TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Edvard Notberg

Tudengiprogrammide automaatkontrollid programmeerimise kursustel

Bakalaureusetöö (9 EAP)

Juhendaja: Reimo Palm, PhD

Tartu 2024

Tudengiprogrammide automaatkontrollid programmeerimise kursustel

Lühikokkuvõte:

Automaatkontrollid on programmeerimise kursustel väärtuslik abivahend nii tudengitele kui ka õppejõududele. Selleks et automaattestid täidaksid oma eesmärke, peavad need olema täpsed ja kontrollima kõiki ülesande eeldusi. Käesolev bakalaureusetöö keskendub Tartu Ülikooli kursuse "Programmeerimine" praktikumiülesannete automaatkontrollide analüüsimisele ja täiustamisele. Ühtlasi antakse ülevaade automaatkontrollide loomise vahendist TSL (*Test Specific Language*) ja tuuakse välja selle tugevused ning puudujäägid.

Võtmesõnad:

programmeerimine, programmeerimise kursus, automaatkontroll, Test Specific Language

CERCS: P175 Informaatika, süsteemiteooria, S281 Arvuti õpiprogrammide kasutamise metoodika ja pedagoogika

Automated assessments of student programs in programming courses

Abstract:

In programming courses, automated assessments are a valuable tool for both students and instructors. For automated tests to fulfill their purpose, they must be accurate and test all the requirements of an assignment. This bachelor's thesis focuses on analyzing and improving the automated assessments for the practical exercises in the Computer Programming course at the University of Tartu. Additionally, the thesis provides an overview of the tool used to create automated tests, TSL (Test Specific Language), and outlines its strengths and weaknesses.

Keywords:

programming, programming course, automated tests, Test Specific Language

CERCS: P175, Informatics, systems theory, S281 Computer-assisted education

Sisukord

Sissejuhatus

Sisseastumise infosüsteemi (SAIS) [1] 2023. aasta andmete järgi on arvutiteadus Tartu Ülikoolis üks populaarsemaid erialasid [2]. Arvutiteaduse õppekava sisaldab mitut kohustuslikku programmeerimise kursust, kus läbi erinevate ülesannete lahendamise tutvuvad tudengid uute programmeerimiskeeltega, arendavad probleemide lahendamise oskusi ja loogilist mõtlemist. Õppimisprotsessi toetamiseks ja õppejõudude koormuse vähendamiseks kasutatakse nendes ainetes õppijate programmide hindamiseks automaatkontrolle, mis annavad nii kontrollijale kui tudengile kiiret tagasisidet.

Möödunud aasta sügissemestril võeti TÜ kursusel "Programmeerimine LTAT.03.001" [3] esmakordselt kasutusele TSL-keeles (*Test Specific Language*) koostatud automaatkontrollid, mis lahendasid eelnevalt kasutusel olnud testide kitsaskohti [4, 5]. Vastavalt juhendajalt saadud andmetele tuvastati sügissemestri jooksul hulk praktikumiülesandeid puudulike automaatkontrollidega – mõne ülesande juures kuvati kasutajale korrektse lahenduse esitamisel veateadet ja osadel harjutustel ei kontrollitud kõiki ülesandes püstitatud nõudeid.

Lõputöö eesmärk on analüüsida kursuse "Programmeerimine" praktikumiülesannete, näidiskontrolltööde ja näidiseksamite automaatkontrolle, tuvastada puudujäägid ja need kõrvaldada.

Töö on jaotatud kuueks peatükiks. Esimeses peatükis antakse ülevaade automaatkontrollist, selle ajaloost ja tuuakse välja selle kitsaskohad. Järgmine peatükk tutvustab TÜ kursuseid, kus kasutatakse automaatkontrolle, ning samuti õppekeskkonda Lahendus. Kolmandas peatükis kirjeldatakse TSL-keelt ning tuuakse näiteid automaatkontrollidest. Metoodika peatükk keskendub etappidele, milleks oli jaotatud olemasolevate automaatkontrollide analüüsimine. Viiendas peatükis tehakse kokkuvõte valminud töö tulemustest ja leidudest ning viimases peatükis arutatakse, millist mõju avaldab valminud bakalaureusetöö programeerimiskursustele ja TSL-keele arengule.

1. Automaatkontrollid

Vastavalt Mayersi [6] definitsioonile on tarkvara testimine arvutiprogrammi uurimise protsess, mille eesmärk on koodivigade leidmine. Vigade all mõeldakse defekte programmis, mis põhjustavad lahkheli oodatud ja tegelike tulemuste vahel. Tarkvara korrektses toimimises veendumiseks võib programmi käitumist uurida nii käsitsi kui ka kasutades automaatkontrolle, kuid üldjuhul antakse võimalusel lahenduse kontrollimise töö arvutile. Selle jaoks kirjutatakse kood, mis uurib etteantud programmi ja võrdleb saadud väljundeid õigete vastustega [7].

Automaatkontrollil on mitmeid eeliseid käsitsi testimise ees:

- Kiire rakendamine - sama töömahu puhul võtab automaatne kontrollimine kordades vähem aega.
- Inimliku hooletuse välistamine - väsimusest ja puudulikust tähelepanelikkusest tingitud vigadest hoidumine.
- Võimalus suures mahus andmeid salvestada ja analüüsida.
- Taaskasutatavus - automaatkontrolli saab rakendada korduvalt.

1.1 Automaatkontrollide ajalugu

Esimesed automaatkontrollid võeti kasutusele 1959. aastal, kui New Yorgi Rensselaeri Polütehnilises Instituudis kontrolliti kahekümne tudengi programmeerimise ülesannete lahendused assemblerkeeles automaatselt, kasutades selleks nn automaatset hindajat ja arvutit IBM 650 [8]. Kontrollimise protsessi käigus sisestati õpilase lahendusega perfokaart aukkaardilugejasse, mille järel käivitas automaatne hindaja sisse loetud koodi ning veatu programmi jooksutamise tulemusena andis väljundiks "*Problem complet*e", vigade esinemisel tuli väljundiks "*Wrong answer*". Esimesel katsetamisel sooritas automaatne hindaja kahekümne tudengi tööde hindamise viie minutiga, peale mida otsustati kasutada sama tudengirühma lahenduste hindamiseks automaatkontrolle kogu kursuse vältel ning ka hilisematel sama aine kursustel. Automaatkontrollide kasutus kiirendas senist lahenduste hindamist umbes kolmekordselt, mis võimaldas säästa õppejõudude aega ja seeläbi tõsta kursusel osalevate tudengite piirarvu [8].

Suur edasiminek automaatkontrollide süsteemide arengus toimus 1980. aastaste lõpus, kui hakkasid tekkima käsurea ja graafilise kasutajaliidesega programmeerimisülesannete

testimise platvormid [9]. Üheks näiteks on 1989. aastal valminud tudengiprogrammide testimise tarkvara TRY [10], mis võimaldas õppijatel oma lahendust iseseisvalt kontrollida ja tuvastada vigu. Hiljem arendati Inglismaal mitmeid sama funktsionaalsusega süsteeme, millest kujunesid välja täieulatuslikud õppeplatvormid [9]. Sellised süsteemid olid Liverpooli Ülikoolis loodud programmide testimise tarkvara ASSYST [11] ja Warwicki Ülikooli automaatne lahenduste esitamise ja hindamise süsteem BOSS [12].

Tartu Ülikool tegi märkimisväärse sammu programmeerimisharjutuste hindamise automatiseerimisel 2013. aastal, kui bakalaureusetöö raames koostas TÜ tudeng, Karl-Aksel Puulmann, koostöös juhendaja Aivar Annamaaga esimese ulatusliku automaatkontrollimise süsteemi, Python Grader [13]. Rakendusega sai testida enamiku Pythoni keeles toimuvate algkursuste ülesannete lahendusi.

1.2 Automaatkontrollide kitsaskohad

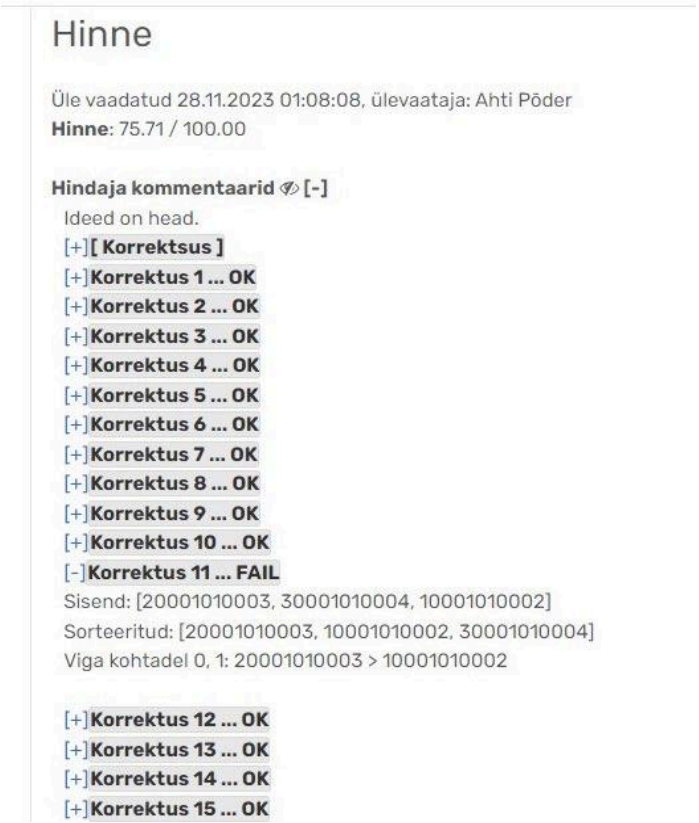
Vaatamata automaattestide tugevatele külgedele, ei saa nad programmeerimise ülesannete lahenduse hindamisel käsitsi ülevaatust siiski täies mahus asendada. Peab arvestama plagiaatide võimalikkusega ja ka koodi loetavusega, millele kontrollprogramm ei pruugi olla mõeldud hinnangut andma. Vaatamata sellele, et programmeerimisse sissejuhatavad kursused on mõeldud tutvustama õppijale programmeerimiskeele olemust, on oluline pöörata tähelepanu ka koodi loetavusele [14]. Ühe koodi loetavusele pühendatud uurimistöö [15] autorid väidavad, et programm peab olema kirjutatud nii, et teine inimene saaks sellest võimalikult kiiresti aru. Selline lähenemine on ka kõigil tarkvaraarendusega tegelevatel ettevõtetel [14], mistõttu on kasulik varakult ära õppida arusaadava koodi kirjutamine.

Automaatkontrolle kasutades peab arvestama ka potentsiaalsete ohtudega, millest kirjutab Ala-Mutka [16] oma uurimistöös: tudengi programm võib sisaldada pahavara, mis võib kontrollija arvutile liiga teha, näiteks kustutada faile. Lisaks võib ebakorrektne programm kasutada ootamatult palju mälu või protsessori aega, mis segab parasjagu arvutis jooksvate programmide toimimist. Nende probleemide vältimiseks soovitab Ala-Mutka kasutada isoleeritud keskkonda tudengite lahenduste jooksutamiseks.

2. Automaatkontrollid Tartu Ülikooli programmeerimise kursustel

Tartu Ülikooli programmeerimise kursustel on tudengite tööde kontrollimiseks kasutusel automaatkontrollid, mis säästavad kontrollija aega ja annavad lahendajale kiiret tagasisidet ning seeläbi toetavad õppimisprotsessi. Sellest hoolimata vaadatakse lahendused läbi ka käsitsi, et veenduda töö autentsuses või tuvastada vigu, mida automaatkontroll ei märka. Sellised kursused on näiteks "Programmeerimine LTAT.03.001" [3] ja "Algoritmid ja andmestruktuurid LTAT.03.005" [17], kus esimeses kasutatakse programmeerimiskeelena Pythonit ja viimases Javat.

Ülesannetes, mille lahendustele on koostatud automaatkontrollid, saab üldjuhul iga tudeng iseseisvalt oma lahendust katsetada. Enamasti kuvavad automaattestid kasutatud sisendit, saadud väljundit ning lühikest sõnumit, mis viitab kas testi läbimisele või läbikukkumisele. Sellega on tudeng teadlik, kas tema lahendus on korrektne või vajab veel täiendusi. Joonisel 1 on kujutatud aine Algoritmid ja andmestruktuurid 4. kodutöö automaatkontrollide väljundit keskkonnas Moodle VPL (Virtual Programming Lab).



Joonis 1. Aine "Algoritmid ja andmestruktuurid" 4. kodutöö automaatkontrollid (2023)

Automaatteste on aastate jooksul koostatud paljude programmeerimisülesannete jaoks, ent praktikumides lahendatavad harjutused võivad vajada tudengitelt endiselt teatud määral käsitsi testimist. Seda olukorra saab leevendada, luues täiuslikud automaatkontrollid ka praktikumiülesannetele, pakkudes igale tudengile kiiret ja detailset tagasisidet lahendusele ning innustades iseseisvamat õppimisprotsessi. Selle tulemusena poleks õpilastel vaja nii palju abi küsida, mis omakorda vähendaks praktikumijuhendajate töökoormust.

Teisest küljest peab automaatkontrollidega ülesannete puhul olema aga tähelepanelik, sest need ei pruugi alati olla täiuslikud. Näiteks võib ülesandel olla ainult üks test, mis hindab vaid ühte võimalikku lahenduskäiku, ning on oht, et tudengi lahendus ei tööta korrektselt mõne teise sisendiga. Samuti on võimalik, et automaatkontroll ei ole mõeldud tuvastama kõiki ülesande kirjelduses defineeritud eeldusi. Näiteks kui ülesande kirjelduses on tingimus, et programm peab kasutama rekursiooni, aga automaatkontroll kontrollib ainult vastust ja võimalikku rekursiooni mittekasutamist see ei tuvasta.

2.1 Kursus “Programmeerimine”

Käesolevas bakalaureusetöös on põhifookuses automaatkontrollide koostamine Tartu Ülikooli aine “Programmeerimine LTAT.03.001” [3] praktikumiülesannetele. Sellel kursusel tutvutakse programmeerimiskeelega Python ning kursuse läbinuil on teadmised põhilistest programmeerimise konstruktsioonidest, algoritmidest ja andmetüüpidest ning -struktuuridest. Kursuse ülesehitus sisaldab nii iseseisvat tööd kui ka kontaktõpet, mis eeldab, et enne praktikumi tutvub tudeng materjaliga ja oskab tunnis käsitletavatel teemadel kaasa rääkida. Kuna tudeng peab lahendama ülesandeid iseseisvalt, on automaatkontrollid vajalikud, sest nende abil saab tudeng tagasisidet, mis näitab, kas ta on õigel teel. Tabelis 1 on esitatud 2023. aasta kursuse struktuur, mis näitab, milliseid teemasid käsitletakse erinevatel nädalatel.

Tabel 1. Kursuse “Programmeerimine” ülesehitus

1. Muutuja ja avaldis
2. Tingimuslause
3. Funktsioon
4. Korduslause
5. Lihtsam failitöötlus. Sõned

6. Kontrolltöö 1
7. Järjend
8. Järjend 2
9. Kahekordne tsükkel
10. Andmestruktuurid
11. Andmestruktuurid 2
12. Kontrolltöö 2
13. Objektorienteeritud programmeerimine
14. Objektorienteeritud programmeerimine 2
15. Rekursioon

Möödunud sügisel tuvastati kursusel mitu ülesannet vigaste automaatkonrollidega. Puudused tulenesid kursuse üleminekust uuele platvormile ja uut tüüpi automaatkontrollidele, mida käsitletakse detailsemalt järgmistes peatükkides.

2.2 Keskkond Lahendus

Õppeaastani 2022/2023 paiknesid kursuse automaatkontrollid Moodle keskkonnas, kuid eelmise aasta sügisel mindi üle platvormile Lahendus, kus enamik Python Graderis kirjutatud automaatkontrolle asendati uute testidega TSL-keeles (vt peatükki 3).

Lahendus on programmeerimise õpetamiseks ja õppimiseks loodud platvorm, kus saab luua ja hallata kursuseid ning koostada automaatkontrolle programmeerimisülesannete lahendustele [18]. Keskkond on avatud lähtekoodiga ja selle kasutamine ning täiendamine on kättesaadav kõigile – iga haridusasutus saab keskkonda kohandada vastavalt vajadustele, ning nad ei pea kulutama raha ega aega uue süsteemi väljatöötamiseks ning selle ülalpidamiseks. Käesoleval õppeaastal kasutatakse Lahenduse keskkonda enam kui 100 koolis erinevates õppeainetes ja valikkursustes[1].

Üleminek uuele platvormile oli tingitud mitmest olulisest eelisest Moodle'i keskkonna ees, millest ühed kaalukamad on mugavam kursuse korraldamise viis ja lihtsam

[1] Juhendajalt saadud andmed.

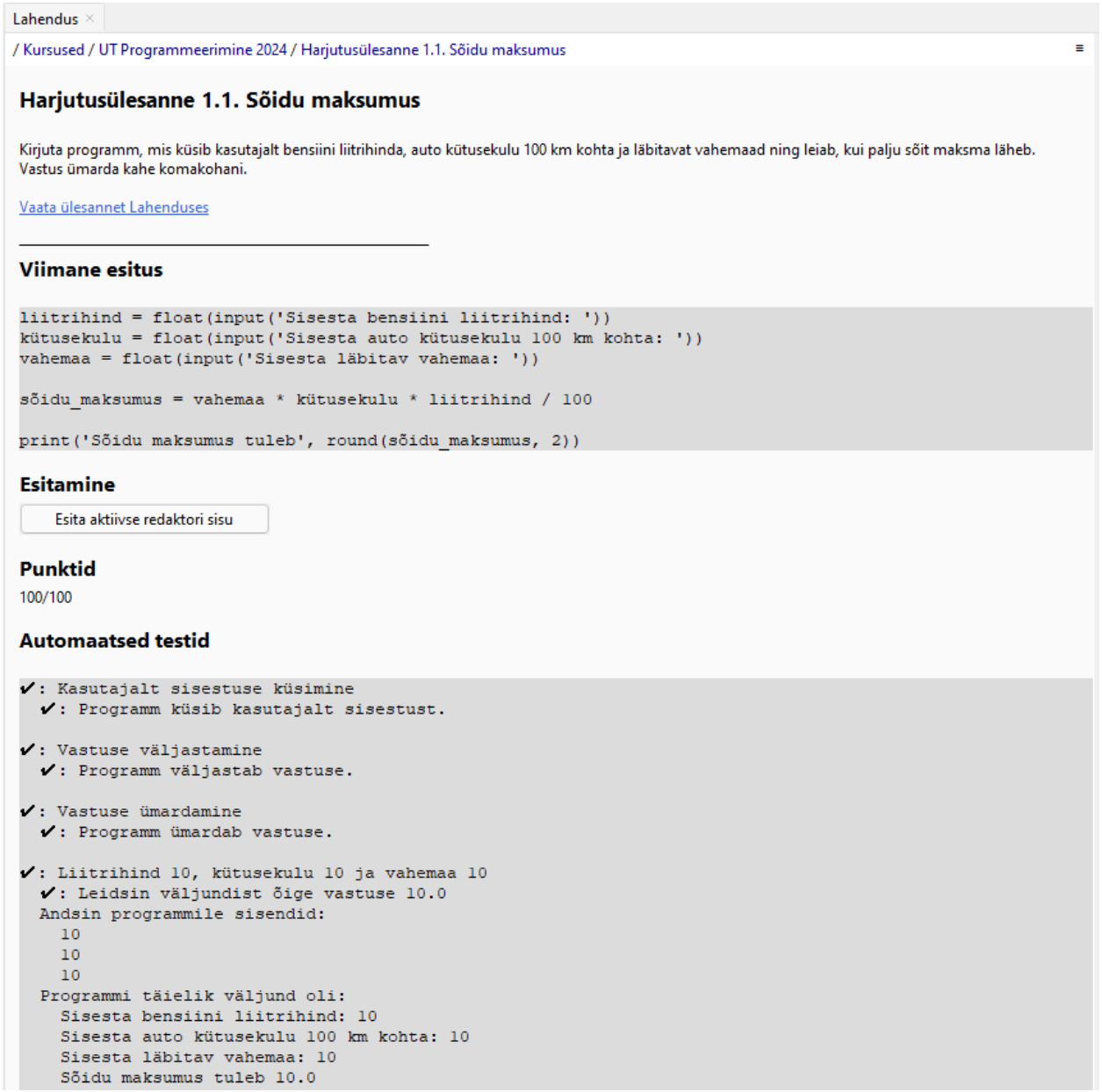
automaatkontrollide loomine ning haldamine. Erinevalt Moodle'i keskkonnast saab Lahenduses salvestada kõik kursuse harjutused üheskoos automaatkontrollidega ülesandekogusse, kust saab ülesandeid lisada ükskõik kui paljudele kursustele [4].

Lisaks eelnevale on Lahendusel ka nägusam kasutajaliides ning töö kirjutamise hetkel arendab Lahenduse meeskond täieulatuslikku funktsionaalsust, millega saab koostada automaatkontrolle otse kasutajaliideses. Eesmärk on võimaldada kõigil õppejõududel luua ja hallata automaatkontrolle ilma süvenemata programmeerimiskeelde, milles testide sisu kirja pannakse. Hetkeseisul on Lahenduse keskkonnas automaatkontrollide koostamise võimalused järgmised:

1) TSL – Pythoni keeles ülesannetele automaatkontrollide defineerimise märgistuskeel;
2) Silmused PostgreSQL – SQL-keeles ülesannetele automaatkontrollide koostamise moodul;
3) Python Grader – 2013. aastal Karl-Aksel Puulmanni ja Aivar Annamaa loodud süsteem, mille abil saab koostada automaatkontrolle programmeerimisülesannetele Pythonis;
4) Tkinter – pildituvastuse moodul graafilise väljundiga ülesannete hindamiseks.

Muuhulgas on Lahenduse keskkond aines "Programmeerimine" eelistatud ka integratsiooni tõttu arenduskeskkonnaga Thonny [19]. Thonny on Tartu Ülikoolis Aivar Annamaa loodud rakendus, mis on eelkõige mõeldud programmeerimiskeele Python õppimiseks. Thonny on algajasõbralik ning pakub nn samm-sammult programmi töökäigu visualiseerimise võimalust. See funktsioon toetab õppimisprotsessi, aidates kasutajal paremini mõista programmi loogikat ja kiiremini leida vigu. Alates 2021. aasta sügisest on õppijatel võimalus paigaldada Thonnysse Lahenduse pistikprogamm[2] [20], kust kasutaja näeb kursuse kõigi ülesannete kirjeldusi ning saab esitada oma koodi otse Thonny rakendusest Lahenduse keskkonda ja saada tagasisidet automaatkontrollide väljunditest. Järgneval joonisel on toodud vaade sügissemestri aine "Programmeerimine" esimese nädala esimesest praktikumiülesandest Thonny pistikprogrammis Lahendus.

[2] Lisandprogramm, mis on hõlpsalt installitav ja laiendab senise programmi võimalusi [21].



Lahendus

/ Kursused / UT Programmeerimine 2024 / Harjutusülesanne 1.1. Sõidu maksumus

Harjutusülesanne 1.1. Sõidu maksumus

Kirjuta programm, mis küsib kasutajalt bensiini liitrihinda, auto kütusekulu 100 km kohta ja läbitavat vahemaad ning leiab, kui palju sõit maksma läheb. Vastus ümarda kahe komakohani.

Vaata ülesannet Lahenduses

Viimane esitus

```
liitrihind = float(input('Sisesta bensiini liitrihind: '))
kütusekulu = float(input('Sisesta auto kütusekulu 100 km kohta: '))
vahemaa = float(input('Sisesta läbitav vahemaa: '))

sõidu_maksumus = vahemaa * kütusekulu * liitrihind / 100

print('Sõidu maksumus tuleb', round(sõidu_maksumus, 2))
```

Esitamine

Esita aktiivse redaktori sisu

Punktid

100/100

Automaatsed testid

```
✔: Kasutajalt sisestuse küsimine
  ✔: Programm küsib kasutajalt sisestust.

✔: Vastuse väljastamine
  ✔: Programm väljastab vastuse.

✔: Vastuse ümardamine
  ✔: Programm ümardab vastuse.

✔: Liitrihind 10, kütusekulu 10 ja vahemaa 10
  ✔: Leidsin väljundist õige vastuse 10.0
  Andsin programmile sisendid:
    10
    10
    10
  Programmi täielik väljund oli:
    Sisesta bensiini liitrihind: 10
    Sisesta auto kütusekulu 100 km kohta: 10
    Sisesta läbitav vahemaa: 10
    Sõidu maksumus tuleb 10.0
```

Joonis 2. Ülesande vaade Thonny pistikporgrammis Lahendus

3. TSL

TSL (*Test Specific Language*) on Tartu Ülikooli doktorant Eerik Muuli loodud deklaratiivne märgistuskeel, mida kasutatakse automaatkontrollide koostamiseks [5]. Märgistuskeele abil saab defineerida juhiseid, näiteks sisendeid, väljundeid ja veateateid, mida peab automaatkontroll kasutama koodi hindamisel. Peale juhiste kirja panemist kontrollib TSL JSON-faili[3] süntaksianalüsaator faili korrektsust, mille järel genereerib kompilaator[4] automaatkontrolli Pythoni faili [5].

3.1 TSL-keele eelised

TSL lahendab eelnevalt kasutusel olnud automaatkontrollide loomise tööriista Python Graderi puudused. Vana süsteemi suurimaks puuduseks oli liigne keerukus, mille tõttu tegelesid kontrollprogrammi kirjutamisega vaid vähesed Tartu Ülikooli programmeerimise õppejõud ning ka neil võis tihti ühe automaatkontrolli koostamine kujuneda ajakulukaks [5]. Sellest tulenevalt on Eerik Muuli märgistuskeele arendamises keskendutud lihtsale koodi ülesehitusele ja süntaksile, mis ei valmistaks raskusi õppejõul automaatkontrollide kirjutamisel ning olemasolevate testide täiendamisel. TSLi eelis seisneb ka selles, et erinevalt Python Graderist kõrvaldab TSL vajaduse automaatkontrolli fail iseseisvalt genereerida – vajalik on ainult juhiste faili loomine, mille järgi koostab süsteem testfaili. Selline lähenemine kiirendab automaatkontrollide loomist kordades. Hetkeseisul on TSLi abil võimalik defineerida automaatkontrolle üksnes ülesannetele Pythonis, kuid märgistuskeelt on võimalik täiendada selliselt, et saaks kirjutada teste ka teistes programmeerimiskeeltes ülesannetele [5].

TSLi eeliste paremaks mõistmiseks on alljärgnevatel joonistel 3 ja 4 välja toodud Python Graderis ja TSL-keeles koostatud automaatkontrollide võrdlus aine “Programmeerimine” kolmanda nädala teise praktikumiülesande näitel [22]. Ülesande kirjeldus on järgmine:

> “Suurte failide puhul ei ole mugav arvestada faili suurust baitides. Kirjuta funktsioon *teisenda*, mis võtab argumendiks baitide arvu ja tagastab sõne, kus baidid on teisendatud sobivatesse ühikutesse, sobiva täpsusega. Näiteks

[3] Lihtne andmevahetusvorming, mis on hõlbus inimlugemiseks ja -kirjutuseks [21].
[4] Programm, mis transleerib lähtekoodi objektkoodiks [23].

teisenda(37065) võiks tagastada sõne *"36.2 KB"*. Teisendusreeglid on: 1 KB = 1024 B, 1 MB = 1024 KB jne. Vastus tagasta ühe komakoha täpsusega.”

```
from grader import *
import random

FUNKTSIOONI_NIMI = 'teisenda'
FUNKTSIOONI_ARGUMENTE = 1

def lahendus(suurus):
    if suurus < 1024:
        return str(suurus) + " B"
    elif suurus < 1024 ** 2:
        uussuurus = round(suurus / 1024, 1)
        return str(uussuurus) + " KB"
    elif suurus < 1024 ** 3:
        uussuurus = round(suurus / 1024 ** 2, 1)
        return str(uussuurus) + " MB"
    else:
        uussuurus = round(suurus / 1024 ** 3, 1)
        return str(uussuurus) + " GB"


@test
@expose_ast
@set_description(f"Leidub funktsioon '{FUNKTSIOONI_NIMI}'")
def test_lahendus_py_fail(m, AST):

    assert hasattr(m.module, FUNKTSIOONI_NIMI), f"Ei leidnud funktsiooni nimega '{FUNKTSIOONI_NIMI}'"

    found_func = False
    func_print_count = 0
    for node in ast.walk(AST):
        if isinstance(node, ast.FunctionDef) and \
                node.name == FUNKTSIOONI_NIMI:

            assert not found_func,
            "Leidsin programmist mitu funktsiooni '{}' definitsiooni. Funktsioone tuleb defineerida vaid üks kord!"
            .format(
                    FUNKTSIOONI_NIMI)
            found_func = True

            assert len(
                node.args.args) == FUNKTSIOONI_ARGUMENTE,
            "Funktsioon '{}' peaks olema defineeritud {} argumendiga, praegu leidsin definitsioonist {} argumenti."
            .format(
                FUNKTSIOONI_NIMI, FUNKTSIOONI_ARGUMENTE, len(node.args.args))

            for func_node in ast.walk(node):
                if isinstance(func_node, ast.Call) and \
                        hasattr(func_node, 'func') and \
                        hasattr(func_node.func, 'id') and \
                        func_node.func.id == 'print':
                    raise AssertionError(f"Funktsiooni '{FUNKTSIOONI_NIMI}' sees pole vaja midagi väljastada!")

@test
@expose_ast
@set_description(f"Test 1: Väljund baitides")
def test_lahendus_py_fail(m, AST):
    sisend = 1023

    korrektne_lahendus = lahendus(sisend)

    esitatud_lahendus = getattr(m.module, FUNKTSIOONI_NIMI)(sisend)

    assert esitatud_lahendus == korrektne_lahendus,
    f"Ootasin vastuseks '{korrektne_lahendus}' aga sain '{esitatud_lahendus}'"
```

Joonis 3. Python Graderi automaatkontroll

```
{
    "language": "python3",
    "validateFiles": true,
    "requiredFiles": [
        "lahendus.py"
    ],
    "tslVersion": "1.0",
    "tests": [
        {
            "type": "function_execution_test",
            "name": "Test 1. 1023",
            "pointsWeight": 1,
            "visibleToUser": true,
            "id": 1,
            "functionName": "teisenda",
            "functionType": "FUNCTION",
            "arguments": [
                "1023"
            ],
            "returnValueCheck": {
                "returnValue": "'1023 B'",
                "beforeMessage": "",
                "passedMessage": "Funktsioon tagastab oodatud väärtuse 1203 B",
                "failedMessage": "Funktsioon ei tööta nii nagu vaja, funktsioon peab tagastama arvu 1203 B"
            }
        }
    ]
}
```

Joonis 4. TSL-keeles koostatud automaatkontroll

Mõlemal joonisel kontrollivad automaattestid funktsiooni *teisenda* tagastuväärtust sisendiga 1023, mis ülesande püstituse järgi peab olema sõne “1023 B”. Automaatkontrolle võrreldes on näha, et TSL-keeles kood on lühem, kergemalt loetavam ja arusaadavama süntaksiga.

3.2 TSLi automaatkontrollid

Selles peatükis tutvustatakse lähemalt TSL-keele omadusi. Kuna praeguseks hetkeks puudub terviklik dokumentatsioon TSLi kohta, siis eesmärk on pakkuda väike juhend neile, kes tulevikus tahavad koostada automaatkontrolle TSLi abil või lihtsalt keelest aru saada. Siin peatükis tutvustatakse ainult kahte testitüüpi ning ammendav dokumentatsioon TSLi funktsionaalsusest on toodud töö lisas 2.

Iga TSL-keeles koostatud fail on kindla ülesehitusega: kõigepealt defineeritakse kontrollitava koodi programmeerimiskeel ja versioon, seejärel määratakse TSLi versioon, peale mida pannakse kirja automaatkontrollid järjendisse *tests* [5]. Joonisel 4 sisaldab automaatkontrollide järjend ainult ühte funktsiooni käivitamise testi, mis on ühtlasi peale programmi käivitamise testi enim kasutatud testide tüüp kursuse “Programmeerimine” ülesannetes.

Alloleval joonisel 5 on näide kursuse “Programmeerimine” esimese nädala esimese praktikumiülesande [24] programmi käivitamise testist. Ülesande kirjeldus:

> “Kirjuta programm, mis küsib kasutajalt bensiini liitrihinda, auto kütusekulu 100 km kohta ja läbitavat vahemaad ning leiab, kui palju sõit maksma läheb. Vastus ümarda kahe komakohani.”


```
"type": "program_execution_test",
"name": "Liitrihind 10, kütusekulu 10 ja vahemaa 10",
"pointsWeight": 1,
"visibleToUser": true,
"id": 8724,
"standardInputData": [
    "10",
    "10",
    "10"
],
"genericChecks": [
    {
        "id": 8733,
        "checkType": "ALL_OF_THESE",
        "expectedValue": [
            "10"
        ],
        "dataCategory": "CONTAINS_NUMBERS",
        "beforeMessage": "",
        "passedMessage": "Leidsin väljundist õige vastuse 10.0",
        "failedMessage": "Ei leidnud programmi väljundist oodatud tulemust 10.0"
    }
]
```


Joonis 5. Programmi käivitamise testi näide

Joonisel 5 kujutatud programmi käivitamise test kontrollib, kas programmi väljund sisaldab õiget arvu. Test jooksutab koodi sisenditega: 10 (liitrihind); 10 (auto kütusekulu 100 km kohta); 10 (läbitav vahemaa) ja kontrollib, kas programmi väljundis leidub arv 10. Järgnevalt on toodud joonistel 4 ja 5 kujutatud testide detailne ülevaade.

Esimesed viis välja kattuvad mõlemal testitüübil:

1) *type* - automaatkontrolli tüüp;
2) *name* - kasutajale kuvatav automaatkontrolli nimetus;
3) *pointsWeight* - automaatkontrolli punktide osakaal;

4) *visibleToUser* - õpilasele automaatkontrolli nähtavust määrav väli;
5) *id* - automaatkontrolli identifikaator.

Programmi käivitamise testi väljad:

1) *standardInputData* - järjend programmile etteantavatest sisenditest;
2) *genericChecks* - järjend alamkontrollidega;
3) *id* - alamkontrolli identifikaator;
4) *expectedValue* - oodatud väärtuste järjend;
5) *checkType* - järjendi *expectedValue* elementide sisaldavuse kontrolli programmi väljundis määrav väli. Väärtus *ALL_OF_THESE* kontrollib, et kõik väärtused järjendist *expectedValue* esineksid programmi väljundis;
6) *dataCategory* - järjendi *expectedValue* elementide tüüpi määrav väli. Väärtus *CONTAINS_NUMBERS* määrab, et oodatud väärtuste järjendi elemendid on arvud.

Funktsiooni käivitamise testi väljad:

1) *functionName* - kontrollitava funktsiooni nimi;
2) *functionType* - funktsiooni tüüp, mis saab olla kas funktsioon või meetod;
3) *arguments* - funktsiooni argumentide järjend;
4) *returnValueCheck* - tagastusväärtuse ja teadete komplekt;
5) *returnValue* - oodatav tagastusväärtus.

Lõpus samuti kattuvad väljad mõlemal testitüübil:

6) *beforeMessage* - enne testi käivitamist kuvatav teade;
7) *passedMessage* - testi eduka läbimise järel kuvatav teade;
8) *failedMessage* - testi ebaeduka läbimise järel kuvatav teade.

4. Metoodika

Selles peatükis antakse ülevaade kursuse “Programmeerimine” harjutusülesannete automaatkontrollide analüüsimise protsessist.

Vastavalt juhendajalt saadud andmetele tuvastati möödunud sügissemestri jooksul hulk ülesandeid vigaste automaatkontrollidega. Kuna puudus nimekiri ebatäiuslike automaatkontrollidega ülesannetest, vaadati läbi kõik kursuse praktikumiülesanded, näidiskontrolltööd ja eksami näidised, mis paiknevad ülesandekogus “UT PROGRAMMEERIMINE 2023” keskkonnas Lahendus. Analüüsi eesmärk oli tuvastada puudulikke automaatkontrolle, et hiljem selle info põhjal teste parandada ja täiendada. Uuendatud automaatkontrollidega ülesanded lisati uude ülesandekogusse “UT PROGRAMMEERIMINE 2024”.

Automaatkontrollide analüüsimine jagunes neljaks etapiks:

1. Ülesande lahendamine ja programmi esitamine keskkonnas Lahendus.
2. Olemasolevate automaatkontrollide väljundite uurimine – testi eduka läbimise tulemusena liiguti otse viimase etapi juurde. Veateadete esinemisel loeti veel kord läbi ülesande kirjeldus ning otsiti veakohti valminud lahenduses. Kui veenduti, et lahendus on korrektne ja automaatkontroll vigane, liiguti järgmise etapi juurde. Lahenduste valideerimise viise vaadeldakse lähemalt peatükis 5.1.
3. Leitud vigade liigitamine - iga defektse automaatkontrolli leidmisel oli tarvis määrata põhjus, miks see vigane oli. Selleks võis olla kas automaatkontrolli koostaja hooletus või viga TSL-keele defineerimises. Kuna TSL on kursusel “Programmeerimine” vaid aasta kasutusel olnud abivahend [4], mida pidevalt täiendatakse, võib esineda olukordi, kus vähese testimise tõttu esineb loogikavigu keele olemuse kirjapanekus. Näited TSLi funktsionaalsuse vigadest on toodud peatükis 5.2.
4. Automaatkontrollide katvuse hindamine – iga ülesande puhul tuli veenduda, et testid katavad kõiki ülesande kirjeldusele vastavaid stsenaariumeid. Kui leiti, et mõne ülesande puhul ei kontrollita kõiki püstitatud nõudeid, lisati puuduolevad testid.

Lisaks eeltoodule märgati 2023. aasta esimese neljateistkümne nädala automaatkontrollides paljude ülesannete jaoks tarbetuid väljasid, mis ei mõjutanud mingil moel lahenduse hindamist. Näiteks sisaldasid kõik programmi käivitamise testid sisend- ja väljundfailide

kontrolle, isegi kui ülesanne ei nõudnud failidega töötamist. Selle tõttu uuendati teste selliselt, et jäeti alles need väljad, mis olid vajalikud konkreetse ülesande lahenduse kontrollimisel, koos kolme mittekohustusliku testiväljaga, mille abil saab määrata testi kasutajale nähtavust (*visibleToUser*), testi punktide osakaalu (*pointsWeight*) ning testi nime (*name*). Selline automaatkontrollide defineerimise viis vähendab automaatkontrolli faili ridade arvu umbkaudu 30% võrra, mille tulemusena kiirenevad uute testide loomise ja olemasolevate testide uuendamise protsessid.

Peale testide ülesehituse uuendamise pöörati tähelepanu ka ülesannete kirjeldusele, kuna 2023. aasta üleandekogus harjutuste puhul oli tihti toodud näide programmi või funktsiooni tööst, mis oli mõnikord kasutajale kohe nähtav, teinekord aga peidetud selliselt, et näite nägemiseks tuli teha nupuvajutus ülesande kirjeldusele. Kursuse vastutava õppejõuga konsulteerides otsustati ühtlustada ülesannete kirjeldused, muutes näited programmi ja funktsiooni tööst kasutajale nähtavaks lisa nupuvajutusi tegemata.

5. Tulemused

Bakalaureusetöö raames valmis keskkonnas Lahendus uus kursus “UT Programmeerimine 2024”, mis sisaldab 71 programmeerimisülesannet, millest 62 on praktikumiülesanded, 6 näidiskontrolltööde ülesanded ja 3 eksami näidisülesanded. Töö käigus lahendati läbi kõik harjutused, parandati üheksa ülesande automaatkontrollid, tehti juurde 19 automaatkontrolli, ühtlustati 64 ülesande automaatkontrolli ülesehitust ning 32 ülesande kirjeldust. Muuhulgas koostati töö käigus ka esimene terviklik dokumentatsioon TSL-keele kohta, mis on leitav kirjaliku töö lisas 2.

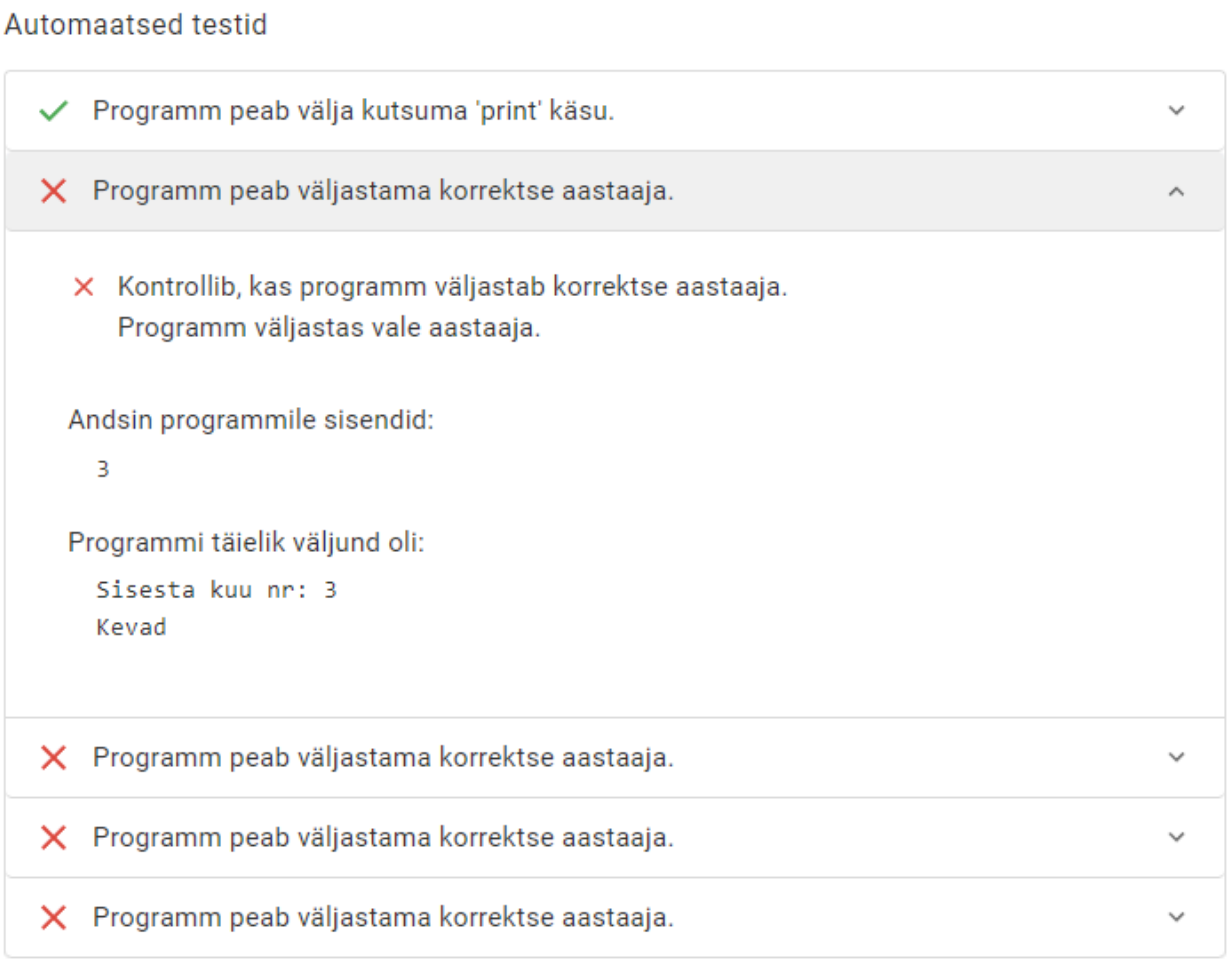
Valminud automaatkontrollide korrektsuses veendumiseks jooksutati teste kõigi võimalike ülesannete nõuetele vastavate sisendite kombinatsioonidega. Koostatud lahendused, mis loodi automaatkontrollide valideerimiseks, läbisid edukalt kõik testid kõikides ülesannetes. Selline valideerimisprotsess annab kindluse, et automaatkontrollid tagavad täpse ja järjepideva tagasiside tudengite esitatud lahendustele.

Jägnevalt antakse detailsem ülevaade leitud puudulikest ja ebatäiuslikest automaatkontrollidest, tuuakse näiteid ülesannetest, analüüsitakse vigaste automaatkontrollide põhjusi ja parandusi ning vaadeldakse puudujääke TSL-keeles.

5.1 Lüngad TSL-keeles

Analüüsides 2023. aasta ülesandekogu automaatkontrolle ei tuvastatud ühtegi automaatkontrolli, mis oleks vigane üksnes testi koostaja arvutusvea või ebakorrektse programmi tõttu - puudused olid tingitud hoopis TSL-keele omadustest. Järgmises loetelus on toodud ülesanded, kus veakohad automaatkontrollides tulenesid TSL-keele definitsioonist: 1.5, 2.1, 11.2. Märgistuskeele puudujääki analüüsitakse harjutuse 1.5 [24] näitel. Ülesande kirjeldus:

> “Keegi üritas kirjutada programmi, mis küsib kasutajalt kuu numbri ja väljastab väikeste tähtedega ekraanile selle aastaaja nime, kuhu see kuu kuulub. Näiteks kui kasutaja sisestab 3, peaks ekraanile ilmuma kevad.”



Joonis 7. Ülesande 1.5 2023. aasta automaatkontrollide väljundid

Joonisel 7 on näha viis testi, millest autori lahendus läbis edukalt ainult esimese ning ülejäänud kontrollide puhul kuvatakse viga teatega, et on väljastatud vale aastaaeg. Vaadates teise automaatkontrolli väljundit, joonisel 7, on näha, et programmi väljund sisendiga 3 on sõne “Kevad”, mis vastab ülesande püstitusele. Kontrollprogrammi veidra käitumise mõistmiseks uuriti automaatkontrolli ülesehitust. Joonisel 8 on kuvatud ülesande 1.5 TSL-faili osa, kus on defineeritud programmi käivitamise test, mis jooksutab kasutaja koodi sisendiga 3.

```
"type": "program_execution_test",
"name": "Programm peab väljastama korrektse aastaaja.",
"pointsWeight": 1.0,
"visibleToUser": true,
"inputs": null,
"passedNext": null,
"failedNext": null,
"id": 2,
"standardInputData": [
    "3"
],
"inputFiles": [
],
"genericChecks": [
    {
        "id": 9001,
        "checkType": "ANY_OF_THESE",
        "nothingElse": null,
        "expectedValue": [
            "kevad",
            "Kevad"
        ],
        "elementsOrdered": false,
        "dataCategory": "EQUALS",
        "beforeMessage": "Kontrollib, kas programm väljastab korrektse aastaaja.",
        "passedMessage": "Programm väljastas õige aastaaja.",
        "failedMessage": "Programm väljastas vale aastaaja."
    },
    {
        "id": 9002,
        "checkType": "NONE_OF_THESE",
        "nothingElse": null,
        "expectedValue": [
            "suvi",
            "Suvi",
            "talv",
            "Talv",
            "sügis",
            "Sügis"
        ],
        "elementsOrdered": false,
        "dataCategory": "EQUALS",
        "beforeMessage": "Kontrollib, et programm ei väljastaks vale aastaaega",
        "passedMessage": "Programm väljastas õige aastaaja.",
        "failedMessage": "Programm väljastas vale aastaaja."
    }
],
```


Joonis 8. Ülesande 1.5 2023. aasta teine automaatkontroll

Joonisel 8 asuv automaatkontroll koosneb kahest alamtestist, mis on defineeritud järjendis *genericChecks*, kus esimene test aktsepteerib väärtusi “kevad” suure või väikese algustähega ning teine kontrollib, et programmi väljundis puuduksid aastaajad, mis pole kevad.

Automaatkontrolli käitumise parema mõistmise eesmärgil katsetati esimese alamkontrolli välja *dataCategory* väärtuse muutmist. Vaadeldavas automaatkontrollis nõuab väärtus *EQUALS*, et programmi väljund võrduks ükskõik mis elemendiga järjendist *expectedValue*, kuid, andes välja *dataCategory* väärtuseks *CONTAINS_LINES* või *CONTAINS_STRINGS*, läbis autori kood automaatkontrolli edukalt. Kõnealuses testis kontrollib *CONTAINS_LINES*, et programmi väljundis leiduks rida, mis võrdub ükskõik mis elemendiga järjendist *expectedValue*. Katsetades niiviisi automaatkontrolli käitumist, jõuti järeldusele, et viga testis oli põhjustatud sellest, et automaatkontroll arvestab Pythoni sisestamise käsu *input* sisu kuvamist ka programmi väljundiks ehk joonisel 7 teise automaatkontrolli väljundis olevat rida “Sisesta kuu nr: 3” loetakse samuti väljundiks, kuid tegelikult tuleks seda rida ignoreerida. Kuna möödunud aasta automaatkontrollide loomisel kasutati joonisel 8 kuvatud konstruktsiooni, võib oletada, et TSL-keeles tehti muudatusi, mille tulemusena ei erista automaatkontrollid enam programmi sisestuskäske ja programmi väljundeid.

Lisaks eeltoodule leiti kursuse esimese näidiskontrolltöö esimest varianti [25] lahendades veel üks TSLi nõrkus. Ülesande püstituse järgi peab programm lugema etteantud failist lindude andmed ning lõpuks salvestama töödeldud andmed faili nimega “vaadeldud.txt”. Ülesande automaatkontrolli väljundfaili testis kontrollitakse, kas programm loob nõutud faili vastava sisuga. Automaatkontrolli testimise eesmärgil esitati kõigepealt korrektne lahendus ning seejärel programm, mis ei loo väljundfaili. Mingil põhjusel läbisid mõlemad lahendused automaatkontrollid edukalt. Töö kirjutamise hetkel seletust taolisele automaatkontrolli käitumisele ei leitud.

Viimane lünk TSL-keeles, mis avastati praktilise töö käigus, on automaatkontrolli välja *visibleToUser* puudulik funktsionaalsus. Määrates väljale väärtus *false*, peaks olema automaatkontroll õpilase eest ära peidetud, kuid testimisel leiti, et automaatkontrolli nähtavus ei muutu.

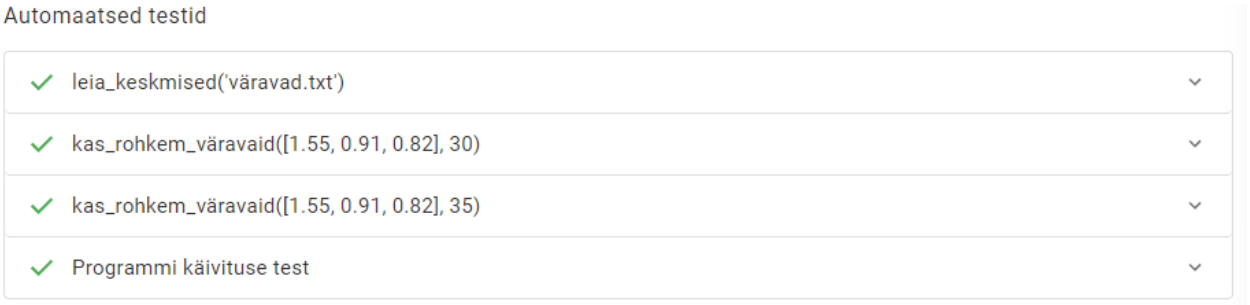
Kõikidest TSL-keele puudustest informeeriti märgistuskeele arendajat.

5.2 Ebatäiuslikud automaatkontrollid

Vastavalt peatükis 4 kirjeldatud automaatkontrollide analüüsimisele hinnati iga ülesande juures automaattestide katvust veendumaks, et kõik ülesande kirjelduses püstitatud nõuded oleksid programmis täidetud. Selle tulemusena täiendati ülesannete 2.3, 2.4, 2.6, 7.4, 9.1, 11.2 ja 12.1 automaatkontrolle. Lisaks sellele lisati kursuse esimese nädala praktikumiülesannetele automaatkontrollid sisestamise, väljastamise ja ümardamise käskude kasutamise kontrollid ja ülesannetele, kus olid mittekohustuslikud lisanuputamised, tehti juurde automaatkontrollid punktikaaluga 0. Selline ülesanne on näiteks 7.5.

Järgnevalt on toodud ülevaade 2023. aasta ülesandekogu 12. nädala näidiskontrolltöö esimesest variandist [26], kus automaatkontrollid ei kata kõiki ülesande eeldusi. Ülesande kirjelduse osa, mida automaatkontrollid täies mahu ei kata, on järgmine:

> "Failis *väravad.txt* on kirjas jalgpalliliiga mängudes löödud väravate arvud. Ühel real on mängija löödud väravad mängudes tühikutega eraldatult. Kõigis ridades on samapalju arve. Mängijate arv ja vooru mängude arv pole ette teada. Kokku mängitakse liigas 38 mängu. Koosta funktsioon *kas_rohkem_väravaid*, mis võtab argumendiks seni mängitud mängudes löödud väravate keskmiste tulemuste järjendi ning hooajal oodatavate väravate arvu (täisarv). Funktsioon tagastab tõeväärtuse True, kui kõikide mängijate seni löödud väravate keskmised on sama suured või suuremad kui oodatav keskmine väravate arv (oodatava keskmise väravate arvu saame hooajal oodatavate väravate arvu jagamisel 38-ga). Vastasel juhul tagastab funktsioon tõeväärtuse False."



Joonis 9. Ülesande 12.1 2023. aasta automaatkontrollid

Joonisel 9 on kuvatud neli automaatkontrolli, millest esimene kontrollib funktsiooni *leia_keskmised* tööd, järgmised kaks hindavad funktsiooni *kas_rohkem_väravaid* ning viimane on kogu programmi tööd kontrolliv test. Vastavalt ülesande kirjeldusele peab funktsioon *kas_rohkem_väravaid* tagastama tõeväärtuse *True*, kui kõikide mängijate seni löödud väravate keskmised on sama suured või suuremad kui oodatav keskmine väravate arv, kuid vaadeldavas automaatkontrollide komplektis ei kaeta esimest juhtu. Sellest tulenevalt võib esitatud programmis kõnealune funktsioon tagastada tõeväärtuse *False*, kui kõikide mängijate seni löödud väravate keskmised on sama suured kui oodatav keskmine väravate arv ja lahenduse eest saadakse ikka täispunktid. Sellise olukorra vältimiseks lisati test, mis kontrollib funktsiooni tagastusväärtust järjendiga [1.2, 1, 1.1] (mängijate seni löödud väravate keskmised väärtused) ja arvuga 38 (hooajal oodatavate väravate arv) ning õigeks vastuseks loetakse ainult tõeväärtust *True*.

6. Arutelu

Selles peatükis arutletakse, millist mõju avaldab valminud bakalaureusetöö programmeerimise kursustele ja TSL-keele arengule.

Lõputöö raames valminud automaatkontrollid edendavad programmeerimise õpetamis- ja õppimisprotsesse, võimaldades nii praktikumijuhendajatel kui ka tunnis osalejatel rohkem toetuda testide tagasisidele ja olla kindlad, et lahendusi kontrollitakse korrektselt ning piisava põhjalikkusega. Samuti on need eeskujuks teiste automaatkontrollide koostamisele tulevikus. Kuna praktikumiülesanded ja näidiskontrolltööde ning näidiseksami ülesanded muutuvad aastate lõikes vähe, jääb selle töö panus oluliseks mitmeks aastaks tulevikus.

Samuti viidi töö käigus läbi ka ulatuslik TSL-keele testimine. Kuna TSLi kui uudse programmeerimise harjutuste automaakontrollimise vahendi eesmärk on toetada programmeerimise õpetamist ja õppimist, on äärmiselt oluline, et see oleks täpne ning usaldusväärne ja nii tudeng kui õpetaja saaksid olla kindlad automaatkontrolli tagasiside korrektsuses. Töö käigus avastatud märgistuskeele vead on suureks abiks keele täiustamisele ja parandamisele. Ühtlasi pakub autor välja TSLi edasiarenduse ideena võimaldada märgistuskeele abil automaatkontrollide loomist ka kursustel, mis kasutavad programmeerimiskeelena Javat, et lihtsustada testide loomist ka nendes ainetes.

Lisaks eeltoodule edastati kursuse “Programmeerimine” vastutavale õppejõule kõikide ülesannete lahendused, millest võivad abi saada nii praktikumide läbiviijad kui ka tudengid juhul, kui peaksid tekkima raskused harjutuste lahendamisel. Peale selle, kuna sügissemestri kursus “Programmeerimine” pole Tartu Ülikoolis ainuke programmeerimise kursus keeles Python, on võimalus kasutada selle aine ülesandeid ja automaatkontrolle ka teistel kursustel, näiteks aines “Programmeerimise alused MTAT.03.236” [27]. Muuhulgas toodi välja ka harjutused, kus ülesande püstitust saaks autori arvates selgemini kirja panna.

Ühtlasi koostati töö raames ka esimene terviklik TSL-keele dokumentatsioon. Kuna töö kirjutamise hetkel oli võimalik TSL-keele kohta infot leida ainult keskkonna Github repositooriumist[5] *easy* [28], kus TSLi olemus defineeritud on, või vaadates teiste inimeste

[5] Füüsiline või virtuaalne hoiukoht, sealhulgas faili või kataloogina [21].

loodud automaatkontrolle, on valminud dokumentatsioon abistav kõigile, kellel on vaja tutvuda TSL-keelega ja koostada või uuendada automaatkontrolle.

Kokkuvõte

Bakalaureusetöö eesmärk oli analüüsida ja täiendada kursuse “Programmeerimine LTAT.03.001” praktikumiülesannete, näidiskontrolltööde ja eksami näidiste automaatkontrolle, et tagada tudengiprogrammide täpset ja põhjalikku kontrollimist ning seeläbi vähendada õppejõudede koormust. Eesmärk sai täidetud 2024. aasta kevadel, kui uuendati vastavad automaatkontrollid, mis paigutati keskkonda Lahendus ülesandekogusse “UT PROGRAMMEERINE 2024” ja kursusele “UT PROGRAMMEERIMINE 2024”.

Lisaks sellele viidi töö käigus läbi TSL-keele põhjalik testimine, mille käigus tuvastati mitmed keele kitsaskohad. Nende puudujääkide parandamine aitab kaasa märgistuskeele täiustamisele, mis omakorda tõstab automaatkontrollide kvaliteeti. Samuti valmis töö raames esimene terviklik TSLi dokumentatsioon, mis on väärtuslikuks abivahendiks neile, kes soovivad tulevikus TSL-keeles automaatkontrolle luua.

Viidatud kirjandus

[1] Sisseastumise infosüsteemi kodulehekülg https://www.sais.ee/ (09.05.2024)

[2] Oidermaa, J., Ülevaade: Eesti ülikoolide populaarsemad erialad, 2023, https://novaator.err.ee/1609026296/ulevaade-eesti-ulikoolide-populaarsemad-erialad (09.05.2024)

[3] Tartu Ülikooli õppeinfosüsteem, kursus "Programmeerimine LTAT.03.001", https://ois2.ut.ee/#/courses/LTAT.03.001/version/700527cd-b682-45a8-254a-b3eaa4a04a0b/details (06.12.2023)

[4] Albre, J., 2023, Tartu Ülikooli kursuse Programmeerimine ülesannete automaatkontrollide uuendamine ja üleviimine keskkonda lahendus.ut.ee, Tartu Ülikooli arvutiteaduse instituudi bakalaureusetöö, https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=77597 (08.05.2024)

[5] Muuli, E., Palm, R., Lepp, M., 2023, Simplifying the creation and maintenance of automated assessments of programming tasks via Test Specific Language, *Proceedings of the 2022 6th International Conference on Education and E-Learning*, 14-20, https://doi.org/10.1145/3578837.3578840

[6] Myers, G. J., Badgett, T., Thomas, T. M., Sandler, C., The Art of Software Testing, 2nd ed., Hoboken, New Jersey: John Wiley & Sons, Inc., 2004

[7] Charitsis, C., Piech, C., Mitchell, J. C., 2022, Simplifying Automated Assessment in CS1, *Proceedings of the 2021 4th International Conference on Education Technology Management*, 226-231, https://dl.acm.org/doi/10.1145/3510309.3510345

[8] Hollingsworth, J., 1960, Automatic graders for programming classes, *Journal on Educational Resources in Computing*, 3:10, 528-529, https://doi.org/10.1145/367415.367422

[9] Douce, C., Livingstone, D., Orwell, J., 2005, Automatic test-based assessment of programming: A review, *Journal on Educational Resources in Computing*, 5:3, 4-es, https://doi.org/10.1145/1163405.1163409

[10] Reek, K. A., 1989, The TRY system -or- how to avoid testing student programs, *Proceedings of the twentieth SIGCSE technical symposium on Computer science education*, 112–116, https://doi.org/10.1145/65293.71198

[11] Jackson, D., Usher, M., 1997, Grading student programs using ASSYST, *Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education*, 335–339, https://doi.org/10.1145/268084.268210

[12] Joy, M., Luck, M., 1998, Effective electronic marking for on-line assessment, *ACM SIGCSE Bulletin*, 30:3, 134–138, https://doi.org/10.1145/290320.283096

[13] Puulmann, K., 2014, Python module for automatic testing of programming assignments, Tartu Ülikooli arvutiteaduse instituudi bakalaureusetöö, https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=40971&year=2014 (08.04.2024)

[14] Setiawan, I., Maryono, D., Basori, B., 2019, The Analysis of Software Source Code Readability: Case Study at Education of Informatics and Computer Engineering Study Program of Sebelas Maret University, *Journal of Informatics and Vocational Education*, https://doi.org/10.20961/joive.v2i1.35695

[15] Johnson, J., Lubo, S., Yedla, N., Aponte, J., Sharif, B., 2019, An Empirical Study Assessing Source Code Readability in Comprehension, *2019 IEEE International Conference on Software Maintenance (ICSM)*, 513-523, https://doi.org/10.1109/ICSME.2019.00085

[16] Ala-Mutka, K. M., 2005, A Survey of Automated Assessment Approaches for Programming Assignments, *Computer Science Education*, 15:2, 83-102, https://doi.org/10.1080/08993400500150747

[17] Tartu Ülikooli õppeinfosüsteem, kursus "Algoritmid ja andmestruktuurid LTAT.03.005", https://ois2.ut.ee/#/courses/LTAT.03.005/version/c8a59cf3-f5bc-3bff-f628-687d9855048b/details (06.12.2023)

[18] Programmeerimise õppimise ja õpetamise keskkonna lahendus.ut.ee kasutustingimused, https://docs.google.com/document/d/1dk1Pp3hXJEX7HllQFdMFo5AXhgzy4zhZv3Qt6-xI_CI/edit#heading=h.df01thmxe31v (29.12.2023)

[19] Annamaa, A., 2015, Introducing Thonny, a Python IDE for Learning Programming, *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, 117-121, https://doi.org/10.1145/2828959.2828969

[20] Keskkonna Lahendus pistikprogramm tarkvaras Thonnys, https://pypi.org/project/thonny-lahendus/ (10.05.2024)

[21] Andmekaitse ja infoturbe sõnastik, https://akit.cyber.ee/ (09.04.2024)

[22] Tartu Ülikool, Arvutiteaduse instituudi 2023/24 õppeaasta kursuse "Programeerimine" kolmanda nädala praktikumiülesanded, https://courses.cs.ut.ee/2023/programmeerimine/fall/Main/Praks3 (23.04.2024)

[23] Info- ja sidetehnoloogia terminite seletav sõnaraamat, http://www.vallaste.ee/ (03.01.2024)

[24] Tartu Ülikool, Arvutiteaduse instituudi 2023/24 õppeaasta kursuse "Programeerimine" esimese nädala praktikumiülesanded, https://courses.cs.ut.ee/2023/programmeerimine/fall/Main/Praks1 (23.04.2024)

[25] Tartu Ülikool, Arvutiteaduse instituudi 2023/24 õppeaasta kursuse "Programeerimine" viienda nädala praktikumiülesanded ja näidiskontrolltööd, https://courses.cs.ut.ee/2023/programmeerimine/fall/Main/Praks5 (23.04.2024)

[26] Tartu Ülikool, Arvutiteaduse instituudi 2023/24 õppeaasta kursuse "Programmeerimine" 12. nädala näidiskontrolltöö ülesanded, https://courses.cs.ut.ee/2023/programmeerimine/fall/Main/KT2naide (29.04.2024)

[27] Tartu Ülikooli õppeinfosüsteem, kursus “Programmeerimise alused MTAT.03.236”, https://ois2.ut.ee/#/courses/MTAT.03.236/version/c0cd5116-5bc0-17a1-719a-166149437aba/details (29.04.2024)

[28] TSL-keele arhitektuur programmeerimise keeles Kotlin, https://github.com/kspar/easy/tree/master/tsl-common/src/commonMain/kotlin/tsl/common/model (29.04.2024)

Lisad

Lisa 1. Praktilise töö materjalid

Bakalaureusetöö raames loodud kursus “UT PROGRAMMEERIMINE 2024” ja uuendatud automaatkontrollid asuvad aadressil https://lahendus.ut.ee/courses/137/exercises. Materjalide nägemiseks tuleb küsida ligipääsu TÜ aine “Programmeerimine LTAT.03.001” vastutavalt õppejõult.

Lisa 2. TSLi Dokumentatsioon

Lisa 2.1 TSLi testide ühised väljad

Väli	Selgitus	Kohustuslik
type	Testi tüüp (nt programmi käivitamise test). Lisas 2.2 on loetletud kõik testide tüübid.	Jah
name	Kasutajale kuvatav testi nimetus sõnena. Vaikeväärtuseks testi tüüp.	Ei
pointsWeight	Testi punktide osakaal, mida arvutatakse valemist: 100 / (kõigi testide punktide osakaalude summa) * punktide osakaal. Vaikeväärtus 1.0.	Ei
visibleToUser	Kasutajale testi nähtavust määrav väli. Vaikeväärtus *True*.	Ei
id	Testi identifikaator.	Jah

Lisa 2.2 TSLi testide tüübid

Testi nimetus	Väljad	Selgitus
program_execution_test	*standardInputData, inputFiles, genericChecks, outputFileChecks, exceptionCheck*	Programmi käivitamise test. Kontrollib, kas nt programm väljastab õige väärtuse.
program_contains_try_except_test	*programContainsTryExcept*	Programm sisaldab veatöötluskonstruktsiooni võtmesõnadega *try* ja *except*.
program_calls_print_test	*programCallsPrint*	Programm sisaldab väljastamise käsku *print*.

program_contains_loop_test	*programContainsLoop*	Programm sisaldab tsüklit.
program_imports_module_test	*genericCheck*	Programm impordib moodulid.
program_contains_keyword_test	*genericCheck*	Programm sisaldab võtmesõna.
program_calls_function_test	*genericCheck*	Programm kutsub välja funktsiooni.
program_defines_function_test	*genericCheck*	Programm defineerib funktsiooni.
program_defines_class_test	*genericCheck*	Programm defineerib klassi.
program_defines_subclass_test	*className, superClass, beforeMessage, passedMessage, failedMessage*	Programm defineerib alamklassi.
program_calls_class_test	*genericCheck*	Programm kutsub välja klassi.
program_calls_class_function_test	*genericCheck*	Programm kutsub välja klassi funktsiooni.
function_execution_test	*functionName, functionType, arguments, standardInputData, inputFiles, genericChecks, returnValueCheck, outputFileChecks, outOfInputsErrorMsg, functionNotDefinedErrorMsg,*	Funktsiooni käivitamise test. Kontrollib, kas nt funktsioon tagastab õige arvu.

	tooManyArgumentsProvidedErrorMsg	
function_contains_loop_test	*functionName, containsLoop*	Funktsioon sisaldab tsüklit.
function_contains_keyword_test	*functionName, genericCheck*	Funktsioon sisaldab võtmesõna.
function_contains_return_test	*functionName, containsReturn*	Funktsioon sisaldab tagastamise käsku *return*.
function_calls_function_test	*functionName, genericCheck*	Funktsioon kutsub välja funktsiooni.
function_calls_print_test	*functionName, callsCheck*	Funktsioon kasutab väljastamise käsku *print*.
function_is_recursive_test	*functionName, isRecursive*	Funktsioon on rekursiivne.
function_defines_function_test	*functionName, genericCheck*	Funktsioon defineerib funktsiooni.
function_imports_module_test	*functionName, genericCheck*	Funktsioon impordib mooduli.
function_contains_try_except_test	*functionName, containsTryExcept*	Funktsioon sisaldab veatöötluskonstruktsiooni võtmesõnadega *try* ja *except*.
function_is_pure_test	*functionName, containsLocalVars*	Funktsioon kasutab ainult lokaalseid muutujaid.
class_imports_module_test	*className, genericCheck*	Klass impordib mooduli.

class_defines_function_test	*className, genericCheck*	Klass defineerib funktsiooni.
class_calls_class_test	*className, genericCheck*	Klass kutsub välja klassi.
class_function_calls_function_test	*className, classFunctionName, genericCheck*	Klassi funktsioon kutsub välja funktsiooni.
class_instance_test	*className, createObject, classInstanceChecks*	Klassi isendi loomise test.

Lisa 2.3 TSLi testide väljad

Väli	**Selgitus**	**Alamväljad (vt lisa 2.4)**	**Kohustuslik**
standardInputData	Sisendite järjend sõnedest.		Ei
inputFiles	Järjend sisendfailidega.	*fileName, fileContent*	Ei
genericCheck	Järjend alamkontrollidega.	*id, checkType, nothingElse, expectedValue, elementsOrdered, dataCategory, ignoreCase, beforeMessage, passedMessage, failedMessage*	Ei
outputFileChecks	Järjend väljundfaili kontrollidega.	*fileName, checkType, nothingElse, expectedValue, elementsOrdered,*	Ei

		dataCategory, ignoreCase, beforeMessage, passedMessage, failedMessage	
exceptionCheck	Järjend erindi viskamise kontrolliga.	*mustNotThrowException, beforeMessage, passedMessage, failedMessage*	Ei
programContainsTryExcept	Erinditöötluskonstruktsiooni võtmesõnadega *try* ja *catch* sisaldavuse kontroll.	*mustNotContain*, *beforeMessage, passedMessage, failedMessage*	Jah
programCallsPrint	Väljastamise käsu *print* sisaldavuse kontroll.	*mustNotCall*, *beforeMessage, passedMessage, failedMessage*	Jah
programContainsLoop	Tsükli sisaldavuse kontroll.	*mustNotContain*, *beforeMessage, passedMessage, failedMessage*	Jah
className	Klassi nimi.		Jah
superClass	Ülemklassi nimi.		Jah
functionType	Funktsiooni tüüp. Võimalikud väärtused *FUNCTION* ja *METHOD*. Viimast kasutatakse klassi funktsioonide testimisel.		Jah
functionName	Funktsiooni nimi.		Jah

arguments	Järjend funktsiooni argumentidega.		Ei
returnValueCheck	Järjend funktsiooni oodatud tagastusväärtustega.		Ei
outOfInputsErrorMsg	Veateade, mis kuvatakse, kui programm küsib rohkem sisendeid, kui automaatkontrollil anda on. Vaikeväärtus: “Programm küsis rohkem sisendeid kui testil oli anda”.		Ei
functionNotDefinedErrorMsg	Veateade, mis kuvatakse, kui programmis ei leita testitavat funktsiooni. Vaikeväärtus: “Funktsioon ei ole defineeritud”.		Ei
tooManyArgumentsProvidedErrorMsg	Veateade, mis kuvatakse, kui funktsioon nõuab vale arvu argumente. Vaikeväärtus: “Funktsioon võtab sisendiks vale arvu argumente”.		Ei

Lisa 2.4 TSLi testide väljade alamväljad

Väärtus	**Selgitus**	**Kohustuslik**
fileName	Failinimi sõnena.	Jah
fileContent	Faili sisu sõnena.	Jah
checkType	Vastuse kontrollimise tüüpi määrav väli (vt lisa 2.5).	Jah

nothingElse	Määrab, kas vastus võib sisaldada muud sisu peale selle, mis on määratud oodatud väärtuste järjendis. Võimalikud väärtused *null*, *False* ja *True*. Vaikeväärtus *null*.	Ei
expectedValue	Järjend oodatud väärtustega sõnedena.	Jah
elementsOrdered	Määrab, kas vastus peab järjestatud nii nagu elemendid järjendis *expectedValue*. Võimalikud väärtused on *true* ja *false*. Vaikeväärtus *false*.	Ei
dataCategory	Järjendi *expectedValue* elementide tüüpi määrav väli. Lisas 2.6 on loetletud välja võimalikud väärtused.	Jah
ignoreCase	Määrab, kas programmi väljundi suur- ja väiketähed on võrdväärsed. Väärtuse *true* puhul on kõik tähed võrdväärsed ning väärtuse *false* puhul eristatavad.	Ei
beforeMessage	Enne alamkontrolli teostamist kuvatav teade.	Jah
passedMessage	Alamkontrolli eduka läbimise teade.	Jah
failedMessage	Alamkontrolli nurjunud läbimise veateade.	Jah
mustNotThrowException	Määrab, kas programm peab viskama erindi. Väärtuse *true* puhul oodatakse, et programm ei viska erindit, ning väärtuse *false* korral peab programm andma erindi.	Jah
mustNotContain	Määrab, kas programm või funktsioon peab sisaldama testitüübile vastavaid elemente. Väärtuse *true* puhul oodatakse, et programm ei sisalda testitüübile vastavaid elemente, ning väärtuse *false* korral peavad programmis leiduma testitüübile vastavad elemendid.	Jah
mustNotCall	Määrab, kas programmis peab esinema käsk *print*. Väärtuse *true* puhul oodatakse, et programm ei sisalda *print* käsku, ning väärtuse *false* korral peab programm sisaldama väljastamiskäsku.	Jah

returnValue	Funktsiooni oodatav tagastusväärtus.	Jah

Lisa 2.5 TSLi väljade *checkType* ja *checkTypeLong* väärtused

Väärtus	**Selgitus**
ALL_OF_THESE	Programmi väljund sisaldab kõiki väärtusi, mis kuuluvad järjendisse *expectedValue*.
ANY_OF_THESE	Programmi väljundis leidub vähemalt üks väärtus järjendist *expectedValue*.
MISSING_AT_LEAST_ONE_OF_THESE	Programmi väljundis puudub vähemalt üks väärtus järjendist *expectedValue*.
NONE_OF_THESE	Programmi väljundis puuduvad kõik väärtused järjendist *expectedValue*.
ANY	Ainult välja *checkTypeLong* väärtusena. Programmi väljundis leidub ükskõik milline väärtus.
NONE	Ainult välja *checkTypeLong* väärtusena. Programmi väljundis puudub väärtus.

Lisa 2.6 TSLi välja *dataCategory* väärtused

Väärtus	**Selgitus**
CONTAINS_LINES	Väljund sisaldab ridu, mis leiduvad järjendis *expectedValue*.
CONTAINS_NUMBERS	Väljund sisaldab arve, mis leiduvad järjendis *expectedValue*.
CONTAINS_STRINGS	Väljund sisaldab sõnesid, mis leiduvad järjendis *expectedValue*.

EQUALS	Väljund võrdub järjendi *expectedValue* elemendiga.

Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Edvard Notberg,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose **Tudengiprogrammide automaatkontrollid programmeerimise kursustel**, mille juhendaja on Reimo Palm, 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.

Edvard Notberg
15.05.2024