

## 11. hafta 1. ders

---- fonksiyonlara parametre verilmesi

- **function-name** arg1 arg2 arg3 argN shellde olduğu gibi argüman verilir.

- \$ vi [pass](#)

```
function demo()
{
    echo "All Arguments to function demo(): $*"
    echo "First argument $1" // 1. argüman
    echo "Second argument $2" // 2. argüman
    echo "Third argument $3" // 3. argüman
    return
}
# Call the function
demo -f foo bar // çağırma işlemi
```

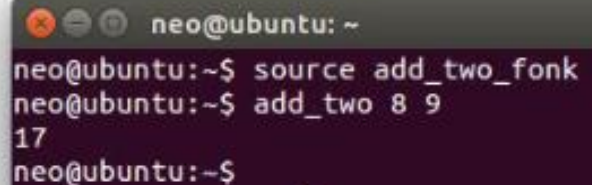
----- örnek

```
function add_two { // iki değer alır ve onu toplar
    (( sum=$1+$2 ))
    return $sum // sonucu döner
}
```

add\_two 1 3 // çağırma işlemi

echo \$? // dönen değeri dönüştürür. hata varsa -1 ya da 0, hata yoksa 1 ya da daha yüksek değer döner.

```
1 function add_two()
2 {
3     (( sum= $1 + $2 ))
4     echo $sum
5 }
6 #
7 # sonuc=$(add_two $1 $2)
8 # echo $sonuc
```

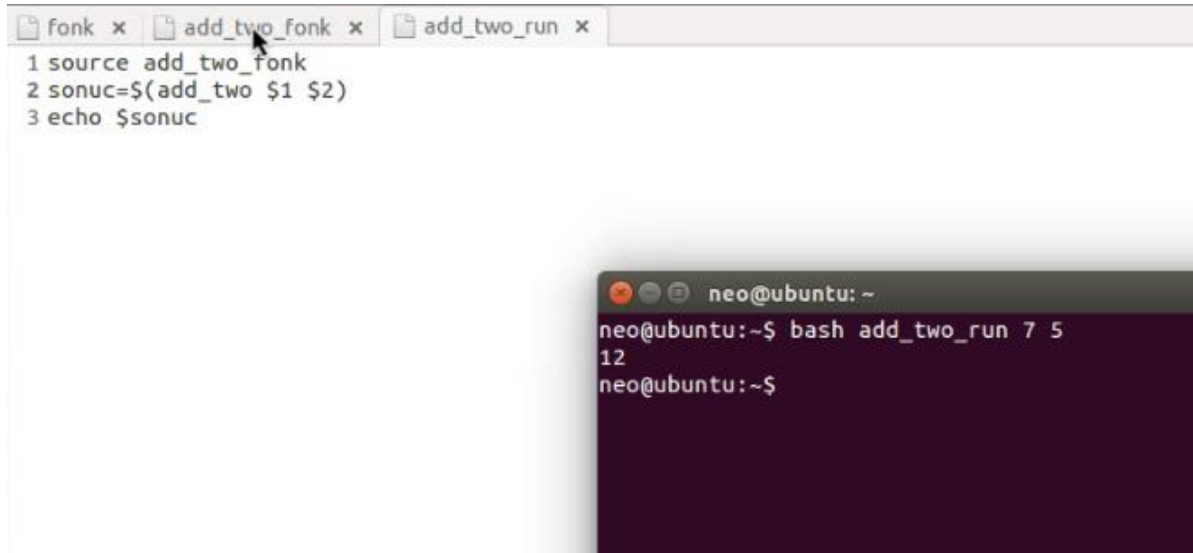


```
neo@ubuntu: ~
neo@ubuntu:~$ source add_two_fonk
neo@ubuntu:~$ add_two 8 9
17
neo@ubuntu:~$
```

fonksiyonun çağırılması.

- fonksiyonun kullanılmadan önce tanımlanması gerekiyor.

### Script üzerinden fonksiyon çağırılması;



The screenshot shows a terminal window with three tabs: 'fonk', 'add\_two\_fonk', and 'add\_two\_run'. The 'add\_two\_fonk' tab is active and contains the following script:

```
1 source add_two_fonk
2 sonuc=$(add_two $1 $2)
3 echo $sonuc
```

The 'add\_two\_run' tab is also visible, showing the execution of the script with arguments 7 and 5:

```
neo@ubuntu:~$ bash add_two_run 7 5
12
neo@ubuntu:~$
```

parametre verilmezse hata verir.

- Scripte çalıştırılırken fonksiyondan dönen değer bir değişkene aktarılır. değişken yazdırılır.
- Fonksiyonu tanıtırken argüman verilmezse; hata verir ve run time errordür. (.exe üretilir ama koşarken hata verir.) Tekrar tanıtmadan argümanla çalıştırılırsa hata vermeden çalışır.
- **echo \$? //** dönen değeri dönüştürür. hata varsa -1 ya da 0, hata yoksa 1 ya da daha yüksek değer döner.

### ----- Yığın işlemleri

popd, pushd, dirs

- **pop** = ekle
- **push** = çek
- **dirs** = tümünü listele

-----Pushd > boş yığına veri ekleme

pushd() { // bir argüman lacak dizinse yığına ekleyecek

REQ="\$1"; // girilen değeri aldık rq ya ttık

if [ -z "\$REQ" ] ; // boş mu

then REQ=. ; // boşsa bulunduğun yerde kal

fi

if [ -d "\$REQ" ] ; then // parametre dizinmi

cd "\$REQ" > /dev/null 2>&1 // dizinse oraya git hata ya da çıktı varsa ekrana çıktı verme

if [ \$? -eq 0 ] ; then // son koşan komut (\$) 1 mi 0 mı (0 true, 1 false) cd değişmişse yani

\_DIR\_STACK="`pwd`:\$ \_DIR\_STACK" ;

// işlem başarılıysa yığına ekle bulunduğun dizini göster dizinin değişmiş hali

export \_DIR\_STACK ; dirs

// \_DIR\_STACK bunu öncekinin üstüne ekle yığın mantığı yeni gelen üste eklenir. sonra dirs fonk çalışır

else

echo "ERROR: Cannot change to directory \$REQ." >&2

// cd çalışmadıysa hata verecekyetki nedeniyle girilmemiş olabilir.

fi

else

echo "ERROR: \$REQ is not a directory." >&2 // girilen argüman dizin değil

fi

unset REQ // req ile işi bitti ve değişkeni yok etti

}

## -----Dirs

```
dirs() {  
    OLDIFS="$IFS" // çevre değişkeni olan IFS yi oldifs de tutuyoruz. daha sonra düzeltereğiz  
    IFS=: // boşluk olarak belirli olan ifs ye : değeri verilir.  
    for i in $_DIR_STACK // daha önce push ile eklediğimiz yığın elemanları for a aktarılır.  
    do  
        echo "$i \c" // yığınlar son eklenenden başlanıp sıralanır. \c -> kursor aynı satırda kalır.  
    done  
    echo // alt satıra inme olmazsa kullanıcı adımız aynı satırın sonuna eklenir.  
    IFS="$OLDIFS" // IFS eski haline geri döner.  
}
```

Sonuç olarak \c den dolayı yan yana yazılır değerler.

\c -> kursor aynı satırda kalır

echoya -e parametre de verilmeli. bu sayede \x durumunda x karakteri yorumlanır.

```
neo@ubuntu:~$ source fonk  
neo@ubuntu:~$ dirs  
  
neo@ubuntu:~$ pushd  
/home/neo  
neo@ubuntu:~$ dirs  
/home/neo  
neo@ubuntu:~$
```

ilk ekleme

```
neo@ubuntu:~$ source fonk  
neo@ubuntu:~$ dirs  
  
neo@ubuntu:~$ pushd  
/home/neo  
neo@ubuntu:~$ dirs  
/home/neo  
neo@ubuntu:~$ pushd /  
/ /home/neo  
neo@ubuntu:/ $ pushd /dev  
/dev / /home/neo  
neo@ubuntu:/dev$ pushd /etc  
/etc /dev / /home/neo  
neo@ubuntu:/etc$ pushd /usr/bin  
ERROR: /usr/bin is not a directory.  
neo@ubuntu:/etc$ pushd /usr/bin  
/usr/bin /etc /dev / /home/neo  
neo@ubuntu:/usr/bin$ dirs  
/usr/bin /etc /dev / /home/neo  
neo@ubuntu:/usr/bin$
```

ekleme yapıldıkça eskisinin önüne yazılır.

-----Popd

```
popd() {  
    OLDIFS="$IFS" // ifs tekrar değiştirilir.  
    IFS=:  
    _popd_helper $_DIR_STACK // yığından işlem yaparak eleman alını. popdhelper da bir fonk.  
    IFS="$OLDIFS"  
}  
  
_popd_helper() {  
    POPD="$1" // üstteki fonksiyondan _popd_helper $_DIR_STACK değeri alınır. yığının tamamı  
    if [ -z "$POPD" ] ; then  
        echo "ERROR: The directory stack is empty." >&2 // ilk eleman boşsa ekrana çıktı  
        veriyor.  
        return 1  
    fi  
    shift // 2. elemana geçiliyor.  
    if [ -n "$1" ] ; then // yığın boş değil mi? yığın tek elemanlı mı anlamını da taşıyor.  
        _DIR_STACK="$1" ; // boş değil 2. eleman tutuluyor.  
        shift ; // tekrar kaydırma işlemi yapıyor. 3. elemana geçtik.  
        for i in $@ ; // geriye kalan elemanları yazdırır.  
        do _DIR_STACK="$_DIR_STACK:$i" ;  
            // 2. eleman kaybolmasın diye tuttuk tekrar yığına ekledik  
        done  
    else // yığında 1 eleman vardı $1 in değeri boşsa burası çalışır.  
        _DIR_STACK=  
    fi  
    if [ -d "$POPD" ] ; then // popd bir dizinse  
        cd "$POPD" > /dev/null 2>&1 // tekrar dizin değiştirilir. (eklenen her şey dizindi)  
        if [ $? -ne 0 ] ; then
```

// dirste bu durum kontrol edilip eklenmişti izin değiştirilmişse ya da silinmişse bu durum geçerli olur.

**echo "ERROR: Could not cd to \$POPD." >&2 // dizini değiştiremedi**

**fi**

**pwd // işlem başarılıysa bulunduğu yer**

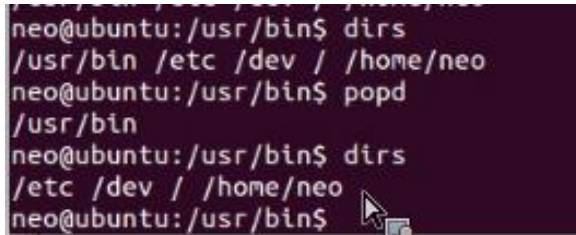
**else**

**echo "ERROR: \$POPD is not a directory." >&2 //çekilmek istenen değer dizin değilse buraya gelir**

**fi**

**export \_DIR\_STACK // export edilerek**

**unset POPD // yok edilir.**



```
neo@ubuntu:/usr/bin$ dirs
/usr/bin /etc /dev / /home/neo
neo@ubuntu:/usr/bin$ popd
/usr/bin
neo@ubuntu:/usr/bin$ dirs
/etc /dev / /home/neo
neo@ubuntu:/usr/bin$
```

- yığın ilk satırda **dirs** çalıştırılmasıyla ekrana basıldı.
- **popd** ile ilk eleman çekildi.
- son satırda yığının son hali var.

**NOTTTTTTTTTT**; shift kullanıldığında 1. eleman 2. eleman olur; 2. eleman 3. eleman olur. kaydırma işlemi olduğu için.

Kodda \$1 ilk önce tüm yığını tutuyordu, daha sonra shift satırında elemanlar kaydı ve \$1 yığının 2. elemanını tutmaya başladı. (neden 2. eleman kaydırma işlemi yaptığı için.)

## 11. hafta 2. ders

kodda **POPD="\$1"** ile ilk eleman tutuluyor.

**echo [options] [string, variables...]**

**-n ->** bulunduğu satırda kalmamızı sağlar **echo -n = printf**

**-e ->** daha sonra eklenen\ ifadesi okunur.

- \c suppress trailing new line
- \a alert (bell) - **bip sesi verir -n ile kullanılabilir.**
- \b backspace -  
   \n new line - **echo -e \n = echo -n = printf**
- \r carriage return -  
   \t horizontal tab – **elemanlar arası yatayda tab kadar boşluk bırakır.**
- \\ backslash – **bir tane \ yazmış oluruz.**

-----Reklendirmee

echo -e "\033[34m Hello Colorful World!" // 033 escape karakter.

[34m mavi yazar

30 – 40 ' a kadar ve daha fazla renk var. bu şekilde renkli yazar.

0m bol yazar.

1m

2m veri eski haline döner

7m arkaplan beyaz yazı rengi siyah

5m çalışmıyormuş

h	ANSI modunu ayarlayın
l	ANSI modunu temizler
m	Karakterleri farklı renklerde gösterin veya BOLD ve Blink gibi efektler
q	Turns keyboard num lock, caps lock, scroll lock LED on or off
s	Stores the current cursor x, y position (col , row position) and attributes
u	Restores cursor position and attributes

h l m q Klavye num kilidini açar, büyük harflerle kilit, kaydırma kilidi LED'i açık veya kapalı s Mevcut imleç x, y konumunu kaydeder (sütun, satır konumu) ve öznitelikler u İmleç konumunu ve niteliklerini geri yükler

## ----- Sistem programlama

gcc, g++ -> derleyiciler

**g++/gcc hello.cpp**

yapılırsa a.out'a çıktı iletilir. Bunun yapılması tavsiye edilmeeez

**g++ -o hello hello.cpp**

o parametresi ile hello.cpp dosyası koşulur. hello adlı dosyadan çıktıya bakılır.

**g++ -o hello hello.cpp util.cpp**

hello.cpp ve util.cpp koşulur ve çıktı hello ya aktarılır. ikisi beraber derlenir

**g++ -c hello.cpp**

**g++ -c util.cpp**

**g++ -o hello hello.o util.o** // -o hem derleme hem de dosya üretme

**g++ -c** compile edilir. sadece derler. objeyi oluşturur.

**g++ -o <file>** çıktı eklenen dosyaya aktarılır. genelde aynı dosya ismi verilir. verilmezse a.out oluşur. onun üstüne yazılır.

**g++ -D** makro üretir

**g++ -I** kütüphaneler eklenir.

**g++ -I (i)** verilen dizin üzerindeki dosyaları ekler

**g++ -L** verilen dizin üzerindeki kütüphaneler kullanılabilir.

**#ifdef DEBUG**

printf("value of var is %d", var); // debug modundaysa bu satırı çalıştır. değilse iften çık

**#endif**

**g++ -DDEBUG -o prog prog.c** // debug modunda çalıştırma

---- **make**

bütün dosyaların düzgünce çalışması.

dosyaların izlenmesi (değiştirilmesi)

dosyalar arası bağımlılıklar. bunları otomatik yapar

[Mm]akefile şeklinde çalışır.



bağılılık zincirinde en bağımsızdan en bağımlıya doğru hareket edilir.

kendisine bağlı olan alt dosyalar oluşturulur

# Makefile for mydb

```
mydb: mydb.o user.o database.o //makefile
```

```
g++ -o mydb mydb.o user.o database.o // dosyalar yazılıyor.
```

```
mydb.o : mydb.cpp mydb.h // bağıladı
```

```
g++ -c mydb.cpp // taba basılmış sonra komut
```

```
user.o : user.cpp mydb.h // bağıladı bağımlılık satırı. target : bağımlılık
```

```
g++ -c user.cpp
```

```
database.o : database.cpp mydb.h // bağıladı
```

```
g++ -c database.cpp
```

bağımlılık satırı:

tabla başlar

**\$@** target hedefi listeler

**\$?** dependencies bağımlılıkları listeler

----- macros çok değinmemiş -D ile oluşturuluyordu.

tekrar bakılması gereken bir konudur.

:P bittiiiiiiiiimm