# Privremeni objekti

- Počev od verzije jezika C++17, ova semantika je značajno izmenjena: rezultat poziva funkcije (uključujući i operatorske funkcije), baš kao i rezultat bilo koje operacije i izraza jeste *vrednost* (*value*), a ne privremeni objekat
- Vrednost je poseban entitet u programu i odnosi se na rezultat izraza koji se dalje može koristiti kao argument poziva funkcije. Na taj način se između poziva funkcija u izrazima prosleđuju vrednosti
- Pravljenje privremenog objekta se sada maksimalno odlaže do trenutka kada postane zaista neophodno, i vrši se samo u nekim situacijama; ovo se naziva *materijalizacija privremenog objekta* (*temporary materialization*); na primer, navedeno izostavljanje kopiranja kod inicijalizacije, kao i na mnogim drugim mestima, gde god je moguće kopiranje izostaviti, nije više dozvoljena, ali neobavezna optimizacija prevodioca, već je definisana semantika: na navedenom mestu se privremeni objekat *nikada* ne pravi
- Referenca (na lvrednost) se može inicijalizovati izrazom koji nije lvrednost samo ako je ta referenca na konstantu; tada se (i pre, i od verzije C++17), pravi privremeni objekat za koji se vezuje ta referenca:

```
complex& r1 = complex(1.,0.);
const complex& r2 = complex(1.,0.);
```

Greška u prevođenju: referenca na nekonstantnu lvrednost ne može se incijalizovati izrazom koji nije lvrednost

- Zbog ovoga, ako neka funkcija ima parametar koji je referenca (na lvrednost), ona treba da bude referenca na konstantu, inače se ta funkcija ne bi mogla pozvati sa argumentima koji to nisu:

```
complex operator+ (const complex&, const complex&);
...c1+complex(1.,0.)
```

Ovaj poziv ne bi bio ispravan kada bi parametar bio referenca na nekonstantu

# Privremeni objekti

- Uništavanje svih privremenih objekata napravljenih pri izračunavanju nekog potpunog izraza (izraza čiji rezultat više nije operand nekog okružujućeg izraza) vrši se kao poslednji korak u izračunavanju tog izraza u koji je pravljenje tih privremenih objekata leksički ugrađeno; drugim rečima, svi privremeni objekti napravljeni tokom izračunavanja izraza ne nadživljavaju pun izraz u kom su leksički napravljeni (ne uključuje dinamički ugnežđene izraze izvršene unutar pozvanih funkcija):

```
c3 = (c1 + c2) * (complex(1.,0.) + c2);
```

- Ako je tokom izračunavanja izraza napravljeno više privremenih objekata, oni se uništavaju po obrnutom redosledu od redosleda njihove inicijalizacije
- Sve ovo važi čak i ako je izračunavanje izraza bacilo izuzetak
- Ako je za privremeni objekat vezana referenca, životni vek tog privremenog objekta produžava se do kraja životnog veka te reference, ali ne tako da on nadživi poziv funkcije u kojoj je napravljen: svi privremeni objekti obavezno se uništavaju pre povratka iz funkcije u kojoj su napravljeni, pa referenca vraćena iz funkcije i dalje može biti viseća