konstruktory a destruktory (o)

- možnost ovlivnění vzniku (inicializace) a zániku (úklid) objektu
- základní myšlenkou je, že proměnná by měla být inicializována (nastavena do počátečního stavu) a zároveň by se při zániku proměnné neměly ztratit získaná data nebo obdržené zdroje (soubory, paměť ...)

- volány automaticky překladačem (nespoléhá se na uživatele)
- konstruktor – první (automaticky) volaná metoda na objekt. Víme, že data jsou (určitě) neinicializovaná.
- destruktor – poslední (automaticky) volaná metoda na objekt

- „konstruktory“ vlastně známe již z jazyka C z definicí s inicializací
- některé „konstruktory“ a „destruktor“ jsou u jazyka C prázdné (nic nedělají)

```
{//příklad(přiblížení) pro standardní typ int
 int a;
//definice proměnné bez konkrétní inicializace = implicitní

 int b = 5.4;
// definice s inicializací. vytvoření, konstrukce proměnné int
// z hodnoty double = konverze (z double na int)

 int c = b;
// konstrukce (vytvoření) proměnné int na základě proměnné
// stejného typu = kopie
…
}  // konec životnosti proměnných – zánik proměnných
// je to prosté zrušení bez ošetření – (u std typů)
// zpětná  kompatibilita
```

Konstruktor

- metoda, která má některé speciální vlastnosti
- stejný název jako třída
- nemá návratovou hodnotu
- volán automaticky při vzniku objektu (lokálně i dynamicky)
- využíván k inicializaci proměnných (nulování, nastavení základního stavu, alokace paměti, …)
- možnost (funkčního zápisu) konstrukce je přidána i základním typům

```
class Trida {
int ii;
public:
Trida(void) {…} // implicitní kontruktor
Trida(int i): ii(i) {…} // konverzni z int
Trida(Trida const & a) {…} // kopy konstruktor
 }
```

- třída může mít několik konstruktorů – přetěžování
- implicitní (bez parametrů) – volá se i při vytváření prvků polí
- konverzní – s jedním parametrem
- kopy konstruktor – vytvoření kopie objektu stejné třídy (předávání hodnotou, návratová hodnota …), rozlišovat mezi kopy konstruktorem a operátorem přiřazení

```
Trida(void)   // implicitní
Trida(int i)  // konverzní z int
Trida(char *c) // konverzní z char *
Trida(const Trida &t) // copy
Trida(float i, float j) // ze dvou parametrů
Trida(double i, Trida &t1) //ze dvou parametrů

Trida a, b(5), c(b), d=b, e(”101001”);
Trida f(3.12, 8), g(8.34, b), h = 5;
// pouze pro ”názornost” !!! Překladač přeloží
Trida   a.Trida(),   b.Trida(5),   c.Trida(b),   d.Trida(b)
e.Trida(”101001”); h.Trida((tmp.)Trida(5))), (nebo výjimečně
h.Trida(5) (nebo špatně) h.operator=((tmp.)Trida(5))),   (
(tmp.)~Trida(); )
//”.Trida” by bylo nadbytečné a tak se neuvádí
```

- konstruktor může použít i překladač ke konverzi (například voláním metody s parametrem int, když máme metodu jen s parametrem Trida, ale máme konverzní konstruktor z int)
- konstruktory se používají pro implicitní konverze, pouze jedna uživatelská (problémy s typy, pro které není konverze)
- explicit - klíčové slovo – zakazuje použití konstruktoru k implicitní konverzi

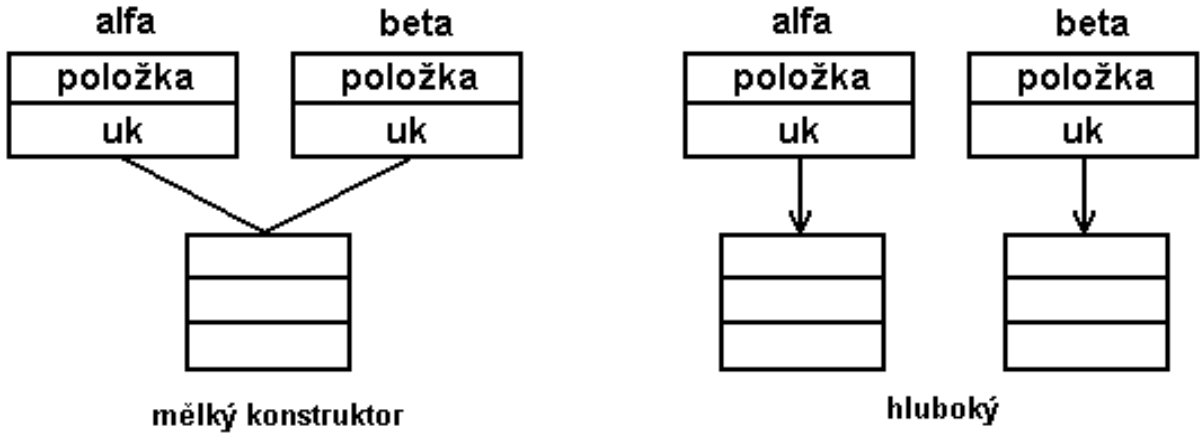
```
class Trida {public:
explicit Trida(int j); // konverzní konstruktor z int
Trida::Metoda(Trida & a);
}


int i;

a.Metoda ( i ); // nelze, implicitní konverze se neprovede
b.Metoda ( Trida(i)); // lze,  explicitní konverze je povolena
```

- u polí se volají konstruktory od nejnižšího indexu
- konstruktor nesmí být static ani virtual
- alespoň jeden musí být v sekci public (jinak zákaz pro běžného uživatele)

- implicitní konstruktor (kvůli zpětné kompatibilitě) vzniká automaticky jako prázdný
- je-li nadefinován jiný konstruktor, implicitní automaticky nevznikne
- není-li programátorem definován kopykonstruktor, je vytvořen překladačem (kvůli zpětné kompatibilitě) a provádí kopii (paměti) jedna k jedné (memcopy)

Vysvětlete jaký je problém při vzniku automatického kopykonstruktoru je-li ve třídě ukazatel?

- automatický kopykonstruktor se chová jako prostá kopie prostoru jedné proměnné do druhé
- pokud objekt nevlastní dynamická data (ukazatel na ně), dojde ke kopii hodnot což je v pořádku
- jsou-li dynamická data ve třídě (tj. jsou odkazována ukazatlelm) dojde ke kopii ukazatele. Potom dva objekty ukazují na stejná data (a neví od tom) –> problém při rušení dat – první rušený objekt společná data zruší …
- nazýváme je mělké (memcopy) a hluboké kopírování (shallow, deep copy – vytváří i kopii dat na které se odkazují ukazatele. Ukazatele mají tedy různé hodnoty.)
- řešením je vlastní kopie nebo indexované odkazy



Destruktor

- stejný název jako třída, předchází mu ~ (proč?)
- je pouze jeden (bez parametrů)
- nemá návratovou hodnotu
- volán automaticky překladačem
- zajištění úklidu (vrácení systémových prvků, paměť, soubory, ovladače, ukončení činnosti HW, uložení dat …)

```
~Trida (void)
```

- destruktory se volají v opačném pořadí jako konstruktory
- je možné ho volat jako metodu (raději ne)
- není-li definován, vytváří se implicitně prázdný
- musí být v sekci public