Izrada interaktivnih zadataka iz programiranja

Srkulj, Krešimir

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

DIPLOMSKI RAD

Izrada interaktivnih zadataka iz programiranja

Krešimir Srkulj

Zagreb, 2014.

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

SMJER: Profesor fizike i informatike

Krešimir Srkulj

Diplomski rad

Izrada interaktivnih zadataka iz programiranja

Voditelj diplomskog rada: dr.sc. Maro Cvitan

Ocjena diplomskog rada: _____

Povjerenstvo: 1. _____
2. _____
3. _____
4. _____

Datum polaganja: _____

Zagreb, 2014.

SADRŽAJ

1. Uvod

Razumijevanje i vizualizacija apstraktnih procesa studentima predstavlja problem kako pri učenju programiranja, tako i drugih područja sa sličnim značajkama. Zato je motivacija za ovaj rad bilo dizajnirati interaktivne zadatke iz programskog jezika C kako bi olakšao učenicima i studentima pri vježbanju. Zadatke koji pokrivaju gradivo s operatorima, petljama i jednostavnim računima sume sredine i sl. Student mora prepoznati što će program ispisati. Zadaci su napravljeni tako da svaki put kada se program pokrene generatorom slučajnih brojeva i operatora dobiva se novi jedinstveni zadatak. Da bi se olakšala prilagodba zadataka za sustave za e-učenej kao npr. Merlin/Moodle, u programima je odvojen dio koji generira zadatak od dijela koji očitava točno rješenje i prikazuje zadatak na ekranu. Funkcija koja generira programski zadatak kroz sve primjere ima isti naziv `generator`. Tako se provodi projektna i interaktivna nastava uz korištenje interaktivnih softvera. Takvi zadaci mogu poslužiti u nastavi kao pomoćno sredstvo studentima pri učenju osnovnih elemenata programiranja: izraza, operatora, pokazivača petlji i sl. Također, mogu poslužiti i profesorima za provjeru znanja.

2. Učenje programiranja i tutorski sustavi

Istraživači i stručnjaci, uključeni u proces poučavanja, programiranje nalaze kompleksnom misaonom aktivnošću kojom se definira apstraktan proces. Razumijevanje i vizualizacija apstraktnih procesa učenicima predstavlja problem kako pri učenju programiranja, tako i drugih područja sa sličnim značajkama. Zato, učenje i poučavanje programiranja predstavlja izazov kako i za učenike tako i za nastavnike podjednako. Istraživanja pokazuju da je to opći problem.

Studenti često imaju poteškoća u razvoju algoritama za rješavanje čak i jednostavnih problema i dok možda postoji razumijevanje i znanje tematskog područja, kako primijeniti to znanje stvara probleme. To je čest problem u području programiranja. Po svojoj prirodi, područje programiranja ima i matematičku pozadinu, međutim, vještina primjene samog zahtijeva gotovo umjetnički razvoj. Zato, tradicionalni pristup nastavi programiranja je u velikoj mjeri apstraktan i matematički po prirodi. Razvoj programske vještine, međutim, zahtijeva od korisnika da uzme pravi životni problem i konstruira algoritam da ga riješi. To zahtijeva od korisnika da primijeti odnos između programiranja i stvarnog svijeta kao bi mogao prevesti jedan u drugoga.

2.1. Što student treba naučiti?

Linn i Dalbey [8] definiraju idealan lanac spoznajnog postignuća učenjem programiranja i predlažu ga standardom za usporedbu metoda nastave programiranja. Tri glavne karike lanca su:

- ***svojstva programskog jezika***
- ***vještina oblikovanja programa***
- ***opća sposobnost rješavanja problema***

Svojstva jezika – u cilju zapisivanja programskog rješenja problema predmetnim jezikom, učenik treba razumjeti sintaksu, semantiku, te izražajne mogućnosti jezika.

Vještina oblikovanja programa - je znanje uporabe grupe tehnika čijom se primjenom svojstva jezika kombiniraju u cjelinu kojom se rješava postavljeni problem. Vještina se temelji na znanju predložaka stereotipskih uzoraka koda koji komponiraju više svojstava jezika. Predlošci provode kompleksne funkcije, kao što su sortiranje, pronalaženje najmanjeg višekratnika dva cijela broja, brojanje riječi u zadanom tekstu itd. Programeri planiraju kombiniranje svojstava jezika i predložaka, dekomponiraju problem u dijelove, neovisno rješavaju svaki dio, te parcijalna rješenja povezuju u jedinstvenu cjelinu – program. Testiranjem se utvrđuje korektnost programa.

Opća sposobnost rješavanja problema - dolazi do izražaja pri učenju novih formalnih sustava i postavlja se ciljnim postignućem nastave programiranja. Isti predlošci i proceduralne vještine su zajedničke mnogim ili čak svim formalnim sustavima. Dakle, pristup u kojem će učenik naučiti predloške zapisane jednim formalnim sustavom i pravilima znati prebaciti u novi predmet tekućeg učenja. Tu do izražaja dolazi smisleno učenje i aktivacija postojećeg znanja.

2.2. Što početnicima predstavlja problem?

Lemut i dr. [7] poteškoću učenja programiranja objašnjavaju potrebom primjene niza kompleksnih aktivnosti, koje početnik mora savladati istovremeno. Na primjer, program se testira izvođenjem, uz pažljivo odabrane rubne vrijednosti programskih ulaza, koji će rezultirati izvođenjem svih programskih puteva. Izbor rubnih vrijednosti ulaza zahtjeva poznavanje semantike programskih instrukcija. Nasuprot tome, programer početnik uči instrukcije, pa vrlo teško može samostalno odabrati takve ulaze.

DuBoulay [6] moguće izvore poteškoća nalazi u:

1. Orijentaciji - općoj predodžbi učenika o programiranju i programu.
2. Apstraktnom stroju - razumijevanju računalnog modela kojeg definira programski jezik.
3. Notaciji - sintaksi i semantici jezika.
4. Strukturama - znanju programskih konstrukcija kao kompoziciji instrukcija kojima se rješavaju određeni programski zahtjevi.
5. Pragmatici - vještinama primijenjenim u izgradnji korektnog programa (planiranje, dekompozicija, kodiranje, testiranje, pronalaženje i otklanjanje pogrešaka).

2.3. Tradicionalna nastava programiranja

U tradicionalnoj nastavi programiranja, učitelj postupno objašnjava svojstva jezika kroz izabrane primjere. Razumijevanje instrukcija učenici produbljuju samostalno rješavajući postavljene zadatke, pri čemu učenikov program može sadržavati pogreške kategorizirane kao:

- ***sintaksne pogreške***
- ***semantičke pogreške nezavisne zadatku***
- ***semantičke pogreške zavisne zadatku***

Pridjeljivanje cjelobrojnoj varijabli vrijednosti s pomičnim zarezom ili ponavljanje koje nikada neće završiti su primjeri semantičkih pogrešaka nezavisnih zadatku. Semantičke pogreške zavisne zadatku predstavljaju razliku ostvarenog i namjeravanog ponašanja programa.

Pored toga, učenik može napisati program koji proizvodi korektne izlaze, ali uz primjenu lošeg stila. Cilj učenje programiranja nije postizanje isključivo korektnih izlaza, nego korektnog, stilski ispravnog programa.

Pogreške učeničkih programa u pravilu su individualne i vrlo različite. Učenička skupina istog tečaja može biti izrazito heterogena s obzirom na motivaciju, predznanje, sposobnost i stil učenja. Tradicionalni pristup nastavi programiranja, u kojem jedan učitelj treba ustanoviti i objasniti pogreške svakom učeniku, često pred učitelja stavlja izazovnu i napornu zadaću. Uvođenjem u nastavu programiranja tutorskog sustava, koji je u stanju locirati i objasniti individualne pogreške, proces učenja i poučavanja programiranja se značajno unapređuje i olakšava.

2.4. Kognitivna psihologija učenja

Kognitivna psihologija naglašava značaj smislenog učenja pri usvajanju novog znanja. Aktivacijom postojećeg znanja, nove se informacije povezuju s onima koje učenik posjeduje u svojoj trajnoj memoriji. Konkretni modeli, bliski učeniku, kojima se apstraktni procesi računalnog programa čine vidljivim, pomažu razumijevanju tehničkih informacija. Razumijevanje svojstava jezika se može značajno unaprijediti, opisuju li se akcije programskih instrukcija prirodnim jezikom. Sasvim općenito, pristupi tutorskog poučavanja mogu se svrstati u preliminarno, interaktivno i refleksivno.

Preliminarno poučavanje - profesor unaprijed priprema učenika kako će riješiti konkretan zadatak, najčešće navodeći rješenja sličnih problema. Profesor ne utiče na

postupak rješavanja, a konačno rješenje se ocjenjuje kao korektno ili pogrešno, bez objašnjenja pogrešaka.

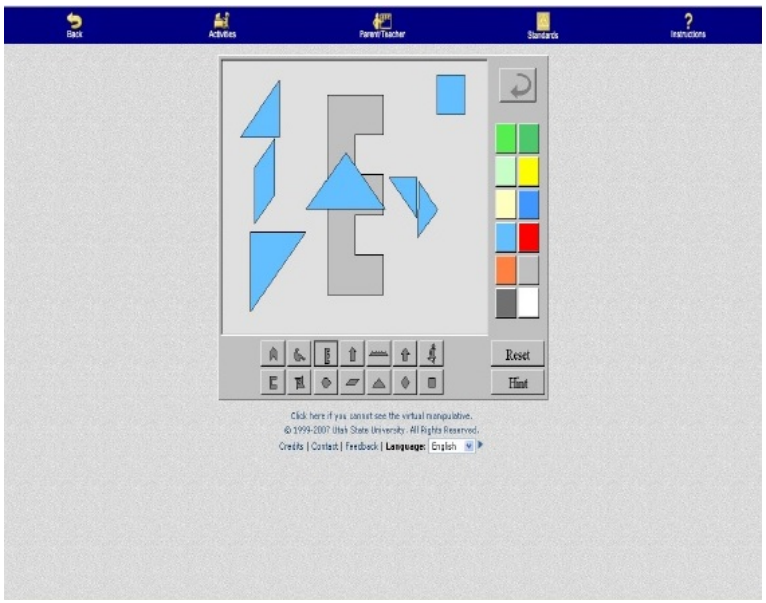
Interaktivno poučavanje – učenik pristupa rješavanju zadatka primjenjujući prezentirano opće znanje područja, bez prethodnih uputa vezanih uz konkretan problem. Profesor odmah intervenira, čim učenik nepovratno odstupi od putanje korektnog rješenja ili bezidejno odustaje od samostalnog traženja rješenja.

Refleksivno poučavanje – učenik samostalno rješava zadatak bez intervencija nastavnika, a potom nastavnik provodi reviziju konačnog rješenja i objašnjava pogreške.

Uspoređujući pristupe tutorskog poučavanja, uočava se potpuni izostanak operativne povratne veze u preliminarnom pristupu. Povratna veza refleksivnog pristupa često može biti zakašnjela, dopuštajući protezanje ranih pogrešaka cijelim programom, zbog čega se mogu kumulirati nove pogreške. Ispravak niza pogreški predstavlja tešku zadaću većini početnika. Poučavanje daje najbolje rezultate primjenom interaktivnog poučavanja, uz povratnu vezu bez odgode, jer ne dopušta da učenik značajno odstupi s putanje korektnog rješenja. Nedostatak ovog pristupa, kada je jedini raspoloživ u sustavu, dolazi do izražaja kod učenika u naprednim fazama učenja.

3. Interaktivna nastava

Interaktivna nastava počinje s filozofijom o poučavanju s tehnologijom i rezultatima. Kombinacija konstruktivizma, interaktivne ploče i Web 2.0 alata je jedan od načina razmišljanja za nove početke poučavanja. U tom okviru i studenti i profesori su u središtu zbivanja. Nastavnici su odgovorni za planiranje, podučavanje i prilagođavanje na tehnologiju. Studenti su odgovorni za izgradnju i pokazivanje znanja, te za suradnju s kolegama kako bi kreirali znanje. U fazi planiranja profesori odlučuju o tome koji će alati poboljšati kognitivnu ekspanziju za studente kao što su implementacija Web 2.0 sučelja koji pomažu studentima u pristupu i obradi informacija. U fazi interaktivne nastave, profesorski model uzima tehnologiju kako bi izgradio znanje i demonstrirao koncepte kroz dinamičku interakciju. U trećoj fazi, profesori olakšavaju izgradnju znanja kroz predavanja u kojima studenti sudjeluju u cijelom nastavnom procesu. To je recipročan proces upotrebe tehnologije kako bi prezentirali i pokazali znanje. Slika 3.1 prikazuje usporedbu tradicionalne nastave s nastavom i učenjem 21. stoljeća.



Slika 3.1: Ilustracija elektroničkog oblika drevne kineske igre puzzle

Promjene se temelje na interaktivnoj komponenti u spoju tehnologije s poučava-

njem i učenjem. Ako prvi stupac tablice predstavlja Web 1.0, statičku verziju interneta, onda tradicionalna nastava također predstavlja više statičku verziju poučavanja. To nam ne daje za pravo reći da se sva tradicionalna nastava smatra statičkom. Web 2.0 predstavlja novi pristup za interakciju sa sadržajem koji više koristi sudjelovanje i interakciju. To predstavlja stupac dva. Opisi u drugoj koloni predstavljaju više razine interakcije studenata i profesora. Student i profesor su zajedno aktivni u procesu učenja kako je i opisano u Web 2.0 pedagogiji. Osim toga, alati koji olakšavaju tu transformaciju su interaktivni po prirodi i moraju se učiti kao dio sistemskog tijela znanja. Ti alati omogućuju profesorima stvoriti interaktivno okruženje putem teorijskih razmatranja i praktične primjene.

$TRADICIONALNIPRISTUP(Web1.0)$	$NOVIPRISTUP(Web2.0)$
Profesor u prvom planu	Student u prvom planu
Samostalan rad	Rad u skupinama
Dostava informacija	Razmjena informacija
Pasivno učenje	Aktivno učenje
Single- media	Multimedia
Reakcijski odgovor	Impulzivan/planiran odgovor
Izolirani, umjetni sadržaj	Originalni, iz stvarnog svijeta sadržaj

Web2.0 predstavlja drugu generaciju servisa i internetskih sredstava koji obuhvaćaju mreže javnih siteova,wikije i folksonomije(metode kolaborativnog tagiranja korištenjem ključnih riječi po slobodnom izboru) kojima je cilj razviti kolaboraciju i zajedničko korištenje među korisnicima[2]. Web2.0 pojam ne sugerira da se radi o novoj verziji weba, nego se odnosi na promjene u odnosu na način na koji se razvijaju aplikacije, način na koji razvojni inženjeri koriste web platformu, te ljudima sa sličnim interesima omogućuju da se vrlo jednostavno povežu u velike online zajednice. Kod web 1.0 prisutna je jednosmjerna komunikacija, gdje web služi jednostavno kao izvor informacija dok web 2.0 komunikaciju pokušava usmjeriti i prema korisnicima i na taj način stvoriti obostranu komunikaciju. Do sada se u osnovnoj primjeni informacijske infrastrukture u obrazovanju koristio isključivo jednosmjeran pristup on-line učenju, odnosno proces učenja svodio se na čitanje i primanje informacija te odgovaranje na dobivena pitanja. Dakle web 1.0 je provodio pasivan način obrazovanja, dok web 2.0 uključuje spajanje sudionika, omogućuje stvaranje novih sadržaja. Na taj način je postignut doprinos svih sudionika u procesu obrazovanja.

Jedan od prigovora na ne interaktivnu nastavu je da se ona temelji na predavanju

(didaktici), stvarajući tako okruženje usmjereno na profesora a ne studente. Sama nastava nije lišena predavanja, dapače predavanje se upotrebljava u kombinaciji s aktivnim demonstracijama. Profesor kemije uz pomoć interaktivne ploče može objasniti lekciju o balansiranju jednadžbi manipulirajući elementima uz pomoć svojih prstiju kako bi demonstrirao koncept vizualno. U integriranom pristupu, govor se koristi za opisivanje procesa. Interaktivna nastava uključuje i profesora koji integrira više oblika medija unutar lekcije kako bi ohrabrio kognitivno sudjelovanje.

Interaktivno može imati više definicija. Na primjer, kada se koriste određene internetske stranice, interaktivnost bi moglo značiti da kliknete na link i pristupite tekstu. U učionici, interaktivnost bi moglo značiti ispunjavanje nekog radnog lista (engl. worksheet). Kod interaktivne ploče, interaktivnost znači da profesor i učenik obavljaju fizičke aktivnosti kao što je prevođenje geometrijskih oblika riječi kako bi se stvorile rečenice. Obje aktivnosti stvaraju interakciju s kognitivnim procesima olakšavajući izgradnju znanja. Interaktivnost također znači da profesor i student aktivno sudjeluju u raspravi. Interaktivna priroda interaktivne ploče daje prednost profesoru kada se upotrebljava na ovaj način. Kada se uzmu u obzir nekadašnji alati za profesore, tada, nije bilo alata napravljenog kako bi se koristio samo za interakciju kroz mentalne i fizičke procese. Većina pribora jednostavno su zahtijevali od studenata da samo gledaju, ali ne i sudjeluju. Korištenje interaktivne ploče je nova pojava u nastavi i još uvijek se mora puno toga naučiti i shvatiti kako bi se ona sve mogla koristiti kao alat za učenje. Interaktivna ploča je alat koji pruža novi način obavljanja zadataka poučavanja.

3.1. Interaktivno učenje

Tradicionalno, studenti sjede i upijaju znanje s predavanja i bilježaka na ploči danih od profesora. Kod interaktivne nastave studenti aktivno sudjeluju u procesu učenja. U okruženju za učenje koje uključuje i interaktivnu ploču studenti su fokusirani na poticaj ili profesora koji je uz ploču ili studenta, koji su verbalno ili fizički u interakciji sa samom pločom. U definicijama interaktivne nastave kao primjer je dan o učenicima koji "povlačenjem" riječi moraju sastaviti rečenicu koja opisuje digitalnu sliku. To je oblik interaktivnog učenja, jer student radi interakciju sa sadržajem kroz kombinaciju apstraktnog i konkretnog. Ova vrsta učenja, usmjerena na studenta, slijedi načela konstruktivnog učenja gradeći pritom interaktivno okruženje za učenje. Studenti se potiču da kontroliraju svoje učenje i da konstruiraju značenje samog.

3.2. Interaktivni alati

Interaktivne ploče slika 3.2, također poznate kao elektronske školske ploče su prikaz monitora koji se proizvode u više veličina od strane različitih tvrtki ka što su Smart Technology, Promethean, Sony i druge. Interaktivne ploče imaju dvije različite funkcije: zaslon i interaktivnost. Kao alat za prikazivanje, nastavnici mogu prikazati specifične sadržaje, datoteke vezane uz temu, programe ili Internet resurse. Kao interaktivni alat, interaktivne ploče omogućuju korisniku da piše i manipulira objektima, koji uključuju i slike i tekst. Interaktivne ploče se spajaju uz pomoć USB porta na računalo (stolno ili prijenosno). Projektor koji dolazi s pločom se također spaja u računalo. Zaslon računala se zatim projicira na interaktivnu ploču i korisnik ima pristup svim datotekama, programima i Internetu pri ruci ili uređaju, ovisno o vrsti ploče. Kombinacija interaktivne ploče, projektora i računala predstavljaju dinamičan sustav koji omogućava fleksibilnost u učionici. Postoje određene prednosti korištenja interaktivne ploče u učionici. Pružaju veću fleksibilnost u tome kako će se lekcije izložiti, poboljšati studentsku interakciju sa sadržajem, omogućiti veću vizualizaciju koncepata za studente, te povećati motivaciju među studentima.

Slika 3.2: Interaktivna ploča s projektorom i računalom

Web 2.0 alati. Postoji velika rasprava oko samog značenja Web 2.0. Iako ne postoji konsenzus o jednoj definiciji, postoje neke osnovne sličnosti u razgovorima o Web 2.0. Ukupni koncept je o zajednici i mogućnosti da se sudjeluje kroz Web baziranu interakciju. Rasprave o Web 2.0 također upućuju na to da ljudi koji koriste internet traže, objavljuju, stvaraju , demonstriraju itd. Mnogo glagola se može koristiti kako bi

se opisala Web 2.0 funkcionalnost. Glagoli definiraju akcije koje bi profesori i studenti mogli učiniti u učionici dok se odvija proces poučavanja i učenja. U učionici se može definirati interaktivna upotreba interneta kako bi se poticao i podržavao proces nastave i učenja, te na taj pomažući u stvaranju interaktivnih okruženja za učenje. Neki od primjera Web 2.0 alata koji se koriste u Hrvatskoj su "Claroline" i "eduHr".

3.3. Interaktivni program

Interaktivni program [3], poznatiji kao kodiranje uživo (eng. live coding), odnosi se na bilo koji računalni programski jezik koji omogućuje kreatoru napraviti promjene u programu, iako je on već pokrenut. U tradicionalnom programiranju, programer ispiše program i zatim ga spremi. Nakon toga pokrene program na računalu. Ako dođe do pogreške, ispisuje novi ili popravlja stari kod i pokreče program ispočetka.

Drugi način za upotrebu interaktivnog programa je omogućiti ulaz od korisnika u tzv. interaktivnu aplikaciju (eng. an interactive application) [5]. To može biti nešto jednostavno kao na primjer pitati korisnika kao se zove i zatim ispisati njegovo ime na ekranu. Program ima interaktivni element mijenjajući vrijednost korisnikova imena bazirano na tome što je korisnik upisao. Kada je program napravljen, nije znao ime korisnika, a vrijednost je bila prazna. Nakon što je saznao ime, stavlja tu vrijednost u program, dok on radi, a zatim ga prikazuje na ekranu.

Ova vrsta interaktivnog programa je u suprotnosti s drugim programskim procesom poznatijim kao skupna obrada (eng. batch processing). Kod skupne obrade, program može raditi bez potrebe ikakvog unosa od strane korisnika. Tu program ima prednost da radi sam bez pomoći korisnika, ali zbog toga ima i jednu veliku manu. Sve informacije koje program treba da bi radio moraju biti unošene u njega od početka. Ako program želi ispisati korisničko ime, treba ga znati otprije, jer nije u mogućnosti tražiti korisnički unos.

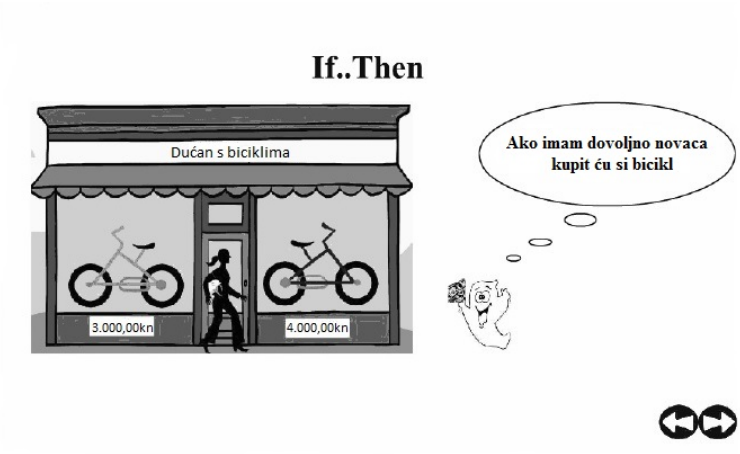
Potrebni su razvojni procesi za stvaranje programa. Ti procesi započinju s time što bi program trebao raditi, zatim, pisanjem koda za njega i na kraju testiranjem samog. Programer se zatim vraća nazad u program, radi promjene na njemu i ponovo ga testira. Ovaj proces se ponavlja sve dok program nije napravljen. Kada se koristi interaktivni program, crta koja razdvaja razvojne faze postaje nejasna. Pisanje programa i pokretanje programa postaju jedno te isto. Umjesto pisanja programa i zatim njegovog pokretanja, programer može napisati program, pokrenuti ga i nastaviti pisati ili raditi promjene u njemu dok je on pokrenut.

4. Vizualizacija u nastavnim sadržajima

Činjenica je da postoji više stilova učenja. Sadržaji u lekcijama su prezentirani tako da zahtijevaju od studenata različite stilove učenja. Kako bi se olakšale sve te razlike u učenju koriste se animacije gdje god je to moguće. Te animacije uključuju tekstualne, grafičke i auditivne elemente kako bi pokušale uključiti studente u djelotvornija okruženja za učenje koja se odnose na sva njihova osjetila. Korištenjem odgovarajućih grafičkih prikaza i animacija studenti će bolje shvatiti potrebne koncepte jer ih mogu bolje vizualizirati i shvatiti kako se oni pojavljuju.

U sljedeća dva poglavlja ilustrirat ću korištenje vizualizacije u nastavi programiranja. Kao primjere ću koristiti dvije kontrolne strukture: izbor (If else petlja) i ponavljanje (While petlja).

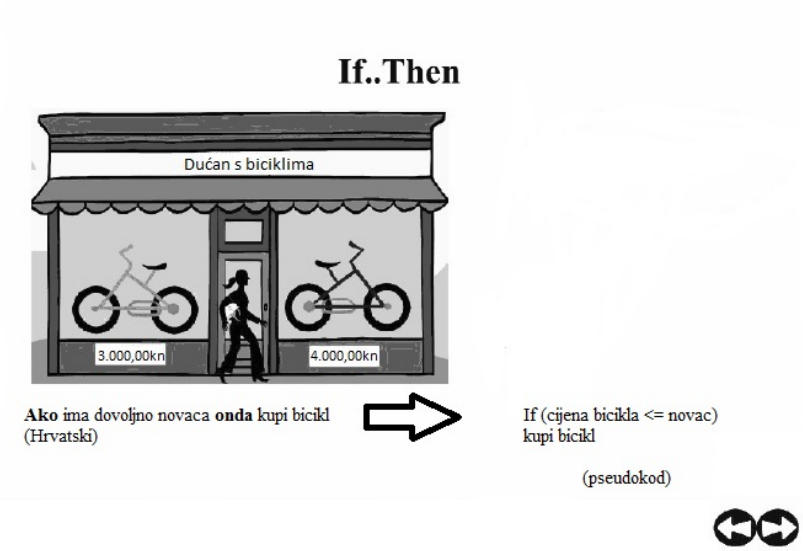
4.1. If petlja



Slika 4.1: Uvod u problem

Problem koji je stavljen pred studente je onaj s kojim se studenti mogu lako identificirati. Na početku su suočeni sa stvarnim problemom gdje moraju odlučiti što će učiniti. Student ima određenu sumu novaca koju može potrošiti, a želi kupiti bicikl 4.1.

Slika 4.2 prikazuje problem pojednostavljen za odluku o količini novaca koju student ima za potrošiti. Srž problema je objašnjen u hrvatskom pripovjedačkom obliku popraćen s ekvivalentnim pseudokodom. To omogućuje studentu da brzo prebaci pripovjedački tekst u više programski oblik.

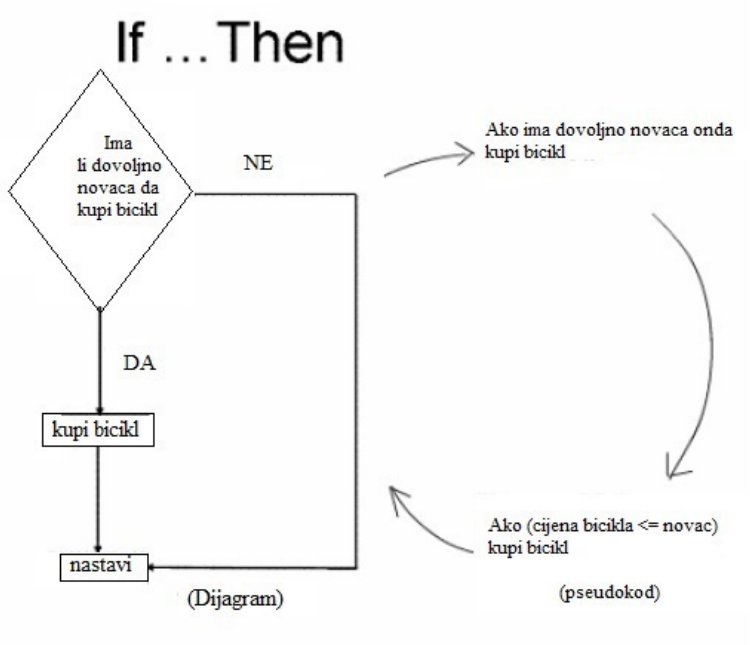


Slika 4.2: Pojednostavljenje problema

Slika 4.3 uvodi dijagram uz pomoć animacije kako bi se napravio dijagram u prirodnom slijedu. On prikazuje različite, ali ekvivalentne poglede na If..Then strukturu. Sama struktura je uvedena logično i sekvencijalno. If..Then..Else struktura je uvedena tek nakon što se student uvježbao i prilagodio na If..Then strukturu.

4.2. While petlja

Problem koji ću koristiti za while petlju je sljedeći: Sunce sja i student se igra s loptom slika 4.4. U prezentaciju se može ubaciti i zvuk kako bi se bolje približila situacija studentima. Mogu se staviti zvukovi poput cvrkuta ptice i udaranje lopte o pod. Uvjet i aktivnost se zatim prikazuju na interaktivnoj ploči slika 4.5 i tako omogućuju studentima da se povežu s uvjetom while petlje sa samom aktivnošću.

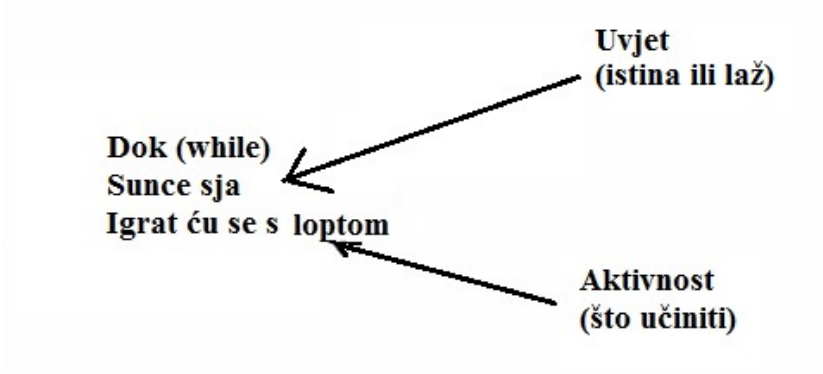


Slika 4.3: Dijagram problema

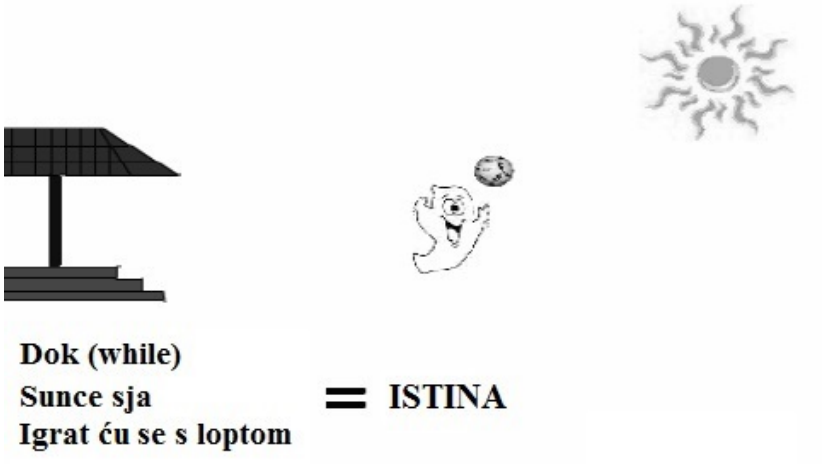
Tada se događaju situacije koje mijenjaju uvjete while petlje, tako objašnjavajući studentima koje su granice while uvjeta.(slike 4.6 i 4.7).

Animacija treba pružati stalno kretanje tako da pogled studenata nije usmjeren na set razdvojenih okvira već kao film s kojim se mogu poistovjetiti. Sadržaj pokazuje while petlju i putove kroz program koji prikazuje vrijednosti varijable, uključujući i izlaz iz while petlje. Nakon toga slijedi dijagram toka i pseudokod.

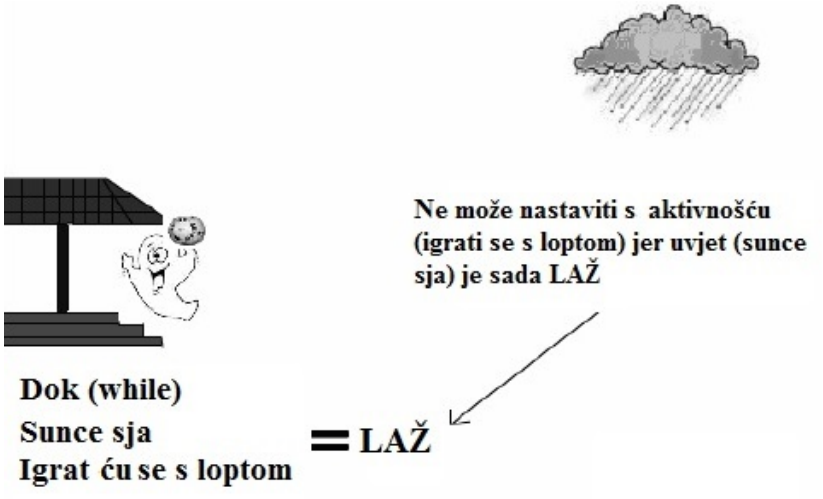
Slika 4.4: Postavljanje problema



Slika 4.5: Uočavanje uvjeta i aktivnosti



Slika 4.6: Uvjet je ispunjen - istina



Slika 4.7: Uvjet nije ispunjen - laž

5. O pythonu

Python je interpreterski, interaktivni, objektu orijentirani programski jezik, kojeg je 1990. godine stvorio Guido van Rossum. Ime je dobio po televizijskoj seriji Monty Python Flying Circus [1]. Od 2000. su ga prihvatile ustanove kao MIT, NASA, IBM, Google, Yahoo i druge. Python ne donosi neke nove revolucionarne značajke u programiranju, već na optimalan način ujedinjuje sve najbolje ideje i načela rada drugih programskih jezika. On je jednostavan i snažan istodobno. Više nego drugi jezici on omogućuje programeru više razmišljanja o problemu nego o jeziku. U neku ruku možemo ga smatrati hibridom: nalazi se između tradicionalnih skriptnih jezika (kao što su Ruby i Perl) i sistemskih jezika (kao što su C, C++ i Java). To znači da nudi jednostavnost i lako korištenje skriptnih jezika (poput Matlab-a), uz napredne programske alate koji se tipično nalaze u sistemskim razvojnim jezicima. Python je besplatan (za akademske ustanove i neprofitnu upotrebu), open-source softvere, s izuzetno dobrom potporom, literaturom i dokumentacijom. Python kod sprema se u tekst datoteke koje završavaju na .py. Brzina izvođenja Python koda istog je reda veličine kao u Javi ili Perlu. Python je napisan u ANSI C i raspoloživ za cijeli niz strojeva i operacijskih sustava uključujući Windows, Unix/Linux i Macintosh. Najviše se koristi na Unix/Linux sustavima. Osim standardnih tipova podataka (brojevi, nizovi znakova i sl.) Python ima ugrađene tipove podataka visoke razine kao što su liste, n-terci i rječnici. Može se izvoditi u različitim okruženjima. Za razvitak programa najlakši je interaktivni način rada u kojem se programski kod piše naredbu za naredbom. Ne postoji razlika u razvojnom i izvedbenom (engl. runtime) okolišu: u prvom se izvodi naredba za naredbom, a u drugom odjednom čitava skripta. Python nudi sve značajke očekivane u modernom programskom jeziku: objektu orijentirano programiranje s višestrukim nasljeđivanjem, dohvaćanje izuzetaka ili iznimki (engl.exception), redefiniranje standardnih operatora, pretpostavljene argumente, prostore imena (engl. namespaces), module i pakete. Pisan je u modularnoj C arhitekturi. Zato se može lako proširivati novi značajkama ili API-ima. (engl. application programming interface).

1996. godine Van Rossum je napisao o nastanku Python-a:

"Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus)[9].

6. Programi

Cilj je bio napraviti zadatke u programskom jeziku Python, koji će slučajnim odabirom generirati brojeve i operatore. Program u sebi sadrži linijske kodove programa C koji se ispisuju na ekranu. Tako student ili učenik dobije zadatak pred sebe iz programskog jezika C. Zadaci koje sam napravio pokrivaju nastavne jedinice Deklaracija varijabli, Aritmetički operatori, Višestruko uvjetno grananje, Programska petlja *for* i Programska petlja *while*. Najveći naglasak je bio na operatorima[4] i zato se oni nalaze u skoro svim zadacima. Svi programi sadrže tri glavne funkcije, od kojih se dvije mijenjaju, a jedna je ista cijelo vrijeme.

Funkcija koja ostaje ista u svim zadacima je `savecode` koja ugrađenu funkciju `open()` poziva i stvara Python objekt tipa `file` koji služi kao veza prema fizičkoj datoteci. Koristi dva argumenta tempc i code. Dakle ta funkcija otvara datoteku 'tempc' za pisanje, zapisuje u nju kod i zatvara datoteku.

Najvažnija funkcija koja se koristi i u svim zadacima je drugačija, je funkcija `generator`. Ona poziva datoteku za pisanje i sadrži naredbu kojom se poziva C kod. U njoj se generiraju brojevi koje koristi C kod, kao i operatori koji se koriste u samom zadatku koji je postavljen pred studente. Operatori koji se generiraju u ovoj funkciji ovise o tome što traženi C kod traži. Zato, neki programi u sebi sadrže operatore kao što su +, /, * , %, a drugi programi operatore kao što su +, * , %, && i ||. Sadrži također i ugrađenu funkciju `dict` koja vraća novi rječnik čiji sadržaj je određen opcionalnim argumentom. Argument može biti niz dvočlanih nizova ili niz ključevima pridruženih vrijednosti. U mom programskom kodu tu dolazi do zamjene brojeva i operacija u C kodu. Poziva se naredbom `code = code00.format(**zamjene)`. Funkcija zatim sprema file i kod.

Nešto malo promjenjiva funkcija u zadacima je funckija `check`. Ona je namijenjena da ispiše kod na ekran studentu ili učeniku, te da traži od njega da upiše rješenje što će program ispisati. Dobiveni unos ovisno o kojem se programu radi, stavlja u listu ili matricu i ispisuje odgovor na ekran. Zatim, ispisuje točan odgovor tako da izvršava ne-Python program (u ovom slučaju C program) naredbom `rezs`

`= os.popen(cmd).readlines()`. Metodom `readlines()` datoteka se čita po recima i vraća se lista stringova pojedinih redaka. Naredba `rezs = map(lambda x: filt( re.split("\s*",x) ), rezs)` čisti \n iz C programa i tako dobivam samo ispis rješenja. Da nema te naredbe rješenja bi bila ispisana kao npr. `[2\n], [3\n]`. Nakon što je funkcija ispisala unos rješenja od studenta i izvršenje samog programa, ona uspoređuje ta dva dobivena podatka. Ako su oba podatka jednaka program ispisuje da je odgovor točan, a ako nisu da je odgovor netočan.

6.1. Prvi zadatak

Prvi zadatak je napravljen na temu aritmetičkih operatora. Program generatorom slučajnih brojeva zadaje neke početne vrijednosti za konstante. U programu se koriste smo cjelobrojne konstante. Osim cjelobrojnih konstanti generiraju se i aritmetički operatori slučajnim odabirom i to tako da prvi operator može biti samo -, * i +, a drugi / i %. Od studenata se očekuje da znaju koristiti aritmetičke operatore i ne bi trebali imati puno problema s rješavanjem samog zadatka. Student mora napisati koji će se broj ispisati na zaslonu nakon izvođenja programa.

```
Sto ce ispisati sljedeci program:


#include <stdio.h>
int main()
{
    int x = 8, y = 10, z = 2, a;
    a = 8 - 10 / 2;
    printf("%d\n", a);
    return 0;
}

Vas odgovor je:4
[[4]]
Tocan odgovor je:
[[3]]
Vas odgovor je netocan
```

```
Sto ce ispisati sljedeci program:


#include <stdio.h>
int main()
{
    int x = 5, y = 5, z = 5, a;
    a = 5 - 5 % 5;
    printf("%d\n", a);
    return 0;
}

Vas odgovor je:5
[[5]]
Tocan odgovor je:
[[5]]
Vas odgovor je tocan
```

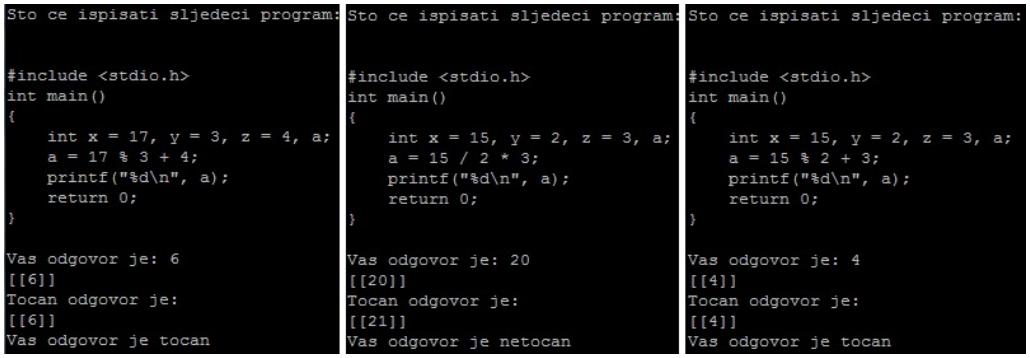
```
Sto ce ispisati sljedeci program:


#include <stdio.h>
int main()
{
    int x = 8, y = 10, z = 1, a;
    a = 8 + 10 % 1;
    printf("%d\n", a);
    return 0;
}

Vas odgovor je:8
[[8]]
Tocan odgovor je:
[[8]]
Vas odgovor je tocan
```

Slika 6.1: Primjeri prvog zadatka s točnim i netočnim rješenjima

6.2. Drugi zadatak

Zadatak je dosta sličan prvom zadatku. Tema koju obrađuje je isto aritmetički operatori. Slučajno se generiraju cijeli brojevi i operatori. Kod drugog zadatka zamijenjena su mjesta operatora i ne koristi sve operatore kao u prvom zadatku već samo neke. Radi toga drugi zadatak je nešto teži od prvog jer traži od studenta da prepozna koji operatori imaju viši prioritet, a koji niži. Zato je drugi zadatak i nešto teži od prvog.

Na prvom mjestu se nalaze operatori / i %, a na drugom * i +. Zadatak je napravljen tako da se ne dobije negativno rješenje.



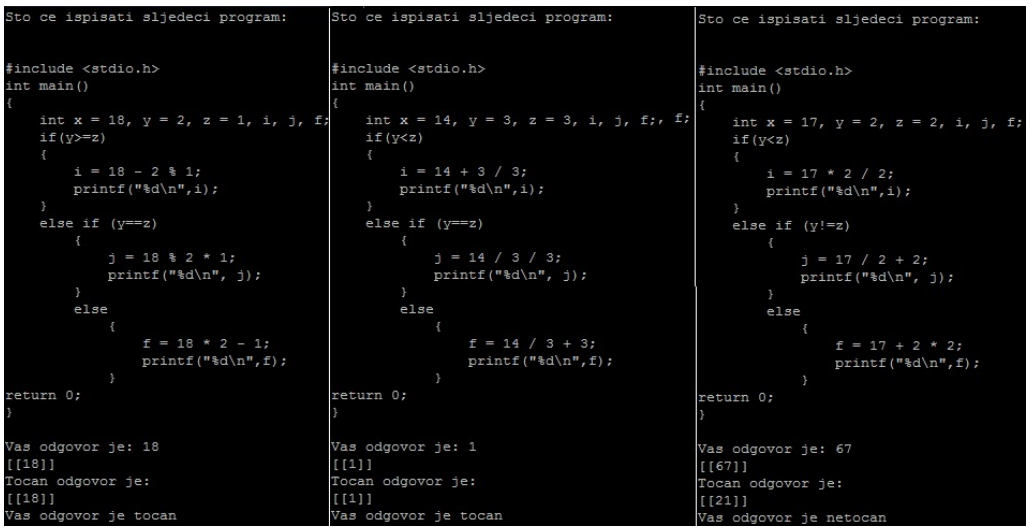
Slika 6.2: Primjeri drugog zadatka s točnim i netočnim rješenjima

6.3. Treći zadatak

Kod trećeg zadatka tema je višestruko uvjetno grananje. Program zahtijeva da se, ovisno o uvjetu, izvode više neovisnih bloka naredbi. Oblik uvjetnog grananja koji se koristi u ovom zadatku je `if -else` naredba. Uz tu naredbu koriste se i relacijski operatori. Dakle, zadatak traži od studenta poznavanje i `if - else` naredbe i poznavanje relacijski operatora. Operatori koji se koriste u ovom zadatku u `if` naredbi su >, <, >= i <=, a u `else` naredbi == i !=. Generatorom slučajnih brojeva izgenerirana su tri cijela broja od kojih se dva uspoređuju i ako je uvjet istina izvršava se naredba. Naredba je ista kao i u prvom i drugom zadatku, odnosno koriste se aritmetički operatori kako bi se izračunala neka varijabla. Student ispisuje jedan broj.

6.4. Četvrti zadatak

U ovom zadatku se obrađuje nastavna jedinica Programska petlja *for*. Jednostavna *for* petlja traži od studenta da ispiše koje vrijednosti ispisuje programski kod. Slučajnim generatorom brojeva izgenerirani su `inicijalizacija`, `uvjet` i `promjena_vrijednosti`. `Inicijalizacija` je u zadanom intervalu od 0 do 5, a `uvjet` u intervalu od 9 do 15. Za uvjet sam postavio dva moguća operatora usporedbe < i <=. Kod `promjene_vrijednosti` sam stavio tri slučaja ++, +=2 i +=3. Studenti moraju prepoznati koji od `promjene_vrijednosti` je u zadatku i što on znači. Od studenta se traži da ispišu koji će niz brojeva će se ispisati na zaslonu.



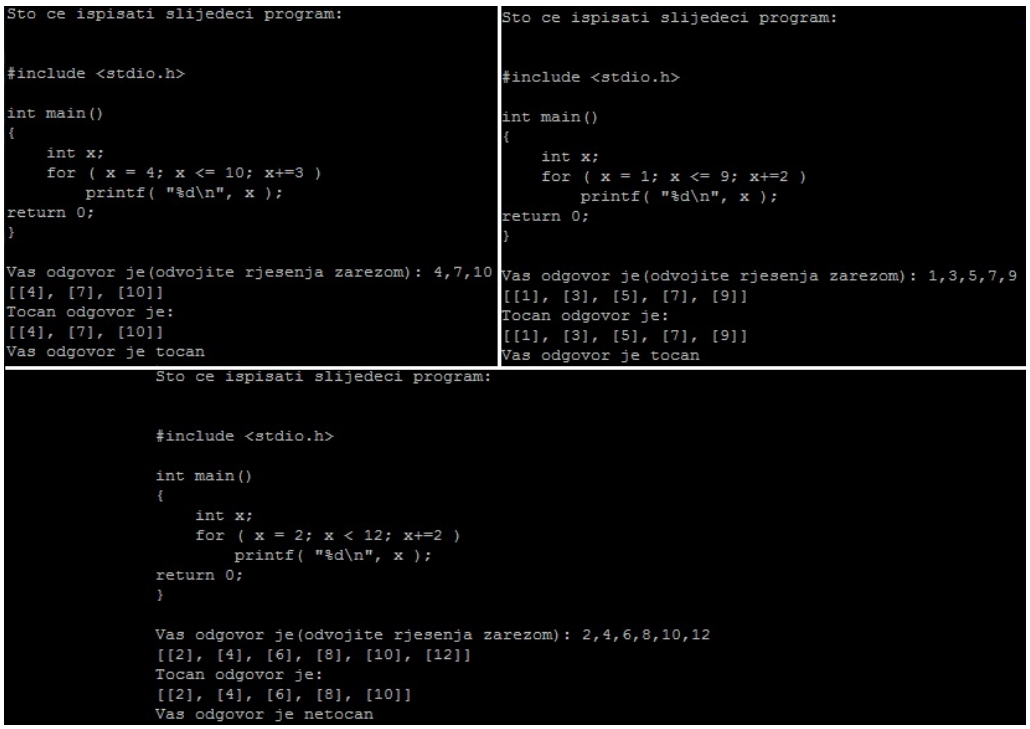
Slika 6.3: Primjeri trećeg zadatka s točnim i netočnim rješenjima

6.5. Peti zadatak

Peti zadatak isto obrađuje nastavnu jedinicu Programska petlja *for*. Razlike u odnosu na gornji zadatak je su da je interval `inicijalizacije` od 9 do 15, `uvjet` od 0 do 5 i kod `promjene_vrijednosti` koriste se –, -=2, -=3. Operatori usporedbe kod `uvjeta` su > i >=. Studenti moraju prepoznati da se radi o padajućim nizovima i od njih se traži da ispišu koji će se brojevi ispisati na ekranu.

6.6. Šesti zadatak

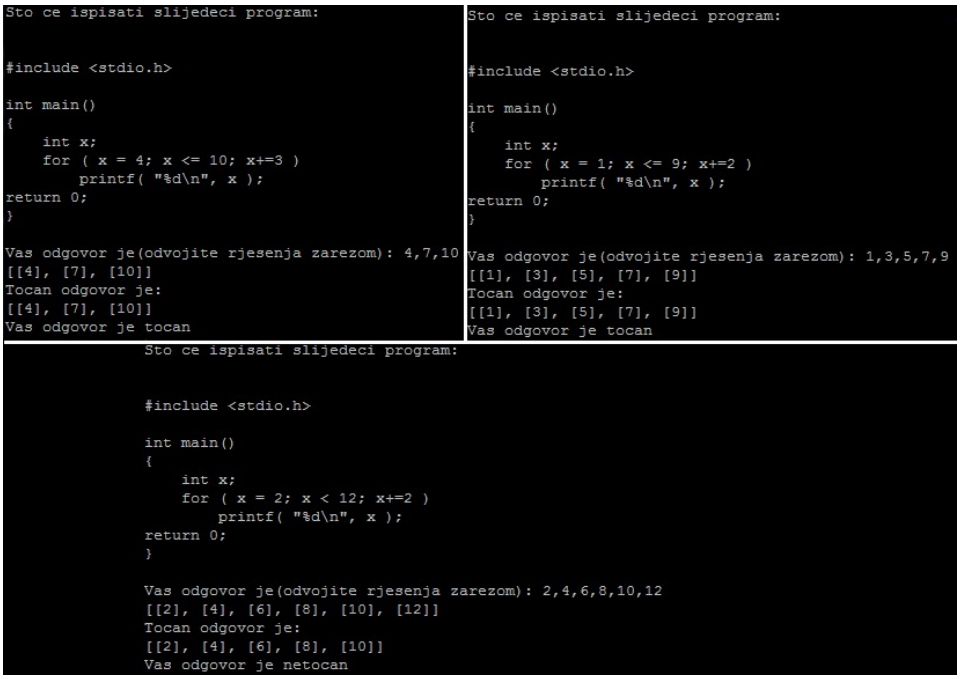
U šestom zadatku se obrađuje Programska petlja *do-while*. Petlja kod koje se uvjet ponavljanja ispituje na kraju bloka naredbi. Početne vrijednosti kontrolnih varijabli *i* i *j* je broj jedan i on se ne mijenja. Blok naredbi unutar petlje je u većini nepromjenjiv, jedini dio koji se mijenja je u računu za kontrolnu varijablu *j*. Hoće li se povećati zbrajanjem ili množenjem. Najviše se generira kod `uvjeta` sa svakim pokretanjem programa. U `uvjetu` se koriste logički operatori: logički i (AND) i logički ili (OR) oznaka && i ||. Uz logičke operatore generiraju se brojevi koje `uvjet` mora zadovoljavati. Jedan uvjet je u intervalu od 5 do 8, a drugi od 5 do 10. Program ispisuje po dvije vrijednosti dokle god je `uvjet` istinit.



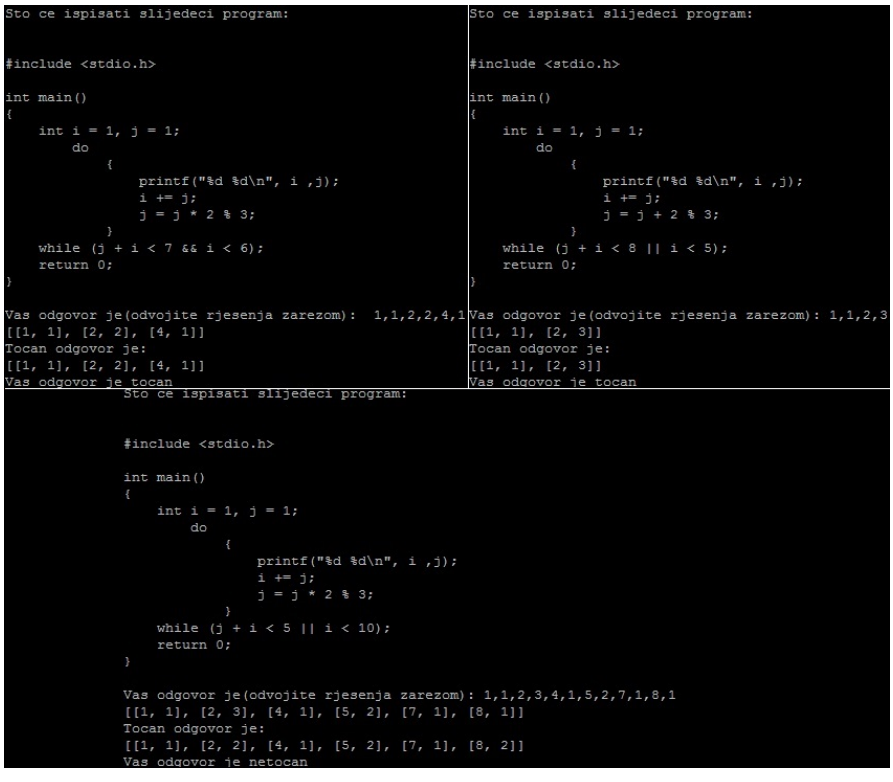
Slika 6.4: Primjeri četvrtog zadatka s točnim i netočnim rješenjima

6.7. Sedmi zadatak

Sedmi zadatak obrađuje nastavnu jedinicu Deklaracija varijabli. `Lista formata` za tip varijabli koja se koristi u ovom programu je float. Program izračunava broj π, pod komentare je napisan i sam broj π do 15-te decimale kao pomoć studentima u zadatku. Vrijednost varijable je poznata, a studenti moraju prepoznati na koju je sve načine program želi ispisati. Načini ispisivanja vrijednosti varijable su ti koji se generiraju svakim pokretanjem programa. Ispis prve je u intervalu od 3 do 5, drugi od 1.2 do 4.5, treći od 1.5 do 4.10 i četvrti od 1.2 do 4.5.



Slika 6.5: Primjeri petog zadatka s točnim i netočnim rješenjima



Slika 6.6: Primjeri šestog zadatka s točnim i netočnim rješenjima

```
Sto ce ispisati slijedeci program:


#include <stdio.h>

Sto ce ispisati sljedeci program:

#include <stdio.h>
#include <math.h>
int main()
{
    M_PI;/*dobije se iznos za pi=3.141592653589793...*/
    printf("%5f %4.3f %1.5f %4.4f\n", M_PI, M_PI, M_PI, M_PI);
    return 0;
}

Vas odgovor je(odvojite rjesenja zarezom): 3.141593,3.142,3.14159,3.1416
[[3.141593, 3.142, 3.14159, 3.1416]]
Tocan odgovor je:
[[3.141593, 3.142, 3.14159, 3.1416]]
Vas odgovor je tocan
```

```
Sto ce ispisati sljedeci program:

#include <stdio.h>
#include <math.h>
int main()
{
    M_PI;/*dobije se iznos za pi=3.141592653589793...*/
    printf("%3f %3.4f %3.7f %1.5f\n", M_PI, M_PI, M_PI, M_PI);
    return 0;
}

Vas odgovor je(odvojite rjesenja zarezom): 3.141593,3.1416,3.1415927,3.14159
[[3.141593, 3.1416, 3.1415927, 3.14159]]
Tocan odgovor je:
[[3.141593, 3.1416, 3.1415927, 3.14159]]
Vas odgovor je tocan
```

```
Sto ce ispisati sljedeci program:

#include <stdio.h>
#include <math.h>
int main()
{
    M_PI;/*dobije se iznos za pi=3.141592653589793...*/
    printf("%4f %3.5f %4.9f %3.3f\n", M_PI, M_PI, M_PI, M_PI);
    return 0;
}

Vas odgovor je(odvojite rjesenja zarezom): 3.141593,3.141593,3.141591654,3.142
[[3.141593, 3.141593, 3.141591654, 3.142]]
Tocan odgovor je:
[[3.141593, 3.14159, 3.141592654, 3.142]]
Vas odgovor je netocan
```

Slika 6.7: Primjeri sedmog zadatka s točnim i netočnim rješenjima

7. Priprema za nastavnu jedinicu

Tema je predviđena za obradu u prirodoslovno-matematičkoj gimnaziji. Za obradu su potrebna dva školska sata. Nastavna cjelina je Programske petlje, a nastavna jedinica Programska petlja *for*. Oblici rada koji se koriste za vrijeme predavanja su: frontalni, individualni i rad u grupama. Nastavne metode koje koristim u radu su metoda pisanja i razgovora, te metoda demonstracije (računalna). Korelacija je samo s jednim predmetom i to je matematika. Nastavna pomagala koje koristim za vrijeme predavanja su ploča, kreda, interaktivna ploča, laser i računala.

- ***Obrazovni zadaci (kognitivni, spoznajni - što će učenici znati/moći napraviti nakon sata): kod for petlji se uvjet ponavljanja nalazi na početku niza naredbi, znat će da se naredba for najčešće primjenjuje u zadacima u kojima je broj ponavljanja unaprijed poznat, kakav oblik ima petlja, napisati programski kod za nekoliko jednostavnih zadataka, niz naredbi unutar programske petlje ne mora se izvršiti nijedanput.***
- ***Funkcionalni zadaci (psihomotorni, djelatni - koje će sposobnosti učenici razvijati na satu): sposobnost usmenog i pismenog izražavanja, logičko zaključivanje, uporaba već stečenog znanja***
- ***Odgojni zadaci (afektivni, doživljajni - koje će odgojne vrijednosti učenici usvajati tijekom sata): aktivno slušanje, izražavanje vlastitog mišljenja, uvažavanje tuđeg mišljenja, kolegijalnost, poticati razvoj radnih navika***

7.1. Uvodni dio sata

Na početku sata profesor će zamoliti učenike da ostave malo mjesta na početku bilježnice za naslov današnje teme jer će do samog naslova doći tijekom predavanja. Često u samom procesu programiranja dolazi do potrebe za ponavljanjem nekih dijelova programa. Ukoliko, recimo trebamo od korisnika tražiti da upiše niz od deset brojeva, na-

redbu za učitavanje treba pisati deset puta. Ako slučajno s tim brojevima treba napraviti još i nekoliko operacija, program postaje jako dugačak i dosta nepregledan. Jednostavnije i preglednije je ako naredbu učitavanja i naredbe s tim brojevima napišemo samo jednom, a pri izvršavanju programa se pozove željeni broj puta. ***Ima li netko možda ima ideju kako se zovu te naredbe ponavljanja nekih dijelova programa?*** Ulazi se u raspravu s učenicima gdje se traži odgovor programske petlje. Nakon što se dođe do tok naziva, zapisujem na ploču veliki naslov Programske petlje. Znači, za ponavljanje dijelova programa upotrebljavaju se naredbe ponavljanja ili programske petlje. Ovisno o mjestu ispitivanja uvjeta mogu se podijeliti na:

- ***programske petlje s ispitivanjem uvjeta na početku***
- ***petlje kod kojih se uvjet ponavljanja nalazi na kraju niza naredbi koje se ponavljaju***

Ako se uzmu u obzir ta dva kriterija u programskom jeziku C razlikuju se:

- ***programske petlje for i while kod kojih se uvjet ponavljanja nalazi na početku niza naredbi***
- ***programska petlja do-while, kod koje se uvjet ponavljanja nalazi na kraju niza naredbi***

Budući da sad znamo kada se koriste koje naredbe, može li se zaključiti koje petlje se moraju izvršiti barem jedanput a koje niti jedanput? Navodi se učenike da se dođe do zaključka kako ***for*** i ***while*** petlje, s obzirom na to da imaju na početku niz naredbi, ako se uvjet ne pokaže istinitim, ne moraju izvršiti nijedanput, dok ***do-while*** petlja će se izvršiti barem jedanput jer se uvjet ponavljanja nalazi na kraju niza naredbi.

7.2. Glavni dio sata

Zapisujem novi naslov na ploču Programska petlja *for*. Naredba ***for*** najčešće se koristi kod zadataka u kojima je broj ponavljanja unaprijed poznat. Svrstava se u najopćenitije vrste petlji i najčešće se upotrebljava. Oblik u kojem se koristi je sljedeći (koristi se interaktivna ploča gdje je prikazano sljedeće):

- ***for (inicijalizacija; uvjet; promjena_vrijednosti)***
  {
  blok naredbi;
  }

Na narednoj stranici se nalazi sljedeće:

- ***inicijalizacija -postavljanje kontrolne varijable (i ili dodatnih varijabli) na početnu vrijednost***
- ***uvjet - uvjet koji kontrolna varijabli mora zadovoljiti da bi se izvršile naredbe u petlji (blok naredbi)***
- ***promjena_vrijednosti - dio u kojem se definira način promjene stanja kontrolne varijable***

Kao što ste primijetili dosta se spominje kontrolna varijabla. ***Što radi kontrolna varijabla?*** Ona kontrolira broj prolazaka petljom, a deklarira se kao i sve ostale varijable koje se rabe u programu. Vrijednost joj se mijenja automatski svakim prolaskom kroz petlju, budući da je to definirano s promjenom_vrijednosti.

Tijek izvršavanja petlje for je sljedeći:

1. Ako je prisutna inicijalizacija, kontrolna varijabla (i /ili dodatne varijable) se postavlja na početnu vrijednost (ako je to izraz, izračuna se njegova vrijednost, a nakon toga se ta vrijednost pridruži kontrolnoj varijabli)

2. Ako je prisutan uvjet, provjerava se njegova istinitost. Ako je uvjet zadovoljen (istinit) izvodi se blok naredbi

3. Ako je prisutna promjena_vrijednosti, promijeni se vrijednost kontrolne varijable. Program se vraća na početak petlje te se ona ponavlja od točke 2

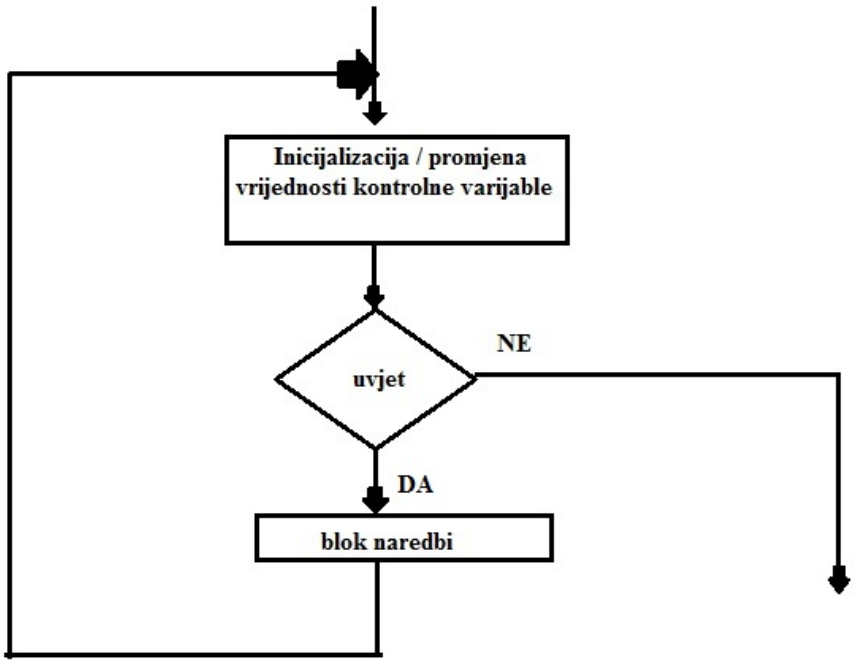
4. Ako je vrijednost uvjeta neistina, blok naredbi se preskače i program se nastavlja prvom naredbom iza bloka.

Prelazi se zatim na sijedeći slajd. Dakle dijagram tijeka petlje ***for*** nalazi se na Slika 7.1

Na sljedećem slajdu se nalazi prvi primjer. Petlja:

- ***for (i=1;i<=10 ; i++)***
  printf(" \n Danas učimo for petlju.");

na zaslonu ekrana će se deset puta ispisati rečenica " Danas učimo for petlju.", svaki put u novom redu. ***Zna li netko zašto svaki put u novom redu?*** Neki od učenika su došli do zaključka da je to zbog konstante \n ispred početka rečenice. ***Zašto se ispisuje deset puta?*** Varijabla i je kontrolna varijabla čija se vrijednost najprije postavlja na 1, Nakon toga provjerava se uvjet (da je i manji od 10). Zadani uvjet je istinit (1 je manji



Slika 7.1: Dijagram tijeka for petlje

od 10), na zaslonu se ispiše zadana rečenica. Zatim se, zbog promjene vrijednosti kontrolne varijable (i++), ta vrijednost poveća za 1(dakle postaje 2) i izvođenje se vraća na početak petlje. Nakon što se vrijednost kontrolne varijable postavi na 11, uvjet neće biti ispunjen i program kreće s izvršavanjem prve naredbe iza naredbe ***for***.

Kao što možete primjetiti vitičaste zagrade { } za oznaku početka i kraja bloka nisu potrebne. ***Znate li zašto?*** Dolazi se zajedno do zaključka da je to zato što se ponavlja samo jedna naredba, pa nije potrebno stavljati naredbe. Ako ima problema sa zaključkom postavlja se potpitanje: ***koliko naredbi se nalazi u petlji?*** Dakle, upotrebljavaju se samo u slučajevima kada se treba ponavljati više naredbi. ***Ima li kakvih pitanja i nejasnoća?*** Ako ima, zajednički pokušavamo doći do zaključaka, ako nema zamolim učenike da napišu i isprobaju program na svom računalu. Kada završe s primjerom 1., prelazimo na primjer 2.

- ***Petlja: for (i=1;i<=10 ; i++)***
  printf("%d", i);

Nakon što smo se upoznali s for petljom u prvom primjeru, što mislite da će na vašem ekranu ispisati ovaj primjer? Učenici izlažu svoja razmišljanja i zajedničkom raspravom dolazimo do zaključka da će na zaslonu biti ispisani brojevi 50, 51, 52,53,54,55,56,57,58 i 59. Većina učenika je pretpostavila da će se ispisati i broj 60, pa ih se upozorava da se ovdje koristio znak "<", a ne "<=" i da se zato ne ispisuje broj 60. Učenici zatim ispisuju kod na svome računalu. Zatim na sljedećem slajdu prelazimo na primjere izgenerirane uz pomoć mog programa Slika 7.2 .

Sto ce ispisati sljedeci program:

```
#include <stdio.h>

int main()
{
    int x;
    for ( x = 0; x <= 12; x+=2 )
        printf( "%d\n", x );
    return 0;
}
```

Sto ce ispisati sljedeci program:

```
#include <stdio.h>

int main()
{
    int x;
    for ( x = 15; x >= 5; x-- )
        printf( "%d\n", x );
    return 0;
}
```

Slika 7.2: Primjeri zadataka for petlje generirani mojim programskim kodom

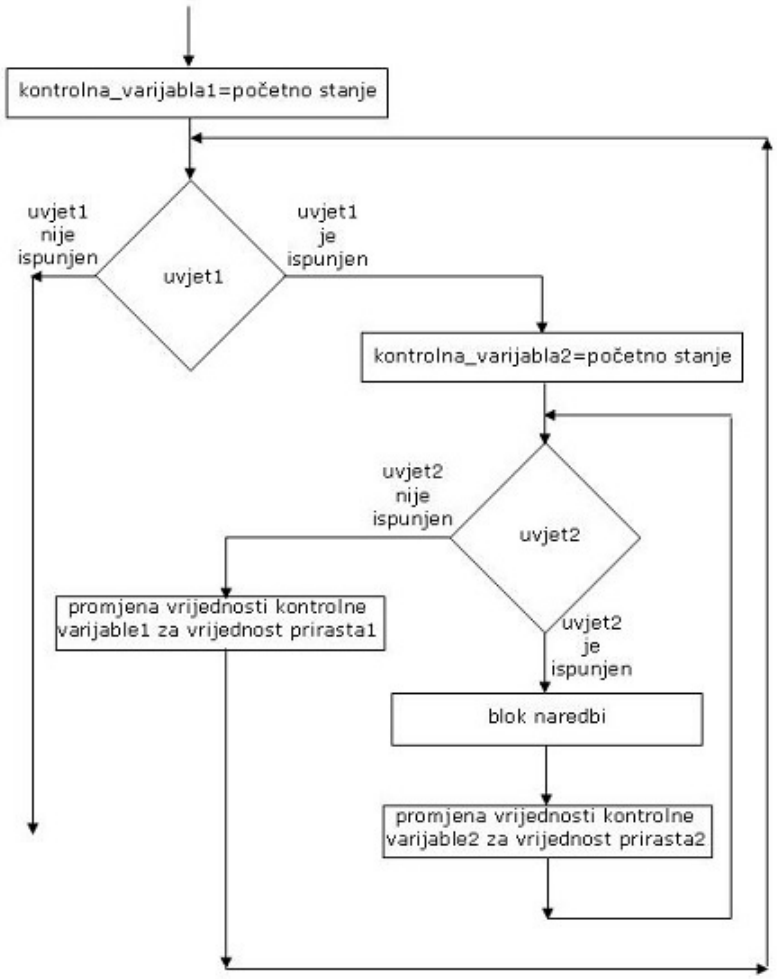
Raspravlja se o primjerima i njihovim rješenjima, te zatim svaki učenik isprobava na svom računalu navedene kodove.

Sljedeći naslov koji zapisujem na ploču je Ugnježđivanje petlje for. Petlje for se mogu pisati jedna unutar druge, tj. biti ugniježđene jedna unutar druge.

- ***for (inicijalizacija1; uvjet1; promjena_vrijednosti1)***
  for (inicijalizacija1; uvjet1; promjena_vrijednosti1)
  {
  blok naredbi;
  }

Pri ulazu u vanjsku petlju, njezina kontrolna varijabla poprima početnu vrijednost. Provjerava se uvjet vanjske petlje, pa ako je ispunjen, ulazi se u unutarnju petlju. Pri ulazu u unutarnju petlju, kontrolna varijabla unutarnje petlje poprima početnu vrijednost, provjerava se uvjet unutarnje petlje, pa ako je ispunjen izvršava se blok naredbi unutarnje petlje. Kada uvjet unutarnje petlje nije ispunjen, izlazi se iz nje i nastavlja se izvršavati vanjska petlja. Provjerava se uvjet vanjske petlje pa ako je ispunjen, ulazi se u unutarnju petlju. Sada se ponavlja postupak izvršavanja unutarnje petlje. Sve se

ponavlja do trenutka kada uvjet vanjske petlje više nije ispunjen. Dakle, ako se vanjska petlja izvršava n puta, a unutrašnja m, ugniježđene petlje izvršit će se n*m puta.



Slika 7.3: Dijagram tijeka ugnježđivanja for petlje

Primjer 3.

- ***int i,j; for (i=0; i<=6; i+=5)***
  for (j=1; i<=7; j+=3)
  printf("\n %d, %d", i, j);

Vrijednosti varijabli i i j mijenjaju se ovako: Za i=0 (početna vrijednost i varijable) varijabla j poprima vrijednosti 1, 4,7, te se na zaslonu ispisuju vrijednosti:

- ***0, 1***
- ***0, 4***
- ***0, 7***

Tada unutrašnja petlja staje, jer ne ispunjava uvjet više. Povećava se vrijednost kontrolne varijable i (i=5), pa se ponovo izvršava unutarnja petlja i poprima vrijednosti: 1,4 i 7. Ispis na zaslonu je sljedeći:

- ***5, 1***
- ***5, 4***
- ***5, 7***

Sljedeće povećanje kontrolne varijable i je na 10 pa uvjet nije istinit što znači da prestaje izvršavanje petlji.Ulazim u raspravu s učenicima ako ima pitanja, dok oni samostalno isprobavaju kod na svome računalu.

7.3. Završni dio sata

Pred kraj drugog sata ponavljamo s čim smo se tijekom sata susreli, te rješavamo još nekoliko primjera sa slajdova ugnježđivanja ako za to ima vremena.

LITERATURA

[1] Python. http://hr.wikipedia.org/wiki/Python_(programski_jezik), 15.07.2014 u 15:00.

[2] Web2.0. http://hr.wikipedia.org/wiki/Web_2.0, 15.07.2014 u 15:00.

[3] Interaktivni program. http://en.wikipedia.org/wiki/Interactive_computing, 15.07.2014 u 16:00.

[4] Operatori u progrmaskom jeziku c. http://www.tutorialspoint.com/cprogramming/c_operators.htm, 15.07.2014 u 16:00.

[5] Popis online stranica s interaktivnim programima. https://www.fractuslearning.com/2011/12/14/programming-for-kids/, 15.07.2014 u 16:00.

[6] Monk J. Du Boulay B, O'Shea T. *The Black Box Inside the Glass Box. In Studying the Novice Programmer*. Hillsdale - New Jersey, 1989.

[7] Du Boulay:. Lemut, Dettori. *Cognitive Models and Intelligent Environments for Learning Programming*. Springer-Verlag, 1993.

[8] Dalbey J. Linn M.C. *Cognitive Consequences of Programming Instruction, in Studying the Novice Programmer*. Hillsdale - New Jersey, 1989.

[9] David Ascher Mark Lutz. *Learning Python*. O'Reilly Media, Inc, 2003.

Dodatak A
Programski kodovi zadataka

Zadatak 1

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import random, os, sys, decimal

code00 = r"""

#include <stdio.h>
int main()
{{
    int x = {n}, y = {m}, z = {k}, a;
    a = {n} {a} {m} {b} {k};
    printf("%d\n", a);
    return 0;
}}
"""

def savecode(tempc,code):
    f = open(tempc,"w")
    f.write(code)
    f.close()

def generator():
    tempc = "temp.c"
    cmd = "gcc %s && ./a.out" % tempc

    num1 = random.randint(5, 10)
    num2 = random.randint(5, 10)
    num3 = random.randint(1, 5)

    mul = (num1 * num2)
    div = (num1 / num2)
    add = (num1 + num2)
    sub = (num1 - num2)
    mod = (num1 % num2)

    correct = mul, div, add, sub, mod

    opMUL = ("-")
    opDIV = ("+")
    opADD = ("*")
    opSUB = ("/")
    opMOD = ("%")

    operators1 = opMUL, opADD, opDIV
    operators2 = opMOD, opSUB

    op1 = random.choice(operators1)
    op2 = random.choice(operators2)

    zamjene = dict(n = num1, m = num2, k = num3, a = op1 , b = op2)

    code = code00.format(**zamjene)

    savecode( tempc, code )
    return (cmd,code)

def check():
    import re
    def filt( L ):
        L = filter( lambda x: x<>'', L )
        L = map( int, L )
        return L

    cmd,code = generator()
    print "Sto ce ispisati sljedeci program:"
    print code
    odg = raw_input("Vas odgovor je:");
    odg1=[]
    odg1 = [int(i) for i in odg.split(',')]
    matrica = []
    while odg1 != []:
        matrica.append(odg1[:1])
        odg1 = odg1[1:]
    print matrica
    #print "snimljen je u datoteku, tako da se prevodi i izvrsava pomocu:"
    #print cmd
    print "Tocan odgovor je:"
    rezs = os.popen(cmd).readlines()
    #print rezs
    #print "ucitamo output u python listu:"
    rezs = map(lambda x: filt( re.split("\s*",x) ), rezs)
```

```
83      print rezs
84      if (matrica == rezs):
85          print "Vas odgovor je tocan"
86      else:
87          print "Vas odgovor je netocan"
88
89  if __name__ == '__main__':
90      check()
```

Zadatak 2

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import random, os, sys, decimal

code00 = r"""

#include <stdio.h>
int main()
{{
    int x = {n}, y = {m}, z = {k}, a;
    a = {n} {b} {m} {a} {k};
    printf("%d\n", a);
    return 0;
}}
"""

##########################################

def savecode(tempc,code):
    f = open(tempc,"w")
    f.write(code)
    f.close()

def generator():
    tempc = "temp.c"
    cmd = "gcc %s && ./a.out" % tempc

    num1 = random.randint(12, 18)
    num2 = random.randint(2, 3)
    num3 = random.randint(1, 4)

    mul = (num1 * num2)
    div = (num1 / num2)
    add = (num1 + num2)
    sub = (num1 - num2)
    mod = (num1 % num2)

    correct = div, add, sub, mod

    opDIV = ("+")
    opADD = ("*")
    opSUB = ("/")
    opMOD = ("%")

    operators1 = opADD, opDIV
    operators2 = opMOD, opSUB

    op1 = random.choice(operators1)
    op2 = random.choice(operators2)

    zamjene = dict(n = num1, m = num2, k = num3, a = op1 , b = op2)

    code = code00.format(**zamjene)

    savecode( tempc, code )
    return (cmd,code)

def check():
    import re
    def filt( L ):
        L = filter( lambda x: x<>'', L )
        L = map( int, L )
        return L

    cmd,code = generator()
    print "Sto ce ispisati sljedeci program:"
    print code
    odg = raw_input("Vas odgovor je: ");
    odg1=[]
    odg1 = [int(i) for i in odg.split(',')]
    matrica = []
    while odg1 != []:
        matrica.append(odg1[:1])
        odg1 = odg1[1:]
    print matrica
    #print "snimljen je u datoteku, tako da se prevodi i izvrsava pomocu:"
    #print cmd
    print "Tocan odgovor je:"
    rezs = os.popen(cmd).readlines()
    #print rezs
    #print "ucitamo output u python listu:"
```

```
    rezs = map(lambda x: filt( re.split("\s*",x) ), rezs)
    print rezs
    if (matrica == rezs):
        print "Vas odgovor je tocan"
    else:
        print "Vas odgovor je netocan"

if __name__ == '__main__':
    check()
```

Zadatak 3

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import random, os, sys, decimal

code00 = r"""

#include <stdio.h>
int main()
{{
    int x = {n}, y = {m}, z = {o}, i, j, f;
    if(y{i}z)
    {{
        i = {n} {a} {m} {b} {o};
        printf("%d\n",i);
    }}
    else if (y{k}z)
        {{
            j = {n} {b} {m} {c} {o};
            printf("%d\n", j);
        }}
        else
            {{
                f = {n} {c} {m} {a} {o};
                printf("%d\n",f);
            }}
return 0;
}}
"""

##########################################

def savecode(tempc,code):
    f = open(tempc,"w")
    f.write(code)
    f.close()


def generator():
    tempc = "temp.c"
    cmd = "gcc %s && ./a.out" % tempc

    num1 = random.randint(12, 18)
    num2 = random.randint(2, 3)
    num3 = random.randint(1, 4)

    mul = (num1 * num2)
    div = (num1 / num2)
    add = (num1 + num2)
    sub = (num1 - num2)
    mod = (num1 % num2)

    equ = (num1 == num2)
    neq = (num1 != num2)
    gre = (num1 > num2)
    les = (num1 < num2)
    greq = (num1 >= num2)
    lesq = (num1 <= num2)

    correct = mul, div, add, sub, mod, equ, neq, gre, les, greq,lesq

    opMUL = ("-")
    opDIV = ("+")
    opADD = ("*")
    opSUB = ("/")
    opMOD = ("%")

    opequ = ("==")
    opneq = ("!=")
    opgre = (">")
    oples = ("<")
    opgreq = (">=")
    oplesq = ("<=")

    operators1 = opMUL, opADD, opDIV
    operators2 = opMOD, opSUB
    operators3 = opSUB, opADD, opDIV

    operatorsA = opgre, oples, opgreq, oplesq
    operatorsB = opequ, opneq

    op1 = random.choice(operators1)
```

```
    op2 = random.choice(operators2)
    op3 = random.choice(operators3)

    opA = random.choice(operatorsA)
    opB = random.choice(operatorsB)

    zamjene = dict(n = num1, m = num2, o = num3, a = op1 , b = op2, c = op3, i = opA, k = opB)

    code = code00.format(**zamjene)

    savecode( tempc, code )
    return (cmd,code)

def check():
    import re
    def filt( L ):
        L = filter( lambda x: x<>'', L )
        L = map( int, L )
        return L

    cmd,code = generator()
    print "Sto ce ispisati sljedeci program:"
    print code

    odg = raw_input("Vas odgovor je: ");
    odg1=[]
    odg1 = [int(i) for i in odg.split(',')]
    matrica = []
    while odg1 != []:
        matrica.append(odg1[:1])
        odg1 = odg1[1:]
    print matrica

    #print "snimljen je u datoteku, tako da se prevodi i izvrsava pomocu:"
    #print cmd
    print "Tocan odgovor je:"
    rezs = os.popen(cmd).readlines()
    #print rezs
    #print "ucitamo output u python listu:"
    rezs = map(lambda x: filt( re.split("\s*",x) ), rezs)
    print rezs
    if (matrica == rezs):
        print "Vas odgovor je tocan"
    else:
        print "Vas odgovor je netocan"

if __name__ == '__main__':
    check()
```

Zadatak 4

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import random, os, sys, decimal

code00 = r"""

#include <stdio.h>

int main()
{{
    int x;
    for ( x = {n}; x {i} {m}; x{j} )
        printf( "%d\n", x );
    return 0;
}}
"""

##########################################

def savecode(tempc,code):
    f = open(tempc,"w")
    f.write(code)
    f.close()


def generator():
    tempc = "temp.c"
    cmd = "gcc %s && ./a.out" % tempc

    num1 = random.randint(0, 5)
    num2 = random.randint(9,15)

    les = (num1 < num2)
    lesq = (num1 <= num2)

    correct = les, lesq

    oa = ("++")
    ob = ("+=2")
    oc = ("+=3")

    oples = ("<")
    oplesq = ("<=")

    operators1 = oa, ob, oc

    operatorsA = oples, oplesq

    op1 = random.choice(operators1)

    opA = random.choice(operatorsA)

    zamjene = dict(n = num1, m = num2, i = opA , j = op1)

    code = code00.format(**zamjene)

    savecode( tempc, code )
    return (cmd,code)

def check():
    import re
    def filt( L ):
        L = filter( lambda x: x<>'', L )
        L = map( int, L )
        return L

    cmd,code = generator()
    print "Sto ce ispisati sljedeci program:"
    print code
    odg = raw_input("Vas odgovor je(odvojite rjesenja zarezom): ")
    odg1=[]
    odg1 = [int(i) for i in odg.split(',')]
    matrica = []
    while odg1 != []:
        matrica.append(odg1[:1])
        odg1 = odg1[1:]
    print matrica
    #print "snimljen je u datoteku, tako da se prevodi i izvrsava pomocu:"
    #print cmd
    print "Tocan odgovor je:"
    rezs = os.popen(cmd).readlines()
    #print "ucitamo output u python listu:"
```

```
83      rezs = map(lambda x: filt( re.split("\s*",x) ), rezs)
84      print rezs
85      if (matrica == rezs):
86          print "Vas odgovor je tocan"
87      else:
88          print "Vas odgovor je netocan"
89
90  if __name__ == '__main__':
91      check()
```

Zadatak 5

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import random, os, sys, decimal

code00 = r"""

#include <stdio.h>

int main()
{{
    int x;
    for ( x = {m}; x {i} {n}; x{j} )
        printf( "%d\n", x );
    return 0;
}}
"""

##########################################

def savecode(tempc,code):
    f = open(tempc,"w")
    f.write(code)
    f.close()


def generator():
    tempc = "temp.c"
    cmd = "gcc %s && ./a.out" % tempc

    num1 = random.randint(0, 5)
    num2 = random.randint(9,15)

    les = (num1 > num2)
    lesq = (num1 >= num2)

    correct = les, lesq

    oa = ("--")
    ob = ("-=2")
    oc = ("-=3")

    oples = (">")
    oplesq = (">=")

    operators1 = oa, ob, oc

    operatorsA = oples, oplesq

    op1 = random.choice(operators1)

    opA = random.choice(operatorsA)

    zamjene = dict(n = num1, m = num2, i = opA , j = op1)

    code = code00.format(**zamjene)

    savecode( tempc, code )
    return (cmd,code)

def check():
    import re
    def filt( L ):
        L = filter( lambda x: x<>'', L )
        L = map( int, L )
        return L

    cmd,code = generator()
    print "Sto ce ispisati sljedeci program:"
    print code

    odg = raw_input("Vas odgovor je(odvojite rjesenja zarezom): ")
    odg1=[]
    odg1 = [int(i) for i in odg.split(',')]
    matrica = []
    while odg1 != []:
        matrica.append(odg1[:1])
        odg1 = odg1[1:]
    print matrica

    #print "snimljen je u datoteku, tako da se prevodi i izvrsava pomocu:"
    #print cmd
```

```
    print "Tocan odgovor je:"
    rezs = os.popen(cmd).readlines()
    #print rezs
    #print "ucitamo output u python listu:"
    rezs = map(lambda x: filt( re.split("\s*",x) ), rezs)
    print rezs
    if (matrica == rezs):
        print "Vas odgovor je tocan"
    else:
        print "Vas odgovor je netocan"

if __name__ == '__main__':
    check()
```

Zadatak 6

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import random, os, sys, decimal

code00 = r"""

#include <stdio.h>

int main()
{{
    int i = 1, j = 1;
    do
        {{
            printf("%d %d\n", i ,j);
            i += j;
            j = j {a} 2 % 3;
        }}
    while (j + i < {n} {c} i < {m});
    return 0;
}}
"""

##########################################

def savecode(tempc,code):
    f = open(tempc,"w")
    f.write(code)
    f.close()


def generator():
    tempc = "temp.c"
    cmd = "gcc %s && ./a.out" % tempc

    num1 = random.randint(5, 8)
    num2 = random.randint(5, 10)

    mul = (num1 * num2)
    sub = (num1 - num2)
    add = (num1 + num2)
    mod = (num1 % num2)

    oand = (num1 and num2)
    oor = (num1 or num2)

    correct = mul, sub, add, mod, oand,oor

    opDIV = ("+")
    opADD = ("*")
    opMOD = ("%")

    opAND = ("&&")
    opOR = ("||")

    operators1 = opDIV, opADD

    operatorsA = opAND, opOR


    op1 = random.choice(operators1)

    opA = random.choice(operatorsA)

    zamjene = dict(n = num1, m = num2, a = op1 , c = opA)

    code = code00.format(**zamjene)

    savecode( tempc, code )
    return (cmd,code)

def check():
    import re
    def filt( L ):
        L = filter( lambda x: x<>'', L )
        L = map( int, L )
        return L

    cmd,code = generator()
    print "Sto ce ispisati sljedeci program:"
    print code

```

```
    odg = raw_input("Vas odgovor je(odvojite rjesenja zarezom): ")
    odg1=[]
    odg1 = [int(i) for i in odg.split(',')]
    matrica = []
    while odg1 != []:
        matrica.append(odg1[:2])
        odg1 = odg1[2:]
    print matrica

    #print "snimljen je u datoteku, tako da se prevodi i izvrsava pomocu:"
    #print cmd
    print "Tocan odgovor je:"
    rezs = os.popen(cmd).readlines()
    #print rezs
    #print "ucitamo output u python listu:"
    rezs = map(lambda x: filt( re.split("\s*",x) ), rezs)
    print rezs
    if (matrica == rezs):
        print "Vas odgovor je tocan"
    else:
        print "Vas odgovor je netocan"

if __name__ == '__main__':
    check()
```

Zadatak 7

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import random, os, sys, decimal

code00 = r"""
#include <stdio.h>
#include <math.h>
int main()
{{
    M_PI;/*dobije se iznos za pi=3.141592653589793...*/
    printf("%{a}f %{b}.{c}f %{d}.{e}f %{g}.{h}f\n", M_PI, M_PI, M_PI, M_PI);
    return 0;
}}
"""

##########################################

def savecode(tempc,code):
    f = open(tempc,"w")
    f.write(code)
    f.close()


def generator():
    tempc = "temp.c"
    cmd = "gcc %s && ./a.out" % tempc

    num1 = random.randint(3, 5)
    num2 = random.randint(1, 4)
    num3 = random.randint(2, 5)
    num4 = random.randint(1, 4)
    num5 = random.randint(5, 10)
    num6 = random.randint(1, 4)
    num7 = random.randint(2, 5)

    zamjene = dict(a = num1, b = num2, c = num3, d = num4, e = num5, g = num6, h = num7)

    code = code00.format(**zamjene)

    savecode( tempc, code )
    return (cmd,code)

def check():
    import re
    def filt( L ):
        L = filter( lambda x: x<>'', L )
        L = map( float, L )
        return L

    cmd,code = generator()
    print "Sto ce ispisati sljedeci program:"
    print code

    odg = raw_input("Vas odgovor je(odvojite rjesenja zarezom): ")
    odg1=[]
    odg1 = [float(i) for i in odg.split(',')]
    matrica = []
    while odg1 != []:
        matrica.append(odg1[:4])
        odg1 = odg1[4:]
    print matrica

    #print "snimljen je u datoteku, tako da se prevodi i izvrsava pomocu:"
    #print cmd
    print "Tocan odgovor je:"
    rezs = os.popen(cmd).readlines()
    #print rezs
    #print "ucitamo output u python listu:"
    rezs = map(lambda x: filt( re.split("\s*",x) ), rezs)
    print rezs
    if (matrica == rezs):
        print "Vas odgovor je tocan"
    else:
        print "Vas odgovor je netocan"

if __name__ == '__main__':
    check()
```

Izrada interaktivnih zadataka iz programiranja

Sažetak

Veliki problem predstavlja razumijevanje i vizualizacija apstraktnih procesa kako studentima tako i profesorima. Kako bi olakšao studentima i učenicima vježbanje programskog koda C izradio sam zadatke koji će im pomoći pritom. Profesorima također mogu pomoći zadaci pri izradi ispita za studente. Tri glavne karike lanca spoznajnog postignuća učenjem za Linn i Dalbey [8] su svojstva programskog jezika, vještina oblikovanja programa i opća vještina rješavanja problema, dok DuBoulay [6] moguće izvore poteškoća nalazi u: orijentaciji, apstraktnom stroju, notaciji, strukturama i pragmatici. Opisana je tradicionalna nastava programiranja koja je bila više usmjerena na profesora, a ne na studente. Kao zadnje poglavlje opisan je kognitivan način učenja i zašto je on važan u programiranju.

Treće poglavlje se bavi interaktivnim nastavom, zašto je ono važno i koja je razlika između Web1.0 i Web.20 servisa i internetskih sredstava. Opisano je kako se interaktivno učenje više usmjeruje na učenima,a manje na profesora i koje interaktivne alate sve profesori mogu koristiti profesori dok poučavaju. Poglavlje interaktivni program opisuje moderne programe koji se danas koriste i kreator može napraviti promjene u programu, iako je on već pokrenut.

U četvrtom poglavlju objašnjavam zašto je vizualizacija važna u nastavnim sadržajima kod programiranja. Kao primjere sam uzeo if i while petlju koji pokazuju kako vizualizacija može značajno pomoći studentu da se poistovjeti s problemom koji je postavljen pred njega.

Šesto poglavlje opisuje i objašnjava zadatke koje sam izradio u Pythonu. Napravio sam sedam zadataka koji pokrivaju gradiva Deklaracija varijabli, Aritmetički operatori, Višestruko uvjetno grananje, programska petlja *for* i Programska petlja *while*. Funkcija `savecode` otvara datoteku 'tempc' za pisanje, zapisuje u nju kod i zatvara datoteku. Funkcija `generator` je najvažnija i drugačija je u svim zadacima. U njoj se generiraju brojevi i operatori koji se koriste u zadacima programskog jezika C. Kako uz pomoć ugrađene funkccije `dict` mijenjam vrijednosti u zadatku. Uz pomoć funkcije `check` program ispisuje kod C programskog jezika i od studenta traži unos rješenja. Zatim program sam izvršava zadatak i ispisuje točno rješenje na ekran. Za kraj program uspoređuje točno rješenje s rješenjem koje je student upisao i na ekranu ispisuje da li je njegovo rješenje točno ili netočno.