C - Pool - Tek1

Subject Day 07

C Pool Managers
looneytunes@epitech.eu

Contents

Instructions

- The subject may change until one hour before turn-in.
- Respect the norm takes time, but is good for you. This way your code will respect the norm since the first written line.
- Ask yourself if it's relevant to let a `main()` function in your turn-in knowing we will add our own.
- The robot will test your turn-in as follows:

```
$> cd ex_01
$> cc *.c ~moulinette/main_ex_01.o -L./lib/my/ -o ex01 -lmy
$> ./ex01
$> [...]
```

- This is a turn-in directory, of course you will only keep in it your final work revision. No temporary file should stand there!
  You shall leave in your directory no other files than those explicitly specified by the exercises.
  If one of your files prevents the compilation with `*.c`, the robot will not be able to do the correction and you will have a `0`. That is why it's in your interest to remove any file that doesn't work.

For every exercise, except the one in which you build your library, your files shall not contain any of the function present in the library.

- Don't forget to discuss about it in the pool section of the forum !
- Turn-in directory:
  `Piscine_C_J07`

Hints Remember it is always better to create your repository at the beginning of the day and to turn-in your work on a regular basis

Hints On the instructions of each exercises, this directory is specified for every turn-in path

Unit Tests

- It is highly recommended to test your functions when you are developing them.
- Usually, it is common to create a function named "`main`" (and a dedicated file to host it) to check the functions separately.
- Create a directory named "`tests`".
- Create a function "`int main()`" in a file named "`tests-exercise_name.c`", stored inside the directory "`tests`" previously created.
- According to you, this function must contains all the necessary call to "`exercise_name`" to cover all possible cases (special or regular) of the function.

Indices

```
Here is a partial list of tests:
- Check the empty strings
- Check the pointer's values
```

00 - Machine test

- Look at saturday's test traces that are in your mailbox and understand your grade.

01 - libmy.a

- Build your `my` library in `Piscine_C_J07/lib/my/` and name it `libmy.a`.
- This library shall contain all the following functions:

```
1   void my_putchar(char c);
2   int my_isneg(int nb);
3   int my_put_nbr(int nb);
4   int my_swap(int *a, int *b);
5   int my_putstr(char *str);
6   int my_strlen(char *str);
7   int my_getnbr(char *str);
8   void my_sort_int_tab(int *tab, int size);
9   int my_power_rec(int nb, int power);
10  int my_square_root(int nb);
11  int my_is_prime(int nombre);
12  int my_find_prime_sup(int nb);
13  char *my_strcpy(char *dest, char *src);
14  char *my_strncpy(char *dest, char *src, int nb);
15  char *my_revstr(char *str);
16  char *my_strstr(char *str, char *to_find);
17  int my_strcmp(char *s1, char *s2);
18  int my_strncmp(char *s1, char *s2, int nb);
19  char *my_strupcase(char *str);
20  char *my_strlowcase(char *str);
21  char *my_strcapitalize(char *str);
22  int my_str_isalpha(char *str);
23  int my_str_isnum(char *str);
24  int my_str_islower(char *str);
25  int my_str_isupper(char *str);
26  int my_str_isprintable(char *str);
27  int my_showstr(char *str);
28  int my_showmem(char *str, int size);
29  char *my_strcat(char *dest, char *src);
30  char *my_strncat(char *dest, char *src, int nb);
31  int my_strlcat(char *dest, char *src, int size);
```

- Your `libmy.a` library shall imperatively be in the correct folder because it will be used to compile all your programs. From now on, you shall not have a single function present in your library in any of your sources.
- Turn-in:
  `Piscine_C_J07/lib/my/libmy.a`

Today you shall also code three functions, those from the exercises below, and include them in your library

01-1 - my_strcat

- Write a function that copies a string after another (look at the man)
- The function shall be prototyped as follows:

```
1   char *my_strcat(char *dest, char *src);
```

- Turn-in:
  `Piscine_C_J07/ex_01/my_strcat.c`

Hints `man strcat`

01-2 - my_strncat

- Write a function that copies `n` characters from a string after another
- The function shall be prototyped as follows:

```
1   char *my_strncat(char *dest, char *src, int nb);
```

- Turn-in:
  `Piscine_C_J07/ex_01/my_strncat.c`

Hints `man strncat`

02 - my_aff_params

- Write a program that displays the received arguments (from the command line).
- We are talking about a program, that's why you shall have a `main` function in a `.c` of your turn-in repository.
- You shall display all the arguments, that includes argv[0].
- All the arguments shall be displayed on different lines.
- Turn-in:
  `Piscine_C_J07/ex_02/my_aff_params`
- Example :

```
$>./a.out test "This is a test " retest | cat -e
./a.out$
test$
This is a test $
retest$
$>
```

03 - my_rev_params

- Write a program that displays all the arguments received on the command line in the reverse order.
- You shall display all the arguments, that includes argv[0].
- All the arguments shall be displayed on different lines.
- Turn-in:
  `Piscine_C_J07/ex_03/my_rev_params`
- Example :

```
$>./a.out test "This is a test " retest | cat -e
retest$
This is a test $
test$
./a.out$
$>
```

04 - my_sort_params

- Write a program that displays all the arguments received on the command line ordered by ascii order.
- You shall display all the arguments, that includes argv[0].
- All the arguments shall be displayed on different lines.
- Turn-in:
  `Piscine_C_J07/ex_04/my_sort_params`
- Example :

```
$>./a.out test "This is a test " retest | cat -e
./a.out$
This is a test $
retest$
test$
$>
```

Bonus - my_strlcat

- Write a function that copies a string after another on a maximum length of `l` characters.
- The function shall be prototyped as follows:

```
1   int my_strlcat(char *dest, char *src, int size);
```

- Turn-in :
`Piscine_C_J07/ex_05/my_strlcat.c`

 *Hints* The strlcat man is on the next page

 *Hints* To test the real function: "gcc strlcat.o test_strlcat.c". The 'strlcat.o' file is located on the intranet with the subject

- Example:

```
int main()
{
  int i;

  i = strlcat("Boobies", "a", 2);
  printf("%d\n", i);
  return (0);
}
$>./a.out
3
```

Bonus (next) - man strlcat

NAME

strlcpy, **strlcat** – size-bounded string copying and concatenation

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

size_t **strlcpy**(char *dst, const char *src, size_t size);
size_t **strlcat**(char *dst, const char *src, size_t size);

DESCRIPTION

The **strlcpy**() and **strlcat**() functions copy and concatenate strings respectively.
They are designed to be safer, more consistent, and less error prone replacements for strncpy(3) and strncat(3).
Unlike those functions, **strlcpy**() and **strlcat**() take the full size of the buffer (not just the length) and guarantee
to NUL-terminate the result (as long as size is larger than 0 or, in the case of **strlcat**(), as long as there is at least one byte free in dst).
Note that you should include a byte for the NUL in size. Also note that **strlcpy**()and **strlcat**() only operate on true "C" strings.
This means that for **strlcpy**() src must be NUL-terminated and for **strlcat**() both src and dst must be NUL-terminated.
The **strlcpy**() function copies up to size - 1 characters from the NUL-terminated string src to dst, NUL-terminating the result.
The **strlcat**() function appends the NUL-terminated string src to the end of dst.
It will append at most size - strlen(dst) - 1 bytes, NUL-terminating the result.

RETURN VALUES

The **strlcpy**() and **strlcat**() functions return the total length of the string they tried to create.
For **strlcpy**() that means the length of src. For **strlcat**() that means the initial length of dst plus the length of src.
While this may seem somewhat confusing it was done to make truncation detection simple.
Note however, that if **strlcat**() traverses size characters without finding a NUL, the length of the string
is considered to be size and the destination string will not be NUL-terminated (since there was no space for the NUL).
This keeps **strlcat**() from running off the end of a string.
In practice this should not happen (as it means that either size is incorrect or that dst is not a proper "C" string).
The check exists to prevent potential security problems in incorrect code.

EXAMPLES

The following code fragment illustrates the simple case:

```
1           char *s, *p, buf[BUFSIZ];
2           ...
3           (void)strlcpy(buf, s, sizeof(buf));
4           (void)strlcat(buf, p, sizeof(buf));
```

To detect truncation, perhaps while building a pathname, something like the following might be used:

```
1           char *dir, *file, pname[MAXPATHLEN];
2           ...
3           if (strlcpy(pname, dir, sizeof(pname)) >= sizeof(pname))
4                   goto toolong;
5           if (strlcat(pname, file, sizeof(pname)) >= sizeof(pname))
6                   goto toolong;
```

Since we know how many characters we copied the first time, we can speed things up a bit by using a copy instead of an append:

```
1           char *dir, *file, pname[MAXPATHLEN];
2           size_t n;
3           ...
4           n = strlcpy(pname, dir, sizeof(pname));
5           if (n >= sizeof(pname))
6                   goto toolong;
7           if (strlcpy(pname + n, file, sizeof(pname) - n) >= sizeof(pname) - n)
8                   goto toolong;
```

However, one may question the validity of such optimizations, as they defeat the whole purpose of **strlcpy**() and **strlcat**().
As a matter of fact, the first version of this manual page got it wrong.

SEE ALSO

snprintf(3), strncat(3), strncpy(3)

HISTORY

The **strlcpy**() and **strlcat**() functions first appeared in OpenBSD 2.4, and made their appearance in FreeBSD 3.3.