Fakultet tehničkih nauka, DRA, Novi Sad

Predmet:
Organizacija podataka

Dr Slavica Kordić
Vladimir Ivković
Nikola Todorović
Marko Vještica

OSNOVNI POJMOVI

Šta su varijable?

- Imenovani prostor u memoriji računara, koji je namenjen čuvanju vrednosti određenog tipa (celobrojne, realne, karakteri, itd).
- Sadrže podatke koje koristi vaš program
- Koriste se za čuvanje podataka koji se koriste u toku izvršavanja programa

Deklarisanje varijabli u C-u

- Pre korišćenja varijabla mora biti deklarisana.
- Deklaracija se vrši navođenjem tipa i naziva varijable.
- Opciono, mogu se pomoću kvalifikatora dodatno specificirati karakteristike varijable.
- VAŽNO: Kada se izvrši deklaracija varijable, njena **vrednost je nedefinisana**.

Primer deklaracije varijabli

- int i;
- char c;
- float f1, f2;
- float f1=7.0, f2 = 5.2;
- unsigned int ui = 0;
  (u poslednja dva slučaja izvršena je i inicijalizacija na početnu vrednost)

Principi imenovanja varijabli

- Za nazive varijabli se mogu koristiti slova, brojevi i znak „_”
  - CSE_5a
  - vrlo_dugo_ime_promenljive (nepraktično)
  - brojac (opisno ime – vrlo praktično, kod čitljiviji)
- Prvi karakter u nazivu promenljive ne sme biti cifra
  - 5a_CSE nije validan naziv promenljive!
- C pravi razliku između malih i velih slova
  - Naziv nije ista promenljiva kao i naziv

Tipovi podataka u C-u

- **char** – jednobajtni znakovni tip.
- **short int** (ili samo short) – celobrojni tip, obično dužine 2 bajta (ređe korišćen tip).
- **int** - celobrojni tip – obično dužine 4 bajta.
- **long int** (ili samo long) – celobrojni tip, dužina 4 ili 8 bajta (ređe korišćen tip).
- **float** – realna vrednost jednostruke preciznosti – obično 4 bajta.

Tipovi podataka u C-u

- **double** – realna vrednost dvostruke preciznosti – obično 8 bajta.
- **long double** - realna vrednost dvostruke preciznosti – obično 8 bajta (ređe korišćen tip).
- funkcija **sizeof()**
  - određuje veličinu memorije u bajtovima koja je potrebna za smeštanje podataka nekog tipa
- Signed nasuprot unsigned tipova

Scanf/Printf

- Štampanje vrednosti
  - **printf ()**
    - ispis na standardni izlaz
- Preuzimanje vrednosti
  - **scanf()**
    - čitanje podataka sa standardnog ulaza

Primer

Napisati program koji primenom operatora sizeof određuje veličinu memorije u bajtovima koja je potrebna za smeštanje podataka tipa: char, int, float, double.

Rešenje:

```
#include <stdio.h>
int main()
{
   printf("Velicina memorije (izrazena u bajtovima) iznosi:");
   printf("\n-za char \t %d", sizeof(char));
   printf("\n-za int \t %d", sizeof(int));
   printf("\n-za float \t %d", sizeof(float));
   printf("\n-za double \t %d", sizeof(double));
   return 0;
}
```

printf/scanf format specifikatori

- Konstrukcija %<format> u format stringu printf/scanf se zamenjuje sa vrednošću odgovarajuće varijable.
- %c – karakter konverzija
- %d – celobrojna konverzija,
- %u – neoznačena celobrojna konverzija

printf/scanf format specifikatori

- %f – konverzija realne vrednosti (float)
- %lf – konverzija realne vrednosti dvostruke preciznosti (double)
  %g – drugi specifikator formata za double (u printf)
- %% - karakter ‘%’ (u printf)
  – *escape* mehanizam

Primer

```
/* Učitava se temperatura u celzijusima i konvertuje u Kelvine*/
#include <stdio.h>

int main( )
{
    double celzijusi, kelvini;

    printf("Unesite temperaturu u celzijusima: ");
    scanf("%lf",&celzijusi);

    kelvini = celzijusi +273.15;
    printf("%lf stepeni celzijusa je  %lf stepeni kelvina\n", celzijusi, kelvini);

    return 0;
}
```

Predstavljanje brojeva u različitim brojnim sistemima

- Cifre brojnog sistema su: 0 do (baza – 1)
- Primer:
  - binarni (b=2): {0, 1}
  - oktalni (b=8): {0, 1, 2, 3, 4, 5, 6, 7}
  - decimalni (b=10): {0, 1, 2, 3,4 ,5 ,6 ,7 , 8, 9}
  - heksadecimalni (b=16): {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}

Primer

```
/* Aritmeticke operacije */
#include <stdio.h>
int main()
{
        int a = 5;
      int b = 3;

        printf("Zbir a+b je : %d\n”,a+b);

        printf("Razlika a-b je : %d\n",a-b);

        printf("Proizvod a*b je : %d\n",a*b);

        printf("Celobrojni kolicnik a/b je : %d\n", a/b);

        printf("Pogresan pokusaj racunanja realnog kolicnika a/b je : %f\n", a/b);

        printf("Realni kolicnik a/b je : %f\n", (float)a/(float)b);

        printf("Ostatak pri deljenju a/b je : %d\n", a%b);
        return 0;
}
```

Tip *char*

- Karakteri: ‘A’, ‘B’, ‘c’, ‘d’, ‘1’, ‘4’, ‘#’, ‘?’, ‘\n’
- char varijabla se koristi za čuvanje tekstualnih znakova:
  - slova,
  - cifara,
  - specijalnih znakova,
  - neštampajućih (belih) znakova,
  - ali može se koristiti i za čuvanje malih celobrojnih vrednosti (0 do 255 ili -128 do 127).

char je samo broj!

- Svakom karakteru se pridružuje numerički kod.
- Postoje različiti skupovi kodova:
  - ASCII (American Standard Code for Information Interchange) – najčešći.
  - EBCDIC – zastareo, danas se retko koristi.
  - Noviji skupovi karaktera (Unicode).
- Koristićemo ASCII.

Primer

/* koristenje char kao znakovnog tipa i za malu numericku vrednost */

```
#include<stdio.h>
int main()
{
    char znak;
   printf("Unesite znak: ");
   scanf("%c", &znak);
    printf("Znak kao karakter je: %c\n", znak);
    printf("Numericka vrednost znak-a je: %d\n", znak);
    printf("Karakter posle %c je %c\n", znak, znak+1);
    return 0;
}
```

Zadatak

Napisati C program koji računa sumu prvih *n* prirodnih brojeva, pri čemu se *n* zadaje na početku programa.

Upotrebom *while* ciklusa.

Zadatak

Realizovati množenje i deljenje pomoću sabiranja i oduzimanja

Zadatak

Sa standardnog ulaza učitati prirodne brojeve N1 i N2. Koristeći for ciklus, ispisati sve neparne brojeve od N1 do N2 na standardni izlaz

Zadatak

Sa standardnog ulaza učitati prirodne brojeve N i q. Koristeći while ispisati sve brojeve od 2 do N koji su deljivi sa q na standardni izlaz

NIZOVI

Nizovi - deklaracija

- Niz predstavlja kolekciju elemenata istog tipa.
- Primer deklaracije niza je:
  - **int niz[5];**

    /* niz od 5 elemenata tipa int*/
  - **float niz2d [10][10];**

    /* matrica od 10x10 elemenata tipa float*/

Pristupanje elementima niza

- Pristupanje elementima niza se ostvaruje na sledeći način:
- niz[0] = 4;
- niz[1] = 2 * niz[0]; /*niz[1] = 8*/
- niz[2] = niz[0] * niz[1]; /*niz[2] = 32*/
- niz[3] = 5;
- niz[4] = 7;
- a = niz[10];
- Napomena: Indeks niza ide od 0 (a ne 1!) do n-1

Inicijalizacija niza

- Primer inicijalizacije vrednosti elemenata niza

```
for ( i=0; i<5; i++)
        a[i] = 0;
```

Primer unosa elemenata

- Primer unosa vrednosti elemenata niza sa standardnog ulaza:

Unosimo 5 elemenata, index ide od 0 do 4

```
for ( i=0; i<5; i++)
        scanf("%d ", &a[i]);
```

Prikaz elemenata niza

- Štampanje elemenata niza na standardni izlaz.

```
for ( i=0; i<5; i++)
        printf("%d ", a[i]);
```

Primer 1

- Program ilustruje korišćenje statičkih nizova. Dat je niz od maksimalno 30 celobrojnih elemenata. Učitati n elemenata i ispisati ih po učitanom i obrnutom redosledu.

```c
#include <stdio.h>
#define MAX_SIZE 30

int main()
{
    int a[MAX_SIZE];
    int i, n;

    do {
        printf("Unesite broj elemenata niza (maksimalno %d): ", MAX_SIZE);
        scanf("%d", &n);
    } while ( n <= 0 || n>MAX_SIZE);

    for (i = 0; i<MAX_SIZE; i++) {
        a[i] = 0;
    }

    for (i = 0; i<n; i++) {
        printf("a[%d]=",i);
        scanf("%d",&a[i]);
    }

    printf("\n Elementi niza po ucitanom redosledu: \n");

    for (i = 0; i<n; i++) {
        printf("a[%d]=%d\n", i, a[i]);
    }

    printf("\n Obrnuti redosled: \n");

    for (i = n-1; i>=0; i--) {
        printf("a[%d]=%d\n", i, a[i]);
    }

    return 0;
}
```

Zadatak

- Dat je niz od maksimalno 20 realnih elemenata. Učitati n elemenata, naći maksimalnu vrednost, a zatim sortirati niz u rastućem redosledu.

POKAZIVAČI

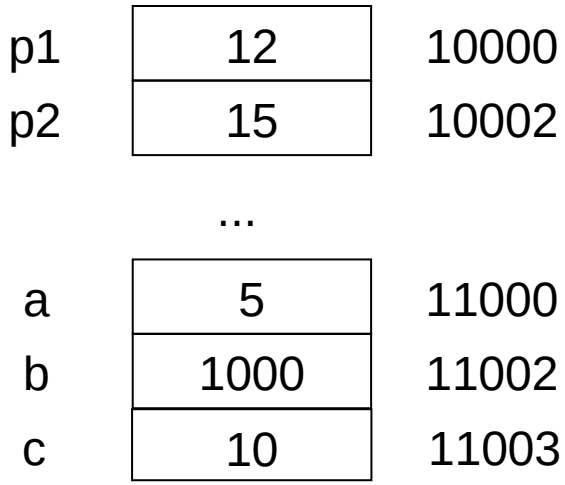
Pokazivači

- Pokazivač je adresa u memoriji, broj.
  - ukazuje na lokaciju u memoriji
  - sadržaj pokazivačke promenljive je adresa (lokacija u memoriji).
- Pokazivačke promenljive pokazuju na druge promenljive ili na početak memorijskog bloka
  - pokazivači na promenljive tipa int, float, itd.

Pokazivači

- Deklaracija:

```
int *p1, *p2;
int a = 5;
int b, c;
```

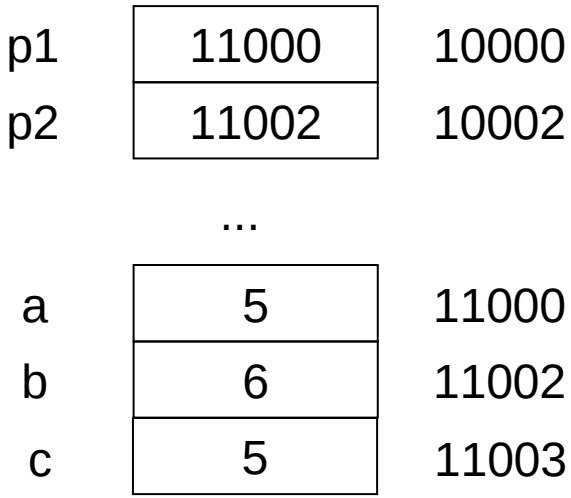


- Dodela vrednosti:

```
p1 = &a;
p2 = &b;
```

- Pristup lokaciji:

```
c = *p1;  // c <- a
*p2 = 6;  // b <- 6
```



Referenciranje

- **Unarni operator &** daje adresu promenljive
- Izraz **p=&a** dodeljuje adresu promenljive a promenljivoj p, pa sada p pokazuje na a
- Da bi se odštampala vrednost pokazivača koristi se **konverzija %p**.

Dereferenciranje

- Primenom **unarnog operatora *** može se posredno pristupiti nekom podatku pomoću memorijske adrese.

Specijalna konstanta NULL

- Konstanta koja se nalazi u `stdio.h`
- Ako pokazivačka promenljiva ima vrednost **NULL**, onda ne pokazuje ni na šta.
- Primer:

```
int *p;
p = NULL;
```

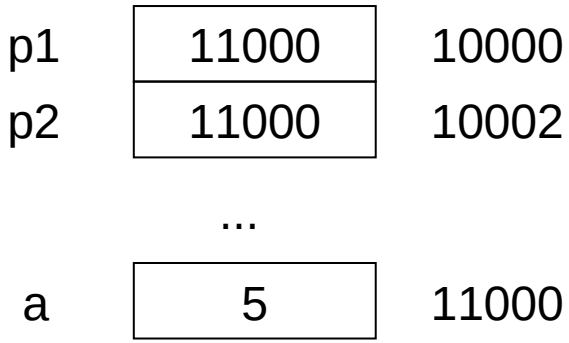
- **Neinicijalizovana vrednost NEMA NULL!**
  - **mora se eksplicitno inicijalizovati na NULL**

Operator dodele vrednosti

- Dodelom vrednosti dobijamo da oba pokazivača ukazuju na isto.

- Primer:

```
int a = 5;
int *p1, *p2;
p1 = &a;
p2 = p1;
```



Primer 2

```c
#include <stdio.h>

int main() {
    int i;
    int *pi;

    i = 7;
    pi = &i;

    printf("Promenljiva - adresa:\t %p, vrednost:\t %d\n\n", &i, i);
    printf("Pokazivac   - adresa:\t %p, vrednost:\t %p\n\n", &pi, pi);
    printf("Pokazivac   - vrednost:\t %p, sadrzaj:\t %d\n\n", pi, *pi);

    i = 10;

    printf("Pokazivac   - vrednost:\t %p, sadrzaj:\t %d\n\n", pi, *pi);

    (*pi)++;

    printf("Promenljiva - adresa:\t %p, vrednost:\t %d\n\n", &i, i);

    return 0;
}
```

Operacije sa pokazivačima

- Sabiranje/oduzimanje sa brojem:

```
int *p;
p = p + 10;
p++;
```

- Poređenje:

```
int *p1, *p2;
…
if (p1 == p2)      // provera da li su
                   // iste adrese
   ...
```

Pokazivači i nizovi

- **Nizovi se mogu posmatrati kao pokazivači**
- Kada se definiše niz, alocira se navedeni broj memorijskih lokacija za smeštaj elemenata niza. Promenljiva koja predstavlja niz se postavlja tako da pokazuje na prvu od ovih lokacija.

Pokazivači i nizovi

- Identifikator niza je zapravo pokazivač na prvi element u memoriji (koja je dodeljena tom nizu).

- Primer:

```
int a[20];  // a – je identifikator niza
int *pa;
```

`pa = a;` je isto što i: `pa = &a[0];`

```
a+2    == &a[2] == pa+2

*(a+3) == a[3]  == *(pa+3)
```

Zadatak iz primera 1 urađen korišćenjem pokazivača

```c
#include <stdio.h>
#define MAX_SIZE 30
int main()
{
    int a[MAX_SIZE];
    int i, n;
    int *p;

    p = a;

    do {
        printf("Unesite broj elemenata niza (maksimalno %d): ", MAX_SIZE);
        scanf("%d", &n);
    } while ( n <= 0 || n>MAX_SIZE);

    for (i = 0; i<MAX_SIZE; i++) {
        *(a+i) = 0;
    }

    for (i = 0; i<n; i++) {
        printf("a[%d]=",i);
        scanf("%d",(a+i));
    }

    printf("\n Elementi niza po ucitanom redosledu: \n");

    for (i = 0; i<n; i++) {
        printf("a[%d]=%d\n", i, *(a+i));
    }

    printf("\n Obrnuti redosled: \n");

    for (i = n-1; i>=0; i--) {
        printf("a[%d]=%d\n", i, *(p+i));    // ovde koristimo pokazivac p
    }

    return 0;
}
```

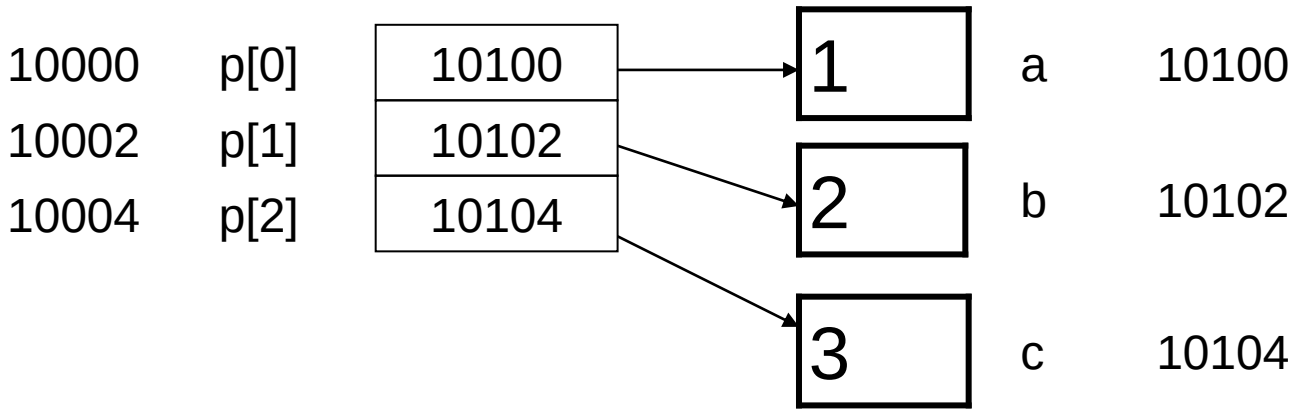
Nizovi pokazivača

- Opšta sintaksa:

```
tip *pok[vel];
```

- Primer:

```
int a=1, b=2, c=3;
int *p[3]; p[0] = &a; p[1] = &b; p[2]=&c;
```



Vrlo česte greške

- Nemoguće je definisati pokazivač na konstantu ili izraz.
- Nemoguće je promeniti adresu promenljive (jer to i ne zavisi od nas).
- Zbog ovoga sledeći izrazi su pogrešni:
  - i = &3;
  - j = &(k+5);
  - k = &(a==b);
  - &a = &b;
  - &a = 150;

Primer 4 - pristup elementima niza preko pokazivača.

```c
#include <stdio.h>

int main ( )
{
    /* Niz se moze inicijalizovati i prilikom deklaracije */
    int a[10] = {1, 2, 3, 4, 5, 6, 7 ,8, 9, 0};

    int *pa;
    int i;

    pa = a;
    printf("%d\n\n", *pa);

    printf("%d\n\n", *(a+1));

    // ispis elementa sa parnim indexom
    for (i=0; i<10; i+=2) {
        pa = a+i;
        printf("%d\n", *pa);
    }

    return 0;
}
```

Zadatak

- Dat je niz od maksimalno 20 realnih elemenata. Učitati n elemenata, naći maksimalnu vrednost, a zatim sortirati niz u rastućem redosledu. Zadatak uraditi upotrebom pokazivača.

FUNKCIJE

Povratne vrednosti - primeri

U sledecim primerima date su deklaracije funkcija bez argumenata sa razlicitim povratnim vrednostima:

```
void f1(); /*Funkcija nema povratnu vrednost*/
void f2(void); /*Funkcija nema povratnu vrednost*/

int f3(); /*Povratna vrednost celobrojna*/
unsigned int f4(); /*Povratna vrednost tipa unsigned int*/
long int f5(); /*Povratna vrednost tipa long int*/

float f6(); /*Povratna vrednost tipa float*/
double f7(); /*Povratna vrednost tipa double*/

int *f8(); /*Povratna vrednost je pokazivac na int*/
```

Parametri funkcije

- Svaki parametar deklariše lokalnu promenljivu koja je vidljiva samo u telu funkcije
- Vrednost ove promenljive se postavlja na vrednost koja je saopštena prilikom poziva funkcije (vrednost prosleđenog argumenta)
- Prestaje da postoji neposredno nakon izlaska iz funkcije, tj. kad se izvrši return.

Parametri funkcije

- Važni koncepti:
  - Lokalna promenljiva u funkciji je potpuno različita od promenljive koja je prosleđena prilikom poziva funkcije
  - Ukoliko funkcija izvrši promenu vrednosti jednog od svojih parametara, to ne utiče na promenljivu čija je vrednost prosleđena funkciji.

Doseg (oblast važenja promenljivih)

- deklaracija promenljivih je vidljiva u sledećim delovima koda u zavisnosti od tipa deklaracije:
  - lokalne promenljive: do kraja bloka
  - parametri funkcija: do kraja funkcije
  - globalne promenljive: do kraja programske datoteke (korišćenje globalnih promenljivih je loša programerska praksa)

Prenos parametara

```
#include <stdio.h>
// PO VREDNOSTI
void f(int i) {
    i = 3;
}

int main() {
    int i = 5;
    f(i);
    printf("%i", i);
    return 0;
}
```

```
#include <stdio.h>
// PO REFERENCI
void f(int *i) {
    *i = 3;
}

int main() {
    int i = 5;
    f(&i);
    printf("%i", i);
    return 0;
}
```

Prenos parametara

```
#include <stdio.h>

int main() {
    int a = 5;
    int *p = &a;
    int b = 3;
    p = &b;
    return 0;
}
```

```
#include <stdio.h>

void change(int **p) {
    int b = 3;
    *p = &b;
}

int main() {
    int a = 5;
    int *p = &a;
    change(&p);
    return 0;
}
```

Prenos po vrednosti

- Ovaj termin znači da se argumenti poziva funkcije prenose u funkciju tako što se njihova vrednost kopira u lokalne promenljive čija imena odgovaraju parametrima funkcije. **Funkcija NEMA direktan pristup polaznim promenljivim.**

Prenos po referenci

- Ovaj termin znači da se argumenti poziva funkcije prenose u funkciju tako što se prosleđuje adresa promenljive, pa preko adrese možemo da čitamo i manjamo polaznu promenljivu. **Funkcija IMA pristup polaznim promenljivim.**

Naredba return

- **Ova naredba omogućava funkciji da vrati neku vrednost, i istovremeno se izvršavanje funkcije završava**
- Vrednost koja se vraća mora odgovarati tipu koji je specificiran kao povratni tip funkcije
- Funkcija može sadržavati više return naredbi, ali prva na koju se naiđe završava izvršavanje funkcije i vraća navedenu vrednost

Zadatak

1. Napisati C program koji učitava dužine kateta i računa dužinu hipotenuze. Za računanje hipotenuze napisati funkciju **hipotenuza.**
2. Dat je niz od maksimalno 30 celobrojnih elemenata. Učitati n elemenata,a zatim izračunati sumu elemenata niza. Učitavanje elemenata, računanje sume i ispis elemenata niza realizovati kao posebne funkcije.

Primer funkcije ispis

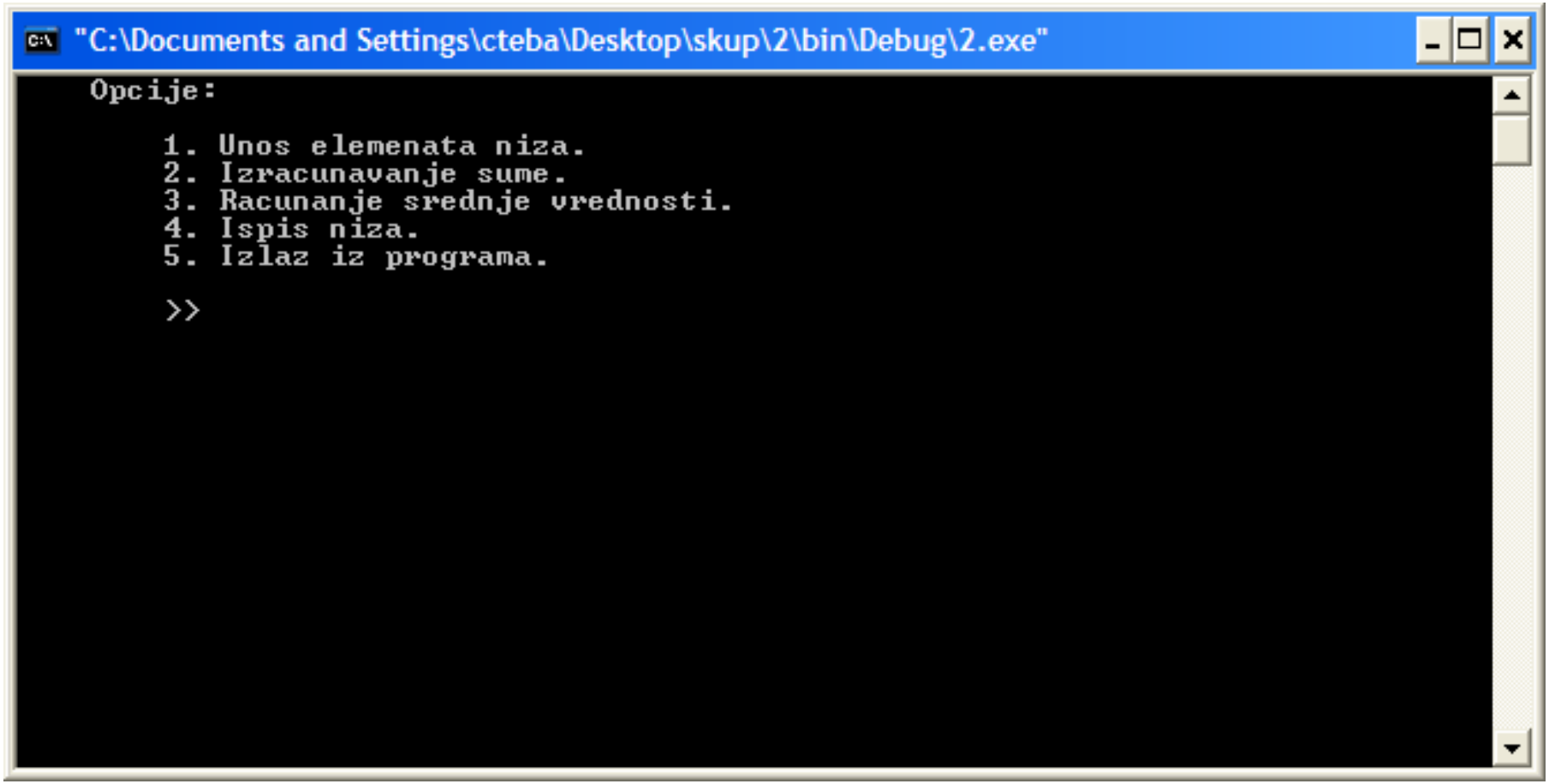
```
void ispis(int *a, int n) {
    int i; // lokalna promenljiva

    printf("[");
    for(i=0; i<n; i++){
        printf("%d", a[i]);
        if( i!=(n-1)) printf(", ");
    }
    printf("]");
}
```

Zadatak

Dat je niz od maksimalno 30 celobrojnih elemenata. Učitati n elemenata,a zatim omogućiti korisniku da bira neku od sledećih radnji: izračunavanje sume elemenata niza, računanje srednje vrednosti elemenata, nalaženje minimuma, nalaženje maksimuma. Omogućiti izvršavanje više radnji (jedna za drugom).

...

Izlaz na konzoli:



...

```c
void unosNiza(int *, int *);
int suma(int *, int);
double avg(int *, int);
void ispis(int *, int);

int main() {

    int a[MAX_SIZE];
    int n, opcija;

    do {
        printf("    Opcije:\n\n");
        printf("\t1. Unos elemenata niza.\n");
        printf("\t2. Izracunavanje sume.\n");
        printf("\t3. Racunanje srednje vrednosti.\n");
        printf("\t4. Ispis niza.\n");
        printf("\t5. Izlaz iz programa.\n ");
        printf("\n\t>> ");
        scanf("%d", &opcija);

        switch( opcija ) {
            case 1: unosNiza(a, &n); break;
            case 2: printf("Suma je: %d.\n\n", suma(a, n)); break;
            case 3: printf("Srednja vrednost je: %lf.\n\n", avg(a, n)); break;
            case 4: ispis(a, n); break;
        }

    } while ( opcija != 5);

    return 0;
}
```

STRINGOVI

Stringovi

String je jednodimenzionalni niz tipa char. Da bi se znalo gde se u nizu završava string dopisuje mu se takozvani završni ili NULL znak: '\0', koji je sastavni deo stringa. Rad sa stringovima predstavija specifican nacin korišćenja niza znakova.

Stringovi

- Za C kompajler string konstanta je svaki niz znakova između navodnika. Znaci između navodnika plus završni znak se registruju u nizu susednih memorijskih lokacija. Zbog prisustva završnog znaka string konstanta "C" nije jednaka znakovnoj konstanti 'C', jer je "C" niz od dva znaka: 'C' i '\0'. Prazna string konstanta " " se sastoji samo od završnog znaka.

Inicijajizacija stringova

- Stringovi se mogu inicijalizovati korišćenjem string konstanti.
- Na primer char s[]="IBM PC";inicijalizuje statički niz s datom string konstantom. Ovaj način inicijalizacije predstavlja skraćeni oblik inicijalizacije niza
- char s[]={'I', 'B','M',' ','P','C','\0'};

Inicijajizacija stringova

char s[]={'I', 'B','M',' ','P','C','\0'};

- Ako bi se u inicijalizaciji izostavio završni znak to više ne bi bio string već samo niz znakova.
- Kao i kod drugih nizova, ime s predstavlja pokazivač na nulti element niza, tako da važi:

  s=&s[0], *s='I', i *(s+1)=s[1]='B'

Učitavanje stringova

- Pri učitavanju stringova treba voditi računa o dve stvari: kako rezervisati memoriju za čuvanje stringa i kako primeniti funkciju za učitavanje stringa. Na početku rada potrebno je odrediti maksimalnu dužinu stringa koji će se učitavati da bi kompajler dodelio traženi memorijski prestor.

  Na primer, deklaracijom

  ```
  char ime[101];
  ```

  uvodi se niz ime od najviše sto jednog elementa.

  Učitavanje se može realizovati pomoću dve bibliotečne funkcije scanf () i gets ().

Funkcija gets()

- **Za učitavanje stringova funkcija gets() se češće koristi od scanf().** Ona prihvata znake sa tastature dok ne naiđe na znak za "novu liniju" (‘\n') koji se proizvodi pritiskom na taster ENTER. Ona ignoriše znak za "novu liniju" i dodaje završni ili nulti znak ('\0') stringa.
- Funkcija gets () ima sledeće zaglavlje:

```
char *gets(char *s)
```

gde je argument s pokazivač na string. Funkcija kojoj se predaje učitani niz kao argument predaje ime niza koji je u stvari adresa nultog elementa niza u koji se kopiraju vrednosti koje prosleđuje gets(). Medutim, treba voditi računa da je vrednost funkcije pokazivač na adresu do koje je učitan string ako je učitavanje korektno, u protivnom vrednost je NULL (nulta adresa koja je definisana u fajlu **stdio.h** sa 0).

- Kontrola ulaza se može realizovati na sledeći način:

```
while (gets(ime)==NULL);
```

Ispis stringova

- **Za ispis stringova se koriste dve bibliotečke funkcije puts() i printf().** Funkcija puts() ispisuje samo stringove, a printf() može ispisivati mešovite podatke. Dok se pri ispisu stringa sa puts() automatski prelazi na novu liniju pri korišćenju printf() to se mora naznačiti sa '\n'. Na primer:

  ```
  printf("%s\n" ,string);
  ```

  ima isti efekat kao i

  ```
  puts(string);
  ```

- **puts()** funkcija ima samo jedan argument - pokazivač na početnu tačku stringa (ili podstringa) koji treba ispisati. Ova fuinkcija ispisuje sve znake počev od tačke na koju pokazuje argument funkcije do završinog znaka ('\0') koga zamenjuje znakom za "novu liniju" tako da se posle ispisa stringa prelazi na novu liniju.

Zadatak

1. Napisati funkciju koja ispisuje string obrnutim redom. Napraviti kratak test program (učitati string i ispisati obrnutim redosledom karaktere) sa i bez upotrebe f-ije strlen(s).
2. Napisati funkciju koja kao parametre uzima jedan string i karakter, a vraća broj pojavljivanja tog karaktera u string. Napisati test program. (Primer: str="tatatatira", c='a', povratna vrednost je 4).

Zadatak

Učitati string,a zatim omogućiti korisniku da bira neku od sledećih radnji: ispis stringa, ispis stringa obrnutim redom, brojanje određenog karaktera u stringu, određivanje dužine stringa, brojanje velikih slova u stringu. Omogućiti izvršavanje više radnji (jedna za drugom).

STRUKTURE *(STRUCT)*

Struktura

- **Struktura predstavlja skup podataka kojim se opisuju neka bitna svojstva objekta. Komponente koje obrazuju strukturu nazivaju se elementi strukture.**
- Primer jednostavne strukture.

```
struct licnost {
        char ime[31];
        char adresa[41];
        unsigned starost;
};
```

Sada mozemo deklarisati dve osobe na sledeci nacin:

```
struct licnost osoba1, osoba2;
```

Elementi strukture

- **Kada imamo promenljivu strukturnog tipa tada elementima date strukture pristupamo uz pomoc operatora '.'**

```
osoba1.starost=20;
osoba2.starost=21;
...
if (osoba1.starost == osoba2.starost)
  printf("Osobe su iste starosti\n");
```

```c
#include <stdio.h>

struct tacka {
    float x;
    float y;
};

void ispis(struct tacka t);

int main() {

    struct tacka t1, t2;

    printf("Unesite X i Y koordinatu tacke 1: ");
    scanf("%f %f", &t1.x, &t1.y);

    printf("Unesite X i Y koordinatu tacke 2: ");
    scanf("%f %f", &t2.x, &t2.y);

    ispis(t1);
    ispis(t2);

    return 0;
}

void ispis(struct tacka t) {
    printf("\n( %f, %f)\n", t.x, t.y);
}
```

Zadatak

- Učitati niz tačaka u ravni (maksimalno 30). Naći tačku koja je najbliža koordinatnom početku. Koristiti funkcije.
- Uraditi isto u prostoru.
- Unos tačaka iz datoteke.
- Naći tačku koja je najudaljenija.
- Učitati dva kompleksna broja, realizovati sabiranje i oduzimanje kompleksnih brojeva.
- Realizovati vektorski proizvod.

DINAMIČKO ZAUZIMANJE MEMORIJE

malloc

```
void *malloc (size_t size);
```

- Za argument funkcije uzima veličinu bloka koji želimo alocirati
- Vraća pokazivač na početak bloka (tj, vraća adresu prvog byta bloka).
- Ako alokacija nije uspešna, vraća NULL.
- NULL je posebna lokacija u memoriji koja sigurno ne sadrži bilo koji podatak.
- Za upotrebu malloc trebamo,

```
#include<stdlib.h>
```

calloc

```
void *calloc(n, size);
```

- Rezerviše memorijski blok dovoljan za memorisanje **n** elemenata svaki veličine **size** bajtova, znači **n*size**.
- Rezervisan memorijski blok je inicijalizovan na 0.
- U slučaju uspešne rezervacije calloc vraća pokazivač koji pokazuje na rezervisan memorijski blok. U protivnom, vraća NULL.

realloc

```
void *realloc(pokaz, size);
```

- Oslobađa rezervisani memorijski blok i rezerviše novi veličine size bajtova.
- Argument pokaz je pokazivac na na početak memorijskog bloka koji se realocira.
- Argument size je unsigned i određuje veličinu realociranog memorijskog bloka.
- Ako je realociranje uspoešno realloc vraća pokazivač koji pokazuje na memorijski blok. U protivnom, realloc vraća 0.

free

- Kada nam memorijski blok koji alcoran s malloc nije više potreban, moramo ga osloboditi. To se radi s funkcijom

```
free(arg)
```

- Argument funkcije free() mora biti pokazivač koji pokazuje na početak prostora koji želimo osloboditi (dealocirati).
- ”curenje memorije”
- **Za svaki malloc() moramo imati jedan free()!**

Primer

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

int main () {
     int *br;
     br = (int *) malloc (sizeof(int));

/* malloc-u se prosledjuje kolicina potrebne memorije u bajtovima, a on vraca
     genericki pokazivac (void*) koji pre koriscenja treba konvertovati u odgovarajuci
     pokazivacki tip.*/

   printf("Unesite celi broj: ");
   scanf("%d", br);

   printf("Ucitan broj je: %d\n\n", *br);

     free(br); /* oslobadjanje (dealociranje) */

     return 0;
}
```

ZADATAK

- Napisati program koji dozvoljava sledeće operacije:
  - Dinamičko kreiranje niza koji ima *n* elemenata. Korisnik unosi *n* sa tastature.
  - Popunjavanje niza vrednostima unesenim preko tastature.
  - Proširivanje niza za *k* lokacija. Vrednosti niza treba kopirati na novu lokaciju. Korisnik unosi *k* sa tastature. Koristiti samo *malloc* i *free*.
  - Anuliranje svih elemenata niza.
- Sve prethodne operacije organizovati u obliku funkcija koje se pokreću odabirom stavke u meniju. Sve operacije rade nad istim nizom.
- *Opcija 1*: Elementi niza su tipa int
- *Opcija 2:* Elementi niza su tipa strukture koja opisuje studenta (broj indeksa, ime, prezime, godina)

DATOTEKE

Datoteke

- podržane vrste datoteka
  - tekstualne datoteke
    - sa konverzijom sadržaja
  - binarne datoteke
    - pristup na nivou bajta
- podržane radnje nad datotekama
  - otvaranje, zatvaranje
  - čitanje, pisanje, pozicioniranje
  - provera statusa
  - kreiranje, brisanje

Datoteke

- pogled na datoteku
  - datoteka kao niz bajtova
    - bez složenije podele na slogove
  - neophodno dodatno programiranje
    - koncept toka - *stream*
- pristup sadržaju datoteke
  - sekvencijalan
    - bajt po bajt (podatak po podatak) po redosledu unutar datoteke
  - direktan
    - pozicioniranje na proizvoljan bajt

Datoteke

- tip podatka koji predstavlja datoteku u programu
  - **FILE***
    - pokazivač na tip FILE
    - **FILE je u osnovi struktura**
      - deklarisana u zaglavlju stdio.h
    - primer deklaracije pokazivača datoteke:

```
FILE *f;
```

Datoteke

- Tok rada:
  - **Otvoriti datoteku – `fopen`**
  - Čitamo i/ili upisujemo podatke u datoteku
  - **Zatvorimo datoteku – `fclose`**

Otvaranje datoteke

- FILE *fopen(const char *path, const char *mode);
  - **path** je putanja do datoteke koja će biti otvorena
  - **kreiranje i otvaranje** nove datoteke ukoliko ne postoji datoteka sa datim imenom
    - **otvaranje** datoteke, ukoliko datoteka sa datim imenom postoji
  - **mode** je režim u kome će datoteka biti otvorena

Otvaranje datoteke

- **mogući režimi rada – tekstualne datoteke**
  - "r": otvori postojeću datoteku u režimu čitanja.
  - "w": otvori datoteku u režimu pisanja. Ako datoteka ne postoji biće kreirana, ili ako već postoji njen sadržaj će biti uništen.
    - Upisivanje pocinje od početka datoteke

Otvaranje datoteke

- **mogući režimi rada – tekstualne datoteke**
  - "a": otvori datoteku u režimu dodavanja novih podataka
    - pisanje počinje nakon postojećeg kraja datoteke
    - datoteka će biti kreirana ako ne postoji
  - "r+" – otvori postojeću datoteku u režimu čitanja i pisanja
  - "w+" – otvori tekstualnu datoteku u režimu pisanja i čitanja
    - ako datoteka ne postoji biće kreirana,
    - ako već postoji njen sadržaj će biti uništen
  - "a+" - otvori tekstualnu datoteku u režimu dodavanja novih podataka
    - čitanje počinje od početka datoteke
    - pisanje počinje nakon postojećeg kraja
    - ako datoteka ne postoji biće kreirana

Otvaranje datoteke

- **mogući režimi rada - binarne datoteke**
  - dodavanje slova b u opis režima
    - logika ostaje ista kao kod tekstualnih datoteka
  - moguće kombinacije
    - "rb", "wb", "ab"
    - "r+b", "w+b", "a+b"
      - "rb+", "wb+", "ab+" (identično kao prethodna tri)
- primer:

```
FILE *f = fopen("sadrzaj.bin","r+b");
```

Otvaranje datoteke

- FILE *fopen(const char *path, const char *mode);
  - Ako je datoteka uspešno otvorena u zadatom režimu povratna vrednost će biti pokazivač na konkretni FILE.
  - U suprotnom povratna vrednost će biti NULL i kod uzroka greške će biti postavljen u globalnu promenljivu errno.

Otvaranje datoteke

- int fclose(FILE *f);
  - f - pokazivač na prethodno otvorenu datoteku koja treba da bude zatvorena
  - povratna vrednost
    - 0 ako je zatvaranje uspešno
    - konstanta EOF ako je došlo do greške

Otvaranje datoteke

- int fclose(FILE *f);
  - automatsko zatvaranje
    - pri završetku izvršavanja programa
      - kraj *main* funkcije
      - sve otvorene datoteke bivaju zatvorene
    - ne oslanjati se na to
      - eksplicitno zatvoriti svaku otvorenu datoteku

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE *f;
    char *n="sadrzaj.txt";
    if( (f=fopen(n,"r")) == NULL ){
        printf("Datoteka <%s> nije uspesno otvorena.\n", n);
        exit(1);
    }
    printf("Datoteka <%s> je uspesno otvorena.\n", n);
    printf("Zatvaranje datoteke <%s>...\n", n);
    if( fclose(f)==EOF )
        printf("\tNastupila je greska tokom zatvaranja!\n");
    else
        printf("\tZatvaranje uspesno!\n");
    return 0;
}
```

Rad sa tekstualnim datotekama

- čitanje i pisanje
  - obavlja se sekvencijalno
    - od početka datoteke
  - svaki sledeći pristup
    - iza poslednjeg mesta kojem je pristupano u prethodnom čitanju ili pisanju

Rad sa tekstualnim datotekama

- rad sa znakovima - čitanje
- **int fgetc(FILE *f);**
  - čitanje pojedinačnog znaka
  - **f** - pokazivač na prethodno otvorenu datoteku
  - **povratna vrednost**
    - kod procitanog znaka
    - konstanta EOF ako se došlo do kraja datoteke ili se desila neka greška

Rad sa tekstualnim datotekama

- rad sa znakovima - čitanje
- **char *fgets(char *tekst, int n, FILE *f);**
  - čitanje teksta iz datoteke
    - do znaka '\n' ili
    - ukupno n-1 znakova
  - upisuje '\0' na kraj stringa
  - **tekst** - string koji prihvata učitani sadržaj
  - **n** - maksimalni broj znakova za učitavanje (uključujudi '\0')
  - **f** - pokazivač na prethodno otvorenu datoteku
  - **povratna vrednost**
    - adresa na koju je upisan novi sadržaj - pokazivač na prvi karakter stringa **tekst**
    - konstanta NULL ako se došlo do kraja datoteke ili se desila neka greška

Rad sa tekstualnim datotekama

- rad sa znakovima - pisanje
- **int fputc(int c, FILE *f);**
  - upisivanje pojedinačnog znaka
  - **c** - kod znaka za upis
  - **f** - pokazivač na prethodno otvorenu datoteku
  - **povratna vrednost**
    - kod upisanog znaka
    - konstanta EOF ako se desila neka greška

Rad sa tekstualnim datotekama

- rad sa znakovima - pisanje
- **int fputs(const char *tekst, FILE *f);**
  - upisivanje stringa u formi jednog reda datoteke
    - znak '\n' automatski se dodaje na kraj
  - **tekst** - string koji predstavlja sadržaj za upis
  - **f** - pokazivač na prethodno otvorenu datoteku
  - **povratna vrednost**
    - ne-negativna vrednost
    - konstanta EOF ako se desila neka greška

Rad sa tekstualnim datotekama

- **int fscanf(FILE *f, const char *form, a1, a2, ...);**
  - učitavanje, izvršenje konverzije za a1, a2 ... prema zadatom formatu
  - **f** - pokazivač na prethodno otvorenu datoteku u režimu čitanja
  - **form** - specifikacija konverzije
  - **a1, a2, ...** - adrese za smeštanje sadržaja učitanog iz datoteke
  - **povratna vrednost**
    - broj uspešno konvertovanih i smeštenih podataka
    - konstanta EOF ako se pojavio kraj datoteke ili desila neka greška pre prve konverzije

Rad sa tekstualnim datotekama

- **int fprintf(FILE *stream, const char *format, ...);**
  - upis formatiranog stringa u datoteku
  - **f** - pokazivač na prethodno otvorenu datoteku u režimu pisanja
  - **format** - format u kome se string upisuje u file
  - **povratna vrednost**
    - broj uspešno upisanih karaktera u datoteku
    - negativna celobrojna vrednost u slučaju greške

```
// Primer rada sa tekstualnom datotekom – upis
#include <stdio.h>
#include <stdlib.h>
typedef char TString[31]; // 30 karaktera i ‘\0’
int main()
{
    FILE *IZLdat;
    TString ime = “pera”;
    TString prezime =  “peric”;
    NazivIZLdat = “dat1.txt”

    printf("Unesite naziv datoteke u koju zelite da zapisete podatke: ");
    scanf("%s", NazivIZLdat);

    // otvaranje datoteke sa proverom prava na pisanje(w)
    if((IZLdat=fopen(NazivIZLdat,"w")) == NULL) {
        printf("Greska prilikom otvaranja datoteke %s\n",NazivIZLdat);
        // izlaz iz programa u slucaju zabrane pisanja u datoteku
        exit(EXIT_FAILURE);
    }
f
    fprintf(IZLdat,"%s\n%s",ime, prezime); // upis u izlaznu datoteku
    fclose(IZLdat); // zatvaranje datoteke
    printf("Podaci studenta su upisani u datoteci: %s\n", NazivIZLdat);
    return 0;
}
```

```c
// Primer rada sa tekstualnom datotekom – čitanje
#include<stdio.h>
#define MAXL 80

int main()
{
    FILE *pf;
    char str[MAXL];

    pf=fopen("test.txt", "r");

    if(pf!=NULL){
        while(fgets(str, MAXL, pf)!=NULL)
            puts(str);

        fclose(pf);
    } else {
        printf("Nije moguce otvoriti datoteku ili ona ne postoji.");
    }

    return 0;
}
```

Zadatak

- učitati niz od n celih brojeva sa tastature
  - pri čemu je korisnik prethodno zadao n isto preko tastature
- kreirati tekstualnu datoteku sa nazivom naz koja de u svakom redu sadržati po jedan učitani broj, razmak i vrednost tog broja dignutu na drugi stepen
  - pri čemu je korisnik prethodno zadao naz preko tastature
  - primer
    - za niz {3,5,7} datoteka može izgledati ovako

      3 9
      5 25
      7 49

Rad sa binarnim datotekama

- čitanje i pisanje
  - obavlja se sekvencijalno
    - od početka datoteke
    - svaki sledeći pristup
      - iza poslednjeg mesta kojem je pristupano u prethodnom čitanju ili pisanju
  - mogućnost pozicioniranja
    - zadavanje pomeraja u odnosu na referentnu tačku
      - početak, trenutna pozicija, kraj datoteke
    - slededi pristup od poslednje postavljene pozicije
  - programer vodi računa o strukturi datoteke

Rad sa binarnim datotekama

- rad sa podacima - ulaz
- **int fread(void *pok, int size, int n, FILE *f)**
  - učitavanje zadatog broja elemenata u radnu memoriju iz datoteke od prethodne pozicije čitanja
    - nova pozicija u datoteci na mestu iza poslednjeg učitanog bajta
  - **pok** - pokazivač na početak memorijskog bloka od minimalno size*n bajtova
  - **size** - dužina pojedinačnog podatka za učitavanje (u bajtovima)
  - **n** - broj elemenata veličine **size** koji treba učitati iz datoteke
  - **f** - pokazivač na prethodno otvorenu binarnu datoteku u režimu čitanja
  - povratna vrednost
    - broj uspešno pročitanih podataka

Rad sa binarnim datotekama

- rad sa podacima - izlaz
- **int fwrite(const void *pok, int size, int n, FILE *f);**
  - upisivanje zadatog broja elemenata zadate dužine iz memorije u datoteku od mesta završetka poslednjeg pristupa
  - ako pozicija nije kraj datoteke, dolazi do prepisivanja sadržaja
  - ako kapacitet datoteke nije dovoljan, datoteka se povećava
  - nova pozicija u datoteci na mestu iza poslednjeg upisanog bajta
  - **pok** - memorijska adresa od koje počinje čitanje podataka za smeštanje u datoteku
  - **size** - dužina pojedinačnog podatka za smeštanje (u bajtovima)
  - **n** - broj podataka koji treba smestiti u datoteku
  - **f** - pokazivač na prethodno otvorenu datoteku
  - **povratna vrednost**
    - broj uspešno upisanih podataka
    - ako je došlo greške onda je ta vrednost manja od n

Rad sa binarnim datotekama

- rad sa podacima - pozicioniranje
- **int fseek(FILE *f, long ofset, int ref);**
  - postavljanje trenutne pozicije unutar datoteke
  - može i za tekstualne datoteke
  - neophodna kada posle radnje čitanja dolazi pisanje (i obrnuto)
  - **f** - pokazivač na prethodno otvorenu datoteku
  - **ofset** - udaljenost nove pozicije od referentne tačke (u bajtovima)
  - **ref** - referentna tačka u odnosu na koju se posmatra udaljenost nove pozicije
    - SEEK_SET - početak datoteke
    - SEEK_CUR - trenutna pozicija unutar datoteke
    - SEEK_END - kraj datoteke
  - **povratna vrednost**
    - oznaka greške - ako vrednost različita od 0

Rad sa binarnim datotekama

- rad sa podacima - pozicioniranje
- **long ftell(FILE *f);**
  - nalaženje trenutne pozicije unutar datoteke
  - f - pokazivač na prethodno otvorenu datoteku
  - povratna vrednost
    - trenutna pozicija u bajtovima u odnosu na početak datoteke
    - -1L - oznaka greške
- **void rewind(FILE *f);**
  - postavljanje pozicije na početak datoteke
  - f - pokazivač na prethodno otvorenu datoteku
  - nema povratnu vrednost

Zadatak

- učitati niz od *n* celih brojeva sa tastature
  - pri čemu je korisnik prethodno zadao *n* isto preko tastature
- kreirati binarnu datoteku sa nazivom *naz* koja će za svaki učitani broj sadržati njegovu vrednost kao i tu vrednost dignutu na drugi stepen
  - pri čemu je korisnik prethodno zadao *naz* preko tastature
  - primer
    - za niz {3,5,7} datoteka može izgledati ovako -*little endian*

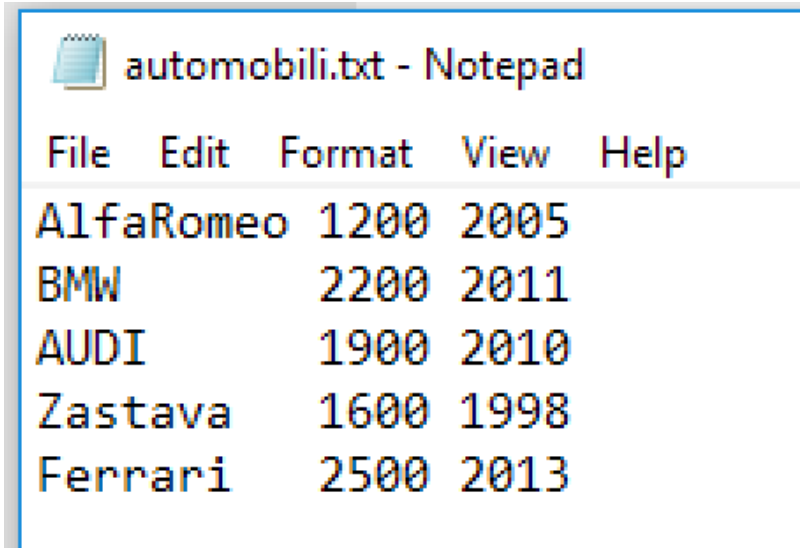
      03 00 00 00 09 00 00 00

      05 00 00 00 19 00 00 00

      07 00 00 00 31 00 00 00

Primer

- Iz ulazne datoteke učitati podatke o automobilima u statički niz
- Za unetu kubikažu vozila pronaći najnovije vozilo sa kubikažom ne većom od zadate



Primer

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

#define MAX_MARKA_LEN 21
#define MAX_NIZ 30

struct automobil {
    char marka[MAX_MARKA_LEN];
    int kubikaza;
    int godiste;
};

FILE *safe_fopen(char *filename, char *mode, int exit_code);
void unos_niza(FILE *in, struct automobil niz[], int *n);
void ispis_niza(FILE *out, struct automobil niz[], int n);
void ispis_auto(FILE *out, struct automobil a);
struct automobil najnoviji_auto(struct automobil niz[], int n, int max_kubikaza);
```

Primer

```
FILE *safe_fopen(char *filename, char *mode, int exit_code) {
    FILE *f = fopen(filename, mode);
    if (f == NULL) {
        printf("Can't open '%s'!\n", filename);
        exit(exit_code);
    }
    return f;
}

void unos_niza(FILE *in, struct automobil niz[], int *n) {
    int i = 0;
    while(fscanf(in, "%s %d %d", niz[i].marka, &niz[i].kubikaza, &niz[i].godiste) == 3) {
        i++;
    }
    *n = i;
}

void ispis_niza(FILE *out, struct automobil niz[], int n) {
    int i;
    for(i=0; i<n; i++) {
        ispis_auto(out, niz[i]);
    }
}
```

Primer

```
void ispis_auto(FILE *out, struct automobil a) {
    fprintf(out, "%s: %d kubika, %d godiste\n", a.marka, a.kubikaza, a.godiste);
}

struct automobil najnoviji_auto(struct automobil niz[], int n, int max_kubikaza) {
    int naj_idx = 0;

    int i;
    for(i=0; i<n; i++) {
        if(niz[i].godiste > niz[naj_idx].godiste) {
            if (niz[i].kubikaza <= max_kubikaza) {
                naj_idx = i;
            }
        }
    }

    return niz[naj_idx];
}
```

```c
int main(int arg_num, char *args[]) {
    if(arg_num != 4) {
        puts("USAGE: ./pretraga 2000 oglasi.txt izvestaj.txt");
        exit(1);
    }

    int max_kubikaza = atoi(args[1]);
    char *in_filename = args[2];
    char *out_filename = args[3];

    printf("max_kubikaza = %d\n", max_kubikaza);
    printf("in_filename = %s\n", in_filename);
    printf("out_filename = %s\n", out_filename);

    FILE *in = safe_fopen(in_filename, "r", 2);
    FILE *out = safe_fopen(out_filename, "w", 3);

    int n;
    struct automobil niz[MAX_NIZ];

    unos_niza(in, niz, &n);
    ispis_niza(out, niz, n);

    fprintf(out, "\n\nNajbolji je:\n");
    ispis_auto(out, najnoviji_auto(niz, n, max_kubikaza));

    fclose(in);
    fclose(out);

    return 0;
}
```

Zadatak

Iz ulazne datoteke učitati niz studenata koji su položili ispit (indeks, ime, ocena). Koristiti funkcije, ime ulazne datoteke prihvatiti kroz argumente komandne linije.

- U izlaznu datoteku "najvisi.txt" ispisati podatke o studentu sa najvišim prosekom
- U izlaznu datoteku "najnizi.txt" ispisati podatke o studentu sa najnižim prosekom
- Na standardnom izlazu ispisati prosečnu ocenu svih studenata.

Zadatak

Prebrojati reči iz ulazne datoteke. U izlaznoj datoteci ispisati broj reči i najdužu pronađenu reč. Koristiti funkcije, imena datoteka prihvatiti kroz argumente komandne linije

DINAMIČKE STRUKTURE PODATAKA

Klasifikacija struktura podataka

Klasifikacija 1

- prema dozvoljenom broju neposrednih prethodnika i sledbenika jednog čvora strukture
  - linearne strukture
  - strukture stabla
  - mrežne strukture
- strukture bez konteksta (semantike)
  - bitne samo karakteristike grafa koji može biti
    - linearan, stablo, mreža

Klasifikacija struktura podataka

Klasifikacija 2

- prema nivou apstraktnosti skupa čiji elementi snabdevaju čvorove i ivice grafa semantikom
  - strukture nad skupom obeležja (strukture obeležja)
    - nazivaju i logičkim strukturama
    - visok nivo apstraktnosti
  - strukture nad skupom podataka (strukture podataka) - mogu biti
    - logičke strukture podataka
    - fizičke strukture podataka

Linearne strukture podataka

- linearne strukture podataka - LinSP
- svaki čvor grafa
  - može da ima najviše jednog direktnog prethodnika
  - može da ima najviše jednog direktnog sledbenika
- klasifikacija
  - aciklične – lanci, otvorene liste i nizovi
  - ciklične – zatvorene liste ili prsten

Predstavljanje LinSP

- u zavisnosti od postupka dodavanja i brisanja elemenata, aciklične linearne strukture se nazivaju:
  - n-torka ili vektor (ako se dodavanje i brisanje ne vrši)
  - stek (ako se dodavanje i brisanje vrši samo na jednom kraju)
  - red (ako se dodavanje vrši na jednom, a brisanje na drugom kraju)
  - dek (ako se i dodavanje i brisanje vrši na oba kraja)

Predstavljanje LinSP

- dva osnovna postupka za predstavljanje linearnih struktura podataka u operativnoj memoriji:
  - sekvencijalna reprezentacija - niz
  - spregnuta reprezentacija – lista

Stek

- stek se može sekvencijalno reprezentovati na sličan način kao i niz fiksne dužine
- problem: ograničen broj elemenata
- Metode:
  - pop (skini element sa vrha steka)
  - push (stavi element na vrh steka)
  - peek (pročitaj element na vrhu steka)
  - empty (da li je stek prazan)

Red

- realizacija pomoću niza fiksne dužine
- ponovo se koriste lokacije oslobođene pri čitanju elemenata iz reda, tako da do prekoračenja može doći tek ako treba upisati N-ti element u red
- metode:
  - ubaci
  - čitaj
  - nađi
- referentni pokazivači: *levi* i *desni*
  - *levi*: pokazuje na poziciju sa koje treba pročitati sledeći element
  - *desni*: pokazuje na prvu slobodnu poziciju na koju je moguće upisati sledeći element

Spregnuta predstava LinSP

- svaki čvor strukture se smešta u jednu lokaciju
- lokacija ima jedno polje za smeštanje pokazivača ka lokaciji neposredno narednog čvora
- neposredno susedni čvorovi nisu, u opštem slučaju, u fizički neposredno susednim lokacijama

Lista

- zadatak 1
  - spregnuta reprezentacija linearne strukture podataka tipa lista

Zadatak

Napisati C program koji realizuje osnovne operacije sa dinamičkom strukturom podataka oblika jednostruko spregnute liste. Element liste sastoji se od dva polja:

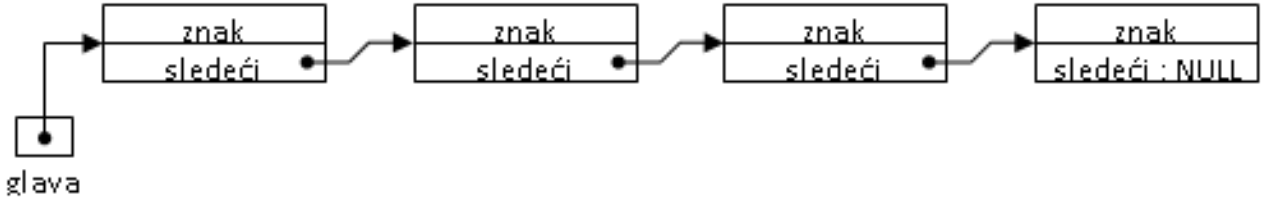
- **podatak** (u ovom slučaju celobrojan broj)
- **pokazivač** na sledeći element liste

Operacije koje treba realizovati sa jednostrukom spregnutom listom su:

- **unos novog karaktera** (ako se ne nalazi u listi),
- **modifikacija karaktera** (umesto jednog karaktera treba upisati drugi karakter),
- **traženje karaktera** u listi i
- **brisanje karaktera** iz liste.

U zadatku se koristi dinamička struktura podataka u obliku jednostruko spregnute liste. Da bismo mogli pristupati elementima liste potreban nam je pokazivač na prvi elemenat liste, koji se naziva **glava** liste.

Grafička predstava jednostruko spregnute liste u ovom zadatku je:



- NULL označava da pokazivač ne pokazuje ni na šta.
- Uopšteno, pokazivačka promenljiva sadrži adresu elementa (sloga liste) na koji pokazuje.

Operacije sa listom su:

1) inicijalizacija liste,
2) unos novog elementa na pocetak liste,
3) listanje liste (prikaz svih elemenata iz liste),
4) brisanje elementa iz liste i
5) brisanje liste (oslobađanje memorije).

1) Model elementa liste

```
typedef struct elem {
    int broj;
    struct elem *sled;
} Elem; // Element liste.
```

2) Unos novog elementa u listu može se obaviti dodavanjem elementa na početak liste ili dodavanjem elementa na kraj liste.

```
Elem *na_pocetak(Elem *lst, int b) {
  Elem *novi = malloc(sizeof(Elem));
  novi->broj = b;
  novi->sled = lst;
  lst = novi;
  return lst;
}
```

2) Unos novog elementa u listu može se obaviti dodavanjem elementa na početak liste ili dodavanjem elementa na kraj liste.

```
Elem *na_kraj(Elem *lst, int b) {
  Elem *novi = malloc(sizeof(Elem));
  novi->broj = b;
  novi->sled = NULL;
  if (!lst){ //ukoliko je lista prazna
    return novi;
  }else {
    Elem *tek = lst;
    while (tek->sled){
        tek = tek->sled;
    }
    tek->sled = novi;
    return lst;
  }
}
```

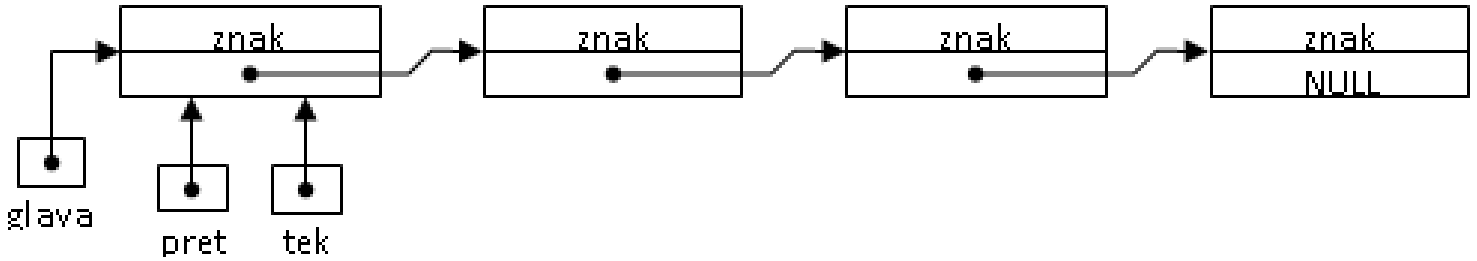
3) Listanje liste predstavlja prikazivanje svih elemenata iz liste

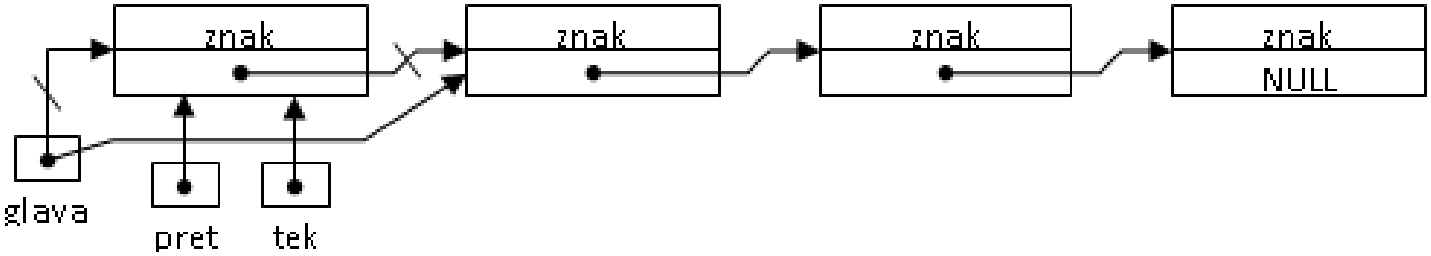
```
void pisi(Elem *lst) {
  while (lst) {
    printf("%d ", lst->broj);
    lst = lst -> sled;
  }
}
```

4) Brisanje elementa iz liste realizuje se tako što se mora element prvo pronaći (vrši se traženje elementa u listi).

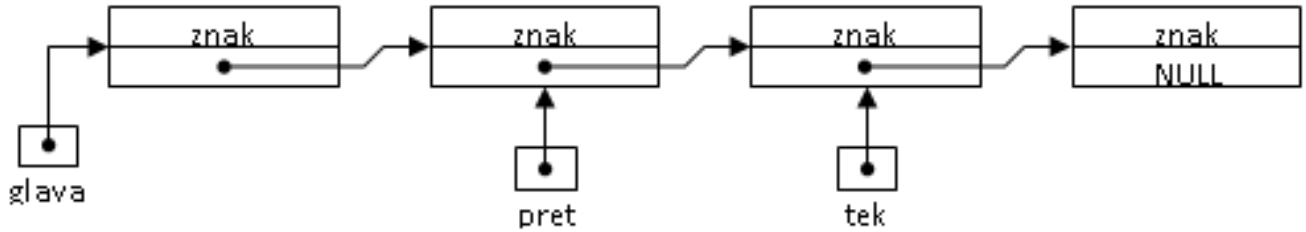
```
Elem *brisiElement(Elem *lst, int b){
  Elem *tek = lst;
  Elem *pret = NULL;

  while (tek){
    if (tek->broj != b) {
      pret = tek;
      tek = tek->sled;
    } else {
      Elem *stari = tek;
      tek = tek->sled;
      if (!pret){ //ukoliko se brise prvi element u listi
        lst = tek;
      }else{
        pret->sled = tek;
      }
      free(stari);
    }
  }

  return lst;
}
```

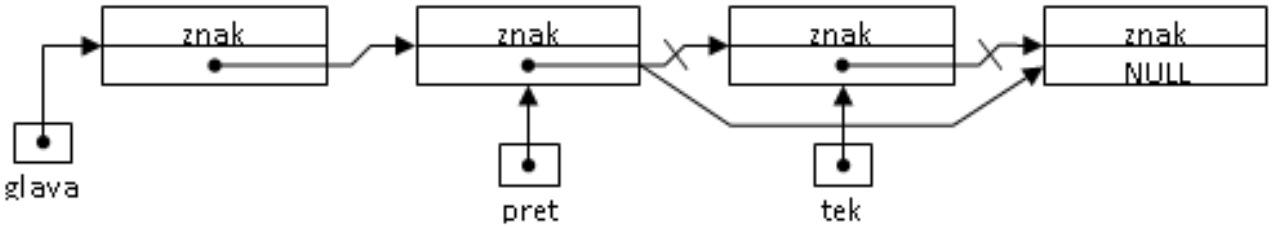
b) Slog koji se briše jeste prvi elemenat liste





a) Slog koji se briše nije prvi elemenat liste

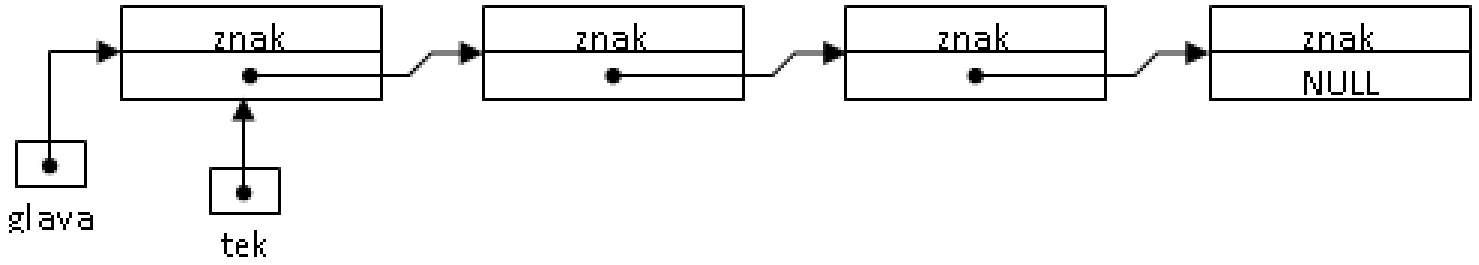


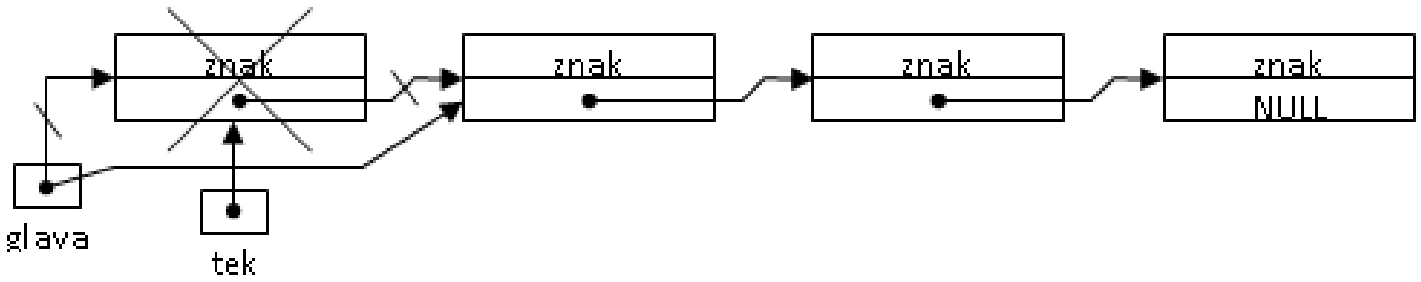


5) Brisanje liste

```
void brisiSve(Elem *lst) {
  while (lst) {
    Elem *stari = lst;
    lst = lst->sled;
    free(stari);
  }
}
```

5) Brisanje liste



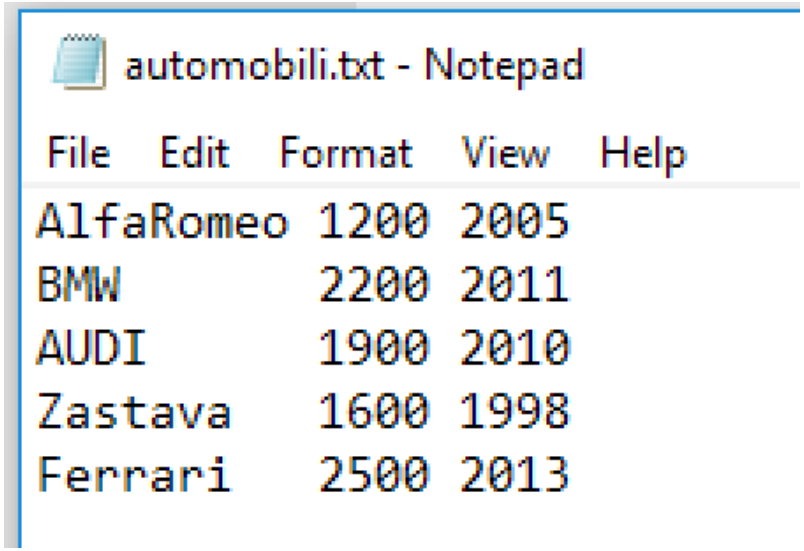


6) Umetanje elementa u uredjenu listu

```
Elem *umetni(Elem *lst, int b) {
  Elem *tek=lst;
  Elem *pret=NULL;

  while (tek && tek->broj < b){
      pret = tek;
    tek = tek->sled;
  }

  Elem *novi = malloc(sizeof(Elem));
  novi->broj = b;
  novi->sled = tek;

  if (!pret) {
    lst = novi;
  }else{
     pret->sled = novi;
  }

  return lst;
}
```

Zadatak

- Iz ulazne datoteke učitati podatke o automobilima u jednostruko spregnutu listu.
- Za unetu kubikažu vozila pronaći najnovije vozilo sa kubikažom ne većom od zadate.



Zadatak

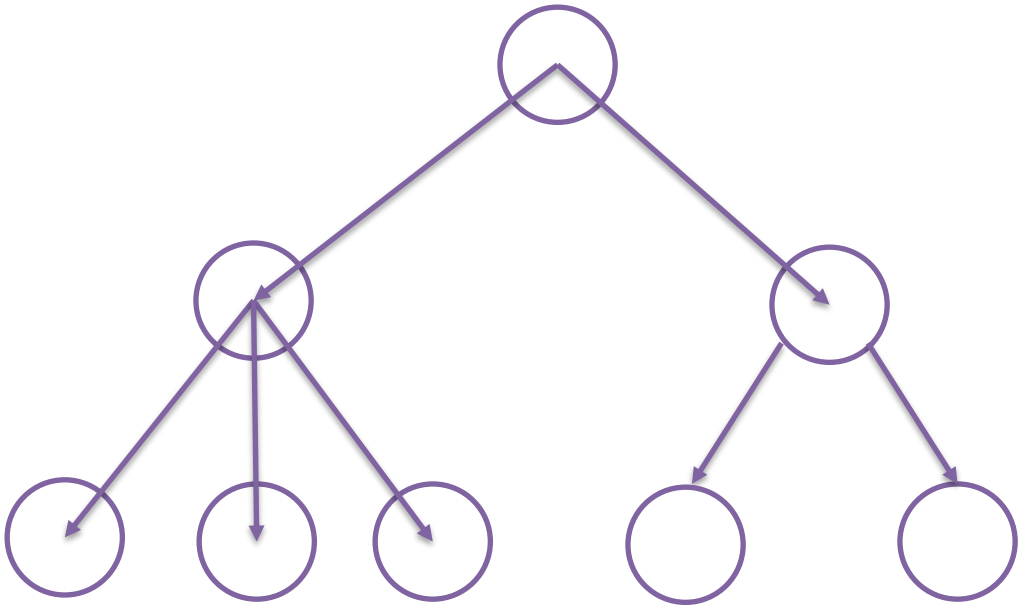
- Napisati C program za evidenciju studenata.
  - Svaki student je opisan:
    - imenom (do 20 karaktera),
    - prezimenom (do 20 karaktera),
    - brojem indeksa (do 10 karaktera) i
    - godinom upisa studija.
  - Program treba da obezbedi sledeće:
    - Unos podataka o studentima na proizvoljnu poziciju u odgovarajuću strukturu,
    - Prikaz svih studenata iz odgovarajuće strukture,
    - Brisanje studenta sa zadatim brojem indeksa, iz odgovarajuće strukture
    - Čitanje podataka o studentima iz datoteke *studenti.txt*, i smeštanje u odgovarajuču strukturu
      - podaci su delimitirani jednim blank znakom
    - Snimanje strukture u datoteku *studenti.txt*.

Zadatak

- Proširiti zadatak 1 sledećim funkcionalnostima
  - Prilikom dodavanja novog studenta, omogućiti proveru da li već postoji student sa tim brojem indeksa,
  - Čuvanje podataka o studentima u binarnoj datoteci *studenti.bin,*
  - Čitanje podataka o studentima iz binarne datoteke *studenti.bin* i njihovo smeštanje u odgovarajuću strukturu,
  - Za svakog studenta potrebno je evidentirati broj položenih ispita,
    - omogućiti izmenu broja položenih ispita za studente, na osnovu broja indeksa.

[Strukture tipa stabla]

- svaki čvor
  - može imati najviše jednog direktnog prethodnika i do n direktnih sledbenika,
    - $0 \leq n \leq N-1$,
    - N - kardinalni broj skupa koji se snabdeva strukturom



Strukture tipa stabla

- strukture stabla su aciklične strukture
- postoji put od najmanjeg do svakog drugog čvora
- postoji bar jedan maksimalan čvor
- broj grana u stablu iznosi N-1

Strukture tipa stabla

- najmanji čvor – koren
- maksimalni čvor – list
- nivovska hijerarhija:
  - koren je čvor prvog nivoa hijerarhije
  - proizvoljan čvor s_i nalazi se na k-tom nivou hijerarhije, ako se nalazi na kraju puta dužine k-1, a put počinje u korenu stabla
- broj nivoa hijerarhije h – visina stabla

Strukture tipa stabla

- red stabla:
  - za stablo se kaže da je n-arno, odnosno reda n, ako svakom čvoru koji nije list odgovara maksimalno n direktno potčinjenih čvorova
  - n=2 – binarno stablo
- puno stablo:
  - svi listovi se nalaze na istom rastojanju od korena (h-1)

Strukture tipa stabla

- kompletno stablo:
  - svi čvorovi, koji ne predstavljaju listove, imaju svih n odlaznih potega, odnosno svih n direktno podređenih čvorova
- balansirano stablo
  - za svaki čvor važi da se broj čvorova u svakom njegovom podstablu ne razlikuje za više od jedan
- optimalno balansirano stablo
  - stablo reda n, čiji su svi čvorovi na nivoima od 1 do h-2 kompletni

Predstava binarnog stabla

- semantika pridružena čvoru stabla smešta se u memorijsku lokaciju
- grana stabla se može reprezentovati bilo putem fizičkog pozicioniranja, bilo putem pokazivača

Rekurzivne funkcije

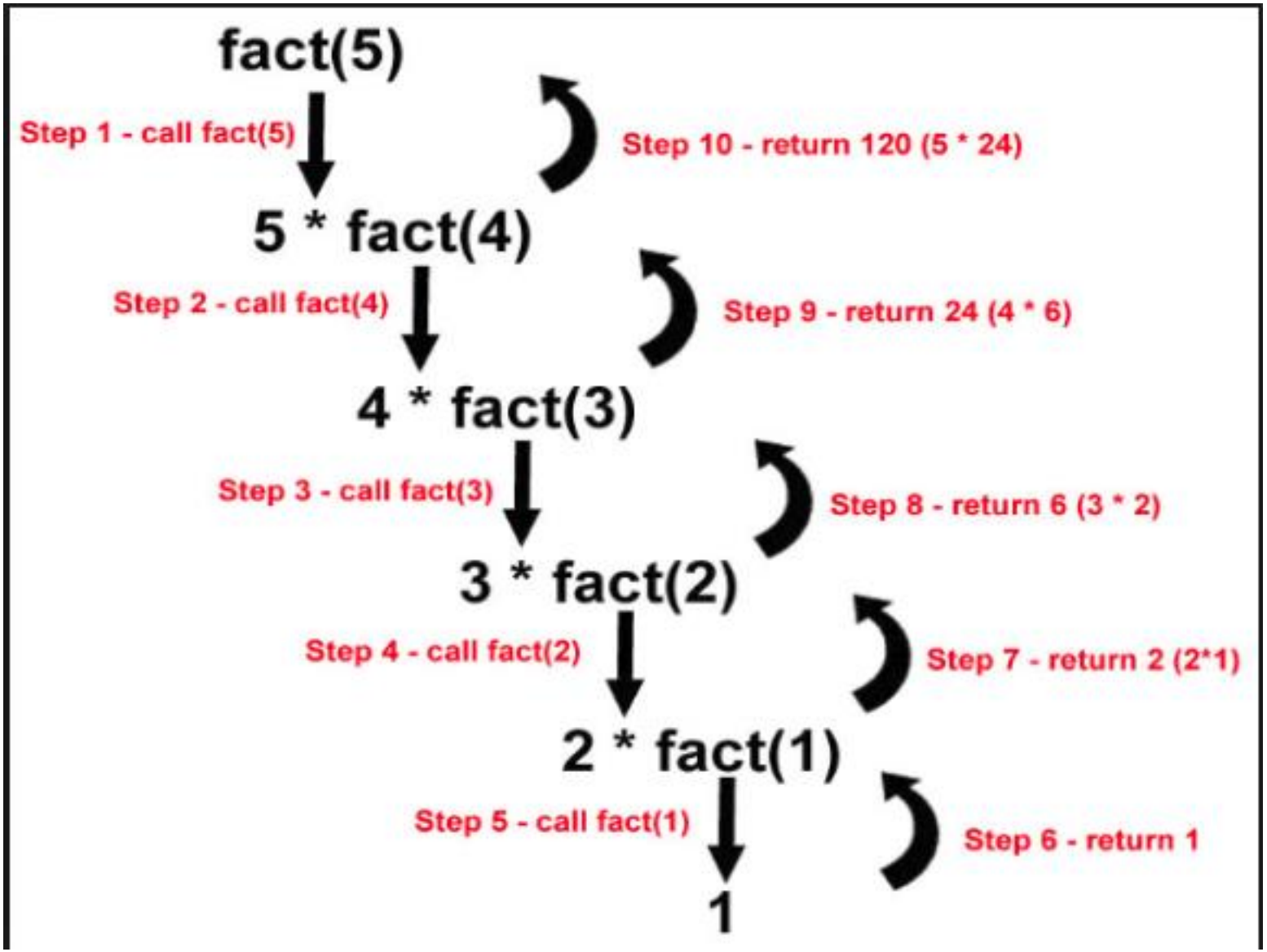
- Funkcija koja poziva samu sebe
- Mora postojati uslov za izlazak iz rekurzije

Rekurzivne funkcije

- Računanje faktorijela

```
unsigned long long int factorial(unsigned int i) {

                        //USLOV ZA IZLAZAK IZ REKURZIJE
    if(i <= 1) {
        return 1;
    }
    return i * factorial(i - 1);
}
```

Rekurzivne funkcije



1) Model elementa stabla

```
typedef struct b_cvor_st
{
  int inf; //TIP inf;
  struct b_cvor_st *desni;
  struct b_cvor_st *levi;
} BCVOR;
```

2) Unos novog elementa u stablo:

- Kreira se novi čvor
  - novi = (BCVOR *)malloc(sizeof(BCVOR));
- prođe se kroz stablo i kada se stigne do kraja stabla (naiđe na list), umetne se ispod njega (levo ili desno zavisi od sadržaja)

```
BCVOR *kreirajCvor(int br){

  BCVOR *novi = (BCVOR *)malloc(sizeof(BCVOR));
  novi->inf = br;
  novi->levi = NULL;
  novi->desni = NULL;

  return novi;

}
```

2) Unos novog elementa u stablo:

- Kreira se novi čvor
  - novi = (BCVOR *)malloc(sizeof(BCVOR));
- prođe se kroz stablo i kada se stigne do kraja stabla (naiđe na list), umetne se ispod njega (levo ili desno zavisi od sadržaja)

```
void dodaj(BCVOR **cvor, BCVOR *novi){

   if(*cvor == NULL){
       *cvor = novi;
       return;
   }

   if( novi->inf < (*cvor)->inf){ // br treba da ide levo
       dodaj( &((*cvor)->levi) , novi);
   }
   else if( novi->inf > (*cvor)->inf){
       dodaj( &((*cvor)->desni) , novi );
   }
   else return; // vec postoji taj broj u stablu


}
```

3) Listanje stabla predstavlja prikazivanje svih elemenata iz liste

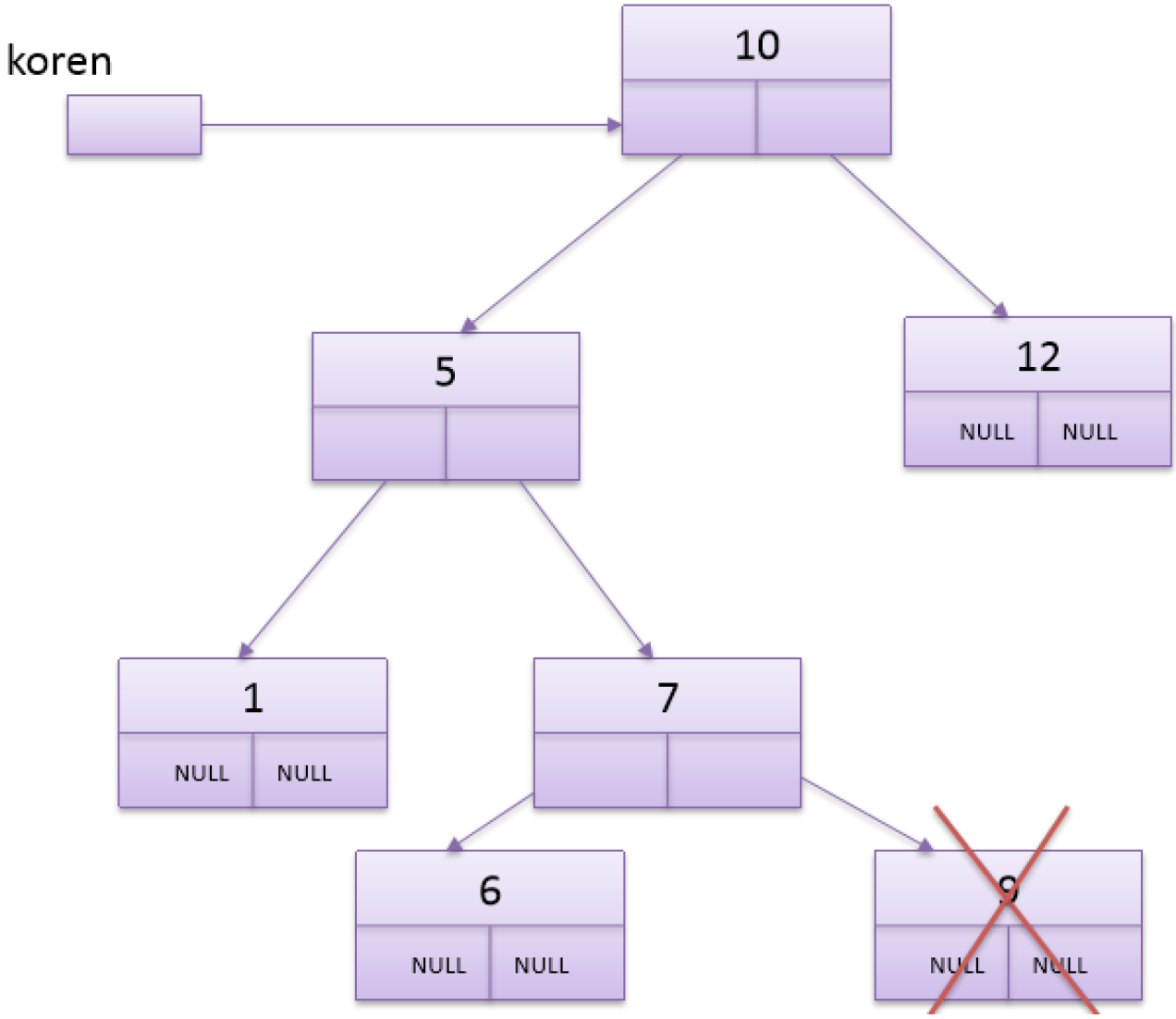
```
void ispisi(BCVOR *cvor){

   if(cvor != NULL){
       printf("\n");
       printf("Cvor:%d", cvor->inf);
       if(cvor->levi != NULL){
        printf(" Levi:%d", cvor->levi->inf);
       }else{
        printf(" Levi: NULL”);
       }
       if(cvor->desni != NULL){
        printf(" Desni:%d", cvor->desni->inf);
       }else{
        printf(" Desni: NULL");
       }
       ispisi(cvor->levi);
       ispisi(cvor->desni);
   }

}
```

4) Traženje elementa u stablu

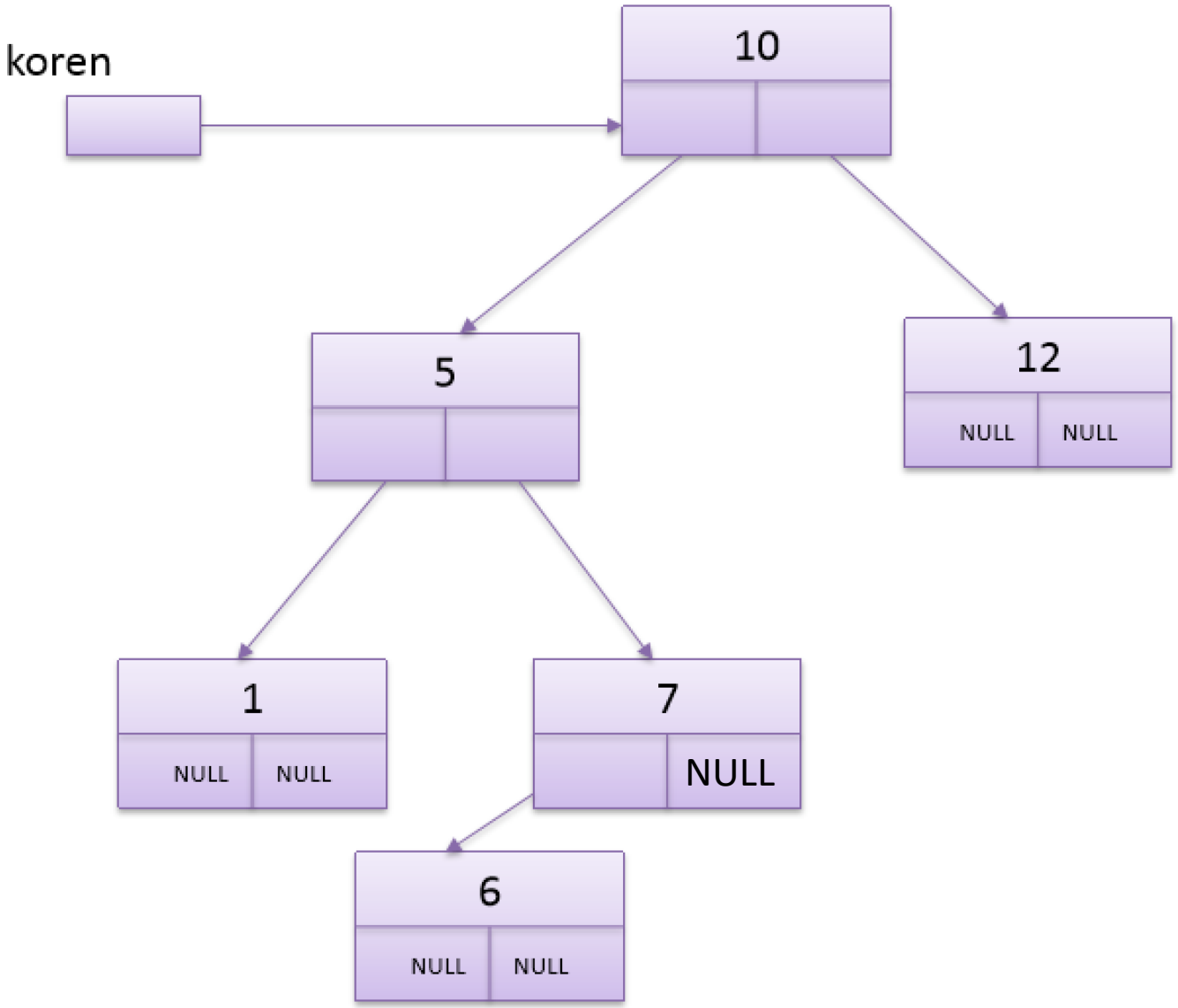
```
BCVOR *trazi(BCVOR *cvor, int br){

    if(cvor == NULL){
        return NULL;
    }

    if(cvor->inf == br){
        return cvor;
    }else if(br < cvor->inf){
        return trazi(cvor->levi, br);
    }else{
        return trazi(cvor->desni, br);
    }
}
```

5) Brisanje elementa iz stabla:

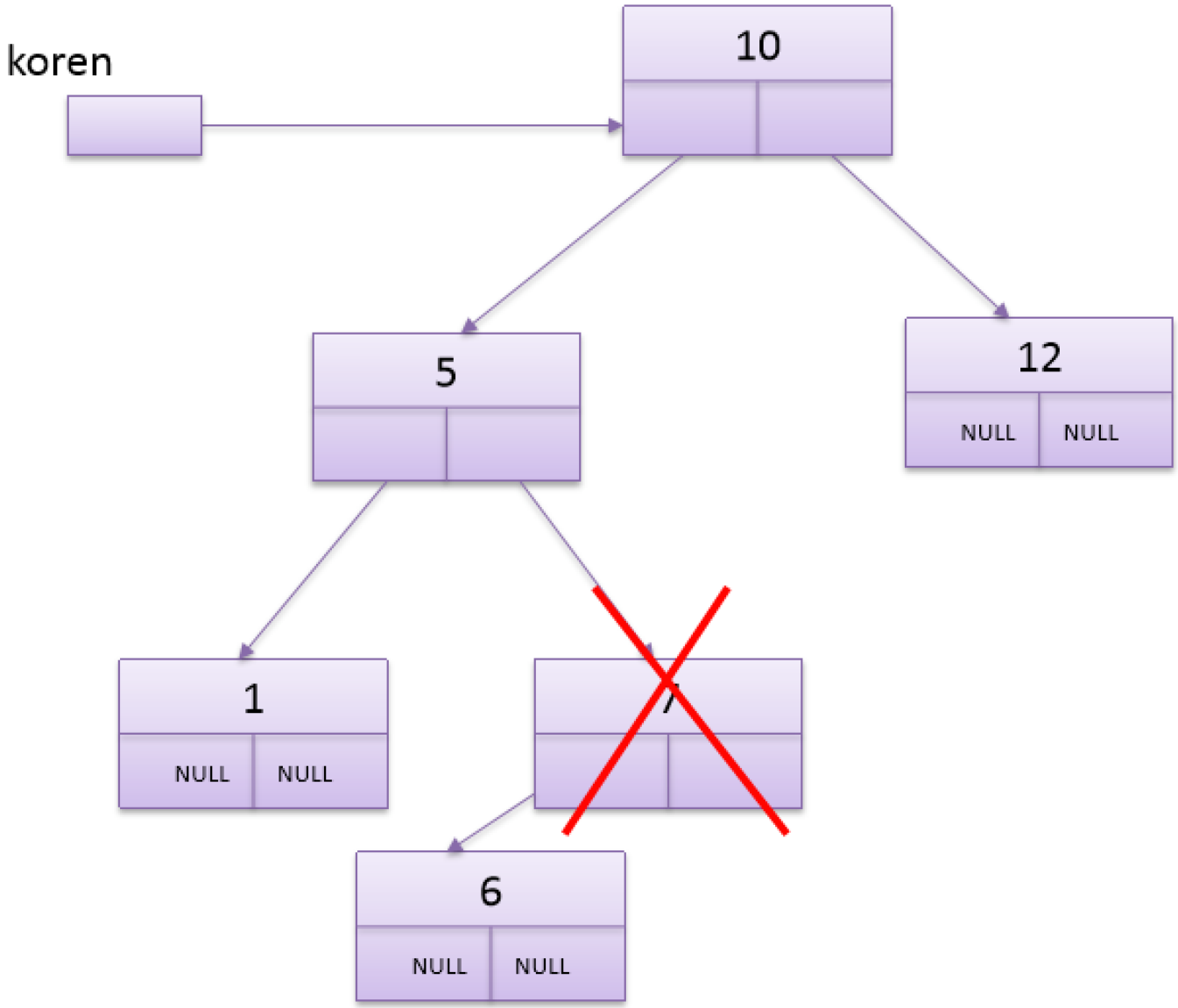
- Krene se od korena i gleda se da li je sadržaj traženog čvora manji ili veći od tekućeg
- Ako je manji, ide se u levo podstablo
- Ako je veći, ide se u desno podstablo
- Ako je jednak, obriše se čvor i obavi se prevezivanje
  - ako je čvor koji se briše list, samo se obriše, a onaj čvor koji je ukazivao na njega se ažurira (NULL)
  - ako čvor koji se briše ima samo jedan podčvor, samo se obriše, a onaj koji je ukazivao na obrisanog se preveže na podčvor
  - ako čvor koji se briše ima oba podčvora, on se obriše, a na njegovo mesto se uveže najmanji element u njegovom desnom podstablu
    - najmanji element koji je uvezan umesto izbrisanog se uklanja sa originalne lokacije i radi se isto prevezivanje


koren
10
5
12
NULL
NULL
1
NULL
NULL
7
6
NULL
NULL
9
NULL
NULL

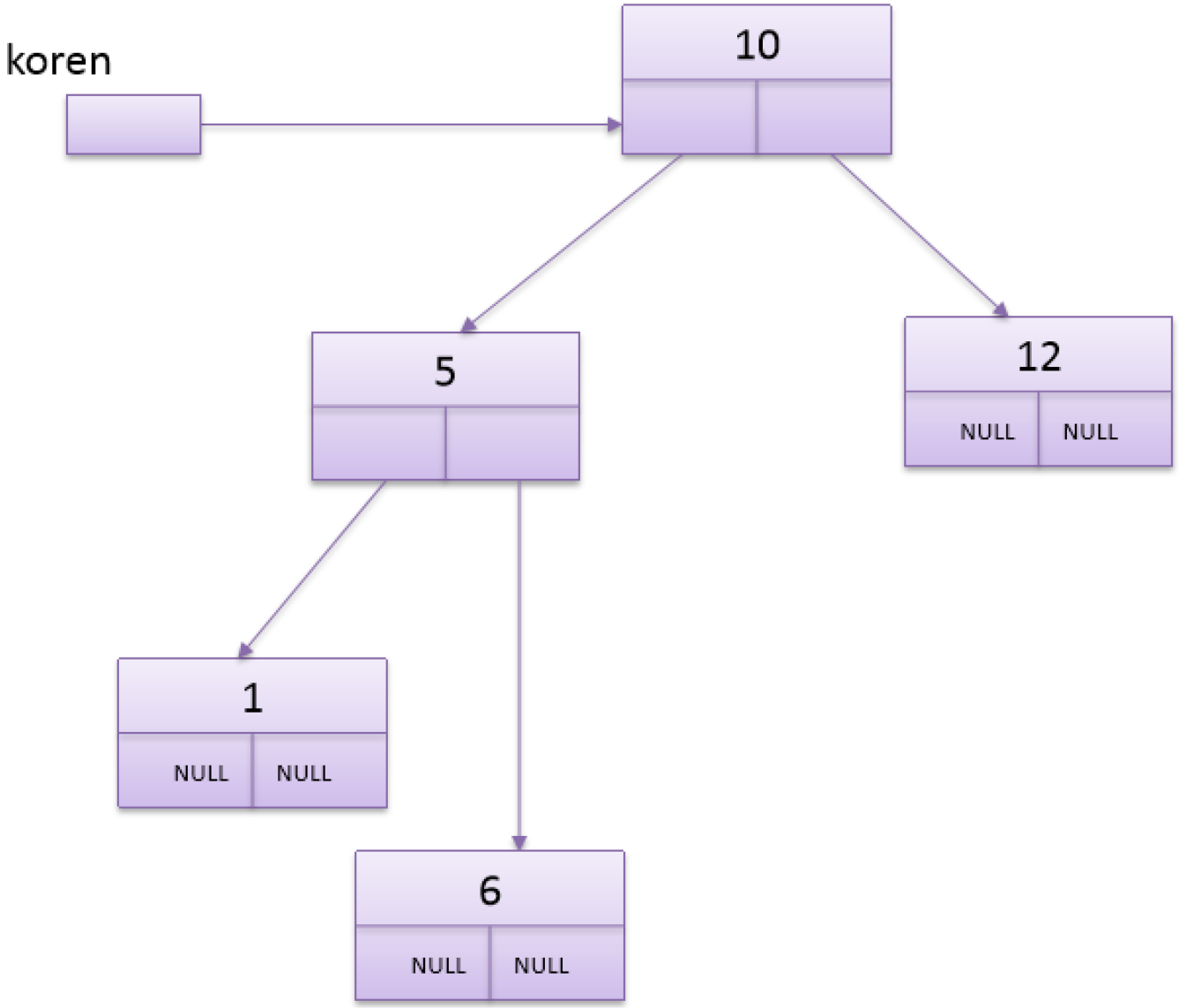
5) Brisanje lista



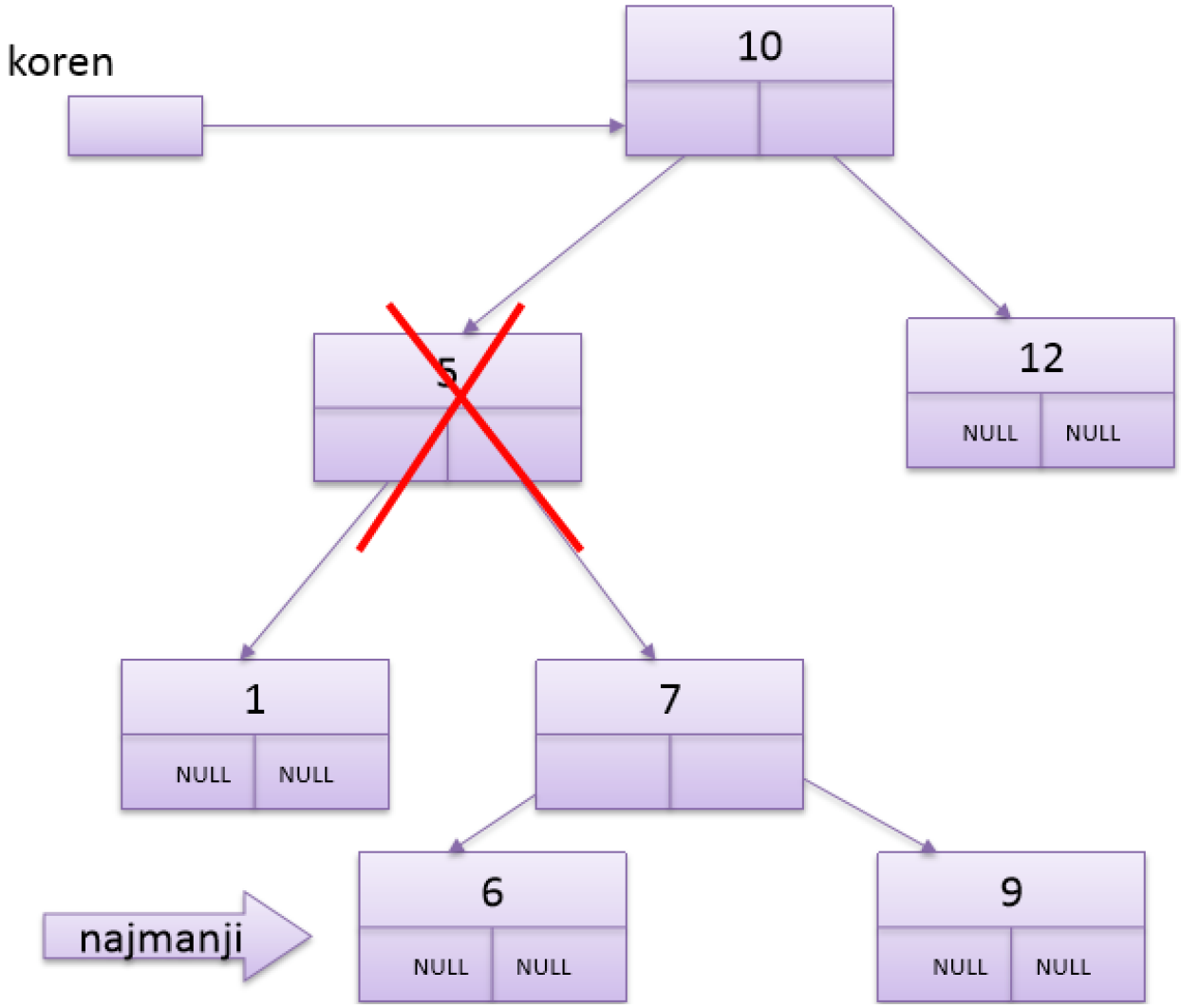
5) Brisanje čvora sa jednim podstablom



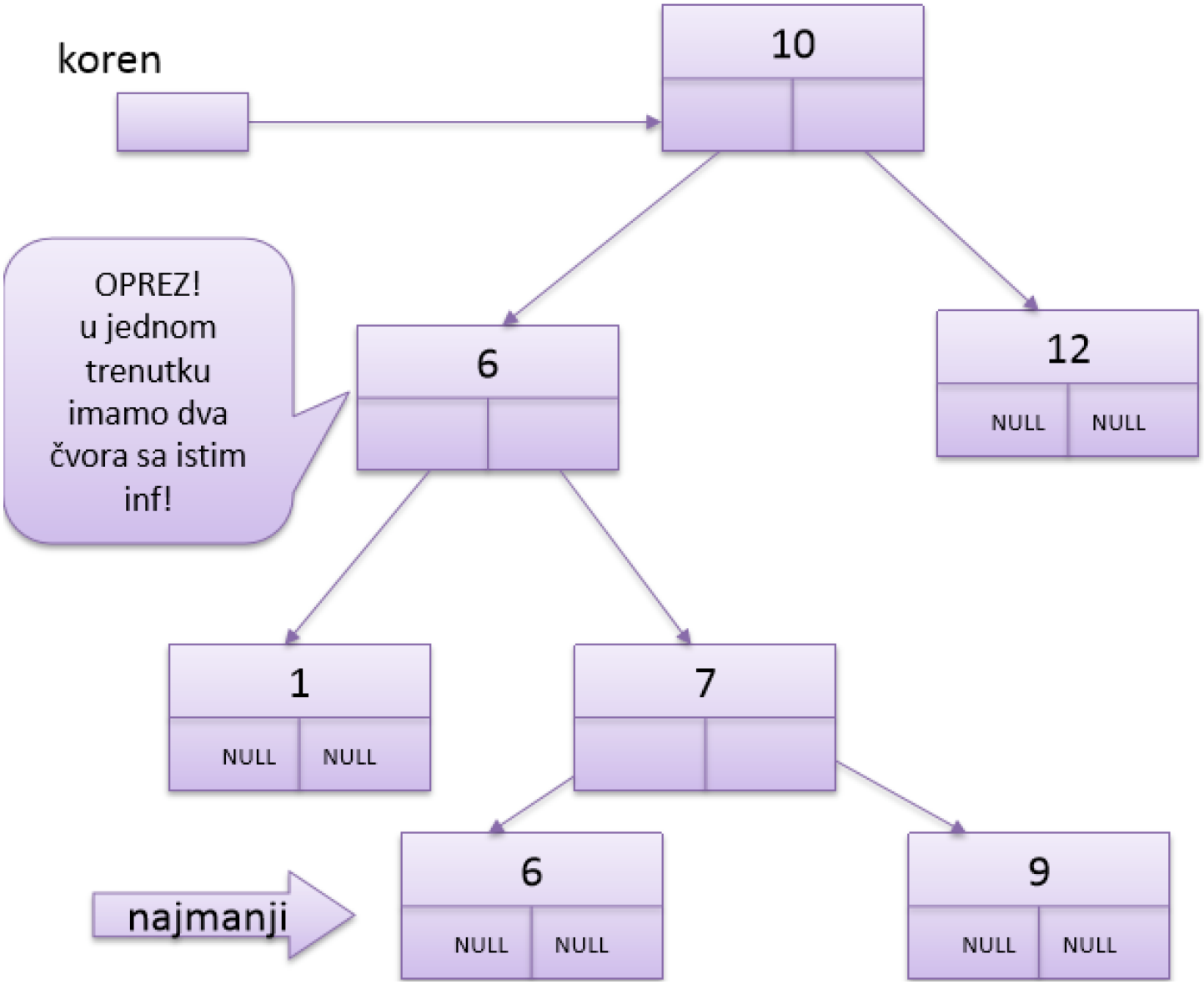
5) Brisanje čvora sa jednim podstablom



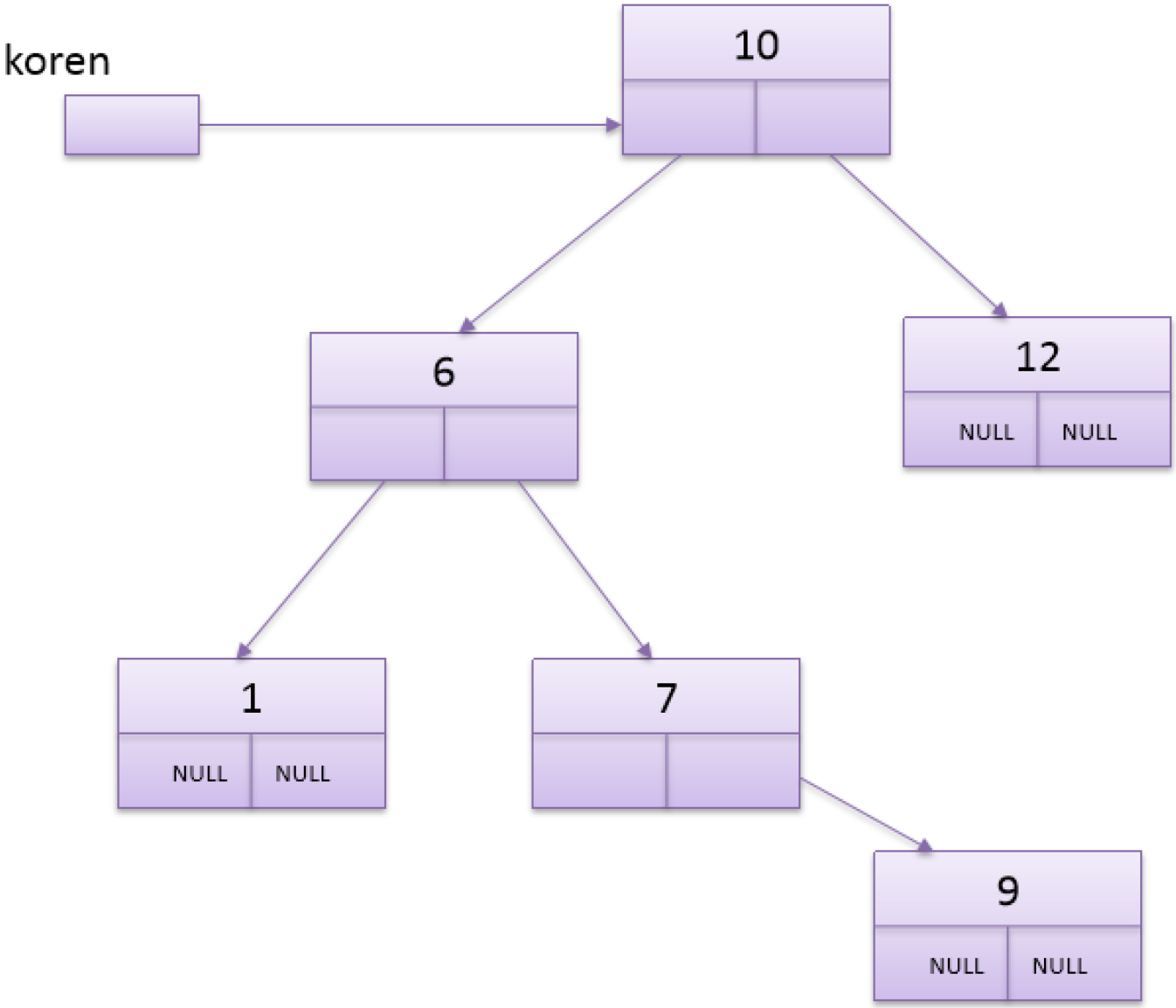
5) Brisanje čvora sa dva podstabla



5) Brisanje čvora sa dva podstabla



5) Brisanje čvora sa dva podstabla



4) Brisanje elementa iz stabla:

```
int list(BCVOR *cvor){
   if(cvor->levi == NULL && cvor->desni == NULL)
     return 1;
   return 0;
}

BCVOR* min(BCVOR *cvor) {

   if(cvor == NULL) {
       return NULL;
   }

   if(cvor->levi == NULL)
       return cvor;
   else{
       BCVOR* m = min(cvor->levi);
       return m;
   }
}
```

4) Brisanje elementa iz stabla:

```
BCVOR* brisiS(BCVOR *cvor, int br){

   BCVOR *pom;
   if(cvor == NULL){
       return NULL;
   }

   if(br < cvor->inf){
       cvor->levi =  brisiS(cvor->levi, br);
       return cvor;
   }
   else if(br > cvor->inf){
       cvor->desni = brisiS(cvor->desni, br);
       return cvor;
   }
```

4) Brisanje elementa iz stabla:

```
}
  else{ // br == cvor->inf, brisi cvor

      // ako je cvor list
      if(list(cvor) == 1) {
        free(cvor);
        return NULL;
      } else {// ako nije list, tj nisu oba pokazivaca NULL

         if(cvor->levi == NULL) { // ako je levi NULL vrati desni
            pom = cvor->desni;
            free(cvor);
            return pom;
         }else if(cvor->desni == NULL) { // ako je desni NULL vrati levi
            pom = cvor->levi;
            free(cvor);
            return pom;
         }else{ // ako su oba razlicita od NULL
            pom = min(cvor->desni);
            cvor->inf = pom->inf;
            cvor->desni = brisiS(cvor->desni, pom->inf);
            return cvor;
         }

       }
  }
}
```

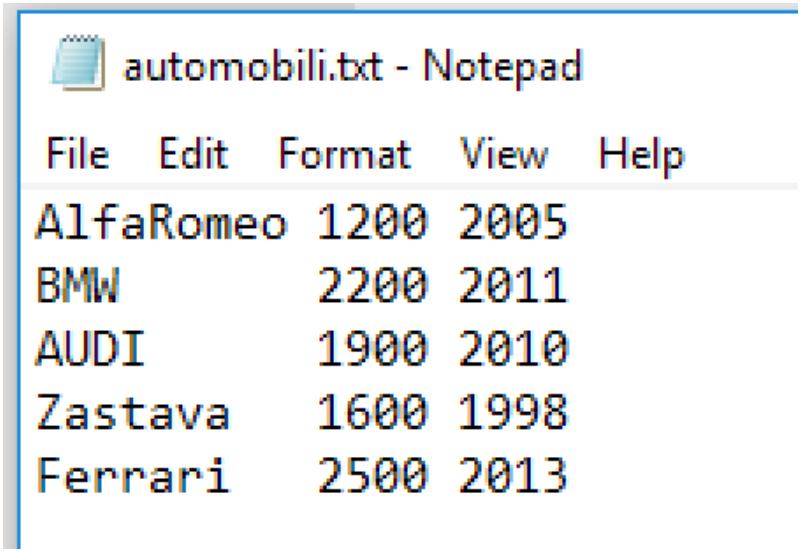
6) Brisanje stabla

- Krene se od korena i briše se levo i desno podstablo, pa onda koren

```
void brisanje_stabla(BCVOR *cvor) {
    if (cvor) // ili if(cvor != NULL)
    {
        brisanje_stabla(cvor->levi);
        brisanje_stabla(cvor->desni);
        free(cvor);
    }
}
```

Zadatak

- Iz ulazne datoteke učitati podatke o automobilima u binarno stablo traženja.
- Raspored slogova po čvorovima vrši se prema kubikaži vozila.
- Za unetu kubikažu vozila pronaći najnovije vozilo sa kubikažom ne većom od zadate.



Zadatak

- Napisati C program za evidenciju studenata.
  - Svaki student je opisan:
    - imenom (do 20 karaktera),
    - prezimenom (do 20 karaktera),
    - brojem indeksa (do 10 karaktera) i
    - godinom upisa studija.
  - Program treba da obezbedi sledeće:
    - Unos podataka o studentima u niz
    - Sortiranje niza po broju indeksa
    - Formiranje spregnutog balansiranog binarnog stabla na osnovu sortiranog niza, pri čemu se raspoređivanje slogova po čvorovima vrši prema broju indeksa
    - Oslobađanje lokacija zauzetih za potrebe niza i stabla

Zadatak

- Proširiti program iz prethodnog Zadatka dodavanjem mogućnosti
  - serijalizacije binarnog stabla u binarnu datoteku
  - deserijalizacije binarnog stabla iz binarne datoteke
  - snimanja grafičkog prikaza binarnog stabla u tekstualnu datoteku
    - u čvorovima prikazati broj indeksa

```
17000
|--------------------|
16500                17500
|----------|         |----------|
16300      16700     17200      17700
```

Zadatak

Zadata je tekstualna datoteka **spisak.txt** koja sadrži spisak imena i prezimena. U svakom redu datoteke nalazi se tačno jedno ime i prezime međusobno razdvojeni prazninama. Datoteka je sortirana po abecednom kriterijumu.

U datoteci **brisi.txt** nalazi se u istom formatu kao u datoteci **spisak.txt** spisak imena i prezimena koje treba izbaciti iz datoteke **spisak.txt**, i formirati datoteku **izlaz.txt.**

Zadatak resiti pomoću liste.

Zadatak

Napisati C program koji iz ulazne datoteke reci.txt čita sekvencijalno reči (svaka reč je u posebnom redu) a potom se te reči ispisuju u obrnutom redosledu u izlaznu datoteku obrnuto.txt. Zadatak rešiti korištenjem steka.