

Uvod u programiranje

- predavanja -

studenii 2025.

13. Funkcije

Nije bilo vođića...



Funkcije

Rekurzivne funkcije

Rekurzivna funkcija

- Funkcija koja poziva samu sebe naziva se *rekurzivnom funkcijom*
 - izravna rekurzija: npr. funkcija f sadrži poziv funkcije f
 - neizravna rekurzija: npr. funkcija f sadrži poziv funkcije g koja sadrži poziv funkcije f
- Primjer definicije i poziva rekurzivne funkcije

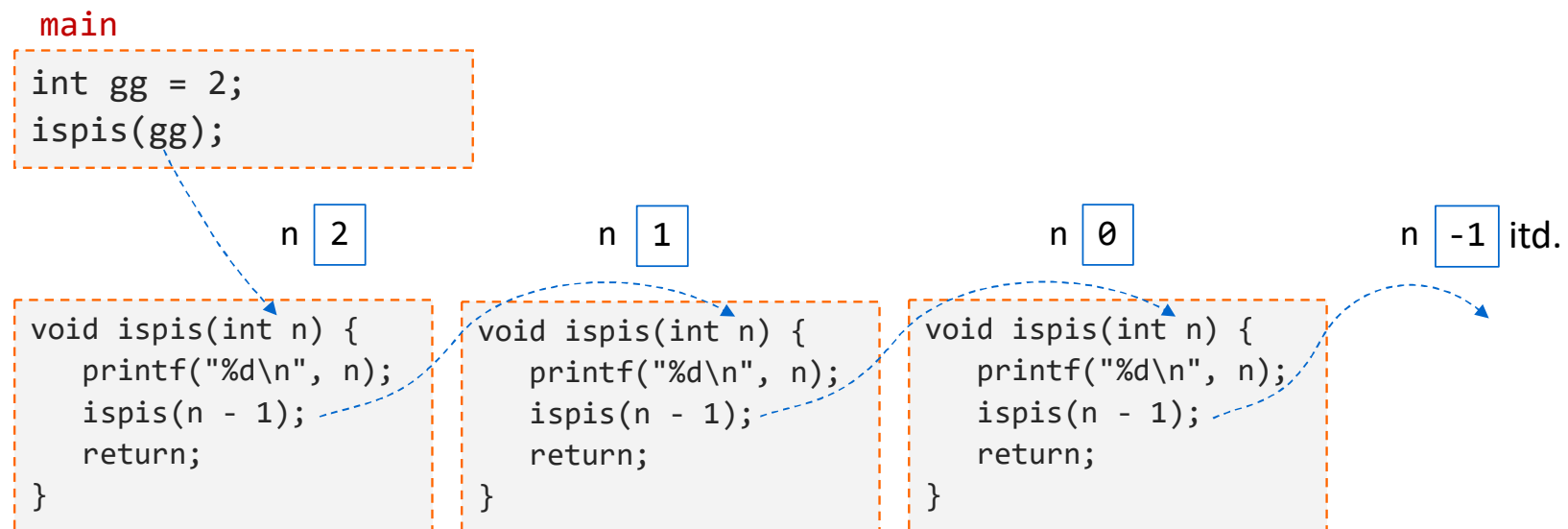
```
void ispis(int n) {  
    printf("%d\n", n);  
    ispis(n - 1);  
    return;  
}  
... u funkciji main  
    int gg = 2;  
    ispis(gg);
```

2↵
1↵
0↵
... ? Kada će se ispisivanje
cijelih brojeva prekinuti?

- Što će biti rezultat poziva funkcije? Koju veliku pogrešku sadrži ova definicija funkcije?

Redoslijed pozivanja

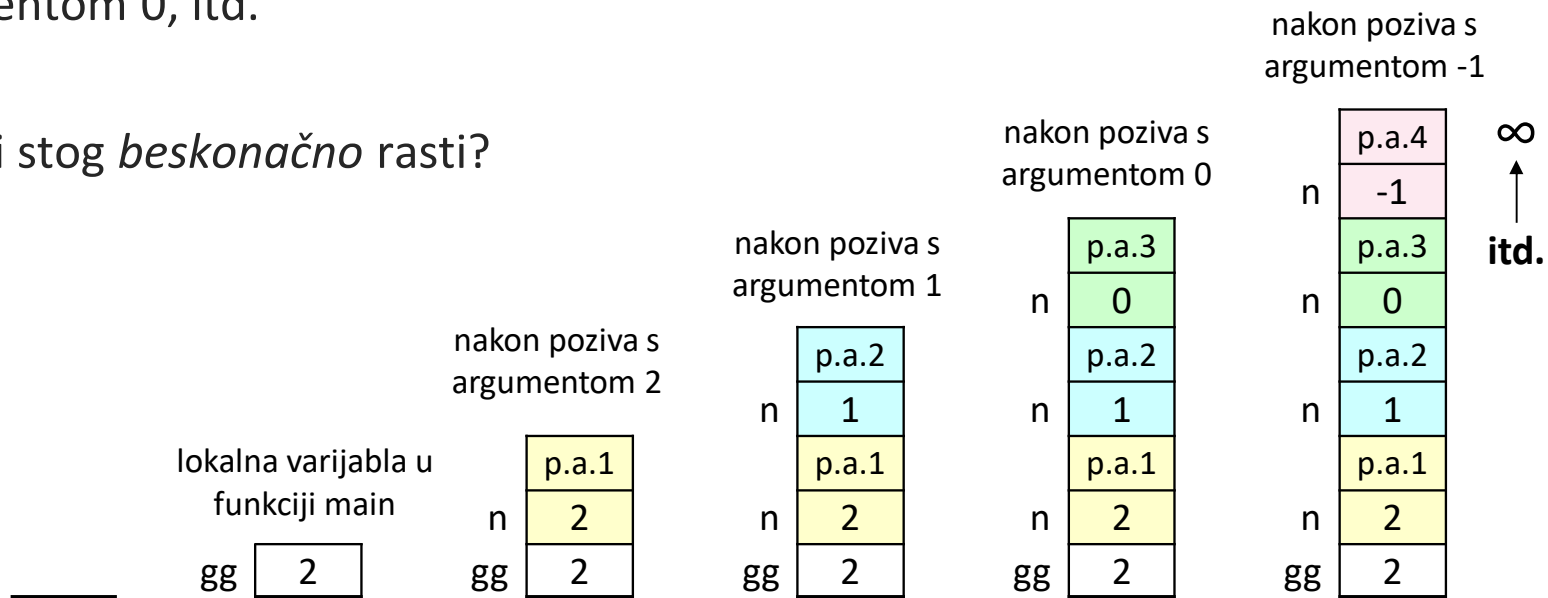
- Zamislimo, radi vizualizacije, da postoji više instanci funkcije `ispis`
- Prije nego funkcija `ispis` pozvana s argumentom `gg=2` završi, poziva funkciju `ispis` s argumentom 1. Prije nego ova završi, poziva funkciju `ispis` s argumentom 0, itd.



Stog u loše definiranoj rekurzivnoj funkciji

- Prije nego funkcija ispis pozvana s argumentom 2 završi (dakle, dok sa stoga još nisu uklonjeni parametar $n=2$ i povratna adresa p.a.1 za povratak u funkciju main), poziva funkciju ispis s argumentom 1. Prije nego ova završi, poziva funkciju ispis s argumentom 0, itd.

- Smije li stog *beskonačno* rasti?



- kada se memorija inicijalno dodijeljena programu za stog potroši, program će se prekinuti zbog pogreške tijekom izvršavanja.

Posljedica loše definirane rekurzivne funkcije

```
void ispis(int n) {  
    printf("%d\n", n);  
    ispis(n - 1);  
    return;  
}
```

```
2↵  
1↵  
0↵  
-1↵  
...  
-392858↵  
-392859↵  
Segmentation fault (core dumped)
```

izvršavanje u operacijskom sustavu Linux

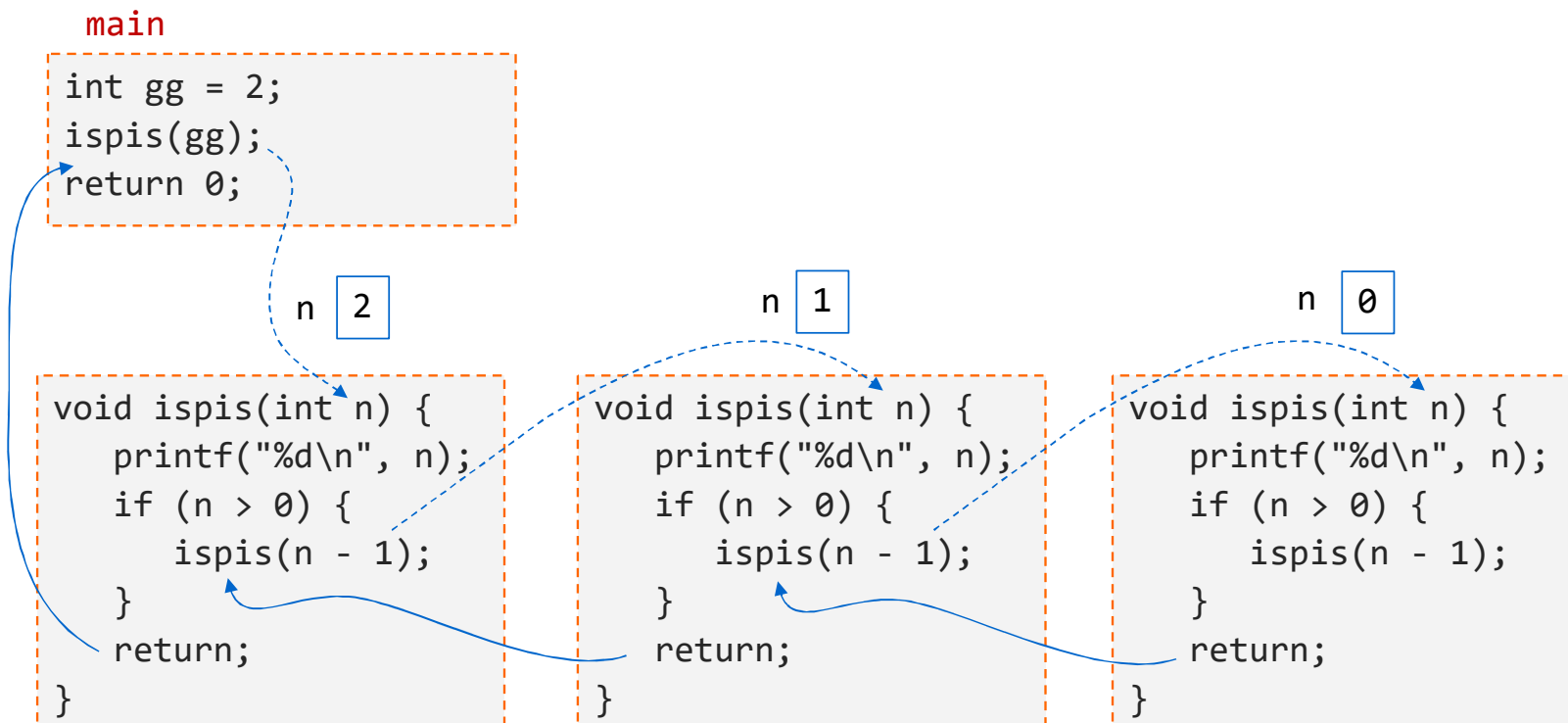
- Rekurzivna funkcija se mora definirati tako da pod nekim uvjetima prestane s daljnjim pozivanjem same sebe

```
void ispis(int n) {  
    printf("%d\n", n);  
    if (n > 0) {  
        ispis(n - 1);  
    }  
    return;  
}
```

```
2↵  
1↵  
0↵
```

Redoslijed pozivanja i povratka

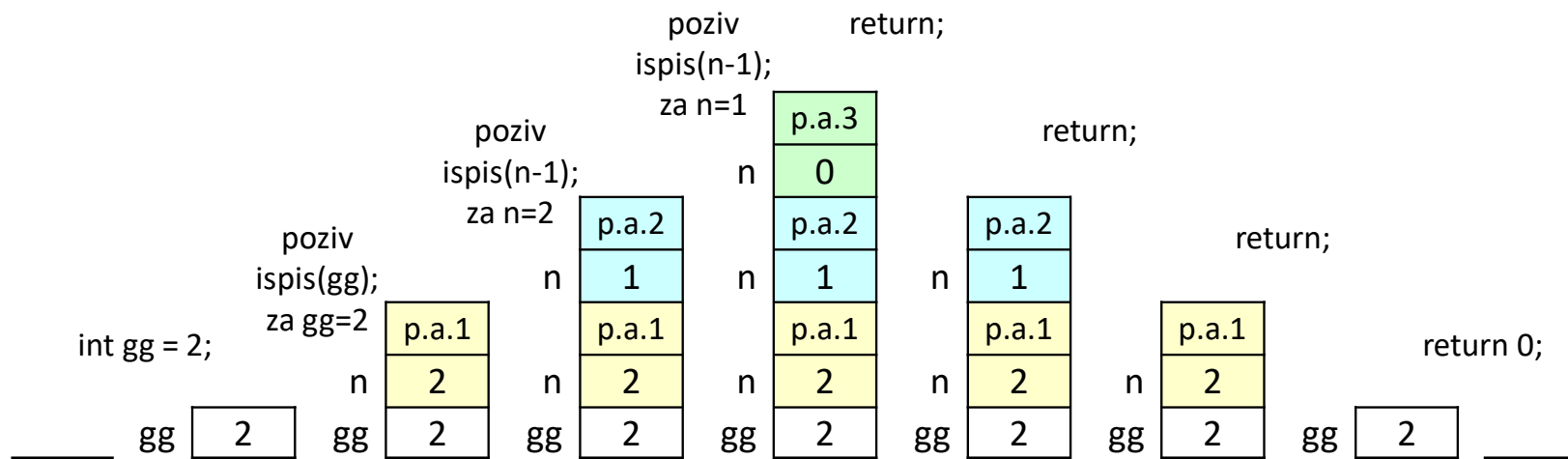
- zamislamo, radi vizualizacije, da postoji više instanci funkcije ispis



Stog u ispravno definiranoj rekurzivnoj funkciji

```
int main(void) {  
    int gg = 2;  
    ispis(gg);  
    return 0;  
}
```

```
void ispis(int n) {  
    printf("%d\n", n);  
    if (n > 0) {  
        ispis(n - 1);  
    }  
    return;  
}
```



Primjer: matematička definicija funkcije

- Rekurzivna definicija funkcije $fact(n)$:

- $$fact(n) = \begin{cases} 1 & \text{za } n = 0 \\ n \cdot fact(n - 1) & \text{za } n > 0 \end{cases}$$

$fact(4) =$

$4 \cdot (fact(3)) =$

$4 \cdot (3 \cdot (fact(2))) =$

$4 \cdot (3 \cdot (2 \cdot (fact(1)))) =$

$4 \cdot (3 \cdot (2 \cdot (1 \cdot (fact(0))))) =$

$4 \cdot (3 \cdot (2 \cdot (1 \cdot (1))))$

Primjer: definicija funkcije u C-u

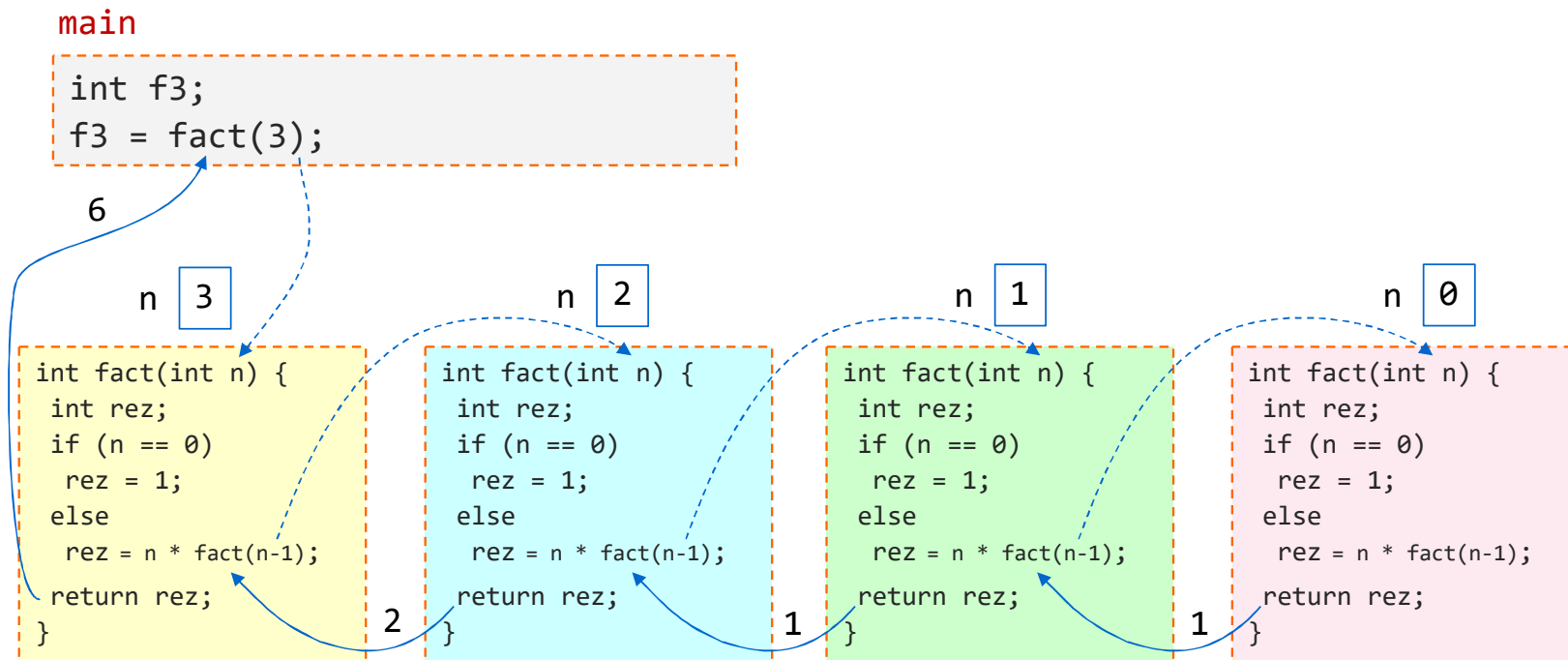
- Matematički rekurzivni izrazi često se bez teškoća pretvaraju u definiciju rekurzivne funkcije u programskom jeziku

```
int fact(int n) {  
    int rez;  
    if (n == 0)  
        rez = 1;  
    else  
        rez = n * fact(n - 1);  
    return rez;  
}
```

```
int fact(int n) {  
    if (n == 0)  
        return 1;  
    else  
        return n * fact(n - 1);  
}
```

Primjer: redoslijed pozivanja i povratka

- Zamislamo, radi vizualizacije, da postoji više instanci funkcije *fact*

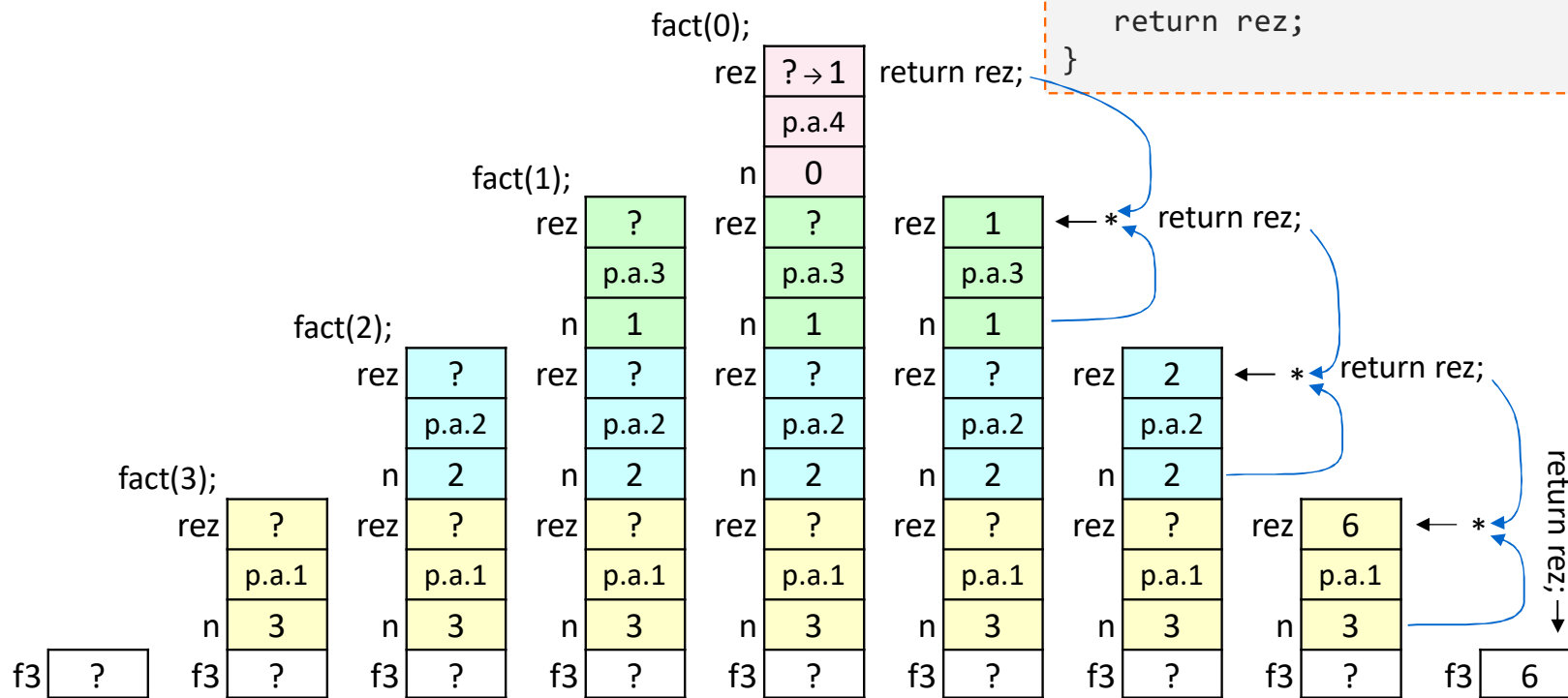


Primjer: stog za poziv fact(3)

main

```
int f3;  
f3 = fact(3);
```

```
int fact(int n) {  
    int rez;  
    if (n == 0)  
        rez = 1;  
    else  
        rez = n * fact(n - 1);  
    return rez;  
}
```



Varijanta s *unsigned long long* umjesto *int*

- Korištenjem ovog tipa podatka, domena funkcije *fact* povećava se na cijele brojeve iz intervala [0, 20]

```
unsigned long long
fact(unsigned int n) {
    unsigned long long rez;
    if (n == 0)
        rez = 1ULL;
    else
        rez = n * fact(n - 1);
    return rez;
}
```

```
unsigned long long
fact(unsigned int n) {
    if (n == 0)
        return 1ULL;
    else
        return n * fact(n - 1);
}
```

Zadatak



- Preradimo naš program iz četvrtog predavanja tako da napravimo rekurzivnu funkciju koja ispisuje binarnu reprezentaciju broja (bez korištenja polja):

Primjer

- Programski zadatak
 - Učitati nenegativni cijeli broj. Nije potrebno provjeravati ispravnost unesenog broja. Ispisivati ostatke uzastopnog dijeljenja učitano broj s 2, a postupak prekinuti kad se dijeljenjem dođe do 0
 - učitani broj može biti 0. Može se dogoditi da se neće ispisati niti jedan ostatak dijeljenja, odnosno da se tijelo petlje neće izvršiti niti jednom
 - Primjeri izvršavanja programa

```
Upisite nenegativan cijeli broj > 11↵
Upisali ste 11↵
Ostatak = 1↵
Ostatak = 1↵
Ostatak = 0↵
Ostatak = 1↵
```

```
Upisite nenegativan cijeli broj > 0↵
Upisali ste 0↵
```

Upisite nenegativan cijeli broj : 11

11 = 1011

Tip podatka *pokazivač*

Uvod

Radna memorija računala

- Radna memorija računala može se promatrati kao kontinuirani niz bajtova, od kojih svaki ima svoj "redni broj", odnosno *adresu*
 - slikom je ilustrirana memorija veličine 4GB

0	00110001
1	11010010
...	...
82560	11000001
82561	00001101
82562	11000001
82563	11101000
...	...
4294967294	00110001
4294967295	00000111

Objekti i vrijednosti u programskom jeziku C

- Objekt (*object*) je područje u memoriji čiji sadržaj reprezentira vrijednost
- Vrijednost (*value*) je interpretacija sadržaja objekta koja se temelji na *tipu i sadržaju* objekta

```
...  
char c = 'B';  
...  
int m = 7;  
...  
float x = -0.75f;  
...
```

...	...	
82560	01000010	} c
...	...	
82642	00000000	} m
82643	00000000	
82644	00000000	
82645	00000111	
...	...	
82714	10111111	} x
82715	01000000	
82716	00000000	
82717	00000000	
...	...	

Adresa objekta

- Za objekt kažemo da se nalazi na adresi A (ili adresa objekta je A) ako je prvi bajt sadržaja objekta pohranjen na adresi A
 - Npr. varijabla m nalazi se na adresi 82642, odnosno adresa varijable m je 82642

- Radi ilustracije pretpostavljeno je da se varijable nalaze na prikazanim adresama. U stvarnosti, nemoguće je znati o kojim se točno adresama radi prije nego se program pokrene (a nije niti važno znati ih unaprijed).

...	...	
82560	00001101	} c
...	...	
82642	00000000	} m
82643	00000000	
82644	00000000	
82645	00000111	
...	...	
82714	10111111	} x
82715	01000000	
82716	00000000	
82717	00000000	
...	...	

Kako se u programu dolazi do vrijednosti objekta

- Pristupanje objektu pomoću *identifikatora*
 - ime varijable (za skalarne tipove), ime varijable i indeks (za polja), ime varijable i ime člana (za strukture), ...
 - navođenjem identifikatora objekta dobiva se *lvalue* koji se može koristiti za čitanje ili postavljanje vrijednosti objekta
 - tip podatka poznat je iz definicije varijable
 - tip je važan: npr. ako tip podatka ne bi bio poznat, ne bi bilo moguće ispravno obavljati operacije

```
double y;  
y = m + x;
```

...	...	
82560	00001101	} c
...	...	
82642	00000000	} m
82643	00000000	
82644	00000000	
82645	00000111	
...	...	
82714	10111111	} x
82715	01000000	
82716	00000000	
82717	00000000	
...	...	

Može li se objektu pristupiti pomoću adrese?

- Može li se do vrijednosti doći pomoću (samo) adrese objekta?
 - npr. ako je poznato da se neki objekt nalazi na adresi 82642?
 - ne, samo adresa nije dovoljna**
- Za ispravno pristupanje objektu potrebna je *i adresa i tip objekta* koji se nalazi na toj adresi
 - adresa i tip objekta predstavljaju jedan oblik *reference* na taj objekt
 - tip objekta kojem se pristupa pomoću *reference* naziva se **referencirani tip** (*referenced type*)

...	...	
82560	00001101	} c
...	...	
82642	00000000	} m
82643	00000000	
82644	00000000	
82645	00000111	
...	...	
82714	10111111	} x
82715	01000000	
82716	00000000	
82717	00000000	
...	...	

Tip podatka *pokazivač (pointer type)*

- Tip podatka koji omogućuje pristupanje objektu pomoću *reference*
 - Ako je referencirani objekt tipa T , tada se za pristupanju objektu koristi tip podatka *pokazivač na T* . Npr. podatak tipa *pokazivač na int* omogućuje pristup objektu tipa int
 - Za tip podatka *pokazivač* ne postoje zasebne ključne riječi (kao za tipove podataka int , $float$, itd.). Tip podatka *pokazivač* opisuje se pomoću naziva referenciranog tipa i znaka $*$

```
int *p1, *p2;  
float *p3;
```

- Varijable $p1$ i $p2$ su tipa pokazivač na int
- Varijabla $p3$ je tipa pokazivač na $float$

Varijable tipa pokazivač

- Za varijablu *tipa pokazivač* vrijedi sve što je do sada navedeno o varijablama ostalih skalarnih tipova, osim:
 - definira se na malo drugačiji način: navođenjem imena *referenciranog tipa* i znaka *** ispred imena varijable
 - pohranjuje podatke tipa pokazivač na referencirani tip

```
int m;
```

```
int *p1;
```

referencirani
tip

varijabla p1 nije tipa *int*, nego
tipa *pokazivač na int*

- dopušteno je u istoj naredbi definirati varijable referenciranog tipa i varijable tipa pokazivača na referencirani tip

```
int m, *p1, *p2, k;
```

Koju vrijednost upisati u varijablu tipa pokazivač

```
int m = 7, *p1;  
p1 = ?
```

- Općenito, ako je *x* varijabla (ili član polja, ili struktura ili član strukture, ...), tada je **&x** pokazivač na *x*
 - &** je tzv. *adresni operator*. Rezultat izraza **&m** je *pokazivač na int* jer je *m* objekt tipa *int*
 - adresa odgovara adresi varijable *m* (82642)
 - rezultat je tipa pokazivač na *int*
 - budući da je rezultat izraza **&m** *pokazivač na int*, smije se pridružiti varijabli *p1* (koja je tipa *pokazivač na int*)

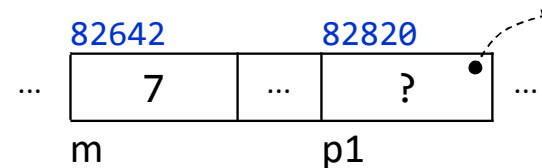
```
p1 = &m;
```

...	...	
82642	00000000	}
82643	00000000	
82644	00000000	
82645	00000111	
...	...	
82820	00000000	}
82821	00000001	
82822	01000010	
82823	11010010	
...	...	

Primjer

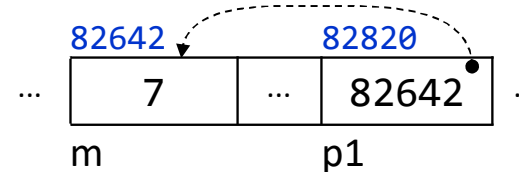
- U nastavku ćemo sadržaj memorije prikazivati na prikladniji način

```
int m = 7, *p1;
```



- varijabla `p1` još uvijek nije inicijalizirana: pokazivač pohranjen u varijabli `p1` "pokazuje u nepoznato"

```
p1 = &m;
```



- u varijablu `p1` sada je upisan podatak tipa *pokazivač na int* kojim se može pristupiti objektu tipa `int` na adresi 82642
 - radi pojednostavljenja, koristit će se kolokvijalni izrazi:
 - naredbom `int *p1;` definiran je pokazivač `p1`
 - naredbom `p1 = &m;` u `p1` je upisana adresa varijable `m`
 - `p1` pokazuje na objekt na adresi 82642
 - `p1` pokazuje na varijablu `m`, `p1` pokazuje na objekt `m`


Inicijalizacija varijable tipa pokazivača uz definiciju

- Jednako kao i varijable drugih tipova, varijable tipa pokazivač mogu se inicijalizirati u trenutku definicije

```
int m, *p1 = &m, *p2 = p1;  
float x, *p3 = &x, y, *p4 = &y;
```

- voditi računa o redoslijedu definicije i inicijalizacije. Objekt čija se adresa izračunava adresnim operatorom mora biti definiran


```
int *p1 = &m, m;
```



Neispravno, može se popraviti premještanjem

- voditi računa o tome da i varijabla tipa pokazivača može sadržavati "smeće" (*garbage value*)

```
int m, *p1;  
int *p2 = p1;  
p1 = &m;
```



U ovom trenutku p1 još uvijek sadrži "smeće"
Može se popraviti premještanjem naredbe

Paziti na razlike u tipovima pokazivača

- Tipovi pokazivača su međusobno različiti ako se razlikuju njihovi referencirani tipovi
 - u varijable jednog tipa pokazivača nije dopušteno upisivati pokazivače drugog tipa

```
int m;  
int *pInt;  
float x;  
float *pFloat;
```

```
pInt = &m;  
pFloat = &x;
```

```
pFloat = pInt;  
pInt = &x;  
pFloat = &m;
```

Neispravno

Neispravno

Neispravno

Adresa nije cijeli broj

- Iako *izgleda* kao cijeli broj, adresa u općem slučaju nije `int` (niti `short`, niti `long`, ...). Stoga nema smisla:
 - pokazivač pohranjivati u varijablu tipa `int`
 - cijeli broj pohranjivati u varijablu tipa pokazivača

```
int *p1;  
int m;  
m = 5;  
p1 = &m;
```

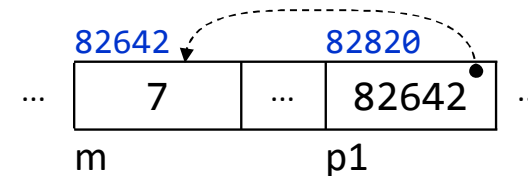
```
p1 = m;  
m = p1;
```

Neispravno

Neispravno

Pristupanje objektu pomoću pokazivača

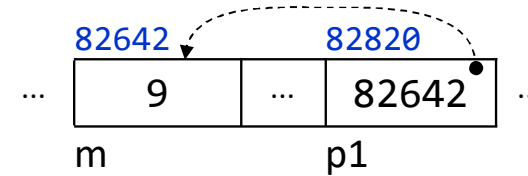
```
int m = 7, *p1 = &m;
```



- objektu (7, tip `int`) na adresi 82642 može se pristupiti:

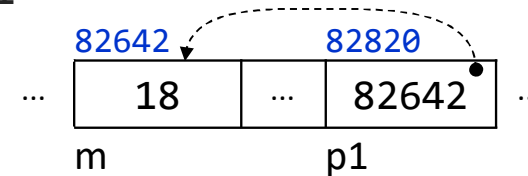
- (naravno) pomoću imena varijable `m`

```
m = m + 2;
```



- ali također i primjenom *operatora indirekcije* (unarni operator `*`) nad pokazivačem pohranjenim u varijabli `p1`

```
*p1 = 2 * *p1;
```



```
printf("%d %d", m, *p1);
```

```
18 18
```

pročitaj cijeli broj s mjesta na kojeg pokazuje `p1`, dobiveni rezultat (tipa `int`) pomnoži s 2 i rezultat upiši na mjesto kamo pokazuje `p1`

Operator indirekcije *

- Operator omogućuje da se objektu pristupi *indirektno* pomoću pokazivača (umjesto *direktno* preko imena varijable)
 - operator je također poznat pod imenom *operator dereferenciranja* jer operator "*dereferencira*" pokazivač (*referencu* na objekt) i tako dolazi do objekta
- Općenito, ako p pokazuje na objekt x, tada je rezultat operacije **p lvalue* koja predstavlja objekt na kojeg pokazuje p
 - to znači: ako je p varijabla koja sadrži pokazivač koji pokazuje na objekt u memoriji koji predstavlja varijablu m, tada se izraz *p može koristiti na svakom mjestu u programu gdje se može koristiti ime varijable m
 - za čitanje vrijednosti (npr. u nekom izrazu)
 - za postavljanje vrijednosti (kao lijeva strana izraza pridruživanja), uz uvjet da je sadržaj objekta izmjenljiv

Neke oznake su pomalo zbunjujuće?

```
int m = 7;
int *p1 = &m;
...
p1 = &m;           Ispravno
*p1 = &m;          Neispravno
```

- Kako to da je u naredbi za definiciju varijable p1 ispravno napisati `*p1 = &m`, a naredba `*p1 = &m;` je neispravna?

- u programskom jeziku C isti simboli u različitom kontekstu mogu imati različito značenje

```
int *p1 = &m;
```

p1 definiraj kao varijablu
tipa pokazivač na int

varijablu koju si upravo definirao,
p1, inicijaliziraj na vrijednost &m

ovdje simbol `*` ne predstavlja operator indirekcije, nego označava da varijabla p1 nije tipa `int`, nego tipa *pokazivač na int*

```
*p1 = &m;
```

neispravno jer je rezultat izraza `*p1` objekt tipa `int`, što znači da se u objekt tipa `int` pokušava upisati vrijednost tipa pokazivač na `int`

Neke oznake su pomalo zbunjujuće?

```
int m = 7;  
int *p1 = &m, *p2 = p1;  
...  
p2 = p1;           Ispravno  
*p2 = p1;          Neispravno
```

```
int *p1 = &m, *p2 = p1;
```

```
p2 = *p1;
```

- Kako to da je u naredbi za definiciju varijable p2 ispravno napisati `*p2 = p1`, a naredba `*p2 = p1;` je neispravna?

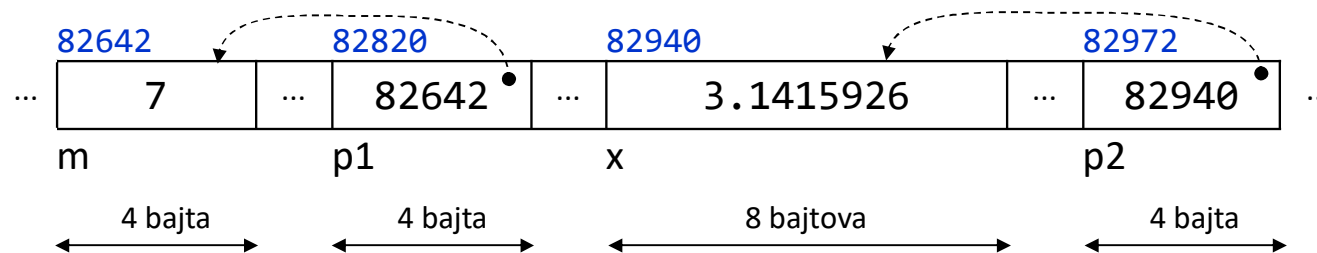
definirana je varijabla p1, inicijalizirana je na &m, zatim je definirana varijabla p2 koja se inicijalizira na vrijednost koja se nalazi u p1.

neispravno jer je rezultat izraza `*p1` objekt tipa `int`, što znači da se vrijednost tipa `int` pokušava upisati u varijablu tipa pokazivač na `int`

Koliko prostora zauzima pokazivač

- Adresa objekta je adresa na kojoj je pohranjen prvi bajt objekta
 - to znači da veličina referenciranog tipa ne bi trebala utjecati na veličinu prostora koju zauzima pokazivač na taj tip

```
int m = 7, *p1 = &m;  
double x = 3.1415926, *p2 = &x;
```



- jednaki prostor (4 bajta) zauzimaju pokazivač p1 na objekt tipa int (koji je veličine 4 bajta) i pokazivač p2 na objekt tipa double (koji je veličine 8 bajtova)

Koliko prostora zauzima pokazivač

- Pokazivači na jednoj platformi (isti operacijski sustav, arhitektura i prevodilac) u principu* zauzimaju jednaku količinu memorije bez obzira na koji tip podatka pokazuju

```
int m = 7, *p1 = &m;  
double x = 3.1415926, *p2 = &x;  
printf("%u %u %u\n", sizeof(p1), sizeof(m), sizeof(*p1));  
printf("%u %u %u", sizeof(p2), sizeof(x), sizeof(*p2));
```

x86_64, Windows, gcc

```
4 4 4  
4 8 8
```

x86_64, Linux, gcc

```
8 4 4  
8 8 8
```

* U praksi je to uglavnom tako, ali s obzirom da C standard takvo pravilo izrijeком ne propisuje, ne smije se u potpunosti isključiti mogućnost da će se na nekoj platformi veličine pokazivača međusobno razlikovati s obzirom na tip podatka na koji pokazuju.

Generički pokazivač (*pointer to void*)

- Referencirani tip pokazivača mora biti poznat kako bi se na temelju adrese (gdje je objekt) i tipa (kojeg tipa je objekt na toj adresi) sadržaj objekta mogao ispravno interpretirati
- Međutim, postoji specijalni tip pokazivača za kojeg to ne vrijedi
 - *generički pokazivač* (u literaturi također: *pokazivač na void*, *pointer to void*) je pokazivač koji može pokazivati na objekt bilo kojeg tipa
 - budući da referencirani tip generičkog pokazivača nije poznat, neće se moći koristiti za pristup objektu (kažemo: generički pokazivač se ne može *dereferencirati*)
 - ali zato je moguće napraviti **eksplicitnu konverziju (*cast*)** generičkog pokazivača na tip pokazivača za kojeg će referencirani tip biti T
 - rezultat sljedeće operacije nad generičkim pokazivačem je pokazivač na tip podatka T

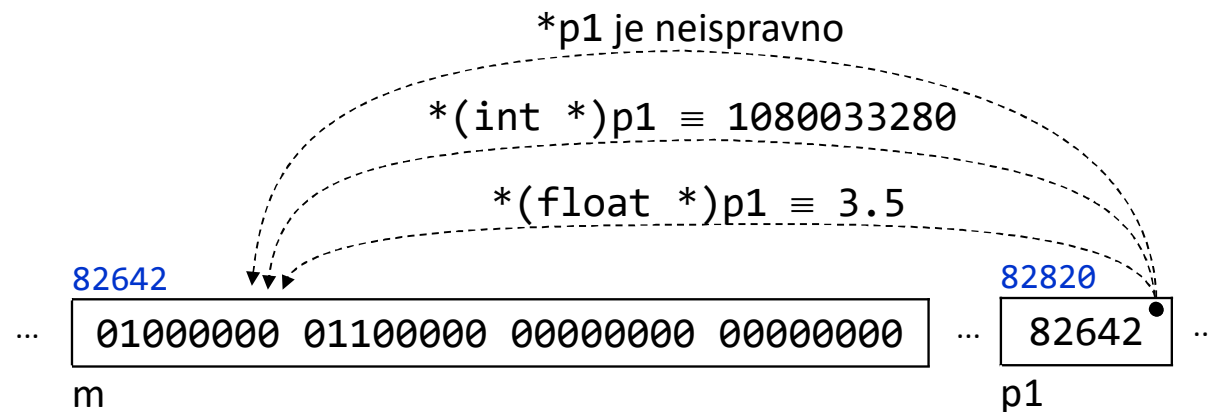
(T *) genericki_pokazivac

Primjer

```
int m = 1080033280;
void *p1;
p1 = &m;
// printf("%d", *p1);
printf("%d\n", *(int *)p1);
printf("%f\n", *(float *)p1);
```

Neispravno jer p1 nije moguće dereferencirati

1080033280
3.500000



1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 2103, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116, 2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 2130, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 2143, 2144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 2152, 2153, 2154, 2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2163, 2164, 2165, 2166, 2167, 2168, 2169, 2170, 2171, 2172, 2173, 2174, 2175, 2176, 2177, 2178, 2179, 2180, 2181, 2182, 2183, 2184, 2185, 2186, 2187, 2188, 2189, 2190, 2191, 2192, 2193, 2194, 2195, 2196, 2197, 2198, 2199, 2200, 2201, 2202, 2203, 2204, 2205, 2206, 2207, 2208, 2209, 2210, 2211, 2212, 2213, 2214, 2215, 2216, 2217, 2218, 2219, 2220, 2221, 2222, 2223, 2224, 2225, 2226, 2227, 2228, 2229, 2230, 2231, 2232, 2233, 2234, 2235, 2236, 2237, 2238, 2239, 2240, 2241, 2242, 2243, 2244, 2245, 2246, 2247, 2248, 2249, 2250, 2251, 2252, 2253, 2254, 2255, 2256, 2257, 2258, 2259, 2260, 2261, 2262, 2263, 2264, 2265, 2266, 2267, 2268, 2269, 2270, 2271, 2272, 2273, 2274, 2275, 2276, 2277, 2278, 2279, 2280, 2281, 2282, 2283, 2284, 2285, 2286, 2287, 2288, 2289, 2290, 2291, 2292, 2293, 2294, 2295, 2296, 2297, 2298, 2299, 2300, 2301, 2302, 2303, 2304, 2305, 2306, 2307, 2308, 2309, 2310, 2311, 2312, 2313, 2314, 2315, 2316, 2317, 2318, 2319, 2320, 2321, 2322, 2323, 2324, 2325, 2326, 2327, 2328, 2329, 2330, 2331, 2332, 2333, 2334, 2335, 2336, 2337, 2338, 2339, 2340, 2341, 2342, 2343, 2344, 2345, 2346, 2347, 2348, 2349, 2350, 2351, 2352, 2353, 2354, 2355, 2356, 2357, 2358, 2359, 2360, 2361, 2362, 2363, 2364, 2365, 2366, 2367, 2368, 2369, 2370, 2371, 2372, 2373, 2374, 2375, 2376, 2377, 2378, 2379, 2380, 2381, 2382, 2383, 2384, 2385, 2386, 2387, 2388, 2389, 2390, 2391, 2392, 2393, 2394, 2395, 2396, 2397, 2398, 2399, 2400, 2401, 2402, 2403, 2404, 2405, 2406, 2407, 2408, 2409, 2410, 2411, 2412, 2413, 2414, 2415, 2416, 2417, 2418, 2419, 2420, 2421, 2422, 2423, 2424, 2425, 2426, 2427, 2428, 2429, 2430, 2431, 2432, 2433, 2434, 2435, 2436, 2437, 2438, 2439, 2440, 2441, 2442, 2443, 2444, 2445, 2446, 2447, 2448, 2449, 2450, 2451, 2452, 2453, 2454, 2455, 2456, 2457, 2458, 2459, 2460, 2461, 2462, 2463, 2464, 2465, 2466, 2467, 2468, 2469, 2470, 2471, 2472, 2473, 2474, 2475, 2476, 2477, 2478, 2479, 2480, 2481, 2482, 2483, 2484, 2485, 2486, 2487, 2488, 2489, 2490, 2491, 2492, 2493, 2494, 2495, 2496, 2497, 2498, 2499, 2500, 2501, 2502, 2503, 2504, 2505, 2506, 2507, 2508, 2509, 2510, 2511, 2512, 2513, 2514, 2515, 2516, 2517, 2518, 2519, 2520, 2521, 2522, 2523, 2524, 2525, 2526, 2527, 2528, 2529, 2530, 2531, 2532, 2533, 2534, 2535, 2536, 2537, 2538, 2539, 2540, 2541, 2542, 2543, 2544, 2545, 2546, 2547, 2548, 2549, 2550, 2551, 2552, 2553, 2554, 2555, 2556, 2557, 2558, 2559, 2560, 2561, 2562, 2563, 2564, 2565, 2566, 2567, 2568, 2569, 2570, 2571, 2572, 2573, 2574, 2575, 2576, 2577, 2578, 2579, 2580, 2581, 2582, 2583, 2584, 2585, 2586, 2587, 2588, 2589, 2590, 2591, 2592, 2593, 2594, 2595, 2596, 2597, 2598, 2599, 2600, 2601, 2602, 2603, 2604, 2605, 2606, 2607, 2608, 2609, 2610, 2611, 2612, 2613, 2614, 2615, 2616, 2617, 2618, 2619, 2620, 2621, 2622, 2623, 2624, 2625, 2626, 2627, 2628, 2629, 2630, 2631, 2632, 2633, 2634, 2635, 2636, 2637, 2638, 2639, 2640, 2641, 2642, 2643, 2644, 2645, 2646, 2647, 2648, 2649, 2650, 2651, 2652, 2653, 2654, 2655, 2656, 2657, 2658, 2659, 2660, 2661, 2662, 2663, 2664, 2665, 2666, 2667, 2668, 2669, 2670, 2671, 2672, 2673, 2674, 2675, 2676, 2677, 2678, 2679, 26



- Tj. uvjerimo se da su ti bitovi doista u memoriji tako postavljeni

P	K	M
01000000010010001111010111000011		
4048f5c3		

Konverzijske specifikacije za printf i scanf

- konverzijska specifikacija %p koristi se za ispis i čitanje podatka tipa pokazivač
 - točan oblik ispisa nije propisan standardom (vrijednost će se ispisati kao broj u dekadskom ili heksadekadskom brojevnom sustavu ili u nekom drugom obliku)
 - argument (pokazivač) koji se ispisuje dobro je eksplicitno konvertirati u generički pokazivač, ali u većini slučajeva može se ispustiti

```
int m = 7, *p1 = &m;  
printf("m je na adresi %p", (void *)p1);  
// printf("m je na adresi %p", p1);
```

Može i bez (void *)

x86_64, Windows, gcc

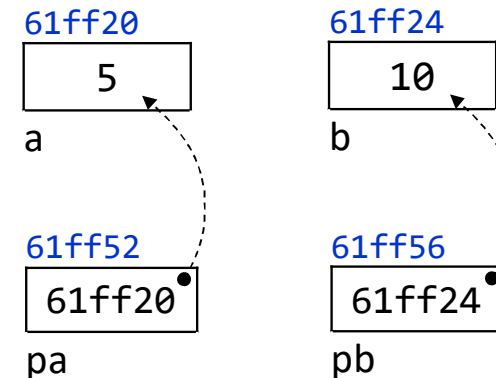
m je na adresi 0061ff28

x86_64, Linux, gcc

m je na adresi 0x7fffd6e8d324

Primjer

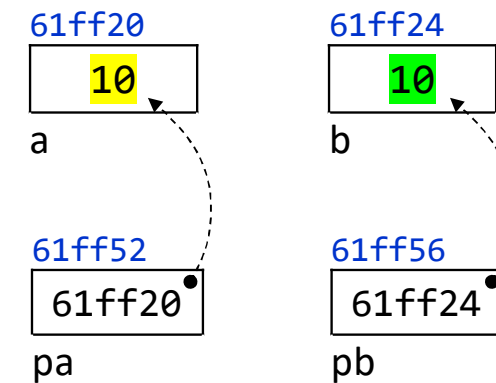
```
int a = 5, b = 10;  
int *pa, *pb;  
pa = &a;    // pretpostavka pa = 61ff20  
pb = &b;    // pretpostavka pb = 61ff24
```



- što će se ispisati sljedećim odsječkom?

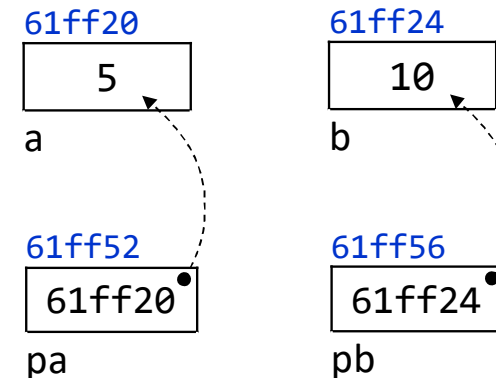
```
*pa = *pb;  
printf("%d %d\n", a, b);  
printf("%p %p\n", (void *)pa, (void *)pb);  
printf("%d %d\n", *pa, *pb);
```

```
10 10  
61ff20 61ff24  
10 10
```



Primjer

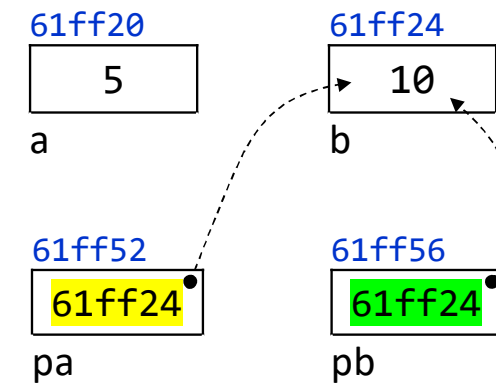
```
int a = 5, b = 10;  
int *pa, *pb;  
pa = &a;    // pretpostavka pa = 61ff20  
pb = &b;    // pretpostavka pb = 61ff24
```



- što će se ispisati sljedećim odsječkom?

```
pa = pb;  
printf("%d %d\n", a, b);  
printf("%p %p\n", (void *)pa, (void *)pb);  
printf("%d %d\n", *pa, *pb);
```

```
5 10  
61ff24 61ff24  
10 10
```



Primjer

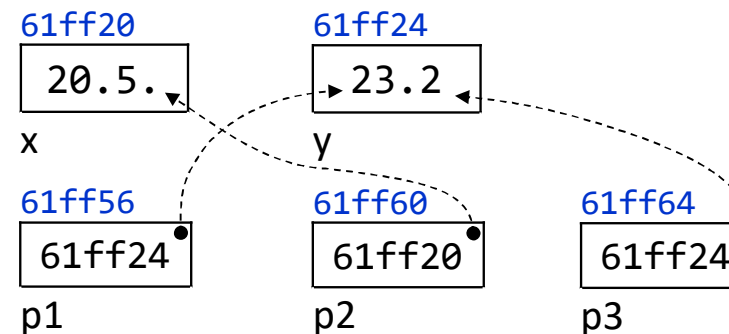
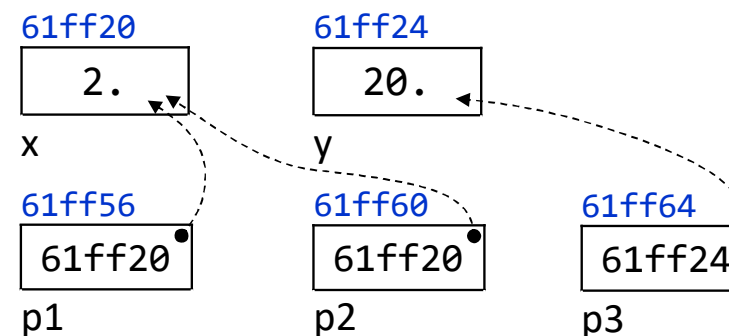
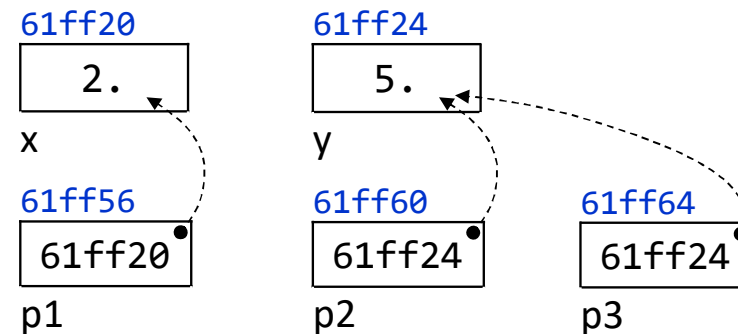
```
float x = 2.f, y = 5.f;  
float *p1, *p2, *p3;  
p1 = &x;  
p2 = p3 = &y;
```

- nacrtati sliku nakon

```
*p3 = *p2 + 3.f * *p2;  
p2 = p1;
```

- i nakon

```
*p2 = *p3 + 0.5f;  
p1 = p3;  
*p1 += 3.2f;
```





Prije sljedećeg predavanja

- Edgar:
 - Tutorial: **nema**
 - **13. vježbe uz predavanja**