TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Kaarel Kangro

Koondhinnete ning hinnetetabelite moodustamise keel DeepMOOC platvormile

Bakalaureusetöö (9 EAP)

Juhendaja(d): Tõnis Hendrik Hlebnikov
Ahti Põder

Tartu 2022

Koondhinnete ning hinnetetabelite moodustamise keel DeepMOOC platvormile

Lühikokkuvõte:

DeepMOOC on arendatav automaattestimisplatvorm programmeerimisalaste ainete läbiviimiseks. Käesoleva töö raames loodi sellele platvormile lahendus koondhinnete ning hinnetetabelite moodustamiseks uue programmeerimiskeele vormis, mille interpretaator on implementeeritud Go keeles. Töös on kirjeldatud selle keele grammatika ning funktsionaalsus. Samuti on testitud interpretaatori töökiirust erinevate andmemahtude ja hindamisskeemide korral.

Võtmesõnad: takrvaraarendus, tarkvaratehnika, interpretaatorid

CERCS: P175 Informaatika, süsteemiteooria

A Language for Forming Composite Grades and Grade Tables for the DeepMOOC Platform

Abstract:

DeepMOOC is an automated testing platform for programming related courses that is currently being developed. As part of this thesis, a solution for forming composite grades and grade tables for this platform has been developed in the form of a new programming language with an interpreter implemented in the Go language. The grammar and functionality of the created language have been described in this work. Additionally, the performance of the interpreter has been tested for different data volumes and grading schemes.

Keywords: software development, software engineering, interpretators

CERCS: P175 Informatics, systems theory

Sisukord

Sissejuhatus

Arvutipõhiste ülesannete eeliseks on võimalus anda automatiseeritud tagasisidet. Samuti on võimalik nendega suurendada ülesannete lahendamise mängulisust, pakkudes võimalust võrrelda erinevate lahenduste tulemusi, sealhulgas näiteks kiirust ja täpsust.

Uuringutes [1,2] on leidnud, et kiire automaatse tagasiside andmine parandab õppetulemusi ning suurendab õpilaste motivatsiooni. Neid tulemusi toetavad ka käesoleva lõputöö juhendajate kogemused nende läbiviidavates ainetes.

Eelnevat silmas pidades algatati 2021. aastal projekt DeepMOOC, mille eesmärk on luua paindlik tagasisideplatvorm programmeerimisalaste ainete läbiviimiseks.

Käesolev bakalaureusetöö on seotud DeepMOOC platvormi hindamiskomponendiga, ning selle eesmärk on pakkuda lahendus erinevatele ülesannetegruppidele koondhinnete määramiseks ning hinnetetabeli moodustamiseks.

Selle töö ülesehitus on jaotatud neljaks peatükiks. Esimeses peatükis tutvustatakse ülesande olemust ning vaadatakse üle mõningad alternatiivsed lahendused. Sellele järgnevas peatükis kirjeldatakse tööprotsessis kasutatud vahendeid. Kolmas peatükk on pühendatud loodud lahenduse tutvustamisele. Viimases peatükis analüüsitakse töö tulemust, tuuakse esile töö käigus ilmnenud probleemid ning nende lahendused ja arutletakse edasiste arendusvõimaluste üle.

1 Ülesande püstitus

Selles peatükis tutvustatakse nõudeid loodavale hinnete arvutamise vahendile, tuuakse esile mõne võimaliku lahendusvõimaluse eelised ja puudujäägid ning lõpuks tutvustatakse kavandatud lahendust.

1.1 Nõuded

Kuna eesmärk on luua võimalus üksikute ülesannete info põhjal kokkuvõttev andmekogum, mille põhjal oleks võimalik kasutajatele näidata hinnetetabelit, siis peab loodav lahendus väljastama andmed selleks sobival kujul.

Samuti peaks loodud lahendus liidestuma võimalikult lihtsalt teiste DeepMOOC platvormi osadega ning selle kasutamine võiks olla aktsepteeritava turvalisusetasemega.

Loodav lahendus peaks olema kasutajatele võimalikult kergesti mõistetav ning mitte nõudma teadmisi andmevahetusstruktuuride detailidest.

Lisaks kõigele eelnevale peaks lahendus suutma aktsepteeritava kiirusega toime tulla oodatavate andmemahtudega.

1.2 Vaadeldud alternatiivid

Selles peatükis antakse ülevaade võimalikest lahendustest, mis ei olnud piisavad kõigi vajaduste täitmiseks, kuid siiski väärivad esiletoomist.

1.2.1 Hindamisfunktsioon Pythonis või mõnes teises tuntud programmeerimiskeeles

Üks võimalik lahendus oleks lasta õppejõududel kirjutada hindamisfunktsioon mõnes olemasolevas programmeerimiskeeles, näiteks Pythonis[1]. Sarnast lahendust pakub üksikute ülesannete hindamiseks näiteks Moodle pistikprogramm Virtual Programming Lab ehk VPL[2], mis võimaldab nende tavalise hindamislahenduse asemel jooksutada käsurea skripti abil mistahes faili, ning ootab väljundit VPL hindamisfunktsioonile arusaadaval kujul [3].

Sellise lahenduse eelis oleks see, et kasutaja saaks kasutada laialt tuntud ja hästi dokumenteeritud programmeerimiskeelt, millega ta võib juba varem tuttav olla ning mis pakub väga laia funktsionaalsust.

See lähenemine aga ei kataks hästi turvalisuse vajadust, kuna enamik levinud programmeerimiskeeli pakub mooduseid ligipääsemiseks operatsiooni- ja failisüsteemile. Sellega seonduvate probleemide vältimiseks oleks vaja hindamisfunktsiooni jooksutada isoleeritud keskkonnas, kus see ei saa kahju tekitada, näiteks Dockeri[3] konteineris, mis aga tähendaks suurt vaeva ülejäänud DeepMOOC platvormi osadega liidestumiseks.

Samuti oleks sellisel juhul vaja kasutajal muretseda selle eest, et väljund oleks DeepMOOC platvormi jaoks sobival kujul, mis tooks kaasa piiranguid süsteemi edasisel arendamisel.

Uue programmeerimiskeele loomise eelised võrreldes laialdaselt kasutavate keeltega on:

- Keelega on võimalik teha ainult operatsioone, mida keele loomisel on otsustatud võimaldada, seega pole ohtu, et kasutaja pääseb ligi operatsioonisüsteemile, kui seda pole tahetud.

[1] https://www.python.org
[2] https://vpl.dis.ulpgc.es
[3] https://www.docker.com/

- Keelde on sisse ehitatud väljundi andmine serverile kasutataval kujul, ilma et kasutaja peaks sellele liigselt mõtlema.

1.2.2 Hindamisvalem

Probleemi saaks üritada lahendada, kasutades tulemuse moodustamiseks kasutaja poolt kirjutatud hindamisvalemit, mis sarnaneks näites Exceli tarkvaras kasutatavate valemitega. Sellist varianti kasutab Moodle oma hinnetetabelis ülesannete gruppide hindamiseks [4].

Sellise lahenduse eelis oleks samuti kasutajatele juba tuttava variandi kasutamine, eriti kuna DeepMOOCi abil õppetöö läbiviijad on suure tõenäosusega juba varem pidanud Moodle hindamisvalemeid kasutama.

Samuti täidaks hindamisvalemid turvalisusevajaduse, kuna nad üldjuhul ei võimalda ligipääsu operatsioonisüsteemile. Samuti oleks võimalikud erinevad väljundid piiratud.

Selle lahenduse puuduseks on see, et valemid on mõeldud lihtsate arvujadadega töötamiseks, kuid DeepMOOC andmed on keerukamad, et võimaldada kasutajale ligipääs ka iga ülesande eri hindamismoodulite tulemustele. Selle probleemi lahendamiseks peaks välja mõtlema mingi viisi andmete teisendamiseks valemitega kasutamiseks sobivale kujule, mis oleks aga tõenäoliselt küllaltki keerukas.

Lisaks võivad sellised valemid minna väga pikaks, mille tagajärg on halvenenud loetavus. See võib tekitada raskuseid tulevikus valemite muutmisel, kuna on raske aru saada mida täpselt tehtud on.

1.3 Kavandatud lahendus

Kõike eelnevat arvesse võttes otsustati luua täiesti uus programmeerimiskeel, mis oleks peamiselt mõeldud ainult selle platvormiga kasutamiseks. Selles peatükis tuuakse välja, kuidas see lahendus rahuldab esile toodud vajadusi ning milliseid andmestruktuure see kasutaks.

1.3.1 Vajaduste täitmine

Programmeerimiskeele loomisel on tarvilik tagada, et selle väljund tuleb alati kindla struktuuriga. See võimaldaks täita vajadust väljastada andmed sobival kujul hinnetetabeli moodustamiseks.

Turvalisusevajaduse täidab see, et uuel keelel oleks olemas ainult see funktsionaalsus, mida on probleemi lahendamiseks vaja. Kuna pole vajadust failidele ligipääsemise või operatsioonisüsteemi käskude täitmise järgi, siis loodavale keelele lihtsalt ei lisataks võimalust neid operatsioone teha.

Lihtsa liidestumise ülejäänud platvormi teenustega on võimalik tagada, kui luua uue keele interpretaator samas keeles, milles on ülejäänud platvormi teenused. Nii on võimalik vältida probleeme, mis tekiks eri programmeerimiskeeltes olevate rakenduste vahel suhtlemisega.

Lihtne mõistetavus on tagatav, kuna keelt saab luua kasutajamugavust silmas pidades, vältides näiteks probleeme kindlate andmetüüpide pärast muretsemisega. Samuti aitaks programmeerimiskeelel muutujate kasutamisvõimalus ning kommentaaride võimaldamine parandada hinnete arvutamise skeemi loetavust, mis oleks eelis võrreldes lihtsalt hindamisvalemi tõlgendaja loomisega.

Kuna keele kasutusel oleks selgelt defineeritud sisendid ja funktsioonide väljundid, siis ei peaks kasutaja teadma andmevahetusstruktuuride kuju – sellega tegelemine oleks keele interpretaatori ülesanne.

Kiiruse vajadust peaks arendamisel silmas pidama, kuid seda on võimalik testida ning keele implementatsiooni vajadusel parandada, kui peaks juhtuma, et sellega probleeme tekib.

1.3.2 Andmestruktuurid

Kuna väljundist peaks olema võimalik luua hinnetetabelit, siis see peaks sisaldama infot eri testigruppide kohta, ning igal grupil peaks olema omakorda võimalik sisaldada alamgruppe. Seda arvesse võttes leiti, et sobiv andmestruktuur oleks selline, mis sisaldab grupi nime, hinnet ja loetelu alamgruppidest, mis on kõik samasugusel kujul.

Sisendi osas oli keeruline otsust teha, kuna platvorm on alles arendamisel ning see pole veel täiesti kindlalt paigas, millisel kujul andmeid oodata saab. Tehtud on aga eeldus, et ülesannete andmed on võimalik viia kujule, mis sisaldab ülesande nime, identifikaatorit (näites numbrilist koodi), ülesande hinnet ning järjendit selle ülesande hindamisel kasutatud hindamismoodulite tulemustest.

2 Kasutatud vahendid

Selles peatükis antakse ülevaade arendusprotsessi käigus kasutatud töövahenditest.

2.1 Go

Go[4] on Google'i arendatud programmeerimiskeel. Kuna ülejäänud DeepMOOC platvormi tagakomponendi (ingl *back-end*) osad on loodud just seda keelt kasutades, valiti Go ka selle lõputöö loomise vahendiks. Selle eelis on lihtsam suhtlus teiste tagakomponendi osadega.

Järgnevalt on välja toodud töö loomise protsessi jaoks kõige olulisemad Go standardteegi paketid:

- **ast** [5] pakett deklareerib andmetüübid süntaksipuude esitamiseks;
- **reflect** [6] pakett võimaldab Go programmil toimetada suvaliste andmetüüpidega. Selle töö jaoks olulisimad võimalused, mida see pakett pakkus, olid objektide funktsioonide ja väljade leidmine tekstikujul nime järgi ning funktsioonide kutsumine suvalist tüüpi andmetega;
- **constant** [7] pakett implementeerib tüüpideta Go konstantide vahelised operatsioonid. Selle töö jaoks olulisimad funktsioonid, mida see pakett pakkus, olid konstantide väärtuste loomine tekstilisest väärtusest ning väärtuste vaheliste kahendoperatsioonide teostamine (näiteks korrutamine ja võrdsusekontroll) tehtemärgi põhjal.

Töö tegemisel otsustati kasutada võimalikult vähe Go standardteegi väliseid pakette, et tulevikus oleks võimalikult lihtne lahendust täiendada. Samuti aitab selline otsus vältida olukorda, kus välise sõltuvuse seis takistab arendust.

2.2 Arenduskeskkond

Kasutatavaks arenduskeskkonnaks valiti Intellij IDEA Ultimate[5], mis on küll peamiselt mõeldud JVM keeltes arendamiseks, aga pakub ka samaväärset tuge sama firma poolt loodud Go arendamiseks mõeldud kekskonnaga GoLand [8]. Valiku peamiseks põhjuseks oli autori pikaajaline kogemus selle tööriistaga.

2.3 ANTLR v4

ANTLR (Another Tool for Language Recognition) [9] on tööriist, mis võimaldab formaalse keelekirjelduse ehk grammatika põhjal luua programmi, mis tõlgendab struktureeritud tekstilise sisendi parsimispuuks ning genereerib selle puu läbimiseks kasutatava koodi [10]. ANTLR toetab sihtkeelena Go'd [11], mistõttu see sobis käesoleva töö raames kasutamiseks.

ANTLR tööriista kasutamise eelis on, et selle tööriista jaoks loodav grammatikafail on lihtsasti mõistetav ning seda on tulevikus kergem vajadusel muuta ja edasi arendada nii lekseri kui ka parseri muutmise asemel.

[4] https://go.dev/
[5] https://www.jetbrains.com/idea/

3 DeepMOOC platvormi lahenduste hindamise keel

Selles peatükis käsitletakse lõputöö raames valminud tarkvara ülesehitust. Täpsemalt järgnevas kirjeldame hindamiskeele grammatikat ja selle rakendamise funktsionaalsust ja tööpõhimõtteid. Valminud lahenduse lähtekood on leitav lisas I.

3.1 Grammatika

Grammatika on loodud ANTLR tööriista kasutades ning selle definitsioon on leitav lisa I failis „Grade.g4“. Selles peatükis tutvustatakse lähemalt loodud keele grammatilisi konstruktsioone.

3.1.1 Kommentaarid

Selleks, et võimaldada koodi loetavust täiendada, on lubatud kasutada kommentaare. Kommentaarid võivad olla kas reakommentaarid, mille puhul eelneb kommentaarile kaks kaldkriipsu (`//`) ning kommentaari lõpetab rea lõpp, või plokikommentaarid, mis on tekst tähiste `/*` ja `*/` vahel ning võib sisaldada mitut rida.

```
kood // Realõpukommentaar
veel koodi

/*
    Mitmerealine kommentaar
    mis ei hooli reavahetustest
*/
veel koodi
```

Joonis 1. Kommentaarid.

Näiteid kommentaaridest on näha joonisel 1.

3.1.2 Literaalid

Loodud keeles on lubatud viit tüüpi literaalid:

- tekstiliteraal, mis on suvaline tekst jutumärkide vahel;
- täisarvliteraal, mis on lihtsalt täisarv;
- ujukomaarvliteraal, mis on lihtsalt komakohtadega arv;
- tõeväärtusliteraal, mis on väärtus `true` või `false`;
- järjendiliteraal, mis on nurksulgude vahel olev komaga eraldatud kogumik väärtuseid.

```
"tekst" // Tekstiliteraal
1 // Täisarvliteraal
5.4 // Ujukomaarvliteraal
true // Tõeväärtusliteraal
[ false, 5 + 9.7, "tekst"] // Järjendiliteraal
```

Joonis 2. Literaalid.

Näiteid neist on näha joonisel 2.

3.1.3 Muutujad

Loodud keeles on lubatud kasutada muutujaid, mille nimed võivad koosneda ainult ladina tähestiku tähtedest, numbritest ja allkriipsust, kusjuures muutuja ei tohi alata allkriipsu või

numbriga. Muutujale määratakse väärtus süntaksit `muutuja nimi = väärtus kasutades` ning peale seda on võimalik seda muutujat kasutada väärtuse asemel. Juba olemasoleva nimega muutujale on lubatud määrata uus väärtus.

```
1muutuja = "tekst" // Mittesobiv muutuja nimi
_muutuja = 7.4 // Samuti mittesobiv muutuja nimi

muutuja_1 = 1 // Sobiv muutuja nimi
muutuja2 = 5.6 // Samuti sobiv muutuja nimi

muutuja_1 + 2 // väärtus 3

muutuja_1 = 5 + 6 // muutuja väärtuse ülekirjutamine
muutuja_1 - muutuja2 // väärtus 5.4
```

Joonis 3. Muutujate nimed ja kasutamine

Näiteid sobivatest ja mittesobivatest muutujatest saab näha joonisel 3.

3.1.4 Tehted

Keeles on lubatud kaks monaadtehet, 13 binaartehet ja üks ternaartehe.

Lubatud monaadtehted on:

- loogiline eitus (`!`), mis muudab tõeväärtuse vastupidiseks (ehk `true -> false` ja `false -> true`);
- unaarne miinus (−), mis muudab arvväärtuse märgi vastupidiseks.

Lubatud binaartehted on:

- liitmine (+);
- lahutamine (−);
- korrutamine (*);
- jagamine (/);
- jäägi leidmine (`%`);
- loogiline JA (`&&`);
- loogiline VÕI (`||`);
- võrdsus (==);
- mittevõrdsus (`!=`);
- suurem kui (>);
- väiksem kui (<);
- suurem kui või võrdne (>=);
- väiksem kui või võrdne (<=).

```
// Monaadtehted
!true // false
-1 // -1

// Binaartehted
2+5.7 // 7.7
7.4 - 0.3 // 7.1
2 * 5.7 // 11.4
9 / 3 // 3
6 % 4 // 2
true && false // false
false || true // true
3 == 3 // true
1 != 1 // false
4 > 3 // true
4 < 4 // false
2 >= 2 // true
5 <= 3 // false

// Ternaartehe
5 > 6 ? 3 : 9 // 9
```

Joonis 4. Tehted.

Lubatud ternaartehe on tinglik ternaartehe kujul `tõeväärtus ? väärtus kui tõene : väärtus kui väär`.

Näited neist tehetest on joonisel 4.

3.1.5 If ja else

Keeles on lubatud `if` käsk, mis koosneb tingimusest ja käskude kogumikust (mis on ümbritsetud sümbolitega `{` ja `}`), mida täidetakse juhul kui tingimuse väärtustus annab tõeväärtuse `true`. Sellele käsule võib vahetult järgneda `else` käsk koos uue `if` käsu või käskude kogumikuga, mida täidetakse ainult juhul kui eelneva `if` käsu tingimuse väärtus ei olnud `true`.

```
muutuja = 0
if (1 > 2) { // Tingimus on väär, plokis olevaid käske ei täideta
    muutuja = 1
}
// muutuja väärtus on 0

muutuja = 0
if (1 > 2) { // tingimus on väär, täidetakse else plokis olevad käsud
    muutuja = 1
} else {
    muutuja = 2
}
// muutuja väärtus on 2

muutuja = 0
if (1 > 2) { // tingimus on väär, vaadatakse else if tingimust
    muutuja = 1
} else if (1 < 2) { // tingimus on tõene, täidetakse selles plokis olevad käsud
    muutuja = 2
} else { // eelnev if tingimus oli tõene, seda plokki ei vaadata
    muutuja = 3
}
// muutuja väärtus on 2
```

Joonis 5. `If` ja `else` käsud.

Näide neist on näha joonisel 5.

3.1.6 Funktsioonid

Kõik keeles kasutatavad funktsioonid on mõeldud rakenduma vähemalt ühele väärtusele, seega iga funktsiooni süntaks eeldab kuju `väärtus.funktsioon(argumendid)`. Lisaks sellele on palju kasutusel lambdafunktsioonid listide haldamiseks, mille kuju sarnaneb Kotlini[6] lambdafunktsioonidele [12].

[6] https://kotlinlang.org/

```
midagi.funktsioon() // Ühe argumendiga funktsioon,  näiteks avg
// Mitme argumendiga funktsioon
midagi.argumentidegaFunktsioon(argument1, argument2)

// ühe argumendiga lambdafunktsioon ilma argumenti ümber nimetamata, näiteks map
midagi.lambdaFunktsioon {
    kood
}

// ühe argumendiga lambdafunktsioon argumendi ümber nimetamisega, näiteks map
midagi.lambdaFunktsioon(argumendiNimi) {
    kood
}

// mitme argumendiga lambdafunktsioon, näiteks max
midagi.lambdaFunktsioon(argumendiNimi1, argumendiNimi2) {
    kood
}
```

Joonis 6. Funktsioonid.

Näiteid erinevatest funktsioonidest on näha joonisel 6.

3.1.7 Grupeerimiskäsklus

Kuna eesmärk oli keele väljund saada kujul, mille põhjal hinnetetabelit luua, siis oli vaja luua võimalus testide gruppide moodustamiseks ja hindamiseks. Selleks sai keelde loodud käsklus kujul `group muutujanimi = („Grupi ilus nimi“, testid) { käsklused grupi hinde moodustamiseks“ }`.

```
group kt1 = ("Kodutöö 1", testid) {
    hindamiskood
}
```

Joonis 7. Grupeerimiskäsklus.

Näide sellest on joonisel 7.

3.2 Funktsionaalsus

Eelmine peatükk kirjeldas loodud keele struktuuri, järgnevalt aga käsitletakse koodi töötlemisel tehtavaid taustategevusi ning selles keeles kasutatavaid funktsioone.

3.2.1 Andmete eeltöötlus

Go keeles on tüüp `any` ehk tühi liides, mis võib hoida mistahes väärtust [13] ning seega järjend väärtustest tüübiga `any` on järjend mis tahes väärtustest. Go ei tee taustal teisendust kindla tüübiga järjendi ja järjendi tüübiga `any` vahel, seega on vaja enne hindamisfunktsiooni käivitamist kõik järjendid sisendandmetes teisendada ümber järjenditeks tüübiga `any`.

Teisendatud andmed sisestatakse interpretaatori andmetesse konstandina nimega `tests`. Lisaks sellele lisatakse interpretaatori andmetesse et parajasti läbitav funktsioon peaks tagastama väärtuse grupiformaadis, mis on toodud välja peatükis 1.3.2 ning selle grupi nimi.

3.2.2 Väljundi teisendamine soovitud kujule

Iga funktsiooni tagastushekel kontrollib interpretaator, kas parasjagu vaadeldava skoobi infos on määratud, et tagastus on vaja teisendada grupiformaati. Kui jah, siis interpretaator

tagastab saadud tagastusväärtuse asemel (mis peaks olema ujukomaarv) hoopis grupiformaadis väärtuse, millel on info vaadeldavas skoobis olnud testide, soovitud grupi nime, ning skoobis loodud alamgruppide kohta ning mille punktide väli saab väärtuseks saadud tagastusväärtuse.

3.2.3 Funktsioonid

Kõik keeles kasutatavad funktsioonid on defineeritud lisa I failis „functions.go“. Kui hindamiskeeles on tehtud kutse mingile funktsioonile, siis interpretaator vaatab kas see funktsioon on nimetatud failis defineeritud ning kutsub selle välja ning kui sellise nimega funktsiooni ei ole defineeritud, siis programm lõpetab töö veateatega. See teeb keelele uute funktsioonide lisamise lihtsaks, kuna vaja on lihtsalt see funktsioon nimetatud failis implementeerida ning see on koheselt kasutatav hindamiskeeles.

Ühe sisendiga lambdafunktsioonid sätivad sisendi nimeks `item`, kui funktsiooni väljakutsel pole antud sisendile nime.

Töö valmimise seisuga on defineeritud järgmised funktsioonid:

- `map`, mis teisendab järjendi väärtuseid lambdafunktsioonina etteantud viisil;
- `filter`, mis tagastab järjendi need väärtused, mis tagastavad etteantud lambdafunktsiooni rakendamisel väärtuse `true;`
- `any`, mis tagastab `true` juhul kui etteantud lambdafunktsioon tagastab `true` vähemalt ühe järjendi väärtuse korral;
- `all`, mis tagastab `true` juhul kui etteantud lambdafunktsioon tagastab `true` kõigi järjendi väärtuste korral;
- `none`, mis tagastab `true` juhul kui etteantud lambdafunktsioon tagastab `false` kõigi järjendi väärtuste korral;
- `max`, mis tagastab kas suurima väärtuse arvude järjendist või siis suurima väärtuse etteantud lambdafunktsiooni järgi (lambdafunktsioon peab tagastama `true`, kui esimest argumenti peaks käsitlema suuremana kui teist);
- `min`, mis tagastab kas vähima väärtuse arvude järjendist või siis suurima väärtuse etteantud lambdafunktsiooni järgi (lambdafunktsioon peab tagastama `true`, kui esimest argumenti peaks käsitlema väiksemana kui teist);
- `len`, mis tagastab järjendi või teksti pikkuse;
- `sum`, mis tagastab arvujärjendi elementide summa;
- `avg`, mis tagastab arvujärjendi elementide keskmise;
- `contains`, mis tagastab `true` kui mingi osa tekstist on sama mis etteantud tekst;
- `containsRegex`, mis tagastab `true` kui mingi osa tekstist vastab etteantud regulaaravaldisele;
- `hasModule`, mis tagastab `true`, kui testil on moodul etteantud nimega;
- `getModuleResult`, mis tagastab testi etteantud nimega mooduli `result` välja sisu.

Kõik eelmainitud funktsioonid tagastavad täpselt ühe väärtuse (mis võib olla järjend väärtustest), välja arvatud funktsioon filter, mis juhul kui väljakutsumisel määrata väärtus kahele muutujale korraga, tagastab kaks väärtust: järjend väärtustest, mille korral lambdafunktsioon tagastas `true` ning järjend kõigist ülejäänud väärtustest.

```
/* Valitakse välja kõik testid mille punktisumma ületab 50
   ning teisendatakse need testid arvumassiiviks mis koosneb
   kõigi nende testide punktidest, ning lõpuks summeeritakse
   need punktid.
*/
tests.filter{ item.points > 50 }.map(test){ test.points }.sum()
```

Joonis 8. Näide funktsioonide kasutamisest.

Näide funktsioonide kasutamisest on toodud joonisel 8.

4 Tulemuste analüüs

Selles peatükis käsitletakse töö käigus ilmnenud probleeme ja näidatakse, et valminud lahendus täidab sellele püstitatud nõudeid. Lisaks tuuakse välja mõned võimalused töö edasi arendamiseks.

4.1 Tööprotsessi käigus esile kerkinud probleemid

Edasises tutvustatakse paari arendusprotsessi käigus ilmnenud probleemi ning viisi, kuidas probleem lahendati.

4.1.1 Go tüüp any ja järjendid

Töö käigus selgus, et Go keeles ei ole võimalik kasutada kindla tüübiga järjendit `any` tüüpi järjendi rollis. Kuna interpretaator ei tea ette, mis tüüpi argumente funktsioon tahab (et lihtsustada uute funktsioonide lisamist), siis tekkis probleem, kui hindamisprogramm kasutas sisendandmete välja, milleks oli kindla tüübiga järjend.

Selle probleemi lahenduseks sai interpretaatorile loodud andmete eel- ja järelteisendus, mis muudab sisendandmetes olevad järjendid järjenditeks tüübiga `any`, ning töö lõpetamisel teisendab väljundandmetes olevad `any` tüübiga järjendid tagasi oodatud tüüpi järjenditeks.

Need teisendused on leitavad lisas I toodud failis „utils.go“ funktsioonides `toGenericTest` ja `toNonGenericGroup`.

4.1.2 Paketi constant ootamatu käitumine

Pakett constant kasutab oma funktsioonides väärtusetüüpi `constant.Value`, milleks on võimalik teadmata tüübiga väärtuseid teisendada kasutades funktsiooni `constant.Make`. Selgus, et kuigi pakett constant suudab teisendada andmeid tüübiga `float64` sobivale kujule, kui kasutada funktsiooni `constant.MakeFloat64`, ei suuda `constant.Make` sama teha, mis põhjustas ootamatuid probleeme, kuna mõned funktsioonid väljastasid tulemusi `float64` kujul ning ka sisendandmed sisaldasid `float64` tüüpi andmeid.

Selle probleemi lahenduseks lisati interpretaatorile selleks sobivatesse kohtadesse teisendused `float64` tüüpi väärtustest `constant.Make`-le sobivateks `big.Float` tüüpi väärtusteks.

4.2 Nõuete täitmine

Järgnevas näidatakse, kas ja kuidas loodud lahendus vastab peatükis 1.1 välja toodud nõuetele.

4.2.1 Väljundi struktuur

Loodud lahenduse väljund vastab peatükis 1.3.2 toodud struktuurile, millest on võimalik võrdlemisi lihtsasti teha hinnetetabelit, kuna info ülesannete kuuluvuste kohta gruppidesse ning gruppide koondhinnete kohta on andmetes kajastatud. Järelikult on see nõue täidetud.

4.2.2 Liidestus teiste platvormi osadega

Kuna lahendus on loodud samas programmeerimiskeeles kui teised platvormi moodulid, siis on neist võimalik lahenduse funktsioone kutsuda välja neile vahetult viidates ilma suhtlusprotokollidega vaevumata. Seega on ka see nõue adekvaatselt täidetud.

4.2.3 Turvalisus

Loodud keel täidab turvalisuse nõuet, sest sellega on võimalik teha ainult peatükis 3.2.3 esile toodud operatsioone. Neist operatsioonidest ükski ei võimalda kasutajal negatiivselt mõjutada midagi peale hindamise tulemuse.

4.2.4 Mõistetavus ning vajadus lisateadmistele

Kuna DeepMOOC platvorm tervikuna on alles arendusstaadiumis ning pole veel kasutajatele etteandmiseks valmis, polnud võimalik saada kasutajate tagasisidet loodud keele mõistetavusele ülesannete hindamise kontekstis, kuid keele grammatika muutmine on tehtud võimalikult lihtsaks, et tulevikus oleks mugav seda tagasiside põhjal täiendada.

Kasutajatelt ei oodata selle lahenduse kasutamisel teadmisi platvormi andmevahetusstruktuuridest, sest hindamisfunktsiooni sisend ja väljund on interpretaatoris kindlaksmääratud kujul ning vajadusel toimuvad sisendi ja väljundi teisendused taustal, ilma et kasutaja peaks sellest teadma. Järelikult on see osa nõudest täidetud.

4.2.5 Kiirus

Selle nõude täitmise kontrollimiseks genereeriti kaks juhuslike väärtuste kogumit. Esimeses kogumis oli 30 ülesannet, mis on ligikaudu oodatav ülesannete arv. Teises kogumis oli 1000 ülesannet ning selle kogumi eesmärk oli töökiiruse uurimine oodatust tunduvalt suurema andmemahu korral. Mõlemat kogumit hinnati kolme erineva näitehindamisfunktsiooniga ning mõõdeti hinde arvutamiseks kulunud aega.

Tulemused on toodud tabelis 1, milles on näha, et isegi 1000 ülesande korral ei ületanud ühegi funktsiooni ajakulu 100 millisekundit, mis on piisav, et kiirusega probleemide tekkimine oleks ebatõenäoline.

Tabel 1. Hindamisfunktsioonide ajakulud

Hindamisfunktsiooni kirjeldus	Ülesannete arv	Hindamiseks kulunud aeg
Summa kõigist ülesannete punktidest.	30	1 ms
	1000	10 ms
Kõigi mooduli “boonus” punktide keskmine üle kõigi ülesannete, millel on selle nimega moodul.	30	1 ms
	1000	14 ms
Üheksa grupi loomine ülesande nime põhjal (1. grupis on kõik ülesanded mille nimi on kujul “Kodu1midagi”, teises grupis “Kodu2midagi” ja nii edasi).	30	11 ms
Iga grupi tulemuseks on maksimum sellesse gruppi kuuluvate ülesannete punktidest. Lõpptulemus on summa iga grupi punktidest ning grupeerimata ülesannete punktidest.	1000	68 ms

Ajakulu mõõtmiseks kasutatud andmed ning testimiskood on toodud lisas II.

Kokkuvõte

Bakalaureusetöö eesmärk oli luua DeepMOOC platvormile lahendus erinevate ülesannete põhjal koondhinnete ning hinnetetabelite moodustamiseks.

Töö tulemusena valmis uus programmeerimiskeel Go keeles implementeeritud interpretaatoriga, mis autori hinnangul rahuldab püstitatud nõudeid. Loodud keele grammatikat ja funktsionaalsust tutvustati kolmandas peatükis. Tulemuste peatükis on demonstreeritud selle lahenduse vastavust nõuetele, sealhulgas on toodud näiteid keele töökiiruse kohta.

Lahendus sai loodud funktsionaalsuse täiendamise lihtsust silmas pidades, seega on väga kerge luua uusi funktsioone, mida selles keeles kasutada.

Kuna DeepMOOC platvormi veel arendatakse, siis võib selguda vajadus täiendada keele taustal kasutatavaid struktuure ja nende teisendusi. Lisaks võib peale platvormi kasutusvalmis saamist täiendada loodud keele grammatikat kasutajate tagasiside põhjal.

Kindlasti on võimalik keele töökiirust optimiseerida, kuid praeguse seisuga ei tundu selleks vajadust olevat.

Loodud keel vastab nõuetele ning on kasutamiseks valmis, seega said bakalaureusetöö eesmärgid täidetud.

Viidatud kirjandus

[1] Cheng, L.-C., Li W., Tsend, J. C. R. Effects of an automated programming assessment system on the learning performances of experienced and novice learners. *Interactive Learning Environments*, 2021. https://www.tandfonline.com/doi/full/10.1080/10494820.2021.2006237 (08.08.2022)

[2] Sabag, N, Kosolapov, S. Using instant feedback system and micro exams to enhance active learning. *American Journal of Engineering Education (AJEE)*, 2012, nr 3(2), lk 115-122.

[3] Rodríguez-del-Pino, J. C. 5. Advanced features — Virtual Programming Lab for Moodle (VPL) 3.4.3+ documentation https://vpl.dis.ulpgc.es/documentation/vpl-3.4.3+/advancedfeatures.html (08.08.2022)

[4] Grade calculations – MoodleDocs https://docs.moodle.org/311/en/Grade_calculations (08.08.2022)

[5] ast package - go/ast - Go Packages https://pkg.go.dev/go/ast (08.08.2022)

[6] reflect package - reflect - Go Packages https://pkg.go.dev/reflect (08.08.2022)

[7] constant package - go/constant - Go Packages https://pkg.go.dev/go/constant (08.08.2022)

[8] IntelliJ IDEA overview | IntelliJ IDEA https://www.jetbrains.com/help/idea/discover-intellij-idea.html (08.08.2022)

[9] ANTLR https://www.antlr.org/ (08.08.2022)

[10] Parr T. About The ANTLR Parser Generator https://www.antlr.org/about.html (08.08.2022)

[11] antlr4/targets.md at master · antlr/antlr4 https://github.com/antlr/antlr4/blob/master/doc/targets.md (08.08.2022)

[12] High-order functions and lambdas | Kotlin https://kotlinlang.org/docs/lambdas.html (08.08.2022)

[13] A Tour of Go https://go.dev/tour/methods/14 (08.08.2022)

Lisad

I. Lähtekood

Tööga kaasas olevas zip-failis kaustas „grader“ on loodud lahenduse lähtekood.

Olulisemad failid seal kaustas on kaustas:

- „Grade.g4“ on ANTLR v4 grammatika fail, mis määrab ära loodud keele struktuuri
- „visitor.go“, milles on implementeeritud ANTLR-i loodud parseri kasutamise tulemusel saadud parsimispuu tõlgendamine abstraktseks süntaksipuuks
- „astVisitor.go“, milles on implementeeritud abstraktse süntaksipuu läbimise loogika
- „functions.go“, milles on defineeritud kõik funktsioonid, mida saab loodud keeles kasutada
- „structures.go“, milles on defineeritud sisendi ja väljundi andmestruktuurid
- „grader.go“, milles on funktsioon mis hindab sõne kujul saadud hindamisfunktsiooni põhjal saadud järjendit testidest

„gen“ kaustas on ANTLR tööriista poolt „Grade.g4“ failis oleva grammatika põhjal genereeritud failid.

II. Testimisfailid

Tööga kaasas olevas zip-failis väljaspool “grader” kausta on failid, mida kasutati loodud lahenduse kiiruse testimiseks. Neist olulisemad on:

- “tests30.go”, mis sisaldab muutujat 30 hinnatava ülesandega
- “tests1000.go”, mis sisaldab muutujat 1000 hinnatava ülesandega
- “func1.grade”, mis sisaldab loodud keeles funktsiooni kõigi sisendandmetes olevate ülesannete punktide summeerimiseks
- “func2.grade”, mis sisaldab loodud keeles funktsiooni kõigi mooduli “boonus” punktide keskmine leidmiseks üle kõigi ülesannete, millel on selle nimega moodul.
- “func3.grade”, mis sisaldab loodud keeles funktsiooni ülesannete grupeerimiseks nime järgi, iga grupi hindeks selle grupi ülesannete maksimaalse punktide arvu määramiseks ning lõpuks gruppide hinnete ning ülejäänud ülesannete hinnete summeerimiseks
- “test.go”, mis kasutab ülalmainitud faile sisendina loodud lahendusele ning väljastab saadud tulemuse ning tulemuse saamiseks kulunud aja.

III. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, **Kaarel Kangro**,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose **Koondhinnete ning hinnetetabelite moodustamise keel DeepMOOC platvormile**,

mille juhendajad on Tõnis Hendrik Hlebnikov ja Ahti Põder,

reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Kaarel Kangro

08.08.2022