BÖLÜM 5 ARRAY

ARRAY KAVRAMINA GİRİŞ

Arrayler çok sayıda veriyi tutmamızı kolaylaştıran bir yapıdır.

Diyelim ki 100 tane sayı depolayacağız veya 100 sayılık bir listemiz var ve depolama için değişken tanımlamaya çalışıyoruz
İnt sayi1, sayi2, sayi3 .....sayi100; şeklinde bir tanımlama çok uzun sürer ve bu değişkenlerle işlem yapmak çok zordur. Bu zorluktan kurtulmak, çok sayıda veri depolamak veya liste tutmak için array kullanıyoruz.

ARRAY TANIMLAMAK

10 tane veri tutan bir array tanımlayacağız

tutulacak_veri_tipi array_ismi[eleman_sayısı] = {veri, veri, veri....., veri}
(opsiyonel)

tutulacak_veri_tipi kavramı önemli. Bir array aynı veri tipinden verileri tutar yani

int array[5] dediğimiz zaman arrray 5 tane int değer tutar; eğer double tutmak isterseniz type-casting gerçekleşir ve ondalık kısmı kaybedersiniz.
Mesela 3.14, 3 olarak tutulur

double array[5] dediğimiz zaman array 5 tane double tutar; eğer int tutmak isterseniz type-casting gerçekleşir.
Mesela 5, 5.000000 olarak tutulur

double tutan arraye array of double (double array) denir.
int tutan arraye array of int (int array) denir.

yöntemle array nasıl tanımlanır görelim.

1. YÖNTEM

double array[] = {3.14, 2.71, 26, 0.6, 10};

[ ] içine sayı yazmayıp başlangıç değerlerini verdiğimiz bu yöntemde derleyici arrayde kaç tane eleman olduğunu sayar ve bu verileri RAM'de depolar. Depolamayı görselleştirmek istersek:

diyelim ki depolamaya RAM'in 1000. byte'ından başlıyoruz. double tipinden bir değişkenin kapladığı alan byte ise 1000 ile 1007 arası bölge 3.14 için ayrılır. 1008 ile 1015 arası bölge 2.71 için ayrılır yani arraylerdeki elemanlar ard arda depolanır.

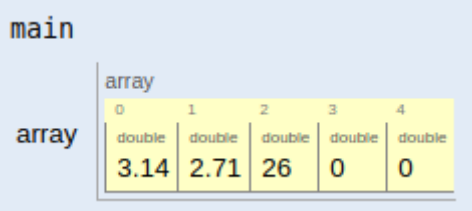
RAM	veri
1000	3.14
1008	2.71
1016	26.000000
1024	0.6
1032	10.000000

2.YÖNTEM

double array[] = {3.14, 2.71, 26, 0.6, 10};

hem değerleri hem eleman sayısını vermek pek tavsiye edilmez ama bazen gereklidir. Mesela 5 elemanlı bir array tanımlayacağız ama 2 tane değeri sonradan alacağız. Böyle bir durumda

double array[5] = {3.14, 2.71, 26}; şeklinde bir tanımlama gayet mantıklıdır. Bu kodu c tutorda çalıştırdığımızda başlangıç değeri verilmeyen elemanların 0 (aslında 0.000000) olarak depolandığını görürüz(bu 0 olayı int array'lerde de geçerlidir)



double array[5] = {0}; tüm elemanları 0 a eşitler

3. YÖNTEM

Bu yöntemde köşeli parantez içine direkt bir sayı yazmak yerine #define komutu ile bir ifade yazıyoruz. Daha önce bahsettiğimiz gibi önişlemci SIZE yerine 5 yazacak. Biz bu işlemi okuma kolaylığı için yapıyoruz.

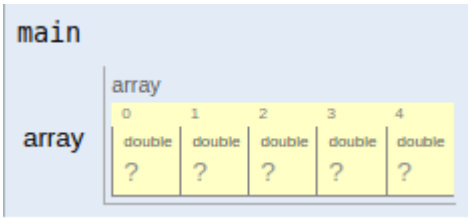
```c
#include <stdio.h>
#define SIZE 5

int main(){

    double array[SIZE] = {1,6,3.14,2.71,5};

    return 0;
}
```

4. YÖNTEM

Peki hiç başlangıç değeri vermezsek ne olur

```c
#include <stdio.h>
#define SIZE 5

int main(){

    double array[SIZE];

    return 0;
}
```

Arrayin durumuna c tutorda bakalım



? işareti rastgele anlamına gelir. Arrayin elemanlarını bastırırsak tutulan sayıların rastgele sayılar 0 veya olduğunu görebiliriz bu sizin derleyicinize bağlıdır.

5. YÖNTEM

Son yöntem ise arrayin boyutunu değişkenler ile tanımlamak. Bu yöntemin geçerliliği derleyicinize ve standarta bağlı. Kısaca söylemek gerekirse bu yöntem c99 ve üstü için çalışıyor

Tek yapmanız gereken köşeli parantez içine bir değişken yazmak

```c
#include <stdio.h>

int main(){

    int a = 5;

    double array[a];

    return 0;
}
```

Bu yöntemi tavsiye etmiyorum çünkü: farklı standartlarda çalışmayabilir ve usul olarak arrayin köşeli parantezine bir değişken değil de sayi yazılır.

İNDEX(SUBSCRİPT) KAVRAMI

Arrayin her hangi bir elemanına erişmek için elemanın array içindeki sırasından yani pozisyonundan yararlanırız. Bu pozisyona İNDEX(İNDEKS) denir ve indeksler 0 dan başlar.

Yani arrayin en başındaki eleman 0. indeks'tedir.
Bizim sayımımıza göre 10. sıra 9. indekstir.
Yani son eleman (arrayin_boyu – 1). indekstir

Bir indeksteki değere ulaşmak için array_ismi[indeks] sentaksını kullanırız. Mesela ilk sıradaki yani 0. indeksteki elemanı

array[0] olarak ifade ederiz.

array[0] ifadesi tıpkı normal bir değişken gibi ekrana bastırabilir veya değiştirilebilir.

```c
#include <stdio.h>

int main(){

    int array[] = {5, 6, 8 ,1 ,9};

    printf("%d\n",array[2]);//8

    array[0] = 10;// 5 -> 10 oldu

    printf("%d\n",array[0]);//10

    return 0;
}
```

eğer array'de olmayan bir indekse erişmek istersek anlamsız bir değer ile karşılaşırız çünkü c array[10] ifadesinde arrayin 10. indeksi var mı diye kontrol etmez. Doğru indeksleri yazmak bizim sorumluluğumuzdur
Bu duruma index error denir.

```c
#include <stdio.h>

int main(){

    int array[] = {5, 6, 8 ,1 ,9};

    printf("%d\n",array[10]);

    return 0;
}
```

İTERASYON

İterasyon tekrarlama anlamına gelir. Arraylerde iterasyon yapmak ise: elemanlar üzerinde sırayla aynı işlemleri yapmaktır.

Bunun en basit örneği bir arrayin elemanlarını sırayla yazdırmaktır.

10 elemanı olan bir array düşünelim. 10 yerine #define ile SIZE yazılmış olsun
int array[SIZE] = {9,6,7,33,9,4,5,7,10,123};

İterasyon için döngüleri, çoğunlukla da for döngülerini kullanırız.

1) İlk elemanı 0. indeks olarak isimlendirdiğimiz için döngüyü 0 dan başlatırız

for(int i = 0; ... ; ...)

2)Sırayla gideceğimiz için i yi her defada 1 arttırır böylece bir sonraki indekse geçeriz

for(int i = 0; ... ;i++)

3) son eleman 9. indeksteki elemandır ve arrayin boyunun 1 eksiğidir. Bu yüzden koşul olarak i < SIZE diyerek i'nin son değerinin SIZE den 1 küçük olmasını sağlayabiliriz.

for(int i = 0; i < SIZE ;i++)

4) son olarak yapılacak işlemin kodunu yazalım. array[i] ifadesinin normal bir int değişkeni gibi olduğu söylemiştik yani bir integer array'in elemanlarını %d ile yazdırabiliriz

printf("%d\n",array[i]);

```c
#include <stdio.h>
#define SIZE 10

int main(){

    int array[SIZE] = {9,6,7,33,9,4,5,7,10,123};

    for(int i = 0; i < SIZE ;i++){
        printf("%d\n",array[i]);
    }

    return 0;
}
```

Çıktı

```
9
6
7
33
9
4
5
7
10
123
```

Bu sefer #define SIZE 10 ile tanımlanmış bir array için kullanıcıdan 10 tane sayı alalım ve arrayi bastıralım

```c
#include <stdio.h>
#define SIZE 10

int main(){

    int array[SIZE];

    for(int i = 0; i < SIZE ;i++){
        printf("bir sayi girin: ");
        scanf("%d",&array[i]);
    }

    for(int i = 0; i < SIZE ;i++){
        printf("%d\n",array[i]);
    }

    return 0;
}
```

COMPANİON VARİABLE

SORU: Kullanıcıdan aldığınız sayıları bir arraye atın. Kullanıcı -1 girdiğinde sayı alımının bittiğini varsayın ve arrayi yatay olarak bastırın:

input: output:3 5 7
3
5
7
-1

Döngülerde yaşadığımız sorunu tekrar yaşıyoruz: kullanıcının kaç tane sayı gireceğini bilemeyiz.

Array'in en önemli özelliği boyutunun sabit olmasıdır yani 10 elemanlı bir array tanımladığımızda 11. sayı girilirse array bunu depolayamaz ve program hata verir Aşağıdaki kod 10 elemanlı bir arrayin 11. elemanına (!) scanf ile veri almayı temsil ediyor

```c
#include <stdio.h>
#define SIZE 10

int main(){

    int array[SIZE];

    printf("bir sayi girin: ");
    scanf("%d",&array[11]);

    return 0;
}
```

kodu çalıştırıp 5 giriyorum

```
bir sayi girin: 5
*** stack smashing detected ***: terminated
Aborted (core dumped)
```

Diyor ki boyumun yetmediği yere erişmeye çalışıyorsun. Yapma.

Bu sorunu yaşamamak için arrayi tanımlarken cömert davranıyoruz ve büyük ihtimalle kullanıcının girmeyeceği sayıda eleman ayırıyoruz. Kullanıcı büyük ihtimalle 100 tane sayı girmeyecek o yüzden 100 elemanlı bir array yaratıyoruz.

```c
#include <stdio.h>
#define SIZE 100

int main(){

    int array[SIZE];

    return 0;
}
```

Diyelim ki kullanıcı 20 tane sayı girecek. Array uzunluğunun 100 olmasına karşın sadece 20 tane eleman bizim işimize yarayacak yani biz 20 eleman boyutunda bir array parçasıyla işlem yapacağız. Bu boyutu tutmak için yardımcı bir değişken (companion variable) tanımlayacağız ve ismine mevcut_boyut ya da current_size diyeceğiz ve henüz sayı almaya başlamadığımız için 0'a eşitleyeceğiz.

```c
#include <stdio.h>
#define SIZE 100

int main(){

    int array[SIZE];
    int current_size = 0;
    //0 cünkü henüz islem yapmadik

    return 0;
}
```

Şimdi sayıları istemeye başlayacağız. Kullanıcı en fazla 100 tane değer girebileceği için 100 defa döngü çalıştıracağız ve her sayı aldığımızda current_size değişkenini 1 arttıracağız ta ki 100 tane sayı girinceye ya da -1 girinceye kadar

```c
for(int i = 0;i < SIZE;i++){
    int sayi;
    printf("bir sayi girin.cikis icin -1: ");
    scanf("%d",&sayi);

    if( sayi == -1)
        break;

    array[i] = sayi;
    current_size++;
}
```

Sayıları aldıktan sonra sıra arrayi bastırmaya geldi. For döngüsünün koşullu ifade bölümüne i < SIZE mı yoksa i < current_size mı yazacağız?

i < current_size yazacağız çünkü SIZE değişkeni arrayin işimize yarayan ya da yaramayan tüm kısımlarını tutarken; current_size verinin anlamlı kısmını yani bizim işimize yarayan kısmını tutuyor.

Tam program:

```c
#include <stdio.h>
#define SIZE 100

int main(){

    int array[SIZE];
    int current_size = 0;

    for(int i = 0;i < SIZE;i++){
        int sayi;
        printf("bir sayi girin.cikis icin -1: ");
        scanf("%d",&sayi);

        if( sayi == -1)
            break;

        array[i] = sayi;
        current_size++;
    }

    for(int i = 0;i < current_size;i++ ){
        printf("%d ",array[i]);
    }

    printf("\n");

    return 0;
}
```

Bu derste index kavramını pekiştirmek için 2 basit soru çözeceğiz.

SORU 1) Verilen arraydeki elemanların ortalamasını bulun

```c
#include <stdio.h>
//toplam ve ortalama

int main() {

  int array[10] = {100,150,200,250,300,350,400,450,500,550};
  //10 elemanli
  int toplam = 0;

  for(int i = 0;i < 10;i++){
      toplam += array[i];
  }

  double ortalama = (double)toplam / 10;

  printf("ortalama: %f\n",ortalama);

  return 0;
}
```

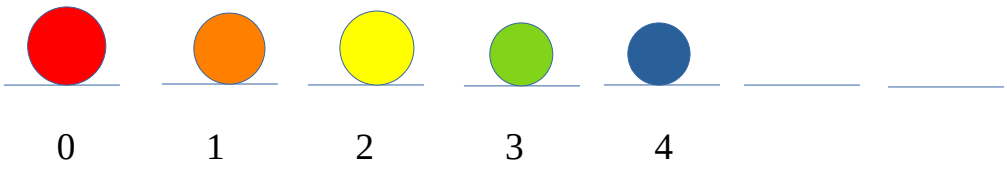
SORU 2) Verilen arrayin en büyük elemanını bulun

```c
#include <stdio.h>

int main() {

    int array[10] = {5,9,3,7,1,55,99,78,6,8};

    int max = array[0];

    for(int i = 1;i < 10; i++){
        //i = 1 cünkü 0. indeksi yukarida kullandik
        if(array[i] > max)
            max = array[i];
    }

    printf("%d\n",max);

    return 0;
}
```

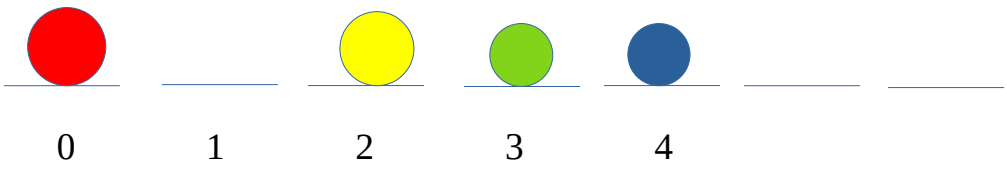
ELEMAN ÇIKARMA

Önceden belirttiğimiz gibi bir arrayin boyutunu değiştiremeyiz ama companion variable ile eleman çıkarır gibi yapabiliriz diyelim ki bir pozisyondaki elemanı çıkartmak istiyoruz.

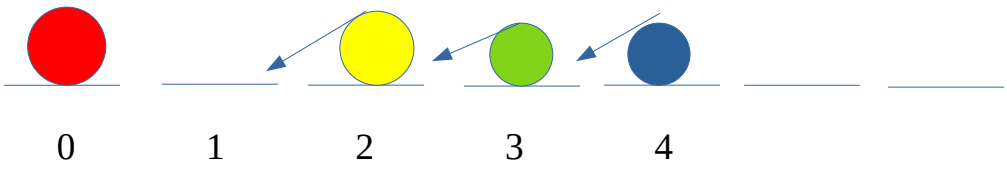
Nasıl yapacağımızı görselleştirelim: Diyelim ki 7 elemanlı ve 5 elemanı anlamlı olan bir dizimiz olsun

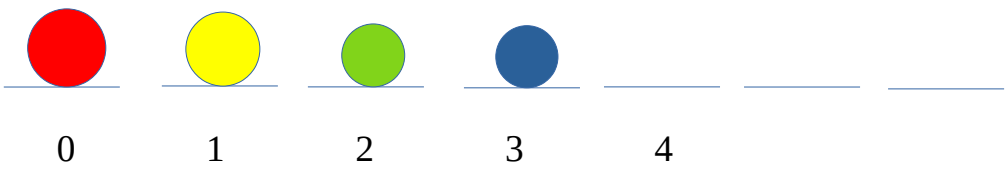


şimdi 1. indeksteki topu çıkaralım

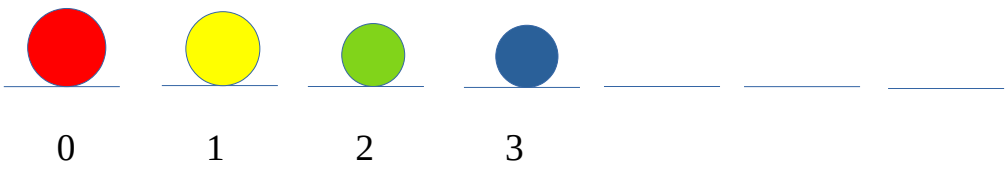


1. indeksin boş kalmaması için ondan sonra gelen topları bir sola kaydırmalıyız. Son top hariç çünkü onun sağında top yok. Topların üst üste binmemesi için kaydırmaya en soldaki toptan başlayalım.





artık 1 tane elemanımız yok bu yüzden mevcut boyu 1 azaltacağız



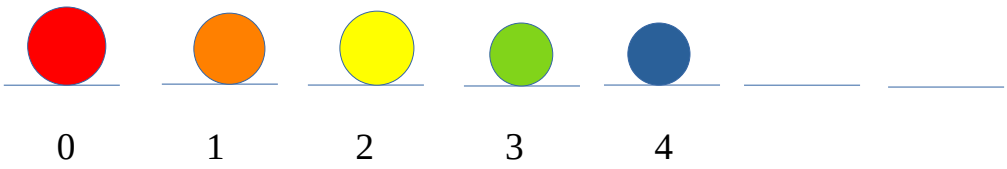
KOD

```c
#include <stdio.h>
#define SIZE 25

int main(){

    int array[] = {1,2,3,4,5,6,7,8,9,10};
    int current_size = 10;

    for(int i = 0;i < current_size;i++ )
        printf("%d ",array[i]);

    printf("\n");

    int yer;
    printf("hangi indeksteki eleman ciksin: ");
    scanf("%d",&yer);

    for(int i = yer;i < current_size - 1;i++){
        array[i] = array[i + 1];
    }

    current_size--;

    for(int i = 0;i < current_size;i++ )
       printf("%d ",array[i]);

    printf("\n");
    return 0;
}
```

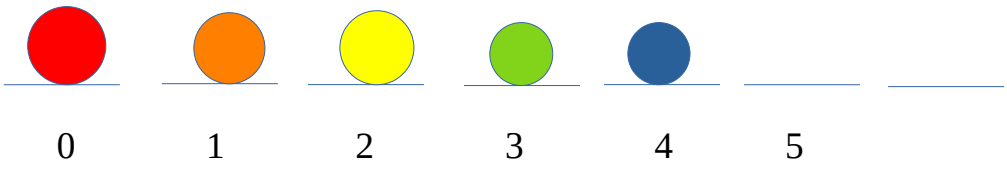
ELEMAN EKLEME

Arraylerin boyutu sabit olduğu için 10 elemanlı bir arrayi 11 elemanlı hale getiremeyiz ama companion variable kavramı ile ekler gibi yapabiliriz:

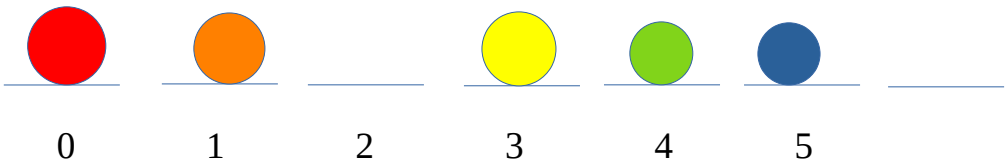
Nasıl yapacağımızı görselleştirelim: Diyelim ki 7 elemanlı ve 5 elemanı anlamlı olan bir dizimiz olsun:



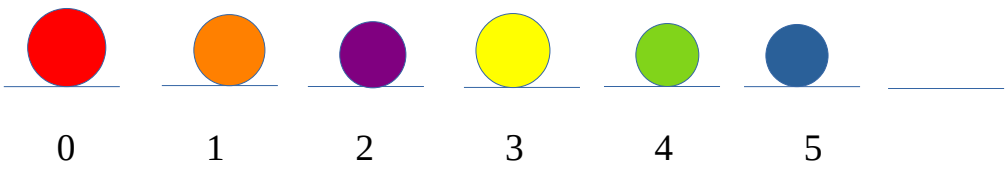
verinin anlamlı kısmına eleman ekleyeceğimiz için mevcut boyutu(current_size değişkenini) 1 arttırıyoruz



Diyelim ki 2. indekse mor bir top koyacağız. İkinci indeksi boşaltmak ve diğer topların mor toptan sonra gelmesini sağlamak için 2. indeks ve sonrasını 1 sağa kaydıracağız. Toplar üst üste binmesin diye kaydırmaya en sağdaki toptan başlayacağız



Şimdi boşalan indekse mor topumuzu koyabiliriz.



KOD

```c
#include <stdio.h>
#define SIZE 25

int main(){
    int array[SIZE] = {1,2,3,4,5,6,7,8,9,10};
    int current_size = 10;

    for(int i = 0;i < current_size;i++ ){
        printf("%d ",array[i]);
    }
    printf("\n");

    current_size++;
    int yer;
    printf("hangi indekse eleman eklensin: ");
    scanf("%d",&yer);

    int sayi;
    printf("hangi sayi eklensin: ");
    scanf("%d",&sayi);

    for(int i = current_size -1;i > yer ;i--){
        array[i] = array[i - 1];
    }

    array[yer] = sayi;

    for(int i = 0;i < current_size;i++ ){
       printf("%d ",array[i]);
       }
    printf("\n");
```

FONKSİYONLARA ARRAY GÖNDERMEK

Bir fonksiyona parametre olarak array verebiliriz. Bu konuda en önemli nokta fonksiyona arrayin uzunluğunu da göndermektir çünkü bir fonksiyon içine aldığı arrayin uzunluğunu bilemez.

sayilar double arrayin elemanlarının toplamını döndüren bir toplam() fonksiyonu yazalım

Elemanlar double olduğu için toplamları da double olacak ve sonuç olarak bir double değer dönecek.

double topla()

parantez içine ise tıpkı bir array tanımlar gibi işlem yapılacak arrayin tipini ve parametre ismini girelim. Parametre ismi array[] olsun. Yanına da uzunluğu ifade eden bir değişken koyalım

double topla(double array[], int size)

[] içine sayı yazmaya gerek yok çünkü içine bir sayı yazdığımızda o sayı compiler tarafından görmezden gelinir. Bu yüzden uzunluğu parametre olarak veriyoruz

Bir başka nokta ise main içinde yazdığımız fonksiyonu kullanırken fonksiyona sadece arrayin ismini veriyoruz ve yanina [] koymuyoruz

```c
#include <stdio.h>
#define SIZE 25

double topla(double array[],int size){
    double toplam = 0;

    for(int i = 0; i < size; i++)
        toplam += array[i];

    return toplam;
}

int main(){
    double sayilar[SIZE] = {1,2,3,4,5,6,7,8,9,10};
    int current_size = 10;

    double toplam = topla(sayilar,current_size);

    printf("%f\n",toplam);
    return 0;
}
```

ARRAYLER VE VOİD FONKSİYONLAR

Diyelim ki ikiyle_carp isimli bir fonksiyon kullanarak bir arrayin tüm elemanlarını 2 ile çarpmak istiyoruz.

Arraylerle uğraşmadan önce bir int değişkenini ikiyle çarpan bir ikiyle_carp fonksiyonunu yazalım.

```c
#include <stdio.h>
#define SIZE 25

int ikiyle_carp(int sayi){

    sayi *= 2;

    return sayi;
}
int main(){
    int sayi = 5;

    int iki_kat = ikiyle_carp(sayi);

    printf("%d\n",sayi);

    return 0;
}
```

çıktının neden 5 olduğunu variable scope dersinde gördük.

Şimdi bu mantıkla ikiyle_carp fonksiyonunu arrayler için yazalım. Fonksiyonun aldığı arrayin elemanlarını ikiyle çarpıp yeni bir arraye koyalım. O arrayi döndürüp başka bir arraye atalım.

```c
#include <stdio.h>
#define SIZE 25

int ikiyle_carp(int array[], int size){
    int sonuc[SIZE];

    for(int i = 0;i < size;i++){
        sonuc[i] = array[i] * 2;
    }

    //array dondurebileceğimizi zannediyoruz
    return sonuc;
}

int main(){

    int sayilar[SIZE] = {0,1,2,3,4,5,6,7,8,9};
    int current_size = 10;

    //array ile atama yapabileceğimizi zannediyoruz
    int iki_kati[SIZE] = ikiyle_carp(sayilar,current_size);
    return 0;
}
```

Kodu çalıştıralım ve hatalarımızı görelim

```
emre@emre-R580-R590:~/Desktop/EGITIM/bolum5$ gcc ders9.c -o ders9
ders9.c: In function ‘ikiyle_carp’:
ders9.c:12:12: warning: returning ‘int *’ from a function with return type ‘int’ makes integer from pointer without a cast [-Wint-conve
rsion]
   12 |     return sonuc;
      |            ^~~~
ders9.c: In function ‘main’:
ders9.c:20:26: error: invalid initializer
   20 |     int iki_kati[SIZE] = ikiyle_carp(sayilar,current_size);
      |                          ^~~~~~~~~~
```

hata diyor ki: ARRAYLERLE ATAMA YAPAMAZSIN.(indeksler değil tüm arrayden bahsediyor)

array döndüremediğimize göre geriye sadece void fonksiyon kullanmak kalıyor

ARRAYLER VE VOİD FONKSİYONLAR 2

Arrayleri void fonksiyonlara göndermeden önce bir değişkeni void fonksiyona gönderelim

```c
#include <stdio.h>
#define SIZE 25

void ikiyle_carp(int sayi){

    sayi *= 2;

}

int main(){

    int sayi = 10;

    ikiyle_carp(sayi);

    printf("%d\n",sayi);

    return 0;
}
```

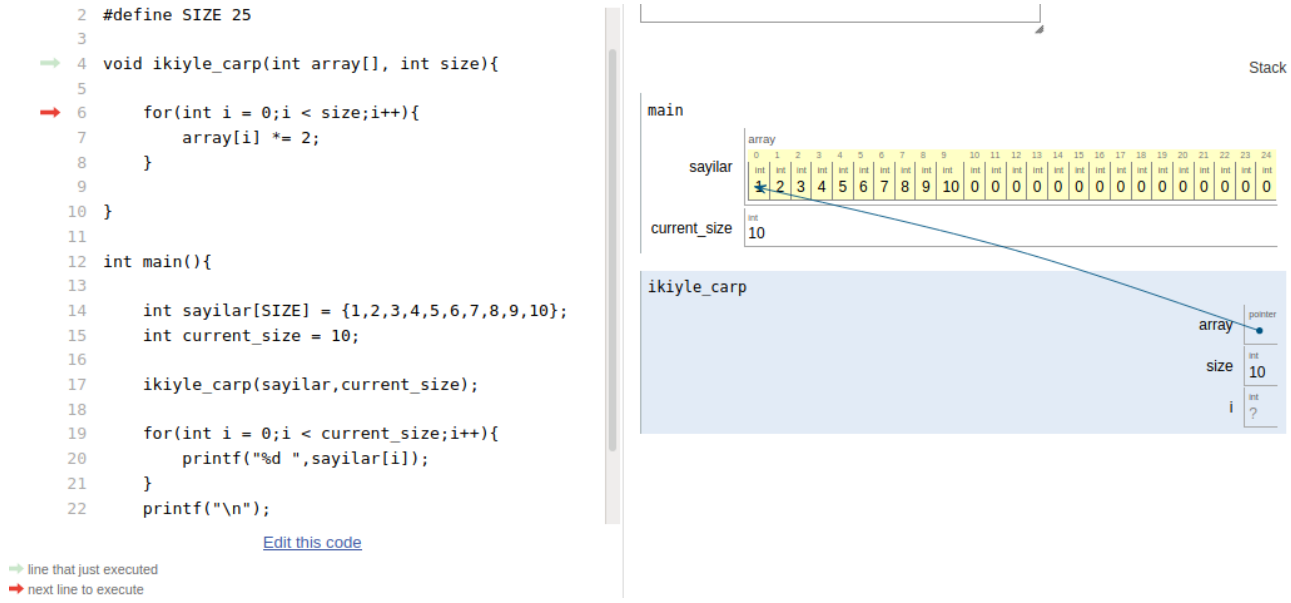
Çıktının 10 olacağını ve sayi değişkeninin değişmeyeceğini scope ve call by value kavramlarından dolayı öngörebiliriz.

Arrayler fonksiyonlara call by value ile gönderilmiyor. Yani değişkenlerden farklı olarak BİR FONKSİYON ALDIĞI ARRAYİ DEĞİŞTİREBİLİR. Gelin ikiyle_carp fonksiyonunu arrayler için yazalım.

```c
#include <stdio.h>
#define SIZE 25

void ikiyle_carp(int array[], int size){

    for(int i = 0;i < size;i++){
        array[i] *= 2;
    }

}

int main(){

    int sayilar[SIZE] = {1,2,3,4,5,6,7,8,9,10};
    int current_size = 10;

    ikiyle_carp(sayilar,current_size);

    for(int i = 0;i < current_size;i++){
        printf("%d ",sayilar[i]);
    }
    printf("\n");

    return 0;
}
```

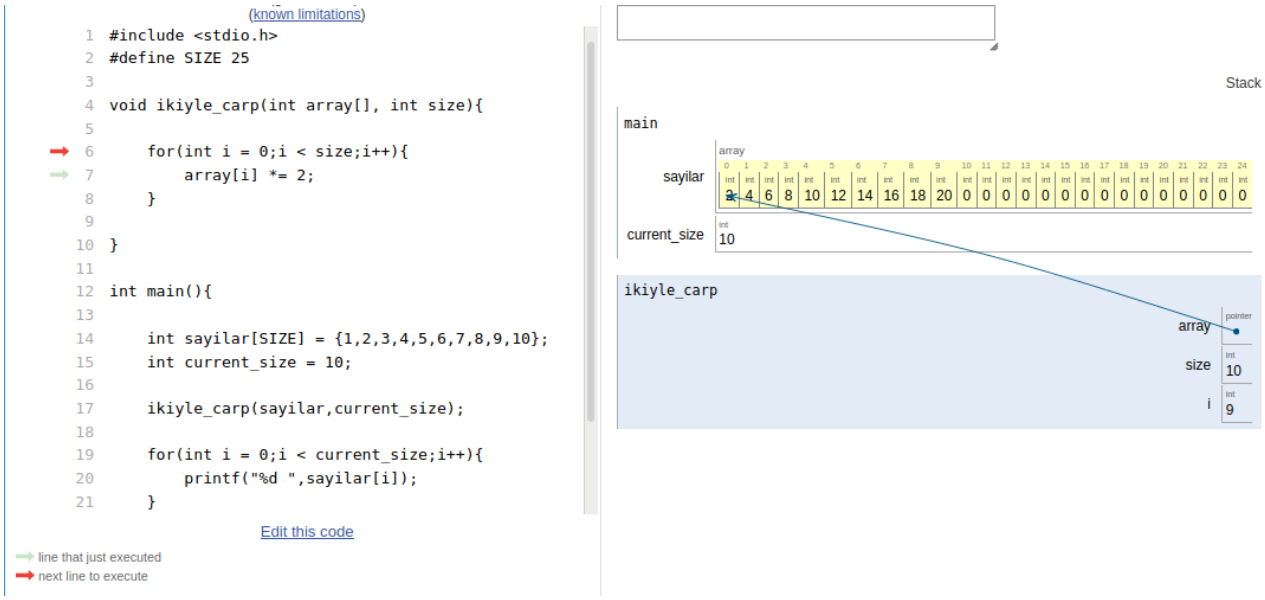
Çıktı: 2 4 6 8 10 12 14 16 18 20

Çıktıdan gördüğünüz gibi array değişmiş neler olduğunu c tutor’da görelim



Şu zamana kadarki fonksiyonlarda görmediğimiz 2 şey görüyoruz
1) Kocaman bir ok
2) array yazısının hemen sağında “pointer” yazıyor

Bu iki farklılık diyor ki:
Arrayler fonksiyonlara fonksiyonlara call by value ile değil “call by value” ile gönderilir. Bunun anlamı: bir fonksiyon, bir arrayin kendisini değiştirebilir.



NOT: arrayin elemanları normal değişkenler gibi call by value ile çağrılır.

LİNEAR SEARCH

Linear search bir arama algoritmasıdır. Bir elemanın array içindeki pozisyonunu bulmaya yarar.
Bu algoritmada arrayin her elemanına sırayla “sen aradığım şey misin” diye sorarız. Eğer bulamazsak yani aranan eleman arrayde yoksa -1 döndürürüz çünkü arrayin negatif indeksi olmaz

```c
#include <stdio.h>
#define SIZE 25

int linear_search(int array[], int size,int aranan){
    int indeks;

    for(int i = 0; i < size; i++){
        if(array[i] == aranan){
            indeks = i;
            return indeks;
        }
    }

    //bulunmazsa -1 dondurulur
    return -1;
}

int main(){

    int sayilar[SIZE] = {1,2,3,4,5,6,7,8,9,10};
    int current_size = 10;

    int bul = 5;//5 sayisinin indeksini bulmaya calisiyorum
    printf("%d\n",linear_search(sayilar, current_size,bul));

    return 0;
}
```

BUBBLE SORT

Bubble sort arrayin elemanlarını sıralayan bil algoritmadır. Ard arda gelen elemanların yerlerini değiştirerek arrayin en küçük elemanını ilk indekse, en büyük elemanı da son indekse koyar

1 5 4 3 2 sayılarını 1 2 3 4 5 haline getirmek istiyoruz

ilk önce ilk iki indeksi karşılaştırıyoruz

---
4 3 2
1 5

2, 1'den büyük ve 1'den solda. Yer değişimi yapmıyoruz

---
1 5 4 3 2
---

sonra bir sağa geçiyoruz yani 1. ve 2. indeksi karşılaştırıyoruz

---
1 3 2
5 4
---

5 4'ten büyük ama solda yani 5, 4'ten önce gelmiş. 5 ve 4 sayısının yerini değiştiriyoruz

---
1 4 5 3 2
---

sonra bir sağa geçiyoruz yani 2. ve 3. indeksi karşılaştırıyoruz

---
1 4 2
5 3

5, 3'ten büyük ve önce gelmiş. 5 ile 3 ün yerini değiştiriyoruz

---
1 4 3 5 2
---

son olarak 3. ve 4. indeksi karşılaştırıyoruz.

---
1 4 3
  5 2
---

tekrar yer değişimi yapıyoruz

---
1 4 3 2 5
---

ilk geçiş bitti. Fark ettiyseniz en büyük sayı sürekli karşılaştırmaya dahil olduğu için arrayin en sonuna geldi. Bir sonraki geçişte 4, 5'in soluna gelerek yerini bulmuş olacak yani her geçişte 1 tane sayı yerine ulaşmış oluyor. Bundan şu sonucu çıkarabiliriz: arrayin uzunluğu kadar geçiş yapacağız.

Ayrıca 5 elemanlı dizimiz için 4 kere karşılaştırma yaptık yani eleman sayısının 1 eksiği kadar karşılaştırma yapıyoruz.

Kodlamaya geçmeden önce iki sayının yer değiştirmesini nasıl yapacağımızı görelim

int a = 5;
int b = 10;
a = b;
b = a; dersek yanlış olur nedenini inceleyelim

a = b satırında a nın değerini 10 yapıyoruz. A şu an 10
b = a satırında ise b nin değerini a yapıyoruz ama a nın değeri de o anda 10 olduğu için b nin değeri değişmiyor ve ve hem a hem de b nin değeri 10 olmuş oluyor.

Bunu önlemek için a nın ilk değerini saklayacak geçiçi bir değişken tanımlıyoruz
int a = 5;
int b = 10;
int gecici = a;

şimdi atamalara geçebiliriz

a = b;
b = gecici;

şimdi algoritmanın kodunu görelim

```c
#include <stdio.h>
#define SIZE 25

void bubble_sort(int array[], int size){

    for(int gecis = 0;gecis < size - 1;gecis++){
        for(int i = 0;i < size - 1;i++){
            if(array[i] > array[i + 1]){

                int gecici = array[i];
                array[i] = array[i + 1];
                array[i + 1] = gecici;

            }
        }
    }
}

int main(){

    int sayilar[SIZE] = {2,1,6,8,7,4,3,9,5,10};
    int current_size = 10;

    bubble_sort(sayilar,current_size);

    for(int i = 0;i < current_size; i++){
        printf("%d ",sayilar[i]);
    }
    printf("\n");
    return 0;
}
```

BİNARY SEARCH(İKİLİ ARAMA ALGORİTMASI)

Binary search, bir arama algoritmasıdır. Linear search'ten farkı çok daha verimli olmasıdır.
2^30 tane elemanı olan bir dizide linear search algoritması en kötü durumda 2*30 sorgulama yaparken, binary search kötü durumda 30 sorgulama yapar fakat verilerin sıralı olması şarttır.

Algoritmanın nasıl çalıştığına bakalım
Diyelim ki aşağıdaki dizide 6 sayısını arıyoruz

```
--------------------------
1 2 3 4 5 6 7 8 9 10
--------------------------
```

10 elemanlı bir dizide ortadaki sayısı bulmak için ilk ve son indeksin ortalamasını alalım
9 + 0 = 9
11 / 2 = 4,5
indeks değeri tam sayı olduğu için bu sayıyı int e çevirelim -> 4

aranan elemanı 4. indeksteki sayı ile karşılaştıralım

```
--------------------------
 1 2 3 4 5 6 7 8 9 10
--------------------------
 aranan = 6, array[4] = 5
```

aranan sayı ortadaki sayıdan büyük olduğu için artık sadece ortadaki sayıdan büyük olan sayılara bakacağız

```
--------------------------
 1 2 3 4 5
           6 7 8 9 10       (işlem yapacağımız kısım yarıya düştü)
--------------------------
```

artık ilk indeks orta indeksin 1 fazlası oldu ve son indeks aynı kaldı. Bu iki indeksin tekrar ortasını bulalım

5 + 9 = 14
14 / 2 = 7

7. indeksteki sayıyı aranan sayı ile karşılaştıralım

```
-------------------------
 1 2 3 4 5
             6 7 8 9 10
-------------------------
 aranan = 6      array[7] = 8
```

aranan sayı ortadaki sayıdan küçük o zaman bir sonraki adımda ortadaki sayıdan küçük sayılara bakacağız

```
-------------------------
 1 2 3 4 5
                   8 9 10
               6 7
-------------------------
```

Bu sefer de bakacağımız ilk indeks aynı kaldı ve son indeks, ortadaki sayının 1 eksiği oldu.

Tekrar orta indeksi bulalım

5 + 6 = 11
11 / 2 = 5,5 -> 5. indeks

aranan = 6   array[5] = 6;

Aranan elemanla orta indeksteki eleman eşit. EVREKA! Artık bu indeksi döndürebiliriz

İkiye bölme işlemi son indeks ilk indeksten büyük veya eşit olduğu sürece devam eder.

KOD

```c
#include <stdio.h>
#define SIZE 25

int binary_search(int array[], int size, int aranan){
    int ilk = 0, son = size - 1;
    int orta;

    while(son >= ilk){
        orta = (ilk + son) / 2;

        if(aranan == array[orta]){
            return orta;
        }
        else if(aranan > array[orta]){
            ilk = orta + 1;
        }
        else if(aranan < array[orta]){
            son = orta - 1;
        }
    }
    return -1;
}

int main(){

    int sayilar[SIZE] = {1,2,3,4,5,6,7,8,9,10};
    int current_size = 10;

    int bul = 10;

    int yer = binary_search(sayilar,current_size,bul);

    if(yer != -1){
        printf("%d\n",yer);
    }else{
        printf("bu eleman yok\n");
    }
    return 0;
```

2 BOYUTLU ARRAYLER(TWO DİMANSİONAL ARRAYS)

İki boyutlu array, bir arrayin elemanlarının da array olmasıdır. Nasıl 2D tanımlayacağımızı temel olarak görelim

tip isim[kaç tane arrayden oluşuyor][bu arraylerin eleman sayısı kaç tane]

mesela 4 tane arrayden oluşan ve her arrayde 5 tane eleman olan sayilar isimde bir int array tanımlayalım

```
int sayilar[4][5];
```

şimdi her elemana başlangıç değeri verelim

```
int sayilar[4][5] = {{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15},{16,17,18,19,20}};
```

2 boyutlu arrayler satır-sütün mantığı ile çalışmamıza olanak verir

```
int sayilar[4][5] = {
        {1,2,3,4,5},
        {6,7,8,9,10},
        {11,12,13,14,15},
        {16,17,18,19,20}
};
```

Eksik sayılar ile array tanımlayalım: bir elemana yani 4 arrayden birine hiç başlangıç değeri vermeyelim ve başlangıç değeri verdiğimiz bazı elemanlara 5 ten az sayı verelim

```
1  int main() {
2
3    int sayilar[4][5] = {
4      {1,2,3,4,5},
5      {6,7,10},
6      {11,12,13,14,15}
7  };
8
9    return 0;
10 }
```

Edit this code

Başlangıç değeri vermediğimiz elemanlar 0 değerini aldı.

Şimdi arrayimzi sayilar[][5] şeklinde tanımlayalım

```
 1  #include <stdio.h>
 2
 3  int main() {
 4
 5    int sayilar[][5] = {
 6      {1,2,3,4,5},
 7      {6,7,8,9,10},
 8      {11,12,13,14,15}
 9      };
10
11
12    return 0;
13  }
```

Bu durumda C kaç tane eleman olduğunu sayıyor ve kaç tane array varsa o kadar satır oluşturuyor.

Şimdi de sayilar[4][] şeklinde tanımlayalım

```
emre@emre-R580-R590:~/Desktop/EGITIM/bolum5$ gcc b.c -o b
b.c: In function 'main':
b.c:5:7: error: array type has incomplete element type 'int[]'
    5 |   int sayilar[4][] = {
      |       ^~~~~~~
b.c:5:7: note: declaration of 'sayilar' as multidimensional array must have bounds for all dimensions except the first
```

Hata diyor ki: Senin için sütün sayabilirim fakat kaç tane sütun olduğunu bana vermek zorundasın çünkü bir arrayde 6 bir arrayde 10 tane eleman olursa ben hangisine göre çalışacağımı bilemem

Özetle 2 boyutlu array tanımlarken ikinci [] arasına sayı vermek zorundayız

2 boyutlu bir arrayin herhangi bir indeksine nasıl ulaşacağımızı görelim
Tıpkı tek boyutlu arrayler gibi saymaya 0 dan başlıyoruz

	0	1	2	3	4
0	sayilar[0][0]	sayilar[0][1]	sayilar[0][2]	sayilar[0][3]	sayilar[0][4]
1	sayilar[1][0]	sayilar[1][1]	sayilar[1][2]	sayilar[1][3]	sayilar[1][4]
2	sayilar[2][0]	sayilar[2][1]	sayilar[2][2]	sayilar[2][3]	sayilar[2][4]
3	sayilar[3][0]	sayilar[3][1]	sayilar[3][2]	sayilar[3][3]	sayilar[3][4]

arrayin en başındaki 1 sayısı, 0. indeksteki arrayin 0. indeksinde. Bu sayıyı bastırmak için

printf(“%d\n”,sayilar[0][0]); satırını kullanacağız.
8 sayısı, 1.indeksteki arrayin 2. indeksinde. O zaman 8 sayısını bastırmak için printf(“%d\n”,sayilar[1][2]); satırını kullanacağız.

2 boyutlu arraydeki sayıları değiştirebiliriz. 20 sayısını 100 yapmak için sayilar[3][4] = 100; diyebiliriz

2D ARRAYLERDE İTERASYON

Döngüler konusunda, iç içe döngüler(nested loops) ile satır-sütün ile çalışabileceğimizi söylemiştik. 2 boyutlu arrayler de satır_sütün mantığıyla çalıştığı için iterasyon yaparken iç içe döngüler kullanacağız.

Sayilar arrayini satır sütün olarak bastıralım

```c
#include <stdio.h>
#define SATIR 4
#define SUTUN 5

int main() {

  int sayilar[SATIR][SUTUN] = {
    {1,2,3,4,5},
    {6,7,8,9,10},
    {11,12,13,14,15},
    {16,17,18,19,20}
    };

    for(int i = 0; i < SATIR;i++){
        for(int j = 0; j < SUTUN;j++){
            printf("%d ",sayilar[i][j]);
        }
        printf("\n");
    }

  return 0;
}
```

Döngüleri açıklamak gerekirse: i = 0,j = 4 olduğunda ilk arrayin 4. indeksindeki elemanını bastırıyoruz yani i değişkeni satırı, j değişkeni de sütünu belirliyor.

Elemanları başka bir şekilde bastırmak istersek, arrayi_bastir(int array[],int size) şeklinde bir fonksiyonla her bir arrayi bastırabiliriz çünkü her satırın bir array olduğunu biliyoruz.

```c
#include <stdio.h>
#define SATIR 4
#define SUTUN 5

void arrayi_bastir(int array[],int size){
    for(int i = 0;i < size;i++){
        printf("%d ",array[i]);
    }
    printf("\n");
}

int main() {

  int sayilar[SATIR][SUTUN] = {
    {1,2,3,4,5},
    {6,7,8,9,10},
    {11,12,13,14,15},
    {16,17,18,19,20}
    };

    for(int i = 0; i < SATIR;i++){
       arrayi_bastir(sayilar[i],SUTUN);
    }

  return 0;
}
```

2 BOYUTLU ARRAYLERİ FONKSİYONLARA GÖNDERMEK

İki boyutlu bir arrayi yazdıran array_bastir() fonksiyon yazalım.
2 boyutlu bir arrayi parametre olarak tanımlamak istersek, tıpkı 2d bir array tanımlar gibi ilk indeksi vermeyip ikinci indeksi yazıyoruz. ilk indekse sayı verirsek bu sayı int mi diye kontrol edilir sonra da görmezden gelinir.

Void array_bastir(int array[][SUTUN], ?)

Sıra uzunlukları vermeye geldi. Sutun sayısını verdiğimiz için ek parametre olarak satır sayısı yeterli olacaktır.

Void array_bastir(int array[][SUTUN], int satir)

Fonksiyonu yazalım

```c
#include <stdio.h>
#define SATIR 4
#define SUTUN 5

void array_bastir(int array[][SUTUN],int satir){

    for(int i = 0;i < satir;i++){
        for(int j = 0;j < SUTUN;j++){
            printf("%d ",array[i][j]);
        }
        printf("\n");
    }

}

int main() {

  int sayilar[SATIR][SUTUN] = {
    {1,2,3,4,5},
    {6,7,8,9,10},
    {11,12,13,14,15},
    {16,17,18,19,20}
    };

    array_bastir(sayilar,SATIR);

  return 0;
}
```

Bu fonksiyon sadece 5 sütuna sahip 2 boyutlu arrayleri alabiliyor. Bu durumdan kurtulmak ve fonksiyonu daha kullanışlı hale getirmek için sayilar arrayinin sutun sayını belirlerken cömert olacağız ve companion variable tanımlayacağız.

```c
#include <stdio.h>

#define SATIR 4
#define SUTUN 100

void array_bastir(int array[][SUTUN],int satir,int sutun){

    for(int i = 0;i < satir;i++){
        for(int j = 0;j < sutun;j++){
            printf("%d ",array[i][j]);
        }
        printf("\n");
    }

}

int main() {

  int sayilar[SATIR][SUTUN] = {
    {1,2,3,4,5},
    {6,7,8,9,10},
    {11,12,13,14,15},
    {16,17,18,19,20}
    };
    int comp_sutun = 5;

    array_bastir(sayilar,SATIR,comp_sutun);

  return 0;
}
```