Project 1

This project marks the beginning of your training to become the software engineer.

Introduction

The **libvc** project builds on the concept you learned during the course. You will create a library of useful functions that you've created so far to reuse in most of your C projects. C programming can be very tedious when you don't have acces to those highly useful C standard functions. This project makes you take the time to re-implement those functions, understand them, and learn to use them.

General Instructions

- You must create the following functions in the order you believe makes most sense. You can use the functions you have already coded to implement the next ones. The difficulty level does not increase by assignment and the project has not been structured in any specific way.
- Your code must be written in accordance with the code standard.
- Your functions should not quit unexpectedly (segmentation fault, bus error, double free, etc) apart from undefined behaviors. If this happens, your project will be considered non functional and will receive a 0 point.
- All heap allocated memory space must properly freed **when necessary**.
- You must submit a C file for each function you create, as well as **libvc.h** file, which will contain all necessary prototypes as well as **macros** and **typedef**s you might need. All those files must be at the root of your repository.
- You must submit a **Makefile** which will compile your source files to a static library **libvc.a**.
- Only the following **libc** functions are allowed : malloc(3), free(3) and write(2), putchar(3) and their usage is restricted. See below.
- You must include the necessary include system files to use one or more of the three authorized functions in your .c files. The only additional system include file you are allowed to use is string.h to have access to the constant NULL and to the type size_t. Everything else if forbidden.
- It will be helpful for you to create test programs for your library even though this work won't have to be submitted and won't be graded. It will give you a chance to easily test your work.

Mandatory part

Technical considerations

- Your **libvc.h** file can contain *macro*s and *typedef*s if needed.
- A string must **ALWAYS** end with a '\0', even if it is not included in the function's description, unless explicitly stated otherwise.
- It is forbidden to use global variables.
- If you need helper-functions to write a complex function, you must define these subfunctions as **static**.
- You must pay attention to your types and wisely use the casts when needed, especially when a void* type is involved. Generally speaking, avoid implicit casts. Example:

```
char *str;

str = malloc(42 * sizeof(*str)); /* Wrong ! Malloc returns a void *
(implicit cast) */
str = (char *) malloc(42 * sizeof(*str)); /* Right ! (explicit cast) */
```

Part 1 - libc standard functions

In this first part, you must re-implement a set of the libc functions, as defined in their **man**. Your functions will need to present the same prototype and behaviors as the originals. Your functions' names must be prefixed by "vc_". For instance strlen becomes vc_strlen.

Functions					
-	memset	bzero	memcpy	memccpy	memmove
-	memchr	memcmp	strlen	strdup	strcpy
-	strncpy	strcat	strncat	strlcat	strchr
-	strrchr	strstr	strnstr	strcmp	strncmp
-	atoi	isalpha	isdigit	isalnum	isascii
-	isprint	toupper	tolower	puts(vc_putstr)	

Part 2 - Additional functions

In this second part, you must code a set of functions that are either not included in the **libc**, or included in a different form. Some of these functions can be useful to wrtie Part 1 functions.

1

vc_memalloc	
Prototype	void * vc_memalloc(size_t size);
Description	Allocates (with malloc(3)) and returns a "fresh" memory area. The memory allocated is initialized to 0. If the allocation fails, the function returns NULL
Param # 1	The size of memory that needs to be allocated.
Return	The allocated memory area
libc	malloc(3)

2

vc_memdel	
Prototype	void vc_memdel(void **ap);

vc_memdel	
Description	Takes as a parameter the address of a memory area that needs to be freed with free(3), then puts the pointer to NULL
Param # 1	A pointer’s address that needs its memory freed and set to NULL.
Return	None
libc	free(3)

3

vc_strnew	
Prototype	char *vc_strnew(size_t size);
Description	Allocates (with malloc(3)) and returns a “fresh” string ending with ’\0’. Each character of the string is initialized as ’\0’. If the allocation fails the function returns NULL.
Param # 1	The size of the string to be allocated.
Return	The string allocated and initialized to 0.
libc	malloc(3)

4

vc_strdel	
Prototype	void vc_strdel(char **as);
Description	Takes as a parameter the address of a string that need to be freed with free(3), then sets its pointer to NULL.
Param # 1	The string’s address that needs to be freed and its pointer set to NULL.
Return	None
libc	free(3)

5

vc_strclr	
Prototype	void vc_strclr(char *s);
Description	Sets every character of the string to the value ’\0’.
Param # 1	The string that needs to be cleared.
Return	None
libc	None

6

vc_striter	
Prototype	void vc_striter(char *s, void (*f)(char *));
Description	Applies the function f to each character of the string passed as argument. Each character is passed by address to f to be modified if necessary.
Param # 1	The string to iterate.
Param # 2	The function to apply to each character of s.
Return	None
libc	None

7

vc_strmap	
Prototype	char *vc_strmap(char const *s, void (*f)(char));
Description	Applies the function f to each character of the string given as argument to create a “fresh” new string (with malloc(3)) resulting from the successive applications of f.
Param # 1	The string to map.
Param # 2	The function to apply to each character of s.
Return	The "fresh" string created from the successive applications of f.
libc	None

8

vc_strsub	
Prototype	char *vc_strsub(char const *s, size_t start, size_t len);
Description	Allocates (with malloc(3)) and returns a “fresh” substring from the string given as argument. The substring begins at start and is of size len. If start and len aren’t refering to a valid substring, the behavior is undefined. If the allocation fails, the function returns NULL.
Param # 1	The string from which create the substring.
Param # 2	The start index of the substring.
Param # 3	The size of the substring.
Return	The substring.
libc	malloc(3)

9

vc_strjoin	
Prototype	char *vc_strjoin(char const *s1, char const *s2);
Description	Allocates (with malloc(3)) and returns a “fresh” string ending with ’\0’, result of the concatenation of s1 and s2. If the allocation fails the function returns NULL.
Param # 1	The prefix string
Param # 2	The suffix string
Return	The “fresh” string result of the concatenation of the 2 strings.
libc	malloc(3)

10

vc_strtrim	
Prototype	char *vc_strtrim(char const *s);
Description	Allocates (with malloc(3)) and returns a copy of the string given as argument without whitespaces at the beginning or at the end of the string. Will be considered as whitespaces the following characters ’ ’, ’\n’ and ’\t’. If s has no whitespaces at the beginning or at the end, the function returns a copy of s. If the allocation fails the function returns NULL.
Param # 1	The string to be trimed.
Return	The “fresh” trimmed string or a copy of s.
libc	malloc(3)

11

vc_strsplit	
Prototype	char **vc_strsplit(char const *s, char c);
Description	Allocates (with malloc(3)) and returns an array of “fresh” strings (all ending with ’\0’, including the array itself) obtained by spliting s using the character c as a delimiter. If the allocation fails the function returns NULL. Example: vc_strsplit("*hello*fellow***students*", ’*’) returns the array ["hello", "fellow", "students"].
Param # 1	The string to split.
Param # 2	The delimiter character.
Return	The array of "fresh" strings result of the split.
libc	malloc(3), free(3)

12

vc_itoa	

vc_itoa	
Prototype	char *vc_itoa(int n);
Description	Allocate (with malloc(3)) and returns a “fresh” string ending with ’\0’ representing the integer n given as argument. Negative numbers must be supported. If the allocation fails, the function returns NULL.
Param # 1	The integer to be transformed into a string.
Return	The string representing the integer passed as argument.
libc	malloc(3)

13

vc_putnbr	
Prototype	void vc_putnbr(int n);
Description	Outputs the integer n to the standard output.
Param # 1	The integer to output
Return	None.
libc	write(2)

14

vc_putchar	
Prototype	void vc_putchar(char c);
Description	Outputs the character c to the standard output.
Param # 1	The character to output
Return	None.
libc	write(2)

15

vc_putendl	
Prototype	void vc_putendl(char const *s);
Description	Outputs the string s to the standard output followed by '\n'
Param # 1	The string to output
Return	None.
libc	write(2)

Submission

Submit your work with your Github repository.