



Libft

Kendi yazdığınız ilk kütüphane

*Özet: Bu projenin amacı bir sonraki projelerinizde de kullanabileceğiniz bir C kütüphanesi yazmaktır.*

*Versiyon: 15*

# İçindekiler

<b>I</b>	<b>Giriş</b>	<b>2</b>
<b>II</b>	<b>Genel Talimatlar</b>	<b>3</b>
<b>III</b>	<b>Zorunlu Kısım</b>	<b>5</b>
III.1	Teknik Hususlar . . . . .	5
III.2	Bölüm 1 - Libc Fonksiyonları . . . . .	6
III.3	Bölüm 2 - Ekstra Fonksiyonlar . . . . .	7
<b>IV</b>	<b>Bonus Kısım</b>	<b>11</b>

# Bölüm I

## Giriş

C programlama çok kullanışlı olan standart fonksiyonlara erişim olmadığı durumlarda çok sıkıcı olabilmektedir. Bu projede çok kullanışlı olan fonksiyonları tekrardan yazacak, çalışma mantıklarını anlayacak, bunları kullanmayı öğreneceksiniz. Baştan yazmış olduğunuz bu kütüphane gelecekteki C projeleriniz için çok faydalı olacaktır.

Bütün bir yıl boyunca `libft` projenizi geliştireceksiniz, eklemeler yapacaksınız. Bu süreçte hangi fonksiyonları kullanmaya izniniz olduğunu kontrol etmeyi unutmayın!

# Bölüm II

## Genel Talimatlar

- Projeleriniz C programlama dilinde yazılmalıdır.
- Projeleriniz Norm'a uygun olarak yazılmalıdır. Bonus dosyalarınız/fonksiyonlarınız varsa, bunlar norm kontrolüne dahil edilir ve bu dosyalarda norm hatası varsa 0 alırsınız.
- Tanımlanmamış davranışlar dışında sizin fonksiyonlarınız beklenmedik bir şekilde sonlanmamalıdır (Segmentasyon hatası, bus hatası, double free hatası, vb.) . Eğer bunlar yaşanır s 0 alırsınız.
- Heap'de ayırmış olduğunuz hafıza adresleri gerekli olduğu durumlarda serbest bırakılmalıdır. Hiçbir istisna tolere edilmeyecektir.
- Eğer verilen görev **Makefile** dosyasının yüklenmesini istiyorsa, sizin kaynak dosyalarınızı **-Wall**, **-Wextra** , **-Werror**, flaglarını kullanarak derleyip çıktı dosyalarını üretecek olan **Makefile** dosyasını oluşturmanız gerekmektedir. **Makefile** dosyasını oluştururken **cc** kullanın ve **Makefile** dosyanız yeniden ilişkilendirme yapmamalıdır (re-link).
- **Makefile** dosyanız en azından **\$(NAME)**, **all**, **clean**, **fclean** ve **re** kurallarını içermelidir.
- Projenize bonusu dahil etmek için **Makefile** dosyanıza **bonus** kuralını dahil etmeniz gerekmektedir. Bonus kuralının dahil edilmesi bu projenin ana kısmında kullanılması yasak olan bazı header dosyaları, kütüphaneler ve fonksiyonların eklenmesini sağlayacaktır. Eğer projede farklı bir tanımlama yapılmamışsa, bonus projeleri **\_bonus.{c/h}** dosyaları içerisinde olmalıdır. Ana proje ve bonus proje değerlendirmeleri ayrı ayrı gerçekleştirilmektedir.
- Eğer projeniz kendi yazmış olduğunuz **libft** kütüphanesini kullanmanıza izin veriyorsa, bu kütüphane ve ilişkili **Makefile** dosyasını proje dizinindeki **libft** klasörüne ilişkili **Makefile** dosyası ile kopyalamanız gerekmektedir. Projenizin **Makefile** dosyası öncelikle **libft** kütüphanesini kütüphanenin **Makefile** dosyasını kullanarak derlemeli ardından projeyi derlemelidir.
- Test programları sisteme yüklenmek zorunda değildir ve puanlandırılmayacaktır. Buna rağmen test programları yazmanızı şiddetle önermekteyiz. Test programları

sayesinde kendinizin ve arkadaşlarınız projelerinin çıktılarını kolaylıkla gözlemleyebilirsiniz. Bu test dosyalarından özellikle savunma sürecinde çok faydalanacaksınız. Savunma sürecinde kendi projeleriniz ve arkadaşlarınızın projeleri için test programlarını kullanmakta özgürsünüz.

- Çalışmalarınız atanmış olan git repolarına yüklemeniz gerekmektedir. Sadece git reposu içerisindeki çalışmalar notlandırılacaktır. Eğer Deepthought sizin çalışmanızı değerlendirmek için atanmışsa, bu değerlendirmeyi arkadaşlarınızın sizin projenizi değerlendirmesinden sonra gerçekleştirecektir. Eğer Deepthought değerlendirme sürecinde herhangi bir hata ile karşılaşılırsa değerlendirme durdurulacaktır.

## Bölüm III

### Zorunlu Kısım

Program adı	libft.a
Teslim edilecek dosyalar	*.c, libft.h, Makefile
Makefile	Evet
Harici fonksiyonlar.	Detaylar aşağıda açıklanmış
Libft kullanılabilir mi?	Kendiniz oluşturacaksınız
Açıklama	Cursusunuz için önemli fonksiyonları içeren kendi kütüphanenizi yazın

#### III.1 Teknik Hususlar

- Global değişken tanımlamak yasaktır.
- Eğer karmaşık fonksiyonları yazmak için alt fonksiyonlara ihtiyacınız varsa bu alt fonksiyonları **static** olarak tanımlamanız gerekmektedir. Bunun nedeni bu fonksiyonları kütüphanenizle birlikte yayınlamamanızı sağlamaktadır. Bunu gelecekteki projeleriniz için de uygulamanız sizin yararınıza olacaktır.
- Reponuzun rootundaki bütün dosyaları yükleyin.
- Kullanılmayan dosyaların yüklenmesi yasaktır.
- Bütün .c dosyaları flagler ile derlenmelidir.
- Kendi kütüphanenizi oluşturmak için `ar` komutunu kullanmanız gerekmektedir. `Libtool` kullanılması yasaktır.

## III.2 Bölüm 1 - Libc Fonksiyonları

Birinci bölümde, bazı libc fonksiyonlarını teknik dokümanlarına göre baştan yazacaksınız. Baştan yazmış olduğunuz fonksiyon orijinali ile aynı prototipi ve davranışı yansıtmalıdır. Fonksiyonlarınızın adı “ft\_” ön ekini içermelidir. Örneğin, `strlen` fonksiyonunu baştan yazıldığında `ft_strlen` halini alacaktır.



Baştan yazacağınız bazı fonksiyonların prototipleri "restrict" etiketini kullanır. "restrict" etiketi c99 standartının bir parçasıdır. Bu yüzden "restrict" etiketini kendi projelerinize eklemek ve `-std=c99` flagini kullanarak derlemek yasaktır.

Aşağıdaki fonksiyonları baştan yazmanız gerekmektedir. Bu fonksiyonlar çalışmak için herhangi bir harici fonksiyona ihtiyaç duymamaktadır.

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

Aynı zamanda aşağıdaki fonksiyonları da `malloc` kullanarak tekrar yazmanız gerekmektedir.

- `calloc`
- `strdup`

### III.3 Bölüm 2 - Ekstra Fonksiyonlar

İkinci bölümde libc kütüphanesinin içerisinde olmayan veya bu kütüphane içerisine farklı bir formda dahil edilmiş bazı fonksiyonları baştan yazacaksınız. Bu fonksiyonlardan bazıları birinci bölümdeki fonksiyonları yazmak için faydalı olacaktır.

<b>Fonksiyon adı</b>	ft_substr
<b>Prototip</b>	char *ft_substr(char const *s, unsigned int start, size_t len);
<b>Teslim edilecek dosyalar</b>	-
<b>Parametreler</b>	#1. Substringin oluşturalacağı string. #2. Substringin ana string içerisindeki başlangıç indisi. #3. Substringin maksimum uzunluğu.
<b>Return değeri</b>	Substring. Eğer allocation hatası varsa NULL dönsün.
<b>Harici fonksiyonlar</b>	malloc
<b>Açıklama</b>	Malloc fonksiyonu kullanılarak memoryde hafıza ayırılır ve belirtilen substringi döner. Substring başlangıç indisinden başlar ve maksimum boyutuna kadar devam eder.

<b>Fonksiyon adı</b>	ft_strjoin
<b>Prototip</b>	char *ft_strjoin(char const *s1, char const *s2);
<b>Teslim edilecek dosyalar</b>	-
<b>Parametreler</b>	#1. Ön ek stringi #2. Son ek stringi
<b>Return değeri</b>	Yeni oluşturulan string. Eğer hafıza ayırmada problem varsa NULL dönecek.
<b>Harici fonksiyonlar</b>	malloc
<b>Açıklama</b>	Malloc kullanarak hafızadan bir parça ayırılır ve çıktı olarak s1 ve s2 stringlerinin birleştirilmiş hali döndürülür.



<b>Fonksiyon adı</b>	ft_strtrim
<b>Prototip</b>	char *ft_strtrim(char const *s1, char const *set);
<b>Teslim edilecek dosyalar</b>	-
<b>Parametreler</b>	#1. Kırılacak string. #2. Kırılması istenen karakterler.
<b>Return değeri</b>	Kırılmış string. Eğer allocation hatasında problem varsa NULL dönecek.
<b>Harici fonksiyonlar</b>	malloc
<b>Açıklama</b>	Malloc kullanarak hafızada yer ayrılır ardından ana stringde kırılmak istenilen karakterlerin hepsi kırılır sonuç olarak elde edilen yeni string döndürülür.

<b>Fonksiyon adı</b>	ft_split
<b>Prototip</b>	char **ft_split(char const *s, char c);
<b>Teslim edilecek dosyalar</b>	-
<b>Parametreler</b>	#1. Bölünecek string. #2. Ayırıcı karakterler.
<b>Return değeri</b>	Ayırma sonucu elde edilen stringler. Eğer allocation hatası olursa NULL dönülecek.
<b>Harici fonksiyonlar</b>	malloc, free
<b>Açıklama</b>	Malloc kullanılarak hafızada yer ayrılır ardından verilmiş olan ayırıcı karakter yardımı ile string parçalara ayrılır ve bu yeni stringler dönülür. Stringlerin NULL pointer ile sonlanması gerekmektedir.

<b>Fonksiyon adı</b>	ft_itoa
<b>Prototip</b>	char *ft_itoa(int n);
<b>Teslim edilecek dosyalar</b>	-
<b>Parametreler</b>	#1. Dönüştürülecek olan integer değeri.
<b>Return değeri</b>	Integer değerini temsil eden string. Eğer allocation hatası olursa NULL dönecek.
<b>Harici fonksiyonlar</b>	malloc
<b>Açıklama</b>	Malloc kullanarak hafızada yer ayrılır ardından integer değerini temsil eden string döndürülür. Negatif sayılar negatif olarak döndürülmelidir.

<b>Fonksiyon adı</b>	ft_strmapi
<b>Prototip</b>	char *ft_strmapi(char const *s, char (*f)(unsigned int, char));
<b>Teslim edilecek dosyalar</b>	-
<b>Parametreler</b>	#1. Üzerinde dolaşılacak string değeri. #2. Her bir karaktere uygulanacak fonksiyon.
<b>Return değeri</b>	F fonksiyonun karakterlere uygulanması sonucu oluşturulan string. Eğer allocation hatası olursa NULL değeri döndürülecek.
<b>Harici fonksiyonlar</b>	malloc
<b>Açıklama</b>	'f' fonksiyonunu 's' stringinin bütün karakterlerine uygular. Değiştirilen stringden yeni bir string yaratılır.

<b>Fonksiyon adı</b>	ft_striteri
<b>Prototip</b>	void ft_striteri(char *s, void (*f)(unsigned int, char*));
<b>Teslim edilecek dosyalar</b>	-
<b>Parametreler</b>	#1. Üzerinde dolaşılacak string değeri. #2. Her karatere uygulanacak fonksiyon.
<b>Return değeri</b>	None.
<b>Harici fonksiyonlar</b>	None
<b>Açıklama</b>	'f' fonksiyonun stringin her karakterine uygular. Eğer gerekli olursa her karakter adresi ile gönderilmelidir

<b>Fonksiyon adı</b>	ft_putchar_fd
<b>Prototip</b>	void ft_putchar_fd(char c, int fd);
<b>Teslim edilecek dosyalar</b>	-
<b>Parametreler</b>	#1. Çıkış karakteri. #2. Üzerine yazılacak olan file descriptor.
<b>Return değeri</b>	None
<b>Harici fonksiyonlar</b>	write
<b>Açıklama</b>	File descriptoru 'c' karakterinin çıktısını yazar.

<b>Fonksiyon adı</b>	<code>ft_putstr_fd</code>
<b>Prototip</b>	<code>void ft_putstr_fd(char *s, int fd);</code>
<b>Teslim edilecek dosyalar</b>	-
<b>Parametreler</b>	#1. Çıkışa verilecek string. #2. Yazılacak olan file descriptor.
<b>Return değeri</b>	None
<b>Harici fonksiyonlar</b>	<code>write</code>
<b>Açıklama</b>	's' stringini verilen file descriptor içerisine yazar.

<b>Fonksiyon adı</b>	<code>ft_putendl_fd</code>
<b>Prototip</b>	<code>void ft_putendl_fd(char *s, int fd);</code>
<b>Teslim edilecek dosyalar</b>	-
<b>Parametreler</b>	#1. Çıkışa verilecek string. #2. Yazılacak olan file descriptor.
<b>Return değeri</b>	None
<b>Harici fonksiyonlar</b>	<code>write</code>
<b>Açıklama</b>	's' string çıktısını sonunda new line karakteri ile birlikte verilen file descriptor'a yazar.

<b>Fonksiyon adı</b>	<code>ft_putnbr_fd</code>
<b>Prototip</b>	<code>void ft_putnbr_fd(int n, int fd);</code>
<b>Teslim edilecek dosyalar</b>	-
<b>Parametreler</b>	#1. Çıkışa verilecek integer değeri. #2. Yazılacak olan file descriptor.
<b>Return değeri</b>	None
<b>Harici fonksiyonlar</b>	<code>write</code>
<b>Açıklama</b>	Integer 'n' değerinin çıktısını verilen file descriptor'a yazar.

# Bölüm IV

## Bonus Kısım

Eğer zorunlu kısmı tamamlamış iseniz, geri kalan kısımda eğlenmenize bakabilirsiniz. Anlayabileceğiniz üzere son kısım bonus puanları alabileceğiniz kısımdır.

Memoryi ve stringleri manipüle edebilen fonksiyonlara sahip olmak çok kullanışlıdır fakat listeleri manipüle edebilen fonksiyonlara sahip olmanın çok daha kullanışlı olduğunu yakında göreceksiniz.

`make bonus` komutu bonus fonksiyonlarını `libft.a` kütüphanesine ekleyecektir.

Listenizin elementlerini temsil etmek için takip eden yapıyı kullanacaksınız. Bu yapı mutlaka sizin `libft.h` dosyanızın içerisine eklenmelidir.

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

Aşağıda `t_list` structının bileşenlerinin tanımı yapılmaktadır. struct:

- `content` : Elementin içerdiği veri. `void *` tipinde olması istediğiniz türde veriyi tutmanızı sağlar.
- `next` : Bir sonraki elementin adresini tutar eğer son elemensa `NULL` değerindedir.

Aşağıdaki fonksiyonlar listelerinizi kolaylıkla manipüle etmenizi sağlayacaktır.

<b>Fonksiyon adı</b>	ft_lstnew
<b>Prototip</b>	t_list *ft_lstnew(void *content);
<b>Teslim edilecek dosyalar</b>	-
<b>Parametreler</b>	#1. Yeni element oluşturacağınız content değişkeni.
<b>Return değeri</b>	Yeni element.
<b>Harici fonksiyonlar</b>	malloc
<b>Açıklama</b>	Malloc kullanarak memoryden yer ayrılır ve yeni element çıktı olarak verilir. Content değişkeni 'content' parametresinin değeri ile başlatılır. Next değişkeni ise NULL değeri ile başlatılmalıdır.

<b>Fonksiyon adı</b>	ft_lstadd_front
<b>Prototip</b>	void ft_lstadd_front(t_list **lst, t_list *new);
<b>Teslim edilecek dosyalar</b>	-
<b>Parametreler</b>	#1. Listenin ilk bağlantısının pointerının adresi. #2. Listeye eklenecek olan elemanın pointerının adresi.
<b>Return değeri</b>	None
<b>Harici fonksiyonlar</b>	None
<b>Açıklama</b>	Listenin başına yeni bir eleman ('new') ekler.

<b>Fonksiyon adı</b>	ft_lstsize
<b>Prototip</b>	int ft_lstsize(t_list *lst);
<b>Teslim edilecek dosyalar</b>	-
<b>Parametreler</b>	#1. Listenin başlangıcı.
<b>Return değeri</b>	Listenin boyunu dönmelidir.
<b>Harici fonksiyonlar</b>	None
<b>Açıklama</b>	Listedeki eleman sayısını bulur.

<b>Fonksiyon adı</b>	ft_lstlast
<b>Prototip</b>	t_list *ft_lstlast(t_list *lst);
<b>Teslim edilecek dosyalar</b>	-
<b>Parametreler</b>	#1. Listenin başlangıcı.
<b>Return değeri</b>	Listenin son elemanı.
<b>Harici fonksiyonlar</b>	None
<b>Açıklama</b>	Listenin son elemanın döner.

<b>Fonksiyon adı</b>	ft_lstadd_back
<b>Prototip</b>	void ft_lstadd_back(t_list **lst, t_list *new);
<b>Teslim edilecek dosyalar</b>	-
<b>Parametreler</b>	#1. Listenin ilk elemanın pointerının adresini döner. #2. Listeye eklenecek olan elemanın adresini döner.
<b>Return değeri</b>	None
<b>Harici fonksiyonlar</b>	None
<b>Açıklama</b>	'new' elemanını listenin en sonuna ekler.

<b>Fonksiyon adı</b>	ft_lstdelone
<b>Prototip</b>	void ft_lstdelone(t_list *lst, void (*del)(void *));
<b>Teslim edilecek dosyalar</b>	-
<b>Parametreler</b>	#1. Free edilecek eleman. #2. İçeriği silmek için kullanılacak fonksiyonun adresi.
<b>Return değeri</b>	None
<b>Harici fonksiyonlar</b>	free
<b>Açıklama</b>	Parametre olarak bir eleman alır ve fonksiyona bir diğer paramtere olarak verilmiş olan 'del' fonksiyonunu kullanarak elemanın memorydeki yerini temizler. 'Next' in memorydeki yeri temizlenmiş olmamalıdır.

<b>Fonksiyon adı</b>	ft_lstclear
<b>Prototip</b>	void ft_lstclear(t_list **lst, void (*del)(void *));
<b>Teslim edilecek dosyalar</b>	-
<b>Parametreler</b>	#1. Listedeki herhangi bir elemanın pointerının adresi. #2. İçeriği silmek için kullanılacak fonksiyonun adresi.
<b>Return değeri</b>	None
<b>Harici fonksiyonlar</b>	free
<b>Açıklama</b>	'del' ve free(3) kullanarak elemanı siler ve hafızadaki yerini temizler. Ayrıca silme işleminde elemanın tüm successorlarını da temizler. Sonuç olarak listenin pointerı NULL' a ayarlanmalıdır.

<b>Fonksiyon adı</b>	ft_lstiter
<b>Prototip</b>	void ft_lstiter(t_list *lst, void (*f)(void *));
<b>Teslim edilecek dosyalar</b>	-
<b>Parametreler</b>	#1. Listedeki herhangi bir elemanın pointerının adresi. #2. Listenin içerisinde gezinmek için kullanılacak olan fonksiyonun adresi.
<b>Return değeri</b>	None
<b>Harici fonksiyonlar</b>	None
<b>Açıklama</b>	Listenin üzerinde dolaşır ve 'f' fonksiyonunu listenin her elemanının içeriğine uygular.

<b>Fonksiyon adı</b>	ft_lstmap
<b>Prototip</b>	t_list *ft_lstmap(t_list *lst, void *(*f)(void *), void (*del)(void *));
<b>Teslim edilecek dosyalar</b>	-
<b>Parametreler</b>	#1. Listedeki herhangi bir elemanın pointerının adresi. #2. Listenin içerisinde gezinmek için kullanılacak olan fonksiyonun adresi.
<b>Return değeri</b>	Yeni liste. Eğer allocation hatası olursa NULL döner.
<b>Harici fonksiyonlar</b>	malloc, free
<b>Açıklama</b>	'lst' listesi üzerinde dolaşır ve 'f' fonksiyonunu listenin her elemanına uygular. Uygulama sonucunda oluşan yeni elemanlardan yeni bir liste oluşturulur. Gerekli olduğu durumlarda delete fonksiyonu kullanılarak elemanın içeriği temizlenebilir. #2. Listenin içerisinde gezinmek için kullanılacak olan fonksiyonun adresi. #3. Gerekli olduğunda elemanın içeriğini temizlemeye yardımcı olan fonksiyonun adresi.