1. Uvod

Ovo uvodno poglavlje neophodno je za razumijevanje terminologije i notacije koja se koristi u ostatku teksta. Također, ono nam bolje objašnjava ciljeve i svrhu cijelog udžbenika. Sastoji se od triju odjeljaka. Prvi od njih definira osnovne pojmove, drugi podrobnije govori o strukturama podataka, a treći daje dodatne napomene o algoritmima.

1.1. Osnovni pojmovi

U ovom odjeljku najprije definiramo pojam strukture podataka, odnosno pojam algoritma. Zatim uvodimo još nekoliko srodnih pojmova: tip podataka, apstraktni tip, implementacija. Na kraju raspravljamo o tome kakve veze svi ti pojmovi imaju s našom željom da bolje programiramo.

1.1.1. Struktura podataka, algoritam

Svaki programer složit će se s tvrdnjom da pri razvoju računalnog programa moramo voditi brigu o dvije stvari: o strukturama podataka i o algortimima. Strukture podataka čine "statički" aspekt programa – to je ono s čime se radi. Algoritmi predstavljaju "dinamički" aspekt – to je ono što se radi.

Računalni program mogli bismo usporediti s kuharskim receptom: kao što recept na svom početku sadrži popis sastojaka (ulje, luk, brašno, meso…) tako i program mora započeti s definicijama podataka. Kao što recept mora opisivati postupak pripreme jela (sjeckanje, pirjanje, miješanje…), tako i program mora imati izvršne naredbe koje opisuju algoritam. Rezultat primjene kuharskog recepta je jelo dobiveno od polaznih sastojaka primjenom zadanog postupka. Rezultat

izvršavanja programa su izlazni podatci dobiveni transformacijom ulaznih podataka primjenom zadanog algoritma.

Kao što se dobro kuhanje odlikuje izborom kvalitetnih prehrambenih sastojaka uz ispravnu primjenu kulinarskih postupaka, tako se i dobro programiranje u podjednakoj mjeri sastoji od razvoja pogodnih struktura podataka kao i razvoja pogodnih algoritama. Programeri to često zaboravljaju, previše se koncentriraju na algoritme, makar primjena dobrih struktura podataka može jednako tako utjecati na kvalitetu cjelokupnog rješenja.

U nastavku slijede preciznije definicije dvaju ključnih pojmova koje susrećemo u programiranju.

Struktura podataka ... skupina varijabli u nekom programu zajedno s vezama između tih varijabli. Stvorena je s namjerom da omogući pohranjivanje određenih podataka te efikasno izvršavanje određenih operacija s tim podatcima (upisivanje, promjena, čitanje, traženje po nekom kriteriju...).

Algoritam ... konačan niz naredbi, od kojih svaka ima jasno značenje i izvršava se u konačnom vremenu. Izvršavanjem tih naredbi zadani ulazni podatci pretvaraju se u izlazne podatke (rezultate). Pojedine naredbe mogu se izvršavati uvjetno, ali u tom slučaju same naredbe moraju opisati uvjet izvršavanja. Također, iste naredbe mogu se izvršiti više puta, pod pretpostavkom da same naredbe ukazuju na ponavljanje. Ipak, zahtijevamo da za bilo koje vrijednosti ulaznih podataka algoritam završava nakon konačnog broja ponavljanja.

Strukture podataka i algoritmi nalaze se u nerazlučivom odnosu: nemoguće je govoriti o jednom a da se ne spomene drugo. U ovom udžbeniku proučavat ćemo baš taj odnos: promatrat ćemo kako odabrana struktura podataka utječe na algoritme za rad s njom, te kako odabrani algoritam sugerira pogodnu strukturu za prikaz svojih podataka. Na taj način upoznat ćemo se s nizom važnih ideja koje čine osnove dobrog programiranja, a također i osnove računarstva u cjelini.

1.1.2. Tip podataka, apstraktni tip, implementacija

Uz "strukture podataka" i "algoritme", ovaj udžbenik koristi još nekoliko naizgled sličnih pojmova. Oni su nam potrebni da bismo izrazili međusobnu ovisnost između podataka i operacija koje se nad njima trebaju obavljati, odnosno ovisnost između operacija i algoritama koji ih realiziraju. Slijede definicije tih dodatnih pojmova.

Tip podataka ... skup vrijednosti koje neki podatak može poprimiti. Primjerice, podatak tipa `int` u nekom C programu može imati samo vrijednosti iz skupa cijelih brojeva prikazivih u računalu.

Apstraktni tip podataka (a.t.p.) ... zadaje se navođenjem jednog ili više tipova podataka te jedne ili više operacija (funkcija). Operandi i rezultati navedenih operacija su podatci navedenih tipova. Među tipovima postoji jedan istaknuti po kojem cijeli apstraktni tip podataka dobiva ime.

Implementacija apstraktnog tipa podataka ... konkretna realizacija dotičnog apstraktnog tipa podataka u nekom programu. Sastoji se od definicije za strukturu podataka (kojom se prikazuju podatci iz apstraktnog tipa podataka) te od potprograma (kojima se operacije iz apstraktnog tipa podataka ostvaruju s pomoću odabranih algoritama). Za isti apstraktni tip podataka obično se može smisliti više različitih implementacija – one se razlikuju po tome što koriste različite strukture za prikaz podataka te različite algoritme za izvršavanje operacija.

Kao konkretni (no vrlo jednostavni) primjer za apstraktni tip podataka u nastavku definiramo apstraktni tip koji odgovara matematičkom pojmu kompleksnih brojeva i njihovom specifičnom načinu zbrajanja i množenja.

Apstraktni tip podataka *Complex*

`scalar` ... bilo koji tip za koji su definirane operacije zbrajanja i množenja.

`Complex` ... podatci ovog tipa su uređeni parovi podataka tipa `scalar`.

`CoAdd(z1,z2,&z3)` ... za zadane `z1,z2` tipa `Complex` računa se njihov zbroj `z3`, također tipa `Complex`. Dakle za `z1` oblika `(x1,y1)`, `z2` oblika `(x2,y2)`, dobiva se `z3` oblika `(x3,y3)`, takav da je `x3=x1+x2`, `y3=y1+y2`.

`CoMult(z1,z2,&z3)` ... za zadane `z1,z2` tipa `Complex` računa se njihov umnožak `z3`, također tipa `Complex`. Dakle za `z1` oblika `(x1,y1)`, `z2` oblika `(x2,y2)`, dobiva se `z3` oblika `(x3,y3)`, takav da je `x3=x1*x2-y1*y2`, `y3=x1*y2+y1*x2`.

Vidimo da se apstraktni tip zadaje kao popis potrebnih tipova podataka te osnovnih operacija koje mislimo obavljati nad tim podatcima. Argumenti i rezultati navedenih operacija moraju biti podatci navedenih tipova. U konkretnoj implementaciji svaki tip trebao bi se opisati kao tip u C-u s istim nazivom. Također, svaka

operacija trebala bi se realizirati kao funkcija u C-u s istim nazivom te istovrsnim argumentima i povratnom vrijednošću. Primijetimo da je u gornjem popisu za svaku operaciju naveden način njezinog pozivanja, a ne prototip za njezino definiranje. Ako operacija mijenja neku varijablu, tada se ta varijabla u pozivu prenosi preko adrese, što je označeno znakom `&`.

Primijetimo da je za apstraktni tip podataka *Complex* važno prisustvo operacija `CoAdd( )` i `CoMult( )`. Bez tih operacija radilo bi se o običnom tipu, i tada kompleksne brojeve ne bismo mogli razlikovati od uređenih parova skalara. Dodatna dimenzija "apstraktnosti" sastoji se u tome što nismo do kraja odredili što je tip `scalar`. Ako za `scalar` odaberemo `float`, tada imamo posla sa standardnim kompleksnim brojevima. Ako `scalar` poistovjetimo s `int`, tada koristimo posebne kompleksne brojeve čiji su realni i imaginarni dijelovi cijeli. Sa stanovišta struktura podataka i algoritama izbor tipa `scalar` zapravo je nebitan. Važan nam je odnos među varijablama a ne njihov sadržaj.

Struktura podataka pogodna za prikaz kompleksnog broja mogla bi se definirati sljedećim složenim tipom.

```
typedef struct {
   scalar re;
   scalar im;
} Complex;
```

Implementacija apstraktnog tipa podataka *Complex* mogla bi se sastojati od naredbe kojom se tip `scalar` poistovjećuje s `int` ili `float`, prethodne definicije tipa `Complex` te od sljedećih funkcija:

```
void CoAdd (Complex z1, Complex z2, Complex *z3p) {
   (*z3p).re = z1.re + z2.re;
   (*z3p).im = z1.im + z2.im;
   return;
}

void CoMult (Complex z1, Complex z2, Complex *z3p){
   (*z3p).re = z1.re * z2.re - z1.im * z2.im;
   (*z3p).im = z1.re * z2.im + z1.im * z2.re;
   return;
}
```

Opisana implementacija apstraktnog tipa *Complex* nije naravno jedina moguća implementacija. Donekle drukčije rješenje dobili bismo kad bismo kompleksni broj umjesto kao `struct` prikazali kao polje skalara duljine 2. Makar je to prilično nebitna razlika, ona bi zahtijevala da se funkcije `CoAdd` i `CoMult` zapišu na drukčiji način.

Daljnje (mnogo živopisnije) primjere apstraktnih tipova podataka i njihovih implementacija susrest ćemo u idućim poglavljima. Naime, svaki od odjeljaka 2.1 – 4.4 obrađuje bar jedan apstraktni tip podataka. Opisuju se svojstva tog apstraktnog tipa, nabrajaju njegove primjene, promatraju razne njegove implementacije te uočavaju prednosti i mane tih implementacija. Dakle, čitanjem idućih poglavlja upoznat ćemo mnogo apstraktnih tipova te još više struktura podataka i algoritama.

U složenijim problemima može se pojaviti potreba da radimo s više apstraktnih tipova u isto vrijeme. Pritom bi oni mogli imati slične operacije. Da ne bi došlo do nedoumice kojem apstraktnom tipu pripada koja operacija, uvest ćemo sljedeću konvenciju: imena svih operacija iz istog apstraktnog tipa moraju početi s dvoslovčanim prefiksom koji jednoznačno određuje taj apstraktni tip. Ovu konvenciju već smo primijenili kod apstraktnog tipa *Complex* gdje su imena za obje funkcije imale prefiks `Co`.

1.1.3. Programiranje postupnim profinjavanjem

Znanje iz ovog udžbenika važno je zato jer nam omogućuje da bolje programiramo. Naime, apstraktni tipovi, strukture podataka i algoritmi predstavljaju vrlo koristan skup alata koji nam pomaže kod rješavanja složenijih programerskih problema.

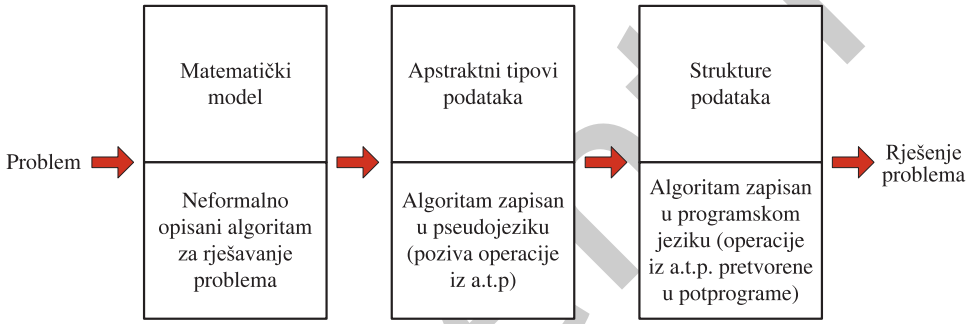
Zaista, kad god se suočimo sa složenijim problemom, tada do rješenja obično ne možemo doći na izravan način. To jest, nije moguće sjesti za računalo i odmah napisati program u C-u. Umjesto toga služimo se metodom postepenog profinjavanja, gdje se do rješenja dolazi postepeno u nizu koraka. Početni koraci definiraju rješenje na neformalan način služeći se nekim općenitim matematičkim modelom. Daljnji koraci razrađuju to rješenje dodajući mu konkretne detalje, sve dok u zadnjem koraku ne dođemo do opisa na razini detaljnosti kakva se traži u programskom jeziku.

Slika 1.1 prikazuje jednu varijantu metode postepenog profinjavanja. Riječ je o varijanti koja se oslanja na apstraktne tipove podataka, a u njoj se postupak rješavanja problema dijeli na tri koraka.

- U prvom koraku sastavlja se matematički model problema, a rješenje se opisuje neformalnim algoritmom koji djeluje u okvirima tog modela.

- U drugom koraku uočavaju se osnovne operacije nad matematičkim objektima, pa se ti objekti zajedno s operacijama pretvaraju u apstraktne tipove. Algoritam se zapisuje u donekle strukturiranom jeziku (pseudojeziku) kao niz poziva operacija iz apstraktnih tipova.

- U trećem koraku odabiremo implementaciju za svaki od korištenih apstraktnih tipova. Na taj način dolazimo do struktura podataka s jedne strane te do funkcija koje implementiraju operacije iz apstraktnih tipova s druge strane. Algoritam se zapisuje u programskom jeziku kao "glavni" program koji poziva spomenute funkcije kao potprograme.



Slika 1.1. Postupak rješavanja problema metodom postepenog profinjavanja

Iz slike 1.1 jasno se vidi uloga uloga apstraktnih tipova, struktura podataka i algoritama u postupku rješavanja problema. Također, vidi se da se programiranje u podjednakoj mjeri sastoji od razvijanja struktura podataka kao i od razvijanja algoritama.

Da bismo ilustrirali našu varijantu metode postepenog profinjavanja, poslužit ćemo se sljedećim primjerom problema. Zamislimo da na fakultetu u istom danu treba održati veći broj ispita. Neka svaki od njih traje jedan sat. Dvorana i nastavnika ima dovoljno. No poteškoća je u tome što su neki ispiti u koliziji: dakle za njih su prijavljeni dijelom isti studenti, pa se oni ne mogu održati istovremeno. Treba pronaći raspored održavanja ispita takav da ispiti u koliziji ne budu u isto vrijeme, a da ukupno vrijeme održavanja bude što kraće.

U nastavku pratimo rješavanje problema rasporeda ispita metodom postepenog profinjavanja.

- U prvom koraku sam problem opisujemo matematičkim modelom obojenog grafa. Riječ je o neusmjerenom grafu gdje svaki vrh odgovara jednom ispitu. Dva vrha su susjedna (povezani su bridom) ako i samo ako su dotični

ispiti u koliziji. Vrhovi su obojeni bojama koje označavaju termine (sate) održavanja ispita. Na taj način iz boja vrhova čita se raspored. Da bi taj raspored bio dopustiv, susjedni vrhovi (dakle oni povezani bridom) moraju biti u različitim bojama. Da bi ukupno vrijeme održavanja ispita bilo što kraće, broj upotrebljenih boja mora biti što manji.

- Kao algoritam za rješavanje problema možemo koristiti sljedeći algoritam bojenja grafa koji nastoji potrošiti što manje boja (makar ne mora nužno postići najmanji mogući broj). Algoritam radi u iteracijama. U jednoj iteraciji bira se jedan neobojeni vrh i gledaju se boje njegovih susjeda; zatim se izabrani vrh oboji jednom od već upotrebljenih boja koju ne koristi ni jedan od susjeda; ako nema takve boje, tada se izabrani vrh oboji novom bojom, čime smo broj upotrebljenih boja povećali za jedan. Iteracije se ponavljaju dok god ima neobojenih vrhova.

- U drugom koraku naš matematički model, dakle obojeni graf, pretvaramo u apstraktni tip. Taj apstraktni tip treba sadržavati tip podataka za prikaz samog grafa te osnovne operacije za rukovanje grafom. Očito će nam trebati operacije: ubacivanja novog vrha, spajanja dvaju postojećih vrhova bridom, pridruživanja zadane boje zadanom vrhu, pronalaženja susjeda zadanog vrha, itd.

- Prije opisani algoritam bojenja grafa sada je potrebno zapisati u terminima osnovnih operacija nad apstraktnim tipom. Primjerice, jedna iteracija gdje se jednom neobojenom vrhu određuje boja koristit će operaciju pronalaženja susjeda vrha, niz operacija čitanja boje vrha te operaciju pridruživanja boje vrhu.

- U trećem koraku razvijamo implementaciju za naš apstraktni tip obojenog grafa. Na primjer, graf se može prikazati strukturom koja se sastoji od vektora boja i matrice susjedstva. Pritom i-ta komponenta vektora sadrži boju i-tog vrha, a (i, j)-ti element matrice sadrži 1 odnosno 0 ovisno o tome jesu li i-ti i j-ti vrh susjedi ili nisu.

- Sastavni dio implementacije apstraktnog tipa su funkcije koje realiziraju operacije iz tog apstraktnog tipa. Primjerice, pridruživanje boje vrhu realizira se funkcijom koja mijenja odgovarajuću komponentu vektora boja, a pronalaženje susjeda vrha realizira se funkcijom koja čita odgovarajući redak matrice susjedstva i pronalazi jedinice u njemu.

- Konačni program sastoji se od strukture za prikaz obojenog grafa, funkcija koje obavljaju osnovne operacije nad grafom te glavnog programa koji poziva funkcije i povezuje ih u algoritam bojenja grafa.

Programiranje postepenim profinjavanjem uz korištenje apstraktnih tipova donosi brojne prednosti u odnosu na "ad-hoc" programiranje. Evo nekih od tih prednosti.

- Lakše ćemo doći do rješenja, a ono će biti bolje strukturirano, razumljivije i pogodno za daljnje održavanje.
- Iskoristit ćemo već postojeća znanja i iskustva umjesto da "otkrivamo toplu vodu". Primjerice, ako naš problem zahtijeva korištenje grafa, tada nećemo morati izmišljati prikaz grafa u računalu već ćemo ga naći u literaturi.
- Možda ćemo ponovo upotrijebiti već gotove dijelove softvera umjesto da ih sami razvijamo. Na primjer, ako je neki drugi programer već razvio implementaciju grafa u C-u, tada uz njegovo dopuštenje tu implementaciju možemo odmah ugraditi u naš program.
- Lakše ćemo se sporazumijevati s drugima. Ako kolegi kažemo da imamo program koji radi s obojenim grafom, kolega će nas bolje razumjeti nego kad bismo mu rješenje opisivali nekim svojim riječima.

1.2. Više o strukturama podataka

U ovom odjeljku govorimo detaljnije o strukturama podataka: od čega se one sastoje, kako se one grade. Objašnjavamo da se struktura podataka sastoji od dijelova koji se udružuju u veće cjeline i međusobno povezuju vezama. Uvodimo posebne nazive za dijelove, načine udruživanja te načine povezivanja. Također, uvodimo pravila kako se strukture prikazuju dijagramima.

1.2.1. Dijelovi od kojih se grade strukture podataka

Rekli smo već da se struktura podataka sastoji od varijabli u nekom programu i veza među tim varijablama. To znači da su dijelovi strukture same te varijable, ili točnije mjesta u memoriji računala gdje se mogu pohraniti podatci. Ipak, varijable mogu biti različitih tipova. Zato razlikujemo sljedeće vrste dijelova.

Klijetka (ćelija) ... varijabla koju promatramo kao nedjeljivu cjelinu. Klijetka je relativan pojam jer se jedna cjelina u jednom trenutku može smatrati nedjeljivom, a kasnije se može gledati unutrašnja građa te iste cjeline. Svaka klijetka ima svoj tip, adresu (ime) i sadržaj (vrijednost). Tip i adresa su nepromjenjivi, a sadržaj se može mijenjati.

Polje (`array` u C-u) ... mehanizam je udruživanja manjih dijelova u veće. Polje čini više klijetki istog tipa pohranjenih na uzastopnim adresama. Broj kli-

jetki unaprijed je zadan i nepromjenljiv. Jedna klijetka unutar polja zove se element polja i jednoznačno je određena svojim rednim brojem ili indeksom. Ugledajući se na C, uzimamo da su indeksi `0, 1, 2, ..., N-1`, gdje je `N` cjelobrojna konstanta.

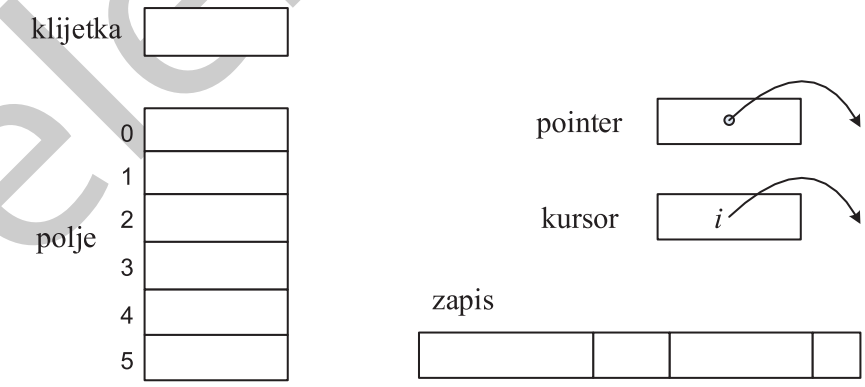
Zapis (slog, `struct` u C-u) ... također je mehanizam udruživanja manjih dijelova u veće. Zapis čini više klijetki, koje ne moraju biti istog tipa, no koje su pohranjene na uzastopnim adresama. Broj, redoslijed i tip klijetki unaprijed je zadan i nepromjenljiv. Pojedina klijetka unutar zapisa zove se komponenta zapisa i jednoznačno je određena svojim imenom.

Pointer ... služi za uspostavljanje veze između dijelova strukture. Pointer je klijetka koja pokazuje gdje se nalazi neka druga klijetka. Sadržaj pointera je adresa klijetke koju treba pokazati.

Kursor ... također služi za uspostavljanje veze između dijelova strukture. Kursor je klijetka tipa `int` koja pokazuje na element nekog polja. Sadržaj kursora je indeks elementa koji treba pokazati.

Pointere i kursore jednim imenom nazivamo *pokazivači*. No ipak, u nastavku ćemo se služiti njihovim posebnim nazivima da bismo naglašavali razliku među njima.

Strukture podataka precizno se definiraju odgovarajućim naredbama u C-u. No isto tako mogu se prikazati i s pomoću dijagrama, što je manje precizno ali je zornije. Svaki dio strukture crta se na određeni način tako da se lakše može prepoznati. Pravila crtanja vidljiva su na slici 1.2.

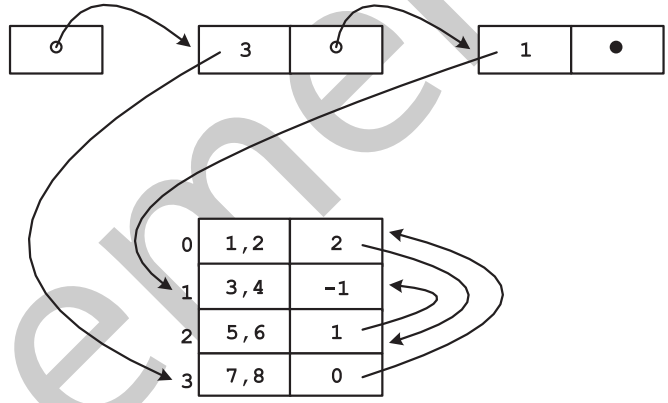


Slika 1.2. Dijelovi strukture podataka, pravila crtanja

Dakle u skladu sa slikom 1.2 klijetka se crta kao kućica, njezina adresa ili ime pišu se pored kućice, a sadržaj unutar kućice. Polje se u pravilu crta kao uspravan niz kućica s indeksima elemenata ispisanim uz lijevi rub. Zapis se crta kao vodoravan niz kućica nejednakih veličina s imenima komponenti ispisanim na gornjem ili donjem rubu. I pointer i kursor crtaju se kao kućica iz koje izlazi strelica. Izgled početka strelice malo je drukčiji, naime adresa unutar pointera označena je kružićem, a cijeli broj unutar kursora može biti eksplicitno prikazan.

1.2.2. Povezivanje dijelova strukture u cjelinu

Strukture se grade grupiranjem dijelova u polja ili zapise te povezivanjem dijelova s pomoću pointera ili kursora. Polja i zapisi mogu se kombinirati. Na primjer, možemo imati polje zapisa, zapis čije pojedine komponente su polja, polje od polja, zapis čija komponenta je zapis, i slično. Komponente zapisa ili elementi polja mogu biti pointeri ili kursori koji pokazuju na neke druge zapise ili polja ili čak neke druge pointere ili kursore. Sve ove konstrukcije mogu se po volji iterirati.



Slika 1.3. Primjer složenije strukture podataka

Slika 1.3 sadrži dijagram strukture podataka koja se sastoji od niza zapisa povezanih pointerima te jednog polja zapisa. Zapisi u polju još su međusobno povezani kursorima. Vidimo da se strelica pointera odnosno kursora crta tako da njezin kraj (šiljak) dodiruje klijetku koju treba pokazati. Također vidimo da se nul-pointer (onaj koji nikamo ne pokazuje) označuje kao puni kružić, a nul-kursor kao -1 (nepostojeći indeks).

Struktura na slici 1.3 zanimljivo izgleda no ima jednu manjkavost: nije jasno čemu služi. Molimo čitatelje da razmisle o mogućim interpretacijama te da ih jave autoru udžbenika.