Számítógép Architektúrák

6-7. Gyakorlat

Idézőjelek közti különbségek

- ’…’, ”…..”, ´….´ közötti különbségek
- echo ’cat $1’ //mindent úgy ír ki, ahogy az idézőjelek között van
- echo ”cat $1” //a változók értéke behelyettesítődik
- echo ´cat $1´ //parancs „kifejtésére”

Paraméterbehelyettesítés

- **${változó:-érték}** – ha változó be van állítva és az értéke nem 0, akkor azt adja vissza, egyébként pedig az értékét.

Pl: ls ${1-~/alma} ha nincs pozicionális paraméter //default paraméterek beállitására

- **${változó:=érték }** – ha változó nem definiált vagy 0 sztring, akkor felveszi az értéket, különben ne veszi fel.

Paraméterbehelyettesítés

- **${változó:?érték }** – ha változó be van állítva az értéke és nem 0, akkor kifejtődik, különben kiíródik az érték és exitál a shell.
- **${változó:+érték }** – ha változó definiált és az értéke nem 0, akkor behelyettesítődik az érték, különben semmi sem fejtődik ki.

Vezérlési szerkezetek – if

- Gyakran előfordul, hogy programunkban valamilyen feltételtől függően szeretnénk parancsot végrehajtani. Például akkor szeretnénk végrehajtani valamilyen utasítást, ha az i változó értéke nagyobb mint **5.** Erre általánosan a következő utasítás szolgál:

C/C++/Java…

if *fetétel* **then** *parancs*;

```
if (i<4) parancs;
else if () {
    parancs(ok);
}
else parancs;
```

if általánosságban

- A feltételben bármilyen két kifejezést összehasonlíthatunk (pl. az **i+1**-et a **j+5**-tel). A kifejezések összehasonlítására használhatjuk a <, >, =, <=, >=, <> (nem egyenlő) jeleket, melyek igaz/hamis (true/false) logikai értékeket adnak vissza. Ha ez a logikai érték igaz (true), akkor a parancs végrehajtódik.

Shell: if

```
if [logikai kifejezés]
  then [utasítás]
elif [logikai
  kifejezés]
  then [utasítás]
else [utasítás]
fi
```

```
if [logikai kifejezés];
then
        [utasítás]
elif [logikai kifejezés];
then
        [utasítás]
else
        [utasítás]
fi
```

if – logikai kifejezése

- A **test** parancs általában. De akár minden olyan parancs, aminek „normális” visszatérési értéke van.

```
if test -s $1
  then echo "letezik"
  else echo "nem letezik"
fi
```

Vegyük észre, ha a fájl létezik, és >0, akkor a test parancs normális visszatérési értékkel tér vissza!

test

1. Numerikus értékek tesztelése

-eq N és M értéke egyenlő

-ne N és M értéke nem egyenlő

-gt N nagyobb mint M

-lt N kisebb mint M

-ge N nagyobb vagy egyenlő mint M

-le N kisebb vagy egyenlő mint M

```
$ test 4 -gt 3
$ echo $?
```

test

2. Fájltipusok tesztelése

-**s** ellenőrzi, hogy a fájl létezik-e

-**d** ellenőrzi, hogy a fájl könyvtár-e

-**w** ellenőrzi, hogy a fájl írható-e

-**r** ellenőrzi, hogy a fájl olvasható-e

-**f** ellenőrzi, hogy az állomány egyszerű állomány-e

test

3. karakterláncok tesztelése

-z //igaz, ha a karakterlánc hossza nulla

-n //igaz, ha a karakterlánc hossza nem nulla

= //ha a két karakterlánc azonos

!= // ha a két karakterlánc nem azonos

karakterlánc // igaz, ha a karakterlánc bájtösszege nem 0

test

4. test visszatérési értéke:

0 ha a kifejezés igaz

1 ha a kifejezés hamis

>1 error hiba történt

case – általánosságban

Az **if** mellett a programozási nyelvek másik fajta elágazására szolgáló utasítása a **case**. Ez egy változó értékétől függően hajtja végre valamelyik parancsot. C formája:

```
switch (változó) {
        case 'a':
                utasítás;
                break;
        case 'd':
                utasítás;
                break;
        defualt:
                utasítás;
                break;
}
```

Shell case:

```
case string0 in
   string1)
        utasítások
        ;;
   string2)
        utasítások
        ;;
….
   *)
   utasítások
   ;;
esac
```

Shell case:

```
case "$1" in
  'start‘)
        /usr/app/startup-script
  ;;
  'stop')
        /usr/app/shutdown-script
  ;;
  'restart')
        echo "Usage: $0 [start|stop]"
  ;;
esac
```

```
case $1 in
   1) echo ”Egy”;;
   2) echo ”Kettő”;;
   3) echo ”Három”;;
   4) echo ”Négy”;;
   5) echo ”Öt”;;
   6) echo ”Hat”;;
   7) echo ”Hét”;;
   8) echo ”Nyolc”;;
   9) echo ”Kilenc”;;
   *) echo ”Ez nem egyjegyű szám”; echo ”Haver!!”;;
esac
```

Példa 1:

Megadja, hogy az argumentumban megadott állomány létezik-e, irható-e

```
if test -s $1                                   //-d, -w, -r, -f
   then echo "Az állomány létezik."
else
   echo "Nem létezik"
fi

# Megjegyzés: A ”touch” paranccsal létrehozott állományokra az else ág hajtódik végre.
```

Példa 1 #2

Megadja, hogy az argumentumban megadott állomány létezik-e, irható-e

```
if [ -s $1 ]                                    //-d, -w, -r, -f
   then echo "Az állomány létezik."
else
   echo "Nem létezik"
fi

# Megjegyzés: A ”touch” paranccsal létrehozott állományokra az else ág hajtódik végre.
```

Példa 2:

- Kiírja melyik fájlban van több karakter

```
if [´cat $1 | wc –c´ –gt ´cat $2 | wc –c´ ]; then
  echo ”$1 –ben több karakter van.”
else
    echo ”$2–ben több karakter van.”
fi

  $./proba lista1 lista2
```

Példa 3:

- Megadja, hogy az argumentumban megadott két karakterlánc azonos-e!

```
if [ $# -eq 2 ]; then
    if [ $1 = $2 ]; then
        echo "A két karakterlánc azonos"
    else
        echo "A két karakterlánc nem azonos"
    fi
        else echo "Két paramétert adj meg!"
fi
```

Példa 4:

- Megadja, hogy az argumentumban megadott két szám közül melyik a nagyobb (első, vagy második) és írja is ki!

```
if [ $1 -gt $2 ]
        then echo $1" nagyobb"
        else echo $2" nagyobb"
fi
```

Példa 5:

- A script megvizsgálja, hogy kapott-e parancssori paramétert! Ha nem kapott kiírja és ha kapott kiírja hogy hányat!

a)

```
if [ -z \$\* ]; then
   echo "Nem adtál meg parancssori paramétert„
else
   echo "A script $# darab parancssori paramétert
   kapott"
fi
```

Példa 5:

b)

```
if [ $# -eq 0 ]
    then echo "Nem adtál meg parancssori
  paramétert"
    else echo "A script $# darab parancssori
  paramétert kapott"
fi
```

c)

```
if test -z $1
      then echo "Nincs parancssori argumentum. "
      else echo "Van parancssori argumentum, de
  kit érdekel."
fi
```

Példa++

- A script osztja az argumentumban megadott harmadik számot a másodikkal.

```
echo "$3/$2="`expr $3 / $2`
```

Példa++

- Két paramétert kap a script, az egyik egy fájl neve, a másik egy karakterminta. Számolja ki, hogy a fájl hány ilyen karaktermintát tartalmaz.

```
echo "A "$1" fájlban"
echo `cat $1 | grep -c $2`
echo "darab "$2" minta található"
```

Példa++

Írj scriptet, melynek 2 argumentuma van: az első egy állománynév a második egy numerikus érték. (Ha a fájlnak kevesebb sora van mint a megadott érték, akkor írjon ki valamit!) A program írja ki a fájl sorszámadik sorát!

```
if [ -s $1 ]; then
   if [ `cat $1 | wc -l` -lt $2 ]; then
         echo "Az állomány $2-dik sora: "`cat $1 | head -$2 |
   tail -1`
      else
         echo "Túl nagy számot adtál meg."
      fi
else
   echo "Az állomány nem létezik"
fi
```

if máshogy „megfogalmazva”

logikai kifejezés && utasítás1 || utasítás2

igaz hamis

[-s $1] && echo „létezik” || echo „nem letezik”

egyéb tulajdonságokat is lehet vizsgálni (r, w, x, d)

Ciklusok: for

- Gyakran előfordul, hogy a programunkban valamit többször meg szeretnénk ismételni. Ilyenkor ciklusokat használunk. Ezeknek több fajtájuk van, most a **for** ciklussal fogunk foglalkozni, melynek a következő szerkezete van:

for *ciklusváltozó* := *kifejezés1* **to** *kifejezés2* **do**
  utasítás;

```
for (i=0; i<10; i++) {
  ...
}
```

Shell ‘for’ szerkezete

- **for** ciklusváltozó **in** lista
- **do**
- utasítás
- **done**

```
for i in `ls`
do
  echo $i
done
```

Példa 1:

Példa: az argumentumként megadott számok összegét adja vissza.
./proba 1 2 3

```
k=0
for i
do
   k="$k + $i"
   #sum=`expr $sum + $i`

done
echo `expr $k`
#echo "Az argumentumban megadott számok összege: "$sum

//ha nem adunk meg listát akkor a poz. paramétereket veszi
   automatikusan
```

- A lista helyén megadott számokat adja vissza

```
k=0
for i in 1 2 3
do
        k="$k + $i"
done
echo `expr $k`
```

Példa 2:

- Írj scriptet, mely listázza a parancssori argumentumban megadott jegyzék tartalmát és az al-jegyzékek tartalmát is.

```
[ -d $1 ] &&
for i in `ls $1`
do
        echo $i
done || echo "Nem jegyzék"
```

- Határozza meg, hogy a argumentumban kapott fájlok összesen és egyenként hány karakterből állnak. Akármennyi paraméter lehet!

```
echo
echo " fájl   karakter"
k=0
for i
do
 echo $i"  "`cat $i | wc -c`
 k="$k + `cat $i | wc -c`"
done
echo
echo "A megadott paraméterekben összesen
   "`expr $k` "karakter van."
echo
```

Példa 4:

- Határozd meg, hogy a argumentumban kapott fájlok összesen hány sorból állnak. Akármennyi paraméter lehet!

```
echo
echo " fajl    sor"
k=0
for i
do
   echo $i"  "`cat $i | wc -l`
   k="$k + `cat $i | wc -l`"
done
echo
echo "A megadott paraméterekben összesen "`expr $k` "sor van."
echo
```

Ciklusok: WHILE …DO (elöltesztelő)

- Ez a ciklus a **for** ciklushoz hasonlóan megismétel néhány parancsot többször egymás után. A különbség abban van, hogy míg a **for** ciklusnál a ciklusváltozó kezdő- és végértéke határozta meg az ismétlések számát (pl. for i:=1 to 8 ciklusnál az adott parancsot nyolcszor ismételte meg), a **while..do** ciklusnál az ismétlések számát nem egy ciklusváltozó, hanem egy feltétel határozza meg (pl. a<b). Amíg a feltétel igaz, addig az adott parancsokat a **while..do** ciklus ismételni fogja.

Magyarul: amíg a feltétel igaz, ismételd a ciklusban levő parancsokat.

A ciklus a következő képen működik:

- a számítógép megnézi, hogy a feltétel igaz-e.
- Ha nem igaz, akkor a ciklusban levő parancsokat nem hajtja végre egyszer sem, hanem a program folytatódik a ciklus utáni utasításokkal.
- Ha a feltétel igaz, végrehajtja a ciklusban levő parancsokat. Ismét megvizsgálja a feltételt, amely ha még mindig igaz, akkor ismét végrehajtja a ciklusban levő parancsokat. Majd ismét megvizsgálja a feltételt, amely ha meg mindig igaz, akkor ismét végrehajtja a parancsokat... Ha a feltétel már nem igaz, akkor a parancsokat nem hajtja végre, hanem folytatódik a program futása a ciklus utáni utasításokkal.

Klasszikus while … do

while *fetétel* **do** *parancs*;

Vagy

while *fetétel* **do** {
  parancs;
}

Shell: While ciklus

```
#!/bin/bash
COUNTER=0
while [ $COUNTER -lt 10 ]; do
   echo The counter is $COUNTER
   let COUNTER=COUNTER+1
done
```

While példa

```
#!/bin/bash
# This generates a file every 5 minutes
while true; do
  touch pic-`date +%s`.jpg
  sleep 300
done
```

While példa

```
while [ $i -le 10 ]
do
  echo $i
  i=`expr $i + 1`
done
```

Példa : végtelen ciklusban kiírni az időt

```
while true
do
  date;
  sleep 1;
done
```

While példa

Írj shell scriptet, mely kiírja, hogy ”Végtelen ciklus vagyok” másodpercenként.

```
while true
do
        sleep 1
        echo "Végtelen ciklus vagyok!"
done
```

Ciklusok: REPEAT..UNTIL (hátultesztelő)

A **repeat..until** ciklusnál a számítógép először végrehajtja a ciklusban levő parancsokat (repeat..until közötti részt), majd utána vizsgálja meg a feltételt. Ha a feltétel igaz, kilép a ciklusból. Ha a feltétel hamis, megismétli ismét a ciklusban levő parancsokat majd ismét megvizsgálja a feltételt. Ennél a ciklusnál tehát egyszer mindenképpen lefutnak a ciklusban levő parancsok.

Ciklusok: REPEAT..UNTIL (hátultesztelő)

- A **repeat..until** ciklusból a számítógép a **while..do** ciklussal ellentétben akkor lép ki, ha a feltétel igaz (a **while..do** ciklusnál akkor lépett ki, ha a feltétel hamis volt).

- Magyarul: ismételd a ciklusban levő parancsokat, amíg a feltétel nem lesz igaz (tehát amíg a feltétel hamis).

Shell: Until cilus

```
#!/bin/bash
COUNTER=20
until [ $COUNTER -lt 10 ]; do
  echo Szamlalo_erteke: $COUNTER
  let COUNTER-=1
done
```

Írj scriptet, mely kiszámolja az argumentumban megadott számok átlagát!

for ciklus

```
sum=0
for i
do
  sum=`expr $sum + $i`
done
echo `expr $sum / $#`
```

Írj scriptet, mely kiszámolja az argumentumban megadott számok átlagát!

while ciklus

```
sum=0
i=1
while  [ $i -le $# ]
do
  sum=`expr $sum + $i`
  i=`expr $i + 1`
done
echo `expr $sum / $#`
```

Írj scriptet, mely kiszámolja az argumentumban megadott számok átlagát!

until ciklus

```
sum=0
i=1
until  [ $i -gt $# ]
do
  sum=`expr $sum + $i`
  i=`expr $i + 1`
done
echo `expr $sum / $#`
```

Vezérlésátadó utasítások

- A nyelv két vezérlésátadó utasítással rendelkezik: **break** és **continue**.

- A **continue** utasítás a break utasításhoz kapcsolódik. A ciklusmagban található continue utasítás hatására azonnal (a ciklusmagból még hátralévő utasításokat figyelelmen kívül hagyva) megkezdődik a következő iterációs lépés.

Vezérlésátadó utasítások

- **Break** utasítás hatására a ciklus végrehajtása befejeződik, és a program folytatja futását a ciklus után.

Vezérlésátadó utasítások

```
for i in `seq 1 10`
do
  if [ `expr $i % 2` -eq 0 ]; then
    #Paros
    echo $i
  else
    #Paratlan
    continue
  fi
done
```

#seq 2 2 10

Vezérlésátadó utasítások

```
...
for i in „valami”
do
  while true
        do
                cmd1
                cmd2
         [ condition ] && break
        done
done
.... ..
```

Függvények

```
#!/bin/bash

HELLO=Hello

function hello {
  local HELLO=World
  echo $HELLO
}
echo $HELLO
hello
echo $HELLO
```

Függvények

```
#!/bin/bash
function quit {
  exit
}
function hello {
  echo Hello!
}
hello
quit
echo foo
```

Paraméteres függvény

```
#!/bin/bash
function quit {
  exit
}
function e {
  echo $1
}
e Hello
e World
quit
echo foo
```