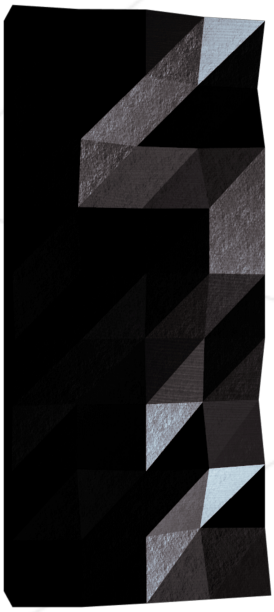

Proiect al modulului Algoritmica I

libft

Staff Academy+Plus contact@academyplus.ro

Sumar: Acest proiect are ca obiectiv sa scrieti o biblioteca de functii uzuale pe care le veti putea folosi in proiectele voastre.

Cuprins

Capitolul I

Preambul

Acest prim proiect marcheza debutul vostru in formarea ca programator. Profitati si cititi: acest articol si invatati incepand de azi ca tipurile diferentiaza un programator de o fiara. Daca nu intelegeti tot nu e grav. Asta vine cu timpul.

Pentru a va acompania muzical pe toata perioada acestui proiect va propunem o lista de formatii demne de interes. Daca nu va place inseamna ca aveti gusturi indoielnice in materie, dar probabil ca aveti alte calitati cum ar fi sa aveti numerosi prieteni pe Facebook sau sa puteti sa va atingeti cotul cu limba. Pe scurt, formatiile sunt listate intr-o ordine arbitrara si nu e exhaustiva. Aceasta va este data cu titlu de exemplu si sunteti incurajati sa explorati bogata lor discografie.

- Between The Buried And Me
- Tesseract
- Chimp Spanner
- Emancipator
- Cynic
- Kalisia
- Porcupine Tree
- Wintersun
- O.S.I
- Dream Theater
- Pain Of Salvation
- Crucified Barbara

Capitolul II

Subiect

II.1 Parte obligatorie

Proiectul `libft` reia conceptul din ziua 06 a piscinei, ca sa faceti o biblioteca de functii utile pe care sa le puteti apoi folosi in majoritatea proiectelor vostre de `C` din acest an si pentru a castiga mai mult timp. Acest proiect va solicita sa scrieti mult cod pe care l-ati scris deja pe parcursul piscinei, lucru ce reprezinta un bun prilej sa reluati aceasta tema, in conditiile in care, practic, nu veti invata nimic nou. Vedeti acest proiect ca reprezentand momentul in care selectati sau echipati personajul vostru dintr-un joc video.

Functiile le veti face in ordinea in care doriti si sunteti incurajati sa le folositi pe cele deja create pentru realizarea celorlalte. Dificultatea nu va creste si ordinea din subiect e arbitrara. Seamana putin cu un joc video in care puteti realiza cautari in ordinea in care doriti si sa renuntati la cele precedente pentru a le facilita pe cele ceurmeaza.

II.1.1 Partea 1 - Functii ale libc

In aceasta prima parte va trebui sa rescrieti un ansamblu de functii ale librariei `libc` precum sunt descrise in `man`-ul lor respectiv pe sistemul vostru. Functiile trebuie sa aiba exact acelasi prototip si acelasi comportament ca cele originale. Numele lor trebuie sa aiba prefixul "`ft_`". De exemplu `strlen` devine `ft_strlen`.

Anumite prototipuri ale functiilor pe care le veti rescrie folosesc calificativul de tip "restrict". Acest cuvant cheie face parte din standardul c99; trebuie deci sa nu-l puneti in prototipul functiei si sa nu compilati cu flag-ul -std=c99.

Va trebui sa rescrieti urmatoarele functii:

- `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`

II.1.2 Partea 2 - Functii suplimentare

In aceasta parte trebuie sa scrieti un anumit numar de functii ce nu figureaza in `libc` sau se gasesc sub o forma diferita. Unele dintre aceste functii pot fi de interes pentru a facilita scrierea functiilor din prima parte.

ft_memalloc	
Prototip	`void * ft_memalloc(size_t size);`
Descriere	Alocare dinamica (cu `malloc(3)`) si returneaza o noua zona de memorie. Memoria alocata este initializata la `0`. In cazul in care alocarea nu reuseste, functia returneaza `NULL`.
Param. #1	Dimensiunea zonei de memorie alocata.
Valoare de retur	Adresa zonei de memorie alocata.
Functii libc	`malloc(3)`

ft_memdel	
Prototip	`void ft_memdel(void **ap);`
Descriere	Ia ca parametru adresa unui pointer, deci zona de memorie alocata dinamic trebuie sa fie eliberata cu `free(3)`, la sfarsitul functiei pointerul trebuie sa fie initializat la `NULL`.
Param. #1	Adresa pointerului la care trebuie sa eliberam memoria si sa initializam la `NULL`.
Valoare de retur	Nimic.
Fonctii libc	`free(3)`.

ft_strnew	
Prototip	`char * ft_strnew(size_t size);`
Descriere	Alocati (cu `malloc(3)`) returnati un nou sir de caractere terminat cu un `'\0'`. Fiecare caracter din sir este initializat la `'\0'`. In cazul in care alocarea nu reuseste, functia returneaza `NULL`.
Param. #1	Dimensiunea sirului de caractere de alocat.
Valoare de retur	Sirul de caractere alocat si inizializat la `0`.
Functii libc	`malloc(3)`

ft_strdel	
Prototip	`void ft_strdel(char **as);`
Descriere	Ia ca parametru adresa unui sir de caractere care trebuie sa fie eliberat cu `free(3)`; la sfarsitul functiei pointerul trebuie sa fie initializat la `NULL`.
Param. #1	Adresa pointerului spre zona de memorie cetrebuie eliberata si initializarea sa la valoarea `NULL`.
Valoare de retur	Nimic.
Functii libc	`Free(3)`.

ft_strclr	
Prototip	`void ft_strclr(char *s);`
Descriere	Atribuie valoarea `'\0'` la toate caracterele din sirul dat in parametru.
Param. #1	Sirul de caractere de sters.
Valoare de retur	Nimic.
Functii libc	Niciuna.

ft_striter	
Prototip	`void ft_striter(char *s, void (*f)(char *));`
Descriere	Aplica functia `f` la fiecare caracter din sirul de caractere trimis ca parametru. Adresa fiecarui caracter este trimisa ca parametru la functia `f` pentru a putea sa o modifice in caz de nevoie.
Param. #1	Sirul de caractere de iterat.
Param. #2	Functia de apelat asupra fiecarui caracter al sirului `s`.
Valoare de retur	Nimic.
Functii libc	Niciuna.

ft_striteri	
Prototip	`void ft_striteri(char *s, void (*f)(unsigned int, char *));`
Descriere	Aplica functia `f` la fiecare caracter din sirul de caractere trimis ca parametru. Adresa fiecarui caracter este trimisa ca al doilea parametru, precizand indexul ca prim parametru la functia `f` pentru a putea sa o modifice in caz de nevoie.
Param. #1	Sirul de caractere de iterat.
Param. #2	Functia de apelat asupra fiecarui caracter al sirului `s` si pe indexul sau.
Valoare de retur	Nimic.
Functii libc	Niciuna.

ft_strmap	
Prototip	`char * ft_strmap(char const *s, char (*f)(char));`
Descriere	Aplica functia `f` la fiecare caracter din sirul de caractere dat ca parametru, pentru a crea un nou sir (cu `malloc(3)`) care va avea in fiecare caracter din sir, caracterul resultat primit de la apelul functiei `f` asupra caracterului cu acelasi index din sir.
Param. #1	Sirul de caractere de iterat.
Param. #2	Functia de apelat asupra fiecarui caracter al sirului `s`.
Valoare de retur	Sirul nou rezultand de la aplicarea succesiva a functiei `f`.
Functii libc	`malloc(3)`

ft_strmapi	
Prototip	`char * ft_strmapi(char const *s, char (*f)(unsigned int, char));`
Descriere	Aplica functia `f` precizand indexul la fiecare caracter din sirul de caractere dat in parametru, pentru a crea un nou sir (cu `malloc(3)`) care va avea in fiecare caracter din sir, caracterul resultat primit de la apelul functiei `f` asupra caracterului cu acelasi index din sir.
Param. #1	Sirul de caractere asupra caruia se face iteratia.
Param. #2	Functia de apelat asupra fiecarui caracter `s` precizand indexul sau.
Valoare de retur	Noul sir rezultat in urma aplicarilor succesive ale functiei `f`.
Functii libc	`malloc(3)`

ft_strequ	
Prototip	`int ft_strequ(char const *s1, char const *s2);`
Descriere	Compara lexical `s1` si `s2`. Daca aceste doua siruri sunt egale functia va returna `1`, daca nu `0`.
Param. #1	Primul sir de comparat.
Param. #2	Al doilea sir de comparat.
Valoare de retur	`1` sau `0` In functie de siruri daca sunt egale sau nu.
Functii libc	Niciuna.

ft_strnequ	
Prototip	`int ft_strnequ(char const *s1, char const *s2, size_t n);`
Descriere	Compara lexical `s1` si `s2` pana la `n` caractere sau pana cand un `'\0'` a fost gasit. Daca aceste doua siruri sunt egale functia va returna `1`, daca nu `0`.
Param. #1	Primul sir de comparat.
Param. #2	Al doilea sir de comparat.
Param. #3	Numarul maxim de caractere de comparat.
Valoare de retur	`1` sau `0` In functie de siruri daca sunt egale sau nu.
Functii libc	Niciuna.

ft_strsub	
Prototip	`char * ft_strsub(char const *s, unsigned int start, size_t len);`
Descriere	Aloca (cu `malloc(3)`) si returneaza o copie de la o parte din sirul trimis ca parametru. Copia va incepe la indexul `start` si are ca lungime `len`. Daca `start` si `len` nu delimiteaza un sir valid, comportamentul este nedeterminat. In cazul in care alocarea nu reuseste, functia returneaza `NULL`.
Param. #1	Sirul de caractere unde trebuie sa cautati partea de copiat.
Param. #2	Indexul de unde incepem sa copiem sirul.
Param. #3	Lungimea partii de sir de copiat.
Valoare de retur	Sirul copiat.
Functii libc	`malloc(3)`

ft_strjoin	
Prototip	`char * ft_strjoin(char const *s1, char const *s2);`
Descriere	Aloca (cu `malloc(3)`) si returneaza un sir de caractere nou, delimitandu-i sfarsitul cu `'\0'` acest sir va contine sirulrile concatenate `s1` si `s2`. Daca alocarea nu reuseste, functia va returna `NULL`.
Param. #1	Sirul de caractere prefix.
Param. #2	Sirul de caractere postfix.
Valoare de retur	Noul sir resultand din concatenarea sirului s1 cu s2.
Fnctii libc	`malloc(3)`

ft_strtrim	
Prototip	`char * ft_strtrim(char const *s);`
Descriere	Aloca (cu `malloc(3)`) si returneaza o copie a sirului transmis in parametru, fara spatii albe “whitespaces” la inceput si la sfarsitul sirului. Consideram ca spatii albe caracterele urmatoare: `' '`, `'\n'` si `'\t'`. Daca `s` nu contine spatii albe, la inceputul si la sfarsitul sirului, functia va transmite o copie a sirului `s`. Daca alocarea nu reuseste, functia va returna `NULL`.
Param. #1	Sirul de caractere ce trebuie decupat.
Valoare de retur	Noul sir de caractere decupat sau o copie al sirului `s`.
Functii libc	`malloc(3)`

ft_strsplit	
Prototip	`char ** ft_strsplit(char const *s, char c);`
Descriere	Aloca (cu `malloc(3)`) si returneaza un nou tablou de caractere (toate sirurile se vor termina cu un `'\0'`, deci si tabloul) care desparte in cuvinte un sir de caractere `s` in functie de un alt caracter `c`. Daca alocarea nu reuseste, functia va returna `NULL`. Exemplu: `ft_strsplit("*salut*les***etudiants*", '*')` trimite tabloul `["salut", "les", "etudiants"]`.
Param. #1	Sirul de despartit in cuvinte.
Param. #2	Caracterul delimitator.
Valoare de retur	Noul tablou de siruri de caractere rezultand din decuparea sirului `s`.
Functii libc	`malloc(3)`

ft_itoa	
Prototip	`char * ft_itoa(int n);`
Descriere	Aloca (cu `malloc(3)`) si returneaza un nou sir de caractere care se va termina cu un `'\0'` raprezentand intregul `n` trimis ca parametru. Trebuie gestionate si numerele negative. Daca alocarea nu reuseste, functia va returna `NULL`.
Param. #1	Intregul care trebuie convertit intr-un sir de caractere.
Valoare de retur	Sirul de caractere care reprezinta intregul transmis ca parametru.
Functii libc	`malloc(3)`

ft_putchar	
Prototip	`void ft_putchar(char c);`
Descriere	Afiseaza caracterul `c` la iesirea standard.
Param. #1	Caracterul de afisat.
Valoare de retur	Niciuna.
Functii libc	`write(2)`.

ft_putstr	
Prototip	`void ft_putstr(char const *s);`
Descriere	Afiseaza sirul de caractere `s` la iesirea standard.
Param. #1	Sirul de caractere de afisat.
Valoare de retur	Niciuna.
Functii libc	`write(2)`.

ft_putendl	
Prototip	`void ft_putendl(char const *s);`
Descriere	Afiseaza sirul de caractere `s` la iesirea standard urmat de un `'\n'`.
Param. #1	Sirul de caractere de afisat.
Valoare de retur	Niciuna.
Fonctii libc	`write(2)`.

ft_putnbr	
Prototip	`void ft_putnbr(int n);`
Descriere	Afiseaza intregul `n` la iesirea standard.
Param. #1	Intregul de afisat.
Valoare de retur	Niciuna.
Functii libc	`write(2)`.

ft_putchar_fd	
Prototip	`void ft_putchar_fd(char c, int fd);`
Descriere	Scrie caracterul `c` intr-un descriptor de fisier `fd`.
Param. #1	Caracterul de scris.
Valoare de retur	Niciuna.
Functii libc	`write(2)`.

ft_putstr_fd	
Prototip	`void ft_putstr_fd(char const *s, int fd);`
Descriere	Scrie sirul de caractere `s` intr-un descriptor de fisier `fd`.
Param. #1	Sirul de caracterere de scris.
Valoare de retur	Niciuna.
Fonctii libc	`write(2)`.

ft_putendl_fd	
Prototip	`void ft_putendl_fd(char const *s, int fd);`
Descriere	Scrie sirul de caractere `s` intr-un descriptor de fisier `fd` urmat de un `'\n'`.
Param. #1	Sirul de caracterere de scris.
Valoare de retur	Niciuna.
Fonctii libc	`write(2)`.

ft_putnbr_fd	
Prototip	`void ft_putnbr_fd(int n, int fd);`
Descriere	Scrie intregul `n` intr-un descriptor de fisier `fd`.
Param. #1	Intergul de scris.
Valoare de retur	Niciuna.
Fonctii libc	`write(2)`.

II.2 Bonus

Daca ati reusit perfect partea obligatorie, in aceasta sectiune va propunem un mod pentru a avansa mai departe. Exact cum ai cumpara un pachet de extensie la un joc video. Bonusurile vor fi luante in considerare doar daca obtineti cel putin un scor de `18/20` la partea obligatorie.

Daca ai niste functii de gestiunea memoriei si a sirurilor de caractere este foarte convenabil, dar va veti da rapid seama ca daca aveti si niste functii de de procesare a listelor ar fi si mai convenabil.

Veti utiliza urmatoarea structura pentru a raprezenta nodurile listei. Aceasta structura trebuie sa fie adaugata in fisierul vostru `libft.h`.

```
typedef struct      s_list
{
    void            *content;
    size_t          content_size;
    struct s_list   *next;
}                   t_list;
```

Descrierea campurilor de la structura `t_list` este urmatoarea:

- `content`: Datele sunnt continute intr-un nod. Tipul `void *` permite stocarea oricarui tip de variabila.
- `content_size`: Dimensiunea datelor stocate. Tipul `void *` nu permite cunoasterea dimensiunii datelor vizate, deci necesita sa o salvam. Exemplu: sirul de caractere `"42"` are o dimensiune de `3 octeti` si intregul 32 biti `42` are o dimensiune de `4 octeti`.
- `next`: Adresa urmatorului nod din lista sau valoarea `NULL` daca este ultimul nod.

Urmatoarele functii va vor permite manipularea usoara a listelor.

ft_lstnew	
Prototip	`t_list * ft_lstnew(void const *content, size_t content_size);`
Descriere	Aloca (cu `malloc(3)`) si returneaza un nou nod. Campurile `content` si `content_size` din noul nod sunt initializate prin **copia** parametrelor functiei. Daca parametrul `content` este nul, campul `content` este initializat la `NULL` si campul `content_size` este initializat la `0` ne tinand cont de valoarea parametrului `content_size`. Campul `next` este initializat la `NULL`. Daca alocarea nu reuseste, functia va returna `NULL`.
Param. #1	Continutul de adaugat in noul nod.
Param. #2	Dimensiunea continutului de adaugat la urmatorul nod.
Valoare de retur	Noul nod.
Functii libc	`malloc(3)`

ft_lstdelone	
Prototip	`void ft_lstdelone(t_list **alst, void (*del)(void *, size_t));`
Descriere	Ia ca parametru adresa la pointer la un nod si elibereaza memoria continutului acestui nod cu functia `del` transmisa ca parametru si apoi eliberaeaza memoria nodului cu functia `free(3)`. Memoria campului `next` nu trebuie in niciun caz sa fie eliberata. La sfarsit, functia va pune pointerul la nod la valoarea `NULL` (asemanator cu functia `ft_memdel` din partea obligatrie).
Param. #1	Adresa unui pointer la un nod de eliberat.
Valoare de retur	Niciuna.
Functii libc	`free(3)`

ft_lstdel	
Prototip	`void ft_lstdel(t_list **alst, void (*del)(void *, size_t));`
Descriere	Ia ca parametru adresa unui pointer la un nod si elibereaza memoria continutului acestui nod cu functia `del` transmisa ca parametru si apoi eliberaeaza memoria nodului cu functia `free(3)`. (acest procedeu se repeta pentru toti succesori nodului primit ca parametru) Memoria campului `next` nu trebuie in niciun caz sa fie eliberata. La sfarsit, functia va pune pointerul la nod la valoarea `NULL` (asemanator cu functia `ft_memdel` din partea obligatrie).
Param. #1	Adresa pointerului la primul nod din lista de eliberat.
Valoare de retur	Niciuna.
Functii libc	`free(3)`

ft_lstadd	
Prototip	`void ft_lstadd(t_list **alst, t_list *new);`
Descriere	Adauga un element `new` in capul listei.
Param. #1	Adresa unui pointer la primul element al listei.
Param. #2	Nodul de adaugat ca prim element al acestei liste.
Valoare de retur	Niciuna.
Functii libc	Niciuna.

ft_lstiter	
Prototip	`void ft_lstiter(t_list *lst, void (*f)(t_list *elem));`
Descriere	Parcurge lista `lst` aplicand la fiecare nod functia `f`.
Param. #1	Pointerul la primul nod al unei liste.
Param. #2	Adresa unei functii ce va fi aplicata asupra tuturor nodurilor de la lista luata ca parametru.
Valoare de retur	Niciuna.
Fonctii libc	Niciuna.

ft_lstmap	
Prototip	`t_list * ft_lstmap(t_list *lst, t_list * (*f)(t_list *elem));`
Descriere	Parcurge lista `lst` si aplica fiecaui nod functia `f` si creaza o noua lista cu ajutorul `malloc(3)` rezultand de la aplicarile succesive. Daca o alocare esueaza, functia returneaza `NULL`.
Param. #1	Pointerul pe primul nod a unei liste.
Param. #2	Adresa unei functii ce va fi aplicata pe toate nodurile listei luata in parametru pentru a crea o noua lista.
Valoare de retur	Noua lista.
Fonctii libc	`malloc(3)`

Daca reusiti sa faceti perfect partea obligatorie si partea bonus, sunteti incurajati sa adaugati si alte functii care vi se par utile, pentru a mari biblioteca. Daca cel care va corecteaza le considera pertinente, puteti primi puncte suplimentare. Exemple: o versiune de `ft_strsplit` ce returneaza o lista de siruri in locul unui tablou de siruri; functia `ft_lstfold` similara functiei `reduce` din `Python` si functiei `List.fold_left` din `OCaml` (atentie la memory leak-uri!), functii de manipulare a tablourilor, stivelor, cozilor, maps, tabelelor de dispersie (hash tables) etc. Limita este doar imaginatia vostra.

II.3 Livrare

- Trebuie sa livrati, in radacina directorului vostru de lucru/livrare, un fisier `auteur` continand login-ul vostru urmat de '\n':

```
$>cat -e auteur
xlogin$
```

- Trebuie sa livrati un fisier in C pentru fiecare functie ce trebuie facuta precum si un fisier `libft.h` ce va contine toate prototipurile ca si `macro`-urile si `typedef`-urile de care puteti avea nevoie. Toate aceste fisiere trebuie sa se gaseasca in racacina directorului vostru de lucru/livrare.

- Va trebui sa livrati un fisier `Makefile` care va compila sursele vostre cu ajutorul unei biblioteci statice numita `libft.a`.

- Fisiserul `Makefile` trebuie sa contina cel putin regulile `$(NAME)`, `all`, `clean`, `fclean` si `re` in ordinea in care vi se pare cea mai adecvata.

- Fisierul `Makefile` trebuie sa compileze proiectul impreuna cu flagurile de compilare `-Wall`, `-Wextra` si `-Werror`.

- Pentru a va facilita sustinerea, trebuie de asemenea sa realizati unul sau mai multe programe de test pentru biblioteca voastra. Bine ca aceasta munca nu trebuie **nu trebuie livrat pe directorul de lucru/livrare si nu va fi evaluat**; puteti sa va testati usor munca si pe cea a celui pa care il veti verifica la sustinere. Sunteti liberi sa va folositi testele sau cele ale celor care sustin si chiar ambele, daca asta va face placere si logistica de baza va este la dispozitie. Nimic nu poate fi mai rau decat sa nu obtii toate punctele pe care le meritati la sustinere, deoarece corectorul nu a avut timp sa evalueze totul la timp, nu? E munca voastra, responsabilitatea voastra.

- Doar continutul ce se afla in directorul vostru de lucru/livrare va fi evaluat la sustinere.

II.4 Consideratii tehnice

- Fisierul vostru `libft.h` pate contine `macro`-uri si `typedef`-uri, conform nevoilor voastre.
- Un sir de caractere se termina **INTOTDEAUNA** cu un `'\0'`, chiar daca acesta este omis din descrietrea unei functii. In caz contrar, va fi indicat in mod explicit.
- Este interzissa se foloseasca variabilele globale.
- Daca aveti nevoie de functii auxiliare pentru scrierea unei functii complexe, trebuie sa definiti aceste functii auxiliare ca `static` in concordanta cu prevederile Normei.

Ca sa stiti ce e o functie statica este un bun inceput: http://codingfreak.blogspot.com/2010/06/static-functions-in-c.html

- Trebuie sa fiti atenti la tipurile pe care le folositi si sa folositi in mod judicios cast-ul atunci cand este necesar, in particular daca un tip `void *` este implicat. Evitati, in orice situatie, cast-urile implicite, indiferent de tipurile respective. Exemplu:

```
char    *str;

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

II.5 Functii autorizate

- malloc(3)
- free(3)
- write(2)

Trebuie bineinteles inclus si `include` de sistem, necesar pentu a fi folosit la una sau alta din cele 3 functii autorizate in fisierul `.c` respectiv. Singurul `include` de sistem pe care sunteti autorizat sa-l folositi in plus este `string.h` pentru a avea acces la constanta `NULL` si la tipul `size_t`. Orice altceva e interzis.

Capitolul III

Instructiuni

- Puteti scrie functiile in ordinea pe care o doriti si et ne pas réussir une fonction n'empêche pas d'avoir les points pour les suivantes si celles-ci sont bonnes.

- Sunteti incurajati sa faceti apel la functiile pe care le-ati scris deja, pentru a le scrie pe cele urmatoare.

- Proiectul trebuie sa fie scris conform Normei. Norminette nu va fi utilizata pentru verificarea Normei care se aplica, deci, pe ansamblu si va fi fi verificat de persoane umane in timpul sustinerii. Erorile de neconformitate cu Norma vor fi notate cu `0` la sustinere.

- In niciun caz functiile voastre nu trebuie sa se incheie intr-un mod neasteptat (Eroare de segmentare, eroare de magistrala, free dublu etc.) in afara de comportamentele nedeterminate. Proiectul va fi, in consecinta, considerat nefunctional si va primi nota `0` la sustinere.

- Toata memoria alocata va trebui eliberata in mod adecvat cand e necesar.

- Trebuie sa livrati, in radacina directorului vostru de lucru/livrare, un fisier `auteur` continand login-ul vostru urmat de '\n':

```
$>cat -e auteur
xlogin$
```

- Nu trebuie niciodata livrati un cod pe care nu l-ati scris voi insiva. In caz ca planeaza un dubiu in acest sens, veti invitat la o intalnire de rescriere a codului pentru a verifica bunele voastre intentii.

Capitolul IV

Notare

Proiectul `libft` se face timp de doua saptamani si are doi timpi.
In primul rand, Moulinette va trece din cand in cand peste livrarile voastre pentru a va evalua prima parte din cea obligatorie (pagina 4 - `libc`). Aceasta Moulinette nu va va trimite informatii detaliate, doar numarul de functii reusite/esuate. Daca vom considera ca un numar suficient de studenti au reusit sa realizeze toate functiile partii 1, vom dezvalui intregii promotii subiectul proiectului `get_next_line` ce trebuie sa se deruleze pe perioada acelorasi doua saptamani ca proiectul `libft`.

Asta are mai multe consecinte:

- Aveti nevoie de mai mult timp pentru a realiza partea 1, in afara de timpul necesar pentru ca sa realizati proiectul `get_next_line`.
- Nu veti avea informatii in legatura cu erorile detectate de Moulinette. Va trebui deci, sa va pregatiti propriile baterii de teste pe care va incurajam sa le partajati pentru a gasi voi insiva erorile si sa facilitati sustinerea.
- Nu veti avea niciun ajutor din partea staff-ului.

In al doilea rand, veti avea parte de o sustinere clasica la finalul proiectului pentru a va obtine nota. La aceasta sustinere, bonusurile nu vor fi luate in considerare decat daca obtineti cel putin 18 din 20 puncte pentru partea obligatorie. Optimizarea calitatii anumitor elemente de cod va fi evaluata si ar putea obtine puncte suplimentare in partea de bonus.

Partea obligatorie se puncteaza cu 20 puncte, iar cea bonus cu 22 puncte, pentru un total maxim de 42 puncte.

Succes tuturor!