compared.
- **n** − The maximum number of characters to be compared.

Return Value

This function return values that are as follows −

- if Return value < 0 then it indicates str1 is less than str2.
- if Return value > 0 then it indicates str2 is less than str1.
- if Return value = 0 then it indicates str1 is equal to str2.

Example

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

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

int main () {
   char str1[15];
   char str2[15];
   int ret;

   strcpy(str1, "abcdef");
   strcpy(str2, "ABCDEF");

   ret = strncmp(str1, str2, 4);

   if(ret < 0) {
      printf("str1 is less than str2");
   } else if(ret > 0) {
      printf("str2 is less than str1");
   } else {
      printf("str1 is equal to str2");
   }

   return(0);
}
```

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

```
str2 is less than str1
```

Description

The C library function **char *strcpy(char *dest, const char *src)** copies the string pointed to, by **src** to **dest**.

Declaration

Following is the declaration for strcpy() function.

```
char *strcpy(char *dest, const char *src)
```

Parameters

- **dest** − This is the pointer to the destination array where the content is to be copied.
- **src** − This is the string to be copied.

Return Value

This returns a pointer to the destination string dest.

Example

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

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

int main () {
   char src[40];
   char dest[100];

   memset(dest, '\0', sizeof(dest));
   strcpy(src, "This is tutorialspoint.com");
   strcpy(dest, src);

   printf("Final copied string : %s\n", dest);

   return(0);
}
```

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

```
Final copied string : This is tutorialspoint.com
```

Description

The C library function **char *strncpy(char *dest, const char *src, size_t n)** copies up to **n** characters from the string pointed to, by **src** to **dest**. In a case where the length of src is less than that of n, the remainder of dest will be padded with null bytes.

Declaration

Following is the declaration for strncpy() function.

```
char *strncpy(char *dest, const char *src, size_t n)
```

Parameters

- **dest** − This is the pointer to the destination array where the content is to be copied.
- **src** − This is the string to be copied.
- **n** − The number of characters to be copied from source.

Return Value

This function returns the final copy of the copied string.

Example

The following example shows the usage of strncpy() function. Here we have used function memset() to clear the memory location.

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

int main () {
   char src[40];
   char dest[12];

   memset(dest, '\0', sizeof(dest));
   strcpy(src, "This is tutorialspoint.com");
   strncpy(dest, src, 10);

   printf("Final copied string : %s\n", dest);

   return(0);
}
```

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

```
Final copied string : This is tu
```

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);
}
```

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

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

Description

The C library function **char *strpbrk(const char *str1, const char *str2)** finds the first character in the string **str1** that matches any character specified in **str2**. This does not include the terminating null-characters.

Declaration

Following is the declaration for strpbrk() function.

```
char *strpbrk(const char *str1, const char *str2)
```

Parameters

- **str1** − This is the C string to be scanned.
- **str2** − This is the C string containing the characters to match.

Return Value

This function returns a pointer to the character in str1 that matches one of the characters in str2, or NULL if no such character is found.

Example

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

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

int main () {
   const char str1[] = "abcde2fghi3jk4l";
   const char str2[] = "34";
   char *ret;

   ret = strpbrk(str1, str2);
   if(ret) {
      printf("First matching character: %c\n", *ret);
   } else {
      printf("Character not found");
   }

   return(0);
}
```

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

```
First matching character: 3
```

Description

The C library function **char *strrchr(const char *str, int c)** searches for the last occurrence of the character **c** (an unsigned char) in the string pointed to, by the argument **str**.

Declaration

Following is the declaration for strrchr() function.

```
char *strrchr(const char *str, int c)
```

Parameters

- **str** − This is the C string.
- **c** − This is the character to be located. It is passed as its int promotion, but it is internally converted back to char.

Return Value

This function returns a pointer to the last occurrence of character in str. If the value is not found, the function returns a null pointer.

Example

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

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

int main () {
   int len;
   const char str[] = "http://www.tutorialspoint.com";
   const char ch = '.';
   char *ret;

   ret = strrchr(str, ch);

   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 - |.com|
```

Description

The C library function **size_t strspn(const char *str1, const char *str2)** calculates the length of the initial segment of **str1** which consists entirely of characters in **str2**.

Declaration

Following is the declaration for strspn() function.

```
size_t strspn(const char *str1, const char *str2)
```

Parameters

- **str1** − This is the main C string to be scanned.
- **str2** − This is the string containing the list of characters to match in str1.

Return Value

This function returns the number of characters in the initial segment of str1 which consist only of characters from str2.

Example

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

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

int main () {
   int len;
   const char str1[] = "ABCDEFG019874";
   const char str2[] = "ABCD";

   len = strspn(str1, str2);

   printf("Length of initial segment matching %d\n", len );

   return(0);
}
```

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

```
Length of initial segment matching 4
```

Description

The C library function **char *strstr(const char *haystack, const char *needle)** function finds the first occurrence of the substring **needle** in the string **haystack**. The terminating '\0' characters are not compared.

Declaration

Following is the declaration for strstr() function.

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

Parameters

- **haystack** − This is the main C string to be scanned.
- **needle** − This is the small string to be searched with-in haystack string.

Return Value

This function returns a pointer to the first occurrence in haystack of any of the entire sequence of characters specified in needle, or a null pointer if the sequence is not present in haystack.

Example

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

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


int main () {
   const char haystack[20] = "TutorialsPoint";
   const char needle[10] = "Point";
   char *ret;

   ret = strstr(haystack, needle);

   printf("The substring is: %s\n", ret);

   return(0);
}
```

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

```
The substring is: Point
```