Predgovor

Kako naučiti programirati?

Kao nastavnik koji predaje uvodni kurs programiranja studentima koji u pravilu nemaju nikakvo ili skoro nikakvo predznanje, često sam bio prinuđen da odgovorim na upravo ovo pitanje. Student koji tek želi da zakorači u svijet programiranja nažalost neće uvijek naići na onu vrstu podrške koja mu/joj je potrebna. Tako će mnogi reći da se programiranje ne može tek tako naučiti – ili si se rodio s tim talentom koji samo trebaš otkriti u sebi ili nisi. Drugi pak tvrde da se naučiti programirati ne može bez mnogo godina truda i odricanja, pa oni koji nisu počeli još u osnovnoj školi kada dođu do fakulteta nemaju nikakvih šansi.

A ja sam baš upoznao i vidio mnoge studente koji u tom svom prvom septembru[1] nisu imali ama baš nikakvo predznanje, ni for petlju nisu znali, da bi do januara ili februara zaista naučili sve ono što treba i bili programeri. Tu je važno odgovoriti na pitanje šta tačno znači "biti programer" ili "znati programirati"? Istina je da učenju programiranja nema kraja, čovjek nikada ne može prestati učiti, uvijek postoje stvari koje ne poznajete.

[1] https://en.wikipedia.org/wiki/Eternal_September

Po meni osoba koja "zna programirati" treba *znati rješavati bilo kakav postavljeni problem*, uključujući i probleme kakve nikada ranije nije vidjela. Ovdje nisam slučajno upotrijebio riječ "rješavati" umjesto "riješiti". Neke discipline su pogodne za ljude koji znaju prepoznati taj jedan "trik" koji predstavlja rješenje zadatka. Programiranje nije jedna od njih.

Kada rješavate programski zadatak vaš prvi cilj je da složeni problem razbijete na više jednostavnijih, dok ne dođete do onoga što znate i možete riješiti. Pri tome je najvažnije da ne odustanete, da se ne prepadnete problema, nego da hrabro krenete u rješavanje korak po korak. Ako postoje neki koraci koje ne znate, možete ih preskočiti – neki drugi član vašeg programerskog tima će možda to uraditi umjesto vas.

Osoba koja na prvi pogled na tekst zadatka počne planirati, crtati, pisati kôd je osoba koja zna programirati. Ne kažem da zna *dobro* pro-

gramirati – to je vještina koja se vjerovatno uči čitav život. Ali ovo je neki početak.

Posmatrao sam mlade ljude koji su uspjeli naučiti programiranje, ali i mnoge koji nisu, i uočio sam da postoje neka pravila. Da bi se uspješno savladalo programiranje bitno je razumijevanje teorije, naravno. Ali sama teorija je beskorisna ako je ne znate primijeniti. Dalje, vrlo je korisno imati i gotova rješenja zadataka koja možete analizirati i iz njih nešto naučiti. No susreo sam se sa mnogima koji su imali oboje, a opet nisu znali programirati, jer su se fokusirali da napamet uče teoriju, pa čak i gotove komade kôda, bez razumijevanja kada se šta koristi. Ovo razumijevanje se, iz mog iskustva, može steći samo tako što se *samostalno pišu programi*. Baš kao što strani jezik možete naučiti samo tako što pričate sa strancima, kao što bicikl ne možete naučiti voziti tako što gledate druge kako voze, tako i programski jezik možete naučiti samo tako što pričate sa računarom.

Možete početi od toga da gotov program prepravljate da biste vidjeli šta i zašto radi. Po mom mišljenju tu je korisno i da vidite neke primjere koji ne rade, te da razumijete zašto ne rade. Ali, prije ili kasnije, moraćete sjesti pred prazan ekran razvojnog okruženja i početi kucati vaš program od onog prvobitnog "Hello world" programa. Taj korak je kao kada zaplivate prvi put bez zaštitnog pojasa, ili kada počnete voziti bicikl bez pomoćnih točkova. Dok to ne napravite, dok ne dovedete program od ideje koja ne postoji nigdje osim u vašoj glavi do programskog kôda koji radi tačno ono što ste zamislili, vi niste programer.

Pisanje programskog kôda najviše podsjeća na polagano objašnjavanje rješenja nekoj izrazito glupoj osobi. Pri tome moramo vrlo dobro znati koje je predznanje te osobe, koje riječi razumije a koje ne i šta pojedine riječi toj osobi znače kako bismo ih na pravi način upotrijebili.

Recimo da imamo nekog robota i želimo da taj robot ode i kupi hljeb. Izdaćemo mu jasnu naredbu:

1. Kupiti hljeb.

No naš robot ne zna šta znači "kupiti hljeb", pa moramo razraditi to rješenje, recimo ovako:

1. Izaći iz kuće.
2. Otići u prodavnicu.
3. Zatražiti hljeb od prodavača.
4. Uzeti hljeb.
5. Dati pare.
6. Vratiti se kući.

Ovo bi svaki robot morao znati, zar ne? Ali evo, ovaj naš robot se žali da ne zna šta znači izaći iz kuće, pa da i to objasnimo

1. Izaći iz kuće.
   (a) Doći do izlaznih vrata.
   (b) Pritisnuti šteku prema dolje.
   (c) Povući vrata prema sebi držeći šteku.
   (d) Pustiti šteku.
   (e) ...

itd. No ovdje smo pretpostavili da su vrata otključana. Da li možemo napraviti ovu pretpostavku? Ako ne, trebamo nakon koraka (e) ako vrata nisu otvorena koristiti ključ te ponoviti korake od (b) dalje. Suština je da moramo znati za šta je naš robot isprogramiran a za šta nije, pa koristiti samo one komande koje robot razumije. Također moramo razmisliti koje pretpostavke smijemo napraviti, a koje ne.

Upravo ovaj način razmišljanja je ono što je potrebno programeru. Najprije moramo biti savršeno načisto šta želimo da računar uradi. Računar nije ljudsko biće, nije nam prijatelj niti šef, to je poslušni robot koji obavlja zadatke koje smo mu dali. Zatim moramo taj zadatak saopštiti jezikom koji računar razumije. Na kraju, moramo biti načisto koje pretpostavke smo napravili i zašto.

Jedan ciničan pogled na ovo oličen je u poslovici "računari ne griješe, ljudi griješe". U suštini, računar uvijek radi tačno ono što mu je programer naredio programom. Ako dođe do greške, to je ili greška u programskom kodu tj. u pravilnoj upotrebi odabranog programskog jezika i biblioteka, ili je greška u pretpostavkama npr. o znanju korisnika. Na kraju, možda se ni korisnik ne pridržava uputstva za upotrebu?

Na sličan način možemo razmišljati o razlikama između programskih jezika. U nekom jeziku možda postoji riječ za sortiranje (slaganje nekog skupa po veličini), a u nekom drugom ne postoji, pa morate korak po korak objasniti proces slaganja po veličini. Ipak, kako budete učili različite jezike, vidjećete da su razlike između njih relativno male. Sam osnovni jezik posjeduje prilično mali broj različitih riječi. Za sve ostalo morate koristiti *gotove biblioteke* koje nude posebne naredbe (riječi) za specifične zadatke kao što je pokretanje robota ili 3D grafika.

U praksi programeri ne moraju pamtiti naredbe, nego ih mogu uvijek potražiti u dokumentaciji. Dakle, sposobnost koju programeri moraju imati je da neki konkretan zadatak objasne tj. prepričaju računaru kao da je u pitanju malo dijete, koristeći samo zadati skup riječi.

Kako steći tu sposobnost? Na žalost, na ovo pitanje nema jednostavnog odgovora, jer u suprotnom zanimanje "programer" ne bi

bilo jedno od najtraženih i najbolje plaćenih zanimanja danas. Mnogi stručnjaci u oblasti smatraju da se ova sposobnost stiče isključivo iskustvom, kroz rješavanje velikog broja različitih konkretnih programerskih problema. I autori ove knjige podržavaju ovakvo mišljenje. Baš kao što se iskustvo i stručnost pilota mjeri brojem sati leta, tako se i sposobnost programera može mjeriti brojem napravljenih programa. Pri tome je važno da su problemi koje rješavate raznovrsni. Ako napravite stotinu sličnih programa nećete naučiti ništa novo.

Stvar koju ćemo sada izjaviti vjerovatno se neće svidjeti mnogim studentima i učenicima, ali ključan momenat kod učenja programiranja je upravo ono "razbijanje glave" kada rješavate zadatak i ne znate kako da ga riješite. Kroz frustraciju problemom grade se nove sinapse (veze u mozgu), koje će vam sljedeći put olakšati rješavanje sličnog problema. Kada vam neko da gotovo rješenje, ta osoba vas je doslovno opljačkala, jer vam je oduzela priliku da naučite neki bitan koncept samostalnim rješavanjem. Dakle, naravno da trebate tražiti pomoć od kolega, ali bitno je da najprije pokušate sami. Kada zatražite pomoć, insistirajte da vam objasne tačno gdje ste pogriješili, zašto je to pogrešno i kako *vaš program* prepraviti tako da radi, a ne prihvatajte neko potpuno različito rješenje kao odgovor.

Kome je namijenjena ova knjiga?

Moramo ovdje posebno istaknuti jednu stvar: udžbenik koji je pred vama *nije* priručnik za programski jezik C (niti C++)! Ovaj udžbenik vam služi kako biste naučili programirati, a igrom slučaja kao osnovni jezik poduke koristi se programski jezik C. Dakle, ova knjiga je namijenjena totalnim početnicima koji o programiranju ne znaju ništa, mada može biti korisna i onim iskusnijim kako bi osvježili svoje znanje, te onima koji poznaju druge programske jezike ali ne i C. Po toj kombinaciji – za početnike, a koristi se C – smatramo da je ova knjiga jedinstvena.

Mnogi smatraju da se ovakva knjiga ne može napisati, odnosno da programski jezik C nije pogodan za početnike. Prilikom učenja novih stvari bitno je da učenika ne zatrpamo viškom informacija koje se ne mogu odjednom memorisati. Zbog toga se u objašnjavanju prave mnoga pojednostavljenja, detalji se preskaču ili ostavljaju za kasnije, koristi se laički, jednostavan jezik. Pojedinci sa početka pasusa tvrde da programski jezik C "ne nagrađuje aproksimacije" (aproksimacija: u matematici, približna vrijednost).

Autori ove knjige se ne slažu. Smatramo da je programski jezik C jezik kao i svaki drugi, s tim što posjeduje osobinu da programi koji sadrže greške često rade, nekada čak daju i tačne rezultate – ali nisu ispravni. Zato je potrebna određena disciplina da se pri pisanju

programa ne prave greške. Nadamo se da čemo čitaoce naučiti toj disciplini kroz savjete, primjere ispravnih i pogrešnih programa, jasne definicije. Ali naš osnovni cilj je da ljudi koji budu čitali ovu knjigu stvore *mentalni model* načina rada računara i izvršavanja programa koji možda nije potpuno ispravan, ali rezultira ispravnim programima u velikoj većini situacija.

Pored toga, važno nam je da čitaoci lakše savladaju druge programske jezike koji nemaju neke od neobičnih osobina C-a. Sve je to u cilju da što prije postanete programeri, makar u prvom periodu i ne budete baš dobri C programeri. Sigurni smo da ćete sve eventualne pogrešne zaključke koje stvorite razjasniti kasnije kako budete čitali napredniju literaturu iz C-a.

Programski jezik C++

Često ćete čuti da se programski jezici C i C++ posmatraju zajedno, pa se koristi fraza "C/C++". Bitno je da znate da su u pitanju dva potpuno različita programska jezika, sa različitim osobinama. Ustvari, programski jezik C++ je prvobitno zamišljen kao *nadskup programskog jezika C*, odnosno plan je bio da se svaki C program bez izmjene može prevesti koristeći C++ prevodilac i pokrenuti. Ipak postoji određen (relativno mali) broj specifičnosti programskog jezika C koje sa C++om neće raditi. Taj broj raste kako koristite novije standarde C-a i C++a. Recimo u novim verzijama C standarda dozvoljeni su nizovi čija je dužina data promjenljivom, koji nisu legalni u izvornom C89 standardu, niti u bilo kojoj verziji C++a. S druge strane, u novijim verzijama C++ standarda konstanta NULL je premještena u posebnu biblioteku i ne koristi se.

Prilikom pisanja ove knjige kao udžbenika za programski jezik C nastojali smo da ona bude korisna i za one koji uče C++. To znači da smo izbjegavali sve osobine C-a koje neće raditi u C++u, kao što su već spomenuti nizovi dužine date promjenljivom. Kod pojava koje bi mogle izazvati probleme (kao što je konstanta NULL) to smo istakli u tekstu.

Ipak, postoje i pored toga neki C programi koji će raditi ako ih kompajlirate sa C prevodiocem, a neće sa C++ prevodiocem, iako ne koriste ništa od onoga što smo naznačili u knjizi. To su programi koji su *u suštini neispravni*, bilo da se ne pridržavaju C standarda ili da se koristi neka osobina koja rezultira nedefinisanim ponašanjem. Zbog toga je jako bitno da se ne oslanjate na to što program radi da biste znali da je ispravan. Kompajler nije alat za provjeru ispravnosti programa! Program da bi bio ispravan treba se pridržavati standarda, a to možete postići tako što koristite samo one osobine jezika koje su navedene u ovoj knjizi.

Kada već učite programski jezik C++, važno je napomenuti da postoje neke stvari koje su sasvim legalne u tom programskom jeziku, ali se preporučuje da radite stvari na neki drugi način koji možda ne postoji u programskom jeziku C. Krenimo od uključivanja biblioteka: za ulaz i izlaz u C-u koristili smo:

```
#include <stdio.h>
```

i zaista ovakva deklaracija će raditi i u C++u. U pitanju je ista ona biblioteka koju koristimo u našim C programima. Ali zbog raznih problema sa C bibliotekama u programskom jeziku C++ definisane su "C++ verzije" istih tih biblioteka koje se neznatno razlikuju u imenu, te je u novijim verzijama C++ standarda korištenje izvornih C biblioteka proglašeno za "prevaziđeno" (eng. deprecated). U pravilu, naziv C++ verzije date biblioteke dobije se tako što se na početak doda slovo "c", a dio ".h" da se izostavi, ovako:

```
#include <cstdio>
```

Kod uključenja biblioteke na ovaj način, preporučljivo je (mada će program možda raditi i bez toga) da se navodi prefiks `std::` prije pristupa funkcijama:

```
std::printf("Zdravo svijete!");
```

Inače u mnogim udžbenicima i kursevima C++a vidjećete da se u primjerima kôda odmah na početku programa navodi sljedeća naredba:

```
using namespace std;
```

tako da se prilikom pristupa bibliotečnim funkcijama i objektima ne mora svaki put pisati `std::`. Ova praksa ipak *nije preporučljiva* između ostalog i zbog pojave "konflikta imenskih prostora", pa vam preporučujemo da to ne radite i da svaki put navodite `std::`.

Oznake

U ovoj knjizi korištene su sljedeće oznake:

Česta greška. Priloženi kod je pogrešan, a razlog zašto ga navodimo je što je to česta greška koju prave početnici. Savjetujemo da pročitate prateći tekst kako ne biste i vi pravili istu grešku.

Zadatak: *Kroz ove riješene primjere nećete dobiti samo pristup rješavanju sličnih problema. Biće korak-po-korak razrađena i razna alternativna rješenja, pa ćete na taj način naučiti mnoge bitne koncepte iz programiranja.*

Na mjestu označenom strelicom↑ pozivamo se na nešto što je objašnjeno u nekom drugom dijelu knjige.

↑ Ovdje će biti data oznaka poglavlja ili eventualno stranice.

✓ *Linija označena kvačicom predstavlja **definiciju** ili **pravilo**. Tu je definisan neki novi pojam ili navedeno bitno pravilo kojeg se trebate pridržavati. Sa strane je istaknut pojam koji definišemo.*

✓ Definicija ili pravilo