

PANEVROPSKI UNIVERZITET APEIRON
FAKULTET INFORMACIONIH TEHNOLOGIJA

Vanredne studije

Smjer „Programiranje i softversko inženjerstvo”

Predmet

PRINCIPI PROGRAMIRANJA

„PREDSTAVLJANJE PODATAKA U JEZIKU C”

(seminarski rad)

Predmetni nastavnik
Prof.dr Branimir Trenkić

Student

Mirza Abdijanović

Index br. 93-22/FITIIT1-pro-180

1.	Uvod	2
2.	Tipovi podataka u jeziku C	3
2.1.	Cijelobrojni ,realni i znakovni tipovi podataka:	3
2.2.	Složeni tipovi:	3
2.3.	Veličina i raspon tipova podataka:	3
2.4.	Specifikatori tipova:	4
3.	Deklaracija i inicijalizacija promenljivih u jeziku C	4
3.1.	Deklaracija promenljivih:	5
3.2.	Inicijalizacija promenljivih	5
3.3.	Lokalne i globalne promenljive	5
3.4.	Konstante i modifikatori tipova	6
4.	Operatori i izrazi u jeziku C	6
4.1.	Aritmetički operatori	6
4.2.	Logički operatori	7
4.3.	Relacioni operatori	7
4.4.	Prioritet i asocijativnost operatora	7
5.	Korišćenje konstanti u jeziku C	8
5.1.	Vrste konstanti	8
5.2.	#define direktiva	9
5.3.	Prednosti korišćenja konstanti	9
6.	Rad sa nizovima u jeziku C	10
6.1.	Deklaracija niza	10
6.2.	Višedimenzionalni nizovi	11
6.3.	Stringovi kao nizovi karaktera	11
6.4.	Prednosti i izazovi rada sa nizovima	11
7.	Rad sa strukturama u jeziku C	12
7.1.	Definicija i deklaracija strukture	12
7.2.	Ugnježđenje struktura	13
7.3.	Prednosti i mane korišćenja struktura	13
8.	Pokazivači u jeziku C	14
8.1.	Pokazivači na funkcije	16
8.2.	Prednosti i pravilno korišćenje pokazivača	17
9.	Efikasnost i optimizacija koda	18
9.1.	Pravilan izbor tipova podataka	18
9.2.	Pravilna upotreba struktura	18
9.3.	Strategije za optimizaciju koda	19
10.	Zaključak	20
11.	Literatura	21

1. Uvod

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

Efikasno upravljanje podacima je temelj optimizovanog i pouzdanog softvera. Kako bi se postigla optimalna izvršavanja, programeri moraju pažljivo razmatrati način na koji podaci izgledaju u memoriji, kako su organizovani i kako im se pristupa. 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.

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.

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

2. Tipovi podataka u jeziku C

Tipovi podataka u jeziku C igraju ključnu ulogu u definisanju i organizaciji informacija koje programi obrađuju. Njihova precizna specifikacija omogućava efikasno upravljanje memorijom i olakšava implementaciju algoritama, čineći jezik C snažnim i fleksibilnim alatom za programiranje.

2.1. 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 zarezmom. 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.

2.2. Složeni tipovi:

Složeni tipovi podataka omogućavaju programerima da grupišu više sličnih podataka pod jednim imenom. Ovde spadaju strukture (`struct`), unije (`union`) i nabiranja (`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.

2.3. Veličina i raspon tipova podataka:

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

a) `char`

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

b) `int`

integer (`int`) ima veličinu od 4 bajta, a opseg vrijednosti koje može zadržati je od -2.147.483.648 do 2.147.483.647.

c) float

Float tip podataka zauzima 4 bajta i može predstavljati vrijednosti u opsegu približno od $\pm 1.2 \times 10^{-38}$ do $\pm 3.4 \times 10^{38}$, sa oko 6 decimala preciznosti.

d) double

Double, obično veličine 8 bajta, ima opseg vrijednosti približno od $\pm 2.3 \times 10^{-308}$ do $\pm 1.7 \times 10^{308}$, sa oko 15 decimala preciznosti.

e) short

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

f) long

Long tip podataka, sa veličinom od obično 4 ili 8 bajta, može zadržavati vrijednosti 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).

g) unsigned

Unsigned modifikator može se koristiti sa gotovo svim tipovima podataka i označava da tip ne može imati negativne vrijednosti. Na primer, unsigned int ima opseg vrijednosti od 0 do 4.29.967.295 umjesto da ima negativne vrijednosti.

2.4. 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.

3. Deklaracija i inicijalizacija promenljivih u jeziku C

Deklaracija i inicijalizacija promenljivih predstavljaju ključne korake u postavljanju i pripremi podataka u jeziku C. Ovi osnovni koncepti omogućavaju programerima da definišu i dodeljuju vrijednosti promenljivama, što je esencijalno za efikasno izvršavanje programa u jeziku C.

3.1. Deklaracija promjenljivih:

Deklaracija promjenljivih u jeziku C predstavlja proces informisanja kompajlera o tipu podataka i imenu promjenljive koja će se koristiti u programu (*slika 1.*). Pravilna deklaracija je ključna za definisanje promjenljivih prije nego što se koriste u programu.

```
int broj;           // Deklaracija cijelobrojne promjenljive
float rezultat;     // Deklaracija realne promjenljive
char znak;          // Deklaracija znakovne promjenljive
```

slika 1.

3.2. Inicijalizacija promjenljivih

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

```
int broj           = 2024; // Inicijalizacija cijelobrojne promjenljive
float rezultat     = 3.14; // Inicijalizacija realne promjenljive
char znak          = 'A';  // Inicijalizacija znakovne promjenljive
```

slika 2.

Pravilna deklaracija i inicijalizacija ključne su za sprječavanje 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.

3.3. 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.

3.4. Konstante i modifikatori tipova

Pored standardnih promjenljivih, u jeziku C imamo i konstante koje se deklariraju 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 obezbjeđuje 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.

4. 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.

4.1. Aritmetički operatori

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

```
int a = 10, b = 5;
int suma = a + b;      // Sabiranje
int razlika = a - b;    // Oduzimanje
int proizvod = a * b;   // Množenje
int kolicnik = a / b;   // Dijeljenje
int ostatak = a % b;    // Ostatak pri dijeljenju
```

slika 3.

4.2. 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.

```
int x = 5, y = 10;
int rezultat_i = (x > 0) && (y < 20); // Logički "i"
int rezultat_ili = (x > 0) || (y < 20); // Logički "ili"
int rezultat_negacija = !(x > 0); // Logička negacija
```

slika 4.

4.3. Relacioni operatori

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

```
int p = 5, q = 8;
int rezultat_jednako = (p == q); // Da li je jednako? Vrijednost = 0
int rezultat_razlicito = (p != q); // Da li je različito? Vrijednost = 1
int rezultat_manje_od = (p < q); // Da li je manje od? Vrijednost = 1
int rezultat_vece_od = (p > q); // Da li je veće od? Vrijednost = 0
int rezultat_manje_ili_jednako = (p <= q); // Da li je manje ili jednako? Vrijednost = 1
int rezultat_vece_ili_jednako = (p >= q); // Da li je veće ili jednako? Vrijednost = 0
```

slika 5.

4.4. Prioritet i asocijativnost operatora

Prioritet i asocijativnost operatora su veoma bitan dio tačnog tumačenja 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.

5. Korištenje konstanti u jeziku C

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

5.1. Vrste konstanti

5.1.1. Cijelobrojne konstante

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

```
const int broj = 42;           // Deklaracija cijelobrojne konstante
const int oktalni_broj = 052;  // Oktalna reprezentacija
const int heks_broj = 0x2A;    // Heksadecimalna reprezentacija
```

slika 6.

5.1.2. Realne konstante

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

```
const float pi = 3.14159;      // Deklaracija realne konstante
const float e = 2.71828e-5;    // Eksponencijalna reprezentacija
```

slika 7

5.1.3. Znakovne konstante

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

```
const char slovo = 'A'; // Deklaracija znakovne konstante
```

slika 8.

5.1.4. Nizovne konstante

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

```
const char ime[] = "Programiranje"; // Deklaracija nizovne konstante
```

slika 8.

5.2. #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.

```
#define PI 3.14159 // Definisanje konstante korištenjem #define
```

slika 9.

5.3. 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.

6. 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.

6.1. Deklaracija niza

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

```
int brojevi[5]; // Deklaracija cijelobrojnog niza sa 5 elemenata
```

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.

```
brojevi[0] = 10; // Pristup prvom elementu niza  
brojevi[2] = 20; // Pristup trećem elementu niza
```

slika 11.

```
int velicina = sizeof(brojevi) / sizeof(brojevi[0]);
```

slika 12.

6.2. Višedimenzionalni nizovi

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

```
int matrica[3][3]; // Deklaracija 2D niza (matrice)
```

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.*).

```
matrica[1][2] = 42; // Postavljanje vrednosti elementa
```

slika 14.

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

6.3. Stringovi kao nizovi karaktera

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

```
char naziv_univerziteta[] = "Apeiron";
```

slika 15.

U ovom slučaju, string "Apeiron" čuva se kao niz karaktera u promjenljivoj `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.

6.4. Prednosti i izazovi rada 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 dio svakodnevnog programiranja u jeziku C. Pažnja na granice i pažljiva manipulacija nizovima ključni su za siguran i efikasan kod.

7. 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.

7.1. Definicija i deklaracija strukture

Definicija strukture u jeziku C (*slika 16.*) 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 16.*) 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 korištenje u programu.

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

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

```
#include <stdio.h>

// Definicija strukture
struct Osoba {
    char ime[30];
    int godine;
    float visina;
};

int main() {
    // Deklaracija i inicijalizacija instance strukture
    struct Osoba osoba1 = {"Ana", 25, 1.75};

    // Pristupanje poljima strukture
    printf("Ime: %s\n", osoba1.ime);
    printf("Godine: %d\n", osoba1.godine);
    printf("Visina: %.2f\n", osoba1.visina);

    return 0;
}
```

slika 16

7.2. Ugnježđenje struktura

U jeziku C, ugnježđenje 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.

```
struct Datum {  
    int dan;  
    int mjesec;  
    int godina;  
};  
  
struct Dogadjaj {  
    char naziv[50];  
    struct Datum datum;  
};
```

slika 17.

7.3. Prednosti i mane korištenja struktura

Prednosti

- **Organizacija podataka:** strukture omogućavaju organizaciju različitih tipova podataka pod jednim imenom, što olakšava grupisanje srodnih informacija.
- **Modularnost:** korištenje struktura omogućava modularni pristup programiranju, gdje se kompleksni podaci mogu podijeliti na manje dijelove koji su lakši za upravljanje.
- **Čitljivost koda:** strukture poboljšavaju čitljivost koda jer omogućavaju jasno definisanje i grupisanje podataka. Imenovanjem struktura i polja, program postaje lakši za razumijevanje.
- **Ugnježđenje:** ugnježđenje 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 promjenama u zahtjevima.

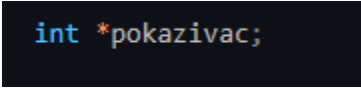
Mane

- **Memorijski prostor:** strukture mogu zauzimati više memorijskog prostora, posebno ako sadrže veliki broj podataka ili kompleksne podatke. Ovo može uticati na efikasnost programa.
- **Kompleksnost koda:** korištenje previše struktura ili duboko ugnježenih struktura može dovesti do kompleksnosti koda, što otežava održavanje i razumevanje.
- **Performanse:** rad s velikim strukturama ili s velikim brojem ugnježenih struktura može uticati na performanse programa, pogotovo ako se podaci često manipulišu.

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.

8. 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 vrijednostima i pristupanje memorijskim lokacijama. Deklaracija pokazivača (*slika 18.*) obuhvata određivanje tipa podataka na koji pokazuje pokazivač.



```
int *pokazivac;
```

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.

U ovoj deklaraciji (*slika 18.*), `pokazivac` je pokazivač na cijelobrojni tip podataka (`int`). Zvezdica (*) 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 vrijednosti na toj memorijskoj lokaciji.

Kroz pravilno razumijevanje i korištenje 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 19.) putem pokazivača omogućava programeru direktnu manipulaciju podacima na određenoj memorijskoj lokaciji.

```
int x = 10;
int *p = &x; // Pokazivač p sada sadrži adresu promjenljive x

// Pristupanje vrijednosti putem pokazivača
printf("Vrijednost x: %d\n", *p);
```

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 promjenljivim količinama podataka ili kada veličina podataka nije poznata pri kompilaciji.

Neke od funkcija za alokaciju memorije su:

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

```
int *niz = (int *)malloc(5 * sizeof(int)); //rezerviše prostor za niz od pet cijelobrojnih vrijednosti.
```

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

```
int *niz = (int *)calloc(5, sizeof(int));
//rezerviše prostor za niz od pet cijelobrojnih vrijednosti i postavlja ih na nulu.
```

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

```
niz = (int *)realloc(niz, 10 * sizeof(int));
//Ovde se proširuje niz na deset elemenata.
```

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

```
free(niz);
```

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 gdje program radi sa promjenljivom ili nepredvidivom količinom podataka.

8.1. 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 promjenljive zahtjeve, poboljšavajući čitljivost koda i olakšavajući održavanje.

Pokazivači na funkcije često se primjenjuju 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 gdje je potrebna modularnost i prilagodljivost koda.

```
#include <stdio.h>

int izvrsi_operaciju(int a, int b, int (*operacija)(int, int)) {
    return operacija(a, b);
}

int sabiranje(int a, int b) {
    return a + b;
}

int oduzimanje(int a, int b) {
    return a - b;
}

int main() {
    int rezultat_sabiranja = izvrsi_operaciju(4, 5, &sabiranje);
    int rezultat_oduzimanja = izvrsi_operaciju(8, 3, &oduzimanje);

    printf("Rezultat sabiranja: %d\n", rezultat_sabiranja);
    printf("Rezultat oduzimanja: %d\n", rezultat_oduzimanja);

    return 0;
}
```

8.2. 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.
- **Efikasni prolazak podataka:** korištenjem pokazivača, možemo efikasno prosljeđivati adrese podataka umjesto 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 koda.
- **Funkcije višeg reda:** omogućavaju implementaciju funkcija višeg reda, gde se funkcije mogu prosljeđivati 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.
- **Implementacija algoritama i struktura podataka:** primjenjujte 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 prosljeđene kao argumenti drugim funkcijama.

Pravilna upotreba pokazivača zahtjeva pažljivo planiranje i razumijevanje kako rade. Korištenjem pokazivača tamo gdje donose stvarnu korist, programeri mogu unaprijediti efikasnost i fleksibilnost svog koda. Međutim, trebali bi biti svjesni potencijalnih rizika i voditi računa o bezbjednosti i održivosti koda.

9. Efikasnost i optimizacija koda

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

9.1. 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.

```
// Korišćenje odgovarajućeg tipa za brojeve  
int broj1 = 42;  
float broj2 = 3.14;
```

Korištenjem odgovarajuće tipove podataka, izbjegava se nepotrebno trošenje memorije i ubrzava pristup podacima.

9.2. 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 razumijevanjem 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 primjer, ako imamo podatke o studentima, možemo kreirati strukturu koja grupiše relevantne informacije o studentu.

Unapređenje efikasnosti pomoću struktura:

Pravilno projektovane strukture mogu unaprijediti efikasnost koda. Na primjer, pristupanje elementima strukture može biti brže i jednostavnije od pojedinačnog rukovanja različitim promjenljivama.

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.

9.3. Strategije za optimizaciju koda

- **Pristup memoriji:** optimizacija pristupa memoriji može se postići kroz smanjenje broja čitanja i pisanja, korištenje lokalnih promjenljivih, i smanjenje broja alokacija i dealokacija memorije.
- **Profiliranje koda:** korištenje alata za profiliranje omogućava programerima identifikaciju tačaka u kodu koje uzimaju najviše vremena izvršavanja. Na osnovu ovih informacija, mogu se usmjeriti naponi ka ključnim dijelovima 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 vještine pomaže programerima da efikasno iskoriste resurse i postignu optimalne performanse u svojim aplikacijama.

10. Zaključak

U zaključku ovog seminarskog rada, duboko sam istražili važnost predstavljanja podataka u jeziku C. Tipovi podataka igraju ključnu ulogu u definisanju i organizaciji informacija unutar programa, pružajući osnovu za efikasno izvršavanje i manipulaciju podacima.

U praksi, tipovi podataka postaju alatke programerima, omogućavajući im da prilagode strukturu podataka prema specifičnostima zadatka. Raznolikost tipova pruža fleksibilnost, a njihova efikasna primena postaje ključna za optimizaciju programa.

Seminarski rad je obuhvatio različite tipove podataka u jeziku C, uključujući cijelobrojne, realne, znakovne i složene tipove. Kroz analizu ovih tipova, stekli smo razumijevanje njihove svrhe, prednosti i specifičnosti primjene. Razmatranje procesa deklaracije i inicijalizacije promjenljivih dodatno doprinosi razumijevanju kako se podaci predstavljaju i koriste u praksi.

Ovaj rad istakao je važnost pažljivog odabira tipova podataka u skladu sa zahtjevima projekta i efikasnosti resursa. Upravljanje podacima u jeziku C zahtjeva balans između optimalne memorije i brzine izvršavanja, što postavlja programere pred izazov izbora pravilnih tipova podataka.

11. Literatura

<https://www.geeksforgeeks.org/data-types-in-c/>

<https://www.geeksforgeeks.org/c-programming-language/?ref=lbp>

<https://www.w3schools.com/c/index.php>

<https://www.programiz.com/c-programming/c-data-types>

<https://www.freecodecamp.org/news/the-c-programming-handbook-for-beginners/>

<https://www.simplilearn.com/c-programming-article>