This course has already ended.

« 3 Helpful Qualifiers and Operators Course materials 5 Round feedback »

ELEC-A7100 / 4. Strings / 4 Precedence

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.

Precedence¶

Understanding the order of evaluation of different operators is important, because that affects the outcome of expressions. If precedence rules are not properly understood, tracking down invalid behavior is difficult. For example, in the case of pointer arithmetics, a forgotten pair of parenthesis may lead to invalid calculation, and invalid memory reference.

An (incomplete) list of the order of evaluation between different operators is listed below. Table 2-1 on page 53 of the K&R book gives a complete list of all the C operators and their precedence. When evaluating an expression, the operators on the top of the list are evaluated before the operators below them (some of the operators are introduced only in the next section).

1. `()` , `[]` , `->` , `.`
2. `++` , `--` , `*` (pointer) , `&` (address-of) , `(TYPE)` (type cast) , `sizeof` (associativity: right to left)
3. `*` , `/` , `%` (arithmetic operators)
4. `+` , `-` (arithmetic operators)
5. `<` , `<=` , `>` , `>=`
6. `==` , `!=`
7. `&&` (logical AND)
8. `||` (logical OR)

In most cases, the operators of similar precedence level are applied from left to right, but group 2 above makes on exception: the operators are associated from right to left.

The precedence order explains why we needed parenthesis when accessing array by the means of pointer arithmetics:

```
int arr[10];
int b;
b = *(arr + 2);
```

accesses the third element in array 'arr', but

```
int arr[10];
int b;
b = *arr + 2;
```

increases the first element in array by 2, because the dereference operator is evaluated before sum (`+` operator). When unsure, using extra pair of parenthesis is not harmful. If you want to ensure your code is clear and understandable, it is a good idea to avoid writing code that excessively depends on the order of evaluation. Usually, it is possible to split complex expressions in multiple parts, even if that may involve slightly more overhead.

It is also useful to know that C does not specify the precedence of operands around an operator. For example, in `int val = funcA() + funcB();` the two functions could be executed in either order. If the two functions have mutual dependencies, for example, they modify a common variable, the outcome of the statement can be unpredictable, and produce different results in different environments.

Task 4.4: NO NEED TO SHOUT!!!¶

Implement a function with the following interface: `char *my_toupper(char *dest, const char *src)` . Function argument *src* points to the string that you should modify as described below. The modified string will be written at address *dest*. In addition the function should return pointer to the modified string (i.e., to the same address *dest* originally points to).

The string should be modified in the following ways:

- all letter characters should be changed to upper case. You can use function **toupper**, that converts one character to upper case for this. *toupper* is defined in header *ctype.h*, so you should add `#include <ctype.h>` at the beginning of your program if you decide to do this.
- If the original string has a question mark ('?'), it should be changed to exclamation mark ('!').
- If the original string has period ('.'), it should be replaced with three exclamation marks.

You will not modify the original string, but write the result in location pointed by the *dest* variable.

For example the following main function:

```
#include <stdio.h>  // for printf
#include <string.h>  // for memset
#include <ctype.h>  // for toupper

int main(void)
{
    char dest[200];

    /* The following helps detecting string errors, e.g. missing final nil */
    memset(dest, '#', 199);
    dest[199] = 0;

    printf("%s",
        my_toupper(dest,
            "Would you like to have a sausage? It will be two euros. Here you are.\n"));

    printf("%s",
        my_toupper(dest,
            "Madam, where are you going? The health care center is over there.\n"));

    return 0;
}
```

Would output:

```
WOULD YOU LIKE TO HAVE A SAUSAGE! IT WILL BE TWO EUROS!!! HERE YOU ARE!!!
MADAM, WHERE ARE YOU GOING! THE HEALTH CARE CENTER IS OVER THERE!!!
```

Note that the *toupper* function does not necessary convert Ä and Ö letters correctly. You don't need to worry about this.

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).

Shout

Select your files for grading

source.c

Choose File No file chosen

Submit

Task 4.5: New string¶

Objective: How do the string functions really work? This exercise might help understanding them.

For this task we assume a new kind of string that does not end at `'\0'` like the normal strings. Instead, the new string terminator is question mark '?'. Therefore we need to re-implement some of the common string processing functions following the new string specifications.

Note that the above-discussed string functions defined in string.h **do not work with this exercise!** The char arrays given to the functions do not necessary contain the usual `'\0'` terminator.

a) Print string¶

Implement function `void qstr_print(const char *s)` that outputs string s until the first instance of the string terminating '?'. However, the hash character should not be printed. For example, if the function gets the following standard C string as input:

```
char *str = "Auto ajoi?kilparataa";
```

it will output:

```
Auto ajoi
```

b) String length¶

Implement function `unsigned int qstr_length(const char *s)` that returns the number of characters in array s before the terminating '?'. The hash character should not be included in count.

For example, if the function gets the following standard C string as input:

```
char *str = "Auto ajoi?kilparataa";
```

it will return:

```
9
```

c) String concatenation¶

Implement function `int qstr_cat(char *dst, char *src)` , that appends a copy of the string *src* to the *dst* string. The function should return the number of the characters in the final *dst* string. The function should copy characters only until the first '?' character and after copying the destination string must also terminate with '?'. (Hint: you can test that the destination string looks correct by using the qstr_print function)

For example, if the function gets the following strings as parameters:

```
char dst[50] = "Auto ajoi?";
char *src = " katua pitkin? lujaa";
```

it should return:

```
22
```

and the string *dst* should look like this:

```
Auto ajoi katua pitkin?
```

d) Substring search¶

Implement function `const char *qstr_strstr(const char *str1, const char *str2)` that searches for the substring *str2* from the string *str1*. The function should return a pointer to the first occurance of *str2* in *str1*. If the substring isn't found, the function should return NULL (note that NULL is defined in **stddef.h** header). Remember that the function should only search until the first '?' character.

For example, if the function gets the following strings as parameters:

```
char *str1 = "Auto ajoi katua pitkin?";
char *str2 = "katu?";
```

it should return a pointer that points to the first character of the word `katua` where the first occurance of the substring `katu` begins.

Points **25 / 25** My submissions **3** Deadline Friday, 2 July 2021, 19:59 To be submitted alone

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

New String

Select your files for grading

source.c

Choose File No file chosen

Submit