Ders 29

Örnek 2 pointer dizinin bittiği yerin adresini tutuyor ise

```c
int main()
{
    int a[10] = { 1, 3, 5, 7, 9, 11, 13, 15, 17, 19 };
    int* p1 = a;
    int* p2 = a + 10;  //p2 dizinin bittiği yerin adresi

    while (p1 != p2) {
        printf("%d\n", *p1++);
    }

}
```

P2 dizinin bittiği yer p1 dizinin sonuna kadar tarayıp sonunda çıkıcak bu daha çok veri yapılarında tarama için kullanılıyor. Daha cok c++ da kullanılır.

Örnek print array fonksiyonu range yöntemi ile

```c
void print_array_r(const int* ps, const int* pe)
{
    while (ps != pe) {
        printf("%3d ", *ps++);
    }
    printf("\n");
}

int main()
{
    int a[10] = { 1, 3, 5, 7, 9, 11, 13, 15, 17, 19 };

    print_array(a, 10);
    print_array_r(a, a + 10);
}
```

Bittiği yere gelene kadar yazdır. Range yöntemi.

Örnek < > karşılaştırması

```c
#include <stdio.h>
#include "nutility.h"


int main()
{
    int x = 10;
    int y = 20;
    int* p = &x;
    int* q = &y;

    if (p > q) {  //unspecified

    }
}
```

Burada bu iki nesneyi karşılaştırmamızın hiçbir anlamı yok derleyiciye bağlı büyük veya küçük olması.

Örnek reverse fonksiyonunu tekrar yapalım karşılaştırm ile.

İlk yöntem

```c
void reverse_array(int* p, int size)
{
    int* pe = p + size;
    int n = size / 2;

    while (n--) {
        swap(p++, --pe);
    }
}
int main()
{
    int a[SIZE];

    randomize();
    set_array_random(a, SIZE);
    print_array(a, SIZE);
    reverse_array(a, SIZE);

}
```

Karşılaştırm ile yapalım

```
void reverse_array(int* p, int size)
{
    int* pe = p + size;

    while (p < pe) {
        swap(p++, --pe);
    }
}
int main()
{
    int a[SIZE];

    randomize();
    set_array_random(a, SIZE);
    print_array(a, SIZE);
    reverse_array(a, SIZE);
    print_array(a, SIZE);

}
```

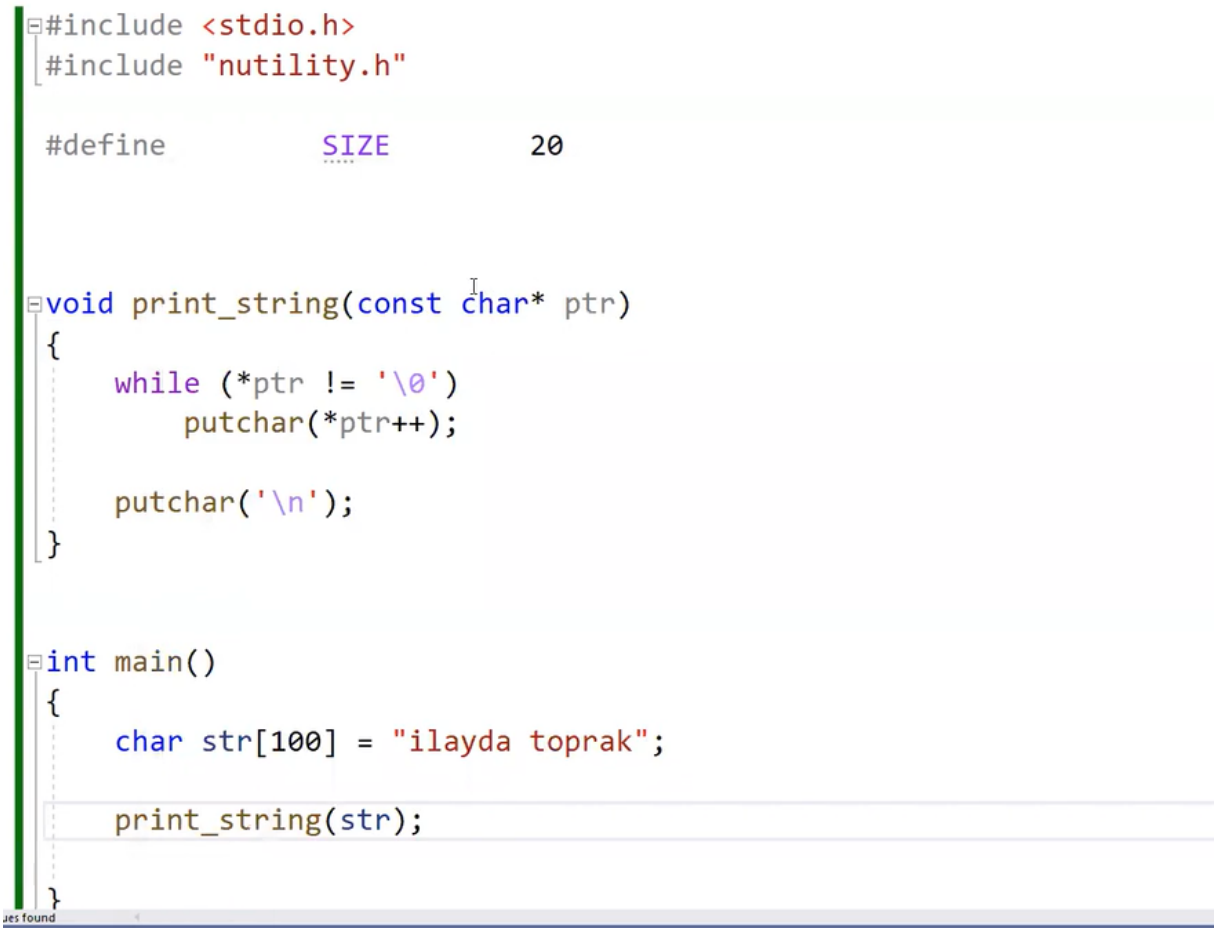
Örnek yanlış size of kullanımı Word

```
void func(const int* ptr)
{
    for (int i = 0; i < sizeof(ptr) / sizeof(*ptr); ++i)
        printf(_Format:"%d ", ptr[i]);
}

int main()
{
    int a[100];


    func(ptr:a);
```

Burada size of içindeki ptr dizinin yanlızca ilk elemanının değerini tutar ptr değişkeni 4 byte ilk eleman 4 byte 1 kere döner. Size bilgisi olmadan fonksiyonlarda işlem yapamayız.

Bu kullanım yanlış .

Ancak dizi string olsaydı yazı karakterleri için size a gerek duymazdık.



```c
#include <stdio.h>
#include "nutility.h"

#define         SIZE        20


void print_string(const char* ptr)
{
    while (*ptr != '\0')
        putchar(*ptr++);

    putchar('\n');
}


int main()
{
    char str[100] = "ilayda toprak";

    print_string(str);

}
```

Sonunda null karakter olma zorunluluğundan dolayı.

Örnek basit

```c
#include <stdio.h>
#include "nutility.h"

int g = 999;

int* func(void)
{
    //...

    return &g;
}


int main()
{
    int* p = func();

    printf("*p = %d\n", *p);
    ++*p;
    printf("g = %d\n", g);
```

*p artık g değişkenini gösteriyor son satırdada 1 arttı 1000 oldu.

```c
#include <stdio.h>
#include "nutility.h"

int g = 999;

int* func(void)
{
    //...

    return &g;
}


int main()
{
    printf("%d\n", *func());
    ++* func();

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


}
```

Değerini değişkende tutmaya gerek yok adres döndürdüğü için *func() ile nesneyi gösteriyor . 3 farklı operatör kullanılmış.

Örnek tuhaf dizi çağırma

```
#include "nutility.h"

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

int* func(void)
{
    //...

    return a;
}


int main()
{
    print_array(a, 5);
    func()[3] = 999;
    print_array(a, 5);


}
```

Daha önce hep a[] diye çağırdık ancak func dizinin ilk adresini döndürmüyor mu.

Func()[3] = 999 dizinin 4. Elemanı oluyor.

Örnek ilginç fonksiyonun argüman olması

```
int *bar(void);
int* foo(void);


int main()
{
    swap(bar(), foo());

}
```

```
#include <stdio.h>
#include "nutility.h"


void func(int* p);
int* foo(void);


int main()
{
    func(foo());

}
```

```c
int g1 = 567;
int g2 = 333;

int* bar(void)
{
    return g1;
}

int* foo(void)
{
    return &g2;
}


int main()
{
    printf("g1 = %d  g2 = %d\n", g1, g2);
    swap(bar(), foo());

    printf("g1 = %d  g2 = %d\n", g1, g2);
```

Başka bir fonksiyona argüman olarak diğer fonksiyonun adres değerini yani gönderdiği değişkenin adresini verdik

Örnek otomatik ömürlü adres döndüren fonksiyon

```c
#include <stdio.h>
#include "nutility.h"

// warning C4172: returning address of local variable or temporary: x

int* scan_value(void)
{
    int x;

    printf("bir sayi girin: ");
    scanf("%d", &x);

    return &x;
}

int main()
{
    int* p = scan_value();

    printf("alinan deger = %d\n", *p);

}
```

X in ömrü bitti ama adresi kullanılıyor bu tanımsız davranış adreste artık x yok değişkeni yok.

```c
char* get_name(void)
{
    char str[100];

    printf("bir isim girin: ");
    sgets(str);

    return str;
}


int main()
{

}
```

Aynı hata kod doğruda çalışsa tanımsız davranıştır.

```c
#include "nutility.h"

// warning C4172: returning address of local variable or temporary: x


    char str[100];

char* get_name(void)
{

    printf("bir isim girin: ");
    sgets(str);

    return str;
}


int main()
{
    char* p;

    p = get_name();

    printf("isim = %s\n", p);
    ///

}
```

Bu geçerli static global değişken

Örnek parametre değeri geri dönüş değeri olan fonksiyon

```c
#include <stdio.h>
#include "nutility.h"


int* foo(int* p)
{
    //code
    *p *= 10;
    //
    return p;
}

void func(int*);

int main()
{
    int x = 45;

    func(foo(&x));


}
```

Foo hem argüman olarak hemde geri dönüş adresi olarak x in adresini kullandı hatta bunu başka bir fonksiyona adres argümanı yapabilirim.

Dizinin en büyük değerinin adresi

```c
#define         SIZE        100

int get_array_max(const int* p, int size);
//int* get_array_max_a(const int* p, int size);


int main()
{
    int a[SIZE];
    ///


}
```

Geri dönüş değeri adres olması ile olmaması arasındaki fark nedir

```
#define         SIZE        100

int* get_array_max(const int* p, int size);

int main()
{
    int a[SIZE];
    ///
    int* pmax = get_array_max(a, SIZE);

    *pmax = -1;

}
```

Dizinin en büyük değerli adresini döndürerek bu adresdeki indise değiklik yapabiliriz farkı bu oluyor.

```
#define         SIZE        100

int* get_array_max(const int* p, int size);

int main()
{
    int a[SIZE];
    ///
    *get_array_max(a, SIZE) = -1;


}
```

Yukardaki örneğin kodu

```
int* get_array_max(const int* p, int size)
{
    const int* pmax = p;

    for (int i = 1; i < size; ++i) {
        if (p[i] > *pmax) {
            pmax = p + i;
        }
    }

    return (int*)pmax;
}


int main()
{
    int a[SIZE];

    randomize();
    set_array_random(a, SIZE);
    print_array(a, SIZE);

    int* pmax = get_array_max(a, SIZE);
    printf("max = %d ve dizinin %d indisli elemani\n", *pmax, pmax - a);
    *pmax = -1;

    print_array(a, SIZE);
    printf("max = %d\n", *get_array_max(a, SIZE));
```

Burada dönen adres en büyük dizinin en büyük değerli nesnesinin adresi

Soru 1 dizinin en büyük elemanından başlayarak hepsini print array ile yazdır.

```c
int main()
{
    int a[SIZE];

    randomize();
    set_array_random(a, SIZE);
    print_array(a, SIZE);
    //285 911 759 542 568 689  41 525 794 713
    //911 759 542 568 689  41 525 794 713
    int* pmax = get_array_max(a, SIZE);

    print_array(pmax, SIZE - (pmax - a));



}
```

Pmax adresinden başlasın sonra

Size eksi index değeri yapıp kaç eleman olduğunu yazsın demek

Soru2 print array ile en büyük elemana kadar

Cevap print_array(a,pmax-a+1)

```c
int main()
{
    int a[SIZE];

    randomize();
    set_array_random(a, SIZE);
    print_array(a, SIZE);
    //285 911 759 542 568 689  41 525 794 713
    //911 759 542 568 689  41 525 794 713
    int* pmax = get_array_max(a, SIZE);

    print_array(pmax, SIZE - (pmax - a));
    print_array(a, pmax - a + 1);


}
```

Soru3 en küçük ile en büyük değeri takas eden fonksiyon

```
    return (int*)pmax;
}


int main()
{
    int a[SIZE];

    randomize();
    set_array_random(a, SIZE);
    print_array(a, SIZE);

    swap(get_array_min(a, SIZE), get_array_max(a, SIZE));
    print_array(a, SIZE);

}
```

Aynı kodu min için yazıp swap fonksiyonuna bu adresleri fonk olarak verdik.

```
int main()
{
    int a[SIZE];

    randomize();
    set_array_random(a, SIZE);
    print_array(a, SIZE);

    int* pmax = get_array_max(a, SIZE);
    int* pmin = get_array_min(a, SIZE);
    swap(pmin, pmax);
    print_array(a, SIZE);

}
```

Değeri pointer ile gösterdik.

Soru 3 min max arası değerleri yazdır.

```c
int main()
{
    int a[SIZE];

    randomize();
    set_array_random(a, SIZE);
    print_array(a, SIZE);

    int* pmax = get_array_max(p:a, size:SIZE);
    int* pmin = get_array_min(p:a, size:SIZE);

    if (pmax > pmin) {
        print_array(pmin, pmax - pmin + 1);
    }
    else {
        print_array(pmax, pmin - pmax + 1);
    }

}
```

Hangisi büyük bilmediğimizden karşılaştırma yapıyoruz çünkü adreslerde büyükten küçük çıkartmalıyız.

Selexion sort dizinin en küçük elemanı ile ilk elemanı takasla sonra diziyi birsonraki elemandan başlat bu şekilde sırala

```c
#define         SIZE        100

int* get_array_min(const int* p, int size)
{
    const int* pmin = p;

    for (int i = 1; i < size; ++i) {
        if (p[i] < *pmin) {
            pmin = p + i;
        }
    }

    return (int*)pmin;
}


void selection_sort(int* p, int size)
{
    for (int i = 0; i < size - 1; ++i) {
        swap(get_array_min(p + i, size - i), p + i);
    }
}
```

Min ile aladığımız değeri sort func ta kullandık.

Örnek statik değişken döndürme hatası.

```
char* get_name(void)
{
    static char buffer[1000];

    printf(_Format:"ismi girin: ");
    scanf(_Format:"%s", buffer);

    return buffer;
}


int main()
{
    char* p1 = get_name();
    char* p2 = get_name();
    char* p3 = get_name();
    char* p4 = get_name();

    printf(_Format:"iste size 4 tane isim aldik (%s) (%s) (%s) (%s)\n", p1, p2, p3, p4);

}
```

Son olarak bütün pointerlar aynı static değişken adresini tutuyordu yani aynı dizininadresi olduğu için son yazırılan hepsi aynı yazıyı yazar.