libft

- `is`

```
#include <stdio.h>
int main()
{
  printf("%d", ft_is___('_'));
}
```

  - `ft_isalnum` = 1-9 ve a-z aralığını kontrol eder.
  - `ft_isalpha` = a ve z aralığı.
  - `ft_isascii` = ascii 0-127 arasını sorgular.
  - `ft_isdigit` = 0-9 aralığı.
  - `ft_isprint` = yazdırılabilir karakterleri sorgular
- `str`
  - `strlen` = Verilen stringin uzunluğunu döndürür.

```
#include <stdio.h>
int main()
{
  printf("%zu", ft_strlen("elmalargelmedilergittiler"));
}
```

  - `strlcpy` = src olarak verilen stringi dst'ye size kadar kopyalar. Sonra nullar.

```
#include <stdio.h>
int main()
{
  char d[] = "Elmalar geldi";
  char s[] = "Amann";
  printf("%zu \n", ft_strlcpy(d,s, 5));
  printf("%s", d);

}
```

  - `strchr` = string içinde bir karakteri arar ve bulursa o karakterin adresini döndürür
    - Burada `return ((char *)s);` const değerin adresini döndürüyor.

```
#include <stdio.h>
int main()
{
  printf("%s \n", ft_strchr("elmlar", 'm'));
}
```

  - `strrchr` = string içinde bir karakteri arar ve bulduğu son karakterin adresini döndür

```
#include <stdio.h>
int main()
{
  printf("%s \n", ft_strrchr("elmlar", 'm'));
}
```

  - `strdup` = parametre olarak aldığı stringi duplicate eder ama bunu yaparken o string için malloc ile yer açar ve bunu bize return eder.

```
#include <stdio.h>
int main()
{
  char s1[] = "Elma";
  char *s2 = ft_strdup(s1);

  printf("%s", s2);
  free(s2);
}
```

- `strlcat` = iki stringi ard arda birleştirip sona null ekler.

```
#include <stdio.h>
int main(void)
{
  char dst[20] = "Hello, ";
  char src[] = "world!";
  size_t size = 20; // size değeri dst dizisinin maksimum uzunluğunu belirtir

  size_t result = ft_strlcat(dst, src, size);

  printf("Concatenated string: %s\n", dst);
  printf("Total length: %zu\n", result);

  return 0;
}
```

- `strncmp` = s1 ve s2 değerlerini karşılaştırır eğer aynı stringler ise 0 verir değilse farklı bir değer verir.

```
#include <stdio.h>
int main()
{
  printf("%d", ft_strncmp("Elma","Elemm", 3));
}
```

- `strnstr` = haystack dizesinde len kadar needle dizesinde arar.

```
#include <stdio.h>
int main()
{
  printf("%s \n", ft_strnstr("elmlar", 'mla', 6);
}
```

- `mem`
  - `memchr` = bellekte s adresinden başlayarak n kadar gider ve c karakterini içeride arar sonra onu döndürür.

```
#include <stdio.h>
int main()
{
  printf("%s", ft_memchr("elmalar geldi", 'l',sizeof("elmalar geldi")));
}
```

  - `memcmp` = iki bellek bölgesini karşılaştırır ve eğer aynı ise 0 verir değilse farklı bir değer.

```
#include <stdio.h>
int main()
{
  printf("%d", ft_memcmp("eelma","elma", 4));
}
```

  - `memcpy` = src adresindeki n değer kadarını dst bölgesine kopyalar
    - bunda null sorgulaması yoktur çünkü bu memory ile ilgili

```
#include <stdio.h>


int main()
```

```
{
    char str1[] = "Hello, World!";
    char str2[] = "Elmları";

    ft_memcpy(str2, str1 + 2, 4);

    printf("%s\n", str2);

    return 0;
}
```

- `memmove` = memcpy ile aynı işlevi var ancak overlap durumuna karşın swap bir alanda kopyalama yapar.

```
#include <stdio.h>


int main()
{
    char str1[] = "Hello, World!";

    ft_memmove(str1, str1 + 2, 4);

    printf("%s\n", str1);

    return 0;
}
```

- `memset` = verilen bellek adresini n kadar verilen (c) değer ile doldurur.

```
#include <stdio.h>
int main()
{
  char s[] = "elmalar";
  printf("%s", ft_memset(s, '-',5));
}
```

- `to`
  - `toupper` = verilen değeri büyük harf yapar.

```
#include <stdio.h>
int main()
{
  int c = 'k';
  printf("%c", ft_toupper(c));
}
```

  - `tolower` = verilen değeri küçük harf yapar

```
#include <stdio.h>
int main()
{
  int c = 'k';
  printf("%c", ft_tolower(c));
}
```

- `other`
  - `atoi` = verilen stringdeki sayısal değerleri tek alır ve onları yazdırır ama bunları “-” ve “+” işaretlerini de hesaplayarak yapar.

```
#include <stdio.h>
int main()
{
  printf("%d", ft_atoi("-1448"));
}
```

  - `bzero` = verilen bellek adresindeki bölgeyi n kadar NULL yapar.

```
#include <stdio.h>
int main() {
    char str[] = "elmalarıye";
```

```
    ft_bzero(str, 5);
    printf("%s\n", str);
    return 0;
}
```

▼ `EXTRA :`

▼ `substr:` verilen stringi start konumundan başlayarak len kadar yeni bir dinamik bellek alanı açar ve stringin o alanının bu bellek alanına yedekler. S

```
#include <stdio.h>
int main()
{
  const char *s = "Hello, world!";
  unsigned int start = 7;
  size_t len = 5;

  char *substr = ft_substr(s, start, len);
  printf("Substring: %s\n", substr);

  free(substr); // Belleği serbest bırakmak için

  return 0;
}
```

▼ `strjoin :` iki stringi bir birinin ardına dinamik bellek olarak ekler sonra stringi doner.

```
#include <stdio.h>
int main()
{
  printf("%s", ft_strjoin("Elma","Armut"));
}
```

▼ `strtrim:` stringimizden verilen değerleri başta ve sonda olmak üzere çıkarır daha sonrasında bunu dinamik belleğe atayıp sonucu döndürür.

```
#include <stdio.h>
int main()
{
  printf("%s", ft_strjoin("-","--ElmaArmut--"));
}
```

▼ `split:` Verilen stringi verilen karaktere göre böler ve bölünen bu stringleri dinamik çok boyutlu bir belleğe atar sonra bize çıktısını verir.

```
#include <stdio.h>
int main()
{
  printf("%s", ft_strjoin("-","Elma-Armut-kek"));
}
```

▼ `itoa:` verilen integer değer çıktı olarak dinamik bellekte bir string şeklinde verilir.

```
int main()
{
  int num = -123;
  char *str = ft_itoa(num);

  printf("%s\n", str);

  free(str);
  return 0;
}
```

▼ `strmapi:` verilen fonksiyonu verilen stringe uygular sonucu yeni bir dinamik bellek alanına ekler ve döndürür.

```
char upper_case(unsigned int index, char c)
{
  if (c >= 'a' && c <= 'z')
```

```
    return c - 32;
  else
    return c;
}

int main()
{
  char str[] = "Hello, World!";
  char *result = ft_strmapi(str, &upper_case);

  printf("Original String: %s\n", str);
  printf("Modified String: %s\n", result);

  free(result);

  return 0;
}
```

- `striteri:` Verilen fonksiyonu verilen stringe uygulanır ama bu sefer doğrudan adrese uygulanır.

```
void print_character(unsigned int index, char *ch)
{
  printf("Character at index %u: %c\n", index, *ch);
}

int main()
{
  char str[] = "Hello, world!";
  ft_striteri(str, print_character);
  return 0;
}
```

- `FD`
  - `putchar_fd:` fd'e verilen c karakterini yazdırır.

```
#include <fcntl.h>
#include <stdio.h>

int main()
{
    int fd = open("test.txt", O_CREAT | O_WRONLY, 7777); //creat dosya oluşturur wronly yazmaya açık yapar sondaki sayılar ch
    ft_putchar_fd('e', fd);
    return 0;
}
```

  - `putstr_fd:` fd'ye verilen stringi yazar.

```
#include <fcntl.h>
#include <stdio.h>

int main()
{
    int fd = open("test.txt", O_CREAT | O_WRONLY, 7777); //creat dosya oluşturur wronly yazmaya açık yapar sondaki sayılar ch
    ft_putstr_fd("kediler uzaylı mı", fd);
    return 0;
}
```

  - `putendl_fd:` fd'deki stringin çıktısını alt satıra geçerek yazdırır.

```
#include <fcntl.h>
#include <stdio.h>

int main()
{
    int fd = open("test.txt", O_CREAT | O_WRONLY, 7777); //creat dosya oluşturur wronly yazmaya açık yapar sondaki sayılar ch
    ft_putendl_fd("kediler uzaylı mı", fd);
    return 0;
}
```

  - `putnbr_fd:` n int değerini fd'ye yazdırır.

```
#include <fcntl.h>
#include <stdio.h>

int main()
{
    int fd = open("test.txt", O_CREAT | O_WRONLY, 7777); //creat dosya oluşturur wronly yazmaya açık yapar sondaki sayılar ch
    ft_putnbr_fd(12323121, fd);
    return 0;
}
```

- `LİNKEDLİST`
  - `lstnew` : Yeni bir node oluşturur.

```
#include <stdio.h>
int main()
{
  printf("%s", (char *)ft_lstnew("test")->content);
}
```

  - `lstadd_front` : Nodu listenin önüne ekler

```
include <stdio.h>

int main()
{
  t_list *node1;

  node1 = ft_lstnew("elma");
  ft_lstadd_front(&node1,ft_lstnew("armut"));

  printf("%s\n", node1->content);
  printf("%s\n", node1->next->content);

}
```

  - `lstsize` listenin node sayısını uzunluğuunu verir.

```
#include <stdio.h>
int main()
{
  t_list *node1,*node2,*node3;
  node1 = (t_list *)malloc(sizeof(t_list));
  node2 = (t_list *)malloc(sizeof(t_list));
  node3 = (t_list *)malloc(sizeof(t_list));

  node1->next = node2;
  node2->next = node3;
  node3->next = NULL;

  printf("%d", ft_lstsize(node1));

}
```

  - `lstlast` listenin son nodeunun adresini verir

```
#include <stdio.h>
int main()
{
  t_list *node1,*node2,*node3;
  node1 = (t_list *)malloc(sizeof(t_list));
  node2 = (t_list *)malloc(sizeof(t_list));
  node3 = (t_list *)malloc(sizeof(t_list));

  node1->next = node2;
  node2->next = node3;
  node3->next = NULL;
  node3->content = "42";

  printf("%s", ft_lstlast(node1)->content);

}
```

▼ `lstadd_back` Nodu listenin sonuna ekler

```
int main()
{
  t_list *node1;

  node1 = ft_lstnew("elma");
  ft_lstadd_back(&node1,ft_lstnew("armut"));

  printf("%s\n", node1->content);
  printf("%s\n", node1->next->content);

}
```

▼ `lstdelone` fonksiyona gelen structın contentini (del) fonksiyonu ile siler ve freeler sadece o değeri

```
#include <stdio.h>

void  *del(void *content)
{
  free(content);
  return (0);
}
int main()
{
  t_list *node1,*node2,*node3;
  char *a = ft_itoa(1234);
  char *b = ft_itoa(5678);
  char *c = ft_itoa(91011);
  node1 = ft_lstnew(a);
  node2 = ft_lstnew(b);
  node3 = ft_lstnew(c);

  node1->next = node2;
  node2->next = node3;
  printf("%s\n", node1->content);

  ft_lstdelone(node1, (void *)(*del));
  printf("%s", node2->content);
}
```

▼ `lstclear`: lst struct yapısının ve bağlı olduğu tüm içerikleri ve yapıları temizler.

```
#include <stdio.h>

void  *del(void *content)
{
  free(content);
  return (0);
}
int main()
{
  t_list *node1,*node2,*node3;
  char *a = ft_itoa(1234);
  char *b = ft_itoa(5678);
  char *c = ft_itoa(91011);
  node1 = ft_lstnew(a);
  node2 = ft_lstnew(b);
  node3 = ft_lstnew(c);

  node1->next = node2;
  node2->next = node3;
  printf("%s\n", node1->content);
  printf("%s\n", node2->content);
  printf("%s\n", node3->content);


  ft_lstclear(&node2, (void *)(*del));
  printf("---------\n");

  printf("%s\n", node1->content);
  printf("%s\n", node2->content);
  printf("%s\n", node3->content);

}
```

▼ `lstiter` : content değerlerine fonksiyonu uygular

```
#include <stdio.h>

void ft_change(void *content)
{
  printf("Test: %s \n ", content);
}

int main()
{
  t_list *node1,*node2,*node3;
  node1 = ft_lstnew("elma");
  node2 = ft_lstnew("nene");
  node3 = ft_lstnew("Kek");

  ft_lstadd_back(&node1,node2);
  ft_lstadd_back(&node2,node3);

  ft_lstiter(node1, ft_change);

}
```

▼ **REHBER**

▼ `const`

`const` anahtar kelimesi, bir değişkenin değerinin değiştirilemez olduğunu belirtmek için kullanılır. Bu, bir değeri atanıp daha sonra değiştirilmesini engellemek için kullanışlıdır

```
const int MAX_VALUE = 100;
const char* MESSAGE = "Hello, world!";
```

Yukarıdaki örnekte, `MAX_VALUE` değişkenine atanan değer daha sonra değiştirilemez ve `MESSAGE` değişkeni bir karakter dizisi temsil eder, bu nedenle değeri değiştirilemez. `const` anahtar kelimesi, hatalı değişikliklere karşı koruma sağlamak ve programın güvenliğini artırmak için kullanılır.

Orneğin;

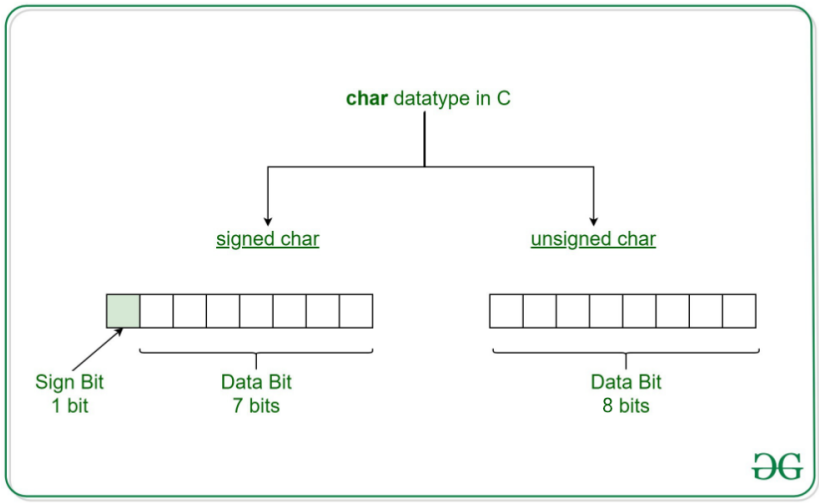
- Hatalı bir kod yazımı.
- Birkaç kişinin çalıştığı bir projede yapılacak değişikliklerden etkilenmemesi için.
- Bir programda, bellek yönetimi hataları ortaya çıkabilir. Örneğin, bir değişkenin bellekte ayrılmış bir alanın sınırlarını aşması veya geçerli bir bellek bölgesine yazma işlemi yapması gibi durumlar. Bu şekilde `const` kullanarak sabit verilerin bellekte değiştirilemez olduğunu belirtmek, programın bellek kullanımını güvence altına alır ve hatalı bellek işlemlerinin önüne geçer.

▼ `unsigned`

`Unsigned`, C dilinde kullanılan bir veri türüdür ve sadece pozitif veya sıfır değerlerini temsil eder. Bu, bir değişkenin negatif değerler alamayacağı anlamına gelir. İşaretli veri türleri (signed) negatif ve pozitif değerler alabilirken, unsigned veri türleri sadece pozitif veya sıfır değerlerini alır.

`unsigned char` karakterleri ve byte değerlerini temsil etmek için kullanılır.

▼ Bellek Boyutu: 1 byte (8 bit)



- Değer Aralığı: 0 ile 255 arasında (2^8 - 1)
- Kullanımı: Genellikle karakterleri temsil etmek için kullanılır. ASCII karakterlerini tutmak veya byte değeri olarak kullanmak için uygundur. ASCII karakter seti sadece 0 ile 127 arasındaki karakterleri kapsar. Farklı karakter kodlama sistemleri ise daha geniş bir karakter yelpazesini kapsar. `unsigned char` veri türü, ASCII karakterlerinin yanı sıra farklı karakter kodlama sistemlerindeki karakterleri de temsil edebilir. Orneğin Unicode.

`unsigned int` daha büyük pozitif sayıları temsil etmek için kullanılır.

- Bellek Boyutu: 4 byte (32 bit)
- Değer Aralığı: 0 ile 4,294,967,295 arasında (2^32 - 1)
- Kullanımı: Geniş bir pozitif sayı aralığını temsil etmek için kullanılır. Bellek boyutu daha büyük olduğu için daha büyük değerleri tutabilir.

▼ `void (pointer)`

`void` işaretçisi `(void pointer)`, herhangi bir türdeki veriye işaret edebilen bir işaretçi türüdür. `void *` türü, işaretçinin temelde boş olduğunu ve bu nedenle programda herhangi bir veri türü adresini tutabileceğini gösterir. Sonra, bu adresi içeren void pointerlar, başka veri türüne dönüştürülebilirler.

- `Tür Dönüşümü:` `void *` işaretçiler, herhangi bir veri türüne işaret edebilir. Ancak, işaretçi üzerinde veriye erişim yapmak istediğinizde, işaretçiyi doğru türe dönüştürmelisiniz. Bu, `void *` işaretçiyi hedef veri türünün işaretçisine (`int *` , `float *` , vb.) dönüştürmek için tip dönüşümü yapmanız gerektiği anlamına gelir.
- Void pointer, herhangi bir veri türüne ilişkin bilgi taşımadığından, işaret ettiği bellek konumundaki veriyi doğrudan kullanamazsınız. Önce void pointer'ı ilgili veri türü işaretçisine dönüştürmeniz gerekmektedir.
- **malloc() ve calloc(), void * tipini döndürür ve bu, bu işlevlerin herhangi bir veri tipinin hafızasını tahsis etmek için kullanılmasına izin verir.**

```
(int*)malloc(sizeof(int))
```

▼ `Ornek1`:

Örneğin, farklı veri tiplerinin öğelerini depolayabilen bir bağlı liste veri yapısı uygulamak istediğinizi varsayalım. Her öğenin değerini depolamak için bir `void *` işaretçisi kullanabilir ve değeri erişmek veya değiştirmek gerektiğinde uygun türe dönüştürebilirsiniz.

İşte `void *` işaretçilerini kullanarak tamsayılar depolayan bir bağlı liste örneği uygulaması:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct node {
    void *data;
    struct node *next;
```

```
} Node;

void add_node(Node **head, void *data) {
    Node *new_node = (Node *) malloc(sizeof(Node));
    new_node->data = data;
    new_node->next = *head;
    *head = new_node;
}

void print_list(Node *head) {
    Node *current = head;
    while (current != NULL) {
        printf("%d ", *(int *) current->data);
        current = current->next;
    }
    printf("\\n");
}

int main() {
    Node *head = NULL;

    int a = 10, b = 20, c = 30;

    add_node(&head, &a);
    add_node(&head, &b);
    add_node(&head, &c);

    print_list(head);

    return 0;
}
```

Bu örnekte, `add_node` fonksiyonu, ikinci argüman olarak `void *` işaretçisi alır ve yeni düğümde depolanacak değeri temsil eder. Bir tamsayı eklerken, `&` operatörünü kullanarak bir tamsayı işaretçisini geçiririz. `print_list` fonksiyonunu kullanarak listeyi yazdırırken, `void *` işaretçisini `(int *)` kullanarak bir `int *` işaretçisine dönüştürür ve ardından tamsayı değerini almak için `*` işaretçisini kullanarak çözümleriz.

▼ `Ornek2:`

```
#include <stdio.h>
#include <stdlib.h>

void* bellekAyrilari(size_t boyut) {
    void* ptr = malloc(boyut);
    return ptr;
}

int main() {
    size_t boyut = 10;
    void* ptr = bellekAyrilari(boyut);

    if (ptr == NULL) {
        printf("Bellek ayrilamadi.\n");
        return 1;
    }

    // Bellek bloğunu kullanma
    for (int i = 0; i < boyut; i++) {
        *((char*)ptr + i) = 'A' + i;
    }

    // Bellek bloğunu yazdırma
    for (int i = 0; i < boyut; i++) {
        printf("%c ", *((char*)ptr + i));
    }
    printf("\n");

    // Belleği serbest bırakma
    free(ptr);

    return 0;
}
```

▼ `size_t`

- unsigned int bir değer dondürür
- asla negatif olmaz

- programcının bir uzunluk belirtildiğini anlamasını sağlar

▼ `File Descriptor`

https://www.codequoi.com/en/handling-a-file-by-its-descriptor-in-c/

▼ **BONUS REHBER**:

▼ `Struct`

Bir `struct`, farklı veri türlerinin birleşimini içeren bir C dilindeki veri yapısıdır. Bu, farklı veri tiplerini tek bir yapı altında gruplandırmamızı sağlar. Yapılar, benzer verilere sahip bir nesne veya varlık modellemesi için kullanılabilir.

Tanımlama yapılırken:

```
struct struct_adi {
    veri_turu1 eleman1;
    veri_turu2 eleman2;
    // ...
};
```

Kod içinde çağırılırken:

```
struct struct_adi degisken_adi;
struct struct_adi *isaretcisi;
```

▼ `Typedef`

İkinci olarak ele alacağımız kısım “**typedef”** kelimesidir. **“typedef”** kelimesini kullandığımızda, **“struct”** için yeni değişken tanımlarken struct ifadesini tekrardan kullanma ihtiyacı duymayız.

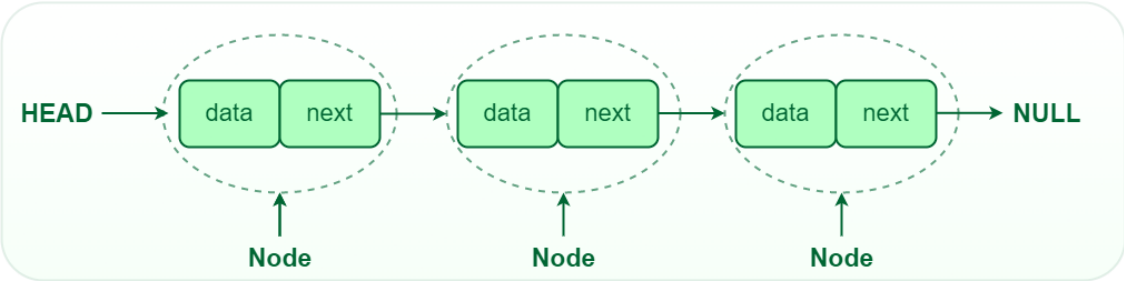
```
struct dogum
{
    int gun;
    int ay;
    int yil;
};

  struct dogum *Ptr,sidar ;
```

```
typedef struct bilgi
{
    int numarasi;
    char *isim;

}ogrenci;

ogrenci ogrenci_bir;
```

▼ `Linked List`

https://log2base2.com/courses/data-structures-in-c/linked-list-basics-c-trial

Extra bolümünde verilen Struct yapısı temelde bir `linked list` oluşturmak için bize verilmiştir. Linked List arda arda dizilmiş struct yapılarından oluşur. Her bir parçasına ise `Node` adı verilir.



```
typedef struct s_list
{
```

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

Bize verilen bu listede `content` bir `void` değerdir ve değerimizi taşır.

`Next` ise bir pointer tutar. Yani listedeki sonraki `node'un` adresini tutar.

- Listemizin son değeri `null` bir adresi gosterir.
- Her `node` `content` ve `next` değerlerini barındırır.