Tartu Ülikool
Arvutiteaduse instituut
Informaatika õppekava

Kerdo Kurs

Automaattestide loomine kursusele „Programmeerimine keeles C++“

Bakalaureusetöö (9 EAP)

Juhendajad: Helle Hein
Varmo Vene

Tartu 2023

Automaattestide loomine kursusele „Programmeerimine keeles C++“

Lühikokkuvõte:

Programmeerimise populaarsuse kiire kasvamise tõttu on vajalik, et erialane haridus ei jääks oma arenguga alla. Seega on väga oluline, et õppematerjale täiendaksid ka praktilised kodutööd, mille efektiivseks haldamiseks ja hindamiseks on vajalik kasutada automaatseid teste. Esitatud lahendusele kiire ja täpse tagasiside saamine on õppijatele väga oluline efektiivse arengu ja heade tulemuste tagamiseks.

Antud bakalaureusetöö raames loodi suure muudatuse läbinud Tartu Ülikooli kursuse „Programmeerimine keeles C++“ kodutööde jaoks automaattestid ja tööriistad nende kirjeldamiseks ja haldamiseks.

Võtmesõnad: automaattestid, programmeerimiskeel C++, Moodle

CERCS: P175 Informaatika, süsteemiteooria; S270 Pedagoogika ja didaktika

Creating automated tests for the course „Programming in C++“

Abstract:

It is important that with the rapid growth of the popularity of programming the state of dedicated education does not fall behind. Thus it is crucial that there are practical exercises that complement the study materials. Automatic exercises should be used to manage and grade these exercises effectively. Receiving precise feedback instantly after submitting a solution is vital for the effective development of skills and getting better grades for submissions.

This thesis focused on creating automatic tests for the course „Programming in C++“ in the University of Tartu which received a significant structural overhaul this semester. Alongside automatic tests, tools were created to assist in the creation of the tests.

Keywords: automated tests, C++ programming language, Moodle

CERCS: P175 Informatics, systems theory; S270 Pedagogy and didactics

Sisukord

Sissejuhatus 4

1. Teoreetiline ülevaade 5

1.1. Programmeerimiskeel C++ 5

1.2. Programmeerimiskeele C++ väärtus õppetöös 6

1.3. Kursus Programmeerimine keeles C++ 6

1.4. Ümberpööratud klassiruumi struktuur 8

1.5. Automaattestid 9

1.6. Moodle VPL keskkond 10

2. Metoodika 12

2.1. Lahenduste testimine 12

2.2. Tööriist go-vpl-tester 12

2.3. Testide koostamine 12

3. Tulemuste analüüs 14

3.1. Tagasiside 14

3.2. Edasine arendamine 16

Kokkuvõte 18

Viidatud kirjandus 19

Lisad 20

I. Loodud testimise tööriistade lähtekood 20

II. Litsents 21

Sissejuhatus

Kuna infotehnoloogia ja programmeerimine muutuvad iga aastaga aina asjakohasemaks ja populaarsemaks ning tööturg aina suureneb, on väga oluline, et erialane haridus areneks samas tempos. Samuti on vajalik, et õpetamiseks kasutatavad materjalid ja tööriistad on piisavalt ajakohased, et tagada efektiivne areng nii teoreetilistes teadmistes kui ka praktilistes oskustes.

Programmeerimise printsiipide ja tavade õpetamiseks sobivad kõik tööturul kasutuses olevad üldotstarbelised programmeerimiskeeled, näiteks Java või Python, kuid kõige põhjalikuma ülevaate annavad programmeerimiskeeled, mis võimaldavad ühte probleemi lahendada mitmel eri viisil ning ei piira ka tavaliselt abstraheeritud võimaluste kasutamist. Üks parimaid näited selleks on C++, mis on siiani aktiivselt tarkvara arendamises kasutusel. Programmeerimiskeel C++ on oma vahendite poolest väga rikkalik ja võimaldab tudengitel lisaks teoreetilistele teadmistele arendada ka praktilisi oskusi.

Käesoleval semestril sai arvutiteaduse instituudi kursus „Programmeerimine keeles C++“ suure struktuurilise muudatuse, mis tõi kaasa kursuse sihtgrupi suurenemise ja viimasel standardi versioonil põhineva õppematerjali loomise. Sihtgrupi suurenemise tõttu oli vajalik kodutööde hindamiseks võtta kasutusele automaattestid, et esitamist ja tagasisidestamist lihtsustada ning vähem aeganõudvamaks teha.

Lõputöö on jaotatud kolmeks peatükiks. Esimeses peatükis kirjeldatakse programmeerimiskeelt C++ ja uuendatud kursust „Programmeerimine keeles C++“. Teises peatükis antakse ülevaade, mis tööriistad testide loomiseks valmistati ning kuidas teste läbi viidi. Viimases peatükis tuuakse välja automaattestide tagasiside ja arutletakse tulemuste kasulikkuse ja edasiarendamise võimaluste üle.

1. Teoreetiline ülevaade

1.1. Programmeerimiskeel C++

Programmeerimiskeel C++ on staatilise tüübisüsteemiga üldotstarbeline kompileeritav keel, mis võimaldab kirjutada tõhusat kõrge- ja madalatasemelist koodi. Keele autori Bjarne Stroustrup väidab [1], et selle tõttu on C++ vägagi sobilik nii üldise otstarbega rakendustarkvara kui sardsüsteemides kasutatavate programmide loomiseks. Programmeerimiskeelt kasutatakse siiani näiteks operatsioonisüsteemide, arvutimängude, otsingumootorite ja muu suurt jõudlust nõudva tarkvara kirjutamisel.

Keel sai alguse 1970ndate aastate lõpus nimega *C with classes*. Nagu nimigi viitab, oli tegemist laiendiga programmeerimiskeelele C, kuhu oli lisatud tugi objektorienteeritud paradigmale. Keele algne eesmärk oli säilitada oma eelkäija efektiivsus ja paindlikkus sardsüsteemide tarbeks tarkvara loomisel samal ajal arenduseks kuluvat aega vähendades [2]. Kuigi eesmärke on aastate jooksul täpsustatud ja täiendatud, on efektiivsus ja paindlikkus siiski säilinud [2].

Kirjutatud programmide efektiivsus tuleneb sellest, et programmi lähtekood kompileeritakse masinkoodiks. Keerukamad keele konstruktsioonid ei lisa seetõttu ka programmidele reeglina üleliigset käitusajal töödeldavat sisu, vaid tõlgitakse samamoodi vaid protsessori instruktsioonideks. Seetõttu ei olene programmide kiirus ja efektiivsus lähtekoodi omadustest ega süntaksi keerukusest.

Keele süntaksi, ülesehituse ja võimalused defineerib C++ ISO standard, kuhu on vabatahtlikult panustanud aastate jooksul palju inimesi. Uue versiooni koostamiseks kulub kolm aastat, mille jooksul oodatakse ettepanekuid uute vahendite jaoks. Ettepanekud kinnitab mittetulundusühing The Standard C++ Foundation, kuhu kuulub ka keele algataja Bjarne Stroustrup [3]. Viimane ametlikult kasutusel olev versioon on C++ 20 ja käesoleva aasta septembrisse on planeeritud ka C++ 23 ametlik kasutuselevõtt [3].

Uue versiooni kasutusele võtmine nõuab standardi toe lisamist kompilaatorisse ja muudesse tööriistadesse. Igal kompilaatoril on enda standardi implementatsioon, mida nimetatakse standardteegiks (ingl k. *standard library*). Standardteek sisaldab standardi dokumendis välja toodud võimalusi, andmetüüpe ja funktsioone. Rikkalik standardteek annab kasutajale tööriistad mugavalt ja efektiivselt C++ koodi kirjutamiseks.

1.2. Programmeerimiskeele C++ väärtus õppetöös

Lisaks standardteegi mugavatele vahenditele, võimaldab keel suhelda otse seadme riistvaraga ning teha selle peal madala taseme operatsioone. See teeb keelest väga hea õppevahendi ülikoolides nii informaatika kui ka arvutitehnika õppekavades.

C++ keeles puudub automaatne mälukoristuse tehnika, mis sageli teistes õpetatavates keeltes olulised on. See tähendab, et kasutaja peab ise haldama kasutatud ressursse ja tegema kindlaks, et nende kasutamisega ei tekiks standardi järgi määratlemata käitumist, mis tähendab, et programmi tulemuses ja käitumises ei saa enam kindel olla [4]. Mälukoristuse tehnikate puudumise tõttu tuleb dünaamilise mälu haldamiseks kasutajal selgelt seda võtmesõnadega *new* ja *delete* väljendada.

Väga palju manuaalset tööd, näiteks dünaamilist mäluhaldust, saab muuta automaatseks ja turvaliseks erinevaid keelekonstruktsioone õigesti kasutades. Üks kasulikumaid neist on näiteks keelele C++ omane tehnika nimega RAII ehk ressursside omandamine on algväärtustamine (ingl. k *Resource Acquisition Is Initialization*). See kujutab endast tehnikat piirata ressursside kasutamist objektide elueaga [5].

Keele erinevate vahendite omavaheline korrektne kasutamine tekitab hea dünaamika, kus keele õppimisel saab süveneda rohkemasse kui ainult koodi kirjutamisse. See tekitab enamasti küll detailsema arusaama programmeerimiskeele tööpõhimõttest ja heast kasutamisest, kuid võib osutuda probleemiks, kui keele õppimisel algteadmistest piisavalt hästi aru ei saada.

Keele autor B. Stroustrup on ise ka veendumusel, et modernse C++ õpetamine on vajalik ja jagab konverentsidel keele õpetamiseks nõuandeid [6]. Autor on välja andnud ka raamatuid, kus kirjeldab C++ loomist ja kirjeldab lühidalt, kuidas keeles programme kirjutada [1].

1.3. Kursus Programmeerimine keeles C++

Kursus Programmeerimine keeles C++ on arvutiteaduse instituudi kursus, mille eesmärgiks on anda tudengitele ülevaade keele põhitõdedest ning tutvustada keeles programmeerimiseks vajalikke tööriistu [7].

Kursuselt eemaldati varasem eeldus, et kursusel osaleja peaks olema läbinud aine „Objektorienteeritud Programmeerimine“ või on objektorienteeritud paradigmaga kursis mingil muul moel. Eelduse eemaldamine langetas kursuse raskustaset ja suurendas kursuse sihtgruppi – nüüd piisas kursusel osalemiseks ainult programmeerimise algteadmistest. See tähendas ka seda, et ainet said võtta ka arvutitehnika eriala tudengid, kelle õppekavas ei ole kohustuslikku ainet, mis objektorienteeritud paradigmat otseselt käsitleks.

Eeldusest lahti saamiseks tehti ümber kursuse ülesehitus ja õppematerjal. Uue ülesehituse loomise käigus loodi kursusele uus ainekava, suurendati kursuse mahtu 3 ainepunktilt 6 peale ning loodi uus materjal, mis on saadaval instituudi kursuste süsteemis. Ainekava ja materjali loomist arutati aine koosolekutel ning teemade valimisel konsulteeriti ka aine algse autoriga. Kursuse vanast ülesehitusest jäi alles ka tudengite endi projekti loomine. Tabelis 1 on välja uuendatud kursuse tööde ja punktide jaotust ning läbimise tingimusi.

Tabel 1. Kursuse uue struktuuri tööde ja punktide jaotus ning läbimise tingimused (kohandatud kursuse kodulehelt [7])

	Aeg	Alampiir	Kokku	Kommentaar
Kodutöö + praktikum	1.-16. nädal		36	Ühest kodutööst ja praktikumist kokku 3 punkti. Kokku 12 tööd.
Projekt	7.-15. nädal	5	14	Esitamine kahes osas, kokku 5 + 9 punkti. Esitlus 3 punkti. 16. nädalal.
1. kontrolltöö	7. nädal		25	
2. kontrolltöö	15. nädal		25	
Lisaülesanded			10	

Kursuse materjali koostamisel lähtuti standardi 20 versioonist, mis on kasutusel olnud aastast 2020. Materjalis võeti eesmärgiks katta peamiseid modernse C++ vahendeid ja tavasid ja hoiduda keerulistest teemadest, mis keele selgeks saamist ei takistanud. Kursuse alguses oli

suurem rõhk üldisemate programmeerimise teemade tutvustamisel, hiljem süveneti aga ainult C++ vahenditele ja võimalustele. Teemade jaotusest saab ülevaate tabelist 2.

Tabel 2. Teemade jaotus nädalate kaupa

1. nädal	Muutujad ja andmetüübid
2. nädal	Keele põhikonstruktsioonid I
3. nädal	Keele põhikonstruktsioonid II
4. nädal	Funktsioonimallid, failitöötlus
5. nädal	OOP I – Klassid
6. nädal	OOP II – Pärilus ja polümorfism
7. nädal	1. kontrolltöö
8. nädal	Dünaamiline mäluhaldus I
9. nädal	Dünaamiline mäluhaldus II
10. nädal	Klassimallid
11. nädal	STL andmestruktuurid I
12. nädal	STL andmestruktuurid II
13. nädal	Erindite töötlemine
14. nädal	Täiendavad teemad
15. nädal	2. kontrolltöö
16. nädal	Projektide esitlemine

Igal (v.a 7., 15., ja 16.) nädalal oli ka materjalile põhinev kodutöö, millest tuli esitada esmane lahendus praktikumieelse päeva ning lõplik lahendus vastava nädala reede õhtuks.

1.4. Ümberpööratud klassiruumi struktuur

Kursuse struktureerimisel võeti kasutusele ümberpööratud klassiruumi mudel, mis on populaarsust kogunud just tehnoloogiaga seotud akadeemilistes keskkondades. Mudeli eesmärk on viia materjali omandamine rohkem individuaalsemaks ja veebipõhisemaks. See tähendab, et tudengid töötavad ettevalmistatud materjalidega iseseisvalt ning praktilistes tundides tegeletakse teadmiste kinnistamisega praktikumiülesandeid lahendades. Õpetaja ei ole enam õppimise keskpunktis, vaid on seal peamiselt tudengitele lisainfo ja toe pakkumiseks. [8]

Tehnoloogia jätkuva kiire arengu ja akadeemilistesse keskkondadesse integreerimisega on vana klassiruumi dünaamika jäänud ajale jalgu. Ümberpööratud klassiruumi struktuurile üleminek stimuleerib väljaspool klassiruumi omandatud huvi, loovust, võimeid, tiimitööd ja teadmisi [8]. See kutsub omakorda tudengites esile arenenuma kriitilise ja iseseisva mõtlemise, mis on programmeerimise õppimisel hädavajalikud [8].

1.5. Automaattestid

Ümberpööratud klassiruumi mudeliga sobivad hästi kokku automaatsed testid, mille kasutamine on kasulik mitmel põhjusel.

Esiteks võimaldavad hästi koostatud automaattestid tudengitel säästa väärtuslikku aega [9]. Nimelt on tudengitel võimalus saada tagasisidet esitatud töö kohta kohe pärast selle esitamist. Saadud tagasiside on piisav, et tudeng saaks ebatäpse lahenduse korral oma koodi parandada. See on kasulik nii õpilastele kui ka juhendajatele, kuna nii on võimalik vältida sageli aeglast ja pikale venivat meilivahetust osapoolte vahel [9].

Teiseks aitavad automaattestid tagada tudengite tööde objektiivse hindamise. Seega hinnatakse kõiki töid täpselt samade kriteeriumide alusel ning hinded kujunevad välja võimalikult lähtudes täpselt ülesande püstitusele. Küll aga ei võta testid arvesse rohkemat kui lahenduse korrektset toimimist ja tulemust, mistõttu võib test peaaegu täpse lahenduse lugeda valeks. Seepärast on hädavajalik, et aine juhendajad vaatavad pärast automaatsete hinnete saamist esitatud lahendused käsitsi üle. See protsess võtab reeglina vähem aega kui kodutöö hindamine ilma automaatse testimiseta. Sellest tulenevalt on juhendajatel rohkem aega lahendusele lisaks veel konstruktiivsema ja lahendusele omasema tagasiside andmiseks.

Kolmandaks pakuvad automaattestid lisaks mugavusele ja efektiivsusele võimaluse lahenduste parandamisel arendada ka oskusi lugeda ja analüüsida erinevaid veateateid ja uue lahenduse jaoks ise informatsiooni otsimist. Eriti usinad ja huvitatud tudengid on kindlasti rohkem altid kindlate teemade kohta veel rohkem süvitsi materjali otsima. Seetõttu puutub õpilane jooksvalt kokku erinevate pärismaailmas esinevate probleemide ja nende lahendustega ning arendab oma praktilist oskust reaalsetele probleemidele lahendusi leida. Ise küsimustele vastuseid ja probleemidele lahendusi otsides arendavad tudengid oma kriitilist ja iseseisvat mõtlemist. Need oskused on programmeerimise õppimisel kriitilised.

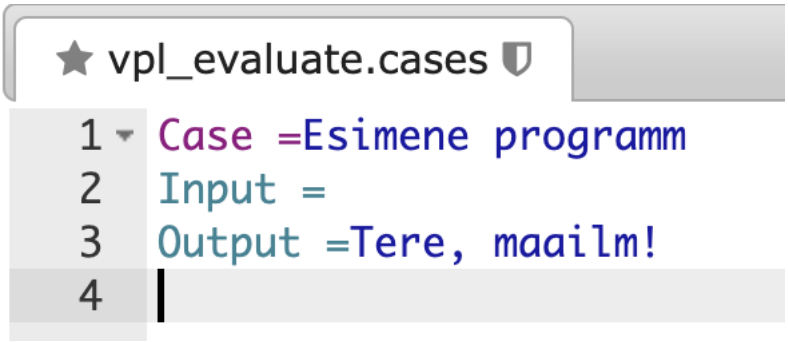
Küll aga ei näita kirjeldatud punktid paremaid eksamitulemusi ja sügavamat materjali mõistmist [9].

1.6. Moodle VPL keskkond

Lisaks instituudi kursuse keskkonnale kasutas aine nüüdsest rohkem Moodle'i keskkonda. See avas võimaluse kasutada lisamoodulit VPL ehk *Virtual Programming Lab*, mis on loodud programmeerimisülesannete läbiviimiseks [10]. Moodul teeb mugavaks ülesannete koostamise, esitamise ja tagasisidestamise: kindlale ülesandele on võimalik esitada lahendus ja vajadusel seda korrigeerida ja parandada [10]. Lisaks tekib iga esitamise juurde kohe tagasiside ja hinne, mis lähevad automaatselt tudengite hinnete alla, mis teeb ülesannete tagasisidestamise ja hindamise väga lihtsaks ja täpseks. Keskkond on kasutusel ka mitmetes teistes Tartu Ülikooli ainetes ning seega on see juba nõuete järgi seadistatud ning testitud.

Testi loomiseks tuleb Moodle'is luua VPL ülesande objekt ja see vastavalt ülesande nõuetele konfigureerida. Ülesande enda konfigureerimine toimub analoogselt teistele Moodle'i vahendite seadistamisele, küll aga erineb testide kirjeldamine ja nende käitamine.

Testide kirjeldamiseks on moodulil oma süntaks, mille abil saab kirja panna testjuhud, mida esitatud lahenduse vastu jooksutatakse. Joonisel 1 on toodud näide testjuhust, mis kontrollib, et käitatud programmi väljastas „Tere, maailm!".



Joonis 1. Näide lihtsast VPL ülesande testjuhust.

Lahenduse esitamisel kuvatakse testjuhtude tulemust kirjeldav väljund, mis on välja toodud joonisel 2.

Joonis 2. Näide korrektse lahenduse tagasisidest.

Moodul võimaldab veel ka juhendajatel pärast iga töö üle vaadata ja vajadusel tagasisidet ning hinnet muuta.

2. Metoodika

2.1. Lahenduste testimine

Kursuse kodutööd järgisid peamiselt kaht struktuuri. Esimene neist nõudis programmi, mis vastava sisendi korral nõutud väljundi annab nõutud väljundi. Teine, rohkem kasutatud struktuur, nõuab üht või mitut funktsiooni või klassi, mis töötaksid eelnevalt kirjeldatud käitumise järgi. Esimene lähenemine on küll parem sisendi ja väljundi harjutamise ja terve programmi kontrollimiseks, kuid väiksemate ülesannete ja kindlate vahendite kasutamist on sellise lähenemisega ebapraktiline teha. Siin on targem rakendada teist lähenemist ja testida igat loodud tükki – funktsiooni või klassi – eraldi.

Sellist lähenemist nimetatakse tarkvaraarenduses üksuste testimiseks (ingl. k *unit testing*) ning peetakse kvaliteetse tarkvara kirjutamisel hädavajalikuks praktikaks. Kursuse automaattestide loomisel lähtuti suuresti üksuste testimise põhimõtetest. Iga kodutöö ülesannet testitakse mitmete erinevate sisendargumentidega kontrollimaks, et lahendus töötab kõigis olukordades.

2.2. Tööriist go-vpl-tester

Moodle'i VPL mooduli paremaks kasutamiseks loodi vabavaraline tööriist go-vpl-tester, mis mooduliga analoogselt kirjeldatud testjuhtusid kontrollib. Tööriist disainiti käitama alamprogrammina esitatud lahendust ja võrdlema sellele antud sisendiga saadud väljundit korrektse lahendusega. Sellise disainitingimuse täitmiseks sobis hästi programmeerimiskeel Go, mille standardteegis on vajalikud vahendid alamprogrammide jooksutamiseks ja nende voogude kontrollimiseks.

2.3. Testide koostamine

Iga kodutöö jaoks loodi testjuhud, mis kontrollisid töö püstituses kirjeldatud ülesannete käitumist vastava sisendi alusel. Joonisel 3 on välja toodud kaks testjuhtu. Esimene neist kasutab lahenduse tulemuse võrdlemiseks korrektse programmi väljundit, teine võrdleb väljundit aga määratud väärtusega.

vpl_evaluate.cases

```
1  [Kolmnurk (tühi)]
2  Input = 0
3  Args = kolmnurk
4  CompareTo = ./correct
5
6  [Kolmnurk (väike)]
7  Input = 1
8  Output = *
9  Args = kolmnurk
10
```

Joonis 3. Näide loodud tööriista testjuhtude kirjeldamisest

Lisaks testjuhtudele vajas iga kodutöö test ka C++ programmi, mis sisendi alusel õige lahenduse funktsiooni käitas ning ülejäänud sisendi sinna õiges formaadis edastas. Kursuse jooksul arendati aga testide süsteeme vastavalt vajadusele edasi ja jõuti lõpuks lahenduseni, kus testjuhud defineeriti, käitati ja väljastati hoopis otse C++ programmis.

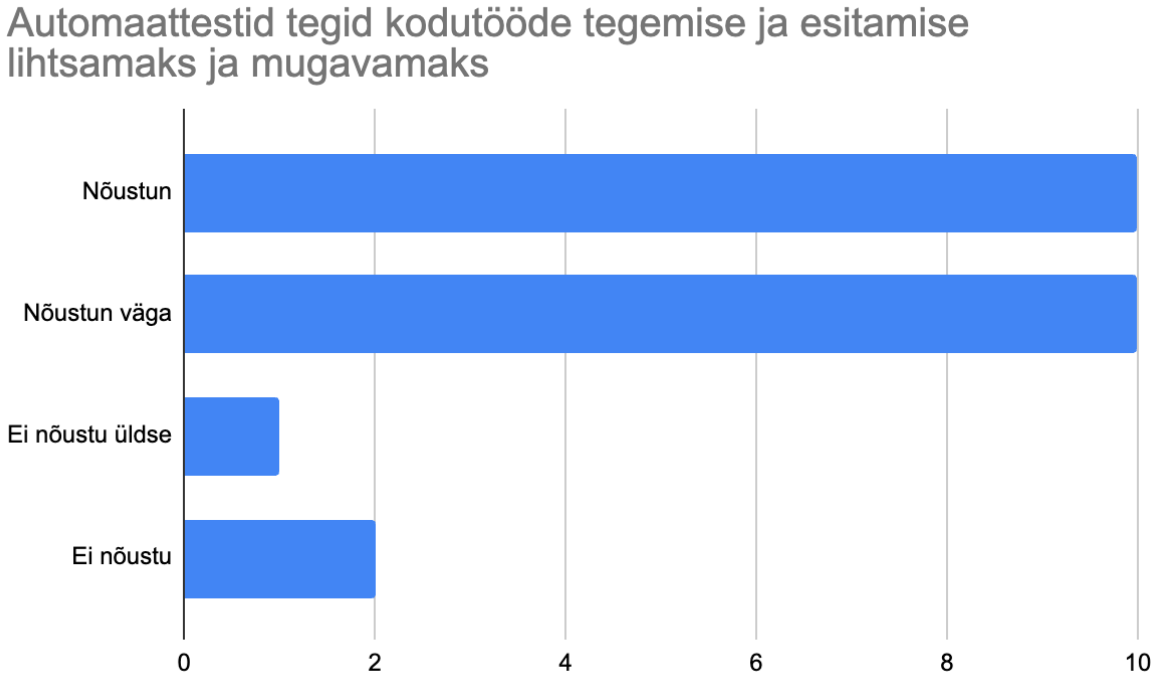
3. Tulemuste analüüs

3.1. Tagasiside

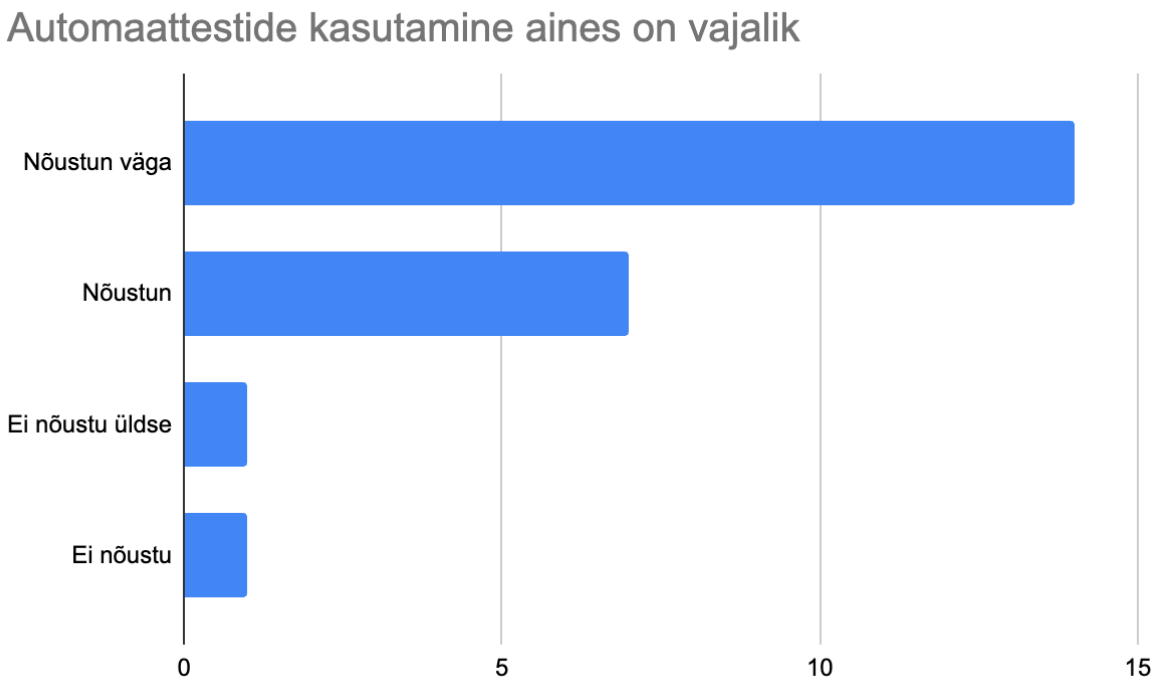
Automaattestide efektiivsuse ja töö kasu hindamiseks viidi kursusel osalenute ja praktikumijuhendajate seas läbi tagasisideküsitlus. Küsitluses uuriti, kas automaattestid tegid tudengite jaoks tööde tegemise ja esitamise mugavamaks. Juhendajatelt uuriti, kas testid tegid hindamise ning tagasisidestamise kiiremaks ja efektiivsemaks. Lisaks anti võimalus avaldada muid mõtteid automaatsete testide paremaks tegemiseks.

Kursusel osalenud 83 tudengist vastas tagasisidele 23. Vastanute seas oli nii informaatika kui arvutitehnika eriala ning üks täiendusõppe tudeng.

Tagasisidest selgus, et üldjoontes oli automaattestide kasutamine vajalik ja tudengitele kasulik. Joonistel 3 ja 4 on näha, et 20 tudengit 23-st nõustuvad, et automaattestide kasutamine tegi kodutöö lahenduste esitamise lihtsamaks ja 21 tudengit on päri väitega, et automaattestide kasutamine aines oli vajalik.

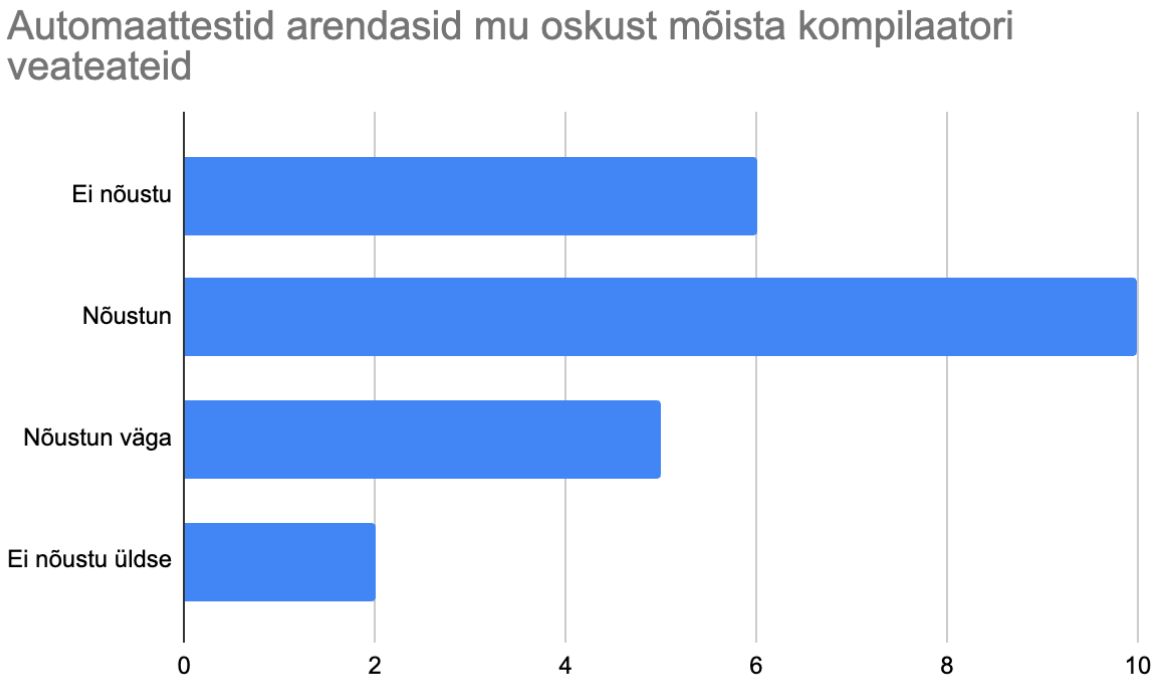


Joonis 3. Tudengite tagasiside jaotus väitele, et testid lihtsustasid tööde esitamist

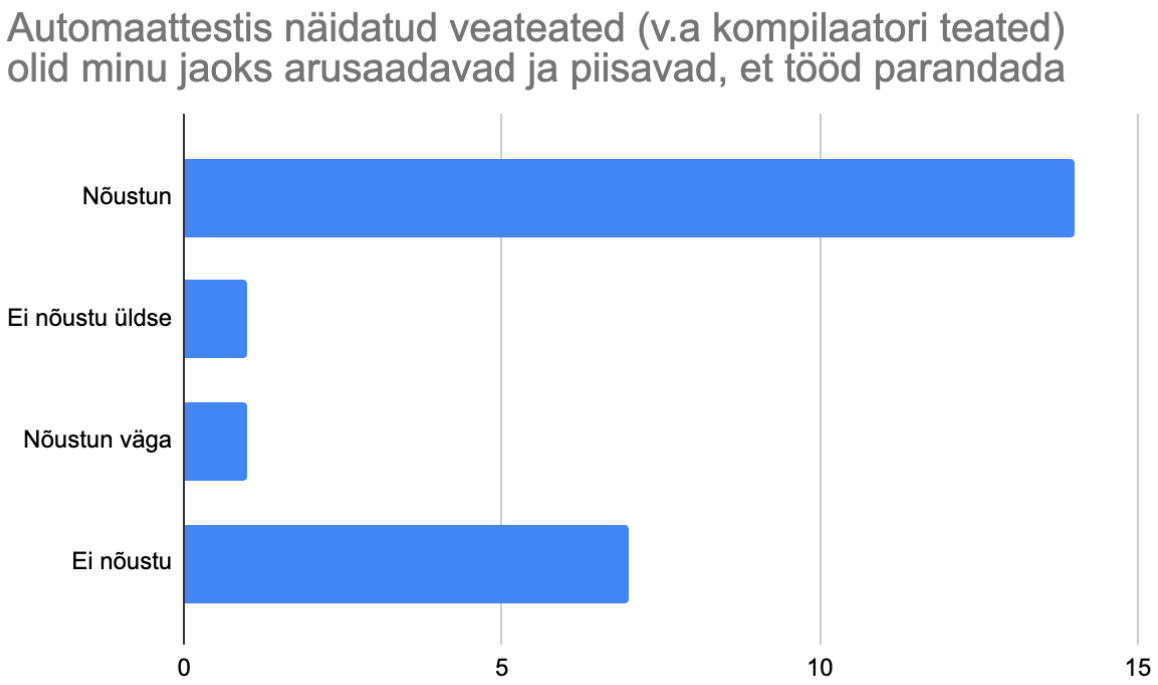


Joonis 4. Tudengite tagasiside jaotus automaattestide kasutamise vajalikkuse kohta

Väitega, et automaattestid arendasid oskust kompilaatori veateadetest paremini aru saada, nõustus vaid 15 vastanut. Sama palju vastanuid arvas ka, et testide endi veateated olid mõistlikud ja arusaadavad.



Joonis 5. Tudengite tagasiside jaotus testide veateadete mõistmise oskuse arendamise kohta



Joonis 6. Tudengite tagasiside jaotus testide endi veateadete arusaadavuse kohta

Lisaks kehvadele veateadetele on tudengid välja toonud ka ideid, kuidas teste paremaks teha. Suurem osa ideedest on seotud pikkade kompilaatori veateadete vältimisega. Veateadete vältimiseks on soovitatud kasutada lisatööriistu. Lisaks on ebamugavust tekitanud ka liiga ranged kompileerimise reeglid.

Praktikumijuhendajate seas läbi viidud küsitlusele vastas 2 juhendajat.

Mõlemad juhendajad nõustuvad väitega, et nende arvates tegi testide kasutamine tudengitele lahenduste esitamise lihtsamaks ja mugavamaks, ning leiavad, et automaattestide kasutamine tegi praktikumide andmise ja lahenduste lõpliku hindamise mugavamaks. Tagasiside andmetel on näha, et mõlemal juhendajal kulus automaattestidega kodutöö hindamiseks ligikaudselt 2 korda vähem aega. See tuleneb võrdlusest ühe kodutöö ilma testita hindamisega.

3.2. Edasine arendamine

Loodud infrastruktuuri ja testide edasiarendamisel tuleks kindlasti integreerida muid tööriistu. Enne lahenduse kompileerimist oleks hea esitatud koodi analüüsida, et nõutud funktsioonide

või klasside lahendused on nõutud kujul olemas. Nii saab testida vaid esitatud ja ignoreerida poolikuid lahendusi. Nii saab hoida ära kompilaatorist tulevaid ebameeldivaid veateateid, mis ei ole seotud ülesande lahendusega, vaid on tingitud näiteks ühe funktsiooni puudumisest. Esitatud lahenduste muutuste analüüsimisel on ka näha, et tihti esitasid tudengid poolikuid lahendusi, mis ei sisaldanud kõik nõutud elemente. Seetõttu esitatud lahenduse kompileerimine ebaõnnestus.

Kindlasti tuleb rõhku panna ka arusaadavate veateadete koostamisele. Hästi koostatud veateadete korral on tudengil selge, mis probleem tema lahenduses esineb ja on võimeline edasi uurima, kuidas seda lahendada.

Kokkuvõte

Bakalaureusetöö eesmärk oli koostada uuendatud struktuuri ja materjalidega kursuse „Programmeerimine keeles C++“ tarbeks automaatteste. Lõputöö raames töötati välja 11 automaatset testi, testide kirjeldamiseks vabavaraline tööriist ning infrastruktuuri põhi edasiste automaattestide loomiseks. Automaattestide efektiivsust hinnati kursuse 2022/2023. õppeaasta kevadsemestri osalejate ja praktikumijuhendajate tagasiside põhjal.

Automaattestide koostamisel lähtuti keele parimatest kasutusviisidest ning fokusseeriti hea ja kvaliteetse koodi nõudmisele. Lisaks kodutööde automaatsetele testidele töötati ka välja dünaamiliselt areneva infrastruktuuri põhi, mis on kogumik C++ lahendustest, mis abistavad edasiste kodutööde testide parendamist ja uute loomist.

Kursusel osalenutelt saadud tagasiside alusel võib väita, et kursusel on kindlasti automatiseeritud testide kasutamine vajalik. Testid olid tudengitele enamjaolt abiks kuid C++ komplekssusest ja ülesehitusest tulenevate iseärasuste tõttu ei andnud testid kohati vigade tekkimisel arusaadavat tagasisidet.

Töö tulemustel on mitmeid edasiarendamise võimalusi. Esiteks saab loodud infrastruktuuri põhja peale hõlpsalt luua teste ka näiteks harjutusülesannete tarbeks. Taristu edasiarendamisel on võimalik lisada tugi täpsemale ja lahendusele omasemale tagasisidele. Teiseks on võimalik ja tagasisidest järelduvalt vaja integreerida testidesse süsteem, mis enne lahenduse käitamist selle struktuuri kontrollib.

Viidatud kirjandus

[1] Stroustrup B., The C++ programming language. Fourth Edition, Addison-Wesley, 2013.

[2] Stroustrup B., "A History of C++: 1979--1991," *History of Programming Languages---II*. New York, USA: Association for Computing Machinery, 1996, p. 699–769.

[3] Standard C++. https://isocpp.org/ (04.05.2023).

[4] Undefined behaviour - cppreference.com. https://en.cppreference.com/w/cpp/language/ub (09.05.2023).

[5] RAII - cppreference.com https://en.cppreference.com/w/cpp/language/raii (09.05.2023).

[6] CppCon, Stroustrup B. CppCon 2017: Bjarne Stroustrup “Learning and Teaching Modern C++” https://youtu.be/fX2W3nNjJIo (09.05.2023).

[7] Programmeerimine keeles C++ - Kursused - Arvutiteaduse instituut https://courses.cs.ut.ee/2023/cpp/spring (09.05.2023).

[8] Flores, Ò., del-Arco, I. & Silva, P. The flipped classroom model at the university: analysis based on professors’ and students’ assessment in the educational field. *Int J Educ Technol High Educ*, 2016, nr 13.

[9] Skalka, Ján. Drlík, Martin. Obonya, Juraj. Automated Assessment in Learning and Teaching Programming Languages using Virtual Learning Environment. IEEE Global Engineering Education Conference (EDUCON), 2019, pp. 689-697, doi: 10.1109/EDUCON.2019.8725127

[10] VPL - Virtual Programming Lab https://vpl.dis.ulpgc.es/ (08.05.2023).

Lisad

I. Loodud testimise tööriistade lähtekood

Lõputöö raames loodud vabavaralise testimise tööriista lähtekood on saadaval järgmisel aadressil: https://github.com/kerdokurs/go-vpl-tester.

Testide endi ja liidese lähtekood on kättesaadav vaid kursuse õppejõududele Moodle'i keskkonnas.

II. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Kerdo Kurs,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose **Automaattestide loomine kursusele *Programmeerimine keeles C*++**,
   mille juhendajad on **Helle Hein** ja **Varmo Vene**,
   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 4.0, 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.

Kerdo Kurs
09.05.2023