String Man Pages

NAME

strlen - calculate the length of a string

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

#include <string.h>

size_t strlen(const char *s);

DESCRIPTION

The **strlen**() function calculates the length of the string pointed to by s, excluding the terminating null byte ('\0').

RETURN VALUE

The **strlen**() function returns the number of bytes in the string pointed to by s.

NAME

stpcpy, strcpy, strcat - copy or catenate a string

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

#include <string.h>

char *stpcpy(char *restrict dst, const char *restrict src);

char *strcpy(char *restrict dst, const char *restrict src);

char *strcat(char *restrict dst, const char *restrict src);

DESCRIPTION

stpcpy()

strcpy()

These functions copy the string pointed to by src, into a string at the buffer pointed to by dst. The programmer is responsible for allocating a destination buffer large enough, that is, strlen(src) + 1. For the difference between the two functions, see RETURN VALUE.

strcat()

This function catenates the string pointed to by src, after the string pointed to by dst (overwriting its terminating null byte). The programmer is responsible for allocating a destination buffer large enough, that is, strlen(dst) + strlen(src) + 1.

RETURN VALUE

stpcpy()

This function returns a pointer to the terminating null byte of the copied string.

strcpy()

strcat()

These functions return dst.

NAME

strdup, strndup, strdupa, strndupa - duplicate a string

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

#include <string.h>

char *strdup(const char *s);

char *strndup(const char s[.n], size_t n);

char *strdupa(const char *s);

char *strndupa(const char s[.n], size_t n);

DESCRIPTION

The **strdup**() function returns a pointer to a new string which is a duplicate of the string s. Memory for the new string is obtained with **malloc**(3), and can be freed with **free**(3).

The **strndup**() function is similar, but copies at most n bytes. If s is longer than n, only n bytes are copied, and a terminating null byte ('\0') is added.

strdupa() and **strndupa**() are similar, but use **alloca**(3) to allocate the buffer.

RETURN VALUE

On success, the **strdup**() function returns a pointer to the duplicated string. It returns NULL if insufficient memory was available, with errno set to indicate the error.

ERRORS

ENOMEM Insufficient memory available to allocate duplicate string.

NAME

strcmp, strncmp - compare two strings

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

#include <string.h>

int strcmp(const char *s1, const char *s2);

int strncmp(const char s1[.n], const char s2[.n], size_t n);

DESCRIPTION

The **strcmp**() function compares the two strings s1 and s2. The locale is not taken into account (for a locale-aware comparison, see **strcoll**(3)). The comparison is done using unsigned characters.

strcmp() returns an integer indicating the result of the comparison, as follows:

- 0, if the s1 and s2 are equal;
- a negative value if s1 is less than s2;
- a positive value if s1 is greater than s2.

The **strncmp**() function is similar, except it compares only the first (at most) n bytes of s1 and s2.

RETURN VALUE

The **strcmp**() and **strncmp**() functions return an integer less than, equal to, or greater than zero if s1 (or the first n bytes thereof) is found, respectively, to be less than, to match, or be greater than s2.

NAME

strstr, strcasestr - locate a substring

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

#include <string.h>
char *strstr(const char ***haystack, const char ***needle**);**
#define _GNU_SOURCE /* See feature_test_macros(7) */
#include <string.h>
char *strcasestr(const char ***haystack, const char ***needle**);**

DESCRIPTION

The **strstr**() function finds the first occurrence of the substring needle in the string haystack. The terminating null bytes ('\0') are not compared.

The **strcasestr**() function is like **strstr**(), but ignores the case of both arguments.

RETURN VALUE

These functions return a pointer to the beginning of the located substring, or NULL if the substring is not found.

If needle is the empty string, the return value is always haystack itself.

NAME

strchr, strrchr, strchrnul - locate character in string

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

#include <string.h>
char *strchr(const char ***s, int** c**);**
char *strrchr(const char ***s, int** c**);**

DESCRIPTION

The **strchr**() function returns a pointer to the first occurrence of the character c in the string s.

The **strrchr**() function returns a pointer to the last occurrence of the character c in the string s.

The **strchrnul**() function is like **strchr**() except that if c is not found in s, then it returns a pointer to the null byte at the end of s, rather than NULL.

Here "character" means "byte"; these functions do not work with wide or multibyte characters.

RETURN VALUE

The **strchr**() and **strrchr**() functions return a pointer to the matched character or NULL if the character is not found. The terminating null byte is considered part of the string, so that if c is specified as '\0', these functions return a pointer to the terminator.

The **strchrnul**() function returns a pointer to the matched character, or a pointer to the null byte at the end of s (i.e., s+strlen(s)) if the character is not found.

NAME

isalnum, isalpha, isascii, isblank, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, isxdigit, isalnum_l, isalpha_l, isascii_l, isblank_l, iscntrl_l, isdigit_l, isgraph_l, islower_l, isprint_l, ispunct_l, isspace_l, isupper_l, isxdigit_l - character classification functions

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

#include <ctype.h>
int isalnum(int c**);**
int isalpha(int c**);**
int iscntrl(int c**);**
int isdigit(int c**);**
int isgraph(int c**);**
int islower(int c**);**
int isprint(int c**);**
int ispunct(int c**);**
int isspace(int c**);**
int isupper(int c**);**
int isxdigit(int c**);**
int isascii(int c**);**
int isblank(int c**);**
int isalnum_l(int c**, locale_t** locale**);**
int isalpha_l(int c**, locale_t** locale**);**
int isblank_l(int c**, locale_t** locale**);**
int iscntrl_l(int c**, locale_t** locale**);**
int isdigit_l(int c**, locale_t** locale**);**
int isgraph_l(int c**, locale_t** locale**);**
int islower_l(int c**, locale_t** locale**);**
int isprint_l(int c**, locale_t** locale**);**
int ispunct_l(int c**, locale_t** locale**);**
int isspace_l(int c**, locale_t** locale**);**
int isupper_l(int c**, locale_t** locale**);**
int isxdigit_l(int c**, locale_t** locale**);**
int isascii_l(int c**, locale_t** locale**);**

DESCRIPTION

These functions check whether c, which must have the value of an unsigned char or **EOF**, falls into a certain character class according to the specified locale. The functions without the "_l" suffix perform the check based on the current locale.
The functions with the "_l" suffix perform the check based on the locale specified by the locale object locale. The behavior of these functions is undefined if locale is the special locale object **LC_GLOBAL_LOCALE** (see **duplocale**(3)) or is not a valid locale object handle.
The list below explains the operation of the functions without the "_l" suffix; the functions with the "_l" suffix differ only in using the locale object locale instead of the current locale.

isalnum()
checks for an alphanumeric character; it is equivalent to **(isalpha(**c**) || isdigit(**c**))**.

isalpha()
checks for an alphabetic character; in the standard **"C"** locale, it is equivalent to **(isupper(**c**) || islower(**c**))**. In some locales, there may be additional characters for which **isalpha**() is true—letters which are neither uppercase nor lowercase.

isascii()
checks whether c is a 7-bit unsigned char value that fits into the ASCII character set.

isblank()
checks for a blank character; that is, a space or a tab.

iscntrl()
checks for a control character.

isdigit()
checks for a digit (0 through 9).

isgraph()
checks for any printable character except space.

islower()
checks for a lowercase character.

isprint()
checks for any printable character including space.

ispunct()
checks for any printable character which is not a space or an alphanumeric character.

isspace()
checks for white-space characters. In the **"C"** and **"POSIX"** locales, these are: space, form-feed (**'\f'**), newline (**'\n'**), carriage return (**'\r'**), horizontal tab (**'\t'**), and vertical tab (**'\v'**).

isupper()
checks for an uppercase letter.

isxdigit()
checks for hexadecimal digits, that is, one of
0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F.

RETURN VALUE

The values returned are nonzero if the character c falls into the tested class, and zero if not.

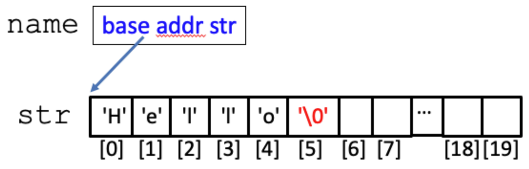
C Strings and Pointers

- Often accessed as (char *)

```
char str[20];
str = “hello”;
char *name;
name = str;
printf(”%s”, str);
printf(”%s”, name);
```

Q: How to print “ello” given these two variables str and name?

how many different ways can you come up with?



C Strings and Pointers

```
char str[20], str2[30], *str3;
str[0] = 'H';
str[1] = 'i';
str[2] = '\0';
```

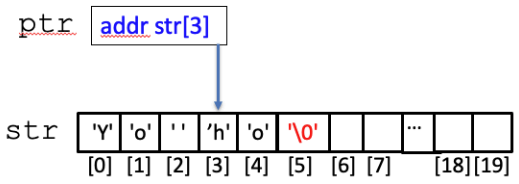
Q: How to initialize str2 and str3 to have the same string value, “Hi”, as str?

(three separate strings, with same string value “Hi”)

strstr: find string in a string

```
char *name, *ptr, str[20];
str = "Yo Ho";
ptr = strstr(str, "ho");
printf("%s", ptr);
```

Q: draw the stack diagram and output of the print statement.



Try out

```
char *str2, *ptr, str[64];
strcpy(str, “Hi How Are you?”);
```

1. code to see if `A` is in the string str (make your code work for any value stored in str, not just this example)
2. if so, create a copy of the string starting at the first `A` in str2.
3. if not concatenate (add to the end of) to str the string “ no As in here.”
4. compare str2 and str strings, and print out a message indicating which is greater than which or if they are equal

strcpy, strlen, strcat, strcmp, strstr, strchr