



Libft

ձեր առաջին անձնական գրադարանը

*Հակիրճ. այս նախագծի նպատակը  $C$  գրադարան կոդավորելն է՝  
վերահսմամբավորելով սովորական ֆունկցիաները, որոնք թույլատրված  
են օգտագործել ձեր մյուս նախագծերում:*

# Ցանկ

<b>I</b>	<b>Ներածություն</b>	<b>2</b>
<b>II</b>	<b>Ընդհանուր ցուցումներ</b>	<b>3</b>
<b>III</b>	<b>Պարտադիր մաս</b>	<b>5</b>
III.1	Տեխնիկական նկատառումներ . . . . .	5
III.2	Մաս 1 - Libc ֆունկցիաներ . . . . .	6
III.3	Մաս 2 - Հավելյալ ֆունկցիաներ . . . . .	7
<b>IV</b>	<b>Բոնուսային մաս</b>	<b>12</b>

# Գլուխ I

## Ներածություն

Շրագրավորումը շատ հոգնեցուցիչ կարող է լինել, երբ ստանդարտ օգտակար ֆունկցիաները ծրագրավորողին հասանելի չեն: Այս նախագիծը ձեզ հնարավորություն է տալիս վերակոդավորել այդ ֆունկցիաները, հասկանալ դրանք ու սովորել ճիշտ օգտագործել: Այս գրադարանը ձեզ կօգնի ձեր բոլոր Շ նախագծերի վրա աշխատելիս:

Այս նախագծի օգնությամբ ձեզ հնարավորություն ենք տալիս ինքնուրույն ավելացնել ֆունկցիաների ցուցակը: Ժամանակ հատկացրե՛ք ու տարվա ընթացքում հարստացրե՛ք ձեր libft գրադարանը:

## Գլուխ II

# Ընդհանուր ցուցումներ

- Ձեր նախագիծը պետք է գրվի Norm-ին համապատասխան: Եթե ունեք բոնուսային ֆայլեր/ֆունկցիաներ, դրանք ևս ստուգվելու են Norm-ով, և Norm-ի սխալի դեպքում կստանաք 0 միավոր:
- Բացառությամբ չսահմանված վարքագծերի (*սեզմենտացիայի սխալ, bus սխալ, կրկնակի free* և այլն)՝ ձեր ֆունկցիաները չպետք է անսպասելի ավարտվեն: Եթե դա տեղի ունենա, նախագիծը կհամարվի ոչ ֆունկցիոնալ և գնահատման ժամանակ կստանաք 0 միավոր:
- Հատկացված դինամիկ բաշխվող հիշողությունն անհրաժեշտության դեպքում պետք է ամբողջությամբ պատշաճ կերպով ազատվի: Արտահոսքերն անընդունելի են:
- Եթե նյութը պահանջում է, ապա պետք է հանձնել Makefile, որը ձեր ելակետային ֆայլը կկազմարկի պահանջվող ելքի՝ -Wall, -Wextra և -Werror դրոշակներով, իսկ ձեր Makefile-ը չպետք է վերակապի:
- Ձեր Makefile-ն պետք է պարունակի առնվազն \$(NAME), all, clean, fclean և re կանոնները:
- Ձեր նախագծի հետ բոնուսներ հանձնելու համար պետք է Makefile-ում ներառել bonus կանոն, որը կավելացնի բոլոր հնարավոր վերնագրերը, գրադարանները կամ ֆունկցիաները, որոնք արգելված են նախագծի հիմնական մասում: Բոնուսները պետք է լինեն առանձին ֆայլում՝ \_bonus.{c/h}-ում: Պարտադիր և բոնուսային մասերի գնահատումը կատարվում է իրարից անկախ:
- Եթե նախագիծը թույլ է տալիս օգտագործել ձեր libft-ը, ապա պետք է պատճենեք դրա ելակետային ֆայլերը և համապատասխան Makefile-ը libft պանակում՝ իր համապատասխան Makefile-ով: Ձեր նախագծի Makefile-ը պետք է կազմարկի գրադարանը՝ օգտագործելով դրա Makefile-ը, իսկ հետո՝ կազմարկի նախագիծը:
- Խրախուսելի է ձեր նախագծի համար թեստային ծրագրերի ստեղծումը, որոնք, սակայն, **հաշվի չեն առնվի և չեն գնահատվի**: Դա հնարավորություն կընձեռնի հեշտությամբ թեստավորել ձեր և ձեր ընկերների աշխատանքը: Այս թեստերը

հատկապես օգտակար կլինեն պաշտպանության ժամանակ: Ավելին, պաշտպանության ընթացքում ազատորեն կարող եք օգտագործել ինչպես ձեր, այնպես էլ այն ուսանողների թեստերը, ում աշխատանքը գնահատում եք:

- Ձեր աշխատանքը պետք է ներառել հանձնարարված git պահոցում: Կգնահատվի միայն git պահոցի աշխատանքը: Եթե նախատեսվում է, որ Deepthought-ն է գնահատելու ձեր աշխատանքը, ապա դա կարվի ձեր ընկերների գնահատումից հետո: Եթե Deepthought-ի գնահատման ընթացքում աշխատանքի ցանկացած մասում որևէ սխալ ի հայտ գա, գնահատումը կընդհատվի:

## Գլուխ III

# Պարտադիր մաս

Ծրագրի անուն	libft.a
Հանձնվելիք Փայլեր	*.c, libft.h, Makefile
Ստեղծելի ֆայլ	Այո
Արտաքին ֆունկցիաներ	Մանրամասն՝ ներքևում:
Libft թույլատրված	Կիրառելի չէ
Նկարագրություն	Գրե՛ք ձեր անձնական գրադարանը, որը ներառում է ձեր դասընթացի համար ամենակարևոր ֆունկցիաները:

### III.1 Տեխնիկական նկատառումներ

- Գլոբալ փոփոխականների կիրառությունը արգելվում է:
- Եթե ենթաֆունկցիաներ պետք լինեն բարդ ֆունկցիաներ գրելու համար, այդ ենթաֆունկցիաները պետք է սահմանել որպես `static`՝ դրանք ձեր գրադարանի հետ հրապարակելուց խուսափելու համար: Լավ կլինի, եթե հետագա նախագծերի դեպքում էլ այդպես վարվեք:
- Բոլոր ֆայլերը պետք է հանձնել պահոցի արմատում:


### III.2 Մաս 1 - Libc ֆունկցիաներ

Այս հատվածում պետք է վերակոդավորել մի շարք `libc` ֆունկցիաներ՝ ինչպես սահմանված է `man`-ում: Ձեր ֆունկցիաները պետք է ներկայացված լինեն նույն նախատիպով ու բնութագրով ինչ նախնականը: Ձեր ֆունկցիաների անունները պետք է սկսվեն `«ft_»` -ով: Օրինակ՝ `strlen`-ը դառնում է `ft_strlen`:



Այս ֆունկցիաների նախատիպերը օգտագործում են "restrict" հատկանիշ բառը: Այս բանալի բառը c99 ստանդարտի մասն է: Հետևաբար, արգելվում է դրա օգտագործումը ձեր նախատիպերում ու դրա կազմարկումը -std=c99 դրոշակով:

Պետք է վերակողմադրել հետևյալ ֆունկցիաները: Այս ֆունկցիաները արտաքին ֆունկցիաներ չեն պահանջում:

- 
- `memset`
  - `bzero`
  - `memcpy`
  - `memccpy`
  - `memmove`
  - `memchr`
  - `memcmp`
  - `strlen`
  - `strncpy`
  - `strlcat`
  - `strchr`
- `strchr`
  - `strnstr`
  - `strncmp`
  - `atoi`
  - `isalpha`
  - `isdigit`
  - `isalnum`
  - `isascii`
  - `isprint`
  - `toupper`
  - `tolower`

Պետք է նաև վերակոդավորել հետևյալ ֆունկցիաները՝  
օգտագործելով “malloc” ֆունկցիան:

- calloc
- strdup

### III.3 Մաս 2 - Հավելյալ ֆունկցիաներ

Երկրորդ մասում պետք է գրել մի շարք ֆունկցիաներ, որոնք կա՛մ ներառված չեն libc-ում, կա՛մ ուրիշ ձևաչափով են ներառված: Այս ֆունկցիաներից մի քանիսը օգտակար կլինեն մաս 1-ի ֆունկցիաները գրելիս:

<b>Ֆունկցիայի անուն</b>	ft_substr
<b>Նախատիպ</b>	char *ft_substr(char const *s, unsigned int start, size_t len);
<b>Հանձնվելիք ֆայլեր</b>	-
<b>Պարամետրեր</b>	#1. տողը, որից պետք է ենթատող ստեղծել #2. 's' տողի ենթատողի առաջին մասը #3. ենթատողի առավելագույն երկարությունը
<b>Վերադարձի արժեք</b>	Ենթատող. տեղակայումը ձախողվելու դեպքում՝ NULL:
<b>Արտաքին ֆունկցիաներ</b>	malloc
<b>Նկարագրություն</b>	Տեղակայում է (malloc(3)-ի միջոցով) ու 's' տողից մի ենթատող է վերադարձնում: Ենթատողը սկսվում է «start»-ով և ունի ամենամեծ «len» չափ:

<b>Ֆունկցիայի անուն</b>	ft_strjoin
<b>Նախատիպ</b>	char *ft_strjoin(char const *s1, char const *s2);
<b>Հանձնվելիք ֆայլեր</b>	-
<b>Պարամետրեր</b>	#1. նախաձանցային տող #2. վերջաձանցային տող
<b>Վերադարձի արժեք</b>	Նոր տողը. տեղակայումը ձախողվելու դեպքում՝ NULL:
<b>Արտաքին ֆունկցիաներ</b>	malloc
<b>Նկարագրություն</b>	Տեղակայում է (malloc(3)-ի միջոցով) ու վերադարձնում նոր տող, որը 's1'-ի ու 's2'-ի համակցման արդյունքն է:



<b>Ֆունկցիայի անուն</b>	ft_strtrim
<b>Նախատիպ</b>	char *ft_strtrim(char const *s1, char const *set);
<b>Հանձնվելիք ֆայլեր</b>	-
<b>Պարամետրեր</b>	#1. տողը, որ պատրաստվում էք կտրել #2. այն նիշերի հղումները, որոնք պետք է կտրվեն
<b>Վերադարձի արժեք</b>	Կտրված տողը. տեղակայումը ձախողվելու դեպքում` NULL:
<b>Արտաքին ֆունկցիաներ</b>	malloc
<b>Նկարագրություն</b>	Տեղակայում է (malloc(3)-ի միջոցով) և վերադարձնում է 's1'-ը ` 'set'-ում նշված նիշերը տողի սկզբից ու վերջից ջնջած:

<b>Ֆունկցիայի անուն</b>	ft_split
<b>Նախատիպ</b>	char **ft_split(char const *s, char c);
<b>Հանձնվելիք ֆայլեր</b>	-
<b>Պարամետրեր</b>	#1. տողը, որ պետք է կտրել #2. բաժանարար նիշը
<b>Վերադարձի արժեք</b>	Կտրման արդյունքում ստացված նոր տողերի զանգված: Տեղակայումը ձախողվելու դեպքում` NULL:
<b>Արտաքին ֆունկցիաներ</b>	malloc, free
<b>Նկարագրություն</b>	Տեղակայում է (malloc(3)-ի միջոցով) և վերադարձնում է տողերի զանգված, որը ստացվում է 's' տողը կտրելով, որտեղ 'c' նիշը օգտագործվում է որպես բաժանարար: Չանգվածը պետք է ավարտվի NULL ցուցիչով:

<b>Ֆունկցիայի անուն</b>	ft_itoa
<b>Նախատիպ</b>	char *ft_itoa(int n);
<b>Հանձնվելիք ֆայլեր</b>	-
<b>Պարամետրեր</b>	#1. ամբողջ թիվը, որը պետք է փոխակերպվի
<b>Վերադարձի արժեք</b>	Ամբողջ թիվը ներկայացնող տողը: Տեղակայումը ձախողվելու դեպքում` NULL
<b>Արտաքին ֆունկցիաներ</b>	malloc
<b>Նկարագրություն</b>	Տեղակայում է (malloc(3)-ի միջոցով) և վերադարձնում է տող, որը ներկայացնում է որպես արգումենտ ստացած ամբողջ թիվը(integer): Պետք է հաջողությամբ մշակել բացասական թվերը:

<b>Ֆունկցիայի անուն</b>	ft_strmapi
<b>Նախատիպ</b>	char *ft_strmapi(char const *s, char (*f)(unsigned int, char));
<b>Հանձնվելիք ֆայլեր</b>	-
<b>Պարամետրեր</b>	#1. տողը, որում պետք է իտերացիա կատարել #2. ֆունկցիան, որ պետք է կիրառալ յուրաքանչյուր նիշի վրա
<b>Վերադարձի արժեք</b>	'f'-ի հաջորդական կիրառումից ստեղծված տողը: Տեղակայումը ձախողվելու դեպքում` NULL:
<b>Արտաքին ֆունկցիաներ</b>	malloc
<b>Նկարագրություն</b>	'f' ֆունկցիան կիրառում է 's' տողի բոլոր նիշերի վրա, որպեսզի ստեղծի նոր տող (malloc(3)-ի միջոցով)` 'f'-ի հաջորդական կիրառումից առաջացած:

<b>Ֆունկցիայի անուն</b>	ft_putchar_fd
<b>Նախատիպ</b>	void ft_putchar_fd(char c, int fd);
<b>Հանձնվելիք ֆայլեր</b>	-
<b>Պարամետրեր</b>	#1. որպես էլքային արժեք վերադարձվող նիշը #2. ֆայլի նկարագրիչը, որում պետք է գրել
<b>Վերադարձի արժեք</b>	Ոչ մի
<b>Արտաքին ֆունկցիաներ</b>	գրել
<b>Նկարագրություն</b>	Տրված ֆայլի նկարագրիչին վերադարձնում է 'c' նիշը՝ որպես էլքային արժեք:

<b>Ֆունկցիայի անուն</b>	ft_putstr_fd
<b>Նախատիպ</b>	void ft_putstr_fd(char *s, int fd);
<b>Հանձնվելիք ֆայլեր</b>	-
<b>Պարամետրեր</b>	#1. որպես էլքային արժեք վերադարձվող տողը #2. ֆայլի նկարագրիչը, որում պետք է գրել
<b>Վերադարձի արժեք</b>	Ոչ մի
<b>Արտաքին ֆունկցիաներ</b>	Գրել
<b>Նկարագրություն</b>	Նշված ֆայլի նկարագրիչին վերադարձնում է 's' տողը որպես էլքային արժեք:

<b>Ֆունկցիայի անուն</b>	ft_putendl_fd
<b>Նախատիպ</b>	void ft_putendl_fd(char *s, int fd);
<b>Հանձնվելիք ֆայլեր</b>	-
<b>Պարամետրեր</b>	#1. որպես էլքային արժեք վերադարձվող տողը #2. ֆայլի նկարագրիչը, որում պետք է գրել
<b>Վերադարձի արժեք</b>	None
<b>Արտաքին ֆունկցիաներ</b>	write
<b>Նկարագրություն</b>	Նշված ֆայլի նկարագրիչին վերադարձնում է 's' տողը՝ որին հաջորդում է նոր տող:

<b>Ֆունկցիայի անուն</b>	ft_putnbr_fd
<b>Նախատիպ</b>	void ft_putnbr_fd(int n, int fd);
<b>Հանձնվելիք ֆայլեր</b>	-
<b>Պարամետրեր</b>	#1. ամբողջ թիվը, որ պետք է մուտքագրել #2. ֆայլի նկարագրիչը, որում պետք է գրել
<b>Վերադարձի արժեք</b>	Ոչ մի
<b>Արտաքին ֆունկցիաներ</b>	գրել
<b>Նկարագրություն</b>	Տրված ֆայլի նկարագրիչին վերադարձնում է 'n' ամբողջ թիվը՝ որպես էլքային արժեք:

## Գլուխ IV

# Բոնուսային մաս

Եթե պարտադիր մասը հաջողությամբ ավարտեք, առաջ գնալը ձեզ հաճելի կլինի: Այս վերջին մասը ձեզ բոնուսային միավորներ կտա:

Հիշողությունն ու տողերը կառավարող ֆունկցիաներ ունենալը շատ օգտակար է, բայց շուտով կբացահայտեք, որ ցուցակներ կառավարելը էլ ավելի օգտակար է:

Ձեր ցուցակի տարրերը ներկայացնելու համար կօգտագործեք այս ստրուկտուրան: Ստրուկտուրան պետք է ավելացնել ձեր `libft.h` ֆայլին:

`make bonus`-ը բոնուսային ֆունկցիաները կավելացնի `libft.a` գրադարանում:

`_bonus`-ը `.c` ֆայլերին ու վերնագրին այս ֆայլում հարկավոր չէ ավելացնել : `_bonus`-ը ավելացրեք միայն այն ֆայլերին, որոնք ձեր սեփական բոնուսային ֆունկցիաներն են պարունակում:

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

Ահա `t_list` struct-ի դաշտերի բնութագրման օրինակ՝

- Տարրում պահպանված տվյալներ. `void *`-ը թույլ է տալիս պահպանել ցանկացած տեսակի տվյալներ:
- Հաջորդ տարրի հասցեն կամ `NULL`, եթե վերջին տարրն է:

Հետևյալ ֆունկցիաները ձեզ թույլ կտան հեշտությամբ օգտագործել ձեր ցուցակները:

<b>Ֆունկցիայի անուն</b>	ft_lstnew
<b>Նախատիպ</b>	t_list *ft_lstnew(void *content);
<b>Հանձնվելիք ֆայլեր</b>	-
<b>Պարամետրեր</b>	#1. նոր տարր ստեղծելու նյութ
<b>Վերադարձի արժեք</b>	Նոր տարր
<b>Արտաքին ֆունկցիաներ</b>	malloc
<b>Նկարագրություն</b>	Տեղակայում է (malloc(3)-ի միջոցով) և վերադարձնում նոր տարր: 'content' փոփոխականը սկզբնարժեքավորված է 'content' պարամետրի արժեքով: 'next' փոփոխականը սկզբնարժեքավորված է NULL-ով:

<b>Ֆունկցիայի անուն</b>	ft_lstadd_front
<b>Նախատիպ</b>	void ft_lstadd_front(t_list **lst, t_list *new);
<b>Հանձնվելիք ֆայլեր</b>	-
<b>Պարամետրեր</b>	#1. Ցուցիչի հասցեն փոխանցվում է ցուցակի առաջին հղմանը: #2. տարրի ցուցիչի հասցե, որ պետք է ավելացնել ցուցակին
<b>Վերադարձի արժեք</b>	Ոչ մի
<b>Արտաքին ֆունկցիաներ</b>	Ոչ մի
<b>Նկարագրություն</b>	Ցուցակի սկզբում 'new'-ին տարրեր է ավելացնում:

<b>Ֆունկցիայի անուն</b>	ft_lstsize
<b>Նախատիպ</b>	int ft_lstsize(t_list *lst);
<b>Հանձնվելիք ֆայլեր</b>	-
<b>Պարամետրեր</b>	#1. ցուցակի սկիզբ
<b>Վերադարձի արժեք</b>	ցուցակի երկարություն
<b>Արտաքին ֆունկցիաներ</b>	Ոչ մի
<b>Նկարագրություն</b>	Հաշվում է տարրերի քանակը տողում:

<b>Ֆունկցիայի անուն</b>	ft_lstlast
<b>Նախատիպ</b>	t_list *ft_lstlast(t_list *lst);
<b>Հանձնվելիք ֆայլեր</b>	-
<b>Պարամետրեր</b>	#1. ցուցակի սկիզբ
<b>Վերադարձի արժեք</b>	ցուցակի վերջին տարր
<b>Արտաքին ֆունկցիաներ</b>	Ոչ մի
<b>Նկարագրություն</b>	Վերադարձնում է ցուցակի վերջին տարրը:

<b>Ֆունկցիայի անուն</b>	ft_lstadd_back
<b>Նախատիպ</b>	void ft_lstadd_back(t_list **lst, t_list *new);
<b>Հանձնվելիք ֆայլեր</b>	-
<b>Պարամետրեր</b>	#1. Ցուցիչի հասցեն փոխանցում է ցուցակին: #2. Ցուցիչի հասցեն փոխանցում է ցուցակում ավելացվելիք տարրին:
<b>Վերադարձի արժեք</b>	Ոչ մի
<b>Արտաքին ֆունկցիաներ</b>	Ոչ մի
<b>Նկարագրություն</b>	'new' տարրը ավելացնում է ցուցակի վերջում :

<b>Ֆունկցիայի անուն</b>	ft_lstdelone
<b>Նախատիպ</b>	void ft_lstdelone(t_list *lst, void (*del)(void *));
<b>Հանձնվելիք ֆայլեր</b>	-
<b>Պարամետրեր</b>	#1. դատարկվելիք տարր #2. ֆունկցիայի հասցե, որ օգտագործվել է նյութը ջնջելու համար
<b>Վերադարձի արժեք</b>	Ոչ մի
<b>Արտաքին ֆունկցիաներ</b>	Դատարկ
<b>Նկարագրություն</b>	Մի տարր է վերցնում որպես պարամետր ու ջնջում է տարրի պարունակած հիշողությունը՝ օգտագործելով որպես պարամետր տրված 'del' ֆունկցիան: Դատարկում է տարրի հիշողությունը: 'next'-ի հիշողությունը չպետք է ջնջվի:

<b>Ֆունկցիայի անուն</b>	ft_lstclear
<b>Նախատիպ</b>	void ft_lstclear(t_list **lst, void (*del)(void *));
<b>Հանձնվելիք ֆայլեր</b>	-
<b>Պարամետրեր</b>	#1. դատարկվելիք տարր #2. ֆունկցիայի հասցե, որ օգտագործվել է նյութը ջնջելու համար
<b>Վերադարձի արժեք</b>	Ոչ մի
<b>Արտաքին ֆունկցիաներ</b>	Ոչ մի
<b>Նկարագրություն</b>	Չնջում ու դատարկում է տրված տարրը ու այդ տարրին հաջարդող յուրաքանչյուր տարր՝ օգտագործելով 'del' ֆունկցիան ու free(3)-ը: Ցուցակի ցուցիչը պետք է նշված լինի NULL: #2. այն ֆունկցիայի հասցեն, որ օգտագործվել է տարրի նյութը ջնջելու համար



<b>Ֆունկցիայի անուն</b>	ft_lstiter
<b>Նախատիպ</b>	void ft_lstiter(t_list *lst, void (*f)(void *));
<b>Հանձնվելիք ֆայլեր</b>	-
<b>Պարամետրեր</b>	#1. տարրին փոխանցված ցուցիչի հասցե #2.ցուցակը իտերացիա անելու համար օգտագործված ֆունկցիայի հասցեն
<b>Վերադարձի արժեք</b>	Ոչ մի
<b>Արտաքին ֆունկցիաներ</b>	Ոչ մի
<b>Նկարագրություն</b>	'l-ին' ցուցակը իտերացիա է անում ու կիրառում է 'f' ֆունկցիան յուրաքանչյուր տարրի պարունակության վրա:

Ձեր libft-ին կարող եք ավելացնել կամայական ֆունկցիա՝ ըստ ձեր հայեցողության: