Course
ELEC-A7100
Course materials
Your points
Microsoft Teams
Code Vault

This course has already ended.

« 1 Basics Course materials 3 Helpful Qualifiers and Operators »
ELEC-A7100 / 4. Strings / 2 Functions for string handling

Fourth round tasks as a `ZIP file.`

We will now look into **strings** in the C language. Actually strings are just an application of C arrays, with **char** - type array members (i.e., the individual characters). In the beginning you will learn to handle simple strings. The we step aside a for a bit, and take a look at few useful specifiers in C language, needed in understanding the string functions, that are discussed in the end of this section.

Functions for string handling¶

The standard library contains functions that are helpful in operating with strings. They are defined in include header , so if you want to use them, add **#include <string.h>** in your program. The detailed descriptions of the functions can be found in the Unix manual pages, that can be accessed by the "man" command in the command line shell, or use the man content provided in the web.

Here are a few useful functions:

- strlen returns the length of a given string. The exact form of the function is `size_t strlen(const char *s)`, i.e., it takes a pointer to string as a parameter, and returns the number of characters in the return before the terminating NULL character.
- strcmp compares two null-terminated strings. The exact form is `int strcmp(const char *s1, const char *s2)`. The function returns 0, if the strings are same, or non-zero if they are different.
- strcpy and **strncpy** copy a string to another location. The exact form is `char *strcpy(char *dst, const char *src)` or `char *strncpy(char *dst, const char *src, size_t n)`, where the string pointed by **'src'** is copied to location pointed by **'dst'**. The difference with latter strncpy is that it copies at most 'n' characters, even if the original string was longer. This is useful to protect against overflow of the destination buffer. The destination buffer needs to be properly allocated before copying. These functions return the destination pointer.
- strcat and **strncat** append a string after the other string, to make them a single concatenated string. The exact form is `char *strcat(char *s1, const char *s2)` or `char *strncat(char *s1, const char *s2, size_t n)`, where the string pointed by **'s2'** is appended to the end of string **'s1'**. Again, care should be taken to ensure that the destination buffer in **'s1'** has enough space. The latter form of the function helps in that, because the **'n'** parameter gives the upper limit to the number of characters to be appended.
- strchr finds the first instance of given character in a string. The exact form is `char *strchr(const char *s, int c)`, where **'c'** is the character that is sought from string **'s'**. The function returns pointer to the first instance of the given character, or **NULL** if the character was not found in the string.
- strstr tries to find the first instance of a substring in another string. The exact form is `char *strstr(const char *s1, const char *s2)`, where **'s2'** is the string that is sought from within string **'s1'**. The function returns the pointer to the start of the first instance of the substring within **'s1'**, or **NULL** if fully matching substring was not found.

Below is an example that demonstrates the above functions, and a few other aspects related to strings in practice.

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

int main(void)
{
    char buffer[100];  // 100 bytes for working with strings

    const char *a_name = "Jaakko Jokunen";  // unmodifiable string

    // Copies a_name to the buffer we created. It is less than 99 characters,
    // so everything should be safe. The string must not be more than 99
    // characters (plus the trailing nil character).
    strcpy(buffer, a_name);

    // Buffer should now contain a string that we can output
    printf("buffer after strcpy: %s\n", buffer);

    // Adds something to follow whatever is currently in buffer
    strcat(buffer, " is my friend.");

    printf("buffer after strcat: %s\n", buffer);
    printf("strlen of the buffer: %lu\n", strlen(buffer));
    printf("size of the buffer: %lu\n", sizeof(buffer));

    // where is the first 'f'?
    char *ptr = strchr(buffer, 'f');

    printf("buffer from first 'f' onwards: %s\n", ptr);

    // where is "is"?
    ptr = strstr(buffer, "is");

    printf("buffer from \"is\" onwards: %s\n", ptr);

    return 0;
}
```

Task 4.2: Add name¶

Implement function **addname** below that adds a new name (*addme*) after string *buffer*, followed by a comma. The *buffer* has space for at most 30 characters, and everything above that must be ignored. The *main* function calls *addname* for several times, and it may overwrite the buffer, unless you somehow take this into account.

You will most likely need **strncat** and **strlen** functions.

Points 10 / 10 My submissions 2 Deadline Friday, 2 July 2021, 19:59 To be submitted alone

This course has been archived (Saturday, 31 December 2022, 20:00).

Add Names

Select your files for grading

source.c

Choose File No file chosen

Submit

Task 4.3: Count substring¶

Objective: Practice the use of string functions (although the exercise could be done without them as well).

Write function `int num_substr(const char *str, const char *sub)` that counts how many times string **'sub'** occurs in string **'str'**, and return that as return value. For example, call `num_substr("one two one twotwo three", "two")` should return **3**, because "two" occurs three times in the longer string. Note that the spaces do not have any special role in these string manipulation operations – they are just normal characters like everything else.

Hint: Function **strstr** might be helpful here. It is also useful to observe that you can process partial strings by using a pointer to the middle of string (or any array in general). In such case the function ignores the characters before the pointer, and continues processing from the pointed location.

Points 20 / 20 My submissions 1 Deadline Friday, 2 July 2021, 19:59 To be submitted alone

This course has been archived (Saturday, 31 December 2022, 20:00).

Count Strings

Select your files for grading

source.c

Choose File No file chosen

Submit