BÖLÜM 6 POİNTERLAR

POİNTER NEDİR

İlk derslerimizde değişken tanımlamanın hafızada yer ayırma olduğunu söylemiştik. Diyelim ki bir int değişkeni tanımlıyoruz ve hafızada ayrılan yer 1000. byte'tan başlıyor.
1000 1001 1002 1003. byte'lar int değişkenini tutuyor. Değişkenin tutulduğu byte'ların ilkine "değişkenin adresi" diyoruz ve bu adresi (1000. byte) pointer adlı değişkenlerde tutabiliyoruz.

POİNTER TANIMLAMA

aşağıdaki yapıyla pointer tanımlayabiliriz:

adresi_tutulacak_değişkenin_türü *pointer_ismi = adres ;

yapıda gördüğünüz * işareti tanımladığımız değişkenin bir pointer olduğunu belirtiyor.

Diyelim ki "sayi" isminde bir int değişkenimiz var ve bu değişkenin adresini "sayi_ptr" isminde bir pointera atıyoruz

int sayi = 5;

int *sayi_ptr = ?;

peki bir değişkenin adresini nasıl ifade ederiz. Uzun zamandır kullandığımız scanf fonksiyonuna bir göz atalım

scanf("%d", &sayi);

bu satır bildiğimiz gibi kullanıcıdan alınan değeri sayi isimli değişkenin adresine koyuyordu. O zaman bir değişkenin adresini, değişkenin başına & koyarak elde edebiliriz.

O halde:

int *sayi_ptr = &sayi;

artık pointerımız bir değer (adres değeri) tutuyor. Bu değeri printf ve "%p" dönüşüm belirteciyle yazdırabiliriz.

```
printf("%p",sayi_ptr);
```

```
emre@emre-R580-R590:~/Desktop/EGITIM/bölüm6$ ./ders1
0x7fff796c2c0c
emre@emre-R580-R590:~/Desktop/EGITIM/bölüm6$ ./ders1
0x7fffa7d13f6c
```

gördüğünüz gibi her çalıştırdığımızda farklı bir çıktı alıyoruz çünkü her program çalıştığında sayi değişkeni için farklı bir adres alıyor. Adres değerini 162lık tabanda yazıyoruz.

DEREFERENCİNG

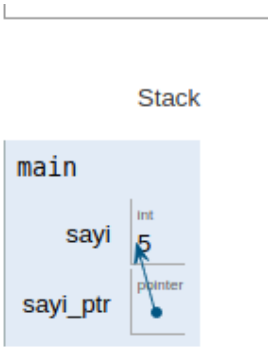
adres elde etmeyi yazdırmayı öğrendik fakat işimiz çoğunlukla o adresteki değeri gerektirecek. Pointerın tuttuğu adresteki değeri pointer isminin hemen soluna * işareti koyarak elde edebiliriz. sayi_ptr pointerının tuttuğu adresteki değişkenin değerini bastıralım.

sayi_ptr int değerini tutuyor bu yüzden yazdırmak için %d belirtecini kullanacağız.

printf(“%d\n”,*sayi_ptr);
burada kullandığımız * işareti ilk gördüğümüz * dan farklı. Tanımlama yaparken kullandığımız *, bir pointer tanımladığımızı; ondan sonra kullandığımız * ise o pointerın tuttuğu adresteki değişkenin değerini ifade ediyor.

Pointerların bir adres tuttuğunu söyledik yine de anlaşılması kolay olsun diye “pointer o değişkenin adresini tutuyor” demek yerine “pointer o değişkeni gösteriyor/işaret ediyor” deriz. Gelin bu işaret etmeyi c tutor da görelim.

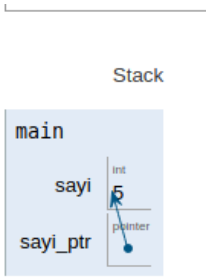
```c
#include <stdio.h>

int main(){

    int sayi = 5;
    int *sayi_ptr = &sayi;

    printf("%p\n",sayi_ptr);

    return 0;
}
```



sayi_ptr, sayi değişkenini bir okla gösteriyor yani sayi değişkeninin adresini tutuyor.

*sayi_ptr ifadesini bir değişken gibi kullanabiliriz mesela 10 arttırabiliriz.

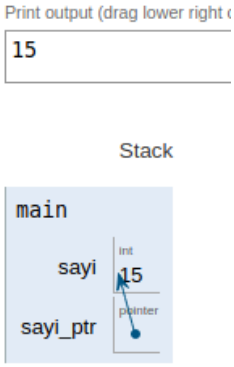
```
1  #include <stdio.h>
2
3  int main(){
4
5      int sayi = 5;
6      int *sayi_ptr = &sayi;
7      *sayi_ptr += 10;
8
9      printf("%d\n",sayi_ptr);
10
11     return 0;
12 }
```



Bir sonraki adımda 7. satır çalışacak.

C (gcc 4.8, C11)
(known limitations)

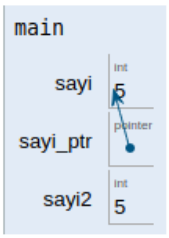
```
1  #include <stdio.h>
2
3  int main(){
4
5      int sayi = 5;
6      int *sayi_ptr = &sayi;
7      *sayi_ptr += 10;
8
9      printf("%d\n",*sayi_ptr);
10
11     return 0;
12 }
```



7 adım çalışıyor ve sayi değişkeninin değeri 15 oluyor. Bu adımda neler olduğunu söyleyelim.
“sayi_ptr nin gösterdiği değişkenin değerini 10 arttır”. sayi_ptr, sayi değişkenini gösterdiği için sayi değişkeninin değeri 10 arttırılıyor ve çıktı olarak 15 alıyoruz.

*sayi_ptr ifadesiyle toplama çıkarma gibi işlemler yapmanın yanı sıra başka bir değişkene de atabiliriz

```
1  #include <stdio.h>
2
3  int main(){
4
5      int sayi = 5;
6      int *sayi_ptr = &sayi;
7      int sayi2 = *sayi_ptr;
8
9      return 0;
10 }
```



7. satırın aslında:
int sayi2 = 5; yazmaktan bir farkı yok

DERS 2

Bu derste pointer tanımlamayla ilgili bir takım detaylara göz edeceğiz

1)Aşağıdaki kodu inceleyin. Çıktı ne olur.

```c
#include <stdio.h>

int main(){

    int sayi = 5;
    int *ptr1 = &sayi;
    int *ptr2 = &sayi;

    *ptr1 = *ptr1 + 5;

    printf("%d",*ptr2);

    return 0;
}
```

Çıktı olarak 10 alıyoruz. Sebebini c tutorda görelim

```c
1  #include <stdio.h>
2
3  int main(){
4
5      int sayi = 5;
6      int *ptr1 = &sayi;
7      int *ptr2 = &sayi;
8
9      *ptr1 = *ptr1 + 5;
10
11     printf("%d",*ptr2);
12
13     return 0;
14 }
```

*ptr1 i arttırdığımız zaman *ptr2 de artıyor çünkü iki pointer da aynı sayıyı gösteriyor Şöyle de ifade edebiliriz: *ptr1 5 arttırıldığında sayi değişkenin değeri 5 artıyor.

11. satırda ise “*ptr2 nin gösterdiği değişkenin değerini yazdır” diyoruz. ptr2 ifadesi, tıpkı ptr1 gibi sayi değişkenini gösteriyor ve 9. satırda değişken 5 arttırıldığı için *ptr2 nin değeri 10 oluyor.

2)int *ptr1, ptr2; dediğimizde iki tane pointer tanımlamıyoruz. Ptr2 bildiğimiz int tipinde bir değişken oluyor. Budan kurtulmak için ptr2 nin soluna da * koymalısınız int *ptr1, *ptr2; gibi
yine de ben pointer tanımlarını farklı satırlarda yapıyorum

int *ptr1;
int *ptr2; şeklinde

3)pointerlara başlangıç değeri vermeyebiliriz mesela

int *ptr = &sayi;

yerine

int *ptr; şeklinde. Bu durumda pointer rastgele bir adres tutar ve bu adreste başka bir program tarafından kullanılan bir veri olabileceği için c başlangıç değeri verilmemiş pointerları (uninitialized pointer) kullanmamıza izin vermez.

```c
#include <stdio.h>

int main(){

    int sayi = 5;
    int *ptr;

    printf("%d\n",*ptr);

    return 0;
}
```

```
emre@emre-R580-R590:~/Desktop/EGITIM/bölüm6$ ./ders2
Parçalama arızası (çekirdek döküldü)
```

eğer mac veya linux kullanıyorsanız bu hatayı “segmentation fault (core dumped)” olarak görebilirsiniz. Benim linux terminalim türkçe olduğu için bu hatayı ben Türkçe olarak alıyorum.

Segmentation fault hataları birçok durumda ortaya çıkabilir ama temel olarak hafızaya izni olmayan (illegal kelimesi ile de ifade ediliyor bu durum) bir iş yaptırmak olarak düşünebilirsiniz

Windowsta ise muhtemelen bu yazıyı göremeyeceksiniz ama siz de benim gibi bir çıktı alamayacaksınız.

4)başlangıç değeri almayan pointerın tehlikeli olduğunu belirttik yine de bir başlangıç değeri kullanmak isterseniz <stdio.h> kütüphanesinde tanımlı olan “NULL” makrosunu kullanabilirsiniz. Bu durumda hiçbir yeri göstermeyen pointer elde edersiniz.
Tehlikeden kurtulsanız da NULL pointerlar bi yeri göstermediği için kullanacak bir değeriniz olmaz ve *ptr ile içindeki değere ulaşmak istediğinizde segmantation fault alırsınız yine de başlangıç değeri olarak NULL vermek iyi bir tercihtir

```c
#include <stdio.h>

int main(){

    int *ptr = NULL;

    printf("%d\n",*ptr);

    return 0;
}
```

```
emre@emre-R580-R590:~/Desktop/EGITIM/bölüm6$ ./ders2
Parçalama arızası (çekirdek döküldü)
```

5)farklı tipteki pointerları birbirine atayamazsınız. İnt pointera double veya char pointer atayamazsınız veya tam tersini de yapamazsınız aksi takdirde durumda uyarı alırsınız

```c
#include <stdio.h>

int main(){

    int *ptr = NULL;
    double *ptr2 = NULL;

    ptr = ptr2;

    return 0;
}
```

```
emre@emre-R580-R590:~/Desktop/EGITIM/bölüm6$ gcc ders2.c -o ders2
ders2.c: In function ‘main’:
ders2.c:8:9: warning: assignment to ‘int *’ from incompatible pointer type ‘double *’ [-Wincompatible-pointer-types]
    8 |     ptr = ptr2;
      |         ^
```

Şimdi pointer kavramını daha iyi anlamanız için bir adet basit soru çözeceğiz.
SORU: iki sayı alın ve büyük olanı yazdırın

```c
#include <stdio.h>

int main(){

    int sayi1;
    int sayi2;
    int *ptr1 = &sayi1;
    int *ptr2 = &sayi2;

    printf("iki tane sayi girin: ");
    scanf("%d %d", ptr1, ptr2);

    int buyuk;
    if(*ptr1 > *ptr2){
        buyuk = *ptr1;
    }
    else{
        buyuk = *ptr2;
    }

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

    return 0;
}
```

scanf kullanırken &sayi1 ve &sayi2 demek yerine pointerları yazdık. Bunu iki şekilde açıklayabiliriz:
1)&sayi1 dediğimizde, sayi1 değişkeninin adresinden bahsediyoruz. Ptr1 in değeri sayi1 değişkenin adresi olduğu için & kullanmaya gerek yok
2)ptr1 i zaten &sayi1 olarak belirledik

CALL BY REFERANCE

daha önce call by value kavramını öğrenmiştik. Aşağıdaki koda bakalım

```c
#include <stdio.h>

void ikiyle_carp(int sayi){
    sayi *= 2;
}

int main(){

    int sayi = 5;

    ikiyle_carp(sayi);

    printf("sayi x 2 = %d\n",sayi);

    return 0;
}
```

Çıktının 5 olacağını biliyoruz çünkü ikiyle_carp fonksiyonu sayi değişkeninin değerini 2 ile çarpıyor yani fonksiyonun çağrıldığı satır aslında ikiyle_carp(5) olarak çalışıyor ve biz main içinde tanımladığımız sayi değişkenini değiştiremiyoruz. Bu sorundan kurtulmanın bir yolunun return etmek olduğunu söylemiştik.

Diğer çözüm ise call by referance dediğimiz, fonksiyona değişkenin değerini değil değişkenin adresini göndermek. Adres kelimesi geçtiğine göre pointer kullanacağız.

```c
#include <stdio.h>

void ikiyle_carp(int *ptr){
    *ptr *= 2;
}

int main(){

    int sayi = 5;

    ikiyle_carp(&sayi);

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

    return 0;
}
```

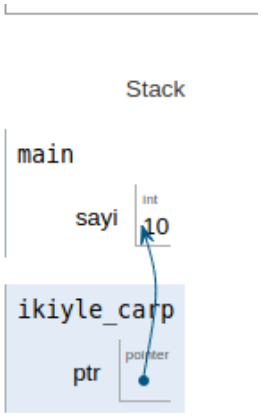
Çıktı olarak 10 alıyoruz demek ki fonksiyon sayi değişkeninin değerini değiştirmiş. Peki bunu nasıl yapmış?

Fonksiyona değişkenin adresini gönderdik (&sayi)

fonksiyon o adresteki değeri iki ile çarptı.

O adresin ismi sayi olduğu için sayi isimli değişken 2 ile çarpıldı

```c
#include <stdio.h>

void ikiyle_carp(int *ptr){
    *ptr *= 2;
}

int main(){

    int sayi = 5;

    ikiyle_carp(&sayi);

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

    return 0;
}
```



ufak bir örnek yapalım:
iki sayının toplamını bulan void topla(int *s1,int *s2, int *toplam) isimli fonksiyonu yazınız

```c
#include <stdio.h>

void topla(int *s1,int *s2, int *toplam){
    *toplam = *s1 + *s2;
}

int main(){

    int sayi1 = 5,sayi2 = 10,toplam;
    topla(&sayi1, &sayi2, &toplam);
    printf("%d\n",toplam);

    return 0;
}
```

ARRAYLER VE POİNTERLAR

Pointerlar ile arraylerin arasında bir ilişki vardır
Bir arrayin ismi o arrayin ilk indeksini gösteren bir pointerdır

```c
#include <stdio.h>

int main(){

    int array[] = {0,1,2,3,4};

    printf("array: %p\n",array);
    printf("&array[0]: %p\n\n",&array[0]);
    printf("*array: %d\n",*array);
    printf("array[0]: %d\n",array[0]);

    return 0;
}
```

Çıktı:

```
array: 0x7fff12eb1540
&array[0]: 0x7fff12eb1540

*array: 0
array[0]: 0
```

Peki başka bir indekse nasıl erişeceğiz.
1. indekse erişmek için

printf(“%d\n”, *(array + 1)); yazmamız yeterli. Tahmin edeceğiniz üzere bu + 1 normal 1 sayısı değil.
array + 1 ifadesi aslında
array + 1 * (arrayin tuttuğu tipin boyutu)
array + 1 * (int tipinin boyutu)
array + 4 byte olarak işlem görüyor

arrayin ilk indeksinden 4 byte sonra arrayin 2. indeksine erişebiliyoruz çünkü arrayin elemanları ard arda yerleştirilir.

Sonuç olarak n. indekse erişmek için *(array + n) kalıbını kullanabiliriz.

*(array) ifadesi de aslında *(array + 0) -> 0. indeks olarak yorumlanabilir

---

Biliyoruz ki aynı tipte iki pointerı birbirine atayabiliriz. Arrayin ismini de başka bir pointera atayabiliriz.

int *ptr = array;

ptr ifadesi de arrayin ilk elemanını gösteriyor. Yukarda yaptığımızı bu ifadeyle de yapabiliriz.

```
int array[] = {0,1,2,3,4};
int *ptr = array;

printf("array: %p\n",array);
printf("ptr: %p\n\n",ptr);
printf("*array: %d\n",*array);
printf("*ptr: %d\n",*ptr);
```

```
array: 0x7fff9b9d6250
ptr: 0x7fff9b9d6250

*array: 0
*ptr: 0
```

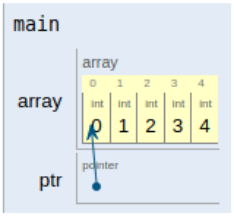
C (gcc 4.8, C11)
(known limitations)

```
1  #include <stdio.h>
2
3  int main(){
4
5      int array[] = {0,1,2,3,4};
6      int *ptr = array;
7
8      printf("array: %p\n",array);
9      printf("ptr: %p\n\n",ptr);
10     printf("*array: %d\n",*array);
11     printf("*ptr: %d\n",*ptr);
12
13     return 0;
14 }
```

Print output (drag lower right corner to resize)

```
array: 0xfff000bc0
ptr: 0xfff000bc0

*array: 0
```

Stack Heap



n. indekse de *(ptr + n) şeklinde erişebiliriz

```
int array[] = {0,1,2,3,4};
int *ptr = array;

printf("2.indeks: %d\n", *(ptr + 2));
```

```
2.indeks: 2
```

PRE-İNCREMENT

daha önce

sayi++;

gibi bir ifade gördük siz

++sayi;

gibi bir ifade görebilirsiniz.

İki kod da sayi değişkenin değerini 1 arttırıyor ama
değişkenden önce ++ koyduğumuz zaman önce değişkeni 1 arttırıyor sonra değişkeni kullanıyor
değişken sonra ++ koyduğumuz zaman önce değişkeni kullanıyor sonra 1 arttırıyor.

```
int sayi = 5;
printf("%d\n", sayi++);
//çıktı 5, sayinin değeri 6
printf("%d\n", ++sayi);
//sayinin değeri 7, çıktı 7
```

ÇIKTI:
5
7

Bu operatörler 1 arttırma yaptığına ve pointerlarla toplama-çıkarma işlemi yaptığına göre ++ ve -- operatörlerini pointerlarla kullanabiliriz.

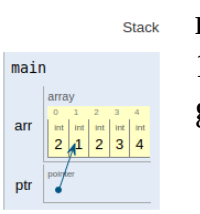
Kod üzerinden bakalım

```
int arr[] = {0,1,2,3,4};
int *ptr = arr;

//ÖNCE PARANTEZ İÇİ YAPILIR

printf("%d\n",(*ptr)++);
/*0. indeksi yazdır sonra 1 arttır
0. indekste şu an 1 var
çıktı: 0*/

printf("%d\n",++(*ptr));
/*0.indeksi 1 arttır sonra yazdır
0. indekste şu an 2 var
çıktı: 2*/
```

```
printf("%d\n",*(ptr++));
/*0. indeksi yazdır sonra bir sonraki indekse geç
şu an ptr arrayin 1. indeksi gösteriyor
çıktı: 2*/
```

```
11    /*0. indeksi yazdır sonra 1 arttır
12    0. indekste şu an 1 var
13    çıktı: 0*/
14
15    printf("%d\n",++(*ptr));
16    /*0.indeksi 1 arttır sonra yazdır
17    0. indekste şu an 2 var
18    çıktı: 2*/
19
20    printf("%d\n",*(ptr++));
21    /*0. indeksi yazdır sonra bir sonraki indekse geç
22    şu an ptr arrayin 1. indeksi gösteriyor
23    çıktı: 2*/
24
25    printf("%d\n",*(++ptr));
26    /*bir sonraki indekse geç sonra o indeksi yazdır
27    şu an ptr arrayin 2.indeksini gösteriyor
28    çıktı: 2*/
29    return 0;
30 }
```

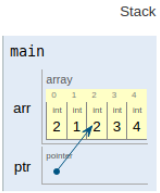
```
2
2
```



mavi kutucukta ok işaretinin 1. indeksi gösterdiğini görebilirsiniz

```
printf("%d\n",*(++ptr));
/*bir sonraki indekse geç sonra o indeksi yazdır
şu an ptr arrayin 2.indeksini gösteriyor
çıktı: 2*/
```

```
15    printf("%d\n",++(*ptr));
16    /*0.indeksi 1 arttır sonra yazdır
17    0. indekste şu an 2 var
18    çıktı: 2*/
19
20    printf("%d\n",*(ptr++));
21    /*0. indeksi yazdır sonra bir sonraki indekse geç
22    şu an ptr arrayin 1. indeksi gösteriyor
23    çıktı: 2*/
24
25    printf("%d\n",*(++ptr));
26    /*bir sonraki indekse geç sonra o indeksi yazdır
27    şu an ptr arrayin 2.indeksini gösteriyor
28    çıktı: 2*/
29    return 0;
30 }

Edit this code
line that just executed
next line to execute
```



mavi kutucukta ok işaretinin artık 2. indeksi gösterdiğini görebilirsiniz

---

NOT: ptr isimli pointerı arttırıp azaltabiliriz fakat bu işlemleri arrayin ismi ile yapamayız:
arr++; ifadesi hata verecektir

NOT: ++ ve -- operatörleri * (dereferencing) operatöründen önceliklidir bu yüzden: *ptr++; ifadesi önce bir sonraki indekse geçer sonra indeksteki elemana erişir. Ayrıca bu ifade okunması zor olduğu için * ve ++ ifadelerini aynı anda kullanacağınız zaman mutluka parantez kullanın

POİNTERLARLA İTERASYON

Daha önce her arrayin aslında birer pointer olduğu söydik. Bu derste bu ilişkiden faydalanarak arraylerde pointer yardımıyla iterasyon yapacağız.

Bir arrayin elemanlarını 3 farklı yöntemle bastralım:

1. YÖNTEM

0. indeks için *(array + 0)
1. indeks için *(array + 1)
1. indeks için *(array + 2)
.
.
n. indeks için *(array + n)

yapısını kullanabileceğimizi biliyoruz. İndekslere sırayla erişmek için for döngüsü ile *(array + i) yapısını kullanabiliriz

```
int array[] = {0,1,2,3,4,5,6,7,8,9};

for(int i = 0;i < SIZE;i++){
    printf("%d ",*(array + i));
}
```

2.YÖNTEM

bir önceki yönteme çok benziyor. Sadece arrayin ismini direk kullanmak yerine başka bir pointera atıp döngüyü tekrar düzenlemek

```
int array[] = {0,1,2,3,4,5,6,7,8,9};
int *ptr = array;

for(int i = 0;i < SIZE;i++){
    printf("%d ",*(ptr + i));
}
```

Boş yere 2. bir pointer tanımladığımız ve 1. yöntemin daha okunaklı ve mantıklı olduğunu düşündüğüm için bu yöntemi pek kullanmayı tavsiye etmiyorum.

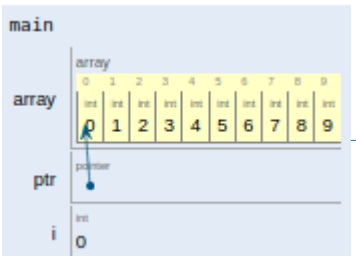
Ek bir pointer tanımlamak gerekiyorsa 3. yöntemi kullanmak daha mantıklı olabilir

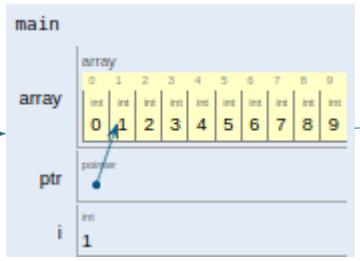
3. YÖNTEM

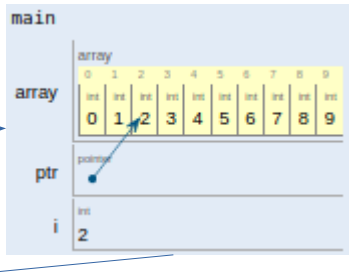
Bir önceki derste ek bir pointer ve ++ operatörü ile indeksler arası geçiş yapmıştık. Bu derste tanımladığımız ek pointer sırayla arrayin indekslerini birer birer arttırarak gösterecek.

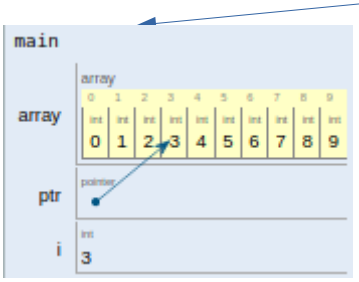
```
int array[] = {0,1,2,3,4,5,6,7,8,9};
int *ptr = array;

//uzunluk kadar tekrarlanan döngü
for(int i = 0;i < SIZE;i++){
    printf("%d ",*ptr);
    ptr++;
}
```

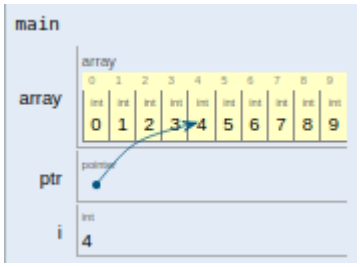
Bu yöntemi bir de c tutorda inceleyelim



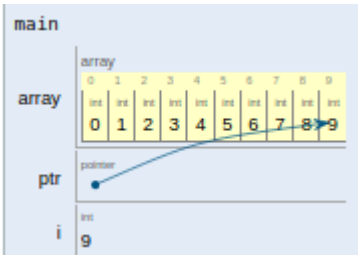




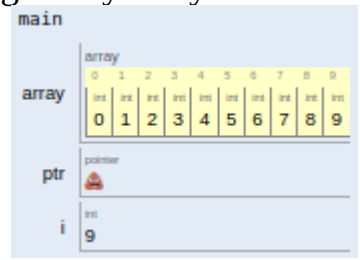




Son indeks



Son indeks yazdırıldıktan sonra ptr 1 kere daha artıyor ve arrayin dışında bir yeri gösteriyor o yeri bastırırsak işimize yaramayan bir değer alırız




```
0 1 2 3 4 5 6 7 8 9
809609472
```

İKİ BOYUTLU ARRAYLER VE POİNTERLAR

Bir önceki derse kullandığımız 1. yöntemi kullanarak 2boyutlu arrayleri pointerla kullanabiliriz

a[i][j] indeksini *(*(a + i) + j) şeklinde ifade edebiliriz.
(Açıklama isterseniz soru cevap kısmına yazabilirsiniz)

2d bir arrayi bastıralım

```c
#include <stdio.h>
#define SATIR 3
#define SUTUN 3

int main(){

    int array[SUTUN][SATIR] = {{0,1,2},{3,4,5},{6,7,8}};

    for(int i = 0;i < SATIR;i++){
        for(int j = 0;j < SUTUN;j++){
            printf("%d ",*(*(array + i) + j));

        }
        printf("\n");
    }

    return 0;
}
```

```
emre@emre-R580-R590:~/Desktop/EGITIM/bölüm6$ ./ders7
0 1 2
3 4 5
6 7 8
```

ARRAYLERİ POİNTER OLARAK FONKSİYONLARA GÖNDERME

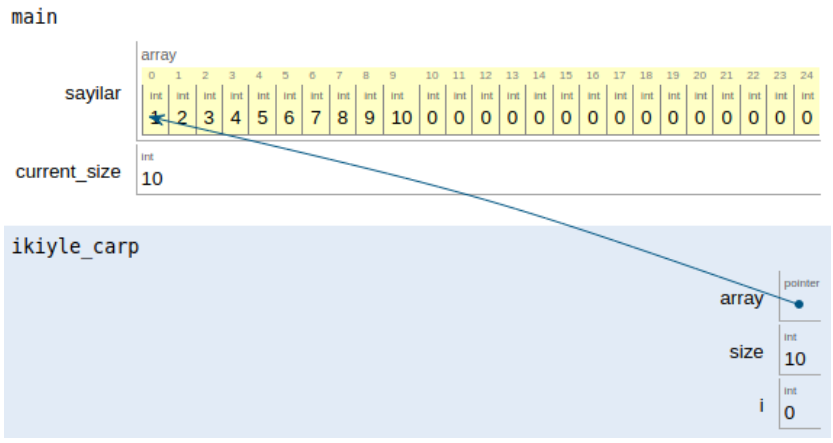
İki boyutlu arrayleri pointer olarak fonksiyonlara gönderme işlemini string konusunda yapacağız. Bu derste tek boyutlu arrayleri fonksiyonlara göndereceğiz.

Kursumuzun en önemli fonksiyonlarından biri olan ikiyle_carp fonksiyonunu hatırlayalım:

```
1   #include <stdio.h>
2   #define SIZE 25
3
4   void ikiyle_carp(int array[], int size){
5
6       for(int i = 0;i < size;i++){
7           array[i] *= 2;
8       }
9
10  }
11
12  int main(){
13
14      int sayilar[SIZE] = {1,2,3,4,5,6,7,8,9,10};
15      int current_size = 10;
16
17      ikiyle_carp(sayilar,current_size);
18
19      return 0;
20  }
```

Bu derste fonksiyon içindeki [] işaretini kullanmadan pointerlar ve dereferencing kavramı ile fonksiyona array göndereceğiz.

17. satıra bakın. Fonksiyon içine arrayin ismini göndermişiz. Arrayin ismi ilk indeksi gösteren bir pointer olduğuna göre aslında fonksiyona array gönderdiğimiz zaman hep bir pointer gönderiyoruz. Bu yüzden bu fonksiyonun c tutordaki görüntüsünde ok işareti ve pointer yazısı çıkıyor.



Şimdi ikiyle_carp fonksiyonunu pointer kullanarak tekrar düzenleyelim

1. YÖNTEM

```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));
    }

    return 0;
}
```

2. YÖNTEM

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

void ikiyle_carp(int *arr, int size){

    for(int i = 0;i < size;i++){
        *arr *= 2;
        arr++;
    }

}

int main(){

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

    ikiyle_carp(sayilar,current_size);

    int *ptr = sayilar;
    for(int i = 0;i < current_size;i++){
        printf("%d ",*ptr);
        ptr++;
    }

    return 0;
}
```

Fonksiyonun içinde dikkat etmemiz gerek bir şey var. main fonksiyonunda sayilar++ gibi bir satırla tanımladığımız bir arrayin ismini arttırıp azaltamıyoruz.

Fakat sayilar arrayini fonksiyona gönderdiğimiz zaman arr isimli pointer sayilar'ı tuttuğu için arttırılıp azaltılabiliyor.

FONKSİYON POİNTERLARI

Bu derste fonksiyonları gösteren fonksiyonları öğreneceğiz.

TANIMLAMAK

Bir fonksiyon pointerı da aşağıdaki yapıyla tanımlayabiliriz.

Tutulacak_fonksiyonun_dönüş_tipi (*isim)(fonksiyonların argümanları) = fonksiyonun adresi

ekrana “merhaba” yazan selamla isimli bir fonksiyon tanımlayalım ve fp isimli pointera gönderelim

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

void selamla(){
    printf("merhaba\n");
}

int main(){

    void (*fp)() = &selamla;

    return 0;
}
```

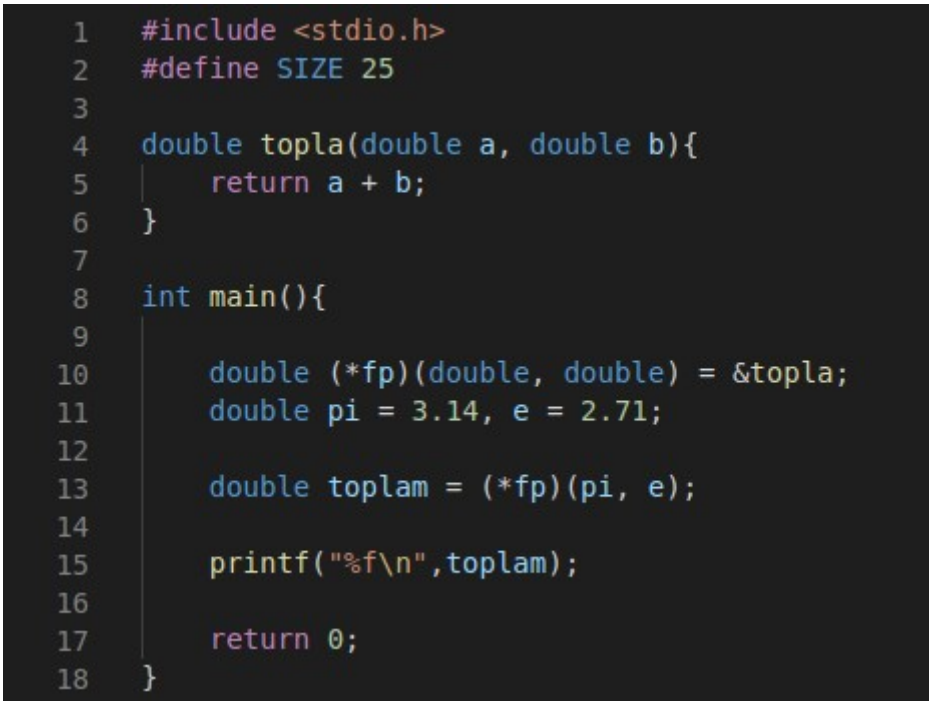
Şimdi pointer kullanarak bu fonksiyonu çağıralım

Bir parantez içinde dereferencing yapıp, ikinci bir parantezde fonksiyonun argümanlarını gönderelim. Selamla fonksiyonunun argümanı olmadığı için ikinci parantez boş kalacak

```c
(*fp)();
```

bu satıra gelindiğinde fp nin tuttuğu fonksiyon olan selamla fonksiyonu çalışacak ve ekrana merhaba yazdırılacak

Örnek yapmak için iki sayının toplamını double olarak döndüren topla fonksiyonu yazalım ve fp isimli bir pointera atayalım



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

double topla(double a, double b){
    return a + b;
}

int main(){

    double (*fp)(double, double) = &topla;
    double pi = 3.14, e = 2.71;

    double toplam = (*fp)(pi, e);

    printf("%f\n",toplam);

    return 0;
}
```

10. satırda ikinci parantezin içine fonksiyonun alacağı parametrelerin tipini yazdık
13. satıra gelindiğinde fp nin gösterdiği fonksiyon pi ve e argümanlarını alarak çalışacak ve döndürülecek sonuç toplam isimli değişkene atanacak

FONKSİYONLARLA ARRAY OLUŞTURMAK

pointerlarla arrayler bağlantılı olduğuna göre fonksiyonlarla array oluşturabiliriz.

Diyelim ki iki sayıyı toplayan çıkaran ve çarpan topla cıkar ve carp fonksiyonlarımızla array oluşturmak ve bu arraye pointer yardımıyla erişmek istiyoruz.

```
double topla(double a, double b){
    return a + b;
}
double cikar(double a, double b){
    return a - b;
}
double carp(double a, double b){
    return a * b;
}
```

Fonksiyonlardan oluşan bir array tanımlamak içinse (*fp[]) yapısını kullanabiliriz. İkinci parantez parametrelerin türlerini
parantez içine ise fonksiyon isimlerini yazmak yeterlidir.

```
double (*fp[])(double, double) = {topla, cikar, carp};
```

Arrayin içindeki fonksiyonlardan birini kullanmak için köşeli parantezin içine istediğimiz fonksiyonun indeksini yazıyoruz ve 2. parantezle argümanları gönderiyoruz.

Pi ve e sayısını çarpmak için

```
int sonuc = (*fp[2])(pi, e);
```

bunun bize faydası ise kodun kısalması. Diyelim ki bu fonksiyonları seçmesi için kullanıcıdan 0 1 veya 2 girmesini istiyoruz.

```
if(secim == 0)
        int sonuc = topla(sayi1 , sayi2);
else if(secim == 1)
        int sonuc = cıkar(sayi1 , sayi2);
else if(secim == 2)
        int sonuc = carp(sayi1 , sayi2);
```

diye ayrı ayrı yazmak yerine bir satırda işimizi bitirebiliriz

FONKSİYON POİNTERINI FONKSİYONA GÖNDERMEK

Fonksiyon pointerları kullanarak bir fonksiyona başka bir fonksiyonu gönderebiliriz topla cikar ve carp fonksiyonlarından birini ve 2 tane sayı alan
double hesapla(double (*fp)(double, double), double a, double b) fonksiyonunu yazalım

```c
#include <stdio.h>

double topla(double a, double b){
    return a + b;
}
double cikar(double a, double b){
    return a - b;
}
double carp(double a, double b){
    return a * b;
}

double hesapla(double (*fp)(double, double), double a, double b){
    double sonuc = (*fp)(a,b);
    return sonuc;

}


int main(){

    double (*fp[])(double, double) = {topla, cikar, carp};

    double pi = 3.14, e = 2.71;

    double sonuc = hesapla(carp, pi, e);
    printf("sonuc: %f\n",sonuc);

    return 0;
}
```

hesapla fonksiyonuna istediğimiz fonksiyonun ismini göndermemiz yeterli olacaktır ama örnekteki carp fonksiyonunu isim yerine indeks olarak da gönderebilirsiniz. Yani 26. satırı şu şekilde de yazabilirsiniz

```c
double sonuc = hesapla((*fp[2]), pi, e);
```