Unix System

filter

Contents

Administrative details

- Your sources shall be turned-in on the `PSU_year_filter` directory
  ex: PSU_2013_filter for the 2013-2014 scolar year
- Your Makefile must create your libmy which must contain the filter function.

`The norm will be checked in every file of your library`

Subject

- Write a function used to read and filter the content of a file.
- Your function must be named filter and shall be prototyped as such :

```
char *filter(char *filename, char *format, ...);
```

- The first param will be the name of a file. You must open this file and read its content.
- The second param will be a format string. This string will give you a list of elements to filter in your file (see under, "Formating")
- The following arguments will be a list of elements to filter. Filtering will be done by removing or editing elements in the file's content.

```
$> cat hunter
Un chasseur sachant chasser sans son chien est un chasseur qui sait chasser
$> cat numbers
-10 -5 0 5 10
$>
```

```
int main()
{
        my_putstr(filter("hunter", ""));
        my_putstr(filter("hunter", "s", "chass"));
}
```

```
$> cc main.c -lmy -L.
$> ./a.out
Un chasseur sachant chasser sans son chien est un chasseur qui sait chasser
Un eur sachant er sans son chien est un eur qui sait er.
$>
```

- A complete list of what can be edited or removed can be found in the "formating" section
- Variadic args given to your function will always be correct in respect to the formating string
- If the formating string is empty, file content must be returned without any change (see example above)

- If any error was to happen, your function must return null. There is never any reason for a function to exit a program.

Formating

A formating string is made of multiple characters. Every single character indicate an element to remove or edit, picked in the following list. Any unrecognized character must be ignored. Any formating character present in this document (whether it is in the "formating" or the "additional features" section) will be linked to the according parameter. If your function does not accept this character, you can ignore it but you must take care of the corresponding argument to avoid an error in your function).

The following list of formating characters are **mandatory** in your project :

- **c** indicate a character to remove

```
$> cat main.c
int main()
{
        my_putstr(filter("hunter", "c", 'c'));
}
$> cc main.c -lmy -L.
$> ./a.out
Un hasseur sahant hasser sans son hien est un hasseur qui sait hasser
$>
```

- **s** indicate a character string to remove

```
$> cat main.c
int main()
{
        my_putstr(filter("hunter", "s", "chasseur"));
}
$> cc main.c -lmy -L.
$> ./a.out
Un  sachant chasser sans son chien est un  qui sait chasser
$>
```

- **i** indicate an integer to remove

```
$> cat main.c
int main()
{
        my_putstr(filter("numbers", "i", 5));
}
$> cc main.c -lmy -L.
$> ./a.out
-10 -5 0  10
$>
```

- **C** and **S** are equivalent to **c** and **s** but are not case-sensitive
- **I** is equivalent to **i** but affect equally positive and negative version of a number

```
$> cat main.c
int main()
{
        my_putstr(filter("numbers", "I", 5));
}
$> cc main.c -lmy -L.
$> ./a.out
-10  0  10
$>
```

- Of course, you may use multiple format character together. Filter will then be applied in the same order than the characters are, each filter using the result of the previous one.

```
$> cat main.c
int main()
{
        my_putstr(filter("hunter", "Ccs", 'C', 'u', "er"));
}
$> cc main.c -lmy -L.
$> ./a.out
Un hass sahant hass sans son hien est n hass qi sait hass
$>
```

Additionnal features

You may add the following format character as additionnal features. Note that even if you choose not to do so, your function must handle them (and their arguments) safely, as descrive in the previous section. Of course, mandatory format character must all be present and work perfectly well for those ones to be taken in account.

- **o**, **x** and **b** will respectivly convert the given number to octal, hexadecimal and binary bases.

```
$> cat main.c
int main()
{
        my_putstr(filter("numbers", "x", 10));
}
$> cc main.c -lmy -L.
$> ./a.out
-10 -5 0 5 A
$>
```

- **O**, **X** and **B** are identical to **o**, **x** and **b**, but are work on both positive and negative version of the number (same as **i** and **I**)

- When a number is present in a format string, it let you indicate a specific occurence of the given parameter to filter. As such, only the given version can be filtered. If there is no such occurence in the file, the filter is ignored.

```
$> cat main.c
int main()
{
        my_putstr(filter("hunter", "3c", 'c'));
}
$> cc main.c -lmy -L.
$> ./a.out
Un chasseur sachant hasser sans son chien est un chasseur qui sait chasser
$>
```

Allowed functions

- fopen
- fclose
- getline
- malloc
- free
- man 3 stdarg