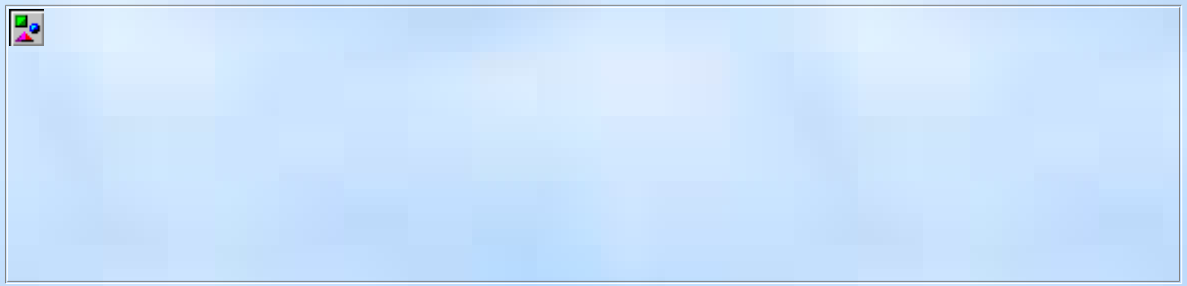
Programiranje u C-u

Pokazivači i datoteke

Pointers & Files

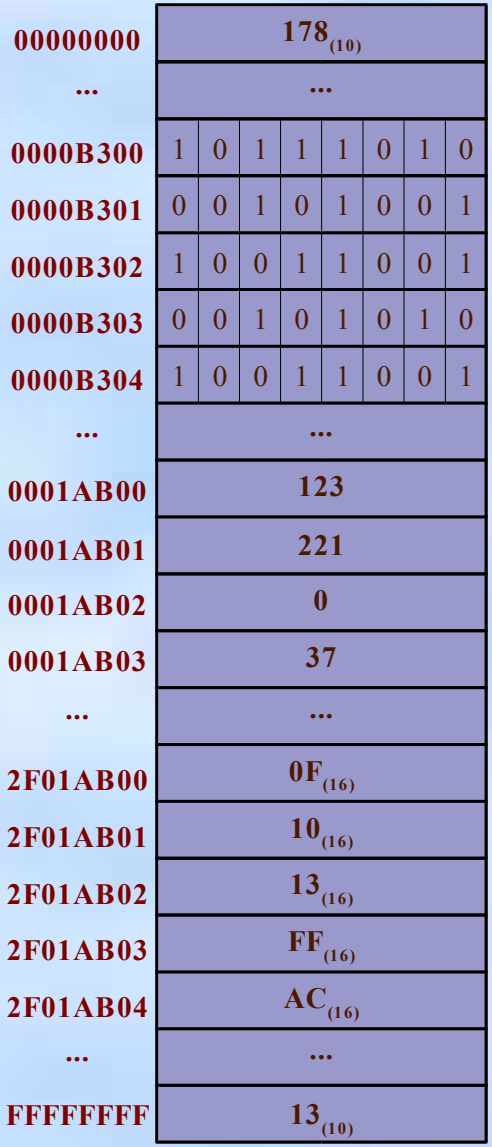
Uvod

- znatno povećavaju mogućnosti jezika
- olakšavaju rješavanje mnogih kompleksnih problema
- omogućuju dinamičko rukovanje memorijom – ušteda memorije
- neispravno korištenje je česti uzrok pogreške u radu aplikacije



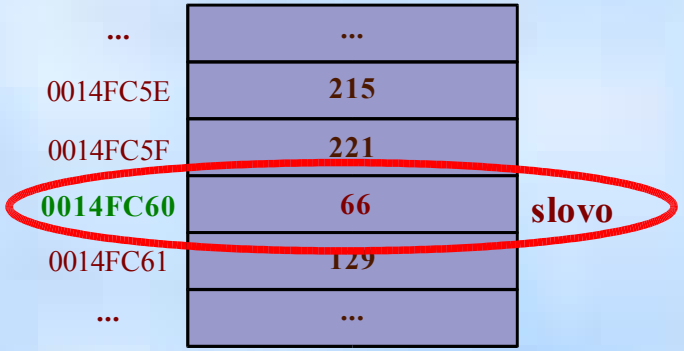
Kako izgleda memorija računala?

- krajnje pojednostavljeno:
  - Niz bajta
  - U svakom je bajtu u svakom trenutku zapisan neki podatak – broj od 0 – $255_{(10)}$, 00 – $FF_{(16)}$, 00000000 – $11111111_{(2)}$
  - Svaki bajt ima svoju adresu
  - Adrese počinju od 00000000 i završavaju na FFFFFFFF



Što je deklaracija varijabli?

- nakon deklaracije:
  char slovo;



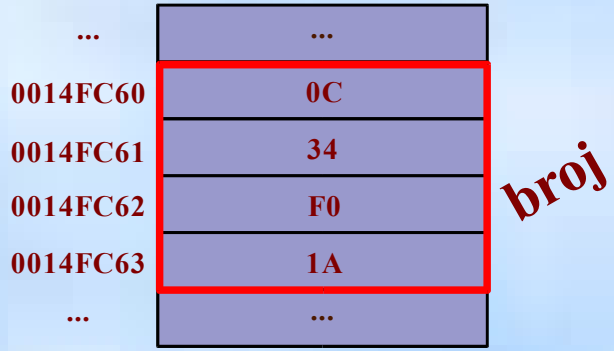
program će:

- Rezervirati na stogu 1B
- "Zapamtiti" adresu rezervirane lokacije – npr. 0014FC60
- Svaki puta kada se koristi varijabla *slovo*, koristiti podatak s te adrese

- printf("%c", slovo); ispisuje: A
- slovo='B'; na adresu te varijable upisuje ASCII kod od B (66)

Što je deklaracija varijabli?

- nakon deklaracije:
  int broj;



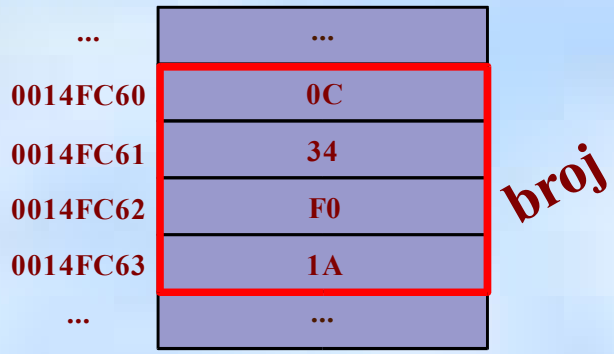
program će:

- Rezervirati na stogu 4B
- "Zapamtiti" adresu prvog bajta rezervirane lokacije – npr. 0014FC60
- Svaki puta kada se koristi varijabla *broj*, koristiti podatak koji POČINJE na toj adresi

- printf("%d", broj); ispisuje: 31
- broj=0x1AF0340C; na adresu te varijable upisuje broj 451949580

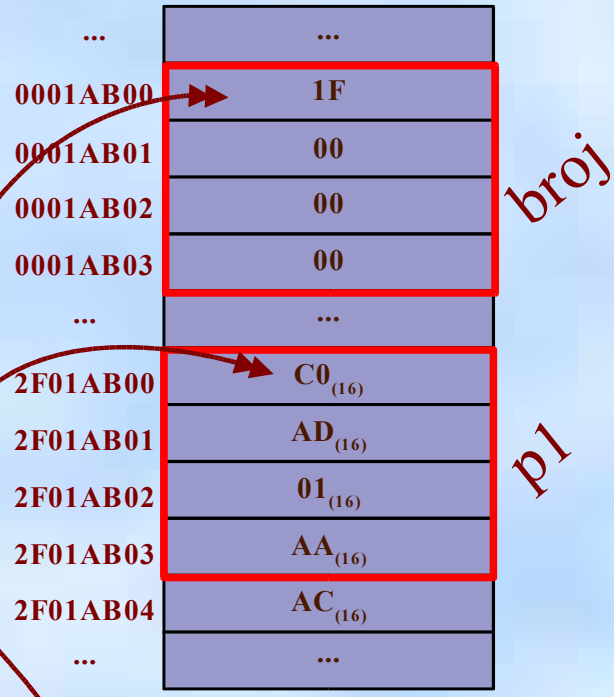
Što je deklaracija varijabli?

- rezerviranje jednog ili više memorijskih mjesta za spremanje vrijednosti varijable koju se deklarira
- pridruživanje imena deklarirane varijable tim memorijskim lokacijama



Razlike pokazivača i “običnih” varijabli

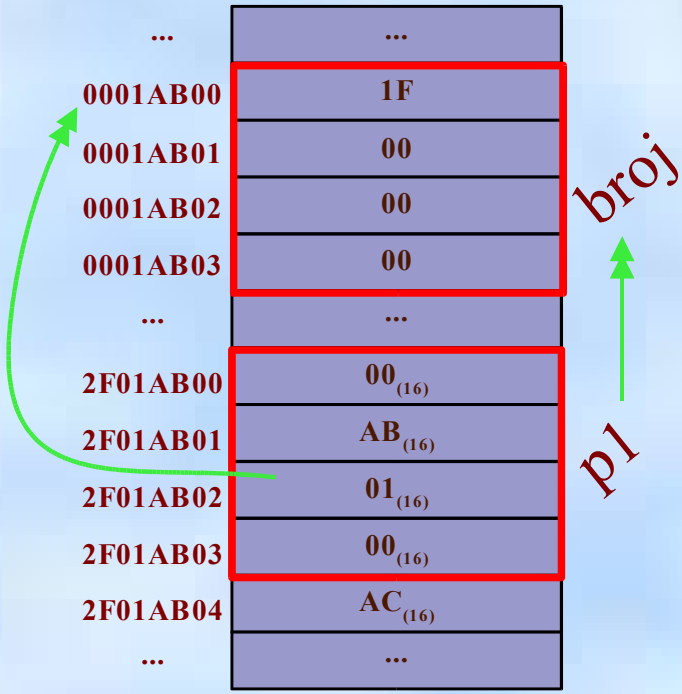
- obična varijabla može sadržavati bilo koju vrijednost – cijeli ili decimalni broj, znak, strukturu i sl.
- pokazivači sadrže isključivo adresu neke druge varijable (obične ili nekog drugog pokazivača)



- na adresi 0001AB00 nalazi se broj 31
- na adresi 2F01AB00 nalazi se adresa: AA01ADC0

Razlike pokazivača i "običnih" varijabli

- pokazivač je velik koliko to zahtijeva njegov sadržaj: kako je za punu adresu potrebno 32 bita, veličina pokazivača je 4B
- ako pokazivač sadrži adresu neke varijable, tada se kaže da on pokazuje na tu varijablu
  - kako je u *p1* sadržano $0001AB00_{(16)}$, a to je upravo adresa varijable *broj*, može se reći da *p1* pokazuje na varijablu *broj*

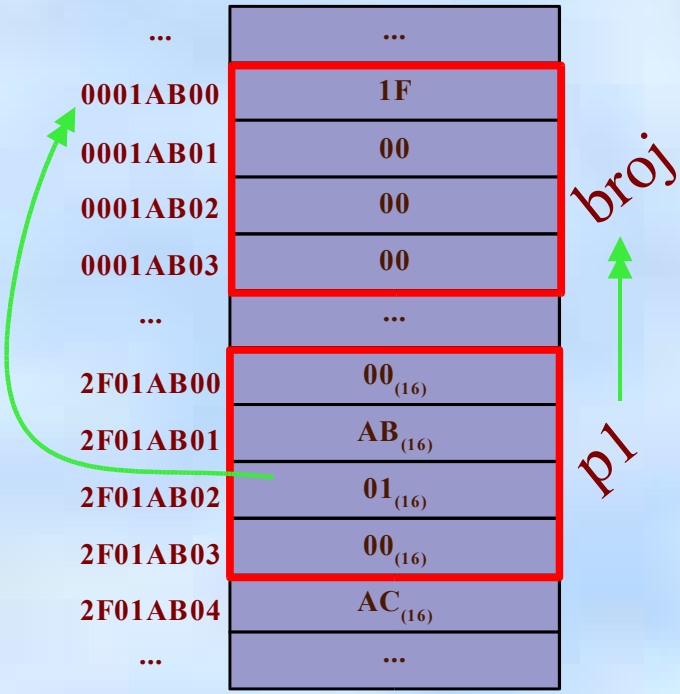


Deklaracija pokazivača

- ispred imena varijable stavlja se znak *
- int *pbroj; deklarira pokazivač na varijablu tipa int
- int *pbroj, broj1; deklarira pokazivač na varijablu tipa int i *običnu* int varijablu *broj1*
- char *p1, *p2; deklarira dva pokazivača na varijablu tipa char

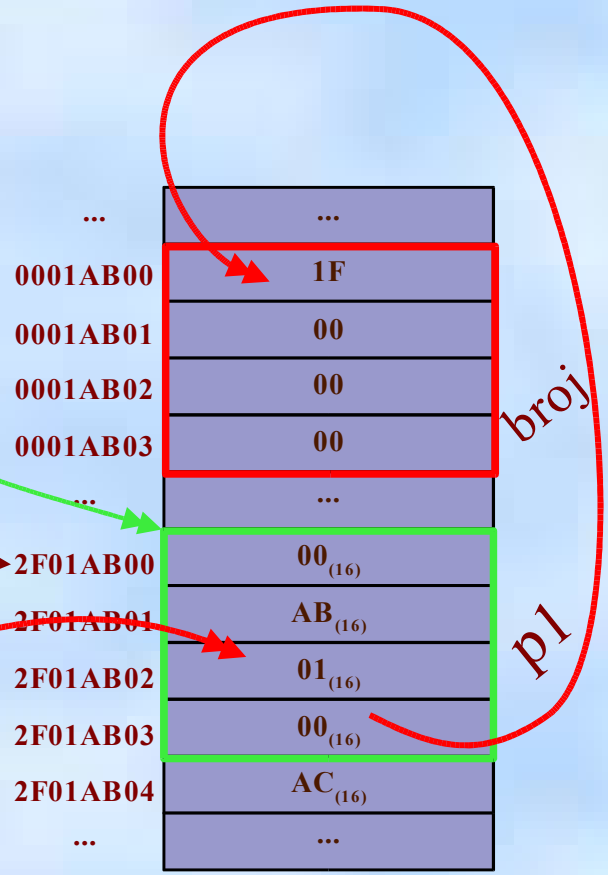
Operatori

- operator adrese: & (eng. ampersand)
  - vraća adresu operanda
  - &broj vraća vrijednost 0001AB00
  - &p1 vraća vrijednost 2F01AB00
- operator dereferenciranja: * (eng. asterisk)
  - vraća sadržaj memorijske lokacije koja je zapisana u pokazivaču
  - *p1 vraća vrijednost $31_{(10)}$, odnosno $1F_{(16)}$



Operatori

- za pokazivač *p1* vrijedi:
  - p1 je jednako 0001AB00
  - &p1 je jednako 2F01AB00
  - *p1 je jednako 0000001F
- za varijablu *broj* vrijedi:
  - broj je jednako 0000001F
  - *broj nema smisla – pogreška kod prevođenja
  - &broj je jednako 0001AB00

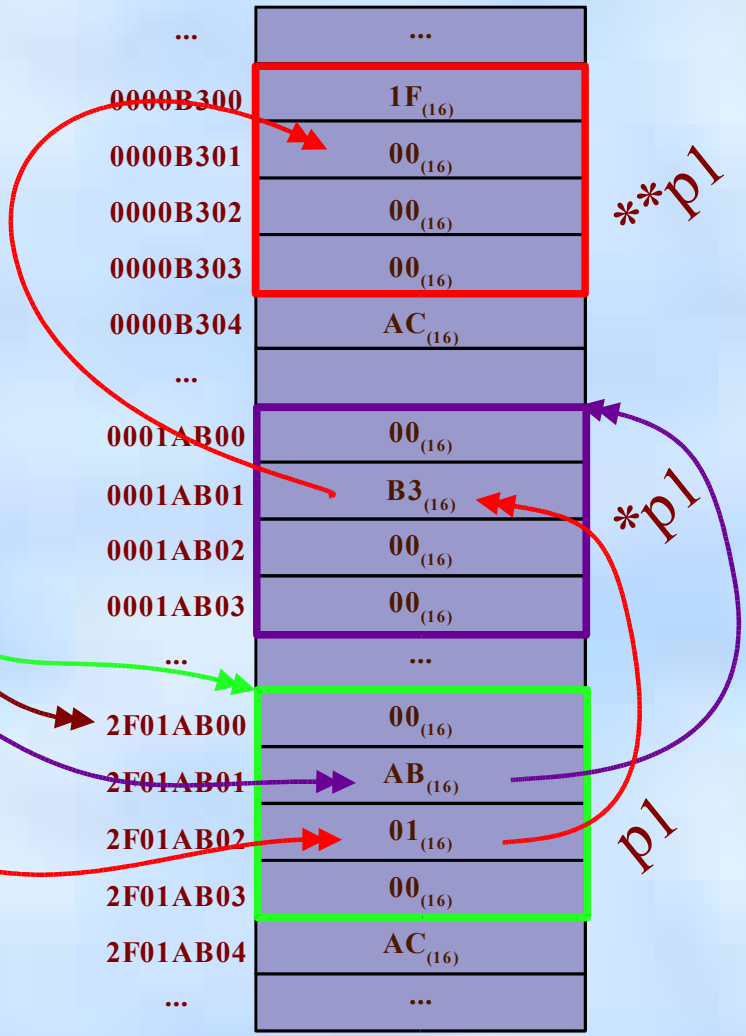


Tipovi pokazivača

- svaki pokazivač ima neki tip
- tip je određen podatkom na kojeg pokazivač pokazuje – prema deklaraciji
  - int *p1; — deklarira pokazivač tipa integer
  - float *p2; — deklarira pokazivač tipa float
  - unsigned short *p3; — deklarira pokazivač tipa unsigned short integer
- NULL je pokazivačka konstanta koja pokazuje na adresu 0
  - NULL je uvijek “dostupan”, a definiran je u header datotekama (u različitoj datoteci, ovisno o oper. sustavu i/ili razvojnoj okolini)

Dvostruki pokazivači

- može se deklarirati i pokazivač na pokazivač:
  - int **p1;
  - &p1 je jednako 2F01AB00
  - p1 je jednako 0001AB00
  - *p1je jednako 0000B300
  - **p1 je jednako 1F



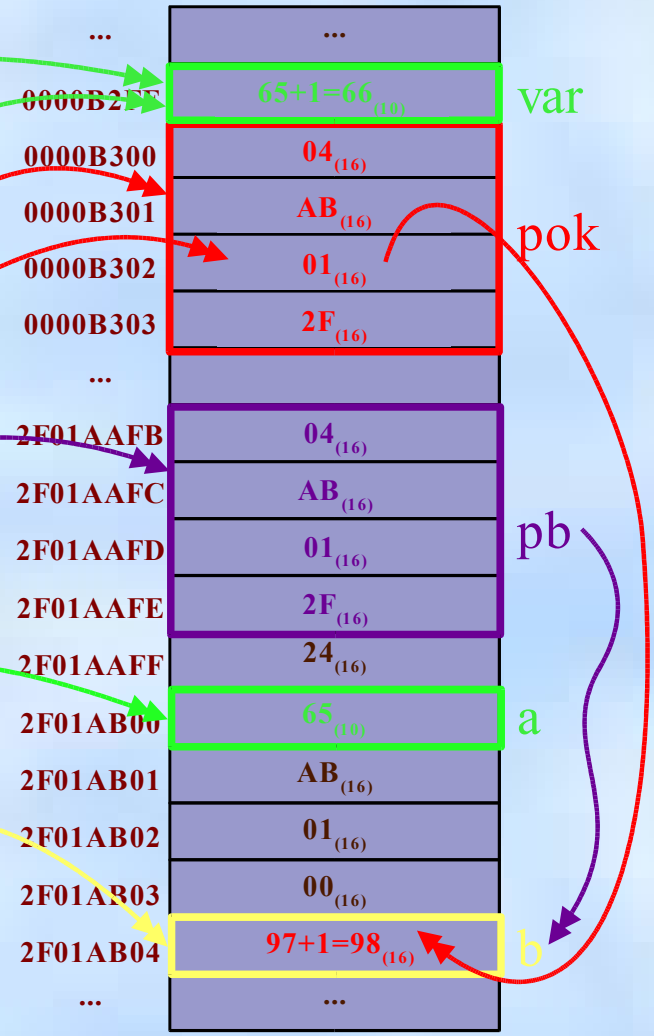
Trostruki, četverostruki, ...

- int ***p3;
- int ****p4;
- int *****p5;
- ...
- int**********...**********pn
- iznad 4. reda se u praksi relativno rijetko koriste

Prijenos argumenata funkcijama

- dva načina prijenosa:
  - pass by value
  - pass by reference

```
void funkcija (char var, char *pok) {
        var ++;
        (*pok) ++;
}
void main() {
        char a=65, b=97;
        char *pb=&b;

        funkcija(a, &b);
        funkcija(a, pb);
}
```



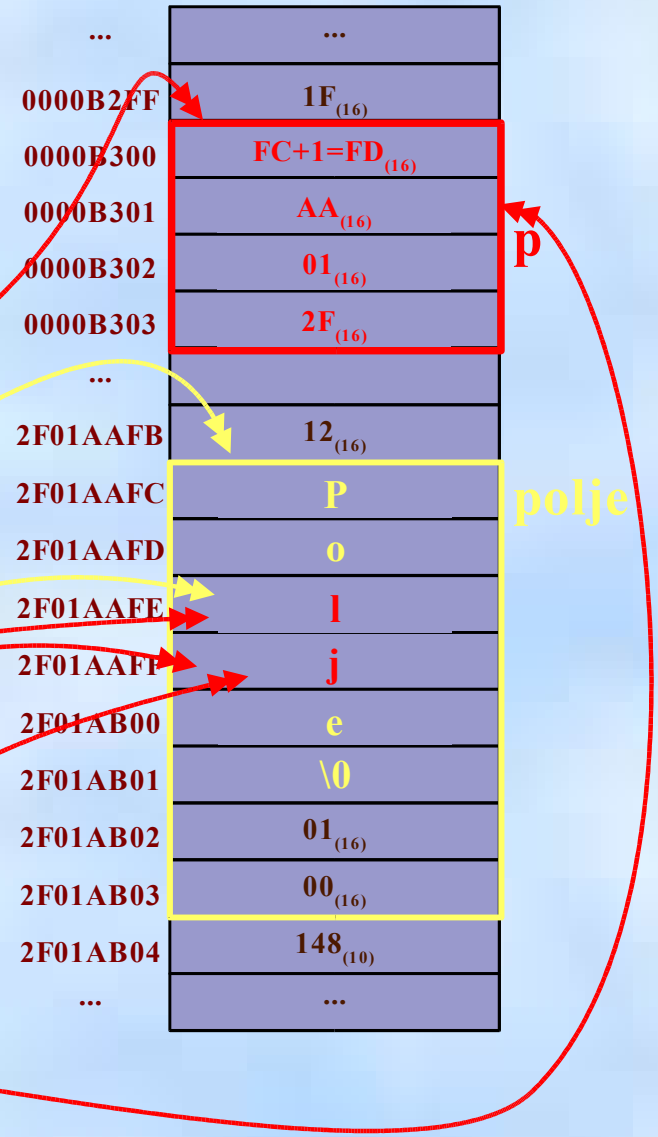
Pokazivači i polja

- usko su povezani
  - ime svakog deklariranog polja može se koristiti kao pokazivač
  - svaki se pokazivač može koristiti kao polje

```
void main() {
      char      polje[8]=”Polje”;
      char      *p;
      p=&polje[0];
      p=polje;
      polje[2]='t';
      p[3]='i';
      *(p+2)='l';
      p++;
      *(p+2)='j';
      polje=p;
}
```

ekvivalentni izrazi

polje je pokazivačka konstanta i ne može joj se mijenjati vrijednost!!!



Pokazivači i polja – ekvivalencije

- kod pokazivača i jednodimenzionalnih polja vrijede sljedeće ekvivalencije

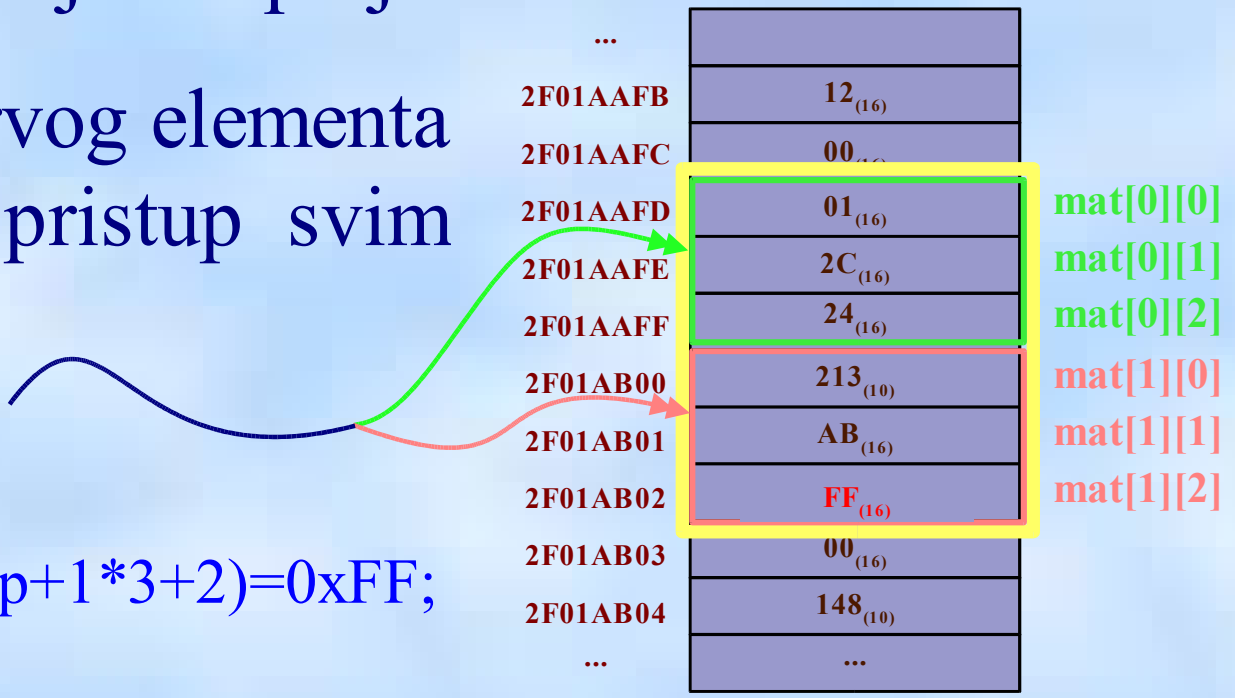
```
int a[10];
int *p;
```

Izraz	Ekvivalentni izraz
&a[0]	a
&a[3]	a+3
*p	p[0]
*(p+1)	p[1]
*(p+2)	p[2]
p=a	p=&a[0]
p++	p=p+1
a++ je zabranjeno	

Dvodimenzionalna polja (matrice)

- memorija za dvodimenzionalna polja alocira se u jednom neprekinutom bloku
- u memoriju se smješta jedan po jedan redak matrice
- poznavanje adrese prvog elementa matrice omogućava pristup svim ostalim elementima

```
char mat[2][3];
int *p=mat;
p[1*3+2]=0xFF; //ili *(p+1*3+2)=0xFF;
```



- općenito: p [željeni_redak*broj_stupaca + željeni_stupac]=...

Inicijalizacija 2D polja

- inicijalizacija elemenata se vrši redom – od prvog elementa prema posljednjem
- neinicijalizirani se elementi postavljaju na vrijednost 0
- inicijalizacijski elementi svakog "redka" se mogu odvojiti zagradama {}

```
char mat[2][3]={0, 1, 2, 3, 4};

char mat[2][3]={ {10, 20}, {30} };
```

“Prolazak” kroz sve elemente 2D polja

```
for(i=0; i<broj_redaka; i++) {
    for(j=0; j<broj_stupaca; j++) {
        ...mat[i][j]...

        - suma svih elemenata:
            suma += mat[i][j];

        - traženje najvećeg elementa
            if(mat[i][j] > max_el) {...

        - traženje najvećeg elementa dijagonale
            if(i == j) {
                if(mat[i][j] > max_diag) {...

    }
}
```

Dinamičko alociranje memorije

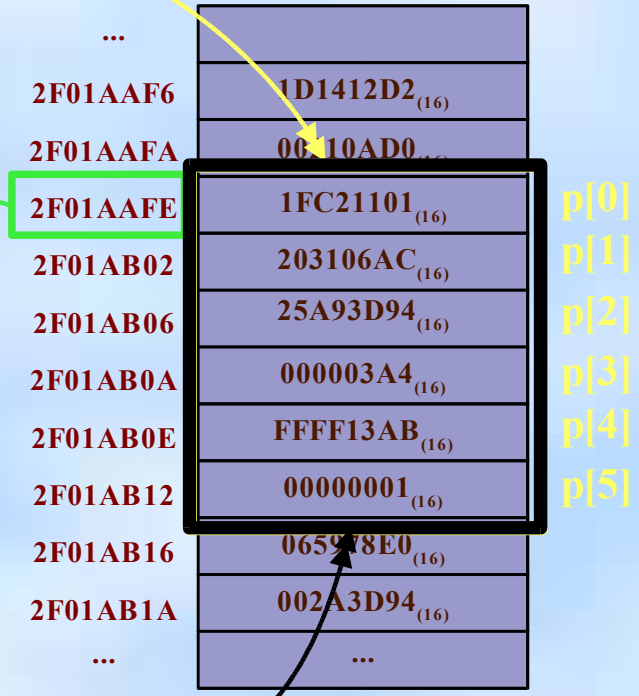
- nije potrebno znati veličine polja u trenutku pisanja programa – efikasno iskorištavanje memorijskih resursa
- statički alocirana memorija (varijable) “traje” koliko i funkcija u kojoj je alocirana
- dinamički alocirana memorija traje dok se funkcijom *free* eksplicite ne oslobodi rezervirani prostor
  - omogućava alociranje memorije unutar bilo koje funkcije, za polje/varijablu/strukturu/... koja se može koristiti unutar bilo koje druge funkcije

Dinamičko alociranje memorije – funkcije

- void *malloc(unsigned int *size*);
  - alocira memorijski blok veličine *size* bajta
- void *realloc(void **memblock*, unsigned int *size*);
  - mijenja veličinu rezerviranog bloka memorije na koji pokazuje *memblock* na veličinu *size*
  - sadržaj memorijskog bloka ostaje sačuvan
  - novi se blok može i ne mora nalaziti na istoj lokaciji
- *malloc* i *realloc* u slučaju pogreške vraćaju NULL
- void free(void **memblock*);
  - oslobađa rezervirani blok memorije na koji pokazuje *memblock*

Dinamičko alociranje memorije – primjer

```
int *alociraj(int velicina)
{
        int             *p;
        p = (int *) malloc( sizeof(int) * velicina);
        return p;
}

void main() {
        int             size, *polje;
        int             i;
        printf("Upisi broj clanova polja: ");
        scanf("%d", &size);
        polje = alociraj( size );
        polje = (int *) realloc(polje, sizeof(int) * size * 2);
        for(i=0; i<size * 2; i++) {
                printf("\nUpisi %d. clan polja: ", i+1);
                scanf("%d", &polje[i]);
        }
        free(polje);
}
```

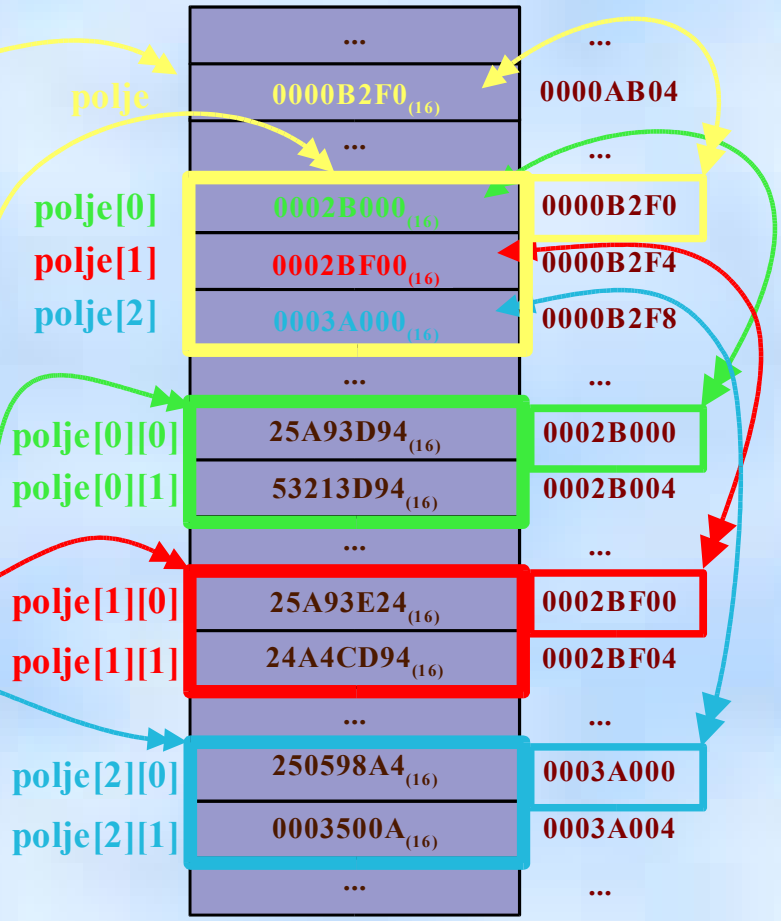


Dinamički alocirana 2D polja

- mogu biti u formi matrice – dimenzija MxN
- svaki redak može imati proizvoljni broj stupaca
- cijelo polje nije smješteno u jednom neprekinutom memorijskom bloku, već se svaki redak nalazi na slučajnim memorijskim lokacijama
- za kreiranje 2D dinamičkog polja, potrebno je:
  a) alocirati memoriju za pokazivače na svaki od redaka
  b) alocirati memoriju za svaki redak te početnu adresu redka smjestiti u odgovarajući pokazivač u polju pokazivača

Dinamički alocirana 2D polja

```
void main() {
        int             br_redaka, br_stupaca;
        int             i, j;
        int             **polje;

        printf("Upisi broj redaka i stupaca polja: ");
        scanf("%d %d", &br_redaka, &br_stupaca);
        polje = (int **) malloc(sizeof(int *) * br_redaka);
        for(i=0; i<br_redaka; i++) {
                polje[i] = (int *) malloc(sizeof(int) * br_stupaca);
        }
        for(i=0; i<br_redaka; i++) {
                for(j=0; j<br_stupaca; j++) {
                        printf("\nUpisi element [%d][%d]: ", i,  j);
                        scanf("%d", &polje[i][j]); // ili *(polje+i) + j
                }
        }
        for(i=0; i<br_redaka; i++) {
                free(polje[i]);         // ili *(polje+i)
        }
        free(polje);
}
```



3 redka i 2 stupca

Dinamički alocirana 2D polja

- korištenjem funkcije *realloc* moguće je dodati mjesto za pokazivač na još jedan redak ili povećati broj stupaca bilo kojeg retka

```
int  **p;
```

Izraz	Ekvivalentni izraz
&p[i][j]	*(p+i)+j
&p[i][0]	*(p+i)
p[i][j]	*(*(p+i)+j)
p[i][0]	**(p+i)

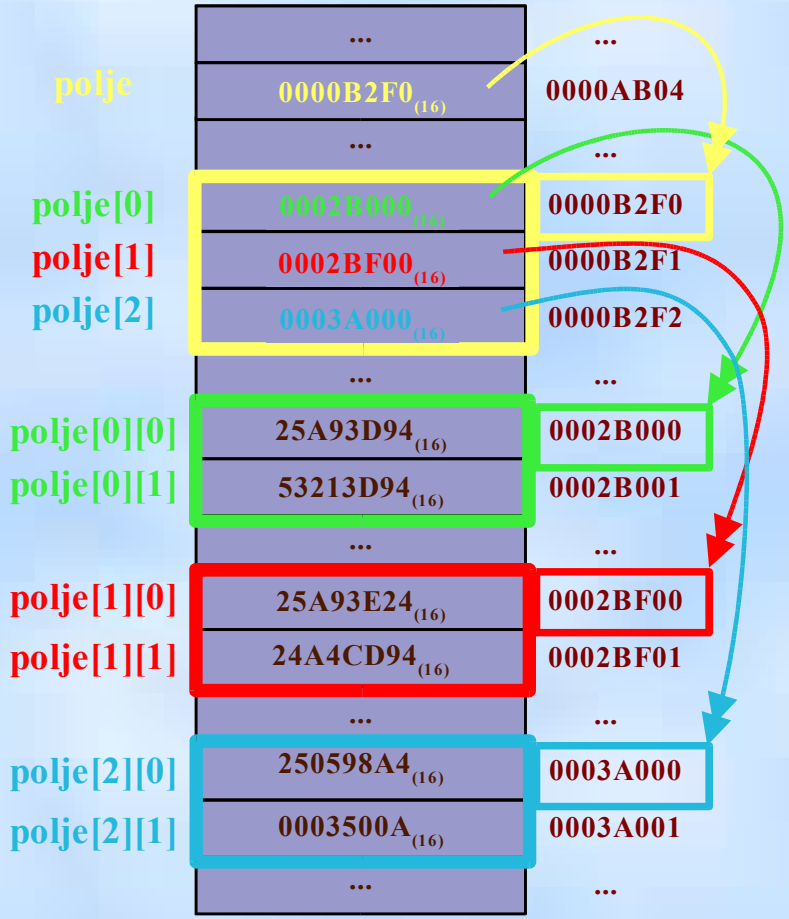
```
int polje[10][20];
int  **p=(int **) polje;
```

Izraz	***NE***-ekvivalentni izraz
&polje[2][3]	&p[2][3]
&polje[2][3]	*(p+2)+3
polje[2][3]	p[2][3]
polje[2][3]	*(*(p+2)+3)

Statička i dinamička 2D polja

char polje[3][2];

char **polje; // za 3 redka i 2 stupca



Dinamička 2D polja i funkcije

```
int **alociraj(int redaka, int stupaca)
{
        int         **polje;
        int         i;
        polje = (int **) malloc(sizeof(int *) * redaka);
        for(i=0; i<redaka; i++) {
                polje[i] = (int *) malloc(sizeof(int) * stupaca);
        }
        return polje;
}

void ispisi(int **array, int redaka, int stupaca)
{
        int         i, j;
        for(i=0; i<redaka; i++) {
                for(j=0; j<stupaca; j++) {
                        printf("%d\t", array[i][j]);
                }
                printf("\n");
        }
}
```

ŠTO NEDOSTAJE???

```
void main()
{
        int         **p;
        int         i, j, a, b, tmp;
        int         redaka, stupaca;
        printf("Upisi broj redaka i stupaca i
                granice intervala: ");
        scanf("%d %d %d %d", &redaka,
                &stupaca, &a, &b);
        p = alociraj(redaka, stupaca);
        srand(time(NULL));
        for(i=0; i<redaka; i++) {
                for(j=0; j<stupaca; j++) {
                        tmp=(int)((float)rand()/
                            RAND_MAX*(b-a)+a+0.5);
                        *(*(p+i)+j) = tmp;
                        printf("\nElement [%d][%d]
                            je %d: ", i, j, tmp);
                }
        }
        ispisi(p, redaka, stupaca);
}
```

Strukture

- predstavljaju novi tip podataka
- deklariraju se korištenjem ključne riječi *struct*
- koriste se za grupiranje varijabli, npr.
  - Marko Marković 111-222 Rijeka
  - Ivo Ivic 222-333 Zagreb
  - ...

```
struct {
        char ime[20];
        char prezime[20];
        char telefon[25];
        char grad[20];
} osoba;
```

Strukture

- deklarira se varijabla *osoba*, s četiri člana: *ime, prezime, telefon* i *grad*
- pojedinom se članu pristupa korištenjem operatora '.': *osoba.ime, osoba.grad, ...*
- tri načina deklariranja

```
struct {
        char ime[20];
        char prezime[20];
        char telefon[25];
        char grad[20];
} osoba1;
```

```
struct osoba{
        char ime[20];
        char prezime[20];
        char telefon[25];
        char grad[20];
};
```

```
struct osoba{
        char ime[20];
        char prezime[20];
        char telefon[25];
        char grad[20];
} osoba1;
```

Strukture - korištenje

```
struct x {
  int a;
  int b;
}var;

void function(struct x);

void  main()
{
  struct x z;

  z.a = 10;
  var.b = 20;
  var.a = var.b + z.a;
  z.a++;
  function(z);

}
```

```
void function(struct x z)
{
  printf("Prvi clan je: %d \n", z.a);
  printf("Drugi clan je: %d \n", z.b);
}
```

Strukture i pokazivači

```
struct x {
  int a;
  int b;
};

void function(struct x *);
void main()
{
  struct x z, *pz;
  pz = &z;
  z.a = 10;
  (*pz).b = ++z.a;

  function(pz);
}
```

```
void function(struct x *pz)
{
  printf("Prvi clan je: %d \n", (*pz).a);
  printf("Drugi clan je: %d \n", pz->b);
}
```

(*pz). i pz-> su ekvivalentni izrazi, a u praksi se zbog jednostavnosti najčešće koristi
pz->

Strukture kao povratni tip funkcije

Napisati funkciju koja vra?a slu?ajno odabranu to?ku unutar zadanog pravokutnika. Zadani pravokutnik nalazi se u prvom kvadrantu koordinatnog sustava (obje koordinate su pozitivne).
Deklarirati i koristiti strukturu "tocka" koja sadrži koordinate to?ke.

```
struct tocka {
    int x;
    int y;
};

struct tocka* nadjiTocku(int x1, int y1, int x2, int y2)
{
    struct tocka* pt;

    pt = (struct tocka*) malloc(sizeof(struct tocka));

    srand((unsigned)time(NULL));
    pt->x = (int)((float) rand() / RAND_MAX * (x2 - x1) + x1 + 0.5);
    pt->y = (int)((float) rand() / RAND_MAX * (y2 - y1) + y1 + 0.5);

    return pt;
}
```

```
//program za provjeru ispravnosti funkcije

void main()
{
  struct tocka* pt;
  pt = nadjiTocku(10, 20, 30, 40);
  printf("x=%d, y=%d\n", pt->x, pt->y);
}
```

Pozivanje programa s parametrima

- main(int argc, char *argv[])
  - *int argc* – broj ulaznih parametara
  - *char *argv[]* – polje stringova – parametara
  - prilikom poziva, parametri/stringovi se odvajaju *space-om*
  - kao prvi se parametar podrazumijeva ime programa

```
#include <stdio.h>

void main(int argc, char *argv[]){
  int i;

  printf("Broj parametara: %d\n", argc);
  for(i=0; i<argc; i++) {
    printf("%s\n", argv[i]);
  }
}
```

uz pozivanje:

```
prog1   /I=marko   /O=+3.5    /L
```

dobiva se ispis:

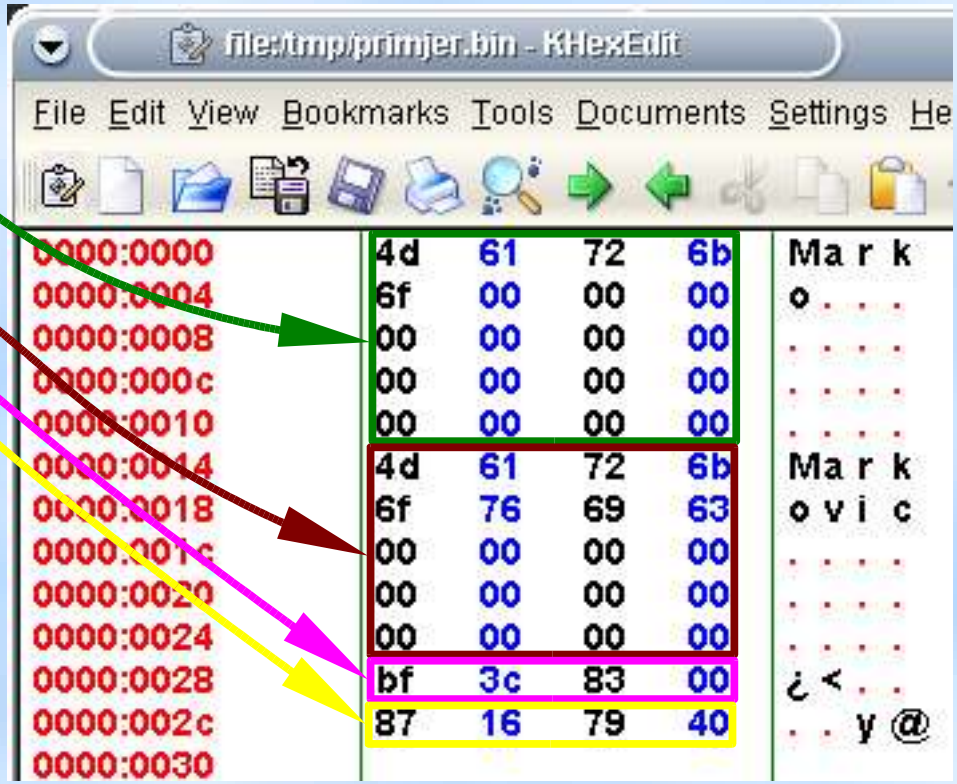
```
Broj parametara: 4
prog1
/I=marko
/O=+3.5
/L
```

Datoteke

- po mogućnosti pristupa dijele se na:
  - direktne – svi su zapisi jednake veličine te se točno zna gdje se svaki pojedini zapis nalazi
  - slijedne – podaci su različitih veličina te se moraju čitati redom svi prethodni podaci, prije nego se dođe do željenog
- po obliku zapisa dijele se na:
  - formatizirane – tekstualna datoteka – svi su podaci zapisani kao znakovi koji se nalaze u određenom formatu – koriste se funkcije *fscanf, fprintf, fgets, fputs, fgetc, fputc*
  - neformatizirane – binarna datoteka – podaci su zapisani minimalnim brojem bajta potrebnih za prikaz određenog tipa varijable – koriste se funkcije *fread* i *fwrite*

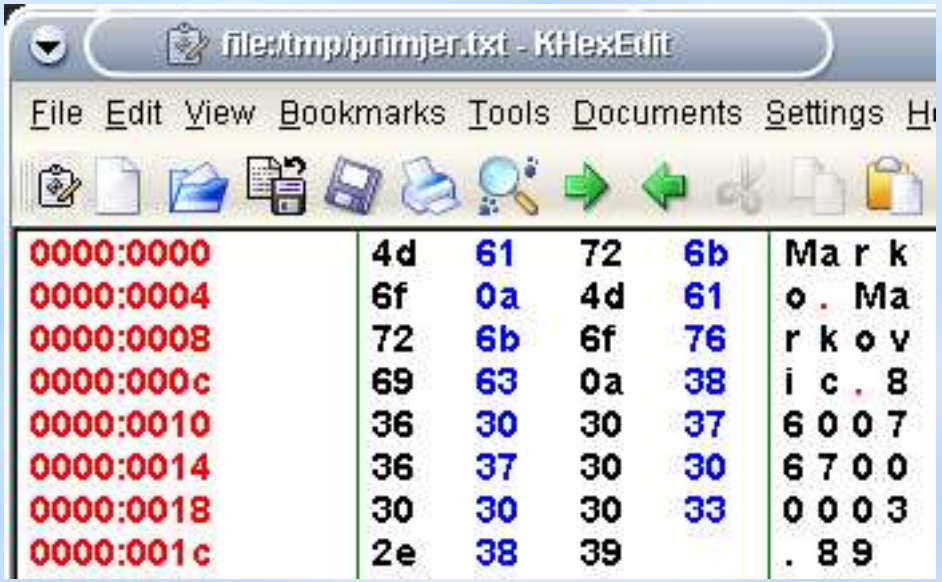
Neformatizirane datoteke - fwrite/fread

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

typedef struct {
        char  ime[19+1];
        char  prezime[19+1];
        int   maticni;
        float ocjena;
}student;

void main()
{
 student citaj, pisi = {"Marko", "Markovic", 8600767,
                        3.892};
 FILE *f;

 if((f=fopen("/tmp/primjer.bin", "w+b")) == NULL) {
   printf("Datoteka se ne moze otvoriti!\n");
   exit(-1);
 }
 fwrite(&pisi, sizeof(pisi), 1, f);
 fseek(f, 0, SEEK_SET);
 fread(&citaj, sizeof(student), 1, f);
 fclose(f);
}
```



Formatizirane datoteke - fprintf/fscanf

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

typedef struct {
        char   ime[19+1];
        char   prezime[19+1];
        int    maticni;
        float  ocjena;
}student;

void main()
{
 student citaj, pisi = {"Marko", "Markovic", 8600767,
                        3.892};

 FILE *f;

 if((f=fopen("/tmp/primjer.txt", "w+t")) == NULL) {
  printf("Datoteka se ne moze otvoriti!\n");
  exit(-1);
 }
 fprintf(f, "%s\n%s\n%7d%09.2f", pisi.ime, pisi.prezime,
        pisi.maticni, pisi.ocjena);
 fseek(f, 0, SEEK_SET);
 fscanf(f, "%19[^\n]\n%19[^\n]%*c%7d%9f", citaj.ime,
        citaj.prezime, &citaj.maticni, &citaj.ocjena);

 fclose(f);
}
```



Formatizirane datoteke - fscanf

fscanf(f, "%19[^\n]\n%19[^\n]%*c%7d%9f", citaj.ime, citaj.prezime, &citaj.maticni, &citaj.ocjena);

%19[^\n] upisuje u *citaj.ime* maksimalno 19 znakova ili do prvog znaka *\n* (na kraj se dodaje '\0')

\n učitava znak *\n* iz datoteke, ali ga nigdje ne zapisuje

%19[^\n] upisuje u *citaj.prezime* maks. 19 znakova ili do prvog znaka *\n* (na kraj se dodaje '\0')

%*c učitava BILO KOJI znak iz datoteke, ali ga nigdje ne zapisuje

%7d upisuje u *citaj.maticni* integer broj od 7 znamenki (promatra se sljedećih 7 znakova)

%9f upisuje u *citaj.ocjena* float broj od 9 znamenki (promatra se sljedećih 9 znakova)

%19[^\n] “Marko” ---> citaj.ime

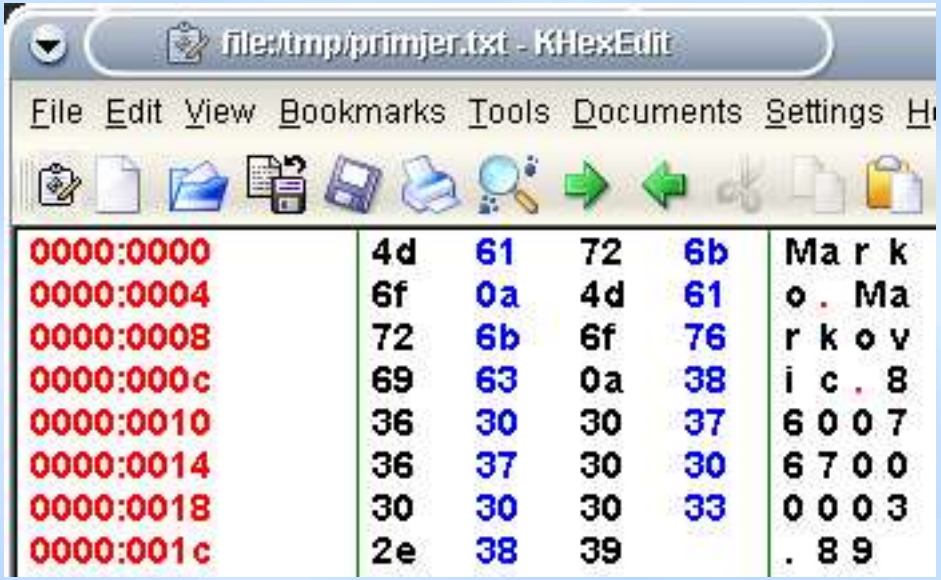
\n \n ---> 

%19[^\n] “Markovic” ---> citaj.prezime

%*c \n ---> 

%7d 8600767 ---> citaj.maticni

%9f 3.89 ---> citaj.ocjena



Pozicioniranje u datoteci

- void rewind(FILE *stream);
  - postavlja indikator pozicije na početak datoteke
  - rewind(f);
- int fseek(FILE *stream, long offset, int whence);
  - postavlja indikator pozicije na *offset* bajta, počevši od *whence*, koji može biti: SEEK_SET, SEEK_CUR i SEEK_END
  - fseek(f, 20, SEEK_SET);
  - fseek(f, -10, SEEK_CUR);
  - fseek(f, 0, SEEK_END);

Pozicioniranje u datoteci

- long ftell(FILE *stream);
  - vraća trenutnu vrijednost indikatora pozicije datoteke
  - pos = ftell(f)
  - za dobivanje veličine datoteke
    - fseek(fp, 0, SEEK_END);
    - size = ftell(fp);
- ako je datoteka otvorena i za pisanje i za čitanje, između tih je operacija potrebno staviti barem jedan poziv funkcijama *rewind* ili *fseek*
  - npr. fseek(f, 0, SEEK_CUR); //ako se ne želi mijenjati pozicija

Programiranje u C-u

Pokazivači i datoteke

Pointers & Files