

Sistem Programlama

DR. ÖĞR. ÜYESİ ABDULLAH SEVİN

İçerik

a) Segmentasyon ihlalleri ve Veri Yolu Hataları

b) Little-Endian

c) Struct'ların hafızada yerleşimi

d) Pointer aritmetiği

e) String işlemler

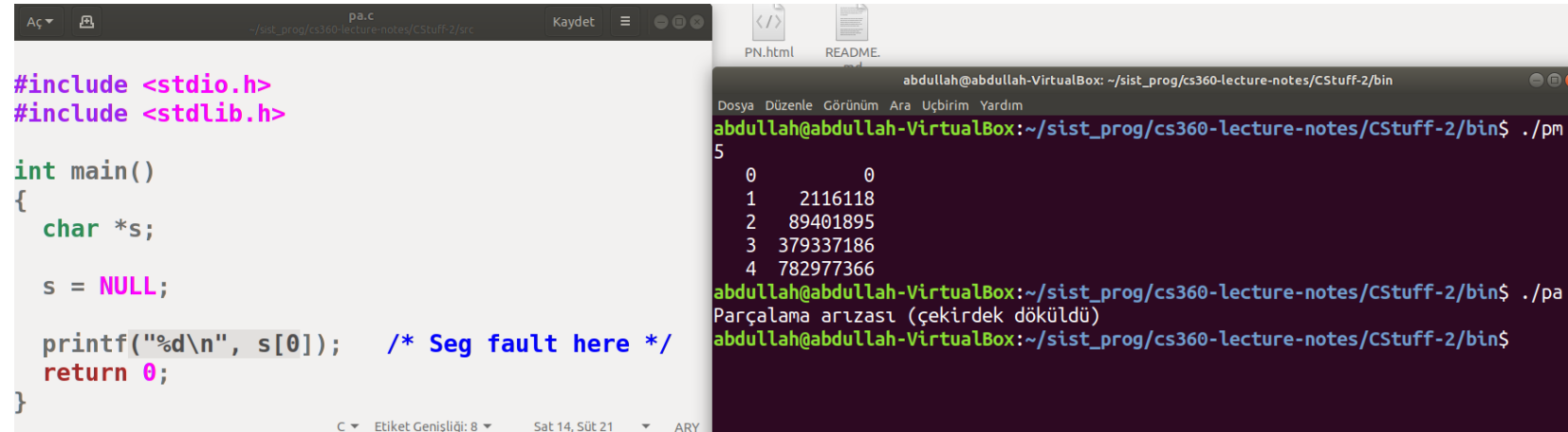
- <http://web.eecs.utk.edu/~jplank/plank/classes/cs360/360/notes/CStuff-2/lecture.html>
- <http://web.eecs.utk.edu/~jplank/plank/classes/cs360/360/notes/Pointer-Arithmetic/index.html>
- <http://web.eecs.utk.edu/~jplank/plank/classes/cs360/360/notes/Strings-In-C/index.html>

Segmentasyon ihlalleri

- ❑ Bellek, dev bir bayt dizisidir.
- ❑ Ancak, bu dizinin belirli bölümlerine erişilemez.
- ❑ Örneğin, genelde 0 - 0x1000 öğelerine erişilemez.
- ❑ Erişilemeyen bir öğeye erişmeye çalıştığınızda, bir segmentasyon ihlali oluşturursunuz.
- ❑ (Eskiden, belleğin içeriğini çekirdek dökümü (core dump) adı verilen bir dosyada da depolardık. Bu günlerde bellek o kadar büyük ki, yapabilmemize rağmen çekirdek dökümleri (core dump) oluşturmuyoruz.)
- ❑ Segmentasyon ihlali; bir pointerı başlatmayı unuttuğumuzda ve ilk adresini yazdırmak istersek.

Segmentasyon ihlalleri

❏ Örnek:



The image shows a code editor on the left and a terminal window on the right. The code editor displays a C program named `pa.c` with the following content:

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

int main()
{
    char *s;

    s = NULL;

    printf("%d\n", s[0]);    /* Seg fault here */
    return 0;
}
```

The terminal window shows the execution of the program. It starts with the prompt `abduallah@abduallah-VirtualBox: ~/sist_prog/cs360-lecture-notes/CStuff-2/bin`. The user runs `./pm`, and the output is:

```
5
0      0
1    2116118
2    89401895
3    379337186
4    782977366
```

After this output, the user runs `./pa`, and the terminal displays the error message: `Parçalama arızası (çekirdek döküldü)`, which translates to "Segmentation fault (core dumped)".

Veri Yolu Hataları

- ❑ Birçok makinede, bir skaler türe her eriştiğinizde, bellekteki değeri sıralı olmalıdır.
- ❑ Bunun anlamı, veri türü 4 bayt ise, bellekteki konumu 4'ün katı olan bir bellek dizininde başlamalıdır.
- ❑ Örneğin, `i` bir `(int *)` ise, o zaman `i` 4'ün katı değilse , bu hatalara sebebiyet verecektir. Bu hata, bir veri yolu hatasıyla kendini gösterir.

Little Endian

- ❑ C ile, bellek ile ilgili "tehlikeli" şeyler yapabilirsiniz.
- ❑ Başka bir deyişle, bir bayt bölgeniz olduğunu varsayalım. Bu bölgeye, istediğiniz herhangi bir türü tutabiliriz- tamsayılar, karakterler, double, struct vs.
- ❑ Tipik olarak, bu esneklikten yararlanmak istemezsiniz, çünkü bu sizi çok fazla belaya sokabilir.
- ❑ Ancak, sistem programları yazarken, bu esneklik genellikle önemlidir.
- ❑ Ayrıca, programlar çalışırken bellekte ve makinenizde neler olup bittiğini anlamanız önemlidir!

Little Endian

```
#include <stdio.h>

typedef unsigned long UL;

int main()
{
    unsigned int array[4]; /* An array of four integers. */
    unsigned int *ip;      /* An integer pointer that we're going to set to one byte beyond array */
    unsigned char *cp;     /* An unsigned char pointer for exploring the individual bytes in array */
    unsigned short *sp;    /* An unsigned short to show two-byte access. */
    int i;

    /* Set array to equal four integers, which we
```

```
abduallah@abduallah-VirtualBox:~/sist_prog/cs360-lecture-notes/CStuff-2/bin$ ./end
ian
Array[0]'s location in memory is 0x7ffda6bd4960. Its value is 0x12345678
Array[1]'s location in memory is 0x7ffda6bd4964. Its value is 0x9abcdef0
Array[2]'s location in memory is 0x7ffda6bd4968. Its value is 0x13579bdf
Array[3]'s location in memory is 0x7ffda6bd496c. Its value is 0x2468ace0
```

```
/* For each value of array, print it out in hexadecimal. Also print out its location in memory. */

for (i = 0; i < 4; i++) {
    printf("Array[%d]'s location in memory is 0x%lx. Its value is 0x%x\n",
        i, (UL) (array+i), array[i]);
}
```

Little Endian

```
/* Now, print out the sixteen bytes as bytes, printing each byte's location first. */

printf("\n");
printf("Viewing the values of array as bytes:\n");
printf("\n");

cp = (unsigned char *) array;

for (i = 0; i < 16; i++) {
    printf("Byte %2d. Pointer: 0x%lx - Value: 0x%02x\n", i, (UL) (cp+i), cp[i]);
}
```

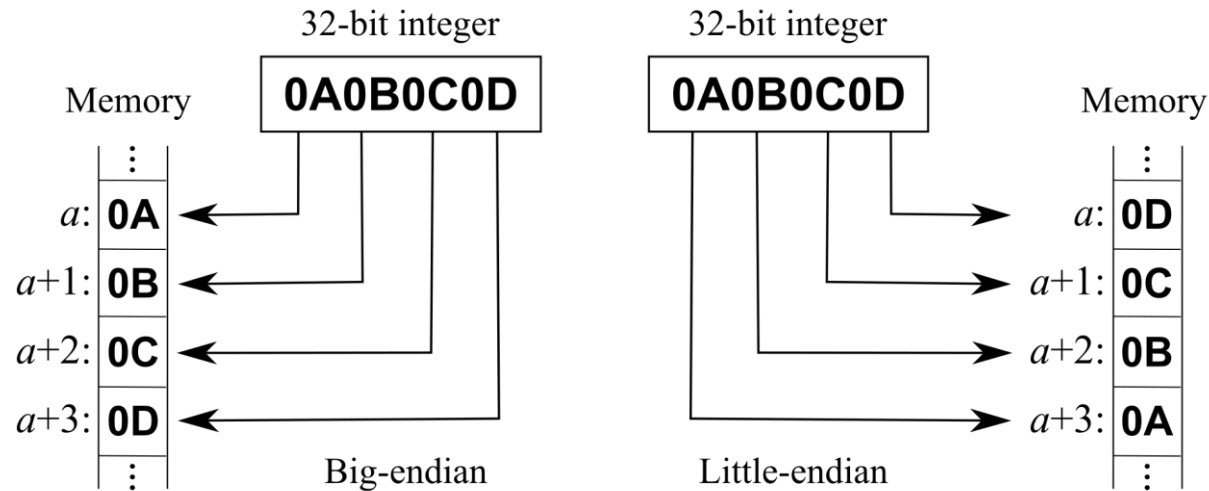
- ❑ Çıktıya, özellikle bayt değerlerine dikkat edin. İlk başta karışık gelebilir!
- ❑ İlk baytın 0x12 olmasını beklemiyor muydunuz?

Viewing the values of array as bytes:

Byte	0.	Pointer:	0x7ffda6bd4960	- Value:	0x78
Byte	1.	Pointer:	0x7ffda6bd4961	- Value:	0x56
Byte	2.	Pointer:	0x7ffda6bd4962	- Value:	0x34
Byte	3.	Pointer:	0x7ffda6bd4963	- Value:	0x12
Byte	4.	Pointer:	0x7ffda6bd4964	- Value:	0xf0
Byte	5.	Pointer:	0x7ffda6bd4965	- Value:	0xde
Byte	6.	Pointer:	0x7ffda6bd4966	- Value:	0xbc
Byte	7.	Pointer:	0x7ffda6bd4967	- Value:	0x9a
Byte	8.	Pointer:	0x7ffda6bd4968	- Value:	0xdf
Byte	9.	Pointer:	0x7ffda6bd4969	- Value:	0x9b
Byte	10.	Pointer:	0x7ffda6bd496a	- Value:	0x57
Byte	11.	Pointer:	0x7ffda6bd496b	- Value:	0x13
Byte	12.	Pointer:	0x7ffda6bd496c	- Value:	0xe0
Byte	13.	Pointer:	0x7ffda6bd496d	- Value:	0xac
Byte	14.	Pointer:	0x7ffda6bd496e	- Value:	0x68
Byte	15.	Pointer:	0x7ffda6bd496f	- Value:	0x24

Little-Endian vs Big-Endian

- ❑ Little-Endian: Verinin düşük değerlikli bitleri bellekte ilk sıralanır.
- ❑ Big-Endian: Verinin yüksek değerlikli bitleri bellekte ilk sıralanır.
- ❑ Aynı Arapça veya Türkçedeki sağdan-soldan yazım kuralı gibi işlemcilerde özgü bellek yazım biçimi



Little-Endian

<u>Memory</u>		array = 0xbff344a8				(array+2) = 0xbff344b0				(array+1) = 0xbff344ac				(array+3) = 0xbff344b4					
		↓				↓				↓				↓					
Addresses	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		
	b	b	b	b	b	b	b	b	b	b	b	b	b	b	b	b	b		
	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f		
	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3		
	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4		
Values as bytes (hex)	a	a	a	a	a	a	a	a	b	b	b	b	b	b	b	b	b		
	7	8	9	a	b	c	d	e	f	0	1	2	3	4	5	6	7	8	
		?	78	56	34	12	f0	de	bc	9a	df	9b	57	13	e0	ac	68	24	?
		array[0] as an integer is: 0x12345678				array[1] as an integer is: 0x9abcdef0				array[2] as an integer is: 0x13579bdf				array[3] as an integer is: 0x2468ace0					

- ❑ Her baytın adresi ve değeri onaltılık olarak listelenmiştir. Daha sonra dört işaretçiyi dizi, (dizi+1), (dizi+2) ve (dizi+3) olarak etiketlenmiştir.

Little-Endian

- ❑ Şimdi cp++ yaptık ve ip, (ip+1), (ip+2) ve (ip+3) tarafından gösterilen dört tam sayıyı yazdırıyoruz.

```
/* Finally, set the pointer ip to be one byte greater than array,  
   and then print out locations and integers. */  
  
printf("\n");  
printf("Setting the pointer ip to be one byte greater than array:\n");  
printf("\n");  
  
cp++;  
ip = (unsigned int *) cp;  
for (i = 0; i < 4; i++) {  
    printf("(ip+%d) is 0x%lx. *(ip+%d) is 0x%x\n", i, (UL) (ip+i), i, *(ip+i));  
}
```

Setting the pointer ip to be one byte greater than array:

```
(ip+0) is 0x7ffda6bd4961. *(ip+0) is 0xf0123456  
(ip+1) is 0x7ffda6bd4965. *(ip+1) is 0xdf9abcde  
(ip+2) is 0x7ffda6bd4969. *(ip+2) is 0xe013579b  
(ip+3) is 0x7ffda6bd496d. *(ip+3) is 0x602468ac
```

- ❑ Bazı makinelerde, ip dördün katı olmadığı için bu kodda bir veri yolu hatası olacaktır. Çıktıyı kafa karıştırıcı bulabilirsiniz ama merak etmeyin, üzerinden geçeceğiz

Little-Endian

Ayrıca, 0x612468ac'deki "61" orijinal diziden olmayan bir bayttır. Başka bir değeri olabilir - ne olması gerektiğini gerçekten bilemeyiz. (Benim kodda 60 çıktı)

<u>Memory</u>		ip = 0xbff344a9				(ip+2) = 0xbff344b1												
		(ip+1) = 0xbff344ad								(ip+3) = 0xbff344b5								
Addresses	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
	b	b	b	b	b	b	b	b	b	b	b	b	b	b	b	b	b	
	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	
Values as bytes (hex)	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	
	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	
	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	
	a	a	a	a	a	a	a	a	b	b	b	b	b	b	b	b	b	
	7	8	9	a	b	c	d	e	f	0	1	2	3	4	5	6	7	
	?	78	56	34	12	f0	de	bc	9a	df	9b	57	13	e0	ac	68	24	
			*ip				*(ip+1)				*(ip+2)				*(ip+3)			
			as an integer is:				as an integer is:				as an integer is:				as an integer is:			
			0xf0123456				0xdf9abcde				0xe013579b				0x612468ac			

Little-Endian

□ Tamsayılar gibi, short'da little-endian ile temsil edilir, bu nedenle ilk short verinin en küçük baytı 0x56 değeriyle 0xbff344a9 adresindedir ve ilk short değerin en büyük baytı 0x78 değeriyle 0xbff344a8 adresindedir:

```
/* Now, set sp to equal array.  Sp is a pointer to shorts.  We print out sp[0] and sp[1]. */

printf("\n");
printf("Finally printing the first four bytes of array as two shorts.\n");
printf("\n");

sp = (unsigned short *) array;
printf("Location: 0x%lx - Value as a short: 0x%04x\n", (UL) sp, sp[0]);
printf("Location: 0x%lx - Value as a short: 0x%04x\n", (UL) (sp+1), sp[1]);
printf("\n");

return 0;
}
```

Finally printing the first four bytes of array as two shorts.

Location: 0x7ffc9b3f6d60 - Value as a short: 0x5678
Location: 0x7ffc9b3f6d62 - Value as a short: 0x1234

Yapıların (Struct) yerleşimi

- ❑ Bazı makineler işaretçilerin bellekte belli aralıklarla hizalanmasını gerektirir.
- ❑ Bu, integer işaretçiler için dördün katları, double işaretçilerin sekizin katları ve short işaretçilerinin ikinin katları olması gerektiği anlamına gelir.
- ❑ Bu gereksinimi karşılamak için derleyiciler ve çalışma zamanı kütüphaneleri (runtime libraries) iki özellekle tasarlanmıştır:
 1. Malloc() her zaman **8'in katları olan işaretçiler döndürür**. Malloc()'un ayırdığı belleği kullanacak veri türünü bilmediğini unutmayın. Bu nedenle, sadece güvende olmak için her zaman 8'in katlarını döndürür.
 2. Derleyici, değişkenleri bellekte sıralı olacak şekilde yapıları(struct) düzenler ve yapının kendisinin temel işaretçisi sekizin katıysa bunlar hizalanır. Bu, derleyicinin bir yapıya biraz dolgu koyabileceği ve onu olması gerektiğini düşündüğünüzden daha büyük yapabileceği anlamına gelir.

Yapıların (Struct) yerleşimi

```
typedef struct {  
    char b;  
    int i;  
} Char_Int;
```

```
typedef struct {  
    char b1;  
    char b2;  
    char b3;  
    char b4;  
    int i1;  
} CCCC_Int;
```

```
typedef struct {  
    char b1;  
    int i1;  
    char b2;  
    int i2;  
} C_I_C_I;
```

```
typedef struct {  
    int i;  
    char b;  
} Int_Char;
```

```
Char_Int *ci;
```

```
ci = (Char_Int *) malloc(sizeof(Char_Int)*2);  
printf("The size of a Char_Int is %ld\n", sizeof(Char_Int));  
printf("I have allocated an array, ci, of two Char_Int's at location 0x%lx\n", (UL) ci);  
printf("&(ci[0].b) = 0x%lx\n", (UL) &(ci[0].b));  
printf("&(ci[0].i) = 0x%lx\n", (UL) &(ci[0].i));  
printf("&(ci[1].b) = 0x%lx\n", (UL) &(ci[1].b));  
printf("&(ci[1].i) = 0x%lx\n", (UL) &(ci[1].i));  
printf("\n");
```

Yapıların (Struct) yerleşimi

- ❑ Yukarıdaki örnekte yer alan Yapı(struct) yalnızca beş bayt kullansa da (b için bir ve i için dört bayt), yapının boyutu 8 bayttır.
- ❑ İşaretçilere baktığımızda b'yi bulduğunuz yerde struct'ın ilk byte'ı olduğunu ve b'den sonra i'nin 4 byte olduğunu görüyoruz.
- ❑ b ve i arasındaki üç bayt kullanılmaz. Neden böyle?
- ❑ Bunun nedeni, i işaretçisinin dördün katı olması ve b'nin adresinin i'nin adresinden önce gelmesi gerektiğidir (bu bir derleyici standardıdır).
- ❑ Bunu yapmanın tek yolu, b'den sonraki üç baytın kullanılmamasıdır.

Yapıların (Struct) yerleşimi

- ❑ CCCC_Int yapısının çıktısına bakalım.
- ❑ Gördüğünüz gibi, bu yapı, belleği en iyi şekilde kullanır -- değişkenler 8 bayt yer kaplar ve veri yapısının boyutu 8'dir. i işaretçisi hizalanmıştır - Mükemmel!

```
abdullah@abdullah-VirtualBox:~/sist_prog/cs360-lecture-notes/CStuff-2/bin$ ./pd
The size of a Char_Int is 8
I have allocated an array, ci, of two Char_Int's at location 0x55f989718260
&(ci[0].b) = 0x55f989718260
&(ci[0].i) = 0x55f989718264
&(ci[1].b) = 0x55f989718268
&(ci[1].i) = 0x55f98971826c

The size of a CCCC_Int is 8
I have allocated an array, cccci, of two CCCC_Int's at location 0x55f989718690
&(cccci[0].b1) = 0x55f989718690
&(cccci[0].b2) = 0x55f989718691
&(cccci[0].b3) = 0x55f989718692
&(cccci[0].b4) = 0x55f989718693
&(cccci[0].i1) = 0x55f989718694
&(cccci[1].b1) = 0x55f989718698
&(cccci[1].b2) = 0x55f989718699
&(cccci[1].b3) = 0x55f98971869a
&(cccci[1].b4) = 0x55f98971869b
&(cccci[1].i1) = 0x55f98971869c
```

Yapıların (Struct) yerleşimi

- ❑ C_I_C_I yapısı daha az optimal olarak düzenlenmiştir; i1 ve i2 işaretçilerinin yerleşiminden emin olmak için, b1'den sonraki üç baytı ve b2'den sonraki üç baytı boşa harcamalıyız.
- ❑ b1'den hemen sonra b2'yi tanımlasaydık, veri yapısının boyutu 16 yerine 12 olurdu.

```
The size of a C_I_C_I is 16
I have allocated an array, cici, of two C_I_C_I's at location 0x55f9897186b0
&(cici[0].b1) = 0x55f9897186b0
&(cici[0].i1) = 0x55f9897186b4
&(cici[0].b2) = 0x55f9897186b8
&(cici[0].i2) = 0x55f9897186bc
&(cici[1].b1) = 0x55f9897186c0
&(cici[1].i1) = 0x55f9897186c4
&(cici[1].b2) = 0x55f9897186c8
&(cici[1].i2) = 0x55f9897186cc
```

```
The size of a Int_Char is 8
I have allocated an array, ic, of two Int_Char's at location 0x55f9897186e0
&(ic[0].i) = 0x55f9897186e0
&(ic[0].b) = 0x55f9897186e4
&(ic[1].i) = 0x55f9897186e8
&(ic[1].b) = 0x55f9897186ec
```

Yapıların (Struct) yerleşimi

- ❑ Şimdi, `Int_Char` için son çıktı satırlarına bakın. Sizi şaşırtabilir:
- ❑ Çoğumuz `sizeof(Int_Char)` öğesinin beş olması gerektiğini düşünür çünkü `i` işaretçisi yapının başlangıcıdır ve hizalanması gerekir ve `b` işaretçisinin yerleşim konusunda endişelenmesi gerekmez.
- ❑ Bu yapılardan yalnızca birini tahsis edersek bu doğru olur. Ancak, eğer iki yapıdan oluşan bir dizi tahsis edersek, o zaman ikincisinin de hizalanması gerekir -- eğer yapının boyutu 5 olsaydı, o zaman `ic[1].i` hizalanmazdı. Bunu düzeltmek için yapının boyutu 8'dir ve `b`'den sonraki üç bayt boşa gider.

```
The size of a Int_Char is 8
I have allocated an array, ic, of two Int_Char's at location 0x55f9897186e0
&(ic[0].i) = 0x55f9897186e0
&(ic[0].b) = 0x55f9897186e4
&(ic[1].i) = 0x55f9897186e8
&(ic[1].b) = 0x55f9897186ec
```

Yaygın Bir Hata Türü

```
/* This program shows how you lose information as you
   convert data from larger types to smaller types. */

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

int main()
{
    char c;
    int i;
    int j;

    i = 10000;
    c = i;          /* We are losing information here, because 10000 cannot be stored in a byte. */
    j = c;

    printf("I: %d,   J: %d,       C: %d\n", i, j, c);
    printf("I: 0x%04x, J: 0x%04x,   C: 0x%04x\n", i, j, c);
    return 0;
}
```


Yaygın Bir Hata Türü

- ❑ c bir karakter olduğu için 10000 değerini tutamaz.
- ❑ Bunun yerine i'nin en düşük baytını, yani 16'yı (0x10) tutar.
- ❑ Sonra j'yi c'ye ayarladığınızda, j'nin 16 olduğunu göreceksiniz.

Pointer Aritmetiği

- ❑ Değişken bildiriminde [size] koyarak bir diziye statik olarak deklare edebiliriz.
- ❑ Örneğin, aşağıdaki değişken bildirimi, on tam sayıdan oluşan bir **iarray** dizisi oluşturacaktır:

```
int iarray[10];
```

- ❑ iarray öğelerine köşeli parantez içinde erişebilirsiniz. Özellikle, iarray'in boyutu hiçbir yerde saklanmaz -- onu kendiniz takip etmeniz gerekir.
- ❑ Gerçekte iarray, dizinin ilk elemanına bir işaretçidir. Başka bir deyişle, dizi için ayrılan 40 bayt vardır (çünkü tamsayıların her biri dört bayttır) ve iarray bunlardan ilkinin işaret eder.
- ❑ İstersek iarray'e ikinci bir işaretçi ayarlayabiliriz ve işaretçiyi artırıp başvurusunu kaldırarak iarray'in öğelerini yazdırabiliriz.

Pointer Aritmetiği

□ For döngüsünde 5 nicelik yazdıracağız. İnceleyelim;

```
/* This program sets a pointer to an array, and then dereferences each of the
   elements of the array using the pointer and pointer arithmetic. It prints
   the pointers in hexadecimal while it does so. */

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

typedef unsigned long (UL);

int main()
{
    int iarray[10];
    int *ip;
    int i;

    /* Set the 10 elements of iarray to be 100 to 109, and print the array's address. */

    for (i = 0; i < 10; i++) iarray[i] = 100+i;
    printf("iarray = 0x%lx\n", (UL) iarray);
```

Pointer Aritmetiği

```
/* Set ip equal to array, and then print the 10 elements using both iarray and ip.
   The following quantities will be printed for each element:

   - The index i (goes from 0 to 9)
   - The value of iarray[i] (goes from 100 to 109)
   - The pointer ip (will start at iarray and increment by four each time).
   - What *ip points to (this will be 100 to 109 again)
   - Pointer arithmetic: (ip-array) -- this will be the value of i.
*/

ip = iarray;

for (i = 0; i < 10; i++) {
    printf("i=%d.   ", i);
    printf("iarray[i]=%d.   ", iarray[i]);
    printf("ip = 0x%lx.   ", (UL) ip);
    printf("*ip=%d.   ", *ip);
    printf("(ip-iarray)=%ld.\n", (UL) (ip-iarray));
    ip++;
}

return 0;
}
```

Pointer Aritmetiği

UNIX> **bin/ptr**

iarray = 0x7fff5fbfdc40

i=0. iarray[i]=100. ip = 0x7fff5fbfdc40. *ip=100. (ip-iarray)=0

i=1. iarray[i]=101. ip = 0x7fff5fbfdc44. *ip=101. (ip-iarray)=1

i=2. iarray[i]=102. ip = 0x7fff5fbfdc48. *ip=102. (ip-iarray)=2

i=3. iarray[i]=103. ip = 0x7fff5fbfdc4c. *ip=103. (ip-iarray)=3

i=4. iarray[i]=104. ip = 0x7fff5fbfdc50. *ip=104. (ip-iarray)=4

i=5. iarray[i]=105. ip = 0x7fff5fbfdc54. *ip=105. (ip-iarray)=5

i=6. iarray[i]=106. ip = 0x7fff5fbfdc58. *ip=106. (ip-iarray)=6

i=7. iarray[i]=107. ip = 0x7fff5fbfdc5c. *ip=107. (ip-iarray)=7

i=8. iarray[i]=108. ip = 0x7fff5fbfdc60. *ip=108. (ip-iarray)=8

i=9. iarray[i]=109. ip = 0x7fff5fbfdc64. *ip=109. (ip-iarray)=9

UNIX>

Pointer Aritmetiği

- ❑ Hex ile ifade edilen adresler makineden makineye değişir.
- ❑ Ancak aralarındaki ilişki her zaman aynı olacaktır.
- ❑ Bu program çalışmaya başladığında, işletim sistemi onu 0x7fff5fbfdc40 ile başlayan 40 baytın iarray'in saklandığı yer olmasını sağlayacak şekilde kurmuştur. Bu nedenle iarray, 0x7fff5fbfdc40'a eşittir.
- ❑ iarray[0], 0x7fff5fbfdc40'ta başlayan dört baytsa, o zaman iarray[1], 0x7fff5fbfdc44'te başlayan dört bayt olmalıdır.
- ❑ Bu nedenle ip, for döngüsünün ikinci yinelemesinde 0x7fff5fbfdc44'e eşittir.
- ❑ ip'e bir eklemek aslında işaretçinin değerine dört ekler. Buna "**işaretçi aritmetiği**" denir - bir işaretçiye x eklediğinizde, gerçekten ona **sx** ekler, burada s, işaretçinin işaret ettiği verinin boyutudur.

Pointer Aritmetiği

- ❑ For döngüsünde yazdırılan son sütun da biraz kafa karıştırıcı.
- ❑ `ip`, `0x7fff5fbfdc44`'e eşittir, bu nedenle `(ip-iarray)`'nin dört eşit olacağını düşünürsünüz. Değil, çünkü derleyici işaretçi aritmetiği yapıyor -- derleyicinin bakış açısından, "`ip-iarray`" dediğinizde, `ip` ve `iarray` arasındaki öğelerin sayısını soruyorsunuz.
- ❑ Bu, öğenin boyutuna bölünen işaretçiler arasındaki fark olacaktır. Bu durumda, $(0x7fff5fbfdc44 - 0x7fff5fbfdc40) / 4$, bire eşittir.

Pointer Aritmetiği

```
/* This program sets a pointer to an array, and then dereferences each of the
   elements of the array using the pointer and pointer arithmetic. It prints
   the pointers in hexadecimal while it does so. */

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

typedef unsigned long (UL);

int main()
{
    int iarray[10];
    int *ip;
    int i;

    /* Set the 10 elements of iarray to be 100 to 109, and print the array's address. */

    for (i = 0; i < 10; i++) iarray[i] = 100+i;
    printf("iarray = 0x%lx\n", (UL) iarray);
```

Pointer Aritmetiği

```
/* Set ip equal to array, and then print the 10 elements using both iarray and ip.  
   The following quantities will be printed for each element:
```

- The index i (goes from 0 to 9)
- The value of iarray[i] (goes from 100 to 109)
- The pointer ip (will start at iarray and increment by four each time).
- What *ip points to (this will be 100 to 109 again)
- Pointer arithmetic: (ip-array) -- this will be the value of i.

```
*/
```

```
ip = iarray;
```

```
for (i = 0; i < 10; i++) {  
    printf("i=%d.  ",      i              );  
    printf("iarray[i]=%d.  ", iarray[i]    );  
    printf("ip = 0x%lx.  ", (UL) ip        );  
    printf("*ip=%d.  ",    *ip            );  
    printf("(ip-iarray)=%ld.\n", (UL) (ip-iarray));  
    ip++;  
}
```

```
return 0;
```

```
}
```

Pointer Aritmetiği

UNIX> **bin/sptr**

iarray = 0x7ffeed0e10f0

i=0. iarray[i]={100.00,200.00}. ip = 0x7ffeed0e10f0. *ip={100.00,200.00}. (ip-iarray)=0.
i=1. iarray[i]={101.00,201.00}. ip = 0x7ffeed0e1100. *ip={101.00,201.00}. (ip-iarray)=1.
i=2. iarray[i]={102.00,202.00}. ip = 0x7ffeed0e1110. *ip={102.00,202.00}. (ip-iarray)=2.
i=3. iarray[i]={103.00,203.00}. ip = 0x7ffeed0e1120. *ip={103.00,203.00}. (ip-iarray)=3.
i=4. iarray[i]={104.00,204.00}. ip = 0x7ffeed0e1130. *ip={104.00,204.00}. (ip-iarray)=4.
i=5. iarray[i]={105.00,205.00}. ip = 0x7ffeed0e1140. *ip={105.00,205.00}. (ip-iarray)=5.
i=6. iarray[i]={106.00,206.00}. ip = 0x7ffeed0e1150. *ip={106.00,206.00}. (ip-iarray)=6.
i=7. iarray[i]={107.00,207.00}. ip = 0x7ffeed0e1160. *ip={107.00,207.00}. (ip-iarray)=7.
i=8. iarray[i]={108.00,208.00}. ip = 0x7ffeed0e1170. *ip={108.00,208.00}. (ip-iarray)=8.
i=9. iarray[i]={109.00,209.00}. ip = 0x7ffeed0e1180. *ip={109.00,209.00}. (ip-iarray)=9.

- ❑ Gördüğünüz gibi, işaretçiler iptr'nin çalışmasından farklıdır. Ancak aralarındaki ilişki aynıdır ve ip'yi her artırdığınızda değeri 16 artar çünkü yapının(struct) boyutu 16 bayttır.

String-strcpy()

```
char *strcpy(char *s1, const char *s2);
```

- ❑ Strcpy(), s2'nin boş/null ile sonlandırılmış bir dize olduğunu ve s1'in, sondaki boş karakter de dahil olmak üzere s2'yi tutmak için yeterli karaktere sahip bir (char *) olduğunu varsayar.
- ❑ Strcpy() daha sonra s2'yi s1'e kopyalar.
- ❑ Ayrıca s1 döndürür.

String Örnek

```
/* Initialize three strings using strcpy() and print them. */

#include <stdio.h>
#include <string.h>

int main()
{
    char give[5];
    char him[5];
    char six[5];

    strcpy(give, "Give");
    strcpy(him, "Him");
    strcpy(six, "Six!");

    printf("%s %s %s\n", give, him, six);
    return 0;
}
```


String Örnek

```
/* What happens when you call strcpy and didn't allocate enough memory? */

#include <stdio.h>
#include <string.h>

typedef unsigned long UL;

int main()
{
    char give[5];
    char him[5];
    char six[5];

    /* Print the addresses of the three arrays. */

    printf("give: 0x%lx  him: 0x%lx  six: 0x%lx\n", (UL) give, (UL) him, (UL) six);

    /* This is the same as before -- nice strcpy() statements, and then print. */

    strcpy(give, "Give");
    strcpy(him, "Him");
    strcpy(six, "Six!");
    printf("%s %s %s\n", give, him, six);

    /* Now, this strcpy() is copying a string that is too big. */

    strcpy(him, "T.J. Houshmandzadeh");
    printf("%s %s %s\n", give, him, six);

    return 0;
}
```

String Örnek

- ❑ Açıkça yukarıdaki örnekle ilgili bir sorun var -- "T.J. Houshmandzadeh" dizisi beş karakterden çok daha büyük.
- ❑ bazı derleyiciler bunu derleyecek, ancak bazıları diğerleri bununla ilgili sıkıntı çıkartabilir.

```
UNIX> gcc -o bin/strcpy2 src/strcpy2.c
```

```
src/strcpy2.c: In function 'main':
```

```
src/strcpy2.c:21: warning: call to __builtin___strcpy_chk will always overflow destination buffer
```

```
UNIX>
```

- ❑ Bu akıllı bir derleyici. Ancak, derleyiciler her şeyi gören ve her şeyi bilen değildir. strcpy() etrafına kendi ifadelerimizi yazarak onu kandırabiliriz -- şimdi sorunu çözemez.

String

```
/* This is the same as strcpy2.c, but I write a procedure to call strcpy(), so that
   even a smart compiler won't figure out that I have a problem. */

#include <stdio.h>
#include <string.h>

typedef unsigned long UL;

void my_strcpy(char *s1, char *s2)
{
    strcpy(s1, s2);
}

int main()
{
    char give[5];
    char him[5];
    char six[5];

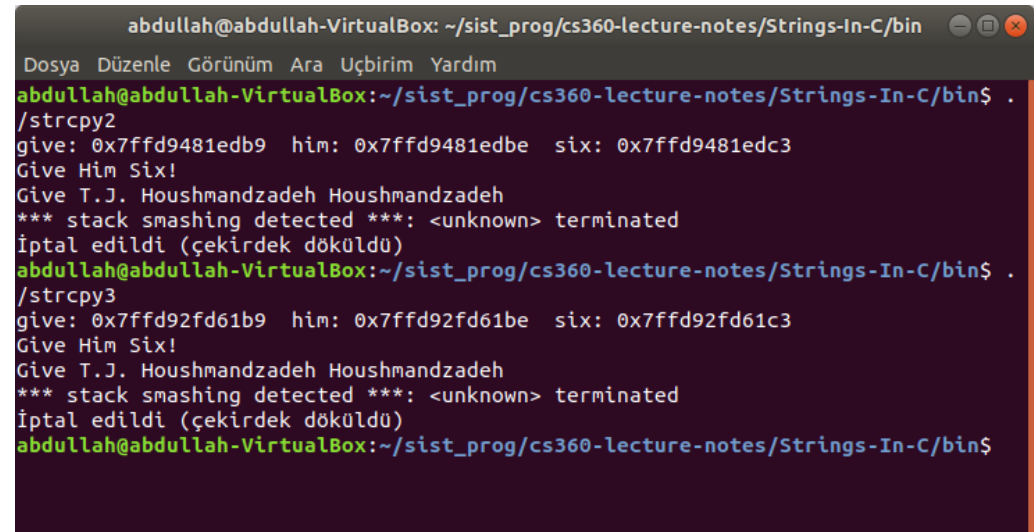
    printf("give: 0x%lx him: 0x%lx six: 0x%lx\n", (UL) give, (UL) him, (UL) six);

    strcpy(give, "Give");
    strcpy(him, "Him");
    strcpy(six, "Six!");

    printf("%s %s %s\n", give, him, six);

    my_strcpy(him, "T.J. Houshmandzadeh");

    printf("%s %s %s\n", give, him, six);
    return 0;
}
```



```
abdullah@abdullah-VirtualBox: ~/sist_prog/cs360-lecture-notes/Strings-In-C/bin
Dosya Düzenle Görünüm Ara Uçbirim Yardım
abdullah@abdullah-VirtualBox:~/sist_prog/cs360-lecture-notes/Strings-In-C/bin$ ./strcpy2
give: 0x7ffd9481edb9 him: 0x7ffd9481edbe six: 0x7ffd9481edc3
Give Him Six!
Give T.J. Houshmandzadeh Houshmandzadeh
*** stack smashing detected ***: <unknown> terminated
İptal edildi (çekirdek döküldü)
abdullah@abdullah-VirtualBox:~/sist_prog/cs360-lecture-notes/Strings-In-C/bin$ ./strcpy3
give: 0x7ffd92fd61b9 him: 0x7ffd92fd61be six: 0x7ffd92fd61c3
Give Him Six!
Give T.J. Houshmandzadeh Houshmandzadeh
*** stack smashing detected ***: <unknown> terminated
İptal edildi (çekirdek döküldü)
abdullah@abdullah-VirtualBox:~/sist_prog/cs360-lecture-notes/Strings-In-C/bin$
```

String-strcat()

`char *strcat(char *s1, const char *s2);`

- ❑ Strcat(), s1 ve s2'nin her ikisinin de boş/null ile sonlandırılmış dizeler olduğunu varsayar.
- ❑ Strcat() daha sonra s2'yi s1'in sonuna birleştirir.
- ❑ Strcat(), s1'de bu ekstra karakterleri tutmak için yeterli alan olduğunu varsayar. Aksi takdirde, ayırmadığınız hafızayı ezmeye başlarsınız.

```
UNIX> bin/strcat
Give
Give Him
Give Him Six!
UNIX>
```

```
/* Using strcpy() and strcat() to create the string "Give Him Six!" incrementally. */

#include <stdio.h>
#include <string.h>

int main()
{
    char givehimsix[15];

    strcpy(givehimsix, "Give");
    printf("%s\n", givehimsix);
    strcat(givehimsix, " Him");
    printf("%s\n", givehimsix);
    strcat(givehimsix, " Six!");
    printf("%s\n", givehimsix);
    return 0;
}
```