

# Sistem Programlama

---

DR. ÖĞR. ÜYESİ ABDULLAH SEVİN

# İçerik

---

a) makefiles

b) Pointers

c) Structs

d) Tip dönüşümü

- <http://web.eecs.utk.edu/~jplank/plank/classes/cs360/360/notes/CStuff-1/lecture.html>
- <http://web.eecs.utk.edu/~jplank/plank/classes/cs360/360/notes/CStuff-2/lecture.html>
- Cstuff-1 klasör örnekleri
- Cstuff-2 klasör örnekleri

# Structure (Yapı)

---

- ❑ Programlama dillerinde, "structure" terimi, farklı veri türlerinin bir araya getirildiği bir bileşik veri türüdür.
- ❑ Bu, programcılara, bir veya daha fazla veri türünden oluşan bir bileşenleri olan yeni bir veri türü oluşturma imkanı verir.
- ❑ Bellekte **stack** bölgesinde saklanır.
- ❑ Örneğin, bir öğrencinin adı, numarası ve notları gibi birçok veriye ihtiyacımız varsa, structure yapısı bu verileri bir arada tutmamızı sağlar.

# Structure (Yapı)

---

- ❑ Verileri toplamamanın diğer bir yolu bir yapı kullanmaktır.
- ❑ Bir struct (yapı), bazı dikkate değer eksiklikler dışında C++'daki sınıfına benzer:
  - ❑ «public/protected/private» yok.
  - ❑ «constructors/destructors» yok.
  - ❑ Varsayılan "kopyalama" yöntemi yok.
  - ❑ Metotlar yok.

# Structure (Yapı)

```
/* A very simple program to show a struct
   that aggregates an integer and a double. */

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

struct intdouble {
    int i;
    double d;
};

int main()
{
    struct intdouble id1;

    id1.i = 5;
    id1.d = 3.14;

    printf("%d %.2lf\n", id1.i, id1.d);
    return 0;
}
```

```
/* This program is identical to src/id1.c,
   except it uses a typedef so that you can
   assign a type to the struct. */

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

typedef struct intdouble {
    int i;
    double d;
} ID;

int main()
{
    ID id1;

    id1.i = 5;
    id1.d = 3.14;

    printf("%d %.2lf\n", id1.i, id1.d);
    return 0;
}
```

**typedef**

# Structure (Yapı)

```
/* This is a C++ program, which shows how you can copy one str

#include <cstdio>
#include <iostream>
using namespace std;

struct intdouble {
    int i;
    double d;
};

int main()
{
    intdouble id1, id2;

    id1.i = 5;           /* Set id1 to 5 and 3.14 as before. */
    id1.d = 3.14;

    id2 = id1;           /* This makes a copy of id and then add

    id2.i += 5;
    id2.d += 5;

    printf("1: %d %.21f\n", id1.i, id1.d);    /* Print them out.
    printf("2: %d %.21f\n", id2.i, id2.d);

    return 0;
}
```

❑ Yapılar **C++'da farklı** semantikte çalışır bu yüzden karışıklığa neden olabilir.

❑ C ve C++'da bir yapı diğerine kopyalanabilir

```
UNIX> bin/id3 1: 5 3.14
2: 10 8.14
UNIX>
```



# Structure (Yapı)

```
/* Copying src/id3.cpp to src/id5.c, and fixing the use of intdouble
   so that it compiles. It works as in C++, copying the struct, but you
   should be wary of it. */

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

struct intdouble {
    int i;
    double d;
};

int main()
{
    struct intdouble id1, id2;

    id1.i = 5;
    id1.d = 3.14;

    id2 = id1; /* THIS IS THE OFFENDING LINE */
    id2.i += 5;
    id2.d += 5;

    printf("1: %d %.2lf\n", id1.i, id1.d);
    printf("2: %d %.2lf\n", id2.i, id2.d);
    return 0;
}
```

❑ ([src/id4.c](#)) hatalı

❑ "intdouble"ın önüne "struct" yazmalıyız, C dilinde. C++'da struct yazmaya gerek yok.

# Structure (Yapı)

```
/* While C doesn't let you copy arrays, it lets you copy a struct that holds
   an array. I don't think this really makes sense, but there is it. In this
   code, we copy 4000 bytes in a single statement. */

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

typedef struct {
    int a[1000];
} SID;

int main()
{
    SID s1, s2;
    int i;

    for (i = 0; i < 1000; i++) s1.a[i] = i;      /* Set s1. */

    s2 = s1;      /* This statement copies 4000 bytes. */

    for (i = 0; i < 1000; i++) printf("%4d %4d\n", s1.a[i], s2.a[i]); /* Print s1 and s2. */

    return 0;
}
```

- ❑ Yapılardaki atama deyimi ile belirtilmemiş sayıda baytı kopyalayabileceğiniz C'nin tek parçasıdır.
- ❑ Bu, dilin bir zayıflığı olarak ifade edilebilir.
- ❑ ``s2 = s1" satırı 4000 bayt kopyalar.
- ❑ Normalde dizileri bu şekilde kopyalamamıza izin vermiyor



# Structure (Yapı)

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

typedef struct {
    int a[1000];
} SID;

void a(SID s)    /* Although this procedure changes element 999 of s, */
{               /* s is a copy of the calling parameter, so it is */
    s.a[999] = -1; /* deleted at the end of the procedure. */
}               /* In other words, the procedure does nothing. */

int main()
{
    SID s1;
    int i;

    for (i = 0; i < 1000; i++) s1.a[i] = i;    /* Set the elements of s1. */

    a(s1);    /* This does nothing, because it modifies a copy of s1 */

    printf("Element 999: %d\n", s1.a[999]);

    return 0;
}
```

- ❑ Bir fonksiyona parametre olarak gönderebiliriz,
- ❑ Fakat burada kopyası oluşturulur ve yapıdaki tanımlanan dizide yapılan değişiklik prosedürden çıktıktan sonra kaybolur **yani bir değişiklik olmaz.**
- ❑ Dizileri gönderdiğimizde dizi değişiyordu!

# Structure (Yapı)

---

- ❑ C++ yapıları hakkında son bir not.
- ❑ Temel olarak [sadeleştirilmiş](#) sınıflardır - içlerine fonksiyonlar koyabilir ve ardından yapı değişkenlerini kullanarak fonksiyonları uygulayabilirsiniz.

```
/* Unlike C structs, you can put methods in C++ structs. */
#include <cstdio>
#include <iostream>
using namespace std;

struct intdouble {
    int i;
    double d;
    void Print();
};

void intdouble::Print()
{
    printf("    %d %.21f\n", i, d);
}

int main()
{
    intdouble id1, id2;

    id1.i = 5;
    id1.d = 3.14;

    id2 = id1;
    id2.i += 5;
    id2.d += 5;

    id1.Print();
    id2.Print();
    return 0;
}
```

# İşaretçi (Pointer)

---

- ❑ İşaretçiler, çoğu insanın C'de hata yaptığı yerlerdir.
- ❑ Bir işaretçi, basitçe hafızanın bir **indeksidir (göstericidir)**.
- ❑ Bellek, iki yoldan biriyle tahsis edilebilir -- **değişkenleri** bildirerek veya **malloc()**'u çağırarak.
- ❑ Bellek tahsis edildiğinde, ona bir işaretçi ayarlayabilirsiniz.
- ❑ Ne zaman **x** bayt bellek ayırsak, bellek dizisinden **x bitişik öğe ayırmış** oluyoruz. (Unutmayalım sanal bellekte, gerçekte nerede bilemiyoruz-MMU biliyor)
- ❑ Bu baytlara bir işaretçi ayarlarsak, o işaretçi bellekte ayrılan **ilk baytın** indisi olacaktır.

# Pointer

```
/* Print out pointers of local variables */

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

int main()
{
    int i;
    char j[14];
    int *ip;
    char *jp;

    ip = &i;
    jp = j;

    printf("ip = 0x%x.  jp = 0x%x\n", ip, jp);
    return 0;
}
```

- ❑ Bu program bir tamsayı (**i**), 14 karakterlik bir dizi (**j**) ve iki işaretçi (**ip ve jp**) ayırır.
- ❑ Daha sonra işaretçileri, **i ve j** için ayrılan belleğe işaret edecek şekilde ayarlanır.
- ❑ Son olarak, bu işaretçilerin değerlerini yazdırır - bunlar bellek dizisinin indeksleridir.
- ❑ 'long unsigned int' uyarısı alıyoruz derlerken , buna geleceğiz.
- ❑ **i değişkeni için 0x7fff2efcdd9c, 0x7fff2efcdd9d 0x7fff2efcdd9e, ve 0x7fff2efcdd9f tahsis edilmiş (int -- 4 bytes)**
- ❑ **0x7fff2efcdda0 - 0x7fff2efcddad arası j dizisi (char -- 1 byte)**

# Pointer

---

□ **i** değişkeni ve **j** dizisinin bitiş adresini yazdırmaya çalışalım !

# Pointer

---

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i;
    char j[14];
    int *ip;
    char *jp;
    ip = &i;
    jp = j;
    // Bitiş adresleri
    printf("i başlangıç adresi = 0x%lx\n", (unsigned long)ip);
    printf("i bitiş adresi      = 0x%lx\n", (unsigned long)(ip + 1));
    printf("j başlangıç adresi = 0x%lx\n", (unsigned long)jp);
    printf("j bitiş adresi      = 0x%lx\n", (unsigned long)(jp + sizeof(j)));

    return 0;
}
```

❑ Evet, bu şekilde olabilir, pointer aritmetiği

# Bellek Düzeni (Varsayılan Stack Yerleşimi)

Örneğin, stack bellekte şu şekilde olabilir:

Değişken	Adres (Örnek)	Boyut
i	0x7ffce4ee8d84	4 veya 8 byte
(Boşluk)	0x7ffce4ee8d88	(Boşluk - Alignment için)
j[0]	0x7ffce4ee8d9a	1 byte
j[1]	0x7ffce4ee8d9b	1 byte
...	...	...

```
abdullah@abdullah-VirtualBox:~/sist_prog/hafta3$ gcc p0.c -o p0
abdullah@abdullah-VirtualBox:~/sist_prog/hafta3$ ./p0
i başlangıç adresi = 0x7ffe0a830f44
i bitiş adresi     = 0x7ffe0a830f48
j başlangıç adresi = 0x7ffe0a830f5a
j bitiş adresi     = 0x7ffe0a830f68
```



# Pointer

```
/* Print out pointers of local variables */

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

int main()
{
    int i;
    char j[14];
    int *ip;
    char *jp;

    ip = &i;
    jp = j;

    printf("ip = 0x%lx.  jp = 0x%lx\n", ip, jp);
    return 0;
}
```

- ❑ ``jp = j" ve ``ip = &i" dikkat !
- ❑ Bunun nedeni, **dizinin bir işaretçiye eşdeğer olmasıdır.**
- ❑ Tek fark, bir dizi değişkenine değer atayamazsınız.
- ❑ Böylece ``jp = j" diyebilirsiniz, ancak ``j = jp" diyemezsiniz.

# Pointer

---

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

void f1(int * ptr, int len)
{
    int i;
    ptr = (int*) malloc (sizeof(int)*len);
    for (i = 0; i < len; i ++)
        ptr[i] = i;
}

int main()
{
    int i, * array;
    f1(array, 5);
    for (i = 0; i < 5; i ++)
        printf("got value %d\n", array[i]);
    return 0;
}
```

# Pointer

---

- ❑ Yukarıdaki kodda, `f1` fonksiyonu bir işaretçi ve bir uzunluk (`len`) parametresi alır. İşaretçi önce bellekten bir dizi oluşturmak için **malloc** fonksiyonu kullanılarak yeniden ayarlanır.
- ❑ Bu dizi, 0'dan **len** değişkenine kadar sayıları içerir. Ancak, bu kodda önemli bir sorun vardır.
- ❑ Ama `ptr` sadece bir kopyadır ve `main` fonksiyonundaki `array` değişkenini etkilemez.
- ❑ `ptr = (int*) malloc (sizeof(int) * len);` ile bellekte yeni bir alan ayrılıyor.
- ❑ Ancak, `ptr` sadece lokal bir değişken olduğu için, fonksiyon bittiğinde bu bilgi kaybolur.
- ❑ `array` değişkeni belleğe işaret etmediği için `array[i]` kullanımı segmentation fault (hata) oluşturur.
- ❑ Eğer bir fonksiyona normal bir pointer (`int *ptr`) gönderilirse, sadece pointer'ın bir kopyası fonksiyona iletilir. Eğer çift pointer (`int **ptr`) gönderilirse, orijinal pointer'ın adresi gönderilir ve fonksiyon içinde değiştirilen adres `main` fonksiyonuna yansır.

# Pointer

---

```
#include <stdio.h>

#include <stdlib.h>

void f1(int ** ptr, int len)
{   int i;

    *ptr = (int*) malloc(sizeof(int)*len);

    if (*ptr == NULL) {

        printf("Bellek tahsisi basarisiz oldu.\n");

        return;   }

    for (i = 0; i < len; i++)

        (*ptr)[i] = i;

}
```

```
int main()

{

    int i, *array;

    f1(&array, 5);

    if (array == NULL) {

        printf("Bellek tahsisi basarisiz oldu.\n");    return -1;

    }

    for (i = 0; i < 5; i++)

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

    return 0;

}
```

# Pointer- Ek Örnek

---

```
#include <stdio.h>

void swap(int* x, int* y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}

int main() {
    int a = 5, b = 10;
    printf("a: %d, b: %d\n", a, b);
    swap(&a, &b);
    printf("a: %d, b: %d\n", a, b);
    return 0;
}
```

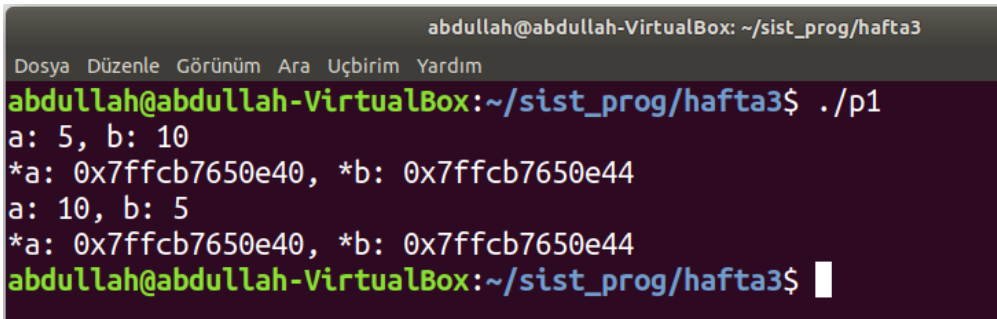
- ❑ Burada, **swap** fonksiyonu iki tamsayı işaretçisi alır.
- ❑ Bu işaretçiler, işaret ettikleri tamsayıların değerlerini değiştirmek için kullanılır.
- ❑ İşaretçilerin **kendileri değil, işaret ettikleri değerler** değiştirilir.
- ❑ Fonksiyonda **temp** adlı geçici bir değişken kullanılarak, değerler birbirine atanır ve değerler değiştirilir.

# Pointer-Ek örnek

---

```
#include <stdio.h>
typedef long unsigned int LUI;

void swap(int* x, int* y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}
```



```
abdullah@abdullah-VirtualBox: ~/sist_prog/hafta3
Dosya Düzenle Görünüm Ara Uçbirim Yardım
abdullah@abdullah-VirtualBox:~/sist_prog/hafta3$ ./p1
a: 5, b: 10
*a: 0x7ffcb7650e40, *b: 0x7ffcb7650e44
a: 10, b: 5
*a: 0x7ffcb7650e40, *b: 0x7ffcb7650e44
abdullah@abdullah-VirtualBox:~/sist_prog/hafta3$
```

```
int main() {

    int a = 5, b = 10;
    int *ap, *bp;

    printf("a: %d, b: %d\n", a, b);
    ap=&a;      bp=&b;

    printf("*a: 0x%lx, *b: 0x%lx\n", (LUI) ap, (LUI) bp);

    swap(&a, &b);

    printf("a: %d, b: %d\n", *ap, *bp);

    printf("*a: 0x%lx, *b: 0x%lx\n", (LUI) ap, (LUI) bp);

    return 0;
}
```

!! Uyarılar düzeltildi

# Tip dönüşümü (Type Casting)

---

```
/* This program assigns a char to an int, and the int to a float.
   Although it looks benign, there are some things going on
   beneath the hood (changing number formats). */

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

int main()
{
    char c;
    int i;
    float f;

    c = 'a';
    i = c;
    f = i;

    printf("c = %d (%c).   i = %d (%c).   f = %f\n", c, c, i, i, f);
    return 0;
}
```

□ Bazen x baytta depolanan bir değişkeni alıp y baytta depolanan bir değişkene atamak istersiniz.

□ Buna ``tip dönüşümü" denir.

- **char** -- 1 byte
- **short** -- 2 bytes
- **int** -- 4 bytes
- **long** -- 4 or 8 bytes, depending on the system and compiler
- **float** -- 4 bytes
- **double** -- 8 bytes
- (pointer -- 4 or 8 bytes, depending on the system and compiler)



# Tip dönüşümü

---

- ❑ Yukarıdakiler gibi bazı tip dönüşümleri çok doğaldır.
- ❑ C derleyicisi bunları itiraz etmeden 😊 sizin yerinize yapacaktır.
- ❑ Diğerlerinin çoğu için, özellikle bir tip dönüşümü yaptığınızı söylemediğiniz sürece, C derleyicisi bir uyarı verecektir.
- ❑ Bu, derleyiciye "**Evet, ne yaptığımı biliyorum**" demenin bir yoludur.

# Tip dönüşümü

❑ Olan şu ki, derleyici printf() format dizgisini ayrıştırıyor ve "%lx"ın **long unsigned int** istediğini, ancak bir **(int \*)** alıyor ve bu yüzden uyarı veriyordu!.

<a href="#">src/p5.c</a>	<a href="#">src/p5a.c</a>
<pre>/* Adding typecast statements to make the    warnings from src/p3.c go away. */  #include &lt;stdio.h&gt; #include &lt;stdlib.h&gt;  int main() {     int i;     char j[14];     int *ip;     char *jp;      ip = &amp;i;     jp = j;      printf("ip = 0x%lx.  jp = 0x%lx\n",         /* Here they are. */         (long unsigned int) ip,         (long unsigned int) jp);     return 0; }</pre>	<pre>/* Using a typedef to make the typecast    statements a little less cumbersome. */  #include &lt;stdio.h&gt; #include &lt;stdlib.h&gt;  typedef long unsigned int LU;      /* Now, I can use "(LU)" rather                                    than "(long unsigned int)." */  int main() {     int i;     char j[14];     int *ip;     char *jp;      ip = &amp;i;     jp = j;      printf("ip = 0x%lx.  jp = 0x%lx\n", (LU) ip, (LU) jp);     return 0; }</pre>

"Evet, bu bir (int \*), ancak (long unsigned bir int) gibi davran, lütfen!. Ne yaptığımı biliyorum."

# Tip dönüşümü

---

- ❑ Bazı makinelerde (Pi gibi, 32 bit-4 byte), hem **işaretçiler-pointer** hem de **int** 4 bayttır.
- ❑ Bu, birçok insanın işaretçileri ve integer'ı birbirinin yerine geçebilir olarak görmesine yol açtı.

# Tip dönüşümü

```
/* A program where we inadvisedly typecast a pointer to an int and back again.
   On machines with 8-byte pointers, this is a buggy activity, because you lose
   data when you typecast from an integer to a pointer. */

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

typedef long unsigned int LUI;

int main()
{
    char s[4];
    int i;
    char *s2;

    /* Copy the string "Jim" to s, then turn the pointer into an integer i.
       Print out the pointer's value, and i's value. */

    strcpy(s, "Jim");
    i = (int) s;
    printf("Before incrementing i.\n");
    printf("i = %d (0x%x)\n", i, i);
    printf("s = %ld (0x%lx)\n", (LUI) s, (LUI) s);

    /* Now increment i, and turn it back into a pointer.
       Print out the pointers, and then attempt to print out what they point to. */

    i++;
    s2 = (char *) i;
    printf("\n");
    printf("After incrementing i.\n");
    printf("s = 0x%lx. s2 = 0x%lx, i = 0x%x\n", (LUI) s, (LUI) s2, i);
    printf("s[0] = %c, s[1] = %c, *s2 = %c\n", s[0], s[1], *s2);
    return 0;
}
```

❑ [src/p8.c](#):

❑ i'yi s'ye eşitlediğinizde, 4 bayt bilgi kaybedersiniz, çünkü **int** dört bayttır ve **işaretçiler** sekiz bayttır.

❑ s2'yi tekrar i'ye ayarladığınızda, i'nin eksik olduğu dört baytı genellikle sıfırlarla, bazen -1'lerle doldurur.

❑ Her iki durumda da, geçersiz bir adres olacak ve bir segmentasyon ihlali alacaksınız:

❑ 32-bit makinede bir sıkıntı olmaz

# Tip dönüşümü

```
/* This is the same as src/p8.c, but we've changed i to a long. */  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
typedef long unsigned int LUI;  
  
int main()  
{  
    char s[4];  
    long i;  
    char *s2;  
  
    strcpy(s, "Jim");  
    i = (long) s;  
    printf("Before incrementing i.\n");  
    printf("i = %ld (0x%lx)\n", i, i);  
    printf("s = %ld (0x%lx)\n", (LUI) s, (LUI) s);  
  
    i++;  
    s2 = (char *) i;  
    printf("\n");  
    printf("After incrementing i.\n");  
    printf("s = 0x%lx. s2 = 0x%lx, i = 0x%lx\n", (LUI) s, (LUI) s2, i);  
    printf("s[0] = %c, s[1] = %c, *s2 = %c\n", s[0], s[1], *s2);  
    return 0;  
}
```

- ❑ Bunun yerine bir int yerine i için long kullanırsak, her şey yolunda gider,
- ❑ çünkü long ve işaretçilerin aynı boyutta olması garanti edilir, ister 4 ister 8 bayt olsun.

# Malloc ve Free

---

- ❑ C'de new veya delete yoktur.
- ❑ İşlevsellikleri **malloc()** ve **free()** kütüphane çağrıları tarafından sağlanır.

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

```
void *malloc(size_t size);  
void free(void *ptr);
```

- ❑ **New gibi** malloc() işletim sisteminden **istenilen bayt miktarını bellekten ayırır.**
- ❑ Ayırdığı veri türü hakkında bilgi vermenizi gerektiren new'in aksine, **malloc()** **yalnızca bayt sayısını** sorar ve başarılı olursa, en az o kadar bayt için işletim sistemi tarafından **bir işaretçi döndürür.**
- ❑ Bir işaretçi olduğu anlamına gelen bir void \* döndürür, ancak **malloc()** **neye işaret ettiğini bilmez.**

# Malloc ve Free

---

- ❑ malloc fonksiyonu, **programcılara çalışma zamanında (runtime) bellek ayırma olanağı** sağlar.
- ❑ Programlar çalışırken, çeşitli nedenlerden dolayı programın **ihtiyaç duyduğu bellek boyutu değişebilir**.
- ❑ Bu durumda, malloc gibi fonksiyonlar, ihtiyaç duyulan bellek boyutunu **çalışma zamanında ayrılabilir ve kullanılabilir** hale getirir.
- ❑ Ayrıca, büyük bellek blokları gibi sabit boyutlu verileri depolamak için dağıtılmamış bellek alanları oluşturmak da mümkündür.
- ❑ Ancak, bu işlem, bazı durumlarda **bellek yönetimi için gereksiz karmaşıklığa neden olabilir**. Bu nedenle, malloc fonksiyonu bu tür durumlarda daha uygun bir çözüm sunar.



# Malloc ve Free

---

- ❑ `malloc()`'tan kaç bayta ihtiyacınız olduğunu bulmak için **`sizeof(type)`** öğesini çağırırsınız.
- ❑ Örneğin, bir tamsayı tahsis etmek için **`malloc(sizeof(int))`** işlevini çağırırsınız.
- ❑ Genellikle bir veri türü dizisi tahsis etmek isteriz. Bunu yapmak için, `sizeof(type)` öğesini öğe sayısı ile çarparsınız.
- ❑ **İşaretçiniz** bu öğelerin ilkini gösterecektir.
- ❑ Bir sonraki öğe, işaretçiden sonra `sizeof(type)` bayt olacaktır.

# Malloc ve Free

```
/* The point of this program is to show how one may pass a region of bytes
   (an array) from procedure to procedure using a pointer. */

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

/* This allocates n integers, error checks and returns a pointer to them. */

int *give_me_some_ints(int n)
{
    int *p;
    int i;

    p = (int *) malloc(sizeof(int) * n);
    if (p == NULL) { fprintf(stderr, "malloc(%d) failed.\n", n); exit(1); }
    return p;
}

/* This takes a pointer to n integers and assigns them to random numbers. */

void fill_in_the_ints(int *a, int n)
{
    int i;

    for (i = 0; i < n; i++) a[i] = lrand48();
}

/* This reads the command line, allocates, assigns and prints n integers. */
```

```
/* This reads the command line, allocates, assigns and prints n integers. */

int main(int argc, char **argv)
{
    int *array;
    int size;
    int i;

    if (argc != 2) { fprintf(stderr, "usage: pm size\n"); exit(1); }
    size = atoi(argv[1]);

    array = give_me_some_ints(size);
    fill_in_the_ints(array, size);

    for (i = 0; i < size; i++) printf("%4d %10d\n", i, array[i]);
    return 0;
}
```

# Malloc ve Free

---

- ❑ `give_me_some_ints()` prosedürü, `n` tam sayıdan oluşan bir dizi tahsis eder ve diziye bir işaretçi döndürür.
- ❑ `fill_in_the_ints()`, diziye bir işaretçi artı boyutunu alır ve onu doldurur.
- ❑ İşaretçileri geçtiğimiz için, dizinin hiçbir kopyası yapılmaz.
- ❑ Başka bir deyişle, `fill_in_the_ints()`, `malloc()` çağrısı tarafından oluşturulan diziye doldurur.
- ❑ Son olarak diziye yazdırıyoruz.

# Malloc ve Free

---

❑ C ve C++ arasındaki farklar;

1. malloc()'a kaç bayt istediğinizi söylemelisiniz.
2. malloc() belleğin nasıl kullanıldığını bilmez -- sadece baytları ayırır.
3. Dizin boyutunu takip etmelisiniz. Bu, bir vektöre kıyasla elverişsizdir.
4. C'de referans değişkenleri yoktur. Parametreler her zaman kopyalanır. Burada kopyalanan işaretçidir, işaret ettiği veri değil.
5. Bu nedenle, yukarıdaki programda dizinin yalnızca bir kopyası vardır.

# Malloc ve Free

---

❑ free(), yeniden kullanılabilmesi için belleği serbest bırakır. C++'da silmeye (delete) benzer.

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

int main() {
    // Bellek alanı ayırmak için malloc kullanımı
    int *ptr = (int*) malloc(5 * sizeof(int));

    // Bellek ayrılamazsa kontrol edin
    if (ptr == NULL) {
        printf("Bellek ayrılamadı.\n");
        exit(0);
    }
```

```
// Bellek alanını kullanmak için örnek veri yazın
for (int i = 0; i < 5; i++) {
    *(ptr + i) = i;
}
// Bellek alanında veri yazdırmak
for (int i = 0; i < 5; i++) {
    printf("%d ", *(ptr + i));
}
// Bellek alanını serbest bırakmak için free kullanımı
free(ptr);
ptr = NULL; // İyi bir uygulama için ptr'nin null'a atanması

return 0;
}
```

# Soru

---

1-) Char bir dizi a-z arası. Pointer ile fonksiyona gönderiyoruz . Fonksiyon bu diziyi ters çeviriyor. Ana fonksiyonda integer olarak yazdır.

2-) Terminalden girilen sayıdan (int) itibaren 100e kadar veriler oluştursun (malloc). Bu elemanları bir fonksiyona gönder (pointer). Fonksiyon elemanları Ters çevirsin. Ana fonksiyonda char olarak yazalım.