Inicijalizacija vrednošću

- Pritom:
  - u svim navedenim slučajevima kada se koriste velike zagrade *{}*, ako je *T* agregatni tip (niz ili klasa pod nekim uslovima, tipično struktura), vrši se agregatna inicijalizacija umesto inicijalizacije vrednošću
  - ako je *T* klasa koja nema podrazumevani konstruktor, ali ima konstruktor koji prihvata argument tipa *std::initializer_list*, vrši se inicijalizacija listom umesto inicijalizacije vrednošću
- Inicijalizacija vrednošću vrši sledeće:
  - Ako je *T* klasa bez podrazumevanog konstruktora, ili sa eksplicitno korisnički definisanim ili obrisanim podrazumevanim konstruktorom, objekat se inicijalizuje podrazumevanom inicijalizacijom (koja nije dozvoljena ako podrazumevani konstruktor ne postoji ili je obrisan)
  - Ako je *T* klasa sa podrazumevanim konstuktorom koji nije ni eksplicitno korisnički definisan niti obrisan, objekat se najpre inicijalizuje nulom, a potom podrazumevanom inicijalizacijom ako ima netrivijalan podrazumevani konstruktor
  - Ako je *T* niz, svaki element se inicijalizuje vrednošću
  - Inače, objekat se inicijalizuje nulom
- Uprošćeno, ako je korisnik sam napravio podrazumevani konstruktor, onda ova inicijalizacija poziva taj konstruktor i radi podrazumevanu inicijalizaciju definisanu u njemu; u suprotnom, incijalizuje objekat nulom, a onda poziva podrazumevane konstruktore podobjekata. Na primer:

```
struct X {
  int i;
  X () {}
};

struct Y {
  int j;
  X x;
};

cout<<X().i<<endl<<Y().j<<endl<<Y().x.i;
```

Eksplicitno definisan korisnički konstruktor, ne inicijalizuje *i*

Implicitno definisan podrazumevani konstruktor

U privremenom objektu X(), *i* će imati neodređenu vrednost.
U privremenom objektu Y(), *j* i *x.i* će imati vrednost 0.

Inicijalizacija kopiranjem

- *Inizijalizacija kopiranjem* (*copy initialization*) se obavlja kada se objekat (ne referenca) tipa *T* (koji je objektni tip) inicijalizuje u sledećim slučajevima:
  - Kada se imenovani objekat (automatski, statički ili vezan za nit) inicijalizuje izrazom iza znaka =:

```
T t = expression;
```

  - Kada se argument prenosi u pozvanu funciju po vrednosti (ne referenci):

```
void f (T t);
f(expression);
```

  - Kada se vraća iz funkcije koja vraća vrednost (ne referencu):

```
T f () {
  …
  return expression;
}
```

  - Kada se baca ili hvata izuzetak po vrednosti (ne referenci):

```
throw expression;
catch (T t) {…}
```

  - Kao deo agregatne inicijalizacije, za inicijalizaciju svakog elementa za koji je zadat inicijalizator:

```
T a[N] = {expression, expression, …};
```