Iseõppiv programm

Lahenduse selgitus loomade arvamise ja õppimise mängule

Peale programmi "`loomad`" katsetamist pidid kõik mõistma, et programmi poolt juurde õpitud loomad ja küsimused lisanduvad andmestikku. Uues mängus ei paku programm enam samas kohas eelmisel korral valesti arvatud looma, vaid esitab lisatud küsimuse ja vastavalt vastusele *(e/j)* pakub vana või siis viimati sisestatud looma. Selle tõsiasja märkamine pidi tekitama pildi andmestiku ülesehitusest – **kahendpuu**. Lisaks tuleb meeles pidada, et kõige tüüpilisem kahendpuus liikumine on ühe **tee** *(path)* **läbimine** – alustatakse juurest ja jõudes leheni.

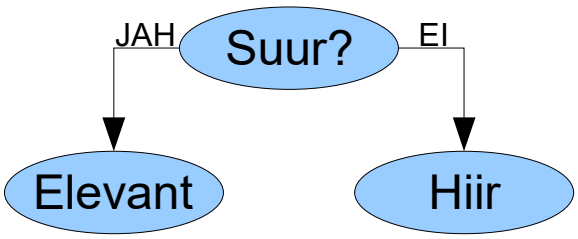
Võrdlus morsepuuga – ka siin läbiti tee puu juurest alates, kuid see ei pidanud tingimata lõppema lehel, vaid hoopis siis, kui tähe kood otsa sai.

Andmestik

Enne programmi kirjutamist peab olema selge, milline on andmestik. Kuni programmeerite kirjutage endale hästi tugevalt kõrva taha:

Tark andmestiku ülesehitus tagab mõistliku ja lihtsa algoritmi / programmi.

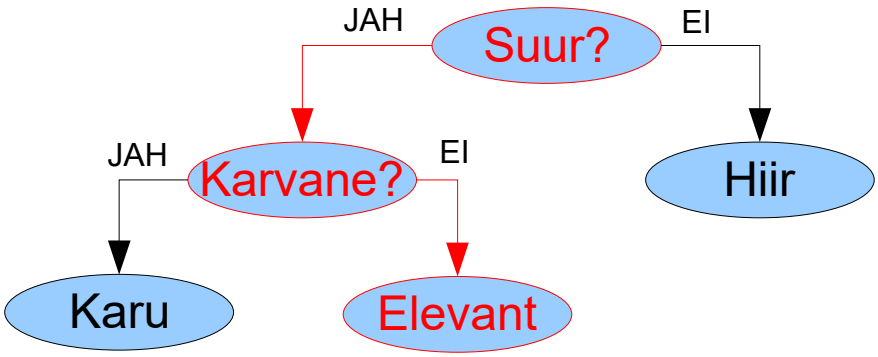
Mis on olemas alguses? Üks küsimus, kaks looma. **Jah**-poolel on suur looma ja **ei** poolel väike loom. Järgnevalt (vt Joonis 1) on kujutatud puu mängu alguses.



Joonis 1: Algne loomapuu

Milline on puu peale ühe suure ja karvase looma (karu) lisamist? Vaata joonist järgmisel leheküljel (Joonis 2).

Kujuta nüüd ette, kuidas toimub looma arvamine. Iga mängu lähtekohaks on puu juur. Oletame, et Sa mõtled loomaks elevandi. 1. küsimus: *„Kas ta on suur?“* Ilmselt on Sinu vastus *jah*. Puus tuleb arvamise jätkamiseks liikuda JAH-suunas. Järgmises puu tipus on küsimus *„Kas ta on karvane?“* Elevant eriti ei ole. Seega on Sinu vastus *ei*. Puus tuleb liikuda EI-suunas ja sellega jõuad puu leheni, kus on kirjas *“Elevant”*. Puus liikumisel on leheni jõudmine märgiks sellest, et kõik täpsustavad küsimused on otsa saanud ning aeg on pakkuda looma. Kui loom on õige, on programmil põhjust rõõmustamiseks. Joonisel (Joonis 2) on läbitud puu sõlmed ja kaared märgitud punase värviga.



Joonis 2: Puu peale karu lisamist, punased on elevandi arvamiseks läbitud tipud

Mõtle kirjelduse peale seni, kuni hakkad aru saama ;)

Jätkame ...

Kui mõeldud loom ei olnud elevant, vaid hoopis jõehobu, kelle kohta nimetatud 2 küsimust võiksid olla samade vastustega, siis tuleb programmi õpetada ja lisada uus loom ning eristav küsimus. Puusse tuleb lisada kaks uut sõlme ja need paigutatakse viimati pakutud looma-sõlmele (elevant) lehtedeks. Millisesse sõlme milline info kirjutada tuleb? Elevandi kohale läheb uus küsimus ja lisatud lehtedesse elevant ja jõehobu.

Loodan, et loogilises plaanis on asi selge. Kui ei, siis mõtle edasi.

Realisatsioonist

Puu sõlme kirjelduse leiab puu materjalist ja morse näitest. Morsepuu eeskujul võiksid viidaväljad nimetada 'yes' ja 'no', et programmi kirjutades vähem segadust oleks.

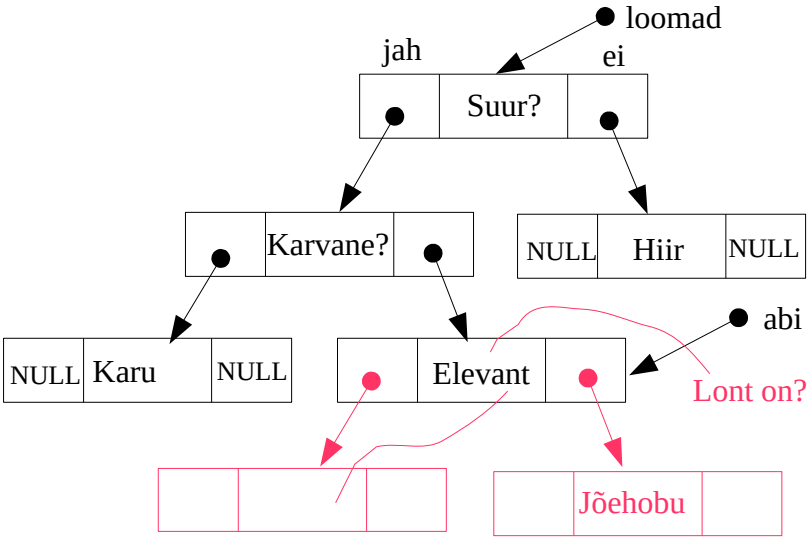
Esimese sammuna tuleks valmis ehitada esialgne puu (nagu Joonis 1). Seda on kõige lihtsam teha nn jõumeetodil. Tee valmis kolm sõlme, pannes aadressid kolme erinevasse aadressitüüpi muutujasse (funktsioon `malloc()`). Seejärel ühenda tipud puukujuliselt viitadega. Pane sõlmedesse vajalikud tekstid (NB! Stringide omistamiseks oli vaja kasutada funktsiooni `strcpy()`). Ära unusta lehtede viidaväljadesse `NULL`-e kirjutamast. Selle sammu läbimiseks võiks olla abi näitest *"Ahel kirvemeetodil"* (`linked_1.c`).

Teiseks sammuks on arvamise tsükkel, mis peab olema nii universaalne, et temaga saab ükskõik kui kõrges puus arvata. Selleks:

1. Jälgi, et viit puu juurele säilitaks oma algse väärtuse. Selleks võta kasutusele mingi abimuutuja, mis on puu juure viidaga ühte andmetüüpi. Selle muutuja abil läbitakse järjest puu tippe – eeskujudeks on morse dekodeerimise ja ühe viidaga ahela läbimine. Abiviida poolt viidatus tipu järgi esita küsimusi ja lõpuks paku looma. Abiviit läbib 2. pildil joonistatud punase tee.
2. Tsükli tüübiks sobib WHILE, sest me ei tea ette, mitu küsimust leheni/loomani viib.
3. Tsükli sisuks on eristavad küsimused, tsükkel lõpeta loomani/leheni jõudmisel, tsükli üks kordus küsib ühe küsimuse ja vastavalt saadud vastusele liigutab abimuutujat edasi kas *yes*- või *no*-suunas.

4. Peale tsükli lõppemist paku loom, rõõmusta, kui see oli õige või täienda, kui vale.
5. Täiendamiseks tuleb korralikult viitadele mõtelda, nagu seda esimest puukest luues tegid. Ja loomulikult sellele, millisesse puu sõlme milline tekst panna tuleb. Et abiviit (`abi`) on pakutud loomaga tipu küljes, on võimalik selle suhtes nii uusi tippe lisada kui ka nendes olevate tekstidega tegeleda. Vaata allpool skeemi (Joonis 3): lisatavad sõlmed ja tekstid on punased, samuti on punase joonega näidatud, kuhu tekstid ümberomistatakse.

Kasutajalt *"jah"* ja *"ei"* vastuseid küsides võiks piirduda *'j'* ja *'e'*-ga ning kasutada `char`-tüüpi muutujat, mida saab tavapärasel viisil võrrelda.



Joonis 3: Uue looma ja küsimuse lisamine

Stringide ja märkide (char) sisestamisest

Märgitüüpi muutujate sisselugemiseks sobib funktsioon `scanf()` sobiliku formaadiga. Või funktsioon `getchar()`. Funktsioon `getchar()` loeb infot alati standardsisendist (meie jaoks siis klaviatuurilt). Mõlema juhul tuleb lisada veel üks funktsiooni `getchar()` väljakutse, et klaviatuuripuhvrist reavahetuse märk välja lugeda

Kuidas terveid lauseid stringidena sisse lugeda?

`scanf()`-funktsiooni häda on selles, et ta lõpetab puhvri lugemise esimese tühiku kohal, seega terve küsimuse lugemiseks ta ei sobi.

`gets()`-i on peetud pikka aega sobilikuks funktsiooniks, et stringe klaviatuurilt sisse lugeda. Kuid viimasel ajal on ta kuulutatud ebaturvaliseks, kuna sellega on võimalik üle deklareeritud stringi (char-ide massiivi) serva kirjutada. Ja selline olukord võib põhjustada ettenägematuid vigu. Viimastest ANSI C standardist on see funktsioon eemaldatud. Sõltub kompilaatori seadetest, kas lubatakse teda kasutada või mitte.

Praeguse seisuga tuleks kasutada klaviatuurilt lugemiseks sama funktsiooni nagu failistki lugemisel:
`fgets(puhver, i, sisendfail);`

`puhver` - muutuja, kuhu sisend loetakse

`i` - täisarv, mis ütleb, mitu märki maksimaalselt loetakse, peab sobima muutuja pikkusega, tegelikult loetakse `i - 1` märki!!

`sisendfail` - klaviatuurilt lugemiseks `stdin` (standardinput)

`fgets()` loeb seni, kuni `i - 1` märki täis saab või kuni reavahetuseni (mis varem saabub). Stringi lisatakse ka reavahetuse märk (kui lugemise lõpetas reavahetus). Kui me ei taha, et ta seal on, siis peame reavahetuse kuidagi likvideerima.

Selleks saab määrata stringi pikkuse ning viimane märk stringi lõpetava `'\0'`-ga üle kirjutada.

Vaata näidet *„Stringide ja märkide lugemisest”* `stringide_ja_m2rkide_lugemine.c`.

Ja lõpuks: näidete viite alt leiad “Funktsioonid loomade arvamise jaoks “, kus on hulk funktsioone antud ülesande lahendamiseks.