Unix’i loeng 13.03.97 1.

```
vaata viimast näidet
loendi=$#
kask=echo
while true
  do
    kask="$kask \$$loendi"
    loendi=`expr $loendi - 1`
    if test $loendi -eq  0
       then break
    fi
  done
eval $kask
```

```
loendi=$#
kask=echo
while true
  do
    kask="$kask \$$loendi"
    loendi=`expr $loendi - 1`
    if test $loendi -gt  0
       then continue
    fi
    eval $kask
    exit
done
```

exit lõpetab shell-P töö

alamK-de list

```
$  ls -l /                    kõikide F-de list
$  ls -l / | grep '^d'        ainult K-d (kuna algavad ‘d‘-ga)
drwxrwxr-x  16 root     sys        3584 Mar  8 01:22 dev
drwxrwxr-x   4 root     sys         512 Mar  1 19:56 devices
…
kui tahad seda kasutada mistahes K-i jaoks:
ls -l $1 | grep '^d'          tee käsuF  ‘lsdir‘
$  lsdir /dev                 OK!
$  lsdir                      mida teeb?
1 positsiooniline parameeter; aga kui anda 2?
$  lsdir /dev /devices        väljastab ainult /dev’i listi
kui tahad mitme parameetri jaoks:
for i                         peaaegu OK! võid anda 2 või rohkem
      do                      parameetrit (väljastatakse üksteise järel)
        ls -l $i | grep '^d'
      done
```

aga mis juhtub, kui pöörduda **13.03.97 2.**

```
$  lsdir                    tsüklit ei täideta kordagi!
lisa kontroll
if test $# = 0                          for i in *
  then lsdir .                             do
  else                                        if test -d $i
    for i                                          then echo $i
      do                                        fi
        ls -l $i | grep '^d'                done
      done
fi
NB! rekursioon

if test $# = 0              mida teeb?
  then lsdir .
  else
    for i
      do
        for j in $i/*
           do
              if test -d $j
                 then echo $j
              fi
           done
      done
fi
```

olgu vaja P-i, mis väljastab kõikide K-de nimed antud alampuus
selleks tuleb minna kõikidesse K-desse, sealt edasi kõikidesse jne.
Proovi ise koostada vastav programm!

13.03.97 3.

Shell’i asendused

tehakse vastavalt prioriteedile

1) resultaadi asendus `date`
2) parameetri või muutuja asendus $loendi
3) tühikute interpreteerimine; eelmiste asenduste tulemused on seotud väljade eraldajatega; harilikult on nendeks tühikud, aga samuti TAB-märgid ja reavahetuse märgid; iga sõna, milles on eraldajad, jagatakse mitmeks; eraldajaid ignoreeritakse tekstis, mis on “” sees
4) F-nimede genereerimine; shell kontrollib iga sona *, ?, [esinemise suhtes; kui mingi sõna sisaldab neid metasümboleid, siis asendatakse ta tähestiku järjekorras F-nimede loeteluga või originaaliga, kui vastavust ei leitud

Shell’i asendused baseeruvad peamiselt teksti asendamisel

Programmeerimine Shell’is

aeglane miks?

Millal kirjutada P shell’is?

• suur operatsioonide hulk, mis on U-i käskude funktsioonid
• failioperatsioonid (otsimine, sortimine, teisendused, jne.)
• andmed on stringid või tekstifailid

mitu tarbijat on S-s

$ who | wc -l

Kas seda ülesannet saab lahendada C-s?

Milline võiks välja näha vastav P?

moodulite kompileerimine C-s 13.03.97 4.

moodulprojekteerimine

võimaldab projekteerida, välja töötada ja siluda mooduleid eraldi

C-s võimalik lähteteksti jagada erinevate F-de vahel;

samuti kompileerida F-e ükshaaval

funktsioon peab olema 1-s F-s - loogiliselt jagamatu ühik, samuti mingi struktuur

tekivad probleemid:

- funktsioonidele peavad olema teada teistes F-des olevad funktsioonid ja struktuurid
  C-s kasut. selleks spetsiaalset kirjeldust ***include***
  #include “side.h“ F-i lisatakse F-i ‘side.h‘ sisu
- mõnede konstantide väärtused peavad olema teada terves P-s
  selleks kasut. ***define***
  #define LPIKKUS 20

harilikult kõik suure P-i kirjeldused on nn. sideF-s

ridu, mis algavad #, töötleb nn. eel(pre)protsessor (teeb muud ka!)

kui tahad näha eelprotsessori töö tulemust, siis käsk

$ cc -P prog.c

töötleb F-i ‘prog.c‘ eelprotsessoriga ja paneb tulemuse F-i ‘prog.i‘

$ cc failA.c failB.c failC.c

tekitab 4 F-i: a.out failA.o failB.o failC.o (kui polnud vigu)

kui muutus ainult failA.c, siis

$ cc failA.c failB.o failC.o

NB! hoia alati vastavuses mooduli teksti ja objektmoodulit

osaline kompileerimine:

$ cc -c failA.c

kui tahad väljundF-le anda nme, parameeter -o; teatab kompilaatorile, et tulemus panna 1. pos. par. näidatud F-i (prog1):

$ cc -o prog1 failA.c failB.c failC.c

make **13.03.97** **5.**

Suured P-d:

- programmistide kollektiiv (10-d, …); töö organiseerimine
  P-de madal kvaliteet
- reeglina moodulitest, mida hoitakse eraldi failides
- nendevaheline side
- sideF-i meetod; on 1 ühine F, kus on vajalikud kirjeldused
  (konstandid, tüübid, muutujad ja lähteteksti osad)
  kui moodulis on kasutatud sideF-i, siis tema objektmoodul saadakse
  2 tekstiF-i kompileerimise tulemusel
- ümberkompileerimise sügavus sõltub sidemetest
- U-s on nn. koordinaator ***make*** moodulitevaheliste sidemete
  korrastamiseks

make on P, millele saab ette anda erinevate moodulite vahelised seoste kirjeldused mingis P-kompleksis; selle info põhjal teeb make kindlaks, et mõningad kompileeritud komponendid on vananenud
vt. pilti
makefile - sidemete kirjeldamise F, mida make kasutab vaikimisi
alam.o : alam.c side.h
makefile-i tuleb lisada U-i käsud vananenud F-i muutmise kohta
need käsud tuleb lisada selle rea järele, kus on kirjeldatud seosed
seega antud juhul:

```
alam.o : alam.c side.h
   cc -c alam.c          -c määrab, et toimub osaline kompileerimine
                         ja tulemus pannakse F-i alam.o
```

analoogiliselt:

```
pea.o : pea.c side.h
   cc -c pea.c
```

P ‘laade‘ saadakse objektmoodulitest pea.o ja alam.o :

```
laade : pea.o alam.o
   cc -o laade pea.o alam.o      -o määrab, et tulemus läheb F-i ‘laade‘
```

seega näeb Makefile välja järgmine**:** **13.03.97 6.**

```
laade    : pea.o alam.o
           cc -o laade pea.o alam.o
pea.o    : pea.c side.h
           cc -c pea.c
alam.o   : alam.c side.h
           cc -c alam.c
```

seda F-i kasutab make sidemete tabeli ja taastamisprotseduuride loomiseks vananenud F-de uuendamisel
oletame, et muutus F ‘side.h‘
andes käsu:
$ make laade
teeb make kindlaks, kas F ‘laade‘ on vananenud ja vajadusel loob temast uue versiooni; selleks avab ta F-i ‘makefile‘, teeb sidemete tabeli ja määrab, millised F-d on vananenud
vananenud on F, kui ta sõltub F-st, mida on uuendatud hiljem
praegu: alam.o, pea.o ja laade
seega täidab make käsud:
cc -c alam.c
cc -c pea.c
cc -o laade pea.o alam.o
saad F-i ‘laade ‘ uue versiooni
oluline: make teeb minimaalse arvu kompileerimisi
effekt suur siis, kui lähteF-e on palju ja sidemed on keerulised
Märkus: makefile - keeruline; lihtsustamiseks kasut. mõningaid reegleid:
1) objektmoodul sõltub lähtetekstist (koos par-ga ‘-c‘)
seega:

```
laade    : pea.o alam.o
           cc -o laade pea.o alam.o
pea.o alam.o      : side.h
```