1 Sissejuhatus

1.1 Kust mida saab ja kuidas käib

Script-Fu on Scheme-keele alamhulk, millele on lisatud rakendusprogrammi GIMP spetsiifilised graafikakäsud. Käske saab täita ükshaaval konsooliaknast (Xtns→Script-Fu→Console...) või koostada käskudest täidetavaid jadasid, nn. Script-Fu skripte. Reeglina on käsufailid (mis on tavalised notepadiga redigeeritavad tekstifailid) programmi GIMP põhikataloogi alamkataloogis `scripts` või siis kasutaja kodukataloogi GIMP kasutajakataloogi alamkataloogis `scripts`.

Käskude kohta on küllaltki ammendav info Xtns→DB Browser... all. Internetis pole just palju teavet, küll aga on skripte. Täielik dokumentatsioon keele kohta puudub. Keele Scheme kohta on küll palju raamatuid ilmunud ja veebilehekülgi kirjutatud.

1.2 Aritmeetikatehted

Script-fu notatsioon on *kõikjal prefiksne*. See tähendab, et **kõik** tehted on kujul `(tehe arg1 arg2 ... argn)`.

Avame Script-Fu konsooliakna ja liidame kaks arvu:
`(+ 2 3)`
Tulemuseks saame 5.
Teisi (aritmeetika)tehteid:

tehte süntaks	tehte tulemus
`(+ a1 a2 a3 ...)`	tagastab etteantud arvude summa
`(- a)`	tagastab etteantud arvu vastandarvu
`(- a b)`	tagastab etteantud arvude vahe
`(* a1 a2 a3 ...)`	tagastab etteantud arvude korrutise
`(/ a)`	tagastab etteantud arvu pöördarvu
`(/ a b)`	tagastab etteantud arvude jagatise
`(fmod a b)`	tagastab (täisarvude) jagamisel tekkiva jäägi
`(abs a)`	tagastab etteantud arvu absoluutväärtuse
`(max a1 a2 a3 ...)`	tagastab etteantud arvudest suurima
`(min a1 a2 a3 ...)`	tagastab etteantud arvudest vähima
`(sqrt a)`	tagastab etteantud arvu ruutjuure
`(exp a)`	tagastab suuruse e^a, kus $e = 2,718\ldots$ on naturaallogaritmide alus
`(ln a)`	tagastab etteantud arvu naturaallogaritmi
`(pow a b)`	tagastab suuruse a^b
`(sin a)`	tagastab radiaanides antud nurga siinuse
`(cos a)`	tagastab radiaanides antud nurga koosinuse
`(tan a)`	tagastab radiaanides antud nurga tangensi
`(asin a)`	tagastab etteantud arvu arkussiinuse (radiaanides)
`(acos a)`	tagastab etteantud arvu arkuskoosinuse (radiaanides)
`(atan a)`	tagastab etteantud arvu arkustangensi (radiaanides)

Ülesanne 1. Selgita välja, kas kõik tehted ikka töötavad. Kuidas tehteid järjest rakendada (näiteks leida $\frac{\sqrt{2}}{e^5} + 3 \cdot 7$). Millised kitsendused on jagamisel? ruutjuurel? logaritmil? astendamisel? arkusfunktsioonidel? (Kas need langevad kokku matemaatikatunnist teadaolevate määramispiirkondadega?)

1.3 Omistamine. Muutujate tüübid

Et väljaarvutatud suurusi edaspidiseks talletada, kasutame muutujaid. Omistame muutuja `m` väärtuseks tehte `(* + (2 3) 4)` tulemuse (s.t. arvu 20):

```
(set! m (* + (2 3) 4))
```

Anname nüüd käsu

```
m
```

süsteem vastab 20. Muutujale `m` on väärtus omistatud.

Protseduuri `set!` üldine kuju ongi:

```
(set! muutuja väärtus)
```

Muutujad võivad olla erinevat tüüpi. On ju arvegi erinevaid: täisarvud, reaalarvud jne. Scriptfus on kasutusel

1) madalamad tüübid (s.t. Scheme standardist tulenevad tüübid)
   - tühitüüp `tc_nil`
   - sümboltüüp `tc_symbol`

- arvutüüp `tc_flonum`
- sõnetüüp `tc_string`
- protseduuritüüp `tc_closure`, `tc_fsubr`, `tc_subr_1`, `tc_subr_2` jne., `tc_subr_2n`
- loenditüüp `tc_cons`
- massiivitüüp `tc_byte_array`, `tc_double_array`, `tc_lisp_array`
- failitüüp `tc_c_file`

2) kõrgemad tüübid (s.t. Script-Fus defineeritud tüübid)
   - täisarvutüüp `INT8`, `INT32`
   - reaalarvutüüp `FLOAT`
   - sõnetüüp `STRING`
   - pilditüüp `IMAGE`
   - kihitüüp `DRAWABLE`, `LAYER`
   - kanalitüüp `CHANNEL`
   - vaatetüüp `DISPLAY`
   - massiivitüüp `INT8ARRAY`, `INT32ARRAY`, `FLOATARRAY` jne.

Piirdume siin märkusega, et esialgu on meie jaoks olulised ainult kõrgemad tüübid. Tegelikult esitatakse kõik kõrgemat tüüpi suurused madalamate tüüpide abil.

Näiteks sõnetüübi kasutamiseks võime kirjutada
`(set! nimi "nipitiri")`
süsteem vastab `"nipitiri"`. Nüüd vastab süsteem samamoodi ka siis, kui kirjutame lihtsalt `nimi`

Peame meeles, et `"5"` ja `5` ei ole üks ja seesama! **Sõne on märkide järjest kirjutis** ja tal ei ole tema märkidest moodustuva (võimaliku) arvulise tähendusega (esialgu) mitte midagi pistmist. Seega, esimene väljendab ühesümbolilist sõnet, teine aga (täis)arvu.

2 Pildi ja kihi loomine, täitmine värviga. Vaate avamine

Kõigepealt valime File→New... ning avame uue pildi. Taustavärviks valime valge. Avanebki kohe aken valge taustavärviga, kusjuures parameetrid (laius, kõrgus jne.) on avatud pildil need, mis avamisdialoogis seatud olid.

Kuidas seda teha programselt? Jagame tegevuse etappideks.

1) Kõigepealt tuleb moodustada uus pilt ning salvestada see muutujasse. Pilte võib olla kolme tüüpi: RGB (s.o. naturaalvärvides, kood 0), Gray (s.o. halltoonides, kood 1) ja Indexed (s.o. ülimalt 256 värviga paletti kasutavad pildid, kood 2). Ette tuleb anda pildi laius ja kõrgus (täisarvud). Pildi moodustamise protseduur on `gimp-image-new`.

   Viime selle ellu: kirjutame konsooliaknas
   `(set! pilt (gimp-image-new 400 200 1))`

   see tähendab, omistame muutujale `pilt` väärtuseks moodustatud 400×200 pikselise halltoonides pildi.

2) Seejärel tuleb moodustada kiht ning salvestada muutujasse. Kihi moodustamisel tuleb ette anda pilt, millele ta läheb, kihi laius, kihi kõrgus, kihi tüüp (RGB koodiga 0, Gray koodiga 2, Indexed koodiga 4), kihi nimi, kihi tuunjus (läbipaistvuse määr — 0% täiesti läbipaistev, 100% täiesti läbipastmatu) ning kihtide sulandamise režiim (palju erinevaid — anname koodiks 0, s.t. normaalrežiim).

   Kirjutame siis konsooliaknasse
   `(set! kiht (gimp-layer-new (car pilt) 400 200 0 "essa" 100 0))`

   Esimene parameeter on `(car pilt)`, mitte `pilt`, seda selletõttu, et Script-Fu käsitleb kõiki suurusi loenditena. Ka muutuja `pilt` on tegelikult loend, mille esimeseks elemendiks on meid huvitav viide pildile (näiteks `(1)`). Kuna parameetriks tuleb anda suurus tüüpi `IMAGE`, mitte loend, siis peame loendist esimese elemendi kätte saama.

   * Korrektsem oleks olnud kirjutada
   ```
   (set! laius (car (gimp-image-width (car pilt))))
   (set! korgus (car (gimp-image-height (car pilt))))
   (set! tyyp (car (gimp-image-base-type (car pilt))))
   (set! kiht (gimp-layer-new (car pilt) laius korgus (* 2 tyyp) "essa" 100 0))
   ```

   Miks? Kuna me ei pruugi tegelikult teada etteantud muutujale `pilt` vastava pildi laiust, kõrgust ja tüüpi. Praegu teadsime, sest me tegime pildi ise, kuid edaspidi võib-olla on pildid mingi muu tegevuse käigus tekkinud ja nende parameetrid pole täpselt teada.

3) Kuigi kihi moodustamisel kasutasime viidet pildile, ei asu moodustatud kiht tegelikult veel pildil. Paigutame ta sinna.
   `(gimp-image-add-layer (car pilt) (car kiht) 0)`

   Kolmas parameeter tähistab, mitmendaks kihiks pildil see lisatakse. Nõutav on veel, et kihi tüüp langeks kokku pildi põhitüübiga.

4) Võime pildist kohe vaate moodustada. Selle kiht on tegelikult praegu suvalist mälus olnud sodi täis.
   `(set! vaade (gimp-display-new (car pilt)))`

   Süsteem avab uue vaate (tööakna) muutujas `pilt` oleva pildi jaoks.

5) Kiht tuleks (esialgu valge) värviga ära täita.
   `(gimp-drawable-fill (car kiht) 2)`

   Siin tähendab 2 täitmiskoodi. Kood 0 tähistab täitmist esiplaani värviga, kood 1 taustavärviga, kood 2 valge värviga ning kood 3 läbipaistvat täitmist.

Et suvalist sodi mitte vaadata, teeme täitmise ära enne vaate avamist. Seega kokkuvõttes oleme saanud järgmise programmi:

```
(set! pilt (gimp-image-new 400 200 1))
(set! laius (car (gimp-image-width (car pilt))))
(set! korgus (car (gimp-image-height (car pilt))))
(set! tyyp (car (gimp-image-base-type (car pilt))))
```

```
(set! kiht (gimp-layer-new (car pilt) laius korgus (* 2 tyyp) "essa" 100 0))
(gimp-image-add-layer (car pilt) (car kiht) 0)
(gimp-drawable-fill (car kiht) 2)
(set! vaade (gimp-display-new (car pilt)))
```

3 Skripti vormistamine ja registreerimine

Ilmselt taoline konsooliaknas käskude sisestamine pikemas perspektiivis meid ei rahulda, vajalik oleks käsud koondada kuhugi faili, kust neid siis arvuti järjest ise täidaks. Moodustame skriptikataloogi (suvalise nimega) skriptifaili, mille laiendiks on `.scm`.

Ülesanne 2. Koosta skript, mille käivitamisel küsitakse kasutajalt loodava pildi laiust, kõrgust ja taustavärvi ning avatakse seejärel vaade loodud RGB-süsteemis pildile. Pildil olgu üks kiht, mis on täidetud etteantud värviga.

3.1 Protseduur `script-fu-register`

Protseduuri süntaks on järgmine:

```
(script-fu-register p6hiprotseduur menyypunkt kirjeldus autor autorikaitse
             kuup2ev pildityyp SF1 SF2 SF3 ... SFn)
```

Esimesed seitse parameetrit on sõnetüüpi. Parameeter `p6hiprotseduur` tähistab menüüpunktist väljakutsutava protseduuri nime, `menyypunkt` tähistab menüüpunkti asukohta GIMP-menüüs. Järgmised neli parameetrit räägivad ise enda eest. `SF`-elemendid on kasutajaliidese elemendid, mille abil kasutaja saab skriptile ette anda väärtusi.

3.2 Ülesande lahendus

Järgnev skript lahendab ülesande 2.

```
(define (script-fu-valge-pilt inLaius inKorgus inVarv)
  (set! pilt (gimp-image-new inLaius inKorgus 0))
  (set! tyyp (car (gimp-image-base-type (car pilt))))
  (set! kiht (gimp-layer-new (car pilt) inLaius inKorgus (* 2 tyyp) "essa" 100 0))
  (gimp-image-add-layer (car pilt) (car kiht) 0)
  (set! vanavarv (car (gimp-palette-get-foreground)))  ;vana värv meelde
  (gimp-palette-set-background inVarv)                 ;uus värv sisse
  (gimp-drawable-fill (car kiht) 1)                    ;täidame taustaga
  (gimp-palette-set-background vanavarv)               ;vana värv tagasi
  (gimp-display-new (car pilt))
)
```

```
(script-fu-register
  "script-fu-varv-pilt"
  "<Toolbox>/Xtns/Script-Fu/Minu skriptid/Värviline pilt"
  "Teeb uue etteantud taustavärvi, laiuse ja kõrgusega pildi"
  "Nipi Tiri"
  "copyright 2003, Indrek Zolk"
  "detsember 2003"
  ""
  SF-VALUE "Pildi laius" "400"
  SF-VALUE "Pildi kõrgus" "200"
  SF-COLOR "Täitmisvärv" '(0 0 0)
)
```

4 Punkti ja pintslitõmbe joonistamine. Tsükkel

Ülesanne 3. Moodusta tühi valge pilt ja värvi sellel esiplaani värviga ühe pikseli ära. Pildi laiust ja kõrgust, samuti värvitava pikseli asukohta võiks küsida kasutajalt.

Ülesande 3 lahendamiseks tuleb

1) Moodustada uus pilt, sellele uus kiht, paigutada kiht pildile ning kiht täita valge värviga.
2) Lahutada esiplaani värv kolmeks komponendiks ning paigutada komponendid järjendisse.
3) Värvida antud koordinaatidega piksel ära.

Oletame, et registreerimisel antakse pildi laius muutujasse `inLaius`, kõrgus muutujasse `inKorgus`, piksli *x*- ja *y*-koordinaadid vastavalt muutujatesse `inX` ja `inY`. Siis pildi initsialiseerimistegevused on järgnevad:

```
(set! pilt (car (gimp-image-new inLaius inKorgus 0)))
(set! kiht (car (gimp-layer-new pilt inLaius inKorgus 0 "kiht" 100 0)))
(gimp-image-add-layer pilt kiht 0)
(gimp-drawable-fill kiht 2)
```

Paigutame esiplaani värvuse muutujasse:
`(set! esivarv (car (gimp-palette-get-foreground)))`
Käsku `(gimp-palette-get-foreground)` tasub kindlasti konsoolilt ise anda, et saada aru, millisel kujul annab süsteem värvuse tagasi. Tegemist on loendiga. Moodustame selle elementidest järjendi tähisega *esi*:

```
(set! esi (cons-array 3 'byte))
(aset esi 0 (car esivarv))
(aset esi 1 (cadr esivarv))
(aset esi 2 (caddr esivarv))
```

Et lõpuni aru saada, mis siin toimub, analüüsime kõiki käske ükshaaval. Kõigepealt luuakse kolmest baidist koosnev järjend tähisega *esi*. Järgmised kolm käsku täidavad järgemööda selle järjendi baidid. Loend *esivarv* on nimelt (ebapuhtana) kujul
`(0 (0 (255 ())))`
(juhul, kui oli tegemist sinisega), s.t. iga loendi element on kujul „mina ja ülejäänud elementide loend“. Taolise loendi esimese elemendi tagastab protseduur `car` ning ülejäänud osa protseduur `cdr`.

Nüüd on meie jaoks mõte käsul `(aset esi 0 (car esivarv))`. Järgmisena soovime saada sisuliselt `(car (cdr esivarv))`. Seda võib lühemalt kirjutada `(cadr esivarv)`. Lõpuks soovime saada sisemise osa esimest elementi, s.o. `(car (cdr (cdr esivarv)))`, sellegi võime lühemalt kirja panna kujul `(caddr esivarv)`.

Teame, et RGB-süsteemis kulub pikseli meelespidamiseks 3 baiti. Seega on suuruse *bytes-per-pixel* väärtuseks praegusel juhul 3.

Kuivõrd pikseli värvimise käsk on kujul
`(gimp-drawable-set-pixel drawable x_coord y_coord num_channels pixel)`
siis annamegi käsu
`(gimp-drawable-set-pixel kiht inX inY 3 esi)`
Kui nüüd veel anda (sõltuvalt vajadusest) üks käskudest
`(gimp-displays-flush)`
või
`(gimp-display-new pilt)`. peaksimegi nägema resultaati.

Vormista skript Script-Fu puusse ning katseta erinevate esiplaani värvidega.

Ülesanne 4. Skript küsigu kasutaja käest värvus, millega piksel joonistada. Vana esiplaani värvus tuleb meelde jätta ja pärast taastada.

Ülesanne 5. Skript ärgu joonistagu mitte pikselit, vaid paigutagu loodud kihile pintslitõmme. Pintslitõmbe tüüpi saab küsida kasutajalt, näiteks kujul

```
SF-BRUSH "pintsel" '("Circle (15)" 1.0 20 0)
```

ning seada protseduuriga `(gimp-brushes-set-brush pintslikood)`. Pintsliga joonistamiseks on protseduur `(gimp-paintbrush-default kiht 2 koordinaadid)`, kusjuures *koordinaadid* on samasugune järjend nagu ennegi, loodav käskudega

```
(set! koordinaadid (cons-array 2 'double))
(aset koordinaadid 0 x_väärtus)
(aset koordinaadid 1 y_väärtus)
```

Vana pintsel ja värv tuleks samamoodi eelnevalt meelde jätta.

Ülesanne 6. Skript küsigu kasutajalt pildi laius, samm, taustavärv ja esiplaani värv ning moodustagu animatsioon, kus piksel jookseb üle pildi. Olgu näiteks taustaks valge, esiplaaniks must, pildi laiuseks 400 ja sammuks 4. Siis tuleb moodustada 101 kihist koosnev pilt, kus esimesel kihil on ära värvitud piksel, mille x-koordinaat on 0. Teisel kihil on ära värvitud piksel, mille x-koordinaat on 4. Kolmandal kihil ära värvitud piksel, mille x-koordinaat on 8 jne. Lõpuks, 101. kihil on ära värvitud piksel, mille x-koordinaat on 400.

Pildi kõrgus valida vabalt, oma suva järgi.

Ülesande 6 lahendamiseks tuleb

1) pilt initsialiseerida
2) jätta meelde vanad värvused, seada uued
3) koostada tsükkel, mida viiakse läbi vajalik arv kordi
4) tsükli igal sammul värvida ära vajalik piksel
5) pärast tsükli lõppu taastada vanad värvused

Pildi initsialiseerimine ei valmista enam raskusi. Värvuste seadmine toimub järgmisel viisil:

```
(set! vanaesi (car (gimp-palette-get-foreground)))
(set! vanataust (car (gimp-palette-get-background)))
(gimp-palette-set-foreground inEsi)
(gimp-palette-set-background inTaust)
```

kus eeldame, et *inEsi* ja *inTaust* tulevad kasutajalt (registreerimise käigus). Pikseli värvimiseks läheb vaja esiplaani värvust järjendina: koostame selle nagu ülal tegime.

Tsükli koostame nii, et seame muutuja i kõigepealt nulliks ning hakkame talle siis järjest sammu juurde liitma. Töötame, kui tulemus pole suurem pildi laiusest:

```
(set! i 0)
(while (<= i inLaius)
  ...
  ...
  (set! i (+ i inSamm))
)
```

Näeme, et protseduuri `while` kehas olevad tegevused toimuvad korduvalt, seni kui tingimus $i \leqslant inLaius$ on täidetud. Pärast igakordset tsüklikäskude täitmist suurendame muutuja i väärtust etteantud sammu võrra.

Tsükli igal sammul tuleb luua uus kiht, täita see taustavärviga ning ära värvida vajalik piksel. Eeldame, et pildi kõrgus on 11, siis värvime kogu aeg pikseleid y-koordinaadiga 5. Tsükli kehasse paigutame seega

```
  (set! kiht (car (gimp-layer-new pilt inLaius 11 0 "kiht" 100 0)))
  (gimp-image-add-layer pilt kiht 0)
  (gimp-drawable-fill kiht 1)
  (gimp-drawable-set-pixel kiht i 5 3 esi)
```

siin eeldasime, et `esi` on esiplaani värvi põhjal koostatud baidijärjend.

Vanade värvuste taastamine pärast tsüklit ei valmista hoopiski raskusi:

```
(gimp-palette-set-foreground esivarv)
(gimp-palette-set-background taustavarv)
```

Kuna tegemist on animatsiooniga, siis peaks ta ka käima panema. Üks võimalus seda teha on menüüst, teine aga skriptist:

```
(gimp-display-new pilt)
(plug-in-animationplay 1 pilt 0)
```

Tegelikult sisaldab animatsioon palju kasutut infot. Võiks ju taustavärvi seada ainult esimesel kihil ning ülejäänud kihid organiseerida läbipaistvana. Taoliste tegevuste eest tasub lasta hoolitseda optimiseerijal.

Seega, enne animatsiooni käikulaskmist tuleks anda käsk `(set! pilt2 (car (plug-in-animationoptimize 1 pilt 0)))` ning avada näitamiseks mitte *pilt*, vaid hoopis *pilt*2.

Lõpuks võiks tulemuse konverteerida indekseeritud värvirežiimi ning salvestada GIF-vormingus. Selle viib praegu kasutaja ise läbi.

Ülesanne 7. Realiseerida pikseli värvimise asemel pintslitõmbe moodustamine. Lasta ka pildi kõrgus sisestada ning paigutada pintslitõmme püstsihis pildi keskele.

Ülesanne 8. Jätta vana piksel/pintslitõmme alles (vihje: kihtide kopeerimiseks on protseduur `(gimp-layer-copy)`).

Ülesanne 9. Realiseerida (kõigile koostatud animatsioonidele) konversioon indekseeritud värvirežiimi ning salvestamine GIF-vormingus.

Ülesanne 10. Realiseerida horisontaalliikumise asemel ringliikumine, liikumine mööda ristküliku rajajoont või veel mingi keerukam (parameetriliste võrrandite abil kirjeldatav?) liikumine. Võimaldada kasutajal laiuse ja sammu asemel sisestada laiuse ja kaadrite arvu.

5 Teksti asetamine. Taastepuhvri töö reguleerimine

Ülesanne 11. Koostada skript, mis kirjutab kasutaja poolt etteantud pildile ja kihile etteantud teksti (kirjasuuruse, -värvi ja kirja peaks kasutaja ette andma).

Teksti paigutamise protseduur `(gimp-text-fontname)` tagastab hõljuva kihi, kus paikneb tekst. Seega tuleb see kiht pärast etteantud kihile ka ankurdada (protseduur `(gimp-floating-sel-anchor)`).

Ülesannet võiks täiendada nõudega arvata kogu tegevus üheks taastepuhvri grupiks. Nimelt, kui kasutaja valib operatsiooni „Undo“, taastatakse ainult viimase operatsiooni eelne olukord. Kuna skript koosneb päris paljudest käskudest, tuleb mitu korda valida „Undo“, et saada skripti töö eelne olukord. Veel hullem on siis, kui skript sisaldab tsükleid.

Seetõttu jagatakse operatsiooonid taastepuhvri gruppideks, gruppide algusse kirjutatakse `(gimp-undo-push-group-start image)`

ning lõppu
`(gimp-undo-push-group-end image)`
kus *image* tähistab pilti, millel operatsioone läbi viiakse.

Taastepuhvri tegevuse saab ka täielikult välja lülitada, kirjutades
`(gimp-image-undo-disable image)`
ning jälle sisse lülitada, kirjutades
`(gimp-image-undo-enable image)`

Üldiselt võib kasutaja olla väga ebameeldivalt üllatunud, kui selgub, et mingi skripti töö tulemusena on taastepuhvri sisu muudetud ning endist olukorda taastada pole võimalik. Seetõttu tuleb taastepuhvri väljalülitamisse suhtuda ettevaatusega.

Esitame siin skripti ka täielikul kujul:

```
(define (script-fu-kiri inPilt inKiht inTekst inSuurus inKiri inEsivarv)
  (gimp-undo-push-group-start inPilt)
  (set! vanaesi (car (gimp-palette-get-foreground)))
  (gimp-palette-set-foreground inEsivarv)
  (set! tekstkiht (car (gimp-text-fontname inPilt inKiht 0 0 inTekst 0 TRUE inSuurus PIXE
  (gimp-floating-sel-anchor tekstkiht)
  (gimp-palette-set-foreground vanaesi)
  (gimp-undo-push-group-end inPilt)
  (gimp-displays-flush)
)

(script-fu-register
  "script-fu-kiri"
  "<Toolbox>/Xtns/Script-Fu/Minu skriptid/Kiri"
  "Paigutab näidatud kihile antud parameetritega teksti"
  "Indrek Zolk"
  "copyright 2004, Indrek Zolk"
  "jaanuar 2004"
  ""
  SF-IMAGE "pilt" 0
  SF-DRAWABLE "kiht" 0
  SF-STRING "tekst" ""
  SF-ADJUSTMENT "kirjasuurus (pikslites)" '(72 2 1000 1 10 0 1)
  SF-FONT "kiri" ""
  SF-COLOR "kirjavärv" '(0 0 0)
)
```

6 Töö märkealaga

Ülesanne 12. Koostada skript, mis võtab sisendisse pildi koos märkealaga ning moodustab sellest lähtuvalt animatsiooni, kus märkealal olev kujutis „sõidab“ vasakult paremale. Kasutaja peaks saama ette anda animatsiooni laiuse (märkeala kordsena) ning kaadrite arvu.

Enne ülesande juurde asumist tuleb välja selgitada vajalikud mõisted. Kui kasutajal oleks tarvis taolist ülesannet lahendada käsitsi, teeks ta seda lõikepuhvri abil: kopeeriks Edit-Copy abil originaalpildil märkeala alla jääva kihiosa puhvrisse ning asetaks selle Edit-Paste abil uuele pildile.

Siin tuleb silmas pidada, et märkeala ei pruugi alati olla ristkülikukujuline, küll aga eksisteerib vähim ristkülik, millesse märkeala veel ära mahub. Sellist ristkülikut nimetame märkeala **ümbriskastiks** (*bounding box*).

Käsklus Edit-Paste asetab puhvris oleva kujutise pildi keskele, moodustades selleks uue hõljuva kihi. Seega on enne mõistlik luua uus kiht, täita see valge värviga, asetada puhvrist kujutis, ankurdada tulemus kihile ja nihutada kihti vajalikus suunas.

Kirjeldame tööks vajaliku etappide kaupa, pannes kirja vajalikud käsud koos formaalsete sisendparameetritega:

1) Paigutada muutujatesse märkeala ümbriskasti laius ja kõrgus
   `(gimp-selection-bounds image) => (non_empty x1 y1 x2 y2)`

2) Luua uus pilt, kus kõrguseks on ümbriskasti kõrgus, laiuseks aga kasutaja poolt antud kordaja korda ümbriskasti laius
   `(gimp-image-new width height type) => (image)`

3) Kopeerida märkeala alla jääv kujutis aktiivselt kihilt lõikepuhvrisse
   `(gimp-image-active-drawable image) => (drawable)`
   `(gimp-edit-copy drawable) => ()`

4) Tsükkel, mis töötab kasutaja poolt antud kaadrite arv kordi
   `(while condition proc1 proc2 ... procn)`

   4.a) Luua uus kiht, paigutada pildile ja täita valge värviga
   `(gimp-layer-new image width height type name opacity mode) => (layer)`
   `(gimp-image-add-layer image layer position) => ()`
   `(gimp-drawable-fill drawable fill_type) => ()`

   4.b) Asetada lõikepuhvri sisu
   `(gimp-edit-paste drawable paste_into) => (floating_sel)`

   4.c) Ankurdada tekkinud hõljuv kiht
   `(gimp-floating-sel-anchor floating_sel) => ()`

   4.d) Nihutada kiht õigetesse koordinaatidesse
   `(gimp-layer-translate layer offx offy) => ()`

5) Optimeerida animatsioon
`(plug-in-animationoptimize run_mode image drawable) => (result)`

6) Konverteerida pilt indekseeritud režiimi
`(gimp-convert-indexed image dither_type palette_type num_cols alpha_dither remove_unused palette) => ()`

7) Salvestada GIF-vormingus animatsioonina
`(file-gif-save run_mode image drawable filename raw_filename interlace loop default_delay default_dispose) => ()`

7 Tekstifailide sisend-väljund. Tsükkel üle tekstifaili ridade

7.1 Faili avamine ja sulgemine

Töö failidega on keeles Script-Fu võrdlemisi ebamugav. Arvestatavad vahendid failidest lugemiseks ja failidesse kirjutamiseks on ainult märkhaaval.

Failist lugemiseks või faili kirjutamiseks tuleb see kõigepealt avada. Seda teeb protseduur `(fopen)`, millel on kaks parameetrit: failinimi ja avamistüüp. Protseduur tagastab failivoo, mille kaudu toimuvad kõik sisend-väljundoperatsioonid.

Näiteks faili `loetelu.txt` avamine lugemiseks toimub järgmiselt:
`(fopen "loetelu.txt" "r")`
ning faili `kiri.txt` avamine kirjutamiseks:
`(fopen "kiri.txt" "w")`
Kui avada avamistüübiga `"a"`, siis jääb failis olemasolev sisu alles. Avamistüübi `"w"` korral vana sisu hävib.

Pärast sisend-väljundoperatsioonide läbiviimist tuleb fail sulgeda. See viiakse läbi protseduuriga
`(fclose fv)`
kus *fv* on suletav failivoog.

7.2 Failist lugemine

Failist *n* märgi lugemiseks kasutatakse protseduuri `(fread n failivoog)` mis tagastab etteantud failivoost *n* märki. Näiteks käsk
`(set! a (fread 1 fv))`
loeb failivoost *fv* täpselt 1 tähemärgi ja paigutab selle muutujasse *a*.

Otstarbekam oleks, kui suudaksime lugeda failist korraga terve rea. Olgu failivooks *fv*.

Valime muutuja *line*, kuhu paigutame märkhaaval rea kuni jõuame realõpumärgini `"\n"` või kuni tähemärki enam lugeda ei õnnestu (faili lõpp):
`(set! line "")`
Loeme esimese märgi ära:

`(set! a (fread 1 fv))`
Töötada tuleb senikaua, kui *a* ei ole tühi ning kui ta erineb realõpumärgist `"\n"`. Seega on jätkamistingimuseks `(and (not (null? a)) (not (equal? a "\n")))`.

Koostame tsükli:

```
(while (and (not (null? a)) (not (equal? a "\n")))
  (set! line (string-append line a))
  (set! a (fread 1 fv))
)
```

Tsükli esimene käsk liidab tähemärgi *a* juurde failivoost loetavale reale *line*. Teine käsk loeb uue märgi.

Kasulik on kogu tegevus organiseerida protseduurina:

```
(define (readline fv)
  (set! line "")
  (set! a (fread 1 fv))
  (while (and (not (null? a)) (not (equal? a "\n")))
    (set! line (string-append line a))
    (set! a (fread 1 fv))
  )
  line
)
```

Ülaltoodus defineeritakse ühe argumendiga protseduur `readline`. Tagastatakse muutuja *line* väärtus, sest see on protseduuri viimaseks käsuks.

Nüüd piisab failist `minufail.txt` esimese rea lugemiseks anda käsud

```
(set! fv (fopen "minufail.txt" "r"))
(set! esimene (readline fv))
(fclose fv)
```

Kõigi või mingi kindla arvu ridade lugemiseks tuleb muidugi koostada tsükkel, mille kehas esineb protseduur `(readline)`.

7.3 Faili kirjutamine

Faili kirjutamiseks on protseduur
`(fwrite s fv)`
mis kirjutab sõne *s* failivoogu *fv*. Näiteks `(fwrite "Tere!" fv)`.

Pärast kirjutamisoperatsioonide läbiviimist tuleb fail kindluse kindluse mõttes „salvestada“, s.t. puhvrites olev info kirjutada reaalsele seadmele. See viiakse läbi protseduuriga
`(fflush fv)`

Muidugi toimub kirjutamine ka protseduuri (`fclose`) käigus, aga kindlam on see ise läbi viia.

7.4 Failide sisend-väljund praktilistes ülesannetes

Ülesanne 13. (Klassika) Koostada skript, millele kasutaja saab ette anda tekstifaili nime. Failis on pildifailide nimed (ilma laiendita). Teada on, et need failid on *JPG*-vormingus. Avada iga fail ning salvestada ümber *PNG*-vormingusse. Pole teada, kui palju pildifaile on.

Näiteks on tekstifail kujul

```
/home/zolki/pildid/pilt1
/home/zolki/pildid/pilt2
/home/zolki/pildid/iluspilt
```

ning eelduseks on, et kataloogis `/home/zolki/pildid` eksisteerivad failid `pilt1.jpg`, `pilt2.jpg` ning `iluspilt.jpg`, mis tuleb siis *png*-vormingusse salvestada.

Ülesande 13 lahendusskeem koosneb järgmistest etappidest:

1. Avada tekstifail
2. Tsükkel, mis töötab, kuni failist loeti mittetühi rida (otstarbekas on esimene rida enne tsüklit välja lugeda). Tsükli kehas:
   a) Loetud reale (failinimi ilma laiendita!) lisada „.jpg“.
   b) Avada saadud nimega fail.
   c) Loetud failinimele ilma laiendita lisada nüüd „.png“.
   d) Salvestada fail *png*-vormingus.
3. Sulgeda tekstifail.

Rea lugemiseks kasutame muidugi ülal defineeritud protseduuri (`readline`).

Küsimusi tekib tõenäoliselt failinimele laiendi lisamisel (punktid 2a ja 2c). Esimene neist on läbi viidav näiteks käsuga

```
(set! jpgfail (string-append (rida ".jpg")))
```

Failide avamine ja salvestamine on DB Browseris küllalt hästi kirjeldatud. Avamiseks tasub valida mitteinteraktiivne viis; mõlemad parameetritena antavad failinimed tasub anda samad.

Salvestamisel on tarvis aktiivse kihi numbrit, selle saab protseduuriga (`gimp-image-active-drawable`).

Tähele tasub panna, et GIMP küll avab pildid, ent ei moodusta nendele vaadet. Seega on töökiirus tunduvalt suurem võrreldes käsitsi salvestamisega, isegi kui inimese hiire liigutamise kiirust mitte arvesse võtta.

Ülesanne 14. Tekstifailis on *jpg*-vormingus pildifailide nimed (ilma laiendita). Kasutaja peab ette andma tekstifaili nime ning taustavärvi. Skript peab eemaldama taustavärvi (see on läbiviidav nii, et valitakse välja kõik, mis EI ole etteantud taustavärviga, kopeeritakse uuele kihile ning kustutatakse vana kiht ära). Seejärel tuleb tulemus salvestada *png*-vormingus. Nii toimida kõigi pildifailidega.

Ülesanne 15. Tekstifailis on *tiff*-vormingus must-valgete pildifailide nimed (ilma laiendita). Kasutaja peab ette andma vähenduskordaja. Skript vähendagu pilti kordajaga etteantud mastaapi, indekseerigu must-valge paletiga ning salvestagu *gif*-vormingus.

Ülesanne 16. Sama, mis ülesanne 15, ent enne vähendamist tuleb pilti udustada (näiteks Gaussian Blur (IIR) — kordajad tulgu kasutajalt) ning pärast vähendamist uuesti teravustada. Nüüd tuleb indekseerida halltoonides paletiga või salvestada näiteks *png*-vormingus.

Taoline udustamine tõstab ekraanil vaadatavate piltide kvaliteeti, juhul kui nad sisaldavad kald- või kõverjooni.

Ülesanne 17. Koosta skript, mis küsib kasutajalt failinime ning salvestab sinna aktiivse seisu (esiplaani ja taustavärv, pintsel, gradient jmt.). Kasulik oleks ka koostada teine skript, mis taolisest „seadistuste faili“ põhjal paigaldab failis olevad seadistused.

Ülesanne 18. Koosta skript, mis subtitreerib pildid. Tekstifailist tuleb ühelt realt lugeda pildifaili nimi ning teiselt pildi alla paigutatav subtiiter. Võimaldada subtiiter paigutada kas pildi pinnale või muuta lõuendi suurust (*canvas size*) ning paigutada subtiiter pildi alla.

8 Ideid arvestustööks

8.1 Funktsiooni graafiku joonistamine

Koostada skript, mille puhul kasutaja saaks funktsiooni $y = f(x)$ avaldise ette anda prefikskujul (kindla muutuva suurusega, olgu selleks näiteks x), näiteks sisestagu kasutaja `(sin (* 2 x))`, see vastaks siis funktsioonile $y = \sin 2x$. Skript moodustagu sellest lähtuvalt pilt koos teljestiku ja funktsiooni kõveraga. Ette peab saama anda pildi suurust, x- ja y-teljestiku lõppväärtust, funktsiooni joonistamise tsükli sammu ning telgede, teljeväärtuste ja funktsiooni graafiku värve (kõik eraldi).

8.2 Konverter

Koostada skript, mille puhul kasutaja saaks ette anda tekstifaili. Tekstifailis on pildifailide nimed ilma laiendita. Kasutaja peab saama dialoogiaknas anda sisendi vormingu, väljundi vormingu ning väljundi režiimi (RGB, halltoonides, indekseeritud). Indekseeritud režiimi korral tuleb võimaldada valida palett ja virvtoonimise tüüp.

8.3 Pöörlemine

Koostada skript, mis koostab animatsiooni, kus kasutaja poolt väljavalitud ala pannakse pöörlema ümber oma keskpunkti. Kaadrite arvu peab kasutaja saama ette anda. Võimaldada valida, kas programm ka salvestab faili (*gif*-vormingus) või mitte.

8.4 Sobivad mõõdud

Koostada skript, mille puhul kasutaja saaks ette anda tekstifaili. Tekstifailis on pildifailide nimed *tiff*-vormingus. Kasutaja peab saama dialoogiaknas anda ette rõht- ja püstsuunalise ääre laiuse ning pildi rõht- ja püstsuunalise üldmõõdu pikselites. Neid ei tohi ületada, pilt tuleb aga vähendada proportsionaalselt sellesse mõõtu, kusjuures üks mõõde peab langema kokku etteantud üldmõõduga. Kasutajal peab saama lasta valida, kas pilti tohib pöörata või mitte. Väljund salvestada *png*-vormingus.