Ders 28

Const kullanımı

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

int main(void)
{
    int ar[] = { 2, 6, 8, 2, 9 };
    int const* p = ar;
    //const int * p = ar;

    int ival = *p;
    printf("%d\n", *p);

    *p = 67;
    p[2] = 45;

}
```

Son 2 ifade sentax hatası

```c
#define isleap(y)    ((y) % 4 == 0 && ((y) % 100 != 0 || (y) % 400 == 0))
#define asize(x)      (sizeof(x) / sizeof(*x)

int isprime(int);
void randomize(void);
void pdline(void);
void print_array(const int*, int);
void set_array_random(int*, int);
void sort_array(int*, int);
void sgets(char*);

void swap(int* x, int* y);
void bsort(int* p, int size);


#endif
```

Print_array const int var bunun nedeni dizinin elemanlarını değiştirmeden sadece ekrana yazdırmak olduğu için.

Örnek dönüşüm hatası

```c
#include <stdio.h>

void print_array(int* p, int size);


int main()
{
    const int primes[] = { 2, 3, 5, 7, 11, 13 };

    print_array(primes, 6);

}
```

Const int int* türüne dönüşürken hata olur uyarı verir vetehlikeli bir işlemdir. Değişmeyecekse fonksiyonda const int * yazmalıyız.

Örnek hata var mı

```c
int main()
{
    int x = 10;
    int* p = &x;

    const int* ptr = p;
}
```

Bu kodda hata yok

Tersi olsaydı syntax hatası değildi amakesinlikle yapmayın.

```c
void func(int*);

int main()
{
    const int x = 5;

    func(&x);
}
```

C++ hata c de uyarı verir yapmayın.

```
int main()
{
    int x = 10;
    int y = 435;
    int* p = &x;
    const int* cp = &y;

    p = cp;
}
```

Burada const int* dan int * a dönüşüm var uyarı verir necatiye göre sakın yapmayın.

```
void func(const char* p)
{
    char* ptr = (char *)p;

}
```

İsteyerek yaptığımızı göstermek için tür dönüştürme operatörü kullanmal ıyız.

Örnek konvensiyon

```
void get_minmax(const int* pa, int size, int* pmin, int* pmax);
void get_minmax(const int pa[], int size, int* pmin, int* pmax);
```

Ama ben üstekini yapıyorum.

Pointer idiom örnekleri

```
int main()
{
    int a[] = { 10, 20, 30, 40, 50 };
    int* p = a;

    *p++ = -1;

}
```

Geçerli ve l value 1. Elemana 1 değerini verdik

değerini gösterirdi

```
#include "nutility.h"

int main()
{
    int a[] = { 10, 20, 30, 40, 50 };
    int* p = a;

    print_array(a, 5);

    *p++ = -1;
    print_array(a, 5);

    *p = -2;

}
```

*p++ adresi 1 arttırdı. *p dizinin 2. Elemanını gösterir suan

Örnek kullanım

```
void set_array_random(int* p, int size)
{
    while (size--) {
        *p = rand() % 1000;
        ++p;
    }
}
```

```
{
    while (size--)
        *p++ = rand() % 1000;
}
```

Es anlamlı

Copy array fonksiyonu çok güzel idiom örneği

```
void copy_array(int* pdest, const int* psource, int n)
{
    while (n--) {
        *pdest = *psource;
        ++pdest;
        ++so
    }
}
```



```
void copy_array(int* pdest, const int* psource, int n)
{
    while (n--) {
        *pdest++ = *psource++;
    }
}
```

örnek

```
int main()
{
    int a[] = { 10, 20, 30, 40, 50 };
    int* p = a;

    print_array(a, 5);

    *++p = -1;

    //*(++p)
```

Hata yok dizinin 2 i elemanı -1 oldu..

```
int main()
{
    int a[] = { 10, 20, 30, 40, 50 };
    int* p = a;

    print_array(a, 5);

    *++p = -1;
    print_array(a, 5);

    *p = -2;
```

Buradada yine 2. Eleman -2 oldu.

Örnek

```
int main()
{
    int a[] = { 10, 20, 30, 40, 50 };
    int* p = a;

    print_array(a, 5);

    ++*p;
    //++(*p);

    print_array(a, 5);

}
```

Burada 1. Eleman 1 artacaktı. Elemanın değeri 11 oldu *p nesnesini 1 arrttırdı.

Şaşırtmalı soru hangi eleman -1 olur

```
#include "nutility.h"

int main()
{
    int a[] = { 10, 20, 30, 40, 50 };

    //*a++ = -1;

}
```

```
#include "nutility.h"

int main()
{
    int a[] = { 10, 20, 30, 40, 50 };

    (& a[0])++;


}
```

Aslında bu ifadeye dönüşüyor yukardaki ifadeler sentax hatası olur.

```
Dikkat!!!

C ve C++ dillerinde

bir dizi ismi

atama operatörünün sol operandı olamaz
++ operatörünün operandı olamaz
-- operatörünün operandı olamaz
```

```
int main()
{
    int a[] = { 10, 20, 30, 40, 50 };
    int* p = a;

    *p++ = -1;

    *a++ = -1;

}
```

İlki geçerli

```
int main()
{
    int a[] = { 10, 20, 30, 40, 50 };

    //++*a;

}
```

++*a; geçerli kullanılabilir. Dizinin ilk elemanını 1 arttırır.

Çok önemli soru

```
#define         SIZE        10

void reverse_copy(int* pdest, const int* psource, int n)
{
    psource += n;

    while (n--) {
        *pdest++ = *--psource;
    }

}
```

Dizinin bittiği yeri pointere atadım sonra –p yaparak 1 eksiği ile işleme girdim bu şekilde tasma olmadı.

Örnek

```
void func(int* p, int size)
{
    while (size--)
        ++*p++;
}

int main()
{
    int a[] = { 10, 20, 30, 40, 50 };

    func(p: a, size: 5);

}
```

Adres 1 artsın gösterdiği değişkende 1 artsın ama kendinden başlayarak artacak.

Örnek geçerli pointer

```
int main()
{
    int a[5] = { 10, 20, 30, 40, 50 };

    int* p = a + 5;

    p
```

Dizinin son elemanından bir fazla adres değeri atadık ancak dizinin bittiği yerde geçerli bir ponter değeri oluşturuyor.

Kodlamada dizinin ilk elemanın adresine dizinin boyutunun eklendiği adrestir bir çok işlemde kullanıyoruz ancak içerik operatörünün operandı yapmıyoruz.

--*p deseydim 1 ekseği yani diziyi sondan yazdırabiklirdim.

```
int main()
{
    int x = 21;

    int* p = &x;

    ++p;
```

Tekil nesneler bir elemanlı dizi gibi işlenir.

Burada pointer değişkeni geçerlidir.