Ders 37

T türü neolursa derleyici uyarı hatası vermez.

```
int main(void)
{
    T x = &x;

}
```

Cevap void*

Void* x = &x olabilir.

Void * void** adresi tutuyor kendi turunun adresini bir tek void yıldız tutabilir herhangi bir tür adresi tutabilir.

////////////////////////////// //

Function pointer

Nasıl function adresinini bir değişkende tutabilirim.

```
#include "nutility.h"

int foo(int, int);

int main(void)
{
    //foo işlevinin adresini bir değişkende tutmak istiyorum
    //bu değişkeni nasıl tanımlamalıyım

    int (*fptr)(int, int);


}
```

Fonksiyon adresini aşağıdaki gibi bir değişken tanımlayarak tutabilirm.

İnt (*fptr) (int,int)

Soru foo adresi tutmak

```c
int foo(int, int);
int bar(int, int);

int main(void)
{
    //foo işlevinin adresini bir değişkende tutmak istiyorum
    //bu değişkeni nasıl tanımlamalıyım

    //int (*fptr)(int, int) = &foo;
    int (*fptr)(int, int) = foo;


}
```

Bar fonksiyonunu fptr ye nasıl atama yaparsın.

```c
int foo(int, int);
int bar(int, int);

int main(void)
{
    //foo işlevinin adresini bir değişkende tutmak istiyorum
    //bu değişkeni nasıl tanımlamalıyım

    //int (*fptr)(int, int) = &foo;
    int (*fptr)(int, int) = foo;

    fptr = bar;
    fptr = &bar;


}
```

Normal pointer adres ataması gibi yapılır ancak tür uyumuna dikkat edilmeli.

//

Strlen için pointer tanımlayınız .

```
int main(void)
{
    //strlen işlevinin adresi ile tanımladığınız bir pointer değişkene ilk değer verin

    size_t(*fp)(const char*) = strlen;

}
```

Strcmp ile yap

```
int main(void)
{
    //strlen işlevinin adresi ile tanımladığınız bir pointer değişkene ilk değer verin

    int (*fptr)(const char*, const char*) = &strcmp;

}
```

Ste _array random için yaz

```
int main(void)
{
    //strlen işlevinin adresi ile tanımladığınız bir pointer değişkene ilk değer verin

    //int (*fptr)(const char*, const char*) = &strcmp;
    void (*fp)(int*, int) = set_array_random;

}
```

//////////////////////örnek//

Parametresi ve geri dönüş değeri strcmp olan fonksiyonu yazınız.

Bu kodu 4 5 yıllık programcılar yazamaz çünkü bunlar typedef ile yapılıyor.

```
//öyle bir fonksiyon bildirin ki
//iki parametresi olsun her bir parametre strcmp gibi fonksiyonların adreslerini istesin
//geri dönüş değeri de strcmp işlevinin adresi olsun

int(*foo(int (*)(const char *, const char *), int (*)(const char*, const char*)))(const char *, const char *);

int main(void)
{

}
```

Örnek//

Function pointer adresini tutan değişkeni yazınız.

```
int main(void)
{
    int (*fp)(const char*, const char*) = &strcmp;
    int (**fpp)(const char*, const char*) = &fp;
    //fp değişkeninin adresi ile bir pointer değişkene ilk değer verin

}
```

Aslında mantık aynı fonksiyonu tutan değişken pointer function

Onuda tutan ** pointer to function pointer.

2 tane ** koyuyoruz.

Örne//

Elemanları fonksiyon pointer olan dizi

```
int main(void)
{
    int (*fp)(const char*, const char*);
    int (*fpa[])(const char*, const char*) = { &strcmp, &strcoll };



    //fp değişkeninin adresi ile bir pointer değişkene ilk değer verin

}
```

Typedef örnekleri //

```
int foo(int, int);

typedef int (*FPTR)(int, int);
```

FPTR eş isim oldu basir 3 kural kullanıldı.

Strcmp için yazalım

```
typedef int (*FCMP)(const char*, const char*);
```

örnek

Parametresi ve geri dönüş değeri strcmp olan fonksiyonu yazınız.

Bu kodu 4 5 yıllık programcılar yazamaz çünkü bunlar typedef ile yapılıyor.

Aynı örneği typedef ile yazalım.

```
////geri dönüş değeri de strcmp işlevinin adresi olsun
//
int(*foo(int (*)(const char *, const char *), int (*)(const char*, const char*)))(const char *, const char *);

FCMP foo(FCMP, FCMP);

//
```

Aynı örnek çok basit oldu.

```
//
int main(void)
{
    //int (*fp)(const char*, const char*);
    FCMP fp;

```

Burada yotum satırı yerine eş isimi kullandık.

Örnekler bütünü

```
//
int(*foo(int (*)(const char *, const char *), int (*)(const char*, const char*)))(const char *, const char

FCMP foo(FCMP, FCMP);

//
//
//
int main(void)
{
    //int (*fp)(const char*, const char*);
    FCMP fp;

    FCMP* fpp = &fp;


    //int (*fpa[])(const char*, const char*) = { &strcmp, &strcoll };
    FCMP fpa[] = { strcmp, strcoll };
//
//
```

Bu şekilde kolay yoldan yazabiliriz.

//

Örnek 1

```
void f2(void)
{
    printf("f2 called!\n");
}

void f3(void)
{
    printf("f3 called!\n");
}

int main()
{
```

```
int main()
{
    void (*fp)(void);

    //fp = &f1;
    fp = f1;

    fp();
    fp = f2;
    fp();
    fp = f3;
    fp();

}
```

```
void f3(void)
```

```
Microsoft Visual Studio Debug Console
f1 called!
f2 called!
f3 called!

D:\KURSLAR\NISAN_2022_C\Release\NISAN_2022_C.exe (process 3268) exited with code 0.
```

Gördüğünüz gibi aynı fonksiyon çağrısı ile 3 farklı fonksiyon çağırabildik

Aslında bu direk pointer antığı aynı pointer nasıl farklı nesneleri görüyorsa fonksiyon pointerı farklı pointerları atnı değişken ile çağırabiliyor. Adreslerinden erişiyoruz çok zor değil aslında.

//

Örnek * fp nasıl olacak

```
{
    printf("f3 called!\n");
}

int main()
{
    void (*fp)(void);

    fp = &f1;

    fp();
    (*fp)();


}
```

Burada derleyici *fp ifadesinide func ın adresi olarak görecek yani değişen bişey yok kodu okurken bunun pointer olduğunu göstermek için bazen *fb kullanıyorum ancak bunu () içinde kullanmazsak öncelik yüzünden yanluş olur.

Örnek espirili

```c
#include <stdio.h>


void func(void)
{
    printf(_Format:"func cagrildi\n");
}

int main()
{
    func();
    (&func)();
    (*func)();

}
```

3 kodda aynı işlevi yapar hepsi adresi demek

Örnek basit call function yapısı

```c
void func(void(*fp)(void))
{
    printf(_Format:"func fonksiyonu cagrildi\n");
    fp();
    pdline();
}
```

Burada printf ve pdline fonksiyonlarının çağırılmasına kim kadar verdi

Kararı func verdi ancak fp fonksiyonun çağırılmasına çağıran taraf karar veriyor.

Foknsiyonun ne olduğunu çağıran biliyor.

```c
void func(void(*fp)(void))
{
    printf("func fonksiyonu cagrildi\n");
    fp();
    pdline();
}

void f1(void)
{
    printf("f1 cagrildi\n");
}

void f2(void)
{
    printf("f2 cagrildi\n");
}

void f3(void)
{
    printf("f3 cagrildi\n");
}

int main()
{
    func(f1);
}
```

Burada func a f1 fonksiyonunu çağırtıyoruz.

Aslında hersey aynı türünü yazdığım değişken adresi gibi fonksiyonununda türünü parametrede yazdım o türe ait bir fonksiyon adresi gönderdim

F2 ve f3 çağıralada bilirdi

```c
int main()
{
    func(&f1);
    func(&f2);
    func(&f3);
}
```

Bu şekildede yapabilirdik.

Örnek çok güzel bir kod örneği

```c
int main()
{
    printf("isupper\n");
    for (int i = 0; i < 128; ++i) {
        if (isupper(i))
            printf("%c", i);
    }
    printf("\n");

    printf("islower\n");
    for (int i = 0; i < 128; ++i) {
        if (islower(i))
            printf("%c", i);
    }
    printf("\n");
}
```

Daha önce bu kodları yazmıştık bunlar standart fonksiyonlar

Bunları tek bir fonksiyon ve çağrı ifadesiyle nasıl yazarız.

```c
void print_chars(const char* pfname, int (*fptest)(int))
{
    puts(pfname);

    for (int i = 0; i < 128; ++i) {
        if (fptest(i))
            printf("%c", i);
    }
    printf("\n");
}

int main()
{
    print_chars("isupper", &isupper);
    print_chars("islower", &islower);
    print_chars("isdigit", &isdigit);
    print_chars("isalnum", &isalnum);
```

Bu fonksiyonların hepsinin tüt aynı bu yüzden hepsini aynı şekilde çağırabiliyorum.

//

örnek pekiştirme

```
(void)(*fgp)(void) = &foo;

void set_func(void (*fptr)(void))
{
    fgp = fptr;
}

void func(void)
{
    fgp();
}


int main()
{
    set_func(&bar);
    func()
}
```

Bu idiom genellikle embeded tarafta çok fazla karşımıza çıkar.

Biz bir global fonksiyon pointer tanımladık.

Fonksiyon pointera ilk değer olarak default bir fonksiyon adresi verdik.

Daha sonra bu global fonksiyon pointeri değiştirebilmek için bir fonksiyon tanımladık bu fonksiyonunda parameresi fonksiyon pointerdır. Bunun ile ilk golabal pointeri değiştirebiliriz.

Eğer main içinde func ı direk çağırırsak o default foo yu çağıracak

Ama set func ile çağıracağı fonksiyonu değiştirebilirim bu çok güzel bir kullanım örneği.

//

Ödev var partition ////////////////////////////////

//

Qsort fonksiyonu öğreneliö

```
void qsort(void *vpa, size_t size, size_t sz, int (*fcmp)(const void*, const void*));
```

//

Örnek qsort ile generic ve function pointer kullanark küçükten büyüğe bir sıralama yapmak

```
int icmp(const void* vp1, const void* vp2)
{
    if (*(const int*)vp1 > *(const int*)vp2)
        return 1;

    if (*(const int*)vp1 < *(const int*)vp2)
        return -1;

    return 0;
}

int main()
{
    int a[SIZE];

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

    qsort(a, SIZE, sizeof(int), &icmp);
}
```

Burada icmp adresini verdik burda fonksiyon tanımlardan void olarak bildirdik ancak ben sıralamak istediğim elemanların int olduğunu biliyorum bu yüzden fonksiyon içinde int karşılaştırdım ve sonucunda 1 -1 veya 0 dönmesini sağladım bu şekilde generic fonksiyonda işimin bir bölümünü icmp ile çağırttırıp yaptırdım.

Birkaç farklı yazım yolu

```
int icmp(const void* vp1, const void* vp2)
{
    return  *(const int*)vp1 - *(const int*)vp2;
}
```

Burada iş görür ancak taşma riski var ama klas bir kod ama riskli yazma

Koşul operatörü ile

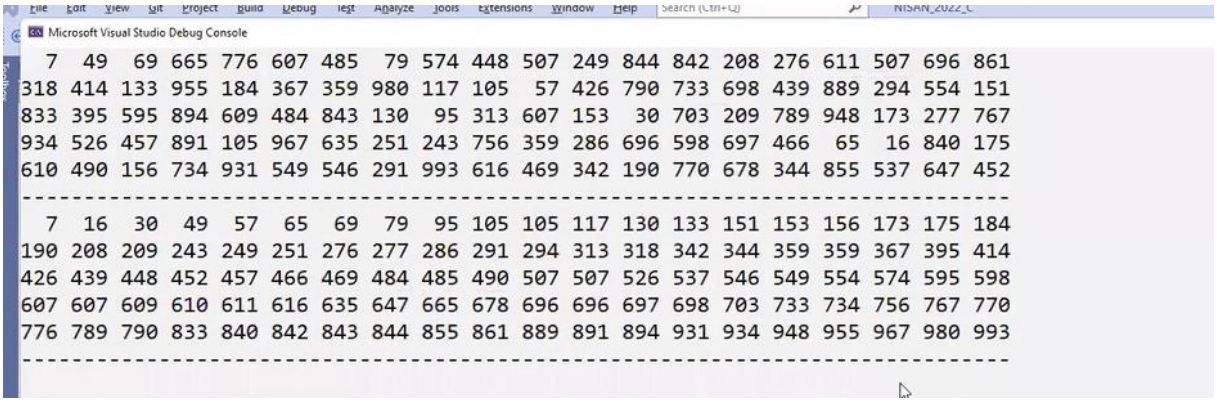
```
int icmp(const void* vp1, const void* vp2)
{
    return *(const int*)vp1 > * (const int*)vp2 ? 1 :
        *(const int*)vp1 < *(const int*)vp2 ? -1 : 0;


}
```

```
int icmp(const void* vp1, const void* vp2)
{
    int left = *(const int*)vp1;
    int right = *(const int*)vp2;

    left > right ? 1 :
    left < right ? -1 : 0
}
```

Hangisini kullanırsakta farketmez sıralama yazpılmış olur.



//////////////////////////////////////

Dizi double olsaydı.

```
int dcmp(const void* vp1, const void* vp2)
{
    if (*(const double*)vp1 > *(const double*)vp2)
        return 1;

    if (*(const double*)vp1 < *(const double*)vp2)
        return -1;

    return 0;
}
```

```c
int main()
{
    double da[] = {
        4.8745,
        41.743,
        -1.2,
        3.4, 0.5, 7.7, 0.234, 7.1234, 2.222 ,3.333 ,-4.56
    };

    qsort(da, asize(da), sizeof(*da), dcmp);

    for (size_t i = 0; i < asize(da); ++i)
        printf("%f\n", da[i]);
}
```

Burada double fonksiyonu küçükten büyüğe sıraldık aynı şeyler sadece sıralamda double kullandık.

//

Kendi generic sort algoritmamızı oluşturalım

```c
void gbsort(void* vpa, size_t size, size_t sz, int (*fcmp)(const void*, const void*))
{
    char* p = (char*)vpa;

    for (size_t i = 0; i < size - 1; ++i) {
        for (size_t k = 0; k < size - 1 - i; ++k) {
            if (fcmp(p + k * sz, p + (k + 1) * sz) > 0) {
                gswap(p + k * sz, p + (k + 1) * sz, sz);
            }
        }
    }
}

int icmp(const void* vp1, const void* vp2)
{
    if (*(const int*)vp1 > *(const int*)vp2)
        return 1;

    if (*(const int*)vp1 < *(const int*)vp2)
        return -1;

    return 0;
}
```

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

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

    gbsort(a, SIZE, sizeof(int), &icmp);

    print_array(a, SIZE);
}
```

Aslında yukarda yaptığımız test kodlarını sakladık ve sadece gbsort algoritmasını yazdık

Burda herzaman yazdığımız bouble sort argoritmasını kullandık

Bu algoritmayı zaten generic olrak void* ile oluşturmayı biliyoruz.

Asıl olay burda karşılaştırma işleminde

Bununu parametresi void* void * olan bir fonksiyon oluşturuyoruz

Bu fonksiyonu zaten dışardan çağırdığımız için işlemi kendi yapıyor bize değerini iletiyor.

Sonrası kolay daha önce yazdığımız gswap ilede swap yapıyoruz.

Buradaki ana mantık parametreden bağımsız biçimde bu kodu her değişken için kullanmak istedğimiz dizinin türünü sadece çağırırken vermek extra her tür için kod yazmamak.

//

Örnek generic partition sorusu ödev

```
void* g_partition(void* vpa, size_t size, size_t sz, int (*fp)(const void*));


int f_isprime(const void* vp)
{
    return isprime(*(const int*)vp);
}

int is_even(const void *vp)
{
    return *(const int*)vp % 2 == 0;
}

int main()
{
    int a[SIZE];

    randomize();
    set_array_random(a, SIZE);
    print_array(a, SIZE);
    //int* pp = g_partition(a, SIZE, sizeof(*a), f_isprime);
    int* pp = g_partition(a, SIZE, sizeof(*a), is_even);
    printf("partisyon indeksi: %d\n", pp - a);

    print_array(a, SIZE);
```

Yukarda test foksiyonu yazdık

Direk isprime yapamazmıydım.

```
print_array(a, SIZE);
int* pp = g_partition(a, SIZE, sizeof(*a), (int (*)(const void *)) &isprime);
```

Bu şekilde tür dönüştürme ile yapılabilir. Buda

Ödevi yaz.

Printf ile sağlamayan ilk adresi yazdır.

//

Örnek zor soru alfabetik sıralama qsort ile yaz

```c
//dikkat
// qsort işlevinin son parametresien adresini geçeceğiniz (callback) fonksiyonun implementasyonunda
// std strcmp işlevini çağırmalısınız

int scmp(const void* vp1, const void* vp2)
{
    //???????????
}

int main()
{
    char* p[] = { ... }

    for (size_t i = 0; i < asize(p); ++i) {
        printf("%s ", p[i]);
    }

    printf("\n\n");
    //qsort fonksiyonuna çağrı yaparak p dizisini sıralayin
    qsort(p, asize(p), sizeof(char*), &scmp);

    for (size_t i = 0; i < asize(p); ++i) {
        printf("%s ", p[i]);
    }
}
```

Ödev bu scmp nasıl yazılmalı

İçeriği soru işareti olan yer strcmp ile karşılaştırma yap.

//

Örnek bsaerce fonksiyonu hazır

```c
#include "nutility.h"

#define  SIZE           20


int icmp(const void* vp1, const void* vp2)
{
    if (*(const int*)vp1 > *(const int*)vp2)
        return 1;

    if (*(const int*)vp1 < *(const int*)vp2)
        return -1;

    return 0;
}

int main()
{
    int a[SIZE];

    randomize();
    set_array_random(a, SIZE);
    qsort(a, SIZE, sizeof(int), &icmp);
    print_array(a, SIZE);

    int sval;

    printf("aranacak degeri giriniz: ");
    scanf("%d", &sval);

    int* p = bsearch(&sval, a, SIZE, sizeof(int), &icmp);
    if (p) {
        printf("bulundu dizinin %d indisli elemani\n", p - a);
    }
    else {
        printf("bulunamadi\n");
```

issues found

Bu fonksiyon sıralanmış dizide bir değeri arıyor.