[1. Uvod 3](#_Toc156046475)

[2. Tipovi Podataka u Jeziku C 4](#_Toc156046476)

[2.1. Cijelobrojni ,realni i znakovni tipovi podataka: 4](#_Toc156046477)

[2.2. Složeni Tipovi: 4](#_Toc156046478)

[2.3. Veličina i raspon tipova podataka: 4](#_Toc156046479)

[2.4. Specifikatori Tipova: 5](#_Toc156046480)

[3. Deklaracija i Inicijalizacija Promenljivih u Jeziku C 5](#_Toc156046481)

[3.1. Deklaracija Promenljivih: 6](#_Toc156046482)

[3.2. Inicijalizacija promjenljivih 6](#_Toc156046483)

[3.3. Lokalne i Globalne Promenljive 6](#_Toc156046484)

[3.4. Konstante i modifikatori tipova 7](#_Toc156046485)

[4. Operatori i Izrazi u Jeziku C 7](#_Toc156046486)

[4.1. Aritmetički Operatori 7](#_Toc156046487)

[4.2. Logički Operatori 8](#_Toc156046488)

[4.3. Relacioni Operatori 8](#_Toc156046489)

[4.4. Prioritet i asocijativnost operatora 8](#_Toc156046490)

[5. Korištenje konstanti u jeziku C 9](#_Toc156046491)

[5.1. Vrste konstanti 9](#_Toc156046492)

[5.2. #define direktiva 10](#_Toc156046493)

[5.3. Prednosti Korištenja Konstanti 10](#_Toc156046494)

[6. Rad sa Nizovima u Jeziku C 11](#_Toc156046495)

[6.1. Deklaracija Niza 11](#_Toc156046496)

[6.2. Višedimenzionalni Nizovi 12](#_Toc156046497)

[6.3. Stringovi kao Nizovi Karaktera 12](#_Toc156046498)

[6.4. Prednosti i Izazovi Radnje sa Nizovima 12](#_Toc156046499)

[7. Rad sa strukturama u jeziku C 13](#_Toc156046500)

[7.1. Definicija i deklaracija strukture 13](#_Toc156046501)

[7.2. Ugnježdenje Struktura 14](#_Toc156046502)

[7.3. Prednosti i mane korištenja struktura 15](#_Toc156046503)

[8. Rad sa Pokazivačima u Jeziku C 15](#_Toc156046504)

[8.1. Pregled Pokazivača 15](#_Toc156046505)

[8.2. Deklaracija Pokazivača 16](#_Toc156046506)

[8.3. Alokacija Memorije 16](#_Toc156046507)

[8.4. Oslobađanje Memorije 17](#_Toc156046508)

[8.5. Pristupanje Vrednostima putem Pokazivača 17](#_Toc156046509)

[8.6. Aritmetika Pokazivača 17](#_Toc156046510)

[8.7. Pokazivači na Funkcije 17](#_Toc156046511)

[8.8. Prednosti Korišćenja Pokazivača 18](#_Toc156046512)

[9. Efikasnost i optimizacija koda 18](#_Toc156046513)

[9.1. Pravilan izbor tipova podataka 18](#_Toc156046514)

[9.2. Pravilna Upotreba Struktura 19](#_Toc156046515)

[9.3. Strategije za Optimizaciju Koda 19](#_Toc156046516)

[9.4. Upotreba Inline Optimizacija 19](#_Toc156046517)

[10. Zaključak 20](#_Toc156046518)

# Uvod

U današnjem digitalnom svijetu, podaci predstavljaju ključnu komponentu svih softverskih sistema. Upravljanje podacima, kako ih predstavljamo, organizujemo i manipulišemo, je od ključnog značaja za razvoj efikasnih i optimizovanih programa i aplikacija. U ovom seminarskom radu, fokusirat će mo se na predstavljanje podataka u jeziku C, jednom od najuticajnijih programskih jezika u historiji programiranja.

Predstavljanje podataka u jeziku C nije samo tehnički aspekt programiranja, već ključni faktor u razumijevanju i optimizaciji softverskih rješenja. Ovaj rad ima za cilj pružiti sveobuhvatan pregled osnovnih pojmova, pravila i tehnika koje programeri koriste pri radu s podacima u jeziku C.

Efikasno upravljanje podacima je temelj optimiyovanog i pouydanog softvera. Kako bi se postigla optimalna izvršavanja, programeri moraju pažljivo razmatrati način na koji podaci izgledaju u memoriji, kako su organizovani, i kako se pristupa njima. Jezik C, poznat po svojoj bliskoj vezi s hardverom, pruža programerima punu kontrolu nad podacima, ali zahtijeva precizno upravljanje.

Osnovni koncepti predstavljanja podataka obuhvataju tipove podataka, deklaraciju promjenljivih, i osnovne operacije nad podacima. Tipovi podataka, poput cijelobrojnih, realnih, znakovnih, i složenih tipova, čine osnovnu građu podataka u jeziku C. Deklaracija promjenljivih određuje kako će podaci biti rezervisani u memoriji, dok osnovne operacije, kao što su aritmetičke i logičke operacije, omogućavaju manipulaciju podacima.

U narednim sekcijama ovog rada, dublje ćemo istražiti ove koncepte i njihovu primjenu u programiranju. Osvrnućemo se na specifičnosti jezika C koje čine predstavljanje podataka u ovom jeziku jedinstvenim i značajnim aspektom programiranja.

# Tipovi Podataka u Jeziku C

U jeziku C, tipovi podataka čine osnovnu gradivu programa, omogućavajući programerima da efikasno predstave i manipulišu podacima. Ovaj segment istražuje različite tipove podataka u jeziku C, pružajući detaljan pregled celobrojnih, realnih, znakovnih i složenih tipova.

## Cijelobrojni ,realni i znakovni tipovi podataka:

Cijelobrojni tipovi uključuju različite formate cijelih brojeva, omogućavajući programerima da rade s različitim rasponima vrijednosti. Ovi tipovi uključuju int, short, long i char. int i char su najčešće korišteni tipovi podataka gdje se int koristi za predstavljanje cijelih brojeva, dok char predstavlja pojedinačne znakove.

Realni tipovi se koriste za predstavljanje brojeva s decimalnim zarezom. Klasični realni tipovi u jeziku C uključuju **float** i **double**, **float** se koristi za manje precizne vrijednosti, dok **double** pruža veću preciznost.

Znakovni tipovi predstavljaju pojedinačne znakove i koriste se za rad s karakterima. Osnovni znakovni tip je **char**, koji može čuvati jedan znak iz ASCII tabele.

## Složeni Tipovi:

Složeni tipovi podataka omogućavaju programerima da grupišu više sličnih ili različitih podataka pod jednim imenom. Ovde spadaju strukture (struct), unije (union) i nabrajanja (enum). Strukture dozvoljavaju kombinovanje različitih tipova podataka u jednoj strukturi, dok unije omogućavaju deljenje istog prostora za više tipova podataka. Enumi omogućavaju definisanje skupa konstanti koje se koriste za predstavljanje brojeva određenog skupa.

## Veličina i raspon tipova podataka:

Različiti tipovi podataka imaju različite veličine i opsege vrednosti koje mogu predstavljati.Neke od ovih vrijednosti su

### char

Veličina tipa podataka char je 1 bajt, a opseg vrednosti koje može predstavljati kreće se od -128 do 127 za znakove ili od 0 do 255 za bez-znakovne vrednosti.

### int

integer (int) ima obično veličinu od 4 bajta (može varirati), a opseg vrednosti koje može zadržati je od -2,147,483,648 do 2,147,483,647.

### float

Float tip podataka zauzima 4 bajta i može predstavljati vrednosti u opsegu približno od ±1.2 × 10^-38 do ±3.4 × 10^38, sa oko 6 decimala preciznosti.

### double

Double, obično veličine 8 bajta (ali može varirati), ima opseg vrednosti približno od ±2.3 × 10^-308 do ±1.7 × 10^308, sa oko 15 decimala preciznosti.

### short

Short, čija je veličina obično 2 bajta, može sadržavati vrednosti u opsegu od -32,768 do 32,767.

### long

Long tip podataka, sa veličinom od obično 4 ili 8 bajta, može zadržavati vrednosti u opsegu od -2,147,483,648 do 2,147,483,647 (za 4 bajta) ili od -9,223,372,036,854,775,808 do 9,223,372,036,854,775,807 (za 8 bajta).

### unsigned

Unsigned modifikator može se koristiti sa gotovo svim tipovima podataka i označava da tip ne može imati negativne vrednosti. Na primer, unsigned int ima opseg vrednosti od 0 do 4,294,967,295 umesto da ima negativne vrednosti.

## Specifikatori Tipova:

Specifikatori tipova se koriste za preciznije definisanje veličine i načina interpretacije podataka. Primjeri uključuju short, long, signed i unsigned specifikatore koji modifikuju osnovne tipove.

Jezik C omogućava konverziju između različitih tipova podataka. Postoje implicitne konverzije koje se automatski dešavaju u određenim situacijama, ali programer takođe može izvršiti eksplicitnu konverziju pomoću operacija kao što su `type-casting`.

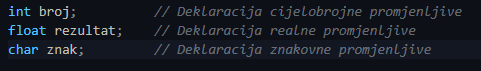
Raznolikost tipova podataka u jeziku C omogućava programerima fleksibilnost i kontrolu nad resursima koje koriste. Razumijevanje karakteristika svakog tipa podataka ključno je za efikasno programiranje u jeziku C. U narednim dijelovima rada, detaljnije ćemo analizirati svaki od ovih tipova, pružajući dublji uvid u njihove karakteristike i optimalnu upotrebu.

# Deklaracija i Inicijalizacija Promenljivih u Jeziku C

Deklaracija i inicijalizacija promjenljivih su osnovni koraci u procesu programiranja u jeziku C. Ova tema ima ključnu ulogu u pravilnom upravljanju memorijom i omogućava programerima da efikasno koriste podatke u svojim programima. U narednim sekcijama istražićemo detalje procesa deklaracije i inicijalizacije promenljivih, istražujući njihovu sintaksu, značaj i primjene.

## Deklaracija Promenljivih:

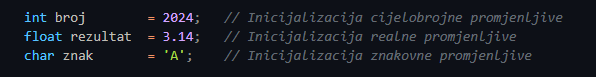
Deklaracija promenljivih u jeziku C predstavlja proces informisanja kompajlera o tipu podataka i imenu promenljive koja će se koristiti u programu(*slika 1.*). Pravilna deklaracija je ključna za definisanje promenljivih pre nego što se koriste u programu. Razmotrićemo sintaksu deklaracije, tipove podataka koji se mogu deklarisati, kao i pravila koja treba poštovati.



*slika 1.*

## Inicijalizacija promjenljivih

Inicijalizacija je proces dodeljivanja početne vrijednosti promjenljivoj prilikom njenog kreiranja. Ovo osigurava da promjenljiva ima početnu vrijednost pre nego što se prvi put koristi u programu. Sintaksa inicijalizacije prati deklaraciju, a koristi se operator dodijele „=“ (*slika 2.*).



*slika 2.*

Pravilna deklaracija i inicijalizacija ključne su za spriječavanje grešaka tijekom izvršavanja programa. Nepravilna upotreba promjenljivih može dovesti do nepredvidljivog ponašanja programa, pa je pažljiva implementacija ova dva koraka od ključnog značaja za pravilan rad softvera.

## Lokalne i Globalne Promenljive

Razlika između lokalnih i globalnih promjenljivih ima direktnu povezanost s oblastima opsega promjenljivih u softveru. Opsega (scope) promjenljivih u programiranju određuje gdje u kodu možemo pristupiti i koristiti određeni identifikator, kao što su promjenljive. Lokalne promenljive imaju lokalni opseg, ograničen na određeni dio koda, kao što je funkcija, dok globalne promjenljive imaju globalni opseg, omogućavajući im da budu vidljive širom cijlog programa.

## Konstante i modifikatori tipova

Pored standardnih promjenljivih, u jeziku C imamo i konstante koje se deklarišu ključnom riječi const. Modifikatori tipova, poput volatile i register, dodatno proširuju mogućnosti deklaracije i inicijalizacije promjenljivih.

* **const**: Oznaka koja ukazuje da se vrijednost promjenljive ne smije mijenjati tijekom izvršavanja programa, čime se obezbijeđuje konstantnost.
* **volatile**: Oznaka koja ukazuje da vrijednost promjenljive može biti promjenjena izvan standardnog toka izvršavanja, čime se sprečavaju optimizacije kompajlera koje bi mogle uticati na neočekivano ponašanje koda.
* **register**: Oznaka koja sugeriše kompajleru da smjesti promjenljivu u registar procesora radi bržeg pristupa, ali kompajler može ignorisati ovu sugestiju.

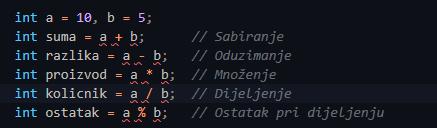
# Operatori i Izrazi u Jeziku C

Operatori i izrazi u jeziku C omogućavaju programerima da manipulišu podacima na različite načine. Operatori se koriste za izvođenje operacija nad promjenljivama, konstantama i drugim izrazima, dok izrazi predstavljaju kombinaciju operatora i operandi koji daju vrijednost. Jezik C podržava različite vrste operatora, uključujući aritmetičke, relacione, logičke i bitovske, što programerima pruža svestranost u radu sa podacima.

Izrazi se mogu sastojati od promjenljivih, konstanti, poziva funkcija, i operatora koji ih povezuju. Pravilno razumijevanje operatora i izraza ključno je za efikasno pisanje programa u jeziku C. Ovi elementi čine osnovnu građevinsku jedinicu u kreiranju algoritama i izražavaju logiku i manipulacije podacima u programima.

## Aritmetički Operatori

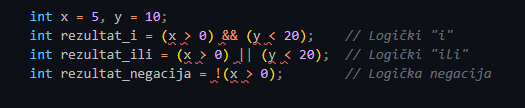
Aritmetički operatori omogućavaju izvođenje matematičkih operacija nad numeričkim vrijednostima. Ovi operatori (*slika 3.*) uključuju sabiranje „+“, oduzimanje „-“ , množenje „\*“, djieljenje „/“ i ostatak pri dijeljenju „%“.



*slika 3.*

## Logički Operatori

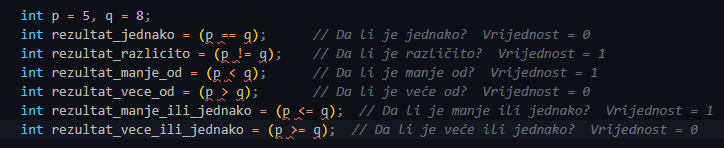
Logički operatori se koriste za izvođenje logičkih operacija nad boolean vrijednostima (**true** ili **false**). Ovi operatori uključuju „&&“ (i), „||“ (ili) i „!“ (negacija). Na slici ispod (*slika 4.*) je prikazano kako ovi operatori kombinuju uslove.



*slika 4.*

## Relacioni Operatori

Relacioni operatori porede vrednosti i vraćaju rezultat u obliku boolean vrijednosti (*slika 5.*). Operatori uključuju „==“ (jednako), „!=“ (različito), „<“ (manje od), „>“ (veće od), „<=“ (manje ili jednako) i „>=“ (veće ili jednako).



*slika 5.*

## Prioritet i asocijativnost operatora

Prioritet i asocijativnost operatora igraju ključnu ulogu u tačnom tumačenju izraza u jeziku C. Operatori s višim prioritetom izvršavaju se prije operatora s nižim prioritetom. Na primjer, aritmetički operatori poput množenja (\*) i dijeljenja (/) imaju viši prioritet od operatora zbrajanja (+) i oduzimanja (-).

U situacijama gdje imamo jednaki prioritet operatora, asocijativnost operatora određuje redoslijed izvršavanja. Na primjer, aritmetički operatori obično imaju lijevu asocijativnost, što znači da će se operacije izvršiti s lijeva na desno.

Programeri moraju biti svjesni prioriteta i asocijativnosti operatora kako bi napisali precizne izraze. Na primjer, u izrazu "a + b \* c", množenje će se izvršiti prije zbrajanja zbog višeg prioriteta množenja.

Razumijevanje ovih svojstava pomaže programerima da napišu izraze koji jasno izražavaju željene operacije i smanjuje potencijal za greške u kodu.

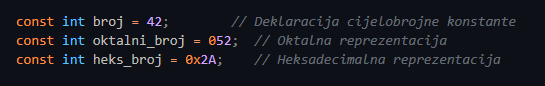
# Korištenje konstanti u jeziku C

Konstante u jeziku C predstavljaju nepromenljive vrednosti koje se koriste u programima kako bi se identifikovale ili pružile vrijednosti koje ne bi trebalo da se mijenjaju tijekom izvršavanja programa.

## Vrste konstanti

### Cijelobrojne konstante

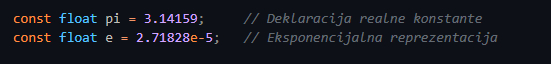
Cijelobrojne konstante(*slika 6.*) predstavljaju nepromjenljive vrijednosti cijelih brojeva. Mogu biti u decimalnom, oktalnom (bazom 8) ili heksadecimalnom (bazom 16) formatu.



*slika 6.*

### Realne Konstante

Realne konstante (*slika 7.*) predstavljaju nepromjenljive vrijednosti sa decimalnim zarezom. Mogu biti u obliku običnog broja ili eksponencijalnog formata.



*slika 7*

### Znakovne Konstante

Znakovne konstante (*slika 8.*) predstavljaju pojedinačne znakove i deklarišu se između jednostrukih apostrofa.



*slika 8.*

### Nizovne Konstante

Nizovne konstante(*slika 8.*) predstavljaju niz karaktera i koriste se za čuvanje teksta ili niza znakova.



*slika 8.*

## #define direktiva

#define direktiva u programskom jeziku C koristi se za definisanje konstanti (*slika 9.*), tj. za dodjeljivanje imena određenim vrijednostima. Ova direktiva omogućava programeru da stvori simboličko ime za određenu vrijednost ili izraz, čime se olakšava čitanje i održavanje koda.



*slika 9.*

## Prednosti Korištenja Konstanti

* **Čitljivost koda**: Konstante obezbjeđuju imenovane vrijednosti koje pomažu u razumijevanju svrhe i upotrebe određenih brojeva u programu.
* **Održavanje koda**: Ako se vrijednost konstante mora promijeniti, dovoljno je izmeniti jedno mesto u kodu, što olakšava održavanje.

Korištenje konstanti u jeziku C pruža stabilnost i čitljivost kodu. Konstante često služe kao imenovane vrijednosti koje olakšavaju razumevanje i održavanje programa.

# Rad sa Nizovima u Jeziku C

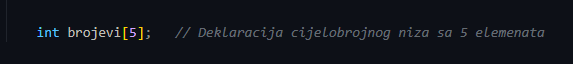
Nizovi u jeziku C predstavljaju strukturu podataka koja omogućava pohranjivanje više elemenata istog tipa podataka. Nizovi olakšavaju rad s većim skupovima podataka, omogućujući programerima da grupišu slične informacije na jednom mjestu.

Jedna od ključnih karakteristika nizova je indeksiranje, gdje svaki element niza ima jedinstveni indeks. Prvi element niza ima indeks 0, drugi ima indeks 1, i tako dalje. Ovo omogućava efikasan pristup pojedinačnim elementima niza.

Rad s nizovima uključuje različite operacije poput inicijalizacije, pristupa elementima, dodavanja i brisanja elemenata, te pretrage i sortiranja. Razumijevanje nizova ključno je za efikasno upravljanje podacima u C programima.

## Deklaracija Niza

Deklaracija niza (*slika 10.*) obuhvata specificiranje tipa podataka elemenata niza i određivanje veličine niza.



*slika 10.*

Ova deklaracija rezerviše prostor za 5 cijelobrojnih elemenata u memoriji kojima možemo pristupiti (*slika 11.*) uz pomoć indeksiranja niza. Ako nam je potrebna veličina niza, nju možemo dobiti uz pomoć operatora „sizeof“ (*slika 12.*). Jedan od izazova u radu s nizovima je pažnja na granice niza. Prekoračenje može dovesti do neželjenih efekata, uključujući nepredviđeno ponašanje programa.



*slika 11.*



*slika 12.*

## Višedimenzionalni Nizovi

Višedimenzionalni nizovi omogućavaju programerima da organizuju podatke u strukturi matrice, koristeći koncept redova i kolona. Na primjer (*slika 13.*), kada deklarišemo 2D niz (matricu).



*slika 13.*

Ova deklaracija kreira matricu dimenzija 3x3, što znači da ima tri reda i tri kolone. Elementi matrice se pristupaju korištenjem dva indeksa, jednog za red i jednog za kolonu (*slika 14.*).



*slika 14.*

Višedimenzionalni nizovi su posebno korisni kada radimo s tabelarnim podacima ili matricama u programiranju.

## Stringovi kao Nizovi Karaktera

Stringovi u jeziku C često se realizuju kao nizovi karaktera (*slika 15.*).



*slika 15.*

U ovom slučaju, string "Apeiron" čuva se kao niz karaktera u promenljivoj ` naziv\_univerziteta `. Svaki karakter u stringu ima svoj indeks, počevši od 0. Ovakva implementacija omogućava manipulaciju tekstualnim podacima, uključujući i korištenje različitih funkcija koje jezik C pruža za rad sa stringovima.

## Prednosti i Izazovi Radnje sa Nizovima

* **Efikasnost**: Nizovi omogućavaju efikasnu pohranu i manipulaciju podacima.
* **Izazovi**: Obratiti pažnju na granice niza kako bi se izbegli potencijalni problemi prekoračenja.

Rad sa nizovima u jeziku C pruža moćan mehanizam za organizaciju i rad sa skupovima podataka. Od deklaracije do pristupa elementima, nizovi su neophodan deo svakodnevnog programiranja u jeziku C. Pažnja na granice i pažljiva manipulacija nizovima ključni su za siguran i efikasan kod. Naredne stranice će se fokusirati na napredne koncepte vezane za rad sa nizovima.

# Rad sa strukturama u jeziku C

Strukture u jeziku C predstavljaju način za organizaciju različitih tipova podataka pod jednim imenom. Ove strukture omogućavaju programerima da grupišu raznovrsne podatke unutar jednog entiteta, što olakšava rad s kompleksnijim podacima.

Uz definiciju strukture, programeri mogu kreirati nove tipove podataka koji sadrže različite varijable različitih tipova. Ove strukture omogućavaju prilagodljivost i organizaciju podataka prema potrebama programa.

## Definicija i deklaracija strukture

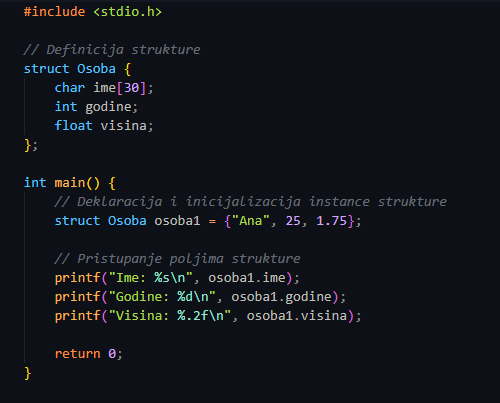
Definicija strukture u jeziku C (*slika 12.*) odnosi se na opisivanje same strukture, uključujući njen tip i sastavne dijelove, tj. polja koja je čine. Ova definicija obuhvata informacije o vrstama podataka koje struktura sadrži i kako su organizovane.

S druge strane, deklaracija strukture (*slika 12.*) je proces rezervisanja memorije za tu strukturu i stvaranje instanci (kopija) strukture koje će se koristiti u programu. Deklaracija ne mora nužno sadržavati potpune informacije o sastavu strukture, već samo najavu tipa strukture i njenog imena, što je dovoljno za korišćenje u programu.

Deklaracija strukture podrazumijeva definisanje strukture i njenih elemenata. Struktura se mora deklarisati pre nego što se koristi u programu.

Pristup elementima strukture vrši se pomoću operatora tačke (`.`) (*slika 16.*).

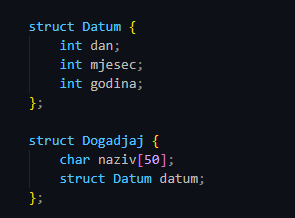
radnika 2: %d\n", radnik2.godine);



*slika 16*

## Ugnježdenje Struktura

U jeziku C, ugnježdenje struktura omogućava programerima da strukture stavljaju unutar drugih struktura (*slika 17.*) kako bi se predstavile složenije strukture podataka. Ovaj koncept omogućava organizaciju podataka na način koji odražava njihovu prirodnu hijerarhiju i pomaže u kreiranju efikasnih i preglednih programa.



*slika 17.*

## Prednosti i mane korištenja struktura

### Prednosti

* **Organizacija podataka**: Strukture omogućavaju organizaciju različitih tipova podataka pod jednim imenom, što olakšava grupisanje srodnih informacija.
* **Modularnost**: Korišćenje struktura omogućava modularni pristup programiranju, gde se kompleksni podaci mogu podeliti na manje, lako upravljive delove.
* **Čitljivost koda**: Strukture poboljšavaju čitljivost koda jer omogućavaju jasno definisanje i grupisanje podataka. Imenovanjem struktura i polja, program postaje lakši za razumevanje.
* **Ugnježdenje**: Ugnježdenje struktura omogućava modeliranje složenih odnosa između podataka i hijerarhijsko organizovanje informacija.
* **Prilagodljivost**: Strukture pružaju fleksibilnost u rukovanju podacima različitih tipova, što čini programski kod prilagodljivim promenama u zahtevima.

### Mane

* **Memorijski prostor**: Strukture mogu zauzimati više memorijskog prostora, posebno ako sadrže veliki broj ili kompleksne podatke. Ovo može uticati na efikasnost programa.
* **Kompleksnost koda**: Korišćenje previše struktura ili duboko ugnježdenih struktura može dovesti do kompleksnosti koda, što otežava održavanje i razumevanje.
* **Performanse**: Rad s velikim strukturama ili s velikim brojem ugnježdenih struktura može uticati na performanse programa, pogotovo ako se podaci često manipulišu.
* **Pristup poljima**: Pristupanje poljima unutar struktura zahteva upotrebu tačka operatora, a neki programeri mogu smatrati da to povećava količinu koda ili otežava rad.

Uprkos ovim potencijalnim manama, pravilno korištenje struktura može poboljšati organizaciju koda i olakšati rad s kompleksnim podacima u jeziku C. Važno je pažljivo balansirati između modularnosti i čitljivosti koda, posebno kada se koriste u složenim projektima.

# Pokazivači u jeziku C

Rad sa pokazivačima u jeziku C predstavlja moćan aspekt programiranja koji omogućava manipulaciju i rad sa memorijskim adresama. Pokazivač je promjenljiva koja sadrži memorijsku adresu druge promjenljive. Ovo omogućava direktnu manipulaciju vrednostima i pristupanje memorijskim lokacijama.

Deklaracija pokazivača (*slika 18.*) obuhvata određivanje tipa podataka na koji pokazuje pokazivač.



*slika 18.*

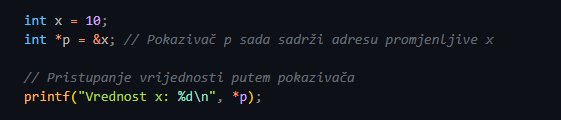
Jedna od ključnih prednosti pokazivača je mogućnost dinamičkog upravljanja memorijom. Programer može dinamički alocirati memoriju tokom izvršavanja programa i osloboditi je kada više nije potrebna. Ovo je posebno korisno u situacijama gde nije poznato koliko memorije će biti potrebno tokom kompilacije.

Deklarisanje pokazivača (*slika 18.*) obuhvata određivanje tipa podataka na koji pokazuje pokazivač.

U ovoj deklaraciji, `pokazivac` je pokazivač na cijelobrojni tip podataka (`int`). Zvjezdica (\*) se koristi kao dio sintakse kako bi označila da se radi o pokazivaču. Nakon deklaracije, `pokazivac` može biti postavljen tako da sadrži adresu neke promjenljive tipa `int`, a zatim se može koristiti za pristup vrednosti na toj memorijskoj lokaciji.

Kroz pravilno razumevanje i korišćenje pokazivača, programeri mogu optimizovati performanse koda, efikasno upravljati resursima i implementirati napredne strukture podataka. Međutim, važno je koristiti pokazivače pažljivo, kako bi se izbegle greške u radu sa memorijom koje mogu dovesti do nepredvidivog ponašanja programa.

Pristupanje vrijednostima (*slika 18.*) putem pokazivača omogućava programeru direktnu manipulaciju podacima na određenoj memorijskoj lokaciji.



*slika 18.*

U programiranju na jeziku C, dinamička alokacija memorije i oslobađanje memorije su ključne operacije koje omogućavaju efikasno upravljanje resursima. Ove operacije koriste funkcije poput `malloc`, `calloc`, „realloc“ za alokaciju memorije i funkciju `free` za oslobađanje memorije. Evo nekoliko informacija o ovim operacijama:

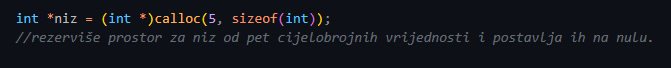
Dinamička alokacija memorije omogućava programu da rezerviše određeni blok memorije tokom izvršavanja programa. Ovo je posebno korisno kada je potrebno raditi s promenljivim količinama podataka ili kada veličina podataka nije poznata pri kompilaciji.

Neke od funkcija za alokaciju memorije su:

* **malloc**: funkcija rezerviše određeni broj bajtova memorije.



* **calloc**: rezerviše prostor za niz i postavlja sve elemente na nulu.



* **realloc**: funkcija se koristi za promjenu veličine već alociranog bloka memorije.



Važno je osloboditi memoriju koja je dinamički alocirana nakon što više nije potrebna. To se postiže upotrebom funkcije `free`.



Ova linija koda oslobađa memoriju koja je bila alocirana za niz. Važno je napomenuti da nakon oslobađanja memorije, pokazivači koji su ukazivali na tu memoriju postaju nevažeći.

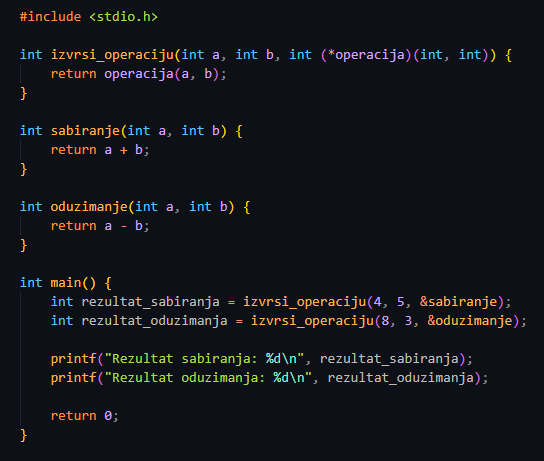
Pravilno upravljanje dinamičkom alokacijom i oslobađanjem memorije ključno je za prevenciju curenja memorije i održavanje efikasnog koda. Ovo je posebno važno u situacijama gde program radi sa promjenljivom ili nepredvidivom količinom podataka.

## Pokazivači na Funkcije

U jeziku C, pokazivači na funkcije omogućavaju programerima da efikasno rukuju funkcijama, uključujući prenos funkcija kao argumenata drugim funkcijama. Ovo dodaje dinamičnost i fleksibilnost u dizajnu programa.

Jedna od ključnih prednosti pokazivača na funkcije je mogućnost dinamičkog izbora funkcija koje će se izvršavati u zavisnosti od uslova ili potreba programa tokom izvođenja. Ovo omogućava prilagodljivost programa na promenljive zahteve, poboljšavajući čitljivost koda i olakšavajući održavanje.

Pokazivači na funkcije često se primenjuju pri implementaciji različitih dizajn obrazaca, uključujući koncepte callback funkcija. U ovom scenariju, funkcija se prenosi kao argument drugoj funkciji, omogućavajući njen poziv u određenim uslovima ili događajima. Ovaj pristup često se koristi u situacijama gde je potrebna modularnost i prilagodljivost koda.



## Prednosti i pravilno korištenje pokazivača

### Prednosti

* **Direktni pristup Memorijskim Lokacijama**: Pokazivači omogućavaju direktni pristup memorijskim lokacijama, što može biti korisno za efikasno manipulisanje podacima i optimizaciju performansi.
* **Efikasno Prolazak Podataka**: Korišćenjem pokazivača, možemo efikasno prosleđivati adrese podataka umesto samih podataka, što smanjuje potrebu za kopiranjem velikih blokova memorije.
* **Dinamička Alokacija Memorije**: Pokazivači su ključni za dinamičku alokaciju memorije, omogućavajući programima da rezervišu i oslobađaju memoriju tokom izvršavanja.
* **Funkcije Višeg Reda**: Omogućavaju implementaciju funkcija višeg reda, gde se funkcije mogu prosleđivati kao argumenti drugim funkcijama, doprinoseći modularnosti i fleksibilnosti koda.
* **Rad sa Niskim Nivoima Adresa**: Korisni su prilikom rada sa niskim nivoima sistema, kao što je rad sa hardverskim resursima ili interakcija sa sistemskim pozivima.

### Pravilno korištenje

* **Rad sa Strukturama Podataka**: Koristite pokazivače kada radite sa složenim strukturama podataka, omogućavajući efikasno rukovanje podacima.
* **Dinamička Alokacija i Oslobađanje Memorije**: Pravilno koristite pokazivače kada je potrebno dinamički upravljati memorijom, kao što je slučaj sa alociranjem i oslobađanjem memorije.
* **Rad sa Niskim Nivoima i Hardverom**: Koristite pokazivače za interakciju sa niskim nivoom sistema, rad sa hardverom ili optimizaciju performansi.
* **Implementacija Algoritama i Struktura Podataka**: Primenjujte pokazivače prilikom implementacije različitih algoritama, kao i struktura podataka kao što su povezane liste, stabla, i slično.
* **Funkcije Višeg Reda**: Upotrebljavajte pokazivače kada želite implementirati funkcije višeg reda, gde funkcije mogu biti prosleđivane kao argumenti drugim funkcijama.

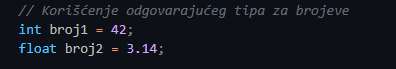
Pravilna upotreba pokazivača zahteva pažljivo planiranje i razumevanje kako rade. Korišćenjem pokazivača tamo gde donose stvarnu korist, programeri mogu unaprediti efikasnost i fleksibilnost svog koda. Međutim, trebali bi biti svesni potencijalnih rizika i voditi računa o bezbednosti i održivosti koda.

# Efikasnost i optimizacija koda

Optimizacija koda predstavlja ključni aspekt prilikom razvoja softvera, posebno u situacijama gde se zahteva visok nivo performansi ili efikasnost resursa. Razumijevanje pravilnog izbora tipova podataka, struktura i strategija optimizacije ključno je za postizanje ciljeva efikasnosti u jeziku C.

## Pravilan izbor tipova podataka

Prvi korak ka optimizaciji koda je pravilan izbor tipova podataka. Ovo uključuje razmatranje veličine podataka, opsega vrijednosti koje tipovi podržavaju, i troškova pristupa i manipulacije podacima.



Koristeći odgovarajuće tipove podataka, izbegava se nepotrebno trošenje memorije i ubrzava pristup podacima.

## Pravilna Upotreba Struktura

Struktura programa i organizacija podataka imaju ključnu ulogu u razvoju efikasnog i održivog koda. Pravilna upotreba struktura može biti od suštinskog značaja za postizanje optimalne performanse i lakše održavanje projekta.

**Definisanje Struktura sa Razumevanjem Podataka**

Prilikom projektovanja struktura, bitno je imati duboko razumevanje podataka koje struktura predstavlja. Jasnije razumevanje podataka omogućava bolje planiranje i organizaciju struktura, čime se smanjuje kompleksnost i poboljšava čitljivost koda.

**Grupisanje Sličnih Podataka**:

Strukture se često koriste za grupisanje sličnih podataka, što pomaže organizaciju i manipulaciju podacima na logičan način. Na primer, ako imamo podatke o studentima, možemo kreirati strukturu koja grupiše relevantne informacije o studentu.

**Unapređenje Efikasnosti Pomoću Struktura**:

Pravilno projektovane strukture mogu unaprediti efikasnost koda. Na primer, pristupanje elementima strukture može biti brže i jednostavnije od pojedinačnog rukovanja različitim promenljivama.

**Povezivanje Struktura za Kompleksne Podatke**:

U složenijim projektima, strukture se mogu povezivati kako bi se predstavili kompleksni odnosi među podacima. Ovo omogućava organizaciju podataka na hijerarhijski način.

**Održavanje Čitljivosti Koda**:

Pravilno projektovane strukture doprinose čitljivosti koda. Dati smisleno ime strukturama i poljima pomaže drugim programerima (uključujući i vas samog u budućnosti) da lakše razumeju svrhu i organizaciju podataka u programu.

Pravilna upotreba struktura zahteva pažljivo planiranje i razmatranje potreba projekta. Kroz korišćenje struktura na efikasan način, programeri mogu olakšati rad, poboljšati performanse i povećati održivost svog koda.

## Strategije za Optimizaciju Koda

* **Pristup Memoriji**: Optimizacija pristupa memoriji može se postići kroz smanjenje broja čitanja i pisanja, korišćenje lokalnih promenljivih, i smanjenje broja alokacija i dealokacija memorije.
* **Algoritamske Optimizacije**: Razumevanje efikasnih algoritama i njihova implementacija često ima značajan uticaj na performanse programa.
* **Paralelizacija Koda**: U situacijama gde je to moguće, distribuiranje zadatka između više niti ili procesa može ubrzati izvršavanje programa.
* **Profiliranje koda**: Korišćenje alata za profiliranje omogućava programerima identifikaciju tačaka u kodu koje uzimaju najviše vremena izvršavanja. Na osnovu ovih informacija, mogu se usmeriti napori ka ključnim delovima koda koji zahtevaju optimizaciju.
* **Upotreba inline optimizacija** uključujući inline funkcije, mogu smanjiti overhead poziva funkcija i ubrzati izvršavanje koda.

Efikasnost i optimizacija koda su vitalni za postizanje visokih performansi u jeziku C. Pravilan izbor tipova podataka, struktura, i primena strategija optimizacije ključni su za postizanje željenih rezultata. Razvijanje ove veštine pomaže programerima da efikasno iskoriste resurse i postignu optimalne performanse u svojim aplikacijama. Naredne stranice će se fokusirati na naprednije tehnike i konkretne primene ovih strategija.

# Zaključak

U svetu programiranja, gde brzina izvršavanja i efikasnost igraju ključnu ulogu, pravilno predstavljanje podataka u jeziku C postaje imperativ. Bitovske operacije pružaju snažne mehanizme za manipulaciju podacima na najnižem nivou, a struktura programa ima značajan uticaj na performanse i čitljivost koda.

Bitovske operacije, poput I, ILI, ekskluzivnog ILI i negacije, omogućavaju programerima preciznu kontrolu nad podacima, često ključnu za optimizaciju algoritama. Razumevanje kako pravilno koristiti ove operacije pridonosi stvaranju efikasnog i brzog koda.

Organizacija podataka, kroz strukture i pravilan izbor tipova podataka, takođe je od suštinske važnosti. Pravilna struktura ne samo da smanjuje potrošnju memorije već i olakšava pristup podacima, doprinoseći bržem izvršavanju programa.

Važno je naglasiti povezanost između viših programskih jezika i paradigmi. Razumevanje kako se određene paradigme prevode u jezik C omogućava programerima fleksibilnost u izboru najefikasnijeg pristupa problemu.

Dinamičnost programiranja očituje se i u nastanku novih paradigmi. Sposobnost prilagođavanja i usvajanja novih tehnologija i pristupa postaje ključna za dugoročni uspeh programera.

Ključno je održavati balans između postizanja visokih performansi i očuvanja čitljivosti koda. Programi koji su brzi, ali nečitljivi, mogu postati teško održivi i podložni greškama.

U zaključku, lekcije o predstavljanju podataka u jeziku C pružile su duboko razumevanje ključnih principa. Naučene veštine ne samo da doprinose efikasnom kodiranju u jeziku C već i stvaraju temelj za dalje usavršavanje u dinamičnom svetu programiranja. Održavanje fokusa na efikasnosti, pravilnom organizacijom podataka i adaptacijom novim trendovima omogućava programerima da ostvare izvanredne rezultate u svetu programiranja.