Prednost destruktora

- Prevoditelj se brine da se pozove funkcija za uništavanje objekta (destruktor)
- Vrijedi samo za objekte kreirane na stogu
- Kod objekata kreiranih na *heap*-u (pomoću *new*) **mora se pozvati *delete*** !
  - Kreator objekta određuje kada će se on uništiti
- Primjer:
  P06_DinamickoPolje_Cpp3_konstruktori

Konstruktor kopiranja (copy-constructor) i poziv po vrijednosti (call by value)

- Što je sa **kopiranjem** objekata ?
- Npr. poziv funkcije u koju se po po vrijednosti (*call by value)* prenosi objekt !
  - Za obične tipove (int, float, char) je to jednostavno – u funkciji se kreira lokalna varijabla istog tipa i (automatski) inicijalizira s prenesenom vrijednošću
- A što ukoliko se prenosi objekt ?

```
void NekaFunkcija(MojaKlasa obj) {
                    ...
}
void main() {
    MojaKlasa a;

    NekaFunkcija(a);
}
```

i semantika *by-value*

Osnovno pitanje kod kreiranja lokalnog objekta:

- P: Pomoću kojeg konstruktora se kreira lokalni objekt *obj* prilikom poziva funkcije *NekaFunkcija*() ?
- O: Poziva se tzv. konstruktor kopiranja (*copy-constructor*)

- P: Gdje je konstruktor kopiranja u našoj klasi?
- O: Nema ga eksplicitno, ali je prevoditelj sam definirao “standardnu” (*default*) verziju ovog konstruktora

Primjer konstruktora kopiranja

- Sintaksa:

```
class MojaKlasa {
   public:
  MojaKlasa() { ... }        // konstruktor 1
  MojaKlasa(int a) { ... }   // konstruktor 2
  MojaKlasa(const MojaKlasa &initObj) { ... }   //
    // konstruktor kopiranja
  ~MojaKlasa() {}     // destruktor
};
```

- Konstruktor kopiranja se prepoznaje po svom parametru – prima **referencu** na objekt istog tipa koji će poslužiti kao osnova za kreiranje novog objekta
  - Referenca ? = skriveni pokazivač

- Podrazumijevana implementacija copy-constructora radi **samo** kopiranje vrijednosti članskih varijabli iz predanog (postojećeg) objekta u instancu novostvorenog objekta
- Ovo je ispravan pristup za jednostavne objekte
  - Jednostavni objekti: Objekti koji u sebi nemaju pokazivače na alocirane resurse (memorija, datoteke, konekcije na bazu, ...)

- Neispravan za našu klasu DinamickoPolje
  - Objekt kreiran pomoću podrazumijevanog copy-konstruktora će pokazivati na **isti** dio memorije koji je alociran za polje u originalnom objektu
- Je li to to ono što želimo kad DinamickoPolje predajemo u funkciju ?
  - **NE !** – od početka se radi o *call by value*! Ukoliko se želi da funkcija radi sa istom instancom objekta, u funkciju se prenosi se **pokazivač** na objekt

Dva načina kopiranja objekata – dubinsko (deep) i površinsko (shallow)

- *Deep copy* (dubinsko kopiranje) – konstruktor kopiranja se implementira tako da kreira **u potpunosti novu** kopiju objekta
  - Za klasu DinamickoPolje to znači alociranje nove memorije i inicijalizaciju elemenata u tom polju s vrijednostima iz objekta predanog konstruktoru kopiranja

Dva načina kopiranja objekata – dubinsko (deep) i površinsko (shallow)

- *Shallow copy* (površinsko kopiranje) – kreira se novi objekt, ali on nastavlja "dijeliti" određeni dio stanja s objektom na osnovu kojega je nastao
  - Za DinamickoPolje to znači da nakon kreiranja objekta pomoću copy-constructora imamo **dva** objekta koji pokazuju na istu alociranu memoriju (polje) !
- Primjer: P07_DinamickoPolje_Cpp4_copy_constructor TEST

Problem kod objekata s pokazivačima na istu alociranu memoriju

- Doći će do problema kad se pozovu destruktori za ta dva objekta
  - Prvi poziv destruktora će biti OK, ali kad se pozove destruktor za drugi objekt, pokušaj oslobađanja već oslobođene memorije će generirati pogrešku !

Reference

- Referenca = pokazivač s malo drukčijom sintaksom
- Nema adresnog operatora i operatora indirekcije (dereferenciranja) !
  - Prevoditelj to obavlja automatski
- Deklaracija reference se prepoznaje po prefiksu &

Primjer reference

```
void main() {
    int a = 5;

    int *pa = &a;   // pokazivač pa pokazuje na varijablu a
    int &ra = a;    // referenca ra također referencira
        varijablu a
    int& ra2; // POGREŠKA – nedostaje inicijalizacija
    extern int& ra3; //OK – inicijalizacije se obavlja na nekom
        drugom mjestu

    *pa = 10;       // varijabla a sada je jednaka 10

    ra = 15;        // a je sada jednako 15
}
```

Primjeri za vježbu:

- Klasa *Trokut*:
  - Izgraditi klasu Trokut koja će predstavljati koncept trokuta kao geometrijskog lika koji ima definiranu duljinu svake od tri stranice i ugrađenu funkcionalnost za izračunavanje površine i opsega trokuta
  - Koje bi se još članske funkcije mogle definirati za ovu klasu?

Primjeri za vježbu (2)

- Klasa *KompleksniBroj*
  - Izgraditi klasu KompleksniBroj koja će predstavljati (matematički) koncept kompleksnog broja. Klasa mora sadržavati dvije članske varijable koje će predstavljati realni i imaginarni dio i pružati funkcionalnost za obavljanje matematičkih operacija s kompleksnim brojevima (zbrajanje, oduzimanje, množenje, dijeljenje, konjugiranje, apsolutna vrijednost, trigonometrijski oblik)