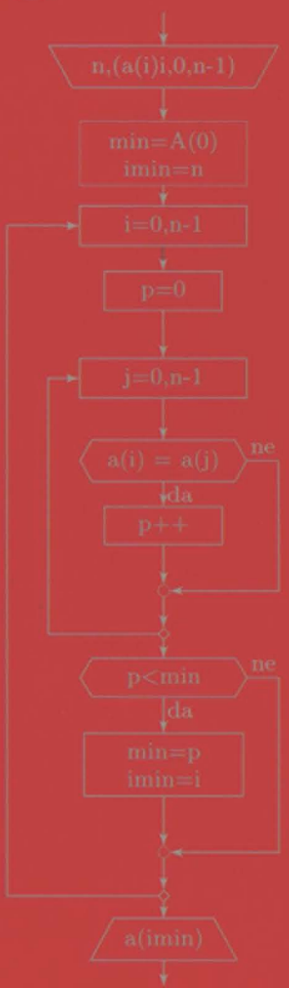
Univerzitet u Nišu
Elektronski fakultet

dr Vladimir Ćirić

Uvod u programiranje i programski jezik C

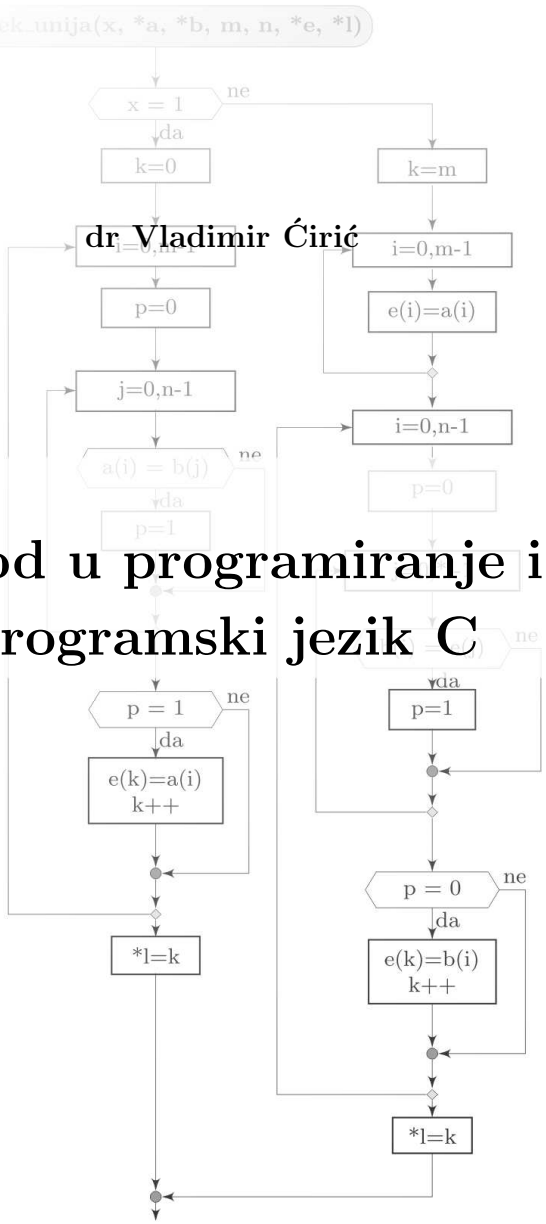
```
1  #include <stdio.h>
2  void main()
3  {
4          int n,i,p,j,min,imin,a[30];
5          scanf ("%d",&n);
6          for ( i = 0; i < n; i++ )
7                  scanf ("%d",&a[i]);
8          min = A[0];
9          imin = n;
10         for ( i = 0; i < n; i++ )
11         {
12                 p=0;
13                 for ( j = 0; j < n; j++ )
14                         if ( a[i] == a[j] )
15                                 p++;
16                 if ( p < min )
17                 {
18                         min = p;
19                         imin = i;
20                 }
21         }
22         printf ("%d", a[imin]);
23 }
```



Edicija: Osnovni udžbenici

Niš 2014.

Univerzitet u Nišu
Elektronski fakultet



dr Vladimir Ćirić

Uvod u programiranje i programski jezik C

Niš, 2014.

UVOD U PROGRAMIRANJE I PROGRAMSKI JEZIK C

Autor Doc. dr Vladimir Ćirić,

Izdavač Elektronski fakultet u Nišu
P.fah 73, 18000 Niš
http://www.elfak.ni.ac.rs/

Recenzenti Prof. dr Dragan Janković, Elektronski fakultet u Nišu
Prof. dr Emina Milovanović, Elektronski fakultet u Nišu
Prof. dr Jelica Protić, Elektrotehnički fakultet u Beogradu

Glavni i odgovorni urednik: Prof. dr Dragan Tasić

Odlukom Nastavno-naučnog veća Elektronskog fakulteta u Nišu, br. 07/05-024/14-006 od 26.12.2014. godine, rukopis je odobren za štampu kao udžbenik na Elektronskom fakultetu u Nišu.

ISBN 978-86-6125-119-1

CIP - Каталогизација у публикацији - Народна
библиотека Србије, Београд

004.42(075.8)
004.432.2C(075.8)

ЋИРИЋ, Владимир, 1977-
Uvod u programiranje i programski jezik C /
Vladimir Ćirić. - Niš :
Elektronski fakultet, 2014 (Niš : Unigraf X-
Copy). - VI, 350 str. : graf.
prikazi ; 24 cm

Na vrhu nasl. str.: Univerzitet u Nišu. - Tiraž
500. - Bibliografija: str. 347. - Registar.

ISBN 978-86-6125-119-1

a) Програмирање b) Програмски језик "C"
COBISS.SR-ID 212405516

Tiraž: 500 primeraka
Štampa: Unigraf X-Copy, Niš

Predgovor

Knjiga dr Vladimira Ćirića, Uvod u programiranje i programski jezik C, vredna je udžbenička literatura koja će biti od koristi, u prvom redu, studentima kojima je osnovna linija obrazovanja vezana za neki od računarskih pravaca, ali i onim studentima koji studiraju različite tehničke discipline, a neophodna su im programerska znanja.

Autor je rad na knjizi zasnovao na svom bogatom, dugogodišnjem programerskom i pedagoškom iskustvu. On pažljivo uvodi čitaoca u svet programiranja kroz poglavlja ove knjige. Početno poglavlje posvećeno je rešavanju problema uz pomoć računara i sadrži pažljivo selektovane činjenice iz istorije računarstva, kao i klasifikaciju programskih jezika. Potom se fokus prebacuje na algoritme, uz demonstraciju načina predstavljanja, opis dijagrama toka, kao i osnovnih i složenih algoritamskih struktura. Treće poglavlje započinje formalnim opisom sintakse programskih jezika, čime je urađena dobra priprema za prezentaciju osnovnih elemenata programskog jezka C, što je zapravo centralni predmet razmatranja u ovom delu. U posebnom poglavlju predstavljene su promenljive, tipovi podataka i operatori. Kako bi student, koji počinje sa programiranjem, imao jasniju sliku o tome sa kakvim podacima se radi i kako podaci mogu biti organizovani, autor je peto poglavlje posvetio osnovnim strukturama podataka. Poslednje, šesto poglavlje veoma sveobuhvatno razmatra funkcije i pomaže studentu da ovlada ovim ključnim sredstvom u programiranju.

Materijal sadrži veliki broj preciznih i dobro postavljenih defincija, objašnjenja su jezgrovita i orijentisana ka studentu, a posebnu vrednost predstavljaju pažljivo kreirani originalni primeri koji ilustruju svaku od tema. Kontrolna pitanja na kraju svakog poglavlja nemaju samo funkciju da provere znanje, već vrlo jasno ukazuju šta je to što student treba da prepozna i kao značajno usvoji iz prethodno prezentovanog materijala. Nadam se da će generacije studenata koje dolaze sa ovom knjigom dobiti dobar uvod u programiranje i da će sigurnim koracima savladati programski jezik C.

U Nišu,
27.10.2014. godine

Prof. dr Ivan Milentijević

2π

Sadržaj

1

Rešavanje problema uz pomoć računara

Prva upotreba reči "računar", odnosno "kompjuter" (eng. *computer*), zabeležena je 1613. godine u kontekstu osobe koja vrši izračunavanja. Ovo značenje je zadržano do sredine 20. veka, mada je već u drugoj polovini 19. veka reč počela da dobija današnje značenje.

Moderno računarstvo razvijalo se kroz konstantno usavršavanje računarskih arhitektura i tehnika programiranja. Počev od relativno jednostavnih uređaja pomoću kojih je bilo moguće izvršiti jednostavne računske operacije, uređaji su evoluirali, a prava ekspanzija različitih programabilnih elektronskih kola desila se sa pojavom integrisanih kola.

Iako je veoma teško identifikovati neki uređaj kao "prvi računar", vredi pomenuti neke od uređaja koji su uticali na razvoj računarstva.

1.1 Mašinska obrada podataka

Još od najranijih vremena ljudi su se trudili da konstruišu sprave koje bi im pomagale u izvođenju računskih operacija. Jedna od rasprostranjenijih i češće sretanih sprava kroz istoriju bio je abakus (računaljka), konstruisan pre oko 5.000 godina. Razvojem novih tehnologija usavršavale su se i računarske sprave. Poznate su mehaničke mašine za sabiranje koje su konstruisali Leonardo da Vinči (1425-1519) i Blez Paskal (1623-1662), kao i Lajbnicova mašina iz 1671. (Gottfried Wilhelm von Leibniz), koja je pored sabiranja i oduzimanja obavljala i operacije množenja i deljenja. Tek prvih decenija ovog veka pojavljuju se prve električne računarske mašine sastavljene od elektromagnetnih releja. Ove mašine su pravile veliku buku pri radu, jer je svaka tranzicija logičkih nivoa bila propraćena mehaničkim pomeranjem kontakata uz pomoć elektromagneta. Amerikanci su tokom Drugog svetskog rata razvili prvi potpuno elektronski računar ENIAC (*Electronic Numerical Integrator and Computer*), koji je radio na principu elektronskih cevi i bio prevashodno

Kontrolna pitanja

1. Ko se smatra prvim programerom i na osnovu rada na kom uređaju?
2. Koji koncept nedostaje prvobitnim uređajima za računanje da bi se mogli smatrati računarima u današnjem smislu te reči?
3. Od čega se sastoji Tjuringova mašina?
4. Modifikovati prelaske stanja Tjuringove mašine iz primera 1.1, tako da se glava za čitanje simbola na kraju programa ne vraća na početak niza simbola.
5. Opisati osnovne elemente Von Nojmanove arhitekture. Na koji način se izvršavaju instrukcije na ovoj arhitekturi?
6. Koje kategorije mašinskih instrukcija postoje?
7. Koje su prednosti, a koje mane programiranja na asemblerskim jezicima?
8. Objasniti proces generisanja izvršnog koda iz programa napisanog na asemblerskom jeziku. Koji su dodatni koraci potrebni kod makro-asemblerskih jezika i zašto?
9. U čemu se ogleda razlika u izvršavanja programa napisanih na interpreterskim jezicima i kompajlerskim jezicima?
10. Koje su faze u rešavanju problema uz pomoć računara?
11. Odrediti vrednosti $\sqrt[2]{5}$, $\sqrt[2]{7}$ i $\sqrt[3]{7}$ korišćenjem iterativnog postupka za izračunavanje vrednosti funkcije $x = \sqrt[n]{a}$:

$$\begin{aligned} x_0 &= \frac{(a+n-1)}{n} \\ x_{i+1} &= \frac{\left((n-1)\cdot x_i + \frac{a}{x_i^{n-1}}\right)}{n}, \quad i = 0, 1, 2, \ldots. \end{aligned}$$

   Kroz korake postupka objasniti primenu ovih postupaka kod ručnih kalkulatora.

12. Šta je sintaksa, a šta semantika programa?

2

Algoritmi

Algoritam je precizno definisani postupak, takav da praćenje koraka od početka pa do kraja algoritma vodi rešenju nekog konkretnog problema. Koraci algoritma su elementarne celine koje se precizno i eksplicitno navode, bez podrazumevanih delova.

Algoritmi se mogu predstaviti na različite načine. Načini za predstavljanje algoritama uključuju:

1. tekstualni opis na prirodnom jeziku,
2. dijagrame toka,
3. pseudokod,
4. strukturograme, i
5. programske jezike.

Algoritmi predstavljeni prirodnim jezikom koriste izraze koji imaju tendenciju da budu razumljiviji i nedvosmisleni. Ovakvo predstavljanje se retko koristi za složene i tehnički zahtevne algoritme. Dijagrami toka, pseudokod i strukturogrami su strukturirani načini predstavljanja algoritama kojima se izbegavaju mnoge nejasnoće i dvosmislenosti tipične za prirodne jezike. Programski jezici, s druge strane, su prvenstveno namenjeni za predstavljanje algoritama u obliku koji se može direktno izvršiti na računaru.

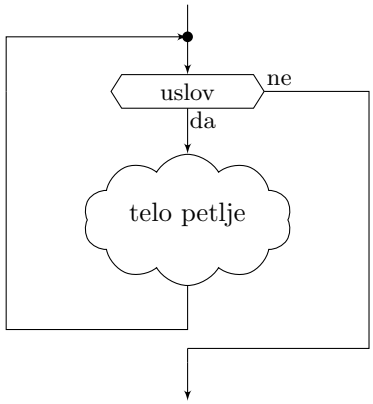
Pored navedenih postoje i drugi načini za predstavljanje algoritama. Jedan od načina koji nije među gore navedenim je predstavljanje algoritma pomoću Tjuringove mašine. Reprezentacija algoritma pomoću Tjuringove mašine uključuje skup simbola, stanja i definicije prelaza iz stanja u stanje, na način ilustrovan na primeru 1.1.

Po detaljnosti algoritma, moguće je izvršiti podelu na tri nivoa:

1. **Opisi na visokom nivou** – su opisi algoritama koji ne zalaze u detalje i u potpunosti ignorišu implementaciju. Na primeru Tjuringove mašine na ovom

Petlja tipa *while-do*

Petlja tipa *while-do*, u prevodu sa engleskog *"sve dok je - radi"*, je osnovna algoritamska struktura kojom se implementira uslovna petlja, kod koje se telo petlje višestruko izvršava, kako joj i ime kaže, sve dok je uslov za izvršavanje tela petlje ispunjen. Ova petlja se kraće naziva i *while* petlja. Dijagram toka ove strukture prikazan je na slici 2.17.



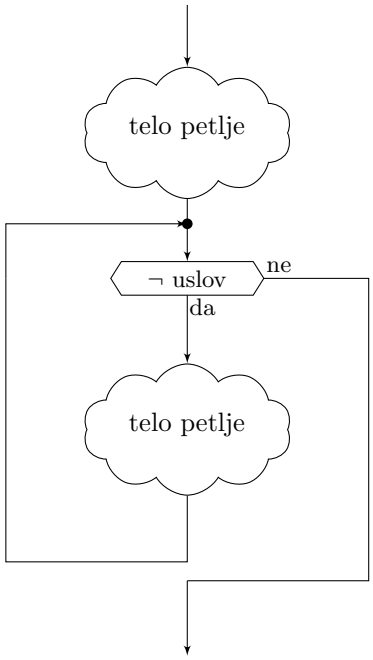
Slika 2.17: Petlja tipa *while*

Kada se u toku izvršenja algoritma naiđe na *while* petlju, proverava se uslov za izvršenje tela petlje. Svi parametri koji figurišu u uslovu moraju biti poznati u tom trenutku, inače je algoritam neodređen i neprecizan. Ukoliko je uslov ispunjen izvršava se telo petlje. Nakon izvršenja tela petlje izvršenje se vraća se na trenutak neposredno pre provere uslova (slika 2.17).

Napomena: Povratna grana iz tela petlje nazad na uslov je uvek prazna (ne sadrži nijedan blok) i crta se sa leve strane tela petlje, kako je prikazano na slici 2.17. Praćenjem ove povratne grane izvršenje se vraća neposredno trenutak pre ispitivanja uslova (slika 2.17). Između tog trenutka (crna tačka iznad uslova na slici 2.17) i samog uslova takođe ne sme biti drugih blokova.

U telu petlje se mogu menjati parametri uslova. Kada se iteracija petlje završi uslov petlje se proverava ponovo. Ako uslov nije ispunjen petlja se završava, čime se prelazi na izvršenje prvog narednog bloka iza petlje.

Primer 2.10 (Algoritam regulatora za grejanje vode)**. Zadatak**: Nacrtati strukturni dijagram toka algoritma po kome radi regulator za grejanje vode. Regulator ima potenciometar za zadavanje željene temperature (T) i senzor za merenje trenutne temperature vode (S). Grejač vode treba da bude uključen sve dok se ne postigne željena temperatura vode. Kada je željena temperatura vode postignuta grejač treba isključiti.



Slika 2.26: Implementacija petlja tipa *repeat-until* pomoću petlje tipa *while*

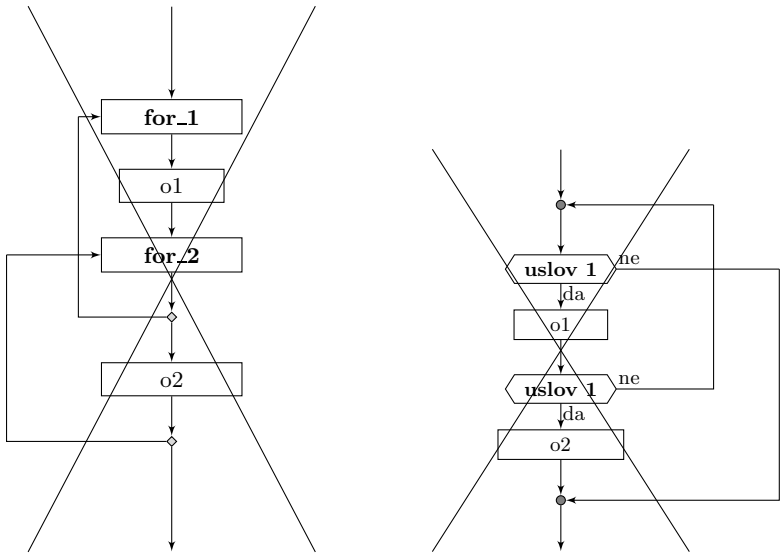
Potrebno je napomenuti da je uslov na dijagramu sa slike 2.26 negiran, da bi zadovoljio sled događaja po kome se kod *while* petlje telo izvršava dok je uslov ispunjen, a ne dok nije, kao kod *repeat-until* petlje.

Ova osobina da je algoritamsku strukturu koja prirodno nameće upotrebu *do-while* petlje moguće implementirati i *while-do* petljom ima za posledicu da neki programski jezici nemaju osnovnu strukturu koja implementira ovu petlju. Bez obzira na ovaj nedostatak u tim programskim jezicima, upotrebom *while-do* petlje moguće je implementirati bilo koji algoritam.

For petlja

Petlja tipa *for* je osnovna algoritamska struktura kod koje se telo petlje izvršava više puta, a broj iteracija jeste poznat neposredno pre početka izvršavanja petlje. Broj iteracija petlje može biti zadat konstantom (npr. 10 ponavljanja), ili može biti zadat parametarski (npr. N ponavljanja), s tim da vrednost parametra u slučaju parametarskog zadavanja mora biti poznata pre početka petlje.

Pored zadatog broja iteracija, *for* petlja ima dodatni brojač čija je vrednost jednaka rednom broju trenutne iteracije. Vrednost ovog brojača se pamti u pro-

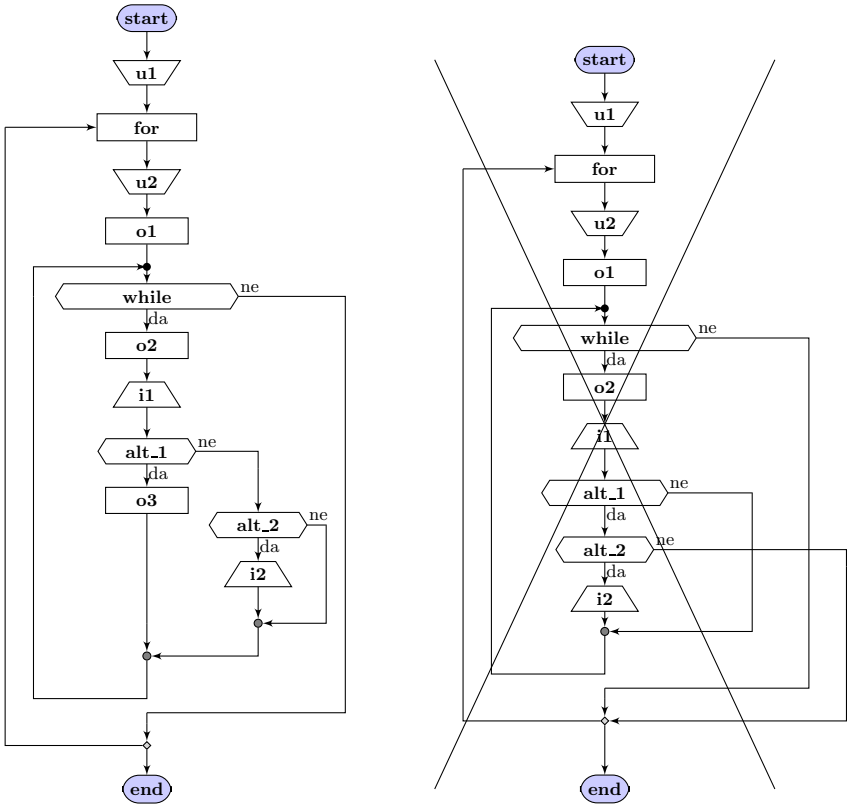


Slika 2.38: Primer nestrukturnih dijagrama toka: a) preklapajuće petlje, b) nestrukturni uslovni skokovi

na koju treba preći ako je uslov ispunjen. Ove naredbe unose nestrukturnost u tok izvršenja i veoma su česte kod mašinskih i asemblerskih jezika. Mašinski i asemblerski jezici nemaju strukture alternacije ni petlje, pa se sve algoritamske strukture implementiraju naredbama uslovnog skoka. Dijagrami toka izvršenja programa na asemblerskom jeziku imaju elemente dijagrama prikazanog na slici 2.38b, često su nestrukturni, i zbog toga ih je vrlo teško pratiti i odrediti semantiku tako napisanog programa. Otklanjanje semantičkih grešaka u kompleksnom nestrukturnom algoritmu je takođe vremenski veoma zahtevan proces.

Na slici 2.39 uporedo su prikazani primeri pravilnog i nepravilnog kombinovanja osnovnih algoritamskih struktura. Na slici 2.39a prikazana je kombinacija dve petlje, *for* i *while*, i dva grananja, označena sa *alt_1* i *alt_2*. Pored ove 4 strukture dijagram toka sadrži i dva bloka za unos (u1 i u2), dva bloka za prikaz (i1 i i2), i tri bloka za obradu (o1, o2 i o3). Posmatrajući strukture po dubini, počev od spoljašnjih struktura ka unutrašnjim, na slici 2.39a razlikujemo:

1. sekvencu ulaza **u1** i
2. **for** petlju (petlja se posmatra kao jedinstvena celina u sekvenci u kojoj se nalazi);



Slika 2.39: Primer složenog strukturnog i nestrukturnog dijagrama toka

2.1. u *for* petlji imamo sekvencu **u2**,

2.2. **o2** i

2.3. **while** petlju (petlja se posmatra kao celina u sekvenci);

2.3.1. u *while* petlji imamo sekvencu **o2**,

2.3.2. **i1** i

2.3.3. **alt_1** grananja (posmatra se kao celina u sekvenci);

2.3.3.1. u da grani *alt_1* grananja **o3**

2.3.3.2. u ne grani grananje **alt_2**

Kao što se iz prethodnog nabrajanja vidi, kod strukturnih algoritama se tačno može reći koja struktura se nadovezuje na koju strukturu, i koja struktura je deo

3

Osnovni elementi programskog jezika C

Programski jezici, bez obzira da li spadaju u jezike niskog, visokog ili veoma visokog nivoa, zahtevaju pisanje iskaza jezika po strogo definisanoj sintaksi. Ovo je neophodno jer su kompajleri ipak samo računarski programi, koji moraju imati jasno definisan ulaz u vidu izvornog koda kako bi generisali izvršnu verziju programa. Sintaksa jezika opisuje pravilan redosled navođenja elemenata jezika. Postoji nekoliko načina za formalno opisivanje sintaksnih konstrukcija programskih jezika. Ove forme se grubo mogu podeliti na forme koje sintaksu opisuju tekstualno, i forme koje opis sintakse predstavljaju grafički. Najpoznatija forma za opis sintakse jezika iz grupe tekstualnih opisa je Bekus-Naurova forma, a najpoznatija grafička reprezentacija sintakse programskih jezika jesu sintaksni dijagrami.

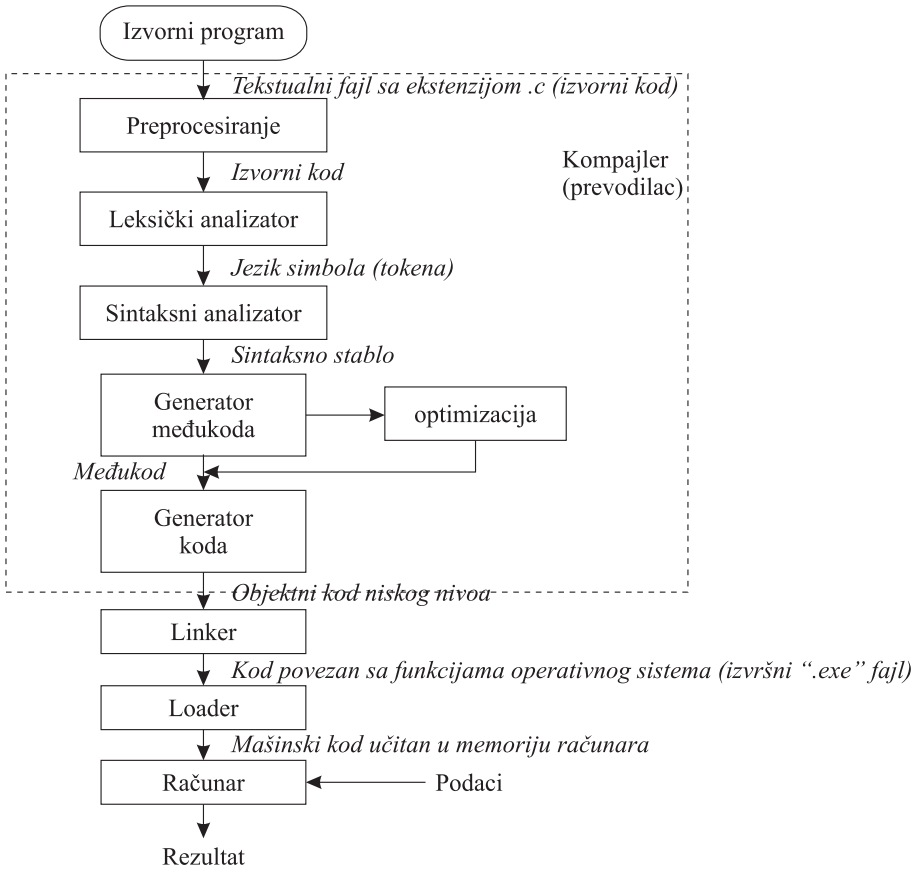
Na početku ovog poglavlja kratko ćemo kroz par primera predstaviti Bekus-Naurovu formu i sintaksne dijagrame, koje ćemo kasnije u ovom poglavlju koristiti za predstavljanje ispravnih sintaksnih konstrukcija programskog jezika C.

3.1 Formalni opis sintakse programskih jezika

Sintaksne konstrukcije programskih jezika nazivaju se i *gramatikom programskog jezika*. Za opis programskih jezika često se koriste kontekstno-slobodne gramatike. U formalnoj teoriji jezika, kontekstno-slobodna gramatika je gramatika u kojoj se svako gramatičko pravilo može izraziti u obliku

$$V \to w,$$

gde je V simbol koji se definiše, a w je niz definisanih i nedefinisanih simbola, koji se koriste za opis simbola V koji se definiše. Termin "kontekstno-slobodna" predstavlja činjenicu da se V može uvek zameniti sa w bez obzira na kontekst u kome se javlja.



Slika 3.5: Proces prevođenja i izvršavanja programa napisanih na programskom jeziku C

Sintaksni analizator će za ovaj niz odgovoriti da je konstrukcija ispravna i da predstavlja dodelu vrednosti memorijskoj lokaciji koja se interno u programu naziva promenljiva x, i generisaće odgovarajući asemblerski kod.

3.3 Azbuka i tokeni C-a

Program na programskom jeziku piše se povezivanjem osnovnih simbola (karaktera), u koje spadaju slova, cifre i specijalni znaci, u logičke celine. Azbuku programskog jezika C čine karakteri dati u tabeli 3.1.

Mala slova	a b c . . . z
Velika slova	A B C . . . Z
Cifre	0 1 2 3 4 5 6 7 8 9
Specijalni karakteri	+ = _ - ( ) * & % $ # ! < > | . , ; : ” ’ / ? { } [ ] \
Nevidljivi karakteri	*blank* (prazno mesto), nova linija, *tab* (prazna mesta)

Tabela 3.1: Skup karaktera programskog jezika C

Definicija 3.1 (Token)**.** Token programskog jezika je elementarna celina sačinjena od simbola azbuke koja ima neko značenje u programskom jeziku.

Tokeni mogu biti:

1. ključne reči,
2. identifikatori,
3. separatori,
4. konstante,
5. literali,
6. operatori.

Kao što je prikazano na slici 3.5, kompajler, odmah nakon faze preprocesiranja, prepoznaje elemente programskog jezika (tokene) i proverava u sintaksnom analizatoru da li im je redosled navođenja odgovarajući.

3.3.1 Ključne reči

Definicija 3.2 (Ključne reči)**.** Ključne reči su reči koje u programskom jeziku imaju specijalno značenje.

U ključne reči spadaju naredbe za kontrolu toka izvršenja programa, naredbe za definisanje konstanti, naredbe za definisanje specijalnih delova koda, itd. Jedna od karakteristika programskog jezika C je relativno mali broj ključnih reči. Programski jezik *Ada*, na primer, ima 62 ključne reči, ali to ne znači da su time mogućnosti C-a umanjene. Sve ključne reči programskog jezika C prikazane su u tabeli 3.2.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Tabela 3.2: Ključne reči programskog jezika C

Primer 3.13 (Program na C-u za određivanje faktorijala broja)**.** Program na programskom jeziku C za određivanje i prikazivanje faktorijala zadatog broja N, sa odgovarajućim komentarima u kodu dat je u nastavku[4].

```
1  /* Ovo je program za odredjivanje faktorijala
2  zadatog broja. Promenljive su:
3  N - ulazni parametar (zadati broj)
4  i - brojac petlje
5  F - prom. u kojoj pamtimo medjurezultate za faktorijal */
6  #include "stdio.h"
7  main()
8  {
9      int N, F, i;       // ovde deklarisemo promenljive
10     scanf("%d",&N);    // unos broja sa tastature
11     F = 1;
12     for (i = 2; i <=N; i++) // for petlja po i od 2 do N
13         F =  F * i;    // prethodnu vrednost mnozimo sa i
14     printf("%d\n",F); // prikaz faktorijala
15 }
```

△

3.3.4 Konstante

Definicija 3.6 (Konstante)**.** Konstante su konkretne vrednosti čija je vrednost napisana direktno u okviru programskog koda.

Konstante mogu biti:

1. celobrojne,
2. realne, i
3. znakovne (karakter konstante).

Celobrojne konstante

BNF notacija celobrojnih konstanti data je u primeru 3.3, EBNF notacija u primeru 3.6, a sintaksni dijagram u primeru 3.10.

Definicija 3.7 (Celobrojna konstanta u C-u)**.** EBNF notacija celobrojne konstante u programskom jeziku C je

$$CeoBroj \quad ::= \quad [\ \mathbf{0x} \mid '0'\ ]\ ['+' \mid '-']\ Cifra\ \{Cifra\}\ [\ l \mid u\ ]$$

[4]Linije programskog koda označene su brojevima sa leve strane. Ovi brojevi nisu sastavni deo programa, već će biti korišćeni za referenciranje iz teksta na određenu liniju koda.

Deklaracija konstanti

U C-u je pored promenljivih moguće deklarisati i konstante i dodeliti im simboličko ime. Za razliku od promenljivih, za konstante se ne odvaja memorijski prostor, već kompajler na svim mestima u izvornom programskom kodu na kojima se nalazi simboličko ime konstante direktno postavlja njenu vrednost. Konstantama u okviru programa nije moguće dodeliti vrednost.

EBNF deklaracije konstante je

$$\begin{aligned} \langle deklaracija\_konstante \rangle &::= \\ &\quad \textbf{const}\ \langle tip \rangle\ \langle identifikator \rangle = \langle konstanta \rangle; \\ \langle tip \rangle &::= \textbf{int} \mid \textbf{float} \mid \textbf{char} \end{aligned}$$

Vrednost iza simbola '=' je obavezna. Na primeru programa 3.5 ilustrovana je upotreba konstanti sa simboličkim imenom.

Program 3.5

```
1 main()
2 {
3     const float Pi=3.1415;
4     float R = 5., Obim;
5     Obim = 2*R*Pi;
6     printf("O=%f\n",Obim);
7 }
```

Iskaz sa funkcijom *printf* u 6. redu listinga programa 3.5 sadrži konverzioni karakter '%f' za prikaz realnih brojeva.

3.6 Standardni ulaz i izlaz

Promenljive u programu mogu dobiti vrednost na jedan od dva načina:

1. dodelom vrednosti konstante, promenljive, ili izraza,
2. unosom sa tastature ili nekog drugog spoljašnjeg uređaja.

Dodelu konstante ili vrednosti izraza promenljivoj moguće je izvršiti pomoću operatora dodele vrednosti '='. Unos sa tastature ili nekog drugog spoljašnjeg uređaja obično se naziva *ulaz*, dok se prikaz, štampanje rezultata i sl. naziva *izlaz*. Programski jezik C nema operatore ni naredbe u vidu ključnih reči za ulaz i izlaz.

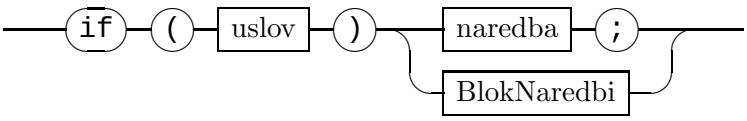
Da bi se koristili ulaz i izlaz potrebno je preprocesorskim direktivama uključiti dodatne biblioteke funkcija koje sadrže funkcije za ulaz i izlaz.

C kompajler dolazi sa velikim brojem standardnih biblioteka. Svaka biblioteka je poseban fajl na disku sa ekstenzijom *.h* (eng. *header file*, prev. *header* - zaglavlje), koji sadrži veliki broj dodatnih funkcija koje mogu značajno proširiti upotrebljivost

3.7.1 *if-then* i *if-then-else*

Ključna reč za naredbu grananja u C-u je *if*. Sintaksni dijagram grananja tipa *if-then* u C-u prikazan je na slici 3.8.

IfThen



Slika 3.8: Sintaksni dijagram grananja tipa *if-then*

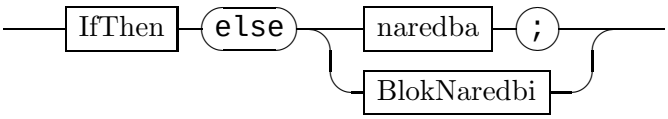
Sintaksni dijagram sa slike 3.8, opisan rečima, kaže da se iza ključne reči *if* u okviru malih zagrada '(' i ')' piše uslov grananja, nakon koga sledi naredba završena simbolom ';', ili blok naredbi. Uslov grananja u C-u može biti celobrojna promenljiva, ili celobrojni ili logički izraz. U slučaju celobrojne promenljive ili celobrojnog izraza uslov nije ispunjen ako je vrednost promenljive, odnosno izraza 0, a jeste ispunjen za bilo koju drugu vrednost.

Naredba ili blok naredbi čine telo grananja, odnosno "da" granu (definicija 2.7, strana 40). Telo grananja će se izvršiti ukoliko je uslov grananja ispunjen, a ukoliko nije telo će se preskočiti i izvršenje će nastaviti od prve naredbe koja se nalazi iza grananja.

Napomena: U telu grananja u C-u može se naći samo jedna naredba. Ukoliko je potrebno u telo grananja smestiti više od jedne naredbe, ove naredbe se moraju grupisati u vidu bloka. Imajući u vidu sintaksu bloka naredbi, ovo praktično rečeno znači da, ukoliko je u telu samo jedna naredba, mogu, ali i ne moraju se pisati zagrade '{' i '}'. Ukoliko se telo sastoji od više naredbi, ove zagrade su obavezne. Ovo važi za sve upravljačke strukture C-a, a ne samo za grananja.

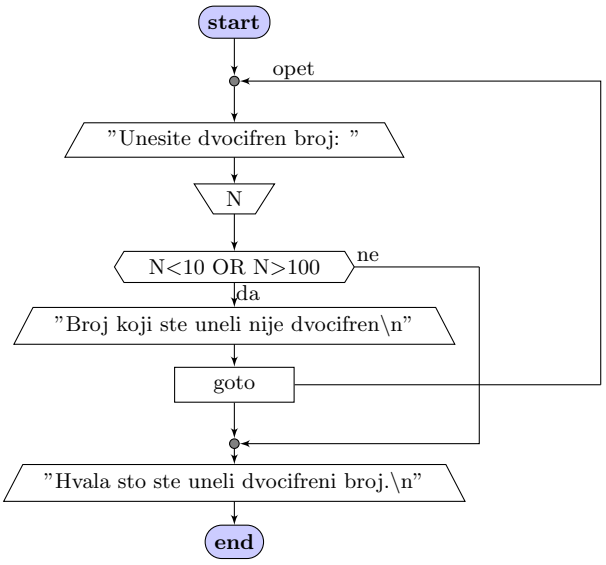
Uz dodatak ključne reči *else*, grananje tipa *if-then* postaje grananje tipa *if-then-else*. Sintaksni dijagram *if-then-else* strukture prikazan je na slici 3.9.

IfThenElse



Slika 3.9: Sintaksni dijagram grananja tipa *if-then-else*

Zaključak iz prethodne napomene za pisanje naredbe ili bloka naredbi sa ili bez zagrada '{' i '}' važi i za *else* granu.



Slika 3.19: Dijagram toka algoritma za programa sa naredbom bezuslovnog skoka

gramu omogućavaju veliku fleksibilnost programeru, ali i unose rizik od pojave semantičkih grešaka koje se teško otklanjaju, jer se program teško prati.

Upotreba naredbi skoka je posebno izražena u asemblerskim jezicima, gde su naredbe bezuslovnog i naredbe uslovnog skoka često jedine dostupne naredbe za kontrolu toka izvršenja programa. Kombinovanjem naredbi bezuslovnog skoka i strukture alternacije[11] moguće je implementirati bilo koju petlju.

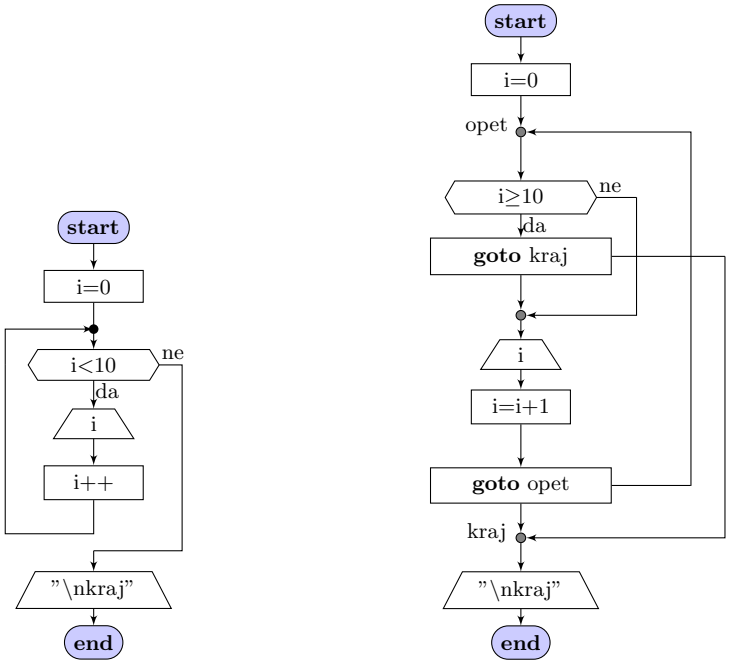
Naredbe bezuslovnog skoka, konkretno u C-u, nalaze primenu i kod modifikacije toka *switch* strukture.

Formiranje strukturne petlje naredbom bezuslovnog skoka

Kombinovanjem grananja i naredbi bezuslovnog skoka moguće je implementirati petlje tipa *while*, *do-while* i *for*. S obzirom na to da je jednostavnije implementirati petlju tipa *do-while*, razmotrićemo prvo ovaj slučaj.

Dijagram toka petlje tipa *do-while* prikazan je na slici 2.24 (strana 50). Ova petlja spada u bezuslovne petlje kod koje se telo petlje ponavlja sve dok se uslov petlje ne ispuni. Uslov petlje nalazi se na kraju petlje, nakon poslednje naredbe tela petlje.

[11] Za alternaciju se u asamblerskim jezicima koriste naredbe uslovnog skoka.



a) b)

Slika 3.21: Dijagram toka algoritma petlje tipa *while*: a) regularne petlje, b) petlje implementirane naredbama bezuslovnog skoka

Potrebno je naglasiti da je alternacija sa slike 3.21 nestrukturna, jer ima jednu ulaznu i dve izlazne grane: regularnu i preko *goto* naredbe. Program za algoritam čiji je dijagram toka dat na slici 3.21b dat je u nastavku.

```
1  #include "stdio.h"
2  main()
3  {
4      int i = 0;
5  opet:
6      if (i>=10)
7          goto kraj;
8      printf("%d, ",i);
9      i++;
10     goto opet;
11 kraj:
12     printf("\nkraj");
13 }
```

4

Promenljive, tipovi podataka i operatori

U prethodnom poglavlju bilo je reči o osnovnim elementima programskog jezika C. Detaljno su razmotreni osnovni elementi i implementacija algoritamskih struktura za kontrolu toka izvršenja programa. Pored projektovanja toka izvršenja algoritma, podjednako je važno razmotriti i podatke koje je potrebno pamtiti za vreme izvršenja programa.

U ovom poglavlju razmotrićemo tipove podataka i operacije koje je moguće izvoditi nad pojedinim tipovima podataka.

4.1 Osnovni tipovi podataka i operatori

Kao što je definicijom 2.4 precizirano (strana 34), promenljiva u programiranju je simboličko ime za lokaciju u memoriji u kojoj je moguće pamtiti vrednosti i čitati ih iz nje. Memorijska lokacija može pamtiti binarni niz nula i jedinica određene dužine.

Za promenljivu je neophodno znati format zapisa u memoriji, odnosno kako "tumačiti" upisane nule i jedinice. Naime, ako kažemo da je sadržaj šesnaestobitne lokacije sa simboličkim imenom A

$$A = (0011\ 1000\ 0000\ 0010)_2,$$

to nije dovoljno. Potrebno je reći u kom formatu je zapis. Ukoliko je celobrojni zapis, tada navedene nule i jedinice predstavljaju dekadni broj 14338. Ukoliko je A realni broj sa 4-bitnim eksponentom i 12-bitnom mantisom u dvojičnom komplementu, tada se radi o broju $-2046 \cdot 10^3$ (ako je baza 10). Naravno, ako je mantisa u nekom drugom formatu, rezultat, odnosno tumačenje sadržaja memorijske lokacije bi bilo drugačije.

$(11111111)_2$ i $m = 0$, što predstavlja beskonačnost, odnosno na engleskom *infinity*, od čega je i skraćenica INF u 3. redu izlaza. △

4.1.2 Znakovni podaci

Često se u programiranju javlja potreba za obradom reči i rečenica govornog jezika, potreba za obradom teksta generalno, ili jednostavno prikazivanje tekstualnih poruka. Za kodiranje znakovnih podataka u programskim jezicima najčešće se koristi ASCII standard[3].

Osnovni ASCII standard specificira numeričke kodove za 128 simbola. Numerički kodovi su u opsegu od 0 do 127. Preslikavanje "simbol - kod" dato je tzv. ASCII tabelom. Simboli ASCII tabele su:

1. mala i velika slova engleske abecede,
2. decimalne cifre od 0 do 9,
3. znaci interpunkcije, i
4. specijalni simboli.

Svi simboli i odgovarajući kodovi ACSII tabele prikazani su na slici 4.3.

000	(nul)	016 ►	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001 ☺	(soh)	017 ◄	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002 ☻	(stx)	018 ↕	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003 ♥	(etx)	019 ‼	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004 ♦	(eot)	020 ¶	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005 ♣	(enq)	021 §	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006 ♠	(ack)	022 ▬	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007 •	(bel)	023 ↨	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008 ◘	(bs)	024 ↑	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009	(tab)	025 ↓	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010	(lf)	026	(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011 ♂	(vt)	027 ←	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012 ♀	(np)	028 ∟	(fs)	044	,	060	<	076	L	092	\	108	l	124	|
013	(cr)	029 ↔	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014 ♫	(so)	030 ▲	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015 ☼	(si)	031 ▼	(us)	047	/	063	?	079	O	095	_	111	o	127	⌂

Slika 4.3: ASCII tabela

Definicija 4.5 (Karakter)**.** Karakterom u programiranju naziva se jedan simbol ASCII tabele.

Napomena: ASCII tabela nije jedini standard za kodiranje znakovnih podataka. Kao što se sa slike 4.3 može primetiti, u tabeli ne postoje slova č, ć, š, ž i đ. Takođe, u tabeli se nalazi samo latinično pismo. UNICODE je skup standarda, od kojih su

[3]ASCII - American Standard Code for Information Interchange

5

Osnovne strukture podataka

Struktura podataka u opštem smislu je skup od više podataka, koji su na neki način povezani. Veze koje postoje između podataka uglavnom modeluju veze koje postoje među podacima u stvarnom svetu za problem koji se rešava na računaru.

Potrebno je naglasiti da je struktura podataka termin koji postoji u programiranju uopšte, i generalniji je od strukturnog tipa podataka (ključna reč *struct* u C-u). Za kreiranje složenijih struktura podataka koriste se osnovni i izvedeni tipovi podataka. Strukturom podataka opisuju se podaci iz domena problema koji se rešava na računaru.

5.1 Linearne i nelinearne strukture podataka

Generalno, sve strukture mogu se podeliti u dve grupe:

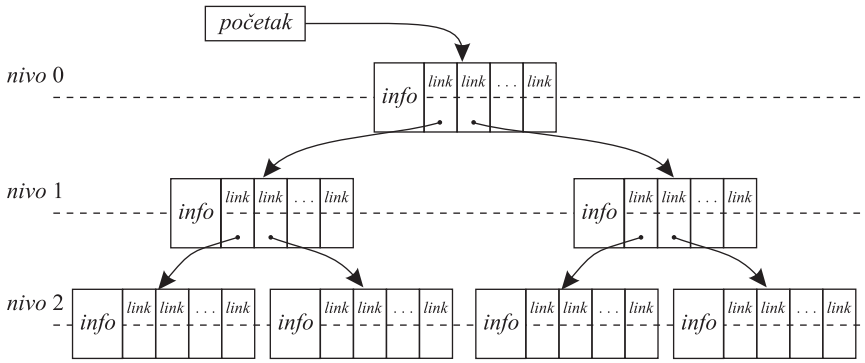
1. linearne, i
2. nelinearne strukture.

Definicija 5.1 (Linearne strukture)**.** Linearna struktura je skup podataka gde svaki podatak ima tačno dva susedna elementa, sem dva elementa na krajevima strukture, koji imaju po jednog suseda.

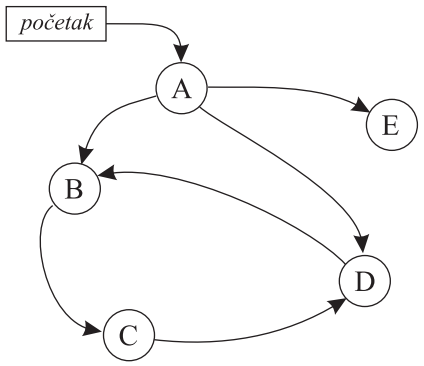
Kaže se da svaki element ima levog i desnog suseda.

Definicija 5.2 (Nelinearne strukture)**.** Nelinearna struktura je skup podataka u kome ima podataka koji imaju više od dva susedna elementa.

Na primer, linearnom strukturom se može opisati muzička numera na CD-u. Naime, digitalni zapis zvuka je niz sukcesivnih celih brojeva, poređanih jedan za drugim, koji redom predstavljaju vrednosti amplitude zvuka u datim trenucima. Ovo je ilustrovano na slici 5.1. Kod zapisa u CD kvalitetu, zvuk se zapisuje sa 44.100 16-bitnih brojeva za jednu sekundu zvuka. Sve manje od toga je lošiji kvalitet zvuka od CD kvaliteta.



Slika 5.31: Struktura stabla



Slika 5.32: Struktura grafa

odgovara po jedna vrsta i po jedna kolona, odnosno čvoru i odgovara i-ta kolona i i-ta vrsta, a na postojanje veze (grane) između čvorova i i j obično ukazuje vrednost elementa $a_{i,j} = 1$. Vrednost $a_{i,j} = 0$ sugeriše da su čvorovi nepovezani.

Matrična reprezentacija grafa sa slike 5.32 prikazana je na slici 5.33. Na slici je posebno naglašena vrednost 1 u preseku vrste koja odgovara čvoru B i kolone koja odgovara čvoru C, što predstavlja postojanje grane između ova dva čvora.

6

Funkcije

Funkcija je izdvojena programska celina (potprogram), čiji je zadatak da na osnovu određenog algoritma transformiše ulazne podatke u novi podatak. Ulazni podaci funkcije nazivaju se **parametri funkcije**, a izlazni podatak se naziva **rezultat funkcije**.

Funkcije u programiranju dobile su naziv na osnovu koncepta matematičkih funkcija. Funkcija se u matematici može predstaviti kao:

$$y = f\left(x_1, x_2, \ldots, x_n\right),$$

gde je sa f označena funkcija, sa x_1 do x_N parametri, a sa y rezultat. Funkcija ima više ulaznih parametara i jedan izlazni rezultat.

U programiranju se pored funkcija koristi i koncept *procedura*. Uobičajeno je u programskim jezicima da se funkcijom naziva potprogram koji vraća jedan rezultat, a da se potprogram koji može vratiti više rezultata naziva procedurom. Mnogi programski jezici imaju i drugačije sintaksne konstrukcije za definisanje ova dva tipa potprograma. Dakle, procedura je potprogram koji može imati više ulaznih parametara, ali za razliku od funkcija može imati i više izlaznih rezultata. U C-u procedure kao poseban koncept ne postoje. Za implementaciju procedura u C-u koriste se funkcije, sa specifičnim načinom prenosa parametara, o kom će biti reči u kasnijim poglavljima.

U zavisnosti od toga ko implementira funkciju, funkcija može biti korisnička ili standardna (bibliotečka). Korisničke funkcije piše sam programer, dok se standardne funkcije isporučuju uz kompajler.

Pomoću funkcija se može izvršiti dekompozicija problema i sam problem rešiti jednostavnije. Dovoljno je da programer pozove funkciju na izvršenje i preda joj parametre, nakon čega će dobiti očekivani rezultat. U slučaju standardnih funkcija nije neophodno da programer zna po kom algoritmu funkcija radi da bi mogao da koristi funkciju. Kaže se da funkcija sakriva (*enkapsulira*, *apstrahuje*) sam postupak kojim ostvaruje svoju funkcionalnost.

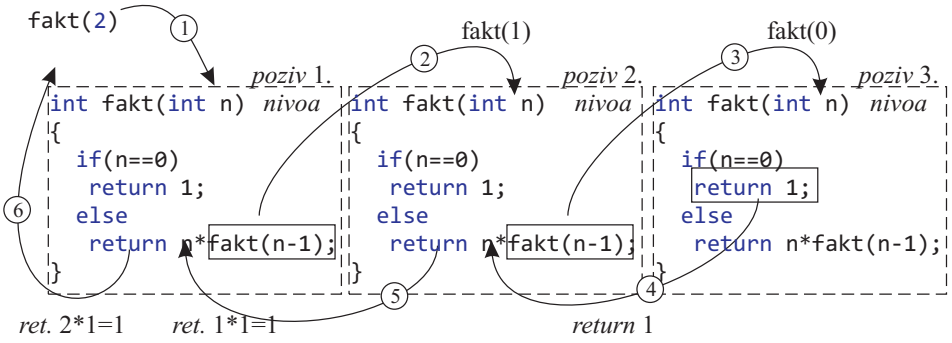
Funkcije se mogu pozvati na izvršenje iz glavnog programa, ili iz druge funkcije. Primer dekompozicije programa koji sadrži funkcije prikazan je na slici 6.1. U pri-

```
10 {
11         printf("Faktorijel broja 5 je %d.", fakt(5) );
12 }
```

Izlaz koji ovaj program daje je:

```
Faktorijel broja 5 je 120.
```

Ilustracija poziva ove rekurzivne funkcije data je na slici 6.9. Kako bi slika bila jednostavnija i preglednija, na slici 6.9 prikazan je poziv rekurzivne funkcije za početnu vrednost $n = 2$. Na slici je strelicama označen redosled poziva i povratka iz različitih nivoa rekurzije, kao i vrednosti kojima se određene instance pozivaju i vrednosti koje vraćaju. Svakoj strelici na slici 6.9 pridružen je redni broj u redosledu izvršenja. △



Slika 6.9: Ilustracija poziva rekurzivne funkcije na primeru funkcije za izračunavanje faktoriela

U cilju realizacije rekurzije, prilikom prelaska sa jednog nivoa rekurzije na naredni nivo, kompajler na stek pamti stanje u kom je trenutna funkcija zaustavljena. Povratkom na prethodni nivo rekurzije, sa vrha steka se uzimaju neophodne informacije, kako bi izvršenje moglo da se nastavi sa mesta gde se stalo.

Napomena: Ukoliko rekurzivna funkcija nema uslov za kraj rekurzije, rekurzivni pozivi nisu ograničeni i dubina rekurzije se povećava sve dok ne napuni stek koji je kompajler rezervisao za potrebe rekurzije. Ovakvi programi se obično završavaju greškom "prekoračenje steka" (eng. *stack overflow*).

Primer 6.9 (Rekurzivna funkcija za određivanje NZD brojeva)**.** NZD se rekurzivno može definisati kao:

$$NZD(n,m) = \begin{cases} n, & n = m \\ NZD(n-m,m), & n > m \\ NZD(m-n,n), & n < m \end{cases} . \tag{6.2}$$

Rekurzivna funkcija za određivanje NZD broja je data u nastavku.

U prvom pozivu će biti učitano prvih 29 karaktera, t.j. tekst: *"Prvi red je u ovom slucaju du"*, a kao poslednji karakter u stringu će biti pridodat simbol za kraj stringa '\0'. Ovaj poziv će ujedno i povećati interni brojač u strukturi `FILE`, kako bi se označio deo fajla koji je pročitan. Sledeći poziv će pročitati tekst do kraja reda, ali ne i naredni red, pa će vrednost u stringu *A* posle drugog poziva biti *"zi od ostalih."*.

Treći poziv će učitati samo drugi red, a četvrti poziv treći red. △

getc

Nekada je zbog prirode konkretnog algoritma za obradu tekstualnog fajla optimalnije i jednostavnije čitati jedan po jedan karakter iz fajla i odmah ih obrađivati, umesto učitavanja cele linije u string, odakle se nakon toga vrši obrada. Funkcija koja omogućava čitanje jednog po jednog karaktera iz tekstualnog fajla je funkcija *getc* iz standardne biblioteke *"stdio.h"*.

1. **Deklaracija funkcije**

   ```
   int getc(FILE *tok)
   ```

2. **Opis dejstva**

   Čita jedan karakter iz fajla, odnosno ulaznog toka podataka.

3. **Parametri**

   tok – Tok sa koga se učitavaju podaci.

4. **Povratna vrednost**

   Funkcija čita podatak tipa *unsigned char*, konvertuje ga i vraća podatak tipa *int*. U slučaju da se prilikom čitanja fajla došlo do kraja fajla, kao i u slučaju greške, funkcija vraća specijalni karakter čiji je ASCII kod jednak vrednosti simboličke konstante iz standardne biblioteke *"stdio.h"* pod nazivom **EOF**. Ime ove simboličke konstante je skraćenica od engleskih reči *End Of File*, što u prevodu znači kraj fajla.

Sledeći primer ilustruje upotrebu funkcije *getc*.

Primer 6.43 (Upotreba funkcije *getc* za učitavanje jednog karaktera iz fajla)**.** Ukoliko je na primer potrebno odrediti broj karaktera u fajlu, to je jednostavnije učiniti čitanjem jednog po jednog karaktera iz fajla. Sledeći program određuje broj karaktera u fajlu *"niz.txt"*.

```
1 #include "stdio.h"
2 main()
3 {
4     int B=0;
5     FILE *f = fopen("niz.txt", "r");
```

Literatura

[1] Brian Kernighan, Dennis Ritchie, "Programski jezik C", Prentice Hall, 1988, prevod Naučna knjiga, Beograd , 1989.

[2] Grupa autora, "Algoritmi i programiranje, zbirka rešenih zadataka na programskom jeziku C", Elektronski fakultet, Niš, 2012, ISBN 978-86-6125-069-9.

[3] Laslo Kraus, "Programski jezik C sa rešenim zadacima", deveto izdanje, Akademska Misao, 2014, ISBN 978-86-7466-441-4.

[4] Vladan Vujičić, "Uvod u C jezik", četvrto izdanje, Institut za nuklearne nauke, Vinča, 1991, ISBN 86-80055-04-2.

[5] Suzana Stojković, Natalija Stojanović, Dragan Stojanović, "Uvod u računarstvo", Elektronski fakultet u Nišu, 2014.

[6] Edmund Berkeley, "Giant Brains, or Machines That Think", John Wiley & Sons, 1949.

[7] Stephen Kochan, "Programming in C", 4th Edition, Addison-Wesley, 2014, ISBN 978-0321776419.

[8] Joyce Farrell, "Programming Logic and Design, Comprehensive", 7th Edition, Cengage Learning, 2012, ISBN 978-1111969752.

Index

Edicija: Osnovni udžbenici

ISBN 978-86-6125-119-1