ALGEBRA

# STRUKTURE PODATAKA I ALGORITMI

Dodatna tema 01

1

## Video materijali

- Video materijali su dostupni na:
  - Napredni SPA 01
    - https://youtu.be/IekWOIfBqXo
  - Napredni SPA 02
    - https://youtu.be/zTqKPRQ3oz8
  - Napredni SPA 03
    - https://youtu.be/QeDozysawVs
  - Napredni SPA 04
    - https://youtu.be/tjlYDzggSo8



ALGEBRA

2

# PRVA VERZIJA KLASE




3

## Uvod

- Cilj: napisati klasu koja će nam omogućiti čuvanje željene količine cijelih brojeva na hrpi. Klasa treba imati konstruktore:
  - Defaultni, koji omogućuje čuvanje 0 elemenata
  - Konstruktor koji prima broj elemenata i rezervira toliko veliko polje na hrpi. Sve elemente postavlja na 0.
  - Konstruktor koji prima broj elemenata i vrijednost i rezervira toliko veliko polje na hrpi. Sve elemente postavlja na zadanu vrijednost.
  - Konstruktor koji prima bilo kakvo polje i kopira ga na hrpu.

  Neka klasa ima i metodu za ispis svih brojeva.




4

# Klasa

```
class Brojevi {
private:
    int kolicina;
    int* polje_brojeva;

public:
    Brojevi();
    Brojevi(int n);
    Brojevi(int n, int val);
    Brojevi(int original[], int n);
    ~Brojevi();
    void ispisi();
};
```

```
Brojevi::Brojevi() {
    kolicina = 0;
    polje_brojeva = nullptr;
}
Brojevi::Brojevi(int n) {
    kolicina = n;
    polje_brojeva = new int[kolicina] { 0 };
}
Brojevi::Brojevi(int n, int val) {
    kolicina = n;
    polje_brojeva = new int[kolicina];
    fill_n(polje_brojeva, kolicina, val);
}
Brojevi::Brojevi(int original[], int n) {
    kolicina = n;
    polje_brojeva = new int[kolicina];
    copy(original, original + n, polje_brojeva);
}
Brojevi::~Brojevi() {
    if (polje_brojeva != nullptr) {
        delete[] polje_brojeva;
    }
}
void Brojevi::ispisi() {
    cout << "(size:" << kolicina << ") ";
    for (int i = 0; i < kolicina; i++) {
        cout << polje_brojeva[i] << " ";
    }
    cout << endl;
}
```



ALGEBRA

5

# Korištenje

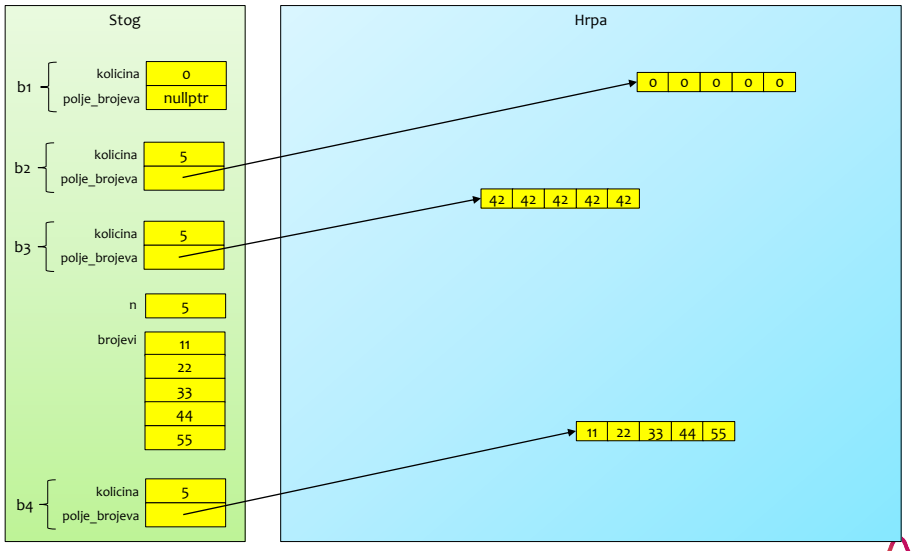
```
Brojevi b1{};
b1.ispisi();

Brojevi b2{ 5 };
b2.ispisi();

Brojevi b3{ 5, 42 };
b3.ispisi();

const int n = 5;
int polje[n]{ 11, 22, 33, 44, 55 };
Brojevi b4{ polje, n };
b4.ispisi();
```



ALGEBRA

6

# Izgled u memoriji






ALGEBRA

7

# PROBLEM S KOPIRANJEM



ALGEBRA

8

# Uvod

- Prva verzija klase djeluje dobro, ali u sebi sadrži ozbiljan problem
- Kao što znamo, kompajler će često automatski generirati copy-constructor
- To nam omogućava pisanje sljedećeg koda:

```
Brojevi b1{ 5, 42 };
b1.ispisi();

Brojevi b2{ b1 };
b2.ispisi();
```

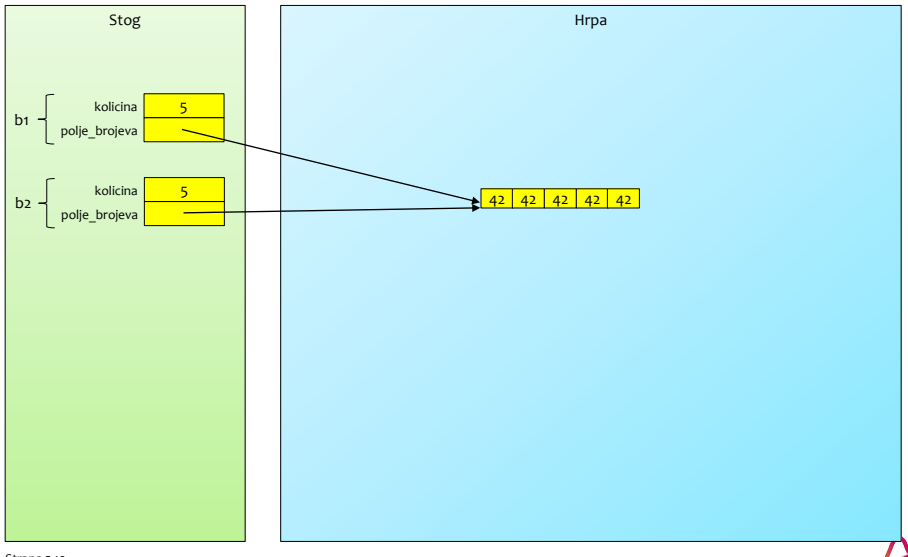
- Zašto nam se program ruši?



ALGEBRA

9

# Izgled u memoriji






ALGEBRA

10

## Objašnjenje

- Copy-constructor doslovno kopira sadržaj svih članova
- To znači da je kopirao i adresu polja na hrpi
- Rezultat je taj da oba objekta pokazuju na isto polje na hrpi
- To se naziva shallow copy
- Kad prvi objekt umre, njegov destruktor ispravno otpušta memoriju s hrpe
- Kad drugi objekt umre, njegov destruktor također pokuša otpustiti memoriju, ali je ta memorija već otpuštena – drugi poziv delete[] ruši program



ALGEBRA

11

## Pola rješenja

- Kao prvi korak u rješenju, napisat ćemo vlastiti copy-constructor koji će također raditi shallow copy

```
…
Brojevi(const Brojevi& orig);
…
```

```
…
Brojevi::Brojevi(const Brojevi& orig) {
    kolicina = orig.kolicina;
    polje_brojeva = orig.polje_brojeva;
    cout << "copy constructor" << endl;
}
…
```



ALGEBRA

12

# Cijelo rješenje

- Sad ćemo promijeniti copy-constructor tako da radi deep copy:

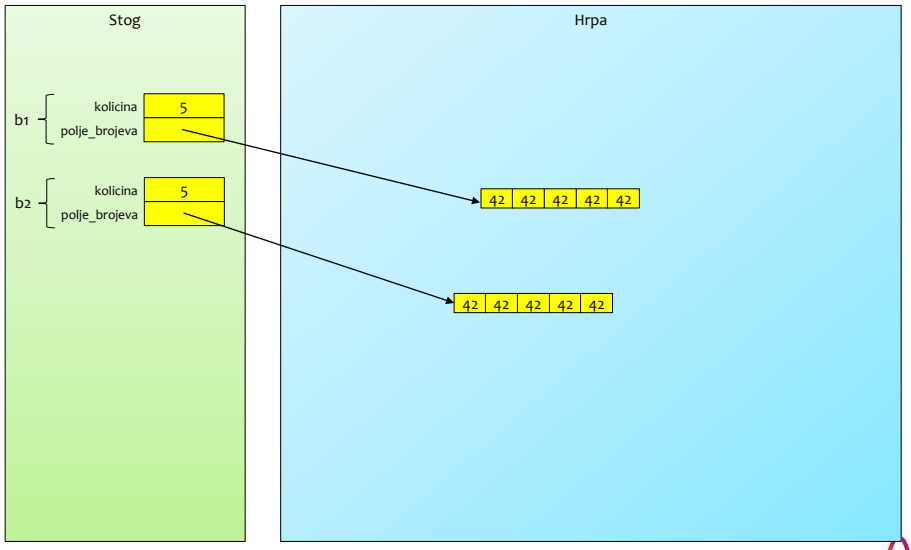
```
…
Brojevi::Brojevi(const Brojevi& orig) {
    kolicina = orig.kolicina;
    polje_brojeva = new int[kolicina];
    copy(orig.polje_brojeva, orig.polje_brojeva + orig.kolicina, polje_brojeva);
    cout << "copy constructor" << endl;
}
…
```



ALGEBRA



# Izgled u memoriji






ALGEBRA

# PROBLEM S PERFORMANSAMA



ALGEBRA

15

## Uvod

- Rješenje koje sad imamo radi ispravno i ne sadrži pogreške
- No, rješenje nije optimalno jer se u određenim situacijama dešava kopiranje koje se može izbjeći
- Primjerice:

```
vector<Brojevi> v;
v.push_back({ 5, 42 });
v.push_back({ 5, 43 });
v.push_back({ 5, 44 });
```

```
value constructor
copy constructor
destructor
value constructor
copy constructor
copy constructor
destructor
destructor
value constructor
copy constructor
copy constructor
copy constructor
destructor
destructor
destructor
destructor
destructor
destructor
```

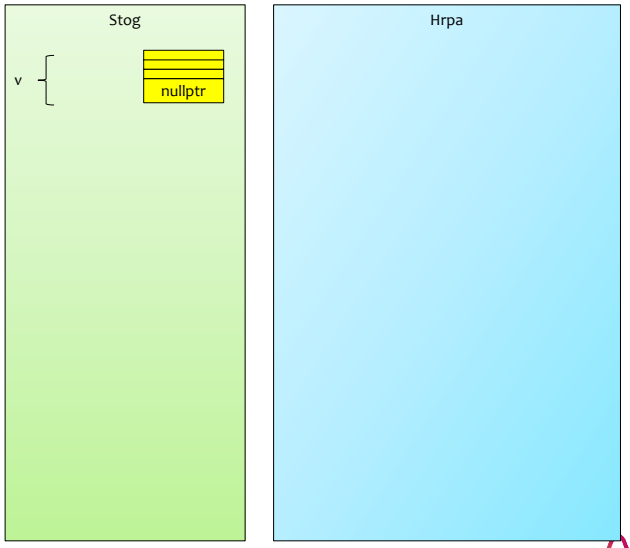


ALGEBRA

16

# Izgled u memoriji

```
vector<Brojevi> v;
v.push_back({ 5, 42 });
v.push_back({ 5, 43 });
v.push_back({ 5, 44 });
```





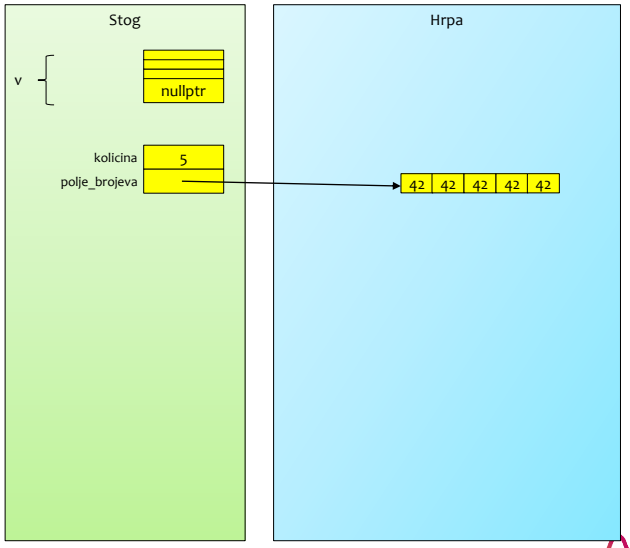

ALGEBRA

17

# Izgled u memoriji

```
vector<Brojevi> v;
v.push_back({ 5, 42 });
v.push_back({ 5, 43 });
v.push_back({ 5, 44 });
```

value constructor





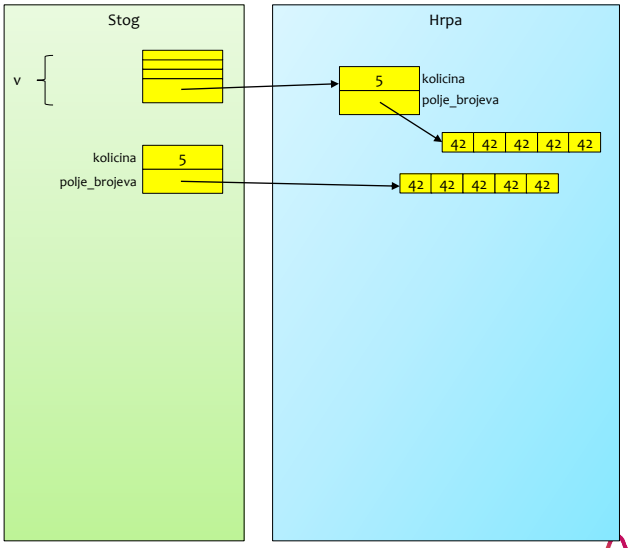

ALGEBRA

18

# Izgled u memoriji

```
vector<Brojevi> v;
v.push_back({ 5, 42 });
v.push_back({ 5, 43 });
v.push_back({ 5, 44 });
```

```
value constructor
copy constructor
```





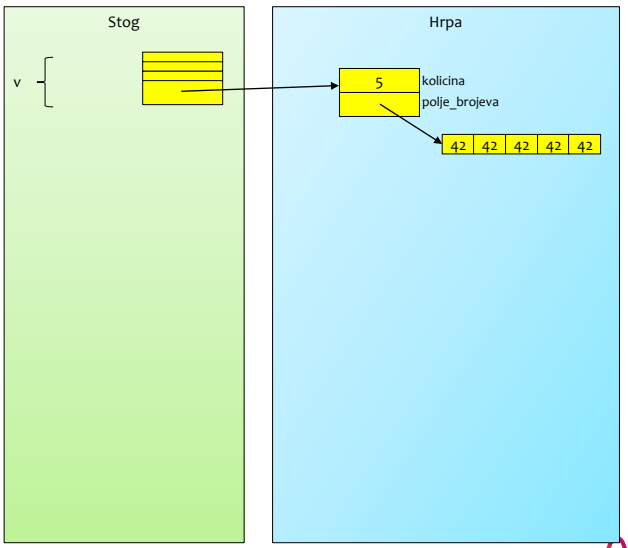

ALGEBRA

19

# Izgled u memoriji

```
vector<Brojevi> v;
v.push_back({ 5, 42 });
v.push_back({ 5, 43 });
v.push_back({ 5, 44 });
```

```
value constructor
copy constructor
destructor
```





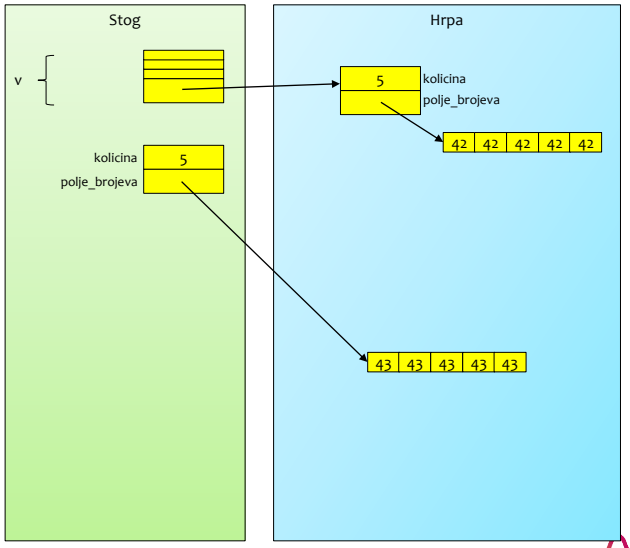

ALGEBRA

20

# Izgled u memoriji

```
vector<Brojevi> v;
v.push_back({ 5, 42 });
v.push_back({ 5, 43 });
v.push_back({ 5, 44 });
```

```
value constructor
copy constructor
destructor
value constructor
```





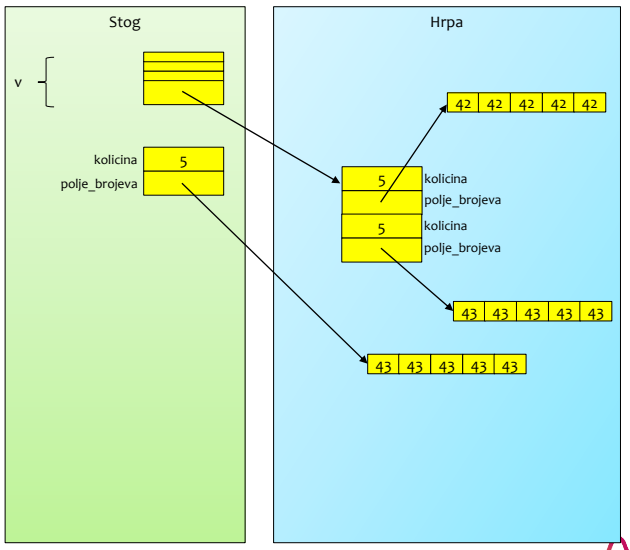

ALGEBRA

21

# Izgled u memoriji

```
vector<Brojevi> v;
v.push_back({ 5, 42 });
v.push_back({ 5, 43 });
v.push_back({ 5, 44 });
```

```
value constructor
copy constructor
destructor
value constructor
copy constructor
copy constructor
destructor
```





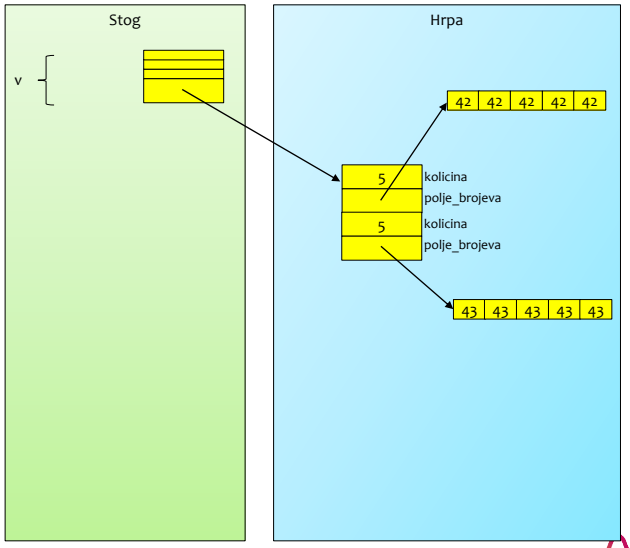

ALGEBRA

22

# Izgled u memoriji

```
vector<Brojevi> v;
v.push_back({ 5, 42 });
v.push_back({ 5, 43 });
v.push_back({ 5, 44 });
```

```
value constructor
copy constructor
destructor
value constructor
copy constructor
copy constructor
destructor
destructor
```





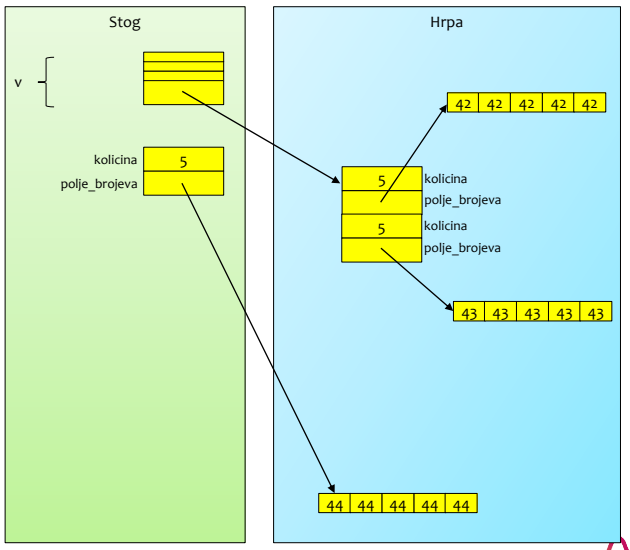

ALGEBRA

23

# Izgled u memoriji

```
vector<Brojevi> v;
v.push_back({ 5, 42 });
v.push_back({ 5, 43 });
v.push_back({ 5, 44 });
```

```
value constructor
copy constructor
destructor
value constructor
copy constructor
copy constructor
destructor
destructor
value constructor
```





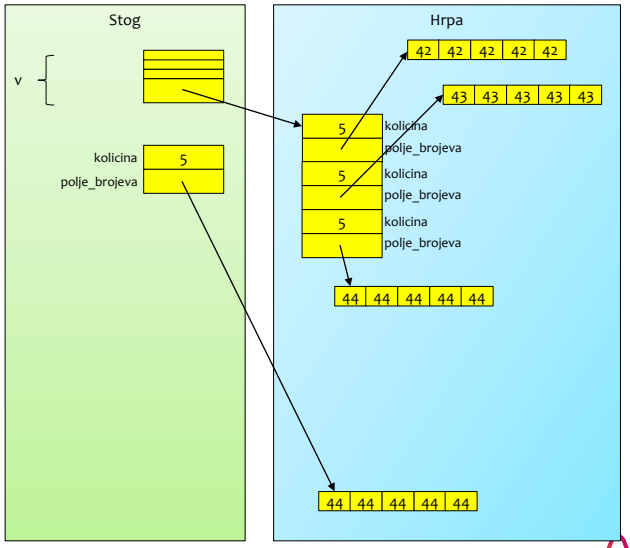

ALGEBRA

24

# Izgled u memoriji

```
vector<Brojevi> v;
v.push_back({ 5, 42 });
v.push_back({ 5, 43 });
v.push_back({ 5, 44 });
```

```
value constructor
copy constructor
destructor
value constructor
copy constructor
copy constructor
destructor
destructor
value constructor
copy constructor
copy constructor
copy constructor
destructor
destructor
```





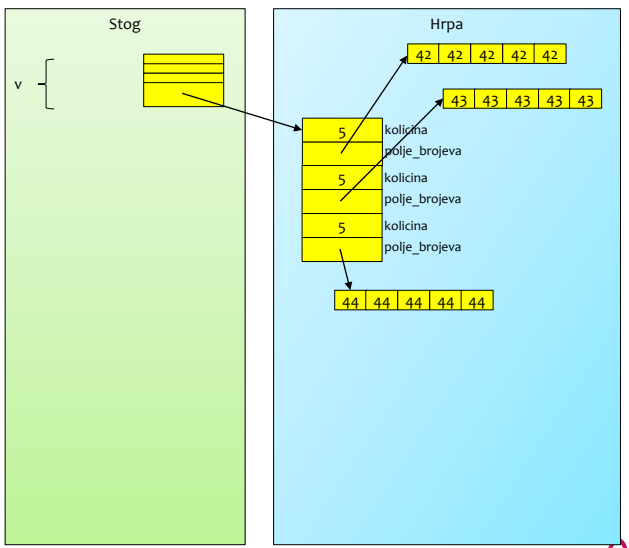

ALGEBRA

25

# Izgled u memoriji

```
vector<Brojevi> v;
v.push_back({ 5, 42 });
v.push_back({ 5, 43 });
v.push_back({ 5, 44 });
```

```
value constructor
copy constructor
destructor
value constructor
copy constructor
copy constructor
destructor
destructor
value constructor
copy constructor
copy constructor
copy constructor
destructor
destructor
destructor
```





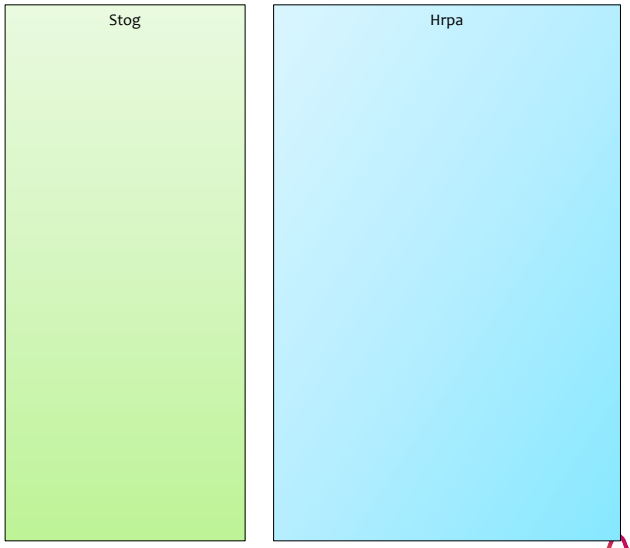

ALGEBRA

26

# Izgled u memoriji

```
vector<Brojevi> v;
v.push_back({ 5, 42 });
v.push_back({ 5, 43 });
v.push_back({ 5, 44 });
```

```
value constructor
copy constructor
destructor
value constructor
copy constructor
copy constructor
destructor
destructor
value constructor
copy constructor
copy constructor
copy constructor
destructor
destructor
destructor
destructor
destructor
destructor
```




ALGEBRA

27

# Nepotrebna kopiranja

Objekt koji će uskoro umrijeti

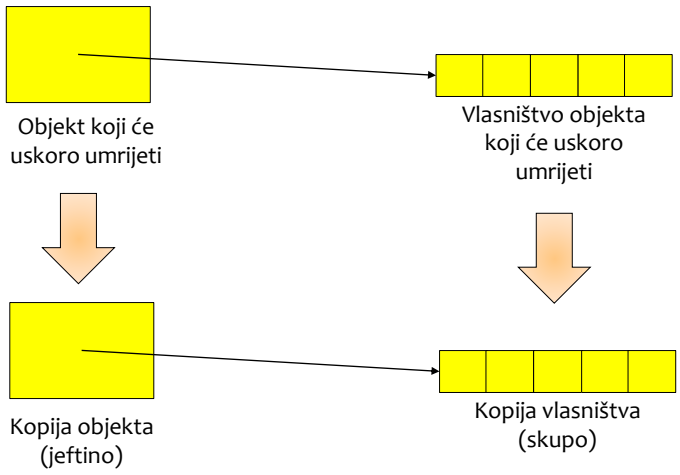
Vlasništvo objekta koji će uskoro umrijeti



ALGEBRA

28

# Nepotrebna kopiranja

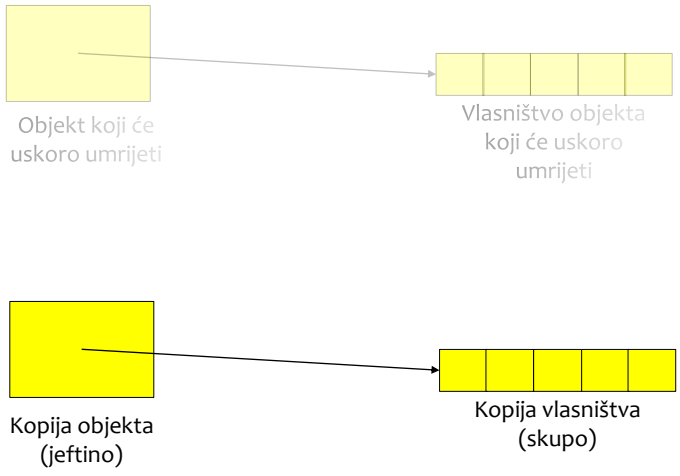





ALGEBRA

29

# Nepotrebna kopiranja






ALGEBRA

30

## Nepotrebna kopiranja

- Kod kopiranja moramo razlikovati:
  - Kopiranje objekta se uvijek mora napraviti
  - Kopiranje vlasništva se nekad može izbjeći
- Kad pričamo o nepotrebnim kopiranjima, pričamo o kopiranju vlasništva
  - Nekad ga želimo kopirati – ako će original i dalje nastaviti živjeti
  - Nekad ga ne želimo kopirati – ako će original umrijeti, ne treba mu vlasništvo, zar ne? Zašto mu ga jednostavno ne bismo uzeli?



ALGEBRA

31

## MOVE-CONSTRUCTOR



ALGEBRA

32

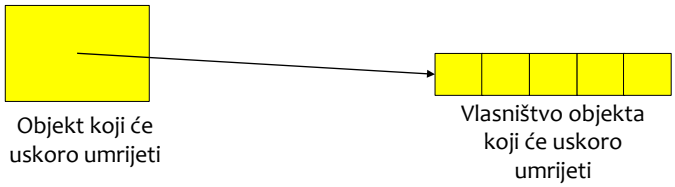
## Move-constructor

- Ideja move-constructora je upravo takva:
  - Objekt se normalno kopira kao i do sada
  - Vlasništvo objekta se jednostavno preuzme od originalnog objekta
  - Naravno, to znači da će originalni objekt ostati bez vlasništva
  - Stoga se move-constructor koristi samo kad će original umrijeti
  - Ako original treba nastaviti živjeti, ne smije se koristiti move-constructor već copy-constructor

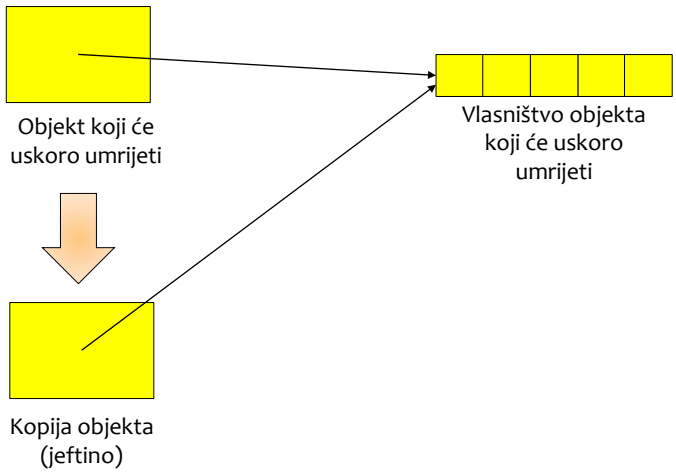




## Move-constructor



## Move-constructor

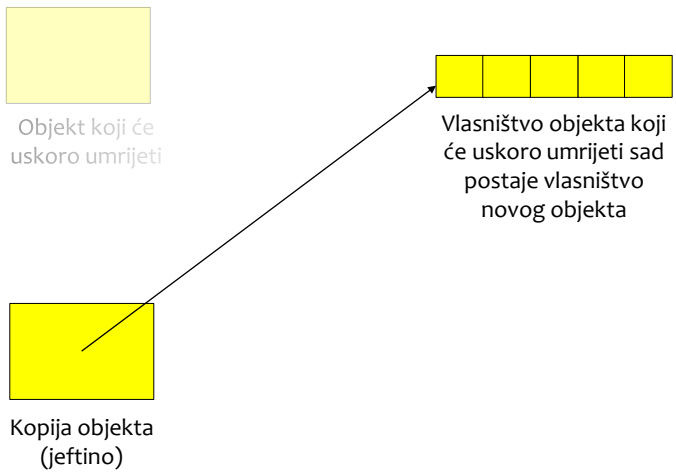





ALGEBRA

35

## Move-constructor






ALGEBRA

36

## Move-constructor

- Move-constructor kao parametar prima referencu na rvalue, tj. na objekt koji će umrijeti upravo nakon te operacije
- Move-constructor od primljenog objekta smije uzeti sve što mu treba jer će primljeni objekt i tako umrijeti






## Move-constructor

```
…
Brojevi(Brojevi&& orig);
…
```

```
…
Brojevi::Brojevi(Brojevi&& orig) {
    kolicina = orig.kolicina;
    polje_brojeva = orig.polje_brojeva; // Drska krađa vlasništva
    orig.polje_brojeva = nullptr;
    cout << "move constructor" << endl;
}
…
```



## Move-constructor

- Pokretanjem programa vidimo da se dio stvari popravio:

```
value constructor
copy constructor
destructor
value constructor
copy constructor
copy constructor
destructor
destructor
value constructor
copy constructor
copy constructor
copy constructor
destructor
destructor
destructor
destructor
destructor
destructor
```

```
value constructor
move constructor
destructor
value constructor
move constructor
copy constructor
destructor
destructor
value constructor
move constructor
copy constructor
copy constructor
destructor
destructor
destructor
destructor
destructor
destructor
```






## Move-constructor

- No, vektor i dalje odbija koristiti naš move-constructor

```
value constructor
copy constructor
destructor
value constructor
copy constructor
copy constructor
destructor
destructor
value constructor
copy constructor
copy constructor
copy constructor
destructor
destructor
destructor
destructor
destructor
destructor
```

```
value constructor
move constructor
destructor
value constructor
move constructor
copy constructor
destructor
destructor
value constructor
move constructor
copy constructor
copy constructor
destructor
destructor
destructor
destructor
destructor
destructor
```



## Move-constructor

- void push_back (const value_type& val);
  - The content of val is copied (or moved) to the new element.
  - If a reallocation happens, the strong guarantee is also given if the type of the elements is either copyable or no-throw moveable.
- Da bi vektor odlučio koristiti move-constructor, moramo naznačiti da move-constructor ne baca iznimku






## Move-constructor

```
Brojevi(Brojevi&& orig) noexcept;
```

```
Brojevi::Brojevi(Brojevi&& orig) noexcept {
    kolicina = orig.kolicina;
    polje_brojeva = orig.polje_brojeva; // Drska krađa vlasništva
    orig.polje_brojeva = nullptr;
    cout << "move constructor" << endl;
}
```



## Move-constructor

- Sad smo se riješili nepotrebnih kopiranja i imamo optimalno rješenje

```
value constructor
move constructor
destructor
value constructor
move constructor
copy constructor
destructor
destructor
value constructor
move constructor
copy constructor
copy constructor
destructor
destructor
destructor
destructor
destructor
destructor
```

```
value constructor
move constructor
destructor
value constructor
move constructor
move constructor
destructor
destructor
value constructor
move constructor
move constructor
move constructor
destructor
destructor
destructor
destructor
destructor
destructor
```





## ISPIS

## Ispis

- Ispis objekta trenutno imamo ovako:

```
for (int i = 0; i < v.size(); i++) {
    v[i].ispisi();
}
```

- Ispis objekta često želimo napraviti ovako:

```
for (int i = 0; i < v.size(); i++) {
    cout << v[i] << endl;
}
```

- Da bismo to napravili, moramo:
  - U klasi preopteretiti `operator<<` pisanjem odgovarajuće friend funkcije
  - Implementirati `operator<<` i napraviti ispis



ALGEBRA

45

## Ispis

```
#pragma once
#include <iostream>
using namespace std;

class Brojevi {
private:
    int kolicina;
    int* polje_brojeva;

public:
    Brojevi();
    Brojevi(int n);
    Brojevi(int n, int val);
    Brojevi(int original[], int n);
    Brojevi(const Brojevi& orig);
    Brojevi(Brojevi&& orig) noexcept;
    ~Brojevi();
    friend ostream& operator<<(ostream& os, const Brojevi& obj);
};
```



ALGEBRA

46

# Ispis

```
ostream& operator<<(ostream& os, const Brojevi& obj) {
    os << "(size:" << obj.kolicina << ") ";
    for (int i = 0; i < obj.kolicina; i++) {
        os << obj.polje_brojeva[i] << " ";
    }
    return os;
}
```



ALGEBRA